ABSTRACT

Title of defense: Deep Video Analytics of Humans:
From Action Recognition to Forgery Detection

Steven Schwarcz, Doctor of Philosophy, 2021

Dissertation directed by: Professor Rama Chellappa
Department of Electrical Engineering

In this work, we explore a variety of techniques and applications for visual problems involving videos of humans in the contexts of activity detection, pose detection, and forgery detection.

The first works discussed here address the issue of human activity detection in untrimmed video where the actions performed are spatially and temporally sparse. The video may therefore contain long sequences of frames where no actions occur, and the actions that do occur will often only comprise a very small percentage of the pixels on the screen. We address this with a two-stage architecture that first creates many coarse proposals with high recall, and then classifies and refines them to create temporally accurate activity proposals. We present two methods that follow this high-level paradigm: TRI-3D and CHUNK-3D.

This work on activity detection is then extended to include results on few-shot learning. In this domain, a system must learn to perform detection given only an extremely limited set of training examples. We propose a method we call a Self-Denoising Neural Network (SDNN), which takes inspiration from Denoising Autoencoders, in order to solve this problem, both in the context of activity detection and image classification.

We also propose a method that performs optical character recognition on real world images when no labels are available in the language we wish to transcribe. Specifically, we build an accurate transcription system for Hebrew street name signs when no labeled training data is available. In order to do this, we divide the problem into two components and address each separately: content, which refers to the characters and language structure, and style, which refers to the domain of the images (for example, real or synthetic). We train with simple synthetic Hebrew street signs to address the content components, and with labeled French street signs to address the style.

We continue our analysis by proposing a method for automatic detection of facial forgeries in videos and images. This work approaches the problem of facial forgery detection by breaking the face into multiple regions and training separate classifiers for each part. The end result is a collection of high-quality facial forgery detectors that are both accurate and explainable. We exploit this explainability by providing extensive empirical analysis of our method's results.

Next, we present work that focuses on multi-camera, multi-person 3D human pose estimation from video. To address this problem, we aggregate the outputs of a 2D human pose detector across cameras and actors using a novel factor graph formulation, which we optimize using the loopy belief propagation algorithm. In particular, our factor graph introduces a temporal smoothing term to create smooth transitions between poses across frames.

Finally, our last proposed method covers activity detection, pose detection, and tracking in the game of Ping Pong, where we present a new dataset, dubbed SPIN, with extensive annotations. We introduce several tasks with this dataset, including the task of predicting the future actions of players and tracking ball movements. To evaluate our performance on these tasks, we present a novel recurrent gated CNN architecture.

Deep Video Analytics of Humans: From Action Recognition to Forgery Detection

by

Steven Schwarcz

Research dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Advisory Committee:

Professor Rama Chellappa, Chair/Advisor
Professor David Jacobs
Professor Abhinav Shrivastava
Professor Christopher Metzler
Professor Shuvra Bhattacharyya

Dedicated to my wife, Melissa. I couldn't have done this without you.

ACKNOWLEDGEMENTS

I would like to start by thanking my advisor, Professor Rama Chellappa, for his support, guidance, and kindness throughout this entire PhD program. His wisdom and knowledge provided me with constant direction throughout my research, and I would not be where I am without him. In particular, he afforded me with opportunities to work on large scale government projects, where I learned a lot both about how research programs are structured and how to perform meaningful research. It has been an honor to work under Professor Chellappa.

It is also an honor to have Professors David Jacobs, Shuvra Bhattacharyya, Abhinav Shrivastava, and Christopher Metzler on my PhD committee, and I would like to thank all of them for graciously agreeing to take on that task.

I would also like to thank everyone whose insightful conversations have helped me over the course of my research, including Dr. Carlos Castillo, Joshua Gleason, Saketh Rambhatla, Dr. Yogesh Balaji, Dr. Rajeev Ranjan, and Professor Abhinav Shrivastava, among others. This also extends to my mentors from outside of UMD, including Tom Pollard at Systems and Technology Research, as well as Dr. Xavier Gibert Serra, Dr. Alexander Gorban, Dr. Dar-Shyang Lee, Dr. Peng Xu, and Dr. Navdeep Jaitly at Google. All of you were incredibly helpful and supportive, not just with respect to my research but also with respect to my career as a whole.

I would especially like to thank my mother and father for their constant support, not just throughout this process but throughout my entire life, as well as my sister Sara. Above all, I would like to thank my wife Melissa, who gave me unwavering support through the highs and the lows, and without whom none of this would have been possible.

Parts of this dissertation were based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R&D Contract No. D17PC00345. The views and conclusions contained herein are those

of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

This dissertation was also supported in part by the Office of Naval Research (N00014-16-P-2040).

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Motivation

In recent years, the development of Deep Neural Networks (DNNs), particularly Deep Convolutional Neural Networks (CNNs), has led to a significant increase in the performance of many Computer Vision systems. This new paradigm has had far-reaching implications across the entirety of the field of Computer Vision, including advancements in many classical areas of Computer Vision and AI, such as image segmentation, facial recognition, object recognition, and countless others.

In this work, we focus on several classes of Computer Vision problems pertaining to humans, including action recognition, forgery detection, and pose detection. These categories cover a large range of applications for real world systems, from surveillance, to very topical issues of data integrity in an age when new methods of faking images or videos of individuals are becoming increasingly prevalent.

For instance, with respect to surveillance, a facility may be equipped with many cameras taking constant footage of a large physical area over a large period of time. In order to be reactive to important events that are filmed, it is imperative to have an automatic system that can identify and report significant events as they happen. To build such a system, one must be able to sift through possibly hundreds or thousands of hours of high-resolution footage in order to find the most significant events to report.

In order to understand what is happening in these videos, one must build vision systems with the ability to identify human poses and actions. These difficult problems can only be tackled by building models of humans and intelligently applying them in dense data when actions are sparse.

Related to the issue of understanding humans in videos is the challenge of identifying when a video of a human is itself a forgery. Recent DNN-based techniques have made it increasingly

easy to fabricate videos of individuals, and it can be very difficult for humans to identify these forgeries, even by professionals. As a result, it is necessary to develop robust automated forgery detection algorithms. Such automated detection is only possible, however, using algorithms that can properly model the appearance of non-forged humans.

1.2 Proposed Approaches

We begin this work in Chapter 2 by discussing several techniques for detecting activities of humans in untrimmed videos, especially videos where the actions of the humans are temporally and spatially sparse. This sparsity is the key challenge of this task: the video may contain long sequences of frames where no actions occur, and the actions that do occur will often only comprise a very small percentage of the pixels on the screen. To address this, we aggregate 2D object detections using hierarchical clustering in order to create activity proposals. The proposals are coarse but have extremely high recall. We then classify these proposals as actions and refine their temporal bounds using a CNN. Using this two stage architecture, which we call Temporal Refinement I3D (TRI3D) after the I3D action recognition architecture [26], we achieve state of the art performance on the DIVA dataset for action recognition.

These results are then improved in two key ways. First, in Chapter 3, we simplify the proposal system by dividing our proposals into overlapping chunks and then aggregating, achieving strong state-of-the-art results on the THUMOS'14 dataset [109]. We then expand the purview of these techniques in Chapter 4 by developing a new technique for handling the low-data regime where there may be only one or five labeled examples of activities on which to train. This technique, which we call Self-Denoising Neural Networks (SDNN), is general enough to yield improvements not just in the few-shot activity detection task but also in few-shot image classification on datasets such as *mini*ImageNet [247], CIFAR-FS [15], and *tiered*ImageNet [183].

In Chapter 5, we present a system that performs optical character recognition on real world images when no labels are available in the language we wish to transcribe. Specifically, we build an accurate transcription system for Hebrew street name signs that do not have labeled training samples. In order to do this, we divide the problem into two components and address each separately: content, which refers to the characters and language structure, and style, which refers to the domain of the images (for example, real or synthetic). We train with simple synthetic Hebrew street signs to address the content components and labeled French street signs to address the style.

Having examined these topics, we explore other human-centric topics in Computer Vision. In Chapter 6 we present a series of methods for exploring the empirical effectiveness of facial forgery detectors when they are limited to the detection of single region of the face. Our results provide an explainable approach for understanding what artifacts are left behind by different facial manipulations methods, and where they appear on the face.

We conclude with work focusing on more fine-grained understanding of Human Pose. Chapter 7 focuses on 3D human pose estimation from video over multiple cameras and multiple actors. To address this problem, we aggregate the outputs of a 2D human pose detector - in our case we use the existing OpenPose detector [24] - across cameras and actors using a novel factor graph formulation, which we optimize using the loopy belief propagation algorithm. In particular, the proposed factor graph introduces a temporal smoothing term to create smooth transitions between poses across frames. Our architecture achieves state-of-the-art on the existing TUM Campus and TUM Shelf datasets [8] for multi-camera multi-person human pose estimation. Chapter 8 narrows the pose and action recognition tasks of previous chapters by focusing on the game of Ping-Pong, introducing a novel architecture for prediction of player actions and presenting the new SPIN dataset for training and benchmarking these tasks.

1.3 Organization

Chapters 2 and 3 outline the TRI-3D and CHUNK-3D human action detection architectures. Chapter 4 introduces the Self-Denoising Neural Network (SDNN) architecture for few-shot detection in images and videos. Chapter 5 discusses a system for sequential Optical Character Recognition in a domain with unlabeled training data by utilizing synthetic data in the same language and real data in a different one. Chapter 6 presents a novel parts-based approach to detecting facial forgeries. Chapter 7 discusses a factor-graph based method for human pose estimation, and Chapter 8 introduces and benchmarks the new SPIN dataset for action and pose recognition in the game of Ping Pong. Finally, we conclude in Chapter 9.

2. A PROPOSAL-BASED SOLUTION TO SPATIO-TEMPORAL ACTION DETECTION IN UNTRIMMED VIDEOS

2.1 Introduction

Action detection in untrimmed videos is a challenging problem. Although methods using deep CNNs have significantly improved performance on action classification, they still struggle to achieve precise spatio-temporal action localization in challenging security videos. There are some major challenges associated with action detection from untrimmed security videos. First, the action typically occurs in a small spatial region relative to the entire video frame. This makes it difficult to detect the actors/objects involved in the action. Second, the duration of the action may vary significantly, ranging from a couple of seconds to a few minutes. This requires the detection

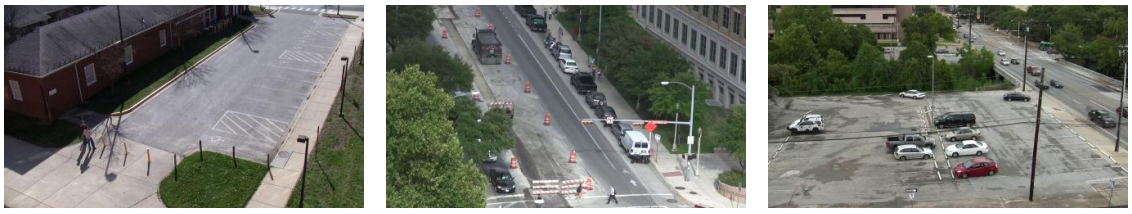


Figure 2.1: Sample images of different scenes of the DIVA dataset which presents challenging action detection scenarios which require algorithms to be robust to large variations in scale, object pose, and camera viewpoint.

procedure to be robust to temporal variation. Existing publicly available action detection datasets such as THUMOS'14 [110] and AVA [85] do not posses these challenges. Hence, algorithms trained on these datasets have sub-optimal performance on untrimmed security videos. In this chapter, we work with the DIVA dataset that has untrimmed security videos. Videos that comprise the DIVA dataset are a subset of those in the VIRAT dataset [161], albeit with newly introduced annotations that make them suitable for the activity detection task. Figure 2.1 shows some sample frames from the DIVA dataset.

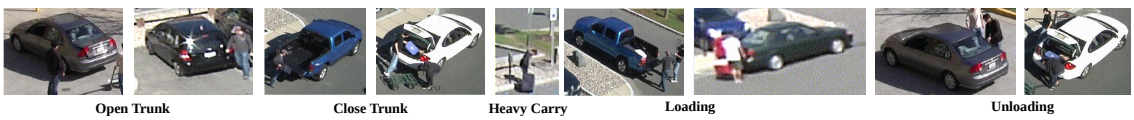


Figure 2.2: Sample frames of some activities of the DIVA dataset.

In this work we introduce a proposal-based modular system for performing spatio-temporal action detection in untrimmed videos. Our system generates spatio-temporal action proposals based on detections from an off-the-shelf detector [73], then classifies the proposals using an existing network architecture with minor changes.

Our proposed system has the advantage that it is both simple and does not require tracking of moving objects. Tracking of objects, often seen as an important component in action recognition systems, presents considerable challenges. Tracking errors generate problems in action recognition from which it is very difficult to recover. On the other hand, object detection has advanced consistently over the past few years, with more sophisticated frame-wise object detectors becoming available. These detectors can be successfully applied to previously unseen videos.

The proposed approach for action detection is based on the observation that we can generate high-recall proposals by clustering object detections. The dense proposals are then applied to a deep 3D-CNN to classify them as either one of the action classes or the non-action class. The temporal bounds for the proposals are also refined to improve localization. We modify the existing I3D [26] network for action classification by adding an additional loss term for temporal refinement. We call the modified network Temporal Refinement I3D (TRI-3D). In summary, this chapter makes the following contributions:

- We introduce a proposal-based modular system for spatio-temporal action detection in untrimmed security videos.
- We propose an algorithm using hierarchical clustering and temporal jittering for generating action proposals using frame-wise object detections.
- We propose the Temporal Refinement I3D (TRI-3D) network for action classification and temporal localization.
- We evaluate our system on the DIVA dataset, which is an untrimmed security video dataset in the wild.

2.2 Related Work

Much research has gone into designing algorithms for action classification from videos. In recent years, many methods have achieved remarkable performance using CNNs for the task of action classification [115, 151, 210, 273]. While [115, 273] use frame-based features, [210] uses a two-stream (RGB and optical-flow) CNN approach to utilize the temporal information of videos.

More recently, researchers have used 3D-CNNs for action classification [26, 98, 192, 234] that simultaneously take in multiple video frames and classify them into actions.

The task of spatio-temporal action detection from untrimmed videos is a challenging problem. Less work has gone into localizing actions, not just along the temporal axis but also in terms of spatial localization. Existing action detection algorithms can be broadly classified into 1) end-to-end systems, and 2) proposal-based systems. The end-to-end action detection approaches feed a chunk of video frames into a CNN which simultaneously classifies and localizes the action. Hou et al. [98] proposed a tube convolutional neural network (T-CNN) that generates tube proposals from video-clips and performs action classification and localization using an end-to-end 3D-CNN. Kalogeiton et al. [113] extracts convolutional features from each frame of a video, and stacks them to learn spatial locations and action scores. Although these end-to-end learning methods may have a simpler pipeline, they are less effective for security videos, where the action is likely to happen in a small spatial region of a frame. On the other hand, proposal-based methods perform action detection in two steps. The first step computes the action proposals, while the second step classifies and localizes the action. Some proposal-based methods are presented in [74, 142, 144, 162, 283]. Unlike existing proposal-based approaches, our method uses hierarchical clustering and temporal jittering to group frame-wise object detections obtained from off-the-shelf detectors in the spatio-temporal domain. It gives us the advantage of detecting variable length action sequences spanning a small spacial region of the video.

In this section we have covered a number of action recognition works, however this list is far from complete. For further action recognition works we point the reader to a more extensive curated list presented in [35].

2.3 DIVA dataset

The DIVA dataset is a new spatio-temporal action detection dataset for untrimmed videos. The current release of the DIVA dataset (DIVA V1) is adapted from the VIRAT dataset [161] with new annotations for 12 simple and complex actions of interest focusing on the public security domain. All actions involve either people or vehicles. Actions include `vehicle U-turn`, `vehicle left-turn`, `vehicle right turn`, `closing trunk`, `opening trunk`, `loading`, `unloading`, `transport heavy carry`, `open` (door), `close` (door), `enter`, and `exit`. The dataset currently consists of 455 video clips with 12 hours and 40 minutes in total captured at different sites. There are 64 videos in the training set, 54 videos in the validation set,

96 videos with annotations withheld in the test set. The remaining videos are for future versions of the dataset. All video resolutions are either 1200×720 or 1920×1080 and humans range in height from 20 to 180 pixels. We show sample images of different scenes and sample frames of some activities in Figure 2.1 and Figure 2.2 respectively. In addition, the number of training instances per action is shown in Figure 2.3.

2.3.1 Challenges

As compared to other action detection datasets, such as the THUMOS'14 [110] and AVA [85] datasets, the DIVA dataset introduces several new challenges for the action detection task that make methods designed for existing action datasets unsuitable. The first issue is the sparsity of actions, both spatially and temporally. For example, exactly half of all videos contain at least 30 seconds of footage where no actions are performed. What makes DIVA particularly challenging is the spatial sparsity of actions: the average size for the bounding boxes of all actions in the training set is 264×142. As a result, when an action is occurring it only takes up on average less than 2.6% of the pixels in any given image, and no action in the entire dataset takes up more than 40%. Additionally, with few exceptions, the similarity of each action and each environment makes it very difficult to use the context of the surrounding scene to assist in classification.

When compared with other setups, where actions are assumed to make up the majority of pixels on any given frame, this motivates the need for a completely different approach focused on the localization of activities. For example, [184] mentions that the smallest anchor size in Faster-RCNN is 128×128 on a 600×600 input image, or 4.5% of the image pixels. This means the average action in DIVA is barely more than half the size of the smallest object detectable by conventional means.

The dataset also contains significant spatial and temporal overlap between activities. This is not just an issue between unrelated activities in the same frame (*e.g.* one person entering a car while another leaves a different car), but is actually more fundamental. For a common example, consider the activities `opening`, `entering`, and `closing`, which apply to a human actor interacting with a car. In order to enter a car, a subject may first open the car door (though it may already be open), and will often close it afterwards (though they may not). All three of these actions are usually performed in quick succession, yet DIVA begins annotation of each activity one second before it begins, and finishes annotating one second after it completes. It is therefore imperative that our system can handle large degrees of spatio-temporal overlap.

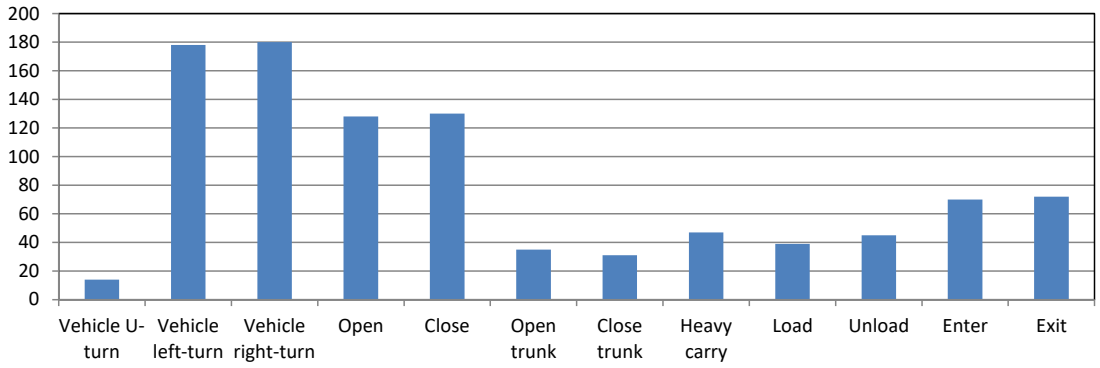


Figure 2.3: The number of training instances per action from the DIVA training set.

Figure 2.4: On the left, the DIVA action `Closing` makes up only a small portion of the image, and the surrounding context has no value for the action classification task. The THUMOS action `Cricket` on the right is much larger in the image, and the entire image's context is useful for classification.

Figure 2.5: Some of object detection and segmentation results for the DIVA dataset using Mask-R-CNN [89].

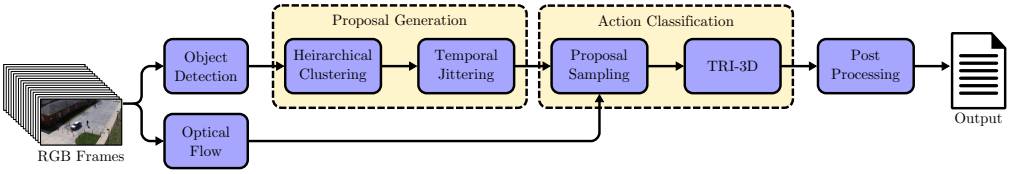


Figure 2.6: Proposed system for spatio-temporal action detection.

2.4 Proposed Approach

Our approach consists of three distinct modules. The first one generates class-independent spatio-temporal proposals from a given untrimmed video sequence. The second module performs action classification and temporal localization on these generated proposals using a deep 3D-CNN. The final module is a post-processing step that performs 3D non-maximum suppression (NMS) for precise action detection. The system diagram for our approach is shown in Figure 2.6. In the following sub-sections we discuss in detail the steps of our proposed approach.

2.4.1 Action Proposal Generation

The primary goal of the action proposal generation stage is to produce spatio-temporal cuboids from a video with high recall and little regard for precision. Although sliding-window search in spatio-temporal space is a viable method for proposal generation, it is computationally very expensive. An alternate solution is to use unsupervised methods to cluster the spatio-temporal regions from a video in a meaningful way. In our approach, we generate the action proposals by grouping frame-wise object detections obtained from Mask-RCNN [89] in the spatio-temporal domain using hierarchical clustering. These generated proposals are further jittered temporally to increase the overall recall.

Object Detection

For object detection, we apply Mask R-CNN [89], an extension of the well-known Faster R-CNN [184] framework. In addition to the original classification and bounding box regression

network of Faster R-CNN, Mask-RCNN adds another branch to predict segmentation masks for each Region of Interest (RoI). In Figure 2.5, we show some sample results from video frames of the DIVA dataset. We observe that Mask-RCNN is able to detect humans and vehicles at different scales, a feature which is useful for detecting the multi-scale actions of the DIVA datasets.

Hierarchical Clustering

The objects detected using Mask-RCNN are represented by a 3-dimensional feature vector $(\mathbf{x}, \mathbf{y}, \mathbf{f})$, where $(\mathbf{x}, \mathbf{y})$ denotes the center of the object bounding box and $\mathbf{f}$ denotes the frame number. We use the SciPy implementation of Divisive Hierarchical Clustering [111, 149] to generate clusters from the 3-dimensional features. We dynamically split the resulting linkage tree at various levels to create k clusters, where k is proportional of video length. The proposals are generated from the bounding box of all detections in the cluster. They are cuboids in space-time and are denoted by $(x_{min}, y_{min}, x_{max}, y_{max}, f_{start}, f_{end})$. This yields an average of approximately 250 action proposals per video on DIVA validation set. Further details regarding the exact implementation can be found in the supplementary material of [77].

Dense Action Proposals with Temporal Jittering

Although the proposals generated using hierarchical clustering reduce the spatio-temporal search space for action detection, they are unable to generate high recall for the following two reasons: 1) The generated proposals are independent of both the action class and cuboid temporal bounds. Hence, they are less likely to overlap precisely with the ground-truth action bounds. 2) Few proposals are generated. A higher recall is achieved with larger numbers of proposals. In order to solve these issues, we propose a temporal jittering approach to generate dense action proposals from the existing proposals obtained from hierarchical clustering.

Let the start and end frames for an existing proposal be denoted by f_{st} and f_{end} respectively. We first choose the anchor frames by sliding along the temporal axis from f_{st} to f_{end} with a stride of s. The anchor frames thus selected are $(f_{st}, f_{st}+s, f_{st}+2s, f_{st}+3s, ..., f_{end})$. For each of the anchor frames f_a, we generate four sets of proposals with temporal bounds (f_a-16, f_a+16), (f_a-32, f_a+32), (f_a-64, f_a+64) and (f_a-128, f_a+128). We choose the proposals frame lengths to be $\{32, 64, 128, 256\}$ based on the average frame lengths for the actions in the DIVA dataset which range between $32-256$ (see Figure 2.7). The pseudo-code for generating dense action proposals is presented in Algorithm 1.

Generating dense proposals using temporal jittering has two advantages. First, the recall is

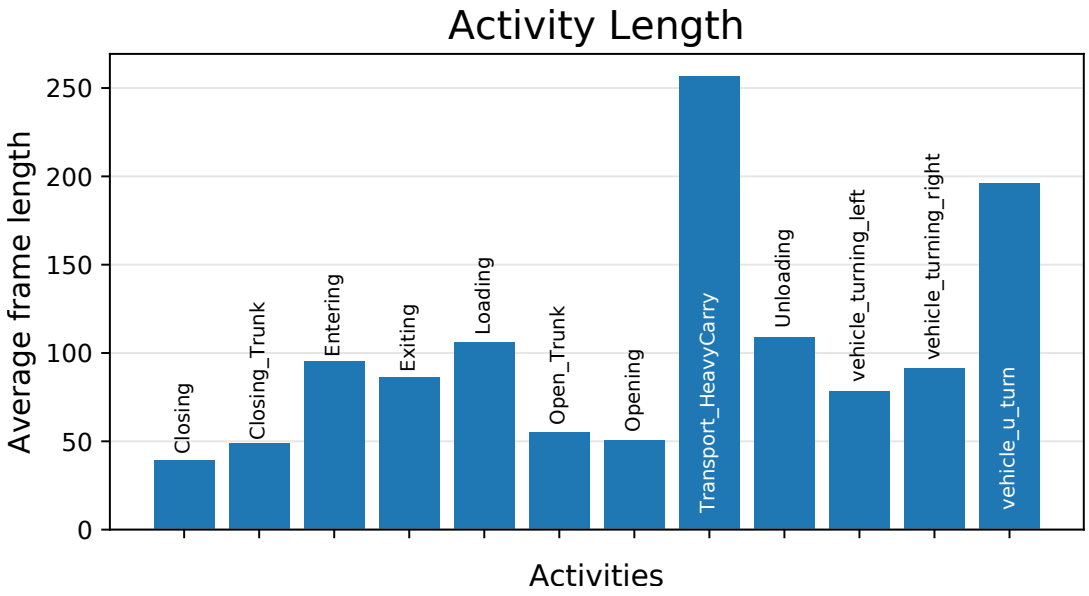


Figure 2.7: The average frame lengths for 12 different activities from the DIVA training set.

higher. Second, having a large number of dense training proposals provides better data augmentation for training, thus improving performance.

Figure 2.8 provides a quantitative comparison of three action proposal generation methods on the DIVA validation set in terms of recall vs 3D Intersection over Union (IoU) with the ground-truth activities: 1) Rule-based 2) Hierarchical clustering, and 3) Temporal Jittering. The rule-based

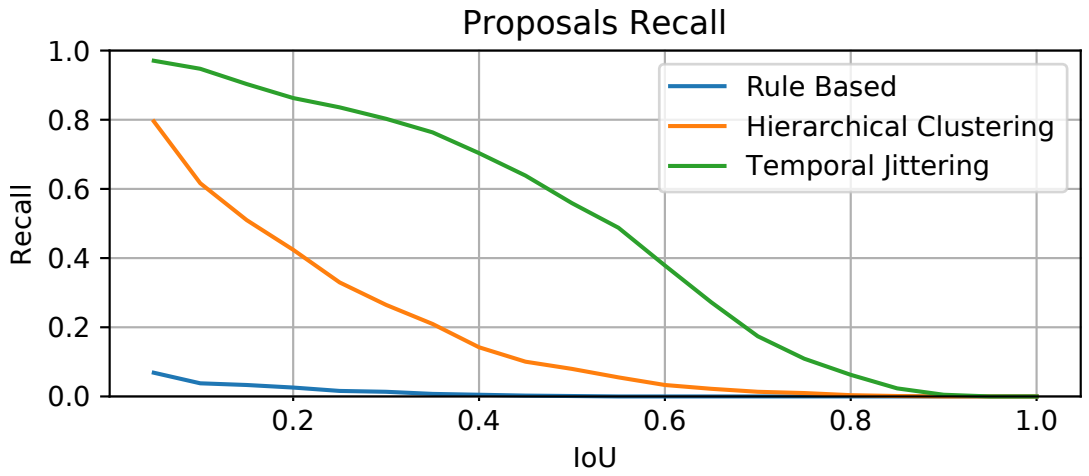


Figure 2.8: The recall for different action-proposal generation methods as a function of spatio-temporal intersection over union (IoU) overlap on the DIVA validation set.

proposal generation method uses hand-crafted rules to associate detections across consecutive frames. For example, a rule stating "a person detection and a car detection closer than 50 pixels is a proposal" can be used to generate action proposals using Prolog. However, the recall with our

Algorithm 1 Dense Proposal Generation

detections $\leftarrow Mask-RCNN($**video**$)$
orig_proposals $\leftarrow hierarchical\_clustering($**detections**$)$
new_proposals $\leftarrow$ **orig_proposals**
s $\leftarrow 15$ **for** `proposal in orig_proposals` **do**

    end
    $\mathbf{x_0}, \mathbf{y_0}, \mathbf{x_1}, \mathbf{y_1} \leftarrow spatial\_bounds($**proposal**$)$
$\mathbf{f_{st}}, \mathbf{f_{end}} \leftarrow temporal\_bounds($**proposal**$)$ **for** `f from` $\mathbf{f_{st}}$ `to` $\mathbf{f_{end}}$ `step` **s do**

    end
    new_proposals$.add(\mathbf{f}-16, \mathbf{f}+16)$
new_proposals$.add(\mathbf{f}-32, \mathbf{f}+32)$
new_proposals$.add(\mathbf{f}-64, \mathbf{f}+64)$
new_proposals$.add(\mathbf{f}-128, \mathbf{f}+128)$

final_dense_proposals $\leftarrow$ **new_proposals**

rule based proposal generation method is poor. On the other hand, hierarchical clustering provides 40% recall at a spatio-temporal IoU of 0.2. Using temporal jittering on top of it increases the recall to 85%. This shows the effectiveness of our dense proposal generation method.

2.4.2 Action Proposal Refinement and Classification

After action proposals are generated, they are refined and classified into actions. The full process is outlined in [77].

2.4.3 Post-processing

At test time, the TRI-3D network outputs the classification score and the refined temporal bounds for an input proposal. Our method for proposal generation creates many highly-overlapping action proposals, many of which are classified as the same class. We prune overlapping cuboids using 3D-NMS. The 3D-NMS algorithm is applied to each of the classes separately and considers two proposals to be overlapping when the temporal IoU overlap is greater than 0.2 and the spatial IoU overlap is greater than 0.05.

2.5 Experimental Evaluation

In this section we present and discuss various experiments which motivate our design choices and describe the overall system performance. All experimental results are reported on the DIVA validation dataset unless otherwise indicated.

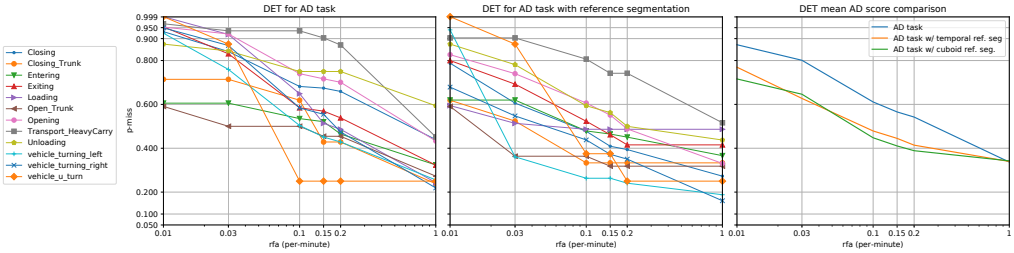


Figure 2.9: The left plot shows the per-class DET curves for the AD task. The center plot shows the per-class DET curves for TRS. The right plot shows the aggregated DET performance on the AD task compared to TRS and CRS.

Net Arch.	Input Mode	Pretrained	Crop	Acc
I3D	opt. flow	True	square	**0.716**
I3D	*RGB*	True	square	0.585
I3D	*RGB+flow*	True	square	0.704

Table 2.1: Classification accuracy for the preliminary classifier study on ground truth proposals from the DIVA validation set. The first row represents our final design. Subsequent rows contain experimental results with various modifications.

2.5.1 DIVA Evaluation Metric

To correctly and objectively assess the performance of the proposed action detection (AD) framework on DIVA, we adopt the measure: probability of missed detection P_{miss} at fixed rate of false alarm per minute $Rate_{FA}$, which is used in the surveillance event detection framework of TRECVID 2017 [160]. This metric evaluates whether the algorithm correctly detects the presence of the target action instances. A one-to-one correspondence from detection to ground-truth is enforced using the Hungarian algorithm, thus each detected action may be paired with either one or zero ground-truth actions. Any detected action which doesn't correspond to a ground-truth action is a false alarm, and any ground-truth action which isn't paired with a corresponding detection is a miss. The evaluation tool used in this work is available on Github [112]. For details of the evaluation metric we refer the reader to TRECVID 2017 1001[160].

2.5.2 System Performance

In this section we present our experiments and results for the entire system. The primary goal of the system is to take untrimmed videos as input and report the frames where actions are taking place. We perform the following experiments in order to gain a better understanding of the system performance, to discover how much impact further improvements to proposal generation may have, and how much error is simply due to improper classification.

- *Action detection (AD)*: This is the primary task of the system. Given an untrimmed video, detect and classify the begin and end frame of each action.

- *AD with temporal reference segmentation (TRS)*: Perform the AD task but with additional temporal reference segmentation, that is, our system is provided the beginning and ending frames of each ground truth action, but not the class or spatial bounds.
- *AD with cuboid reference segmentation (CRS)*: Perform the AD task with both spatial and temporal reference segmentation. In this experiment the system is provided both the start and end frame of each activity as well as the spatial cuboid bounds for each activity.

In order to perform the TRS experiment we add an additional processing step between proposal generation and classification. We adjust the temporal bounds of any proposal which temporally overlaps a reference action. If a proposal overlaps multiple reference actions then multiple copies of the proposal are generated and the temporal bounds of the copies are adjusted to match each of the reference actions. Any proposals which have no temporal overlap with any reference actions are omitted. For this experiment we did not retrain the network and the temporal refinement values are ignored.

The CRS experiment is effectively performing action classification on reference cuboids. This is the same as the preliminary experiments described in Section 3.4.2. The results described here correspond to the experiment referred to in the first row of Table 2.1.

The results of the experiments are shown in Figure 2.9. From these figures we see that there is a high degree of performance variation among different classes. For the AD task the worst performing class at $Rate_{FA}$ of 0.1 is `Transport Heavy Carry` with P_{miss} of 0.935 and the best performing class is `vehicle u-turn` with a P_{miss} of 0.25. However, we see that at $Rate_{FA}$ of 0.01 the `vehicle u-turn` becomes the worst performing action. This appears to be due to the fact that very few training and validation examples are available.

The aggregated results described in Figure 2.9, and equivalently in Table **??**, show the significant improvement gained by providing temporal reference segmentation. Surprisingly, only a small average improvement is observed when providing the cuboid reference segmentation.

DIVA Test Data: The DIVA test dataset annotations are unavailable to the authors at the time that this chapter's work was performed. We submitted a single set of results from our system for independent evaluation and have included these results as well as results from other performers on the DIVA dataset in Table **??**. We have also included results from a publicly available implementation based on [264]. We keep the identities of other the performers at the time of these experiments anonymous.

$Rate_{FA}$	0.01	0.03	0.1	0.15	0.2	1.0
AD	0.870	0.800	0.610	0.563	0.542	0.361
Temporal Ref.	0.770	0.627	0.479	0.445	0.414	0.340
Cuboid Ref.	0.716	0.646	0.448	0.410	0.389	0.342

Table 2.2: Our system's mean P_{miss} at fixed $Rate_{FA}$ on DIVA validation data. These are the values represented in the right plot in Figure 2.9.

R_{FA}	Xu *et al*. [264]	P4	P3	P2	P1	Ours
0.15	0.863	0.872	0.759	0.624	0.710	**0.618**
1.0	0.720	0.704	0.624	0.621	0.603	**0.441**

Table 2.3: Mean P_{miss} versus $Rate_{FA}$ on the DIVA test data for AD task obtained via independent evaluation. DIVA performers (P1-P4) and algorithms other than the baseline [264] have been kept anonymous by request of the independent evaluator. Only the performers better than the baseline are represented here (sorted by mean P_{miss} at $Rate_{FA}$ of 1). A lower P_{miss} value indicates superior performance, and the best performance is indicated in bold.

THUMOS'14 Dataset: In order to compare to other action detection systems we also evaluated on the Temporal Action Detection task of THUMOS'14 [110]. The THUMOS'14 Temporal Action Detection dataset contains 200 annotated validation videos, and 213 annotated test videos for 20 action classes. Since the THUMOS'14 dataset is fundamentally different from DIVA in that actions generally span the majority of the frame, we omit the hierarchical clustering phase and instead perform temporal jittering on the cuboid spanning the entire video.

Following common practice, we trained THUMOS'14 using the validation data for 5 epochs and then evaluated on the test data. The results are shown in Table 3.1. From this table we see that our system, while not designed for videos of this nature, performs well compared to state-of-the-art methods.

2.6 Conclusion

In this chapter we introduced an action detection system capable of handling arbitrary length actions in untrimmed security video on the difficult DIVA dataset. The system presented in this chapter is easily adapted to the THUMOS dataset. This system also leaves room for improvement and the modular design allows for easy integration of such future improvements. Although the DIVA evaluation metrics also includes an action-object detection task we choose to omit evaluation on this metric since our system doesn't explicitly provide object level localization of activities. We leave such improvements and extensions to future work.

tIoU	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Karaman *et al*. [114]	4.6	3.4	2.4	1.4	0.9	-	-
Oneata *et al*. [163]	36.6	33.6	27.0	20.8	14.4	-	-
Wang *et al*. [251]	18.2	17.0	14.0	11.7	8.3	-	-
Caba *et al*. [23]	-	-	-	-	13.5	-	-
Richard *et al*. [185]	39.7	35.7	30.0	23.2	15.2	-	-
Shou *et al*. [204]	47.7	43.5	36.3	28.7	19.0	10.3	5.3
Yeung *et al*. [269]	48.9	44.0	36.0	26.4	17.1	-	-
Yuan *et al*. [271]	51.4	42.6	33.6	26.1	18.8	-	-
Escorcia *et al*. [50]	-	-	-	-	13.9	-	-
Buch *et al*. [22]	-	-	37.8	-	23.0	-	-
Shou *et al*. [202]	-	-	40.1	29.4	23.3	13.1	7.9
Yuan *et al*. [272]	51.0	45.2	36.5	27.8	17.8	-	-
Buch *et al*. [21]	-	-	45.7	-	29.2	-	9.6
Gao *et al*. [65]	60.1	56.7	50.1	41.3	31.0	19.1	9.9
Hou *et al*. [99]	51.3	-	43.7	-	22.0	-	-
Dai *et al*. [41]	-	-	-	33.3	25.6	15.9	9.0
Gao *et al*. [66]	54.0	50.9	44.1	34.9	25.6	-	-
Xu *et al*. [264]	54.5	51.5	44.8	35.6	28.9	-	-
Zhao *et al*. [280]	**60.3**	56.2	50.6	40.8	29.1	-	-
Huang *et al*. [101]	-	-	-	-	27.7	-	-
Yang *et al*. [265]	-	-	44.1	37.1	28.2	20.6	12.7
Chao *et al*. [29]	59.8	**57.1**	53.2	**48.5**	**42.8**	**33.8**	**20.8**
Nguyen *et al*. [155]	52.0	44.7	35.5	25.8	16.9	9.9	4.3
Alwassel *et al*. [6]	-	-	51.8	42.4	30.8	20.2	11.1
Gao *et al*. [64]	-	-	-	-	29.9	-	-
Lin *et al*. [135]	-	-	**53.5**	45.0	36.9	28.4	20.0
Shou *et al*. [203]	-	-	35.8	29.0	21.2	13.4	5.8
Ours	52.1	51.4	49.7	46.1	37.4	26.2	15.2

Table 2.4: Comparison to THUMOS'14 performers on the mAP metric at various temporal IoUs. Missing entries indicate that results are not available. We note that Xu *et al*. [264] is the same system used to compute the DIVA V1 baseline; see Table **??**. The best performance at each tIoU is indicated in bold.

3. ACTIVITY DETECTION IN UNTRIMMED VIDEOS USING CHUNK-BASED CLASSIFIERS

3.1 Introduction

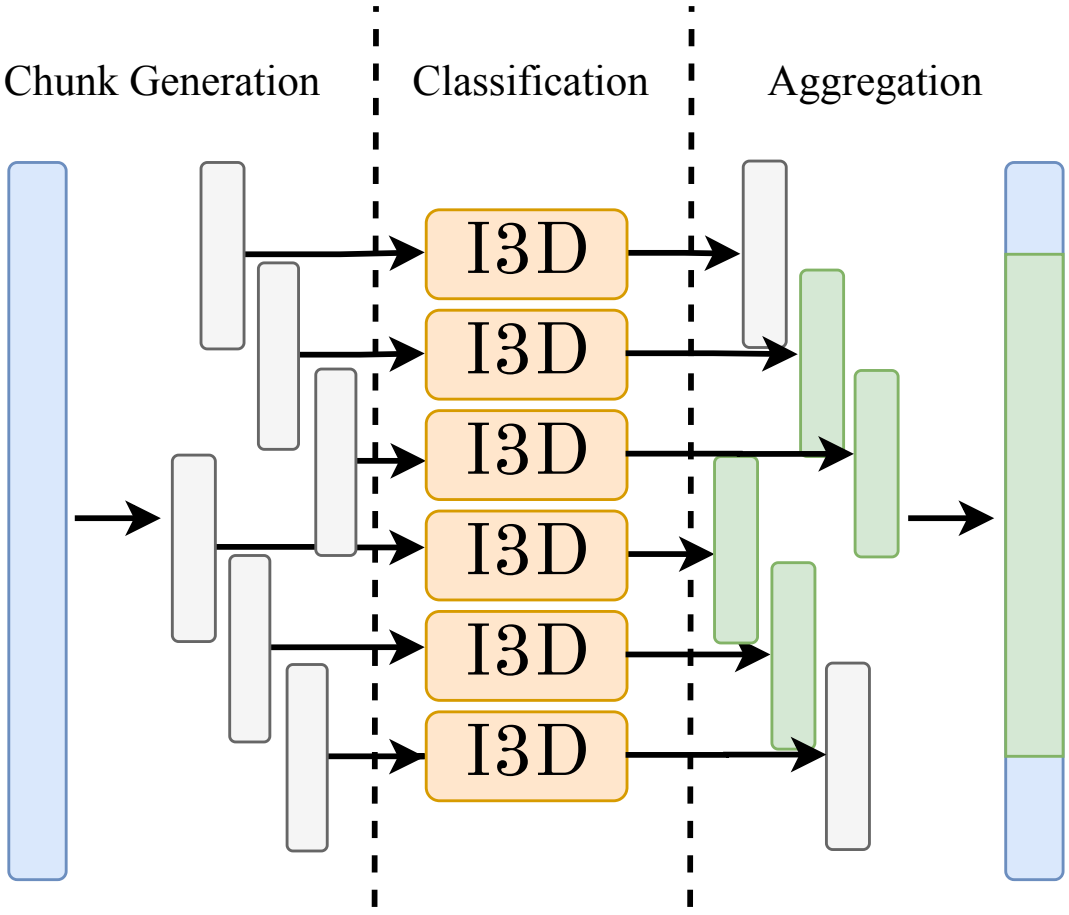


Figure 3.1: An overview of the system we present. Our procedure has three stages: we first generate a set of chunks by breaking up the video into overlapping pieces, then we classify each chunk individually. In the last stage of our system, we aggregate the classified chunks in order to create our final activity detections.

Many such approaches draw on very sophisticated architectures, making use of LSTMs [269, 271] or adaptations of object detection architectures such as Faster R-CNN [41, 264] to spatio-

temporal data.

As in Chapter 2, we focus on the problem of action detection when the data is sparse. We propose a comparatively simpler approach, leveraging the strong progress that has been made in activity classification for trimmed videos. Our main contribution is to show that a strong, carefully-trained activity classifier can achieve state-of-the-performance in activity localization by breaking the activity into many small overlapping chunks and classifying each chunk individually. Further, by learning two auxiliary tasks and applying the learned output in a clever way, it is possible to achieve even more significant performance improvements beyond the state of the art.

The resulting technique, which we call Chunk Association, is straight-forward to implement and use, and is agnostic to the chosen backbone activity classification architecture. We therefore believe that it presents a very general technique that can continue to be useful even as the state-of-the-art in trimmed activity or video classification advances, with the backbone architecture replaced as newer, better classification methods become available.

Our method also presents some very simple trade-offs depending on the use case and hardware available to run our algorithm. It can, for instance, be tuned to run faster or slower by varying the input modality used, while still maintaining a high level of performance.

We demonstrate the effectiveness of chunk association by presenting results on the THUMOS'14 [110] dataset for temporal activity localization. We also show that our method can be extended to perform spatial localization as well by performing experiments on the ActEV dataset for sparse spatio-temporal localization [159]. Finally, we present the results of a series of ablation experiments to better understand the components of our algorithm and validate the design choices that we have made.

3.2 Related Work

In recent years, a significant amount of research has been done on the problem of activity recognition. In this section we distinguish between two types of activity recognition: 1) activity classification, which refers to the problem of classifying videos containing only one activity, where each video is trimmed to the beginning and end of the activity, and 2) activity detection which refers to the problem of determining where, if anywhere, activities are occurring in a video. Despite significant progress made in activity classification, robust systems capable of activity detection for general use have remained elusive.

Activity Classification Following the success of deep CNNs in image classification by AlexNet [122], a number of early attempts were made to adapt CNN-based image classification methods to activity classification [115, 151, 212, 273]. One notable work is the two-stream CNN framework [212] which was able to obtain state-of-the-art performance by combining optical-flow and RGB using two parallel 2D CNNs. Following the two-stream work, a number of activity classification techniques were proposed which used similar strategies [54–56, 240, 252, 253].

The I3D model was introduced along with the Kinetics dataset [27]. I3D is a two-stream architecture which uses 3D convolutions. While not unique for its use of 3D convolutions [108, 234, 240], I3D was able to significantly outperform other activity classification systems like C3D [234] by taking advantage of the carefully curated, large-scale Kinetics dataset [87] and building a system that was well suited for transfer-learning on other datasets.

Activity Detection Activity detection approaches can broadly be categorized into proposal-based or end-to-end systems. In proposal-based activity detection, a collection of subsets of a video are generated to be considered as potential activities. Proposals can be viewed as high-recall, low-precision activity detections. They may be either dependent or independent of the activity class. Once the proposals are collected, classification is applied to distinguish between true and false positives, and, in the case of class-independent proposals, to determine the specific activity.

In 2014, many state of the art systems for activity detection in unconstrained videos utilized Fisher vector representation with dense trajectories evaluated over dense sliding windows [114, 163, 250, 251]. Caba et al. [23] presented an efficient proposal-based method using sparse dictionary learning. The trend of using sparse methods was relatively short-lived; however, as deep learning based approaches became more prevalent. Particularly relevant to our method, Gao et al. [65], motivated by advancements in object detection, introduced the use of cascaded boundary regression for sliding window based proposals. This was found to perform much better than other contemporary methods and provides motivation for the temporal refinement component of our system. Gleason et al. [76] propose a temporal refinement I3D system which utilizes off the shelf classification with additional regression based temporal refinement. Other approaches inspired by object-detection methods are Dai et al. [41] and Xu et al. [264] which both use Faster R-CNN [184]-based systems adapted for temporal activity detection.

Research into proposal-based methods often focus on improving proposal generation and

using off-the-shelf CNN classifiers [22, 50, 66]. Other works focus more on alterations to the classification architecture for use in activity classification and detection [202].

To get an idea of where our algorithm fits into this discussion, in this work we present an algorithm which uses short sliding window-based proposals, and focuses primarily on modifications to off-the-shelf CNN classifiers, along with improvements to proposal aggregation. By proposal aggregation, we are referring to a post-processing step which is used to combine multiple short proposals into a single activity which is discussed in detail in Section 3.3.

An alternative to proposal-based activity detection is all-in-one activity detection, which simultaneously classifies and localizes activities. Richard et al. [185] introduced a probabilistic model for temporal activity detection that jointly models segmentation and classification. Yuan et al. [272] introduce a method for temporal localization by aggregating frame-wise features using an efficient algorithm for computing structured maximal sums. Hou et al. [98] proposed the tube convolutional neural network (T-CNN) which generates tube proposals from video-clips and performs activity classification and localization using an end-to-end 3D-CNN. Kalogeiton et al. [113] extract convolutional features from each frame of a video, and stack them to learn spatial locations and activity scores.

3.3 Method

Our method is composed of three stages: chunk generation, classification, and aggregation. In the chunk generation stage, the video is broken down into overlapping chunks of 64 frames each. These chunks are then fed into a state-of-the-art video classification algorithm and classified according to their activity class. Critically, we also equip our video classifier with two auxiliary tasks: "temporal refinement" and a flag that determines if the activity end points fall within the chunk, which we call "chunk inclusion". These two tasks are easy to learn, but their output plays a critical role in the aggregation stage, where chunks are recombined into activity detections.

3.3.1 Chunk Generation

For each video, we construct a series of chunks X_i consisting of $n = 64$ frames each. Chunk generation begins by designating the first 64 frames of a given video to be the first chunk, and then, moving in strides of $s = 16$ frames at a times, progressively creating chunks out of every set of 64 consecutive frames. Thus, each frame of the video is included in at most 4 different chunks,

and neighboring chunks overlap by 48 frames.

3.3.2 Classification

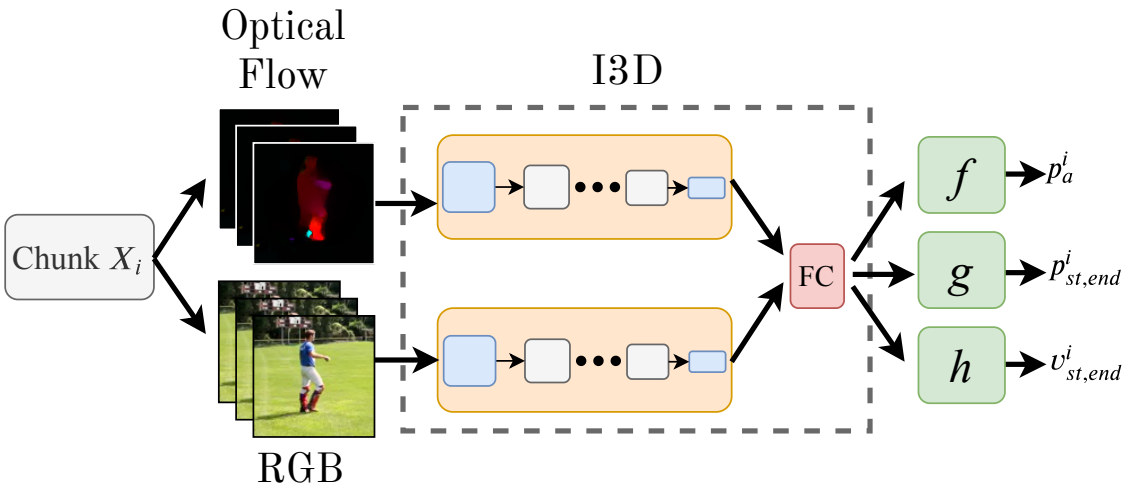


Figure 3.2: An illustration of chunk classification stage of our algorithm. We extract 64 consecutive 224×224 RGB and optical flow frames per chunk X_i. These are fed into I3D, and three outputs are produced: activity class scores p_a^i, chunk inclusion scores (p_{st}^i, p_{end}^i), and temporal localization scores (v_{st}^i, v_{end}^i)

The classification stage of our algorithm is built on existing video-classification methods. Since the video has already been divided into small, manageable chunks, each chunk can be treated as an individual video and subsequently classified.

At the time of this writing, the most powerful open source video classification algorithm is the Inflated 3D Convolution (I3D) architecture [27], we take the 64 frames that compose it and compute optical flow for each one using the TV-L1 optical flow algorithm [274]. We then randomly crop the 256×256 frames into 224×224 and feed them into I3D in a two-stream configuration, similar to the original authors in [27]. We also perform random horizontal flips for additional augmentation. However, unlike [27], we train I3D on three different tasks using three different branches: 1) an activity classification branch f, 2) a chunk inclusion branch g, and 3) a localization branch h. See Figure 3.2 for an outline of our network architecture. In addition, we also make a minor modification to the I3D architecture, and instead of averaging the logits from the two streams to get final results, we concatenate the features in the penultimate fully connected layer and produce a single set of logits.

Activity classification is the standard task for which I3D was designed. We designate a

chunk X_i as belonging to an activity class a if at least 55% percent of the chunk contains an instance of that activity class, with ties going to the activity that occurs on the most frames. We thus train our classifier using standard cross-entropy loss:

$$\mathcal{L}^i_{cls} = \sum_{a=0}^{n_{cls}} -y^i_a \cdot \log(p_a), \tag{3.1}$$

where n_{cls} is the number of classes, and y^i_a is an indicator that is 1 when chunk X_i is assigned class a and 0 otherwise. p^i_a is the softmax output of video classification branch f which corresponds to activity class a, with $a = 0$ designating that no activity has occurred.

The chunk-inclusion task on a chunk X_i simply performs two classification tasks: one to determine if the activity began within X_i, and another to determine if the activity ended within X_i. The inclusion ground truth labels, which we denote y^i_{st} and y^i_{end} for start and end inclusion respectively, are trivial to compute during chunk generation. We learn these with a cross entropy loss as well:

$$\mathcal{L}^i_{inc} = \frac{1}{2}\left(\text{bce}(p^i_{st}, y^i_{st})\right) + \left(\text{bce}(p^i_{end}, y^i_{end})\right) \tag{3.2}$$

where p^i_{st} and p^i_{end} correspond to the two separate sigmoid outputs of the chunk inclusion branch g. bce corresponds to the standard binary-cross entropy loss function.

The final task we learn is temporal refinement, which is performed similarly to [76]. Whenever an activity starts or ends within a chunk, we learn either of two temporal regression parameters r_{st} or r_{end}, which we compute in the ground truth as

$$(r^i_{st}, r^i_{end}) = \left(\frac{\hat{t}_{st} - t^i_c}{n'}, \frac{\hat{t}_{end} - t^i_c}{n'}\right). \tag{3.3}$$

where t^i_c is the center of chunk X_i, $(\hat{t}_{st}, \hat{t}_{end})$ are the start and end frames of the ground truth activity assigned to chunk i, and $n' = n/2$, where $n = 64$ is the chunk length. As an example, if the start and end of the chunk were perfectly aligned with the ground truth, we would have $(r^i_{st}, r^i_{end}) = (-1, 1)$.

We train these values using a smooth$_{L1}$ loss [184]. The temporal localization branch h produces two regression outputs $v^i = (v^i_{st}, v^i_{end})$ according to

$$\begin{aligned}\mathcal{L}^i_{loc} = y^i_{st} \cdot \text{smooth}_{L1}(r_{st} - v^i_{st}) \\ + y^i_{end} \cdot \text{smooth}_{L1}(r_{end} - v^i_{end}).\end{aligned} \tag{3.4}$$

We note that the y^i_{st} and y^i_{end} terms prevent the learning of regression when an activity does not begin or end within the chunk.

Our final loss can thus be written:

$$\mathcal{L}_{full} = \mathcal{L}_{cls} + \lambda \mathbb{I}_{a\geq 1}(\mathcal{L}_{inc} + \mathcal{L}_{loc}), \tag{3.5}$$

where $\mathbb{I}_{a\geq 1}$ is an indicator function that is equal to 1 when the ground truth action has an index greater than 0, and is equal to 0 otherwise (*i.e.* is equal to 1 when the ground truth does not designate "no activity"). λ is a weight parameter that we experimentally set to $\lambda = 0.25$.

3.3.3 Chunk Aggregation

Chunks are aggregated by combining adjacent chunks with similar predictions, informed by the predicted inclusion and temporal refinement values. See [79] for more information.

3.3.4 Spatio-Temporal Activity Detection

Our algorithm can also be extended to work in the context of spatio-temporal activity detection, in the situation where activities are sparse both temporally and spatially, by chunking spatio-temporal proposals instead of the entire video. See [79] for more information.

3.4 Experiments

In this section, we present a series of experiments designed to help understand the impact of various components of our system. The majority of experiments presented in this section were performed on the THUMOS'14 action detection dataset [110]. To show that this approach generalizes to other scenarios we provide additional evaluation results on the ActEV [159] dataset for spatio-temporal activity detection.

3.4.1 Training

We train the backbone I3D network on THUMOS'14 using SGD with learning rate 0.01, momentum 0.9, and weight decay 0.0005. We train on eight Nvidia 1080Ti GPUs for a total of two epochs, lowering the learning rate to 0.001 in the second epoch. We use a batch size of 24 for two-stream training experiments (RGB and optical flow) and 48 for single stream experiments (optical flow or RGB individually). We initialize I3D with weights pre-trained on the Kinetics dataset [27].

For most of the time, no activities are happening and if we trained on all the chunks of our system, we would inevitably bias heavily towards the "no activity" class. Thus, when creating chunks for training, we carefully control the number of chunks we expose the network to, enforcing that only 10% of the samples seen by the network are negatives.

3.4.2 THUMOS'14

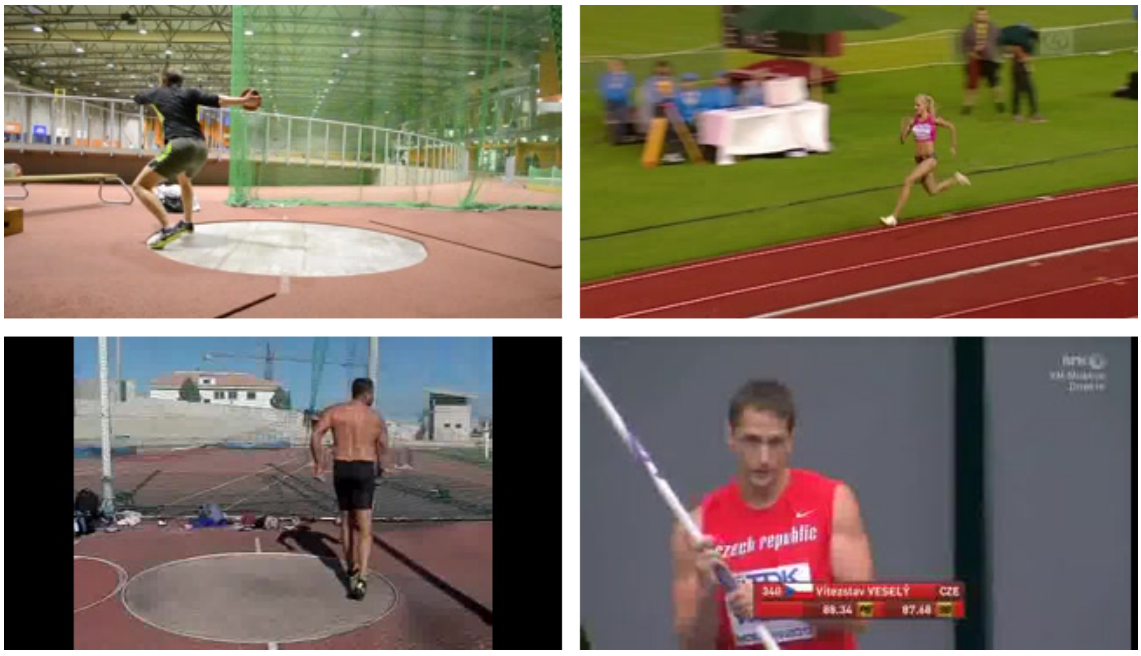


Figure 3.3: Sample frames from videos within the THUMOS'14 dataset.

Dataset

THUMOS'14 is a temporal action detection dataset which consists of 2765 trimmed training videos, 200 untrimmed validation videos, and 213 untrimmed test videos. Since the training data is trimmed, we follow the common practice and perform the majority of our experiments by

training on the validation set and evaluating on the test set [64, 66, 76, 264]. The majority of ablation experiments are performed using a single-stream configuration, using optical flow frames, except where otherwise mentioned.

The only exception to the aforementioned train/evaluate strategy is for experiments used to select the hyperparameters described in Section 3.3 (*e.g.* T_{inc} and s) . For these experiments, we instead use a 70/30 split of the THUMOS validation set to perform training and validation respectively, so as not to evaluate on the test split. This ensures that we don't overfit the hyperparameters of our method to the test data. In order to enforce that the 70/30 split of videos contains the appropriate number of instances of each class, we employ the simple strategy of repeatedly sampling random 70/30 splits until the number of training samples for each class is between 1.05 and 3.38 times the number of instances in validation.

Figure 3.3 shows sample frames from videos within the THUMOS'14 dataset. THUMOS performance is measured in the provided THUMOS'14 scoring tool using the mean Average Precision (mAP) metric, performed at several different temporal Intersection over Union (tIoU) thresholds.

Table 3.1 shows the final performance of our algorithm on THUMOS'14, as compared to other published algorithms. To achieve these final numbers, we use the I3D backbone in a two-stream configuration, taking both the RGB and optical flow frames as inputs.

Ablation experiments

Inclusion and Localization Chunk inclusion and localization (*i.e.* the losses $\mathcal{L}_{inc}$ and $\mathcal{L}_{loc}$) are the key novelties of our approach, providing strong improvements to our algorithm's performance. We demonstrate the importance of these tasks by performing two separate experiments. The first of these, shown in Table 3.2, indicates our performance when we remove each of these components from training entirely. In both cases, performance drops significantly. Unsurprisingly, both localization loss and inclusion loss help with temporal localization so the greatest impact can be seen at the higher tIoU values where precise start and end times are most important.

We also perform another set of experiments where the inclusion and localization values are learned, but are not used during inference. These results are shown in Table 3.3. Here we see the importance of performing the inclusion thresholding and temporal refinement, since removing either one significantly decreases performance. Additionally, the improvements from each are

tIoU	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Karaman *et al*. [114]	4.6	3.4	2.4	1.4	0.9	-	-
Oneata *et al*. [163]	36.6	33.6	27.0	20.8	14.4	-	-
Wang *et al*. [251]	18.2	17.0	14.0	11.7	8.3	-	-
Caba *et al*. [23]	-	-	-	-	13.5	-	-
Richard *et al*. [185]	39.7	35.7	30.0	23.2	15.2	-	-
Shou *et al*. [204]	47.7	43.5	36.3	28.7	19.0	10.3	5.3
Yeung *et al*. [269]	48.9	44.0	36.0	26.4	17.1	-	-
Yuan *et al*. [271]	51.4	42.6	33.6	26.1	18.8	-	-
Escorcia *et al*. [50]	-	-	-	-	13.9	-	-
Buch *et al*. [22]	-	-	37.8	-	23.0	-	-
Shou *et al*. [202]	-	-	40.1	29.4	23.3	13.1	7.9
Yuan *et al*. [272]	51.0	45.2	36.5	27.8	17.8	-	-
Buch *et al*. [21]	-	-	45.7	-	29.2	-	9.6
Gao *et al*. [65]	60.1	56.7	50.1	41.3	31.0	19.1	9.9
Hou *et al*. [99]	51.3	-	43.7	-	22.0	-	-
Dai *et al*. [41]	-	-	-	33.3	25.6	15.9	9.0
Gao *et al*. [66]	54.0	50.9	44.1	34.9	25.6	-	-
Xu *et al*. [264]	54.5	51.5	44.8	35.6	28.9	-	-
Zhao *et al*. [280]	60.3	56.2	50.6	40.8	29.1	-	-
Huang *et al*. [101]	-	-	-	-	27.7	-	-
Yang *et al*. [265]	-	-	44.1	37.1	28.2	20.6	12.7
Chao *et al*. [29]	59.8	57.1	53.2	48.5	42.8	33.8	20.8
Alwassel *et al*. [6]	-	-	51.8	42.4	30.8	20.2	11.1
Gao *et al*. [64]	-	-	-	-	29.9	-	-
Lin *et al*. [135]	-	-	53.5	45.0	36.9	28.4	20.0
Gleason *et al*. [76]	52.1	51.4	49.7	46.1	37.4	26.2	15.2
Yang *et al*. [266]	-	-	51.8	41.5	32.1	22.9	14.7
Murtaza *et al*. [148]	-	-	54.9	47.2	41.5	37.5	**31.6**
Ours	**67.41**	**66.96**	**62.6**	**56.87**	**48.99**	**39.19**	27.82

Table 3.1: Comparison to other algorithms on the THUMOS'14 based on the mAP metric at various temporal IoUs. Missing entries indicate that results are not available. The best performance at each tIoU is indicated in bold.

not independent since activating both of them together yields a bigger increase in performance than they each add individually; for instance, tIoU at 0.7 goes up about 4 points when inclusion thresholding is turned on and 10 points when temporal refinement is used, but nearly 17 points when both are used together.

Input Modality I3D can be trained using only the raw RGB frames of the input chunks, optical flow frames, or both at once. In Table 3.4, we compare the results of training on these three modalities. Our key observation is that optical flow significantly out-performs RGB, while

Inclusion	Localization	0.1	0.5	0.7
No	No	59.09	32.19	16.08
Yes	No	61.53	37.04	21.03
No	Yes	60.45	35.9	19.25
Yes	Yes	**64.56**	**48.35**	**28.22**

Table 3.2: The effects of chunk inclusion and localization on performance at 3 different tIoU thresholds. All results are attained using optical flow as the only input modality. We also note that for this table, when inclusion or temporal localization are used, we also apply inclusion thresholding or temporal refinement respectively. Best results are in bold.

Inclusion	Localization	0.1	0.5	0.7
No	No	61.68	31.03	11.46
Yes	No	64.06	40.55	15.68
No	Yes	61.4	37.51	21.0
Yes	Yes	**64.56**	**48.35**	**28.22**

Table 3.3: The result of training the system using inclusion and localization loss but selectively ignoring those results during evaluation. This is to demonstrate that the performance improvements are not solely a result of multi-task learning.

I3D Streams	0.1	0.5	0.7
RGB	61.56	42.56	22.85
Flow	64.56	48.35	**28.22**
Joint (RGB+Flow)	**67.41**	**49.0**	27.82

Table 3.4: Classification accuracy for the three different input modalities on the THUMOS'14 dataset. Each row represents one of the modalities tested at 3 different temporal IoU thresholds. Best results are in bold.

combining both modalities into a two-stream network is optimal, similar to the observations made by [27] in the original I3D paper. The choice of modality also has important ramifications for the run-time of the algorithm, which we explore in more detail in Section 3.4.3.

Aggregation In order to motivate our choice of aggregation algorithm, we present in Figure 3.4, a plot that shows the predictions of our chunks mapped against the ground truth of a given video. What we find is that, in general, chunks appear to do a good job classifying activities on their own, which motivates our choice to aggregate by connecting adjacent chunks.

In Table 3.5, we compare our chosen chunk aggregation algorithm to a slightly modified algorithm where, instead of merely connecting adjacent chunks, we also connect chunks that are of the same class with up to one or two differing chunks in between. We find that this actually decreases performance, which suggests that chunks being misclassified in the middle of activities are not a serious issue in THUMOS'14, and allowing skipping only hurts performance by merging

Aggregation	0.1	0.5	0.7
2 Skips	58.68	35.93	20.00
1 Skip	61.57	38.46	21.26
No Skips	**64.56**	**48.35**	**28.22**

Table 3.5: The proposed aggregation algorithm compared to an alternate aggregation algorithm in which chunks can be matched even if there are chunks in between of a different class. "1 Skip" and "2 Skip" refer to algorithms that merge chunks with one or two different entries between them, and "No Skips" is the algorithm we describe in Section 3.3.3. Best results are in bold.

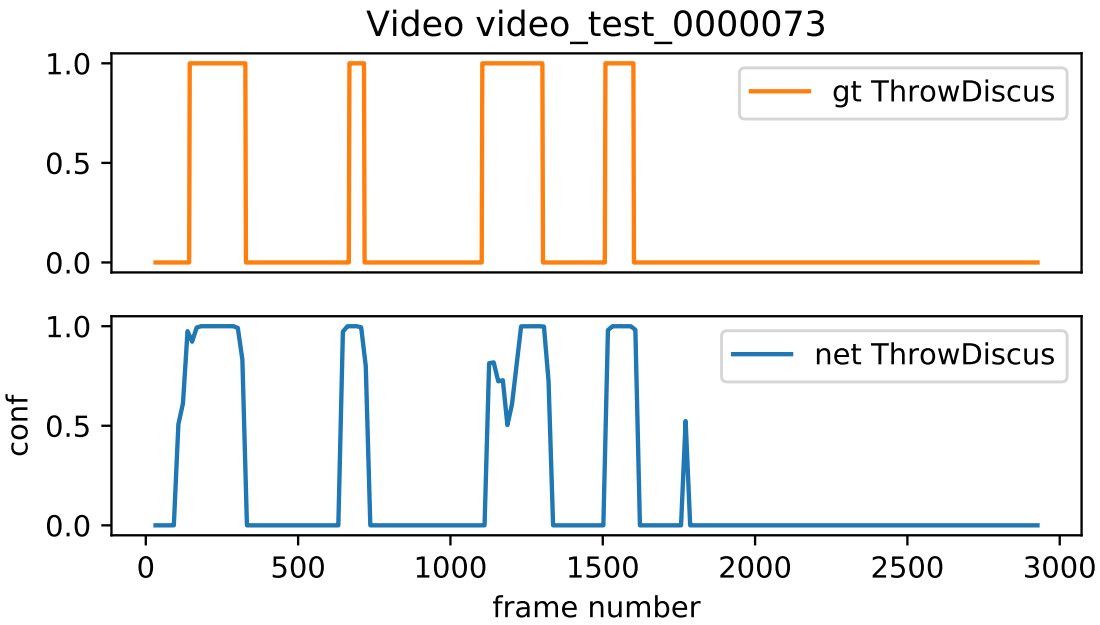


Figure 3.4: A plot of the network's per-chunk predictions (bottom) vs. the ground truth (top) on a sample video in the test set. The bottom plot shows the confidence value of each chunk in the sequence. Chunk confidence is plotted from the location of the chunk's center frame. For more intuitive visualization, the confidence values are suppressed to zero when the ThrowDiscuss class is not the maximal response for the chunk.

chunks that should not be merged.

3.4.3 Timing-Performance Trade-offs

Here, we analyze the inference speed of our system, and show that with the right modifications our system can still achieve near state-of-the-art performance while running at speeds well above real time on a single GPU. Table 3.6 shows the results of these experiments. When optical flow is used, the vast majority of computation time is spent extracting the optical flow frames, and therefore multiple GPUs are required to perform real time inference. On the other hand, when inference is performed using only RGB frames, it is possible to perform computations significantly faster. With RGB input, a single GPU is enough to achieve processing speeds of 235 FPS, significantly faster than real time. In terms of performance, Table 3.4 shows that RGB, while worse than optical flow or RGB+optical flow, still achieves results competitive with the previous state-of-the-art. Disk read times were not considered in this timing analysis.

Thus, in addition to being simple, our algorithm is very well suited to real-world applications where speed would be an important factor. If, in addition, detection needs to be performed

Mode	# GPUs	sec/chunk	Forward FPS	Total FPS
RGB	1	0.055	235.35	235.35
Flow	1	0.046	291.67	22.77
RGB+Flow	1	0.108	126.11	20.65
RGB	8	0.026	467.74	467.74
Flow	8	0.0208	612.0	121.93
RGB+Flow	8	0.0467	268.54	97.17

Table 3.6: Timing analysis of our algorithm under various configurations. "Forward FPS" refers to the frame rate of a forward pass through the network, while "Total FPS" includes the time to process input (*i.e.* optical flow computation). Timing of disk access was not included in this analysis.

on live streamed video, then merely introducing a several-second lag would be sufficient to detect new activities as they happen. Further, since optical flow computation is the only stage of the algorithm with significant computational overhead, it may also be possible to achieve the state-of-the-art performance of optical flow with speeds much closer to that of RGB by estimating the flow or motion information directly from compressed video [253, 261, 277].

3.4.4 ActEV: Spatio-Temporal Detection

Dataset

The ActEV dataset [159] is a spatio-temporal activity detection dataset featuring 18 different activities over 64 training videos, 54 validation videos, and 246 test videos with annotations withheld. All videos within the ActEV dataset are high resolution (1200×720 or 1920×1080) yet the subjects performing actions within the videos are very tiny by comparison, generally ranging from 20 to 180 pixels in height. Therefore, actions within the ActEV dataset tend to be spatially sparse, taking up only a very small portion of the scene. One of the primary challenges of the ActEV dataset is to intelligently avoid processing extraneous pixels. We do this by introducing a proposal-based method for generating chunks, as discussed in Section 3.3.4.

Results

In this section we report the results of our method on the ActEV validation and test sets. The primary metric for ActEV is probability of miss (p-miss) at a false alarm rate of 0.15 which is computed using the provided evaluation tool downloaded from Github [112]. We also report the N-MIDE metric on validation for which lower scores correspond to better temporal localization. Both of these metrics are described in detail in TRECVID 2018 [10].

Method	P-Miss	N-MIDE
Gleason et al. [76]	67.50%	0.239
Ours	**61.71%**	**0.181**

Table 3.7: ActEV validation dataset results. In the left column we report weighted average probability of miss (weighted p-miss), where the weighting is applied to account for any class imbalance. In the right column we report the N-MIDE metric, where lower N-MIDE scores refer to better temporal localization. Both metrics are reported with a fixed rate of false alarm of 0.15.

Method	P-Miss
Aakur et al. [2]	93.4%
Xu et al. [264]	91.30%
Gleason et al. [76]	75.03%
MUDSML	**69.85**%
Ours	76.44%

Table 3.8: ActEV test dataset results. Here we report the weighted probability of miss of our system, which was evaluated on an independent evaluation server [159]. The metric is reported at a rate of false alarm of 0.15.

Table 3.7 shows the results of our algorithm on the ActEV validation data using the method described in Section 3.3.4. In the table we included validation results using the TRI3D system described in [76], which, to our knowledge are the best published results on ActEV. The public leaderboard for the 2018 ActEV challenge was available at [159] where the top performing system achieves a probability of miss of 69.85% at a rate of false alarm of 0.15. Since the top performing system is unpublished as of the submission of this manuscript, it is difficult to comment on where the gain in performance comes from. This is particularly true since teams were allowed and encouraged to collect and use proprietary data to improve performance. Our system significantly out-performed the only other published systems which we know to have been evaluated on ActEV test [2, 264].

We observe that while we achieve a significant improvement over TRI3D [76] on the validation dataset, our system performs slightly worse on the test split as shown in Table 3.8. Since the annotations are not available for the test split it is difficult to draw any conclusions from this phenomenon, however, by contrasting our approach to TRI3D we have some thoughts on this.

On the validation dataset, our method outperforms TRI3D on all but three of the eighteen activities: `activity_carrying`(+6.1%), `Closing`(+5.4%), and `Loading`(+5.4%), where the deltas reported here are the differences in average p-miss at 0.15 rate of false alarm. This indicates that the discrepancy is likely not due to some sort of biasing of our method towards

Figure 3.5: Spatial sparsity in ActEV. On the left is an example frame from ActEV, and on the right is an example frame from THUMOS'14. The green box in the ActEV image shows the spatial location of an instance of the `Closing` activity which covers only a small percentage of the overall image. Contrast this to the THUMOS'14 image where the activity `Cricket_Bowling` takes up a much greater proportion of the image. This figure was original presented in [76].

certain actions. We also rule out the detector and clustering algorithm as being the cause of the discrepancy as both TRI3D and our method use the same approach. The difference between our algorithm and TRI3D at the core is that we use short, overlapping chunks, which are then aggregated, as opposed to varying length cuboids which are not aggregated. Our hypothesis is that the activities in the test dataset are more difficult to determine given the short temporal spans, and instead benefit from the high level context provided by long cuboids. We leave it to future work to test this hypothesis by extending our system to incorporate chunks of varying length.

3.5 Conclusion

We have presented a simple but effective approach to activity detection and classification in untrimmed videos. We have shown, through empirical analysis on the THUMOS'14 and ActEV datasets, that it is in fact possible to achieve results in untrimmed activity detection comparable to state-of-the art simply by making use of strong existing activity classification systems, and that with the addition of two auxiliary tasks it is possible to push these results significantly farther.

4. SELF-DENOISING NEURAL NETWORKS FOR FEW SHOT LEARNING

4.1 Introduction

Despite the ubiquitous power of modern deep learning techniques for image and video classification, learning from very few examples still remains a difficult problem. A visual recognition system deployed in a real world setting may need to identify new classes of which it has only seen one or two examples. Learning in this low-data regime, where one might have access to as little as a single labeled example, generally requires entirely different techniques than traditional supervised learning, and the pursuit of these techniques has launched an entire sub-field of machine learning research known as "few shot learning".

Few shot learning is generally characterized by a two-stage approach. In the first stage, the network is pretrained in a supervised setting with a large labelled dataset of known classes. In the second stage, the network is exposed to a limited number of labeled examples in novel classes that were not included in the pretraining dataset, usually with as few as five or even one sample per class. The final system is then evaluated against a held-out test set of images belonging to the novel classes.

Under this paradigm, the machine learning community has developed a large number of techniques. These techniques can themselves be classified into a broad set of categories, ranging from meta-learning techniques that quickly learn to perform meaningful parameter updates when presented with new data [61, 129, 156, 190, 249], to feature hallucination techniques that generate additional samples for training [88, 256] and transductive techniques that make use of the correlations between novel class samples during inference [45]. In this work, we follow the lead of other metric-based approaches [214, 223, 247] which perform classification using highly representative features. That said, our approach is orthogonal enough to many existing metric-based approaches, and simple enough to implement, that it can be easily combined with existing approaches to achieve easy few shot performance gains.

The conceptual backbone of our approach is similar to that of Denoising Auto-Encoders

[246] (DAEs), which were shown to be useful for few shot learning by Gidaris *et al*. [70]. DAEs are effective for metric learning because they build robust feature extractors that, by removing noise from features, move features towards their most likely configuration in feature space. At inference time, the noise is not applied and the feature is refined to be more representative, in effect creating a more robust prototype for few shot learning.

Our approach expands on this concept by making a few key observations. The first is that in order to counteract the effects of noisy features, it is not actually necessary to reconstruct the features precisely. Instead, it is sufficient merely to ensure that the network can accomplish the same classification task it would have preformed otherwise. This can be enforced simply by adding an additional classification loss immediately after the noise is applied. Our second observation is that this denoising process need not be a separate module from the rest of the network applied iteratively, but can instead be integrated directly into existing network training with almost no architectural changes or differences in learning hyperparameters.

The final result is a relatively straightforward addition to existing neural network architectures. The features of an existing architecture, such as a ResNet [92], are modified before each block with some form of noise. The network is also augmented with auxiliary losses after each block, similar to those used to train *e.g*. Inception [225]. The end result, which we refer to as a Self-Denoising Neural Network (SDNN), is a network that continually refines its features as the network deepens, ultimately producing features that are altogether more robust than the original architecture by itself.

We also show, through detailed ablation experiments , the significance of many of the observations discussed above. For instance, we observe that the auxiliary losses used by our method are essential; merely adding noise to the network as it trains has nearly no effect on performance. It is only by constantly enforcing that the features produce the same evaluation results that denoising produces more robust features.

In summary, our contributions are as follows:

- We present a novel approach to metric learning for few shot visual tasks termed Self-Denoising Neural Networks (SDNNs). In principle nearly any existing deep architecture for visual recognition can be converted into an SDNN using a few simple modifications to the training pipeline.

- We demonstrate the effectiveness of SDNNs on the *mini*ImageNet, *tiered*-ImageNet, CIFAR-FS, and ActEV Surprise Activities datasets. We also show that our method is very general, and can be easily added to existing metric-based few shot techniques.
- We empirically analyze our SDNN architecture through a series of detailed ablation experiments.

4.2 Related Works

Few shot learning [126] for deep learning is a rich area of research, with prior work falling into several broad categories. One of the more prominent categories is optimization-based meta-learning [61, 129, 156, 190, 249]. These methods "learn-to-learn" by pretraining models that are able to produce gradient updates that facilitate quick fine-tune on small numbers of samples. Methods like MAML [61], LEO [190], and Reptile [156] do this by incorporating the fine-tuning step into the learning stage.

Other metric learning techniques deal with minimal data by hallucinating additional samples for training [88, 256]. Still others improve performance by incorporating more general machine learning techniques, like self-supervised learning [68, 217] or knowledge distillation [231].

Metric-based few shot learning algorithms are another large family of techniques [120, 134, 164, 199, 214, 223, 236, 247]. These methods operate by embedding inputs into a feature space where images of the same class are naturally close together by some metric. For example, RelationNet [223] accomplishes this using cosine similarity, and MatchingNet [247] uses Euclidean distance. Though already quite simple, many of these methods [214, 223, 247] utilize a pretraining scheme that mimics the few shot training stage by sampling a small support set of sample classes during training. Other metric-based methods like Cosine Classifiers [30, 69, 174] remove even this training complication, providing a methodology for building robust features without the need for support sets during training. In this chapter, we design our method on top of these Cosine Classifier based methods, motivated in large part by a pursuit of simplicity.

4.3 Method

In the standard few shot learning problem setup for visual tasks, we are initially given access to a set of images [1] and labels $(x_b, y_b) \sim \mathcal{D}^b_{tr}$ from some predetermined set of base classes, where $\mathcal{D}^b_{tr}$ is a training distribution and x_b and y_b are, respectively, images and labels from the N_b base classes $C_b = \{1 \ldots N_b\}$ of our training set. After performing some form of pretraining using $\mathcal{D}^b_{tr}$, we are given a second set of training images and labels $(x_n, y_n) \sim \mathcal{D}^n_{fs}$, with all labels from a set of N_n novel classes $C_n = \{N_b + 1, \ldots, N_b + N_n\}$ which are disjoint from the previously-learned base classes (*i.e.* $C_b \cap C_n = \emptyset$). Importantly, the number of samples (x_n, y_n) provided in this stage is extremely small, often as low as five or even just one. Finally, performance is measured by the classification accuracy of the final system on a test set of samples from the novel classes, $(x_n, y_n) \sim \mathcal{D}^n_{ts}$. In the following text, we first discuss some prerequisites in Sections 4.3.1 and 4.3.2, then provide a detailed description of our method in Sections 4.3.3 and 4.3.4.

4.3.1 Cosine Classifiers

Cosine Classifiers are widely used for few shot learning, originally explored by [69] and [174]. In this context, few shot learning is performed by initially training a neural network feature extractor $f = F(x)$ on the data $(x_b, y_b) \sim \mathcal{D}^b_{tr}$ as described above, as well as a set of weight vectors $W_b = [w_1, \ldots, w_{N_b}]$. The critical difference between Cosine Classifier pretraining and standard supervised training comes in the per-class logit computation. In standard supervised training, the logits for class i would be computed as the dot product between the weights and the embedding: $z_i = f^T w_i$. In contrast, a Cosine Classifier computes the logits as the cosine distance between the weights and the features:

$$z_i = \cos(f, w_i) = \frac{f^T w_i}{\|f\| \|w_i\|} \tag{4.1}$$

which has the advantage of producing features with reduced intra-class variance.

The logits are then converted into probabilities p_i using the softmax function

$$p_i = \exp(\gamma z_i) / \sum_{j \in C_b} \exp(\gamma z_j)$$

[1] Though we use the term image here, x_* could also be a video or any other form of input data.

where γ(4.2)

is a learned inverse temperature parameter. Given these final predictions p_i for each class, the network is trained by optimizing standard cross entropy loss:

$$\mathcal{L}(p) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{tr}^{b}}\left[-\log p_y\right]. \tag{4.3}$$

During the novel-class training stage, the weight matrix W_n for novel classes is computed as $w_i = \bar{z}_i, \forall i \in C_n$, where $\bar{z}_i$ is the average over all values of z_i for all images x_n in the novel class training set with label $y_n = i$. In the final inference stage, the new matrix W_n is used in place of W_b to compute probabilities as described above. In other words, the extracted features of the novel classes are directly averaged across samples of the same class and used as weights for further classification.

4.3.2 Denoising Autoencoders

Originally introduced and explored by Vincent *et al*. [246], Denoising Autoencoders (DAEs) are a form of autoencoding neural network designed to improve feature robustness by reconstructing a given feature vector into a more likely configuration. A DAE $r(\cdot)$ operates on a feature vector $\hat{f} = g(f)$ that his been corrupted from its original form f with some type of noise $g(\cdot)$ (*e.g*. additive Gaussian noise) by attempting to construct a new feature vector $r(\hat{f})$ which is as close as possible to the original feature vector f.

Alain *et al*. [5] show that, in the case of a DAE trained with additive Gaussian noise, preforming inference on noiseless input will actually cause the DAE to estimate the gradient of the density function of its input, and as such the vector $(r(f) - f)$ will point towards more likely configurations of f, *i.e*. the manifold of the input data. Previously, Gidaris *et al*. [70] made use of this fact to refine the novel feature weights W_n using DAEs and improve few shot performance. We use this work as inspiration for our work.

4.3.3 Self-Denoising Neural Networks

We take the above observations about DAEs and use them to expand existing Cosine Classifier-based few shot learners in a different direction from previous research. Rather than add an additional DAE module to the end of the network as was done in [70], we treat the entire existing

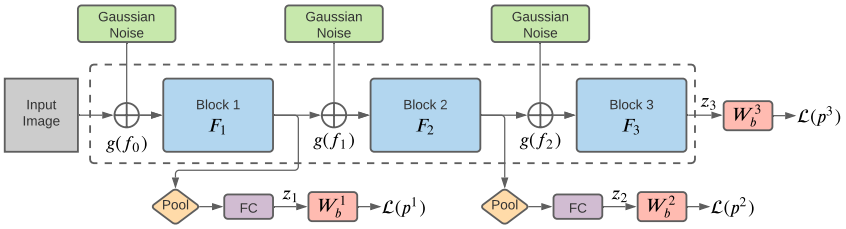


Figure 4.1: A sample SDNN architecture with Gaussian noise. During training, the features after each block are perturbed using additive Gaussian noise. After each block, the features are pooled and passed through an FC layer so that auxiliary losses $\mathcal{L}(p^l)$ can classify the modified features. During inference, no noise is added, and the weights W_b^l are replaced with the weights W_n^l.

network as a set of denoising modules, making minimal architectural changes and achieving considerable performance improvements over simply using a DAE.

To define this new architecture, we first break our feature extraction network F into multiple blocks F_1, F_2, F_3 [2]. Many modern architectures such as ResNet [92] are already naturally organized into blocks, making this division extremely straight-forward. Rather than simply injecting noise at the end and having an additional DAE module denoise the features $f = F(x) = F_3(F_2(F_1(x)))$ produced by the full network, we instead treat each block F_l as its own denoising module by injecting noise at the beginning of each block and denoising the signal before it reaches the next one.

Specifically, given features $f_{l-1} \in \mathbb{R}^{h_{l-1} \times w_{l-1} \times c_{l-1}}$ (where h_l, w_l and c_l represent the height, width, and channels of the feature map at layer l) produced from block F_{l-1}, we compute f_l during initial training as $f_l = F_l(g(f_{l-1})) \in \mathbb{R}^{h_l \times w_l \times c_l}$, where $g(\cdot)$ is a generic noise function that perturbs its input with any of several more specific noise functions, as discussed below in Section 4.3.4. This modification is motivated in part by the observation made in Section 4.3.2 that a DAE naturally adds a vector to the input features which points it towards the data manifold. Our hope is to produce a layer that projects towards the data manifold and processes the features at the same time.

Ideally, we would like f_l to produce an input that is robust to the noise that was added to f_{l-1}. Applying a standard DAE structure would mean reconstructing f_{l-1} from $g(f_{l-1})$, but this would require either adding additional layers to create the a DAE, or enforcing that f_l matches f_{l-1}. Since our goal is to retain the original architecture, both of these solutions are undesirable.

[2] For this example, we use only 3 blocks because that corresponds to the WideResNet [275] network featured most prominently in our experiments section. Other networks, such as ResNet10 [92] or Inception [225] might be broken up into differing numbers of blocks.

Instead, rather than force the network to explicitly denoise f_{l-1}, we enforce that f_l remains useful for classification despite the noise. To do this, we add an additional auxiliary classification loss after each block. More specifically, given an uncorrupted feature $f_l \in \mathbb{R}^{h_l \times w_l \times d_l}$, we perform

$$\begin{aligned} z_i^l &= \cos(\text{FC}(\text{MaxPool}(f_l)), w_i^l) \\ p_i^l &= \text{softmax}(\gamma z_i^l) \end{aligned} \tag{4.4}$$

where FC is a fully-connected layer, and MaxPool is a max-pooling operation across the spatial dimensions of f_l. The final predictions p_i^l are then learned using $\mathcal{L}(p_i^l)$, the same cross entropy loss defined in equation 4.3. The weights w_i^l are similar to the weights W_b described in Section 4.3.1, however they are expanded to include a layer index $W_b^l = [w_1^l, \ldots, w_{N_b}^l]$; in other words, the auxiliary classifiers function identically to the classifier at the end of the unmodified network. This constrains each block to produce the same logits as one another, enforcing the requirement that the network produces features meaningful for classification at every level of the network. Auxiliary losses of this style have been used in classification problems for a long time, *e.g.* when training Inception [225]. The application of noise in previous layers, however, expands the purpose of these additional losses beyond gradient flow and regularization, using them to counteract the added input noise. As we show empirically in Section 4.4.4, these losses are necessary to see gains from denoising. See Figure 4.1 for an illustration of the full architecture.

During the novel class learning stage, noise is no longer added and the new weights W_n^l are constructed in the same way as the weights for the Cosine Classifier by assigning $w_i^l = \bar{z}_i^l, \forall i \in C_n$, where $\bar{z}_i^l$ is the average over all values of z_i^l for all images x_n in the novel class training set with label $y_n = i$. The final predictions p_i are computed from the average of all predictions at each layer: $p_i = \sum_l p_i^l$, where l indexes over the number of blocks in the network (either 2 or 3 in all experiments within this chapter).

In summary, we modify existing neural network training schemes by repeatedly adding noise to the network while forcing it to produce the same features despite the added noise. Thus, though the network is not performing a true denoising reconstruction, it is still developing a meaningful understanding of the feature space, and when the noise is removed during the novel training and inference stages the network is still able to implicitly refine the previous stage's features into a more likely configuration.

As a final note we discuss the similarities of our method with stacked DAEs [245]. Both approaches involve repeated denoising, however we emphasize certain important differences. First of all, stacked DAEs are trained in stages, requiring specific architectures that may require need ot be carefully crafted. Our method, on the other hand, can be easily applied to any existing network architecture just by adding individual pooling and fully connected layers in a few key locations, making it much easier to train and introducing extremely few new hyperparameters. Our method also trains in a single pass using the same hyperparameters as its base architecture, versus the multi-stage training of stacked DAEs. Finally, at a more technical level, our method does not actually autoencode anything; the features are only denoised in the sense that they attempt to produce the same classification outputs as noiseless features. There is no actual reconstruction that occurs.

4.3.4 Types of Noise

The noise function $g(\cdot)$ described in Section 4.3.3 can take any number of forms. In this chapter, we will explore two different constructions: Dropout noise and Additive Gaussian noise.

Dropout Noise: Dropout noise is modelled after the widely used method of Dropout [216]. Dropout works by first generating a random binary mask $m \in \mathbb{R}^{h \times w \times c}$, where $m_{i,j,k} = 0$ with probability p_{drop}, and then computing $g(f) = f \odot m$, where $\odot$ is an element-wise product. We modify dropout by making it non-spatial in the same manner described below for Gaussian noise, *i.e.* sampling $m \in \mathbb{R}^c$ and broadcasting it into the spatial dimensions.

Additive Gaussian Noise: An additive Gaussian noise function $g(\cdot)$ is defined simply as $g(f) = f + v$ where $v \in \mathbb{R}^c$ and $v_i \sim \mathcal{N}(0, \sigma)$ for some standard deviation parameter σ, with $i \in \{1, \ldots c\}$ indexing the c channels of f . Note that we have $f \in \mathbb{R}^{h \times w \times c}$, and are thus implicitly embedding the vector $v \in \mathbb{R}^c$ into $\mathbb{R}^{h \times w \times c}$ by copying values across the spatial dimensions so that the addition may be performed properly. This operation is known in popular tensor-programming libraries like Pytorch [167] as "broadcasting". This formulation, which we call "non-spatial" noise sampling, is an important subtlety in the construction of SDNNs. See Figure 4.4 for an illustration of this procedure.

To see why we must use non-spatial noise, first make the simplifying assumption that the pooling layer in equation 4.4 were an average pooling layer instead of a max pooling layer, and that the network block F was a simple linear layer with weights ϕ. In this case, if v were not

1-dimensional and each index of $v \in \mathbb{R}^{h \times w \times c}$ were sampled separately, then for $F(g(f))$ we would have $F(g(f)) = \phi(f+v)$, and therefore

$$\begin{aligned} \text{AvgPool}(F(g(f)) &= \phi \frac{1}{hw} \sum_{(i,j)}^{h,w} (v_{i,j} + f_{i,j}) \\ &= \phi \frac{1}{hw} \sum_{(i,j)}^{h,w} v_{i,j} + \phi \frac{1}{hw} \sum_{(i,j)}^{h,w} f_{i,j} \\ &\approx \phi \frac{1}{hw} \sum_{(i,j)}^{h,w} f_{i,j} \end{aligned} \tag{4.5}$$

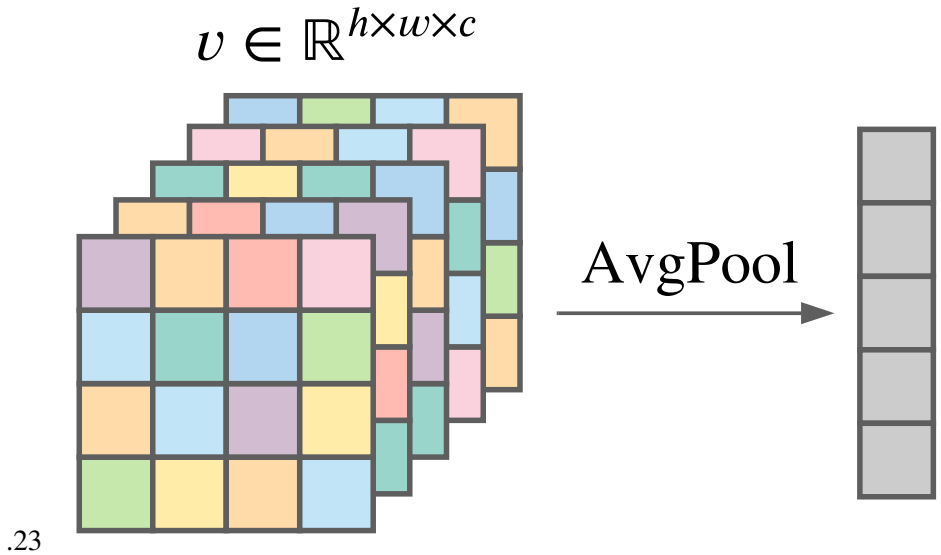
where the last line occurs because $\mathbb{E}(v_{i,j}) = 0$. Thus, in order for the noise to have any effect, it must not be sampled over the spatial component. As we see in Section 4.4.4, non-linearities in the network blocks and max pooling layers mitigate some of these issues, but are not sufficient to completely eliminate the disadvantages of using spatial noise. See Figure 4.4 for an illustration of this point.

4.4 Experiments

In this section, we will first describe the implementation details of SDNN training in Section 4.4.1. We then describe the datasets and metrics used for evaluation in Section 4.4.2 and compare our method against contemporary methods in Section 4.4.3. In Section 4.4.4 we perform ablation experiments to provide a detailed analysis of our method. Finally, in Section 4.4.5 we evaluate our method on the task of few shot learning for human action detection in video.

4.4.1 Implementation Details

One of the most significant advantages of SDNNs is that they can be trained making only minimal modifications to existing networks. As such, our training hyperparameters for image classification mimic those of [68]. More specifically, we perform our 2D image experiments using the WideResNet-28-10 (WRN-28-10) architecture [275]. We perform optimization using Stochastic Gradient Descent (SGD) for 26 epochs, with an initial learning rate of 0.1 and reducing by a factor of 10 after epoch 20 and again after epoch 23. The inverse softmax temperature hyperparameter γ is initialized to 10.



.23

Figure 4.2: Spatial Noise

.23

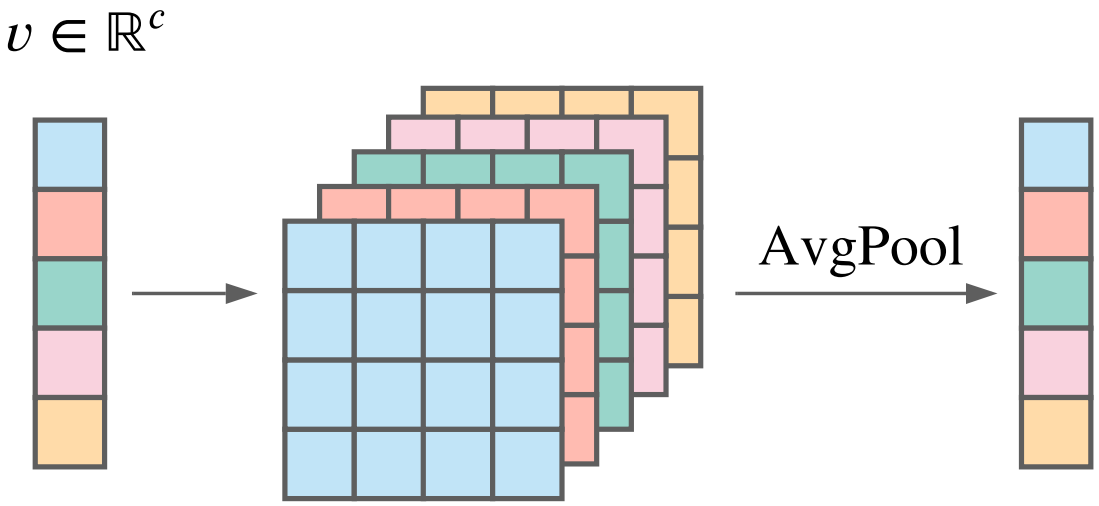


Figure 4.3: Non-Spatial Noise

Figure 4.4: Comparison between spatial and non-spatial noise. (a) Spatial noise perturbs each location in the feature map differently, and therefore averages out to 0 when pooled. (b) Non-spatial noise is constant over each location, and can therefore be pooled without changing its values.

When training the WRN-28-10 backbone, we perform self-denoising on all three residual blocks, adding noise before each block and an auxiliary block afterwards. When pooling, we pool to a spatial size of 2×2 and concatenate the features at all four spatial locations before feeding them into the FC layer. Except where otherwise noted, all SDNN experiments are performed using either additive Gaussian noise with $\sigma = 0.06$ or Dropout with probability 0.1. During inference, no noise is added anywhere in the network, and the logits computed from each auxiliary loss are averaged to make the final predictions.

4.4.2 Image Data and Evaluation Metrics

We perform all our main experiments on three standard datasets used for few shot learning: *mini*ImageNet [247], *tiered*ImageNet [183], and CIFAR-FS [15].

***mini*ImageNet** consists of 100 classes randomly picked from the ImageNet dataset [189] with 600 images of size 84×84 pixels per class. We follow the exact same setup as [68] and others, using 64, 16 and 20 classes as our training, validation and test classes respectively. Also following [68] for consistency, we resample each image to 80×80 before feeding it into the network.

***tiered*ImageNet** consists of 608 classes randomly picked from ImageNet [189]. It consists of 779,165 images in total, all of resolution 84× 84 pixels. We use 351, 97 and 160 classes in our training, validation and test classes respectively, and, again similar to [68], we resample each image during training to 80×80.

CIFAR- FS is a few-shot dataset created by dividing the 100 classes of CIFAR-100 into 64 base classes, 16 validation classes, and 20 novel test classes. There are 60000 images in total in this dataset, each with a resolution of 32× 32 pixels.

Evaluation Metrics: Following standard few shot classification algorithm practices, our experiments are evaluated based on classification accuracy averaged over a large number of episodes. To be more precise, each episode is a N_n-way K-shot problem where K samples are selected at random from N_n randomly-selected novel classes of the parent dataset. The K samples form a support set which the network uses to guide inference. M images are then chosen from the N_n chosen classes to form a test set. The classification accuracy is computed over the $M \times N_n$ images. The final few shot scores are computed as the 95% confidence interval of the accuracy over all the episodes. Except where otherwise noted, we use $N_n = 5$, $M = 15$, and $K = 1$ or $K = 5$ (labelled as "K-Shot" in the appropriate tables).

	Backbone	*mini*ImageNet		CIFAR-FS		*tiered*ImageNet	
		1-Shot	5-Shot	1-Shot	5-Shot	1-Shot	5-Shot
Baseline (Cosine) [69, 174]	WRN-28-10	58.46 ± 0.45%	75.45 ± 0.34%	72.63 ± 0.49%	85.69 ± 0.34%	67.46 ± 0.51%	82.786 ± 0.37%
Shot-Free [182]	ResNet-12	59.04 ± n/a	77.64 ± n/a	69.2 ± n/a	84.7 ± n/a	63.5 ± n/a	82.59 ± n/a
TEWAM [175]	ResNet-12	60.07 ± n/a	75.90 ± n/a	70.4 ± n/a	81.3 ± n/a	-	-
MetaOptNet [129]	ResNet-12	62.64 ± 0.61%	78.63 ± 0.46%	72.6 ± 0.70%	84.3 ± 0.50%	65.99 ± 0.72%	81.56 ± 0.53%
Fine-tuning [45]	WRN-28-10	57.73 ± 0.62%	78.17 ± 0.49%	-	-	66.58 ± 0.70%	85.55 ± 0.48%
Qiao *et al*. [176]	WRN-28-10	59.6 ± 0.41%	73.74 ± 0.19%	-	-	-	-
LEO [190]	WRN-28-10	61.76 ± 0.08%	77.59 ± 0.12%	-	-	66.33 ± 0.05%	81.44 ± 0.09%
DAE [70]	WRN-28-10	61.07 ± 0.15%	76.75 ± 0.11%	-	-	68.18 ± 0.16%	83.09 ± 0.12%
BF3S (CC+Rot) [68]	WRN-28-10	62.93 ± 0.45%	79.87 ± 0.33%	73.62 ± 0.31%	86.05 ± 0.22%	70.53 ± 0.51%	84.98 ± 0.36%
Dropout SDNN (Ours)	WRN-28-10	61.52 ± 0.45%	78.25 ± 0.33%	74.98 ± 0.30%	87.00 ± 0.21%	69.50 ± 0.50%	84.34 ± 0.35%
Gaussian SDNN (Ours)	WRN-28-10	62.12 ± 0.45%	78.97 ± 0.32%	75.31 ± 0.30%	**87.40 ± 0.22%**	69.29 ± 0.50%	84.53 ± 0.35%
Gaussian SDNN + BF3S (Ours)	WRN-28-10	**64.74 ± 0.45%**	**81.47 ± 0.30%**	**75.60 ± 0.30%**	87.30 ± 0.22%	**71.40 ± 0.50%**	**85.90 ± 0.34%**

Table 4.1: Confidence intervals for 5-way classification of several methods, including our own. Inference is performed by sampling images from 5 different novel classes for 2000 iterations on the *mini*ImageNet and *tiered*ImageNet datasets, and 5000 iterations on the CIFAR-FS dataset. "Dropout SDNN" and "Gaussian SDNN" represent our vanilla method as described in Section 4.3.4 using either Dropout noise or additive Gaussian noise, whereas "Gaussian SDNN + BF3S" is a naive combination of our method with the BF3S method [68] intended to show how our method can be effectively combined with others.

4.4.3 Image Evaluation Results

In this section, we compare our proposed SDNN approach against contemporary methods on *mini*ImageNet, CIFAR-FS and *tiered*ImageNet. Table 4.1 lists our performance for each of these datasets, performing inference for 2000, 5000 and 2000 iterations on *mini*ImageNet, CIFAR-FS and *tiered*ImageNet respectively in an episodic fashion by randomly sampling 5 novel classes per episode. "Baseline (Cosine)" is a re-implementation of [69] using a WideResNet-28-10 [275] backbone. This re-implementation is taken directly from the publicly-available code-base for [68] and has not been modified.

We observe that our method by itself with either Dropout or additive Gaussian noise outperforms the baseline on all three datasets by a significant margin, with additive Gaussian noise performing better than Dropout. An unmodified version of our method also performs favorably against many prior methods, even achieving state-of-the-art performance on the "CIFAR-FS" dataset without modification. Notably, our method out-performs the DAE [70] method on all four experiments in which they are comparable. The DAE method itself was trained using an extremely similar set of hyperparameters and architectures as our baseline. This indicates the superiority of our method despite its simplicity - the DAE method required the training of a custom graph-neutral network architecture in a separate stage of pretraining. Our method, by comparison, did not require any additional training stages.

SDNN + BF3S: Table 4.1 also contains an additional experiment, labelled "Gaussian SDNN +

BF3S", which combines a Gaussian SDNN with the rotation prediction task described in [68]. Specifically, BF3S modifies the pretraining procedure by rotating each input image three times in 90-degree increments before feeding all four images into the network. The network is then augmented with an additional residual block whose output is used to train a four-way classifier that determines which of the four possible rotations ($0°, 90°, 180°$, or $270°$) were preformed on the input. Success in this auxiliary task requires the network to develop richer features beyond those needed for classification, and as such improves the quality of extracted features in downstream tasks.

The BF3S architecture is modified into an SDNN in the same way as the vanilla WRN-28-10 architecture - auxiliary losses are added before each block, and Gaussian noise with $\sigma = 0.06$ is added afterwards.

Because we believe this rotation classification task is essentially orthogonal to the SDNN methodology, we are motivated to include this experiment in order to demonstrate that existing few shot methods can be easily and effectively turned into SDNNs. Indeed, we observe from Table 4.1 that "SDNN + BF3S' out-performs all other methods, most importantly the "BF3S" method, supporting our assertion that SDNNs are a broadly applicable technique that will not interfere when added with other, orthogonal techniques.

4.4.4 Ablation Experiments

In this section we analyze the various components of our proposed approach. We report all of our experimental results on the *mini*ImageNet dataset unless otherwise specified. For all experiments in this section, inference is performed for 2000 episodes by sampling images from 5 different novel classes unless otherwise specified.

Aux?	**Noise**	**1-Shot**	**5-Shot**
No	-	$58.43 \pm 0.45\%$	$75.45 \pm 0.34\%$
Yes	-	$60.91 \pm 0.45\%$	$77.84 \pm 0.33\%$
No	Dropout	$58.15 \pm 0.44\%$	$76.10 \pm 0.33\%$
No	Gaussian	$58.73 \pm 0.45\%$	$76.16 \pm 0.34\%$
Yes	Gaussian	$\mathbf{62.12 \pm 0.45\%}$	$\mathbf{78.69 \pm 0.32\%}$

Table 4.2: Confidence intervals for 5-way classification accuracy applying additive Gaussian noise and Dropout noise with and without auxiliary losses on the *mini*ImageNet dataset with a WideResNet backbone. Inference is performed by sampling images from 5 different novel classes for 2000 iterations. "Aux?" indicates if auxiliary losses were used, and "Noise" indicates what type noise was used.

Auxiliary Losses and Noise: SDNNs make essentially two modifications to a standard neural network training procedure: the addition of noise and the training of auxiliary losses. In Table 4.2, we perform the critical experiment of removing each of these components in turn. In doing so, we make two observations. The first observation is that while auxiliary losses are helpful to performance on their own, the addition of noise is critical for achieving the best results, as seen from, for instance, the one-shot performance increasing from 60.91 to 62.12 as Gaussian noise is added.

Our second observation is even more significant: additive Gaussian and Dropout noise have a very small effect on performance if they are not paired with some sort of auxiliary loss - achieving 58.73 one-shot performance as opposed to 58.43. This has significant ramifications - it confirms that the performance gains from using an SDNN exist only because of the specific combination of noise with auxiliary losses. In fact, without the auxiliary losses, the network learns to mostly ignore the noise. We hypothesize that this occurs because adding noise only perturbs the features a very small amount, and if the network is allowed more layers to process the noisy features before classification, the effect of the noise will be mitigated.

Noise Type	Spatial?	**1-Shot**	**5-Shot**
None	-	$60.91 \pm 0.45\%$	$77.84 \pm 0.33\%$
Dropout	Yes	$61.34 \pm 0.45\%$	$78.18 \pm 0.32\%$
Gaussian	Yes	$61.38 \pm 0.45\%$	$78.23 \pm 0.32\%$
Gaussian (AVG)	Yes	$60.68 \pm 0.45\%$	$77.69 \pm 0.33\%$
Dropout	No	$61.52 \pm 0.45\%$	$78.25 \pm 0.33\%$
Gaussian	No	$\mathbf{62.12 \pm 0.45\%}$	$\mathbf{78.69 \pm 0.32\%}$
Gaussian (AVG)	No	$61.47 \pm 0.44\%$	$78.17 \pm 0.32\%$

Table 4.3: Confidence intervals for 5-way classification accuracy sampling either spatial ($v \in \mathbb{R}^{w \times h \times c}$) or non-spatial ($v \in \mathbb{R}^c$) noise, as described in Section 4.3.4. "AVG" stands for the use of average pooling instead of max pooling in our implementation of equation 4.4. Inference is performed by sampling images from 5 different novel classes for 2000 iterations on *mini*ImageNet.

Spatial vs. Non-Spatial: Next, we justify our statements in Section 4.3.4 regarding the need for noise to be non-spatial (*i.e.* to select a noise vector $v \in \mathbb{R}^c$ as opposed to $v \in \mathbb{R}^{w \times h \times c}$). Table 4.3 shows what happens when noise is added with spatial and non-spatial schemes. Our first observation is that, as expected, non-spatial noise out-performs spatial noise. In the case of additive Gaussian noise, we see an improvement from 61.38 to 62.12 in one-shot performance. The improvement for Dropout noise is much smaller to the point of statistical insignificance, improving from 61.43 to 61.52, suggesting that the non-spatial constraint may not be as necessary for Dropout. This could be anticipated, since the arguments for non-spatial noise in Section 4.3.4 do

not necessarily apply to Dropout noise.

In Section 4.3.4 we motivated the need for non-spatial noise by illustrating that spatial noise, when pooled, will have very nearly zero effect. The reason we still see spatial noise effecting performance is that in an SDNN the features are not actually pooled until after they pass through a residual block, which might not act linearly across all of the noise. Additionally, we note that these experiments were performed using max pooling (equation 4.4) as opposed to average pooling, which reacts differently as an operation to 0-mean additive Gaussian noise. We therefore include an additional experiment in Table 4.3, labelled "Gaussian (AVG)" which uses average pooling. As expected, we find no statistically significant difference between spatial Gaussian noise with average pooling and no noise at all. For completeness, we also include results for non-spatial Gaussian noise with average pooling, which confirm that the loss in performance with spatial average pooled noise is not entirely due to the choice of pooling method.

G1	G2	G3	1-Shot	5-Shot
			$60.91 \pm 0.45\%$	$77.84 \pm 0.33\%$
✓			$61.25 \pm 0.45\%$	$78.13 \pm 0.32\%$
	✓		$61.26 \pm 0.45\%$	$78.05 \pm 0.33\%$
		✓	$61.74 \pm 0.45\%$	$78.46 \pm 0.32\%$
✓	✓		$61.26 \pm 0.45\%$	$78.07 \pm 0.32\%$
✓		✓	$61.38 \pm 0.44\%$	$78.34 \pm 0.32\%$
	✓	✓	$61.65 \pm 0.44\%$	$78.57 \pm 0.32\%$
✓	✓	✓	$\mathbf{62.12 \pm 0.45\%}$	$\mathbf{78.69 \pm 0.32\%}$

Table 4.4: Confidence intervals for 5-way classification accuracy applying Gaussian noise at different locations on the *mini*ImageNet dataset with a WideResNet backbone. Inference is performed by sampling images from 5 different novel classes for 2000 iterations. The columns marked "G1" through "G3" indicate whether or not Gaussian noise was added to blocks 1 through 3, respectively.

Which Layers Need Noise? In this section we try to analyze at which point in the network the denoising should be performed. We use the Cosine Classifier with auxiliary losses as our baseline and show all our results in Table 4.4. Here, "G_x" $(x \in \{1,2,3\})$ indicate adding Gaussian noise to the input of block x of the backbone. Initially, we add auxiliary loss at each block without any feature noise. We then add noise to different blocks in turn. We find that performance increases consistently when adding noise to each block, at least when using WideResNet, but we also consistently observe the biggest performance increase when noise is added to the last block. We believe this occurs because the deeper layers of the network possess stronger semantic information, and are therefore most improved by the addition of noise. In our action detection experiments on deeper networks, this motivates us to perform self-denoising only at the later layers

of the network.

σ	1-Shot	5-Shot
0.15	$61.27 \pm 0.44\%$	$78.14 \pm 0.32\%$
0.1	$61.53 \pm 0.45\%$	$78.50 \pm 0.32\%$
0.08	$61.93 \pm 0.45\%$	$78.62 \pm 0.32\%$
0.06	$\mathbf{62.12 \pm 0.45\%}$	$\mathbf{78.69 \pm 0.32\%}$
0.04	$61.82 \pm 0.45\%$	$78.60 \pm 0.32\%$

Table 4.5: Confidence intervals for 5-way classification accuracy applying additive Gaussian noise at different values of σ. Inference is performed by sampling images from 5 different novel classes for 2000 iterations on *mini*ImageNet.

Gaussian Noise Parameters: Table 4.5 shows the results of using different levels of Gaussian noise. We observe that the value of σ has a significant effect on performance. The network achieves the best performance of 62.12% and 78.69% for one-shot and five-shot respectively with a σ value of 0.06. Above 0.06, the performance drops gradually, as the features become too corrupted and the network is unable to denoise them to the required level. Below 0.06, we also start to see performance fall as the noise becomes too small to meaningfully alter the network's features.

Prob	1-Shot	5-Shot
0.2	$60.87 \pm 0.45\%$	$77.43 \pm 0.33\%$
0.15	$61.43 \pm 0.45\%$	$77.96 \pm 0.33\%$
0.1	$\mathbf{61.52 \pm 0.45\%}$	$\mathbf{78.25 \pm 0.33\%}$
0.05	$61.42 \pm 0.45\%$	$78.11 \pm 0.32\%$
0.02	$61.24 \pm 0.45\%$	$78.14 \pm 0.32\%$

Table 4.6: Confidence intervals for 5-way classification accuracy applying Dropout noise at different probabilities of dropping. Inference is performed by sampling images from 5 different novel classes for 2000 iterations on *mini*ImageNet.

Dropout Noise Parameters: Next, we experiment with dropout noise as described in Section 4.3.4. With dropout noise, the model achieves the best performance of 61.52% and 78.25% for one-shot and five-shot respectively with a dropout probability of 0.1. We observe with dropout noise a similar trend to that of additive Gaussian noise. At lower probabilities, there is very little noise to denoise and for higher probabilities, the network is unable to recover the original features effectively.

4.4.5 Few Shot Learning for Action Detection

In addition to our experiments in image classification, we also perform an additional evaluation on the task of human action detection in video. Our evaluation is performed on the "Surprise

	nAUDC (↓)
Cosine Similarity	0.745
Gaussian SDNN	**0.691**

Table 4.7: *nAUDC* scores on the ActEV SDL Surprise Activity evaluation. A lower score is better.

	nAUDC (↓)
Team: UCF	0.633
Team: UMCMU	0.6162
SDNN (Ours)	**0.6151**

Table 4.8: *nAUDC* scores on the ActEV SDL Surprise Activity leaderboard as of the time of this writing. A lower score is better.

Activities" split of the Activities in Extended Video (ActEV) Sequestered Data Leaderboard (SDL) Challenge [158], a public competition in which real-time action detection systems are evaluated on security footage in known or unknown facilities. To submit to the "Surprise Activities" split, teams must produce a system that can perform training with novel classes online on a remote server.

The Surprise Activity evaluation is performed on a separate server with sequestered data, meaning the system is only evaluated on data that the researchers themselves don't have access to. In the case of the "Surprise Activities" split, even validation data is not provided. This makes the task considerably more difficult than the 2D classification performed in the previous sections, since a system must be designed without the ability to determine effectiveness on even small held-out data. This makes the task harder, but also means that the results are not overfit to any sort of validation data.

The difficulty of the "Surprise Activity" task is further increased by computational limitations. In addition to the long training time of the first-stage system, second-stage training is performed on the server itself, which has limited resources. As such, teams are limited to very few submissions a week, usually just one or two. In this context, any sort of hyperparameter tuning to try to overfit to the test data is extremely difficult. Thus, the results we report have only been run once, with no hyperparameter tuning to make the SDNN network perform better. As such, we believe our positive results provide even stronger evidence that SDNNs are an easy and effective way to improve algorithm performance with relatively little work.

The exact novel classes used in evaluation, as well as the number of samples for training and evaluation, are not made public. SDL submissions are evaluated using the *nAUDC* metric, computed by plotting the probability of a missed activity detection at temporal-false-alarm rates

between 0 and 0.2 and calculating the normalized area under the curve.

In addition to the 328 hours of ground-camera data that was annotated for the ActEV SDL, our training makes use of the extensive additional annotations performed by the public in the MEVA annotation repository [39].

For the activity recognition experiments themselves, the conversion of the system into an SDNN uses average pooling down to 1×1 features in the spatial dimensions, while retaining a depth of 8 in the temporal dimension. The features are then concatenated along the temporal dimension and fed into the fully-connected layers, where a cosine classifier makes the final predictions during training. As in the 2D case, during inference the image feature embeddings from the novel class exemplars are used as classification weights.

To perform experiments with our system on the ActEV SDL Surprise Activities split, we have adapted the action recognition pipeline described in [75, 78] to be suitable for the few shot setting. The pipeline itself operates in two stages - a cuboid action proposal stage and an action classification stage - and our modifications are made only to the action classification stage, which is adapted from the I3D architecture [25]. The I3D architecture is itself an extension of the InceptionV1 architecture [225] inflated to use 3D convolutions, and we modify it to be an SDNN by adding Gaussian noise before the "Mixed4e" and "Mixed5b" layers, adding a pooling and cosine classification layer after the "Mixed4f" layer, and replacing the final dot product with a Cosine Classifier as described in Section 4.3.1. As with our previous experiments in 2D, we make no further changes to the training scheme or hyperparameters of the original method.

Table 4.7 shows the results of an SDNN compared to a simple Cosine Classifier baseline. We see that the SDNN performs better than the baseline, with an *nAUDC* score of 0.691 as opposed to 0.745 (a lower score indicates superior performance). This evaluation shows that our method is effective even for video tasks which use larger datasets and deeper architectures.

Table 4.8 shows the current leaderboard scores at the time of this writing. We note in praticular that the UMCMU system uses the same backbone [75, 78] that we do. We see that of the three systems on the leaderboard, our system performs better. We note that our numbers in this table differ from those in Table 4.7. This is because shortly after we performed the experiments in Table 4.7, the data used in evaluation had changed, and our system needed to be resubmitted to be evaluated on the new data. The Cosine Classifier baseline experiment was not repeated in this context, due to the computational limitations of submission outlined above.

Architecture	**1-Shot**	**5-Shot**
3 Blocks, MAX	$75.09 \pm 0.30\%$	$\mathbf{87.32 \pm 0.22\%}$
2 Blocks, AVG	$\mathbf{75.60 \pm 0.30\%}$	$87.30 \pm 0.22\%$

Table 4.9: Comparison of the "Gaussian SDNN + BF3S" experimental results on the CIFAR-FS dataset. The first row of results uses the SDNN hyperparamters described in the chapter (3 blocks with max pooling), and the second row uses those described in this supplementary material (2 blocks and average pooling).

4.5 Conclusion

In this chapter, we have introduced SDNNs, a novel architecture inspired by the ability of DAEs to make features more robust in a few shot learning setting. We have empirically shown the effectiveness of SDNNs on four different datasets across two different tasks, and performed detailed experimental analysis to motivate our construction.

4.6 Supplementary Material

4.6.1 Computational Performance

In terms of parameters, each block requires only a single additional fully connected layer. For the WRN-28-10 architecture with three denoising blocks, this adds exactly 1.312 million parameters (3.6% increase from about 36.5 million without an SDNN) and a training step goes from averaging 0.346 seconds to 0.353 seconds on our system.

4.6.2 Additional Implementation Details

Architecture	**1-Shot**	**5-Shot**
BF3S (CC+Rot) [68]	$70.53 \pm 0.51\%$	$84.98 \pm 0.36\%$
3 Blocks, MAX	$69.28 \pm 0.50\%$	$83.67 \pm 0.35\%$
2 Blocks, AVG	$\mathbf{71.40 \pm 0.50\%}$	$\mathbf{85.90 \pm 0.34\%}$

Table 4.10: Comparison of the "Gaussian SDNN + BF3S" experimental results on the *tiered*ImageNet dataset. The second row of results uses the SDNN hyperparamters described in this chapter (3 blocks with max pooling), and the third row uses those described in this supplementary section (2 blocks and average pooling).

For two of the experiments earlier in the Chapter (the "Gaussian SDNN + BF3S" experiment performed on the CIFAR-FS and *tiered*ImageNet dataset in Table 1) our SDNN implementation only applied noise and auxiliary losses to the last two blocks of the network, as opposed to the three used in every other experiment. Additionally, these two experiments were performed with

average pooling instead of max pooling. We empirically found that this resulted in better performance on those datasets when combined with rotation classification; see Tables 4.9 and 4.10. In particular, the *tiered*ImageNet performance actually dips below the performance of the unmodified BF3S [68] model when all three blocks are used. We hypothesize that this is because the earlier layers of the network are more susceptible to noise when performing the rotation classification task.

All of the networks in this chapter were each trained on a single Nvidia Titan XP GPU.

5. ADAPTING STYLE AND CONTENT FOR ATTENDED TEXT SEQUENCE RECOGNITION

5.1 Introduction

Unsupervised Domain Adaptation is the process of adapting a model trained on data from one fully-labeled domain to work in a different domain where no labels are available. A variety of techniques have been developed towards this end, especially in recent years with the advent of deep learning models that can effectively train on very large labeled datasets.

With respect to the problem of sequential optical character recognition (OCR), where one may possess many images of text but may not have the resources or manpower to label them all, the benefits of domain adaptation are obvious but remain relatively unexplored. To this end, we explore the problem of domain adaptation for OCR in general, and present a novel approach for independently adapting style and content by extending the gradient reversal technique [62]. In order to learn to transcribe unlabeled data, the natural choice is to create a synthetic dataset in the desired language, which can be used to teach a system to recognize the language and structure, *i.e.* content, of the target language. Unsupervised domain adaptation techniques can then be applied to better facilitate the transfer of knowledge between domains.

This approach has some promise, but for best results we would want data that mimics not just the content but also the realistic style of the target data. In other words, we would require a high-quality synthetic dataset that reflects the varying surfaces, illumination conditions, and camera angles of the real data. Building such a dataset would be extremely difficult, presenting a difficult research problem in its own right.

On the other hand, while it certainly is realistic to assume that we do not have labels for OCR data in a language that we want to transcribe, we want to make sure that we use all of our available resources. Many OCR datasets exist for other languages, and while domain adaptation techniques may not always be sufficient for transferring knowledge when the task changes

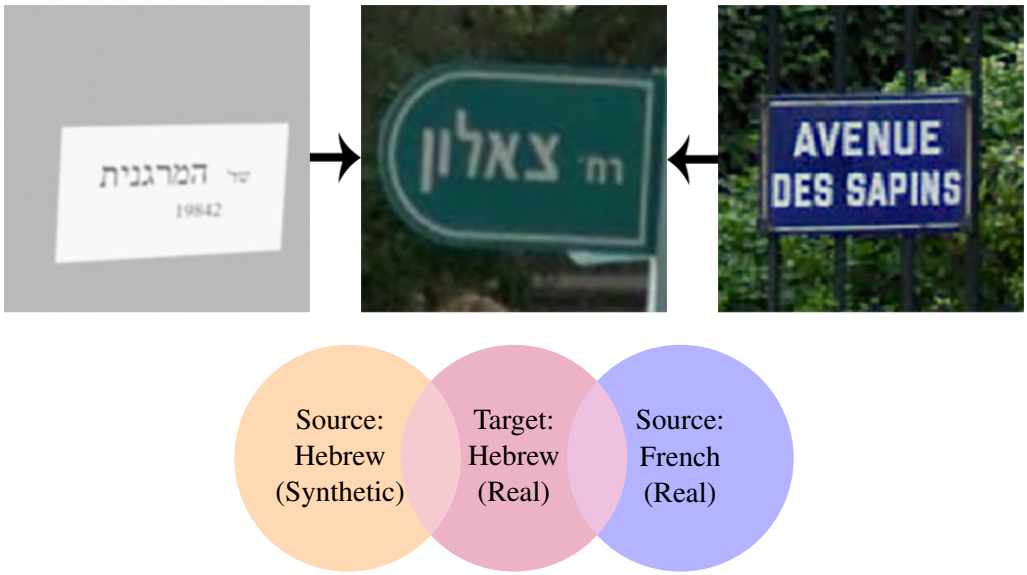


Figure 5.1: We seek to transcribe real images in some language (*e.g.* Hebrew) without access to any labeled training data by using a combination of synthetic data in the same language and labeled real data in a completely different language (*e.g.* French). The synthetic Hebrew data overlaps with the real Hebrew data in content, but not in style, while the real French data overlaps in style but not in content. Thus, the sources are complementary; they each overlap the target significantly, despite having very little overlap with each other.

significantly, it still seems natural that one would be able to use this additional data to improve performance. For example, two datasets containing photographs of images in different languages are still similar in that they are both real images, and therefore share many underlying statistics, specifically those that correspond to the ”realistic” style that they share.

We show in this chapter that the above observation is correct, that it is in fact possible to drastically improve performance in OCR systems using simultaneous multi-task learning in different languages. In fact, our experiments show that including another language during training actually alleviates the need for more realistic synthetic data. It appears that the system learns the ”content” of the first language from the synthetic data, but learns to deal with the realistic ”style” of these images from the second language’s data.

Thus, we show that by properly combining an extremely simple synthetic dataset in one language and a labeled dataset in another language, it is possible to build a system that performs well on a third, completely unlabeled dataset in the first language, as illustrated in Figure 5.1. Remarkably, this is possible even when the two languages being considered use completely different glyphs that are read in different directions. For example, in this chapter we use labeled French street sign images to help read Hebrew street signs, a transfer from a left-to-right language with a

Latin alphabet to a right-to-left language using an Aramaic alphabet.

In order to demonstrate the effectiveness of our technique, we require multiple datasets, not just unlabeled data in our target language but also labeled data in another source languages. To this end, we introduce the Hebrew Street Name Signs (HSNS) and Synthetic Hebrew Street Name Signs (SynHSNS) datasets, which we use in the domain adaptive portion of our system as the target and source domains respectively. To further improve our performance, we also perform multi-task learning on the existing French Street Name Signs (FSNS) dataset [213].

5.2 Related work

5.2.1 Domain Adaptation

Within the field of computer vision, a huge number of unsupervised and semi-supervised domain adaptation techniques have been invented and explored, especially in the context of image classification [86, 193–196, 205], but also in other areas such as semantic segmentation [97, 197, 278], object pose recognition [16] and object detection [32, 103]. In all cases, the goal of these techniques is to match the distributions of some source domain to that of a target domain.

In some cases, this goal is achieved by attempting to explicitly match the moments of the two distributions. For example, Maximum Mean Discrepancy (MMD) [83] is a loss that explicitly minimizes the norm of the difference between two distributions' means, and has been used to good effect in [17,138,238]. Alternatively, work such as [218] and [219] have made significant progress by explicitly aligning the second moments of the source and target domains.

In addition to explicit moment-matching techniques, another technique known as Gradient Reversal (GR) [62, 63] has emerged as a powerful paradigm for deep domain adaptation, serving a fundamental role in many deep domain adaptation systems [17, 32, 97]. GR has even been used effectively for problems completely outside the scope of computer vision, such as machine translation [20]. In the GR setting, a deep network is given an additional discriminator branch that uses deep features to classify samples as originating from either the source or the target domain. The network concurrently trains a feature extractor to fool the discriminator by flipping the sign of the gradient of the discriminator loss with respect to the feature extractor.

An alternative but closely-related deep domain adaptation paradigm uses adversarial learning to minimize domain shift [16, 96, 196, 197, 237]. These techniques are closely related to Gen-

erative Adversarial Networks (GANs) [81] and also use a discriminator to push both feature distributions together.

Domain adaptation has also been used in computer vision for various text-related tasks. For example, domain adaptation techniques have been used to identify fonts in images [257, 258]. Domain adaptation has also been applied to problems involving natural language processing [37, 42, 80], a field related to OCR in its use of language modelling and sequential processing.

5.2.2 Optical Character Recognition

Optical Character Recognition (OCR) is the task of identifying a string of characters in an image. Modern deep-learning-based approaches to OCR generally approach this using a system that first extracts features using a convolutional neural network (CNN) [123] and then extracts the text in a subsequent decoder layer [213, 260]. In particular [260] uses the first several layers of the InceptionV3 architecture [227] to extract features which are then fed through an LSTM with a special form of attention to produce a transcription.

Domain adaptation has also been exploited in the field of sequential OCR. When the target domain consists of a large corpus such as books, the style and linguistic consistency can be leveraged to fine tune a Gaussian based model under maximum likelihood or MAP criteria using Expectation-Maximization [198, 242]. This is also analogous to speaker adaptation using a speaker-independent HMM model [67]. In more recent works [248, 279], style and content separation have been effective in adapting digit recognition from MNIST to SVHN.

Finally, we note that while many of the image classification tasks discussed above demonstrate their effectiveness on the MNIST [128] and SVHN [152] datasets, it is important to emphasize that while this task certainly falls into the category of OCR, it is much simpler than the general task of sequential OCR. MNIST and SVHN both present a single digit at a time for classification, whereas we are concerned with images in which a variable-length series of characters must be identified and classified in the correct order. For this reason, it is not trivial to directly apply the domain adaptation techniques discussed above to the task of sequential OCR. For example, the system on which we perform domain adaptation contains additional Recurrent Neural Network (RNN) and attention components that are not present in any of the non-sequential OCR architectures discussed above.

5.3 Method

We seek to design a system that can transcribe real images in a language for which no real labeled data exists. To do this, we approach the problem from two different sides simultaneously, by using two different datasets to address the "style" and "content" of the images in the dataset. Specifically, we use unsupervised domain adaptation to transfer knowledge about content (the language itself) learned from synthetic data while at the same time using a simple multi-task learning scheme to make the system robust to the style of real images.

We differentiate between three sets of images available to us at training time. The first set of source images $\mathbf{X}_{S_C} = \{x_1^{S_C}, x_2^{S_C}, \ldots, x_{N_C}^{S_C}\}$ is the "content" dataset, representing synthetic images of text in some language while $\mathbf{Y}_{S_C} = \{\mathbf{y}_1^{S_C}, \mathbf{y}_2^{S_C}, \ldots, \mathbf{y}_{N_C}^{S_C}\}$ represents the associated labels. Here, each $\mathbf{y}_i^{S_C}$ is a sequence of integers in some alphabet $\mathcal{A}_C$. For concreteness we will refer to $\mathcal{A}_C$ as the Hebrew alphabet, since that is what we will use in our experiments, but our method could hypothetically work for any language. Similarly, the second set of source images $\mathbf{X}_{S_S}$ and labels $\mathbf{Y}_{S_S}$ represent the style dataset; images and labels for real images of text in some other language using a different alphabet which we denote $\mathcal{A}_S$. Again, for concreteness, we'll refer $\mathcal{A}_S$ as French, but any language, even one using different glyphs, is applicable to our method. We will be using $\mathbf{X}_{S_C}$ for domain adaptation and $\mathbf{X}_{S_S}$ for multi-task training.

The third domain, the target domain, contains only images $\mathbf{X}_T = \{x_1^T, x_2^T, \ldots, x_{N_T}^T\}$. The images in $\mathbf{X}_T$ are photographs of text in the same language as those in $\mathbf{X}_{S_C}$, *i.e.* text that uses $\mathcal{A}_C$ as its alphabet. A key feature in this setup is the assumption that the domain shift between T and each of S_C and S_S is not prohibitively large. That said, S_C and S_S have very little in common with each other, since they do not overlap in either style or content.

5.3.1 Base Architecture

We perform our experiments by extending the architecture introduced in [260]. At a high level, this architecture consists of three components: a CNN feature extractor G_f, a Recurrent Neural Network (RNN) G_r that recurrently outputs characters by processing the extracted visual features, and a spatial attention mechanism that guides the RNN component to look at salient features, which for the purposes of our discussion we fold into the RNN network G_r.

Following [260], we use the first several layers of the Inception V3 CNN architecture [227] for our visual feature extractor G_f; everything up to the "Mixed5D" module. This mapping is

fully convolutional, and we denote its output features as $f = G_f(x, \theta_f)$, where θ_f represents the vector of parameters for G_f. We denote the output of the RNN and spatial attention portions of the network in [260] as $\hat{\mathbf{y}} = G_r(\mathbf{f}, \theta_r) = (\hat{y}_1, \ldots, \hat{y}_n)$. We illustrate this architecture in Figure 5.2.

More precisely, to compute G_r at a specific step t, we first compute a spatial attention mask α_t over visual features f, after which we compute a context vector

$$u_{t,c} = \sum_{i,j} \alpha_{t,i,j} f_{i,j,c} \tag{5.1}$$

which is fed into the RNN as

$$\begin{aligned} \hat{x}_t &= W_c c_{t-1} + W_{u_1} u_{t-1} \\ (o_t, s_t) &= \text{RNNStep}(\hat{x}_t, s_{t-1}) \end{aligned} \tag{5.2}$$

where s_t and o_t denote the internal state and output of the RNN at time t, and c_{t-1} is a one-hot encoding of the previous letter, either from the ground truth during training or as predicted during inference.

Finally, we calculate the distribution over letters as

$$\hat{o}_t = \text{softmax}(W_o o_t + W_{u_2} u_t) \tag{5.3}$$

and assign

$$\hat{y}_t = \arg\max_c \hat{o}_t(c). \tag{5.4}$$

5.3.2 Style Adaptation

To learn the "style" of real imagery, we utilize a simple multi-task learning procedure, training a single network which learns the tasks of transcribing synthetic Hebrew and real French simultaneously. The end result is a system that is is significantly better at transcribing real Hebrew images by implicitly exploiting the style overlap between the real French and Hebrew data. Specifically, we train a single G_f to extract features from both synthetic Hebrew street signs $x^{S_C} \in \mathbf{X}_{S_C}$ and real French street signs $x^{S_S} \in \mathbf{X}_{S_S}$, as in Figure 5.3 (left). The output features f are then fed through two different Attention/RNN components, $G_r^{S_C}$ and $G_r^{S_S}$, to produce two sets of outputs $\hat{\mathbf{y}}^{S_C} = G_r^{S_C}(\mathbf{f}, \theta_r^{S_C}) = (\hat{y}_1^{S_C}, \ldots, \hat{y}_n^{S_C})$ and $\hat{\mathbf{y}}^{S_S} = G_r^{S_S}(\mathbf{f}, \theta_r^{S_S}) = (\hat{y}_1^{S_S}, \ldots, \hat{y}_m^{S_S})$, respectively. We then

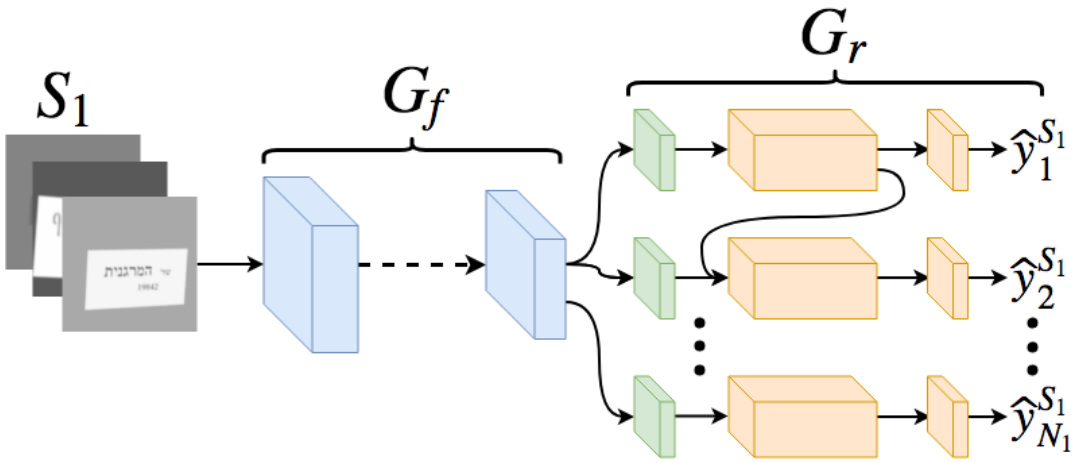


Figure 5.2: The baseline architecture, as described by Wojna *et al*. [260]. A feature extractor G_f is used to extract features, in this case from S_C. These features are then fed into a decoder G_r, which includes a spatial attention component.

train both sets of data separately according to their respective cross-entropy classification losses:

$$\begin{aligned}
&\mathcal{L}_{S_C}(\mathbf{X}_{S_C}, \mathbf{Y}_{S_C}) = \\
&-\mathbb{E}_{(x^{S_C}, \mathbf{y}^{S_C}) \sim (\mathbf{X}_{S_C}, \mathbf{Y}_{S_C})} \left[\sum_{i=1}^{N_C} \sum_{j \in A_C} 1_{[j=y_i^{S_C}]} \log \hat{y}_i^{S_C} \right] \\
&\mathcal{L}_{S_S}(\mathbf{X}_{S_S}, \mathbf{Y}_{S_S}) = \\
&-\mathbb{E}_{(x^{S_S}, \mathbf{y}^{S_S}) \sim (\mathbf{X}_{S_S}, \mathbf{Y}_{S_S})} \left[\sum_{i=1}^{N_S} \sum_{j \in A_S} 1_{[j=y_i^{S_S}]} \log \hat{y}_i^{S_S} \right]
\end{aligned} \tag{5.5}$$

In practice, we actually extend these losses to be autoregressive, as described in [224], where we pass in the ground truth labels as history when we perform training.

In order to learn to label the French images in X_{S_S}, the system must learn to ignore the realistic style of the French images and focus on the content. The realistic style of the French images overlaps heavily with the style of the images in X_T, and, as a result, we hypothesize that the system also learns to ignore the realistic style of the target images, even as it learns the content from the synthetic images in X_{S_C}.

5.3.3 Content Adaptation

While the system described in Section 5.3.2 still learns the content of the Hebrew language from the synthetic data, it does nothing to specifically enforce the similarities between S_C and T; in

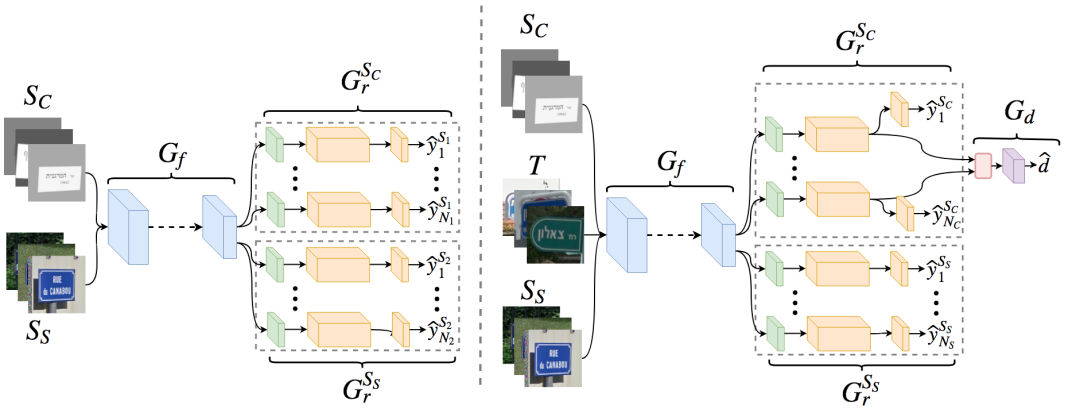


Figure 5.3: (Left) The configuration of the network for multi-task training. The same feature extractor G_f is used to extract features from both S_C and S_S. These features are then fed into separate decoders $G_r^{S_C}$ and $G_r^{S_S}$. (Right) We perform domain adaptation on $G_r^{S_C}$ by aggregating the intermediate RNN values s_t and using gradient reversal on a domain classifier that selects between S_C and T. We do not perform any adaptation with respect to S_S beyond what the network learns through multi-task training.

fact, it does not use T at all during training. To address this, we use the techniques of unsupervised domain adaptation to explicitly adapt the synthetic Hebrew data to the real.

Gradient Reversal

We seek to improve our performance in the target domain in part by directly training our system to be robust to the domain shift between the synthetic and real Hebrew data. More specifically, we wish to reduce the divergence between the features of the source and target distributions. To this end Ben-David *et al.* [14] show that the $\mathcal{H}$-divergence between a source domain S and a target domain T can be computed as

$$\hat{d}_{\mathcal{H}}(S,T) = 2\left(1 - \min_{h\in\mathcal{H}}\left[\hat{\varepsilon}_S(h) + \hat{\varepsilon}_T(h)\right]\right) \tag{5.6}$$

where $\mathcal{H}$ is the set of binary classifiers that assign 1 to samples in the source domain and 0 to samples in the target, and $\hat{\varepsilon}_S(h)$ and $\hat{\varepsilon}_T(h)$ are the empirical classification error on the source and target domains, respectively. It therefore follows that we can minimize the distance $\hat{d}_{\mathcal{H}}(S,T)$ between domains by maximizing the error of all classifiers that distinguish between the domains.

Ganin *et al.* [63] achieve this goal with a technique known as gradient reversal (GR). Here, training is framed as a saddle point problem, where the system is broken into three parts. Features f are extracted using a feature extractor $f = G_f(x,\theta_f)$, and then fed into a task-specific classification branch $G_y(f,\theta_y)$ and a domain-discriminator branch $G_d(f,\theta_d)$. G_d attempts to classify the

domain of any given sample as either source or target using the loss

$$\mathcal{L}_d = -\left(\sum_{x \in \mathbf{X}_S} \log G_d(x) + \sum_{x \in \mathbf{X}_T} \log(1 - G_d(x)) \right). \tag{5.7}$$

In essence G_d is a classifier belonging to the hypothesis class $\mathcal{H}$ described above.

We can then define an energy function

$$E(\theta_f, \theta_y, \theta_d) = \mathcal{L}_y(\mathbf{X}_S, \mathbf{Y}_S) - \lambda \mathcal{L}_d(\mathbf{X}, \mathbf{D}) \tag{5.8}$$

where $\mathbf{D} = (d_1, \ldots, d_n) = (1_{x_1 \in X_S}, \ldots, 1_{x_n \in X_S})$ and λ is a hyper-parameter to control the trade-off between the two losses. $\hat{d}_{\mathcal{H}}(S,T)$ is then minimized at the saddle point

$$\begin{aligned} (\hat{\theta}_f, \hat{\theta}_y) &= \arg\min_{\theta_f, \theta_y} E(\theta_f, \theta_y, \hat{\theta}_d) \\ \hat{\theta}_d &= \arg\max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d). \end{aligned} \tag{5.9}$$

Gradient reversal presents a simple way to optimize this saddle point problem using stochastic gradient descent. To do this, a special Gradient Reversal Layer (GRL) is added between G_f and G_d. On the forward pass of training, the GRL acts as an identity map, but on the reverse pass the GRL multiplies its gradient by -1. This effectively replaces $\frac{\partial \mathcal{L}_d}{\partial \theta_f}$ with $-\frac{\partial \mathcal{L}_d}{\partial \theta_f}$, which as [63] show is sufficient to achieve a saddle point of (5.8).

Adapting The Decoder

A naive way to to apply the techniques of gradient reversal to the architecture described in Section 5.3.1 would be to treat G_r^{Sc} the same we treated G_y in Section 5.3.3: as a simple classifier that acts on the features extracted by G_f. Informally, the intuition is that we would be adapting the visual features to become robust to the change in style between real and synthetic.

However, we explored multiple architectures using this approach, and we experimentally found that the main benefit of domain adaptation is in its ability to improve understanding of the content, and less so in its ability to build robustness to the style. Under this hypothesis, it makes more sense to perform domain adaptation in the RNN portion of the network, where the language

structure is processed.

Thus, we introduce a method that directly adapts the RNN components of the system, which we illustrate in Figure 5.3. Specifically, we leave G_r unchanged, but for each RNN step t we introduce a new value

$$v = \text{GRL}(\max_t s_t) \tag{5.10}$$

We experimentally found that it was essential to aggregate the RNN output using maximization, as averaging or using a softmax attention-based aggregation did not result in a system that performed better than the baseline.

We then use a domain-discriminator G_d on the output, which we calculate as

$$\begin{aligned} w &= W_{d_2}\text{relu}(W_{d_1}v + b_{d_1}) + b_{d_2} \\ \hat{d} &= \text{softmax}(W_{d_3}w). \end{aligned} \tag{5.11}$$

We can then define $\mathcal{L}_d$ as it was defined in Equation 5.7, and our final loss becomes

$$E(\theta_f, \theta_r, \theta_d) = \mathcal{L}_{S_C}(\mathbf{X}_{S_C}, \mathbf{Y}_{S_C}) - \lambda \mathcal{L}_d(\mathbf{X}, \mathbf{D}). \tag{5.12}$$

This modification is essential for success once data from S_S is added to the system, since it performs adaptation on a portion of the network that is not directly enhanced by the additional data.

When combined with multi-task learning, our final energy function becomes

$$\begin{aligned} E(\theta_f, \theta_r^{S_S}, \theta_r^{S_C}, \theta_d) = \\ \mathcal{L}_{S_S}(\mathbf{X}_{S_S}, \mathbf{Y}_{S_S}) + \mathcal{L}_{S_C}(\mathbf{X}_{S_C}, \mathbf{Y}_{S_C}) - \lambda \mathcal{L}_d(\mathbf{X}, \mathbf{D}). \end{aligned} \tag{5.13}$$

During each step of training, we optimize all three components of this loss in a single batch.

Trained on:	SynHSNS	FSNS	HSNS	SynHSNS Acc.	FSNS Acc.	**HSNS Acc.**
FSNS Baseline		✓		*N/A*	64.34%	*N/A*
Baseline	✓			94.68%	*N/A*	18.49%
Fine-Tuning	✓	✓		89.81%	64.34%	29.56%
Multi-Task Training (MT)	✓	✓		94.68%	64.26%	36.54%
Domain Adaptation (DA)	✓		✓	93.57%	*N/A*	38.64%
DA + MT	✓	✓	✓	91.47%	63.39%	**50.16%**

Table 5.1: Full-sequence accuracy on the test data of each dataset for the various systems we discuss in this paper. Check marks indicate which datasets were available during training for each experiment. The most important accuracy results are those of HSNS, the target dataset for our system. We also report performance on the SynHSNS and FSNS datasets, though we note that optimizing performance on these datasets is not the goal of our system. Still, results indicate our system does not completely destroy performance on these secondary tasks, a fact which may be useful in building a more general system.

The complete architecture with all components and unsupervised domain adaptation applied to the decoder is illustrated in Figure 5.3. When training, we use $\lambda = 0.5$, a value which we determined experimentally.

5.4 Experiments

The setup we suggest is both unique and highly specific, so in order to properly evaluate it we introduce two new datasets containing real and synthetic images of Hebrew street name signs. Used in conjunction with the existing FSNS street name dataset, we illustrate the effectiveness of both our domain adaptation technique and a simple multi-task learning approach. We then demonstrate that using both techniques together preforms better than using only a single technique, and provide a detailed empirical analysis of our results.

Following [260], the metric which we report for all of the following techniques is full sequence accuracy, wherein a sample is considered correctly classified only if every character is predicted correctly.

Unfortunately, in the absence of an alternate yet reliable means of performing hyperparameter optimization, we follow [17] and perform out experiments directly on a small set of validation data. We understand that this is not optimal, as the argument can be made that any labeled data available at training time should be used during training, and we therefore hope that in the future the research community will present an alternative means for validation in the unsupervised domain adaptation scenario. For now, we leave the development of such a metric to future work.

Figure 5.4: Sample images from the HSNS, SynHSNS, and FSNS datasets, displayed on the top, middle, and bottom rows respectively.

5.4.1 Datasets

Hebrew Street Name Signs

For our target dataset, we collected approximately 92,000 cropped images of Hebrew street name signs from Israel. We divided these into three different splits of 89,936 test images, 899 validation images and 903 test images, of which only the validation and test images have labels. When splitting the dataset we maintained a geographic distance of at least 100 meters between the location of any training/validation and test images, to ensure that the system does not have exposure to any test signs while training or performing validation. All of these images are 150×150 resolution.

Many Hebrew street signs include certain prefixes that translate to words such as ”street,” ”road,” ”avenue,” *etc*. More often than not, these words are written in a much smaller font than the rest of the sign, making them illegible at 150×150 resolution. Since many Israeli map services

don't include these prefixes, we also decided to exclude them from the transcriptions.

We will be releasing this data as the Hebrew Street Sign Names (HSNS) dataset. Samples from this dataset are shown in Figure 5.4. Although the images are collected in full RGB color and will be released as such, in all of the tests that follow we convert each image into greyscale so as to maintain consistency with our synthetic images, which we describe below.

Synthetic Hebrew Street Name Signs

We elected to use a relatively simple scheme for generating synthetic data. This decision is motivated both by the difficulty of generating more sophisticated natural-looking synthetic data, and by the observation that the synthetic data need only contain the same content as the target data to be useful, as we can use other methods to address the style.

Our synthetic images therefore consist of only straightforward text rendering, a box placed behind the text, a perspective transform, and some slight blur. When rendering the text, we randomly select from one of nineteen different Hebrew fonts. In some cases, we randomly add English text or numbers below or above the Hebrew, which we don't include in the ground truth transcriptions. The size and placement of the text, the parameters of perspective transform, and the amount of blur are all selected randomly. The actual text itself is selected from a list of real Israeli street names. To better match the text distribution of HSNS, we also randomly add small font prefixes which translate to the Hebrew words for "street", "road", "avenue", *etc*. We found that these prefixes were essential for performance, since they are often included in real images but are often too small to read, and including them in the synthetic data signals to the system that they do not need to be transcribed. We generate all images at 150×150 resolution.

In order to simplify the text generation process further, all synthetic images are generated in greyscale. This greatly simplifies the generation process by making it much easier to produce images in a realistic color range. The exact colors for each image are selected randomly, though we do enforce a minimum amount of contrast between the text and the box behind it. We used a solid color for the background, because preliminary tests using more complicated backgrounds (*e.g*. Gaussian noise) did not yield any differences in performance.

We generate roughly 430,000 synthetic images for training, and another 10,000 each for evaluation and testing. For sample images, see Figure 5.4. We release this data along with HSNS as the Synthetic Hebrew Street Name Signs dataset (SynHSNS).

French Street Name Signs

In addition to the two Hebrew-language datasets above, we also use the existing French Street name Signs (FSNS) dataset [213] for our multi-task learning. FSNS contains roughly 1 million training, 20,000 evaluation, and 16,000 test samples of French street name signs, each containing between one and four views of the same sign at 150×150 resolution. For consistency with HSNS and SynHSNS, we only use one of these views during training, taking whichever view is listed first. Similarly, we maintain consistency with SynHSNS by converting each image into greyscale. Sample images from the original FSNS dataset can be seen in Figure 5.4.

5.4.2 Implementation Details

With the exception of the fine-tuning experiment described in Section 5.4.3, all of the training is performed with a learning rate of 0.0047 using Stochastic Gradient Descent with a momentum value of 0.75. We train for 800,000 steps with a batch size of 15 for each domain actually used during training. When domain adaptive components are present, we turn them on starting at 20,000 steps and compute the loss in Equations 5.12 and 5.13 with the value $\lambda = 0.5$. All input images are 150×150 resolution, consistent with the resolution of the data in all three datasets.

5.4.3 Domain Adaptation and Joint Training

Baselines

In order to show the efficacy of our system, we need to demonstrate that our methods perform better than a naive approach. We therefore define our baseline to be the test performance of HSNS on a system trained exclusively on the SynHSNS data. Results of this experiment are reported in Table 5.1 as "Baseline".

Table 5.1 also includes for reference the performance of a system trained exclusively on the version of FSNS used in all experiments, listed as "FSNS Baseline". As described above, our usage of FSNS differs from the standard usage in that we have only used one of the up to four possible views for each sign, and we have removed all color from the image. Therefore, while the number we report here for FSNS is lower than the number [260] reported for essentially the same system, it is important to note that the two experiments were not performed on precisely the same data. We would also like to emphasize that our goal is not to optimize performance on FSNS, but rather on HSNS, and therefore these numbers are included only for reference.

Multi-Task Learning Baselines

We report results on the multi-task learning scheme described in Section 5.3.2, where we train on both the SynHSNS and the FSNS datasets simultaneously. We report this in Table 5.1 as "Multi-Task Training (MT)". As with the baselines described above, HSNS data is not seen during training, yet we still achieve 36.54% accuracy on the HSNS test set. Thus, simply by learning to parse real French images, the model achieves an 18 point improvement when parsing real Hebrew images, supporting our hypothesis that the system is better able to understand the realistic style of the Hebrew data just from seeing the real French data.

In addition to the joint training scheme described above, we also evaluate our method on a simple fine-tuning scheme, listed in Table 5.1 as "Fine-Tuning". In this scheme, we first train the entire system on FSNS alone for 800,000 steps. Then we replace $G_r^{S_S}$ with $G_r^{S_C}$ and train the network for an additional 66,000 steps at a reduced learning rate of 0.002 (additional steps of training did not increase the performance on HSNS). Performance results of both methods are reported in Table 5.1. We see that multi-task learning is superior to fine-tuning, probably because the additional training phase reduces some of the benefits gained by the French data in the first phase.

Domain Adaptation

To evaluate the effectiveness of gradient reversal, we again perform two experiments, both based on the RNN-centric domain adaptation described in Section 5.3.3.

The first of these experiments, denoted in Table 5.1 as "Domain Adaptation (DA)", performs domain adaptation on the RNN portion of the network $G_r^{S_C}$, explicitly optimizing the loss in Equation 5.12 using only HSNS and SynHSNS as input, *i.e.* the architecture shown in Figure 5.3 (right) with $G_r^{S_S}$ and the FSNS input removed.

Our second experiment, denoted "DA+MT", uses all three datasets as input, and is a test of our full system as illustrated in Figure 5.3 (right). This experiment stands out as the only one to make use of all three datasets available to us.

From these experiments, we see that domain adaptation alone, just between HSNS and SynHSNS, is enough to yield a performance increase from 18.49% to 38.64%. What's perhaps more interesting is that combining this with multi-task learning takes the performance to 50.16%.

In particular, the marginal increase from "DA" to "DA+MT" (about 11 points) is not trivial. Similarly, the jump from from "MT" to "DA+MT" (about 14 points) is also quite substantial.

We believe that this supports our hypothesis that domain adaptation targets the content while multi-task learning targets the style, because it suggests that the improvements each technique provides are mostly disjoint, *i.e.* domain adaptation is helping for a different reason than mutli-task learning. If these techniques weren't complementary, and both "DA" and "MT" improved performance by addressing the same features of the target, then we might expect to see a smaller marginal improvement when we used both of them together, since it would suggest that there is very much "overlap" between the techniques.

5.4.4 Visualizing Network Output

We use two different methods of visualization to understand the behavior of the model, following [260]. For the first, we output the attention maps used by the network when decoding the image features, upsampled to the size of the image with nearest neighbor interpolation. Our second method of visualization is based on the Saliency maps of [211]. For a given predicted character at location t, we compute the gradient of its logit $\hat{o}_t$ with respect to the input image x. The saliency of pixel (i, j) is thus $v_{i,j} = \| \frac{\partial \hat{o}_t}{\partial x_{i,j}} \|$.

We display the results of this visualization for the SynHSNS and HSNS datasets on the "DA+MT" experiment. For SynHSNS, we find the expected behavior for a network trained on SynHSNS data: both the attention and saliency maps focus tightly around the letters being described. Importantly, however we also see the same behavior in the case of HSNS, indicating that the network has learned to properly control where it looks even in the case of the real Hebrew data.

5.5 Conclusion

In this chapter, we have explored different approaches to teaching a system to perform sequential OCR on photos of street name signs in a language for which there exists no labeled data. To do this, we introduce two new datasets: the SynHSNS dataset of synthetic Herbrew street sign names, and the HSNS dataset of real unlabeled Hebrew street sign names. Ultimately, we demonstrate that our approach, which leverages existing data in other languages and easily-generated synthetic data in the same language, can be used to greatly improve performance in the

target domain by transferring information about both style and content.

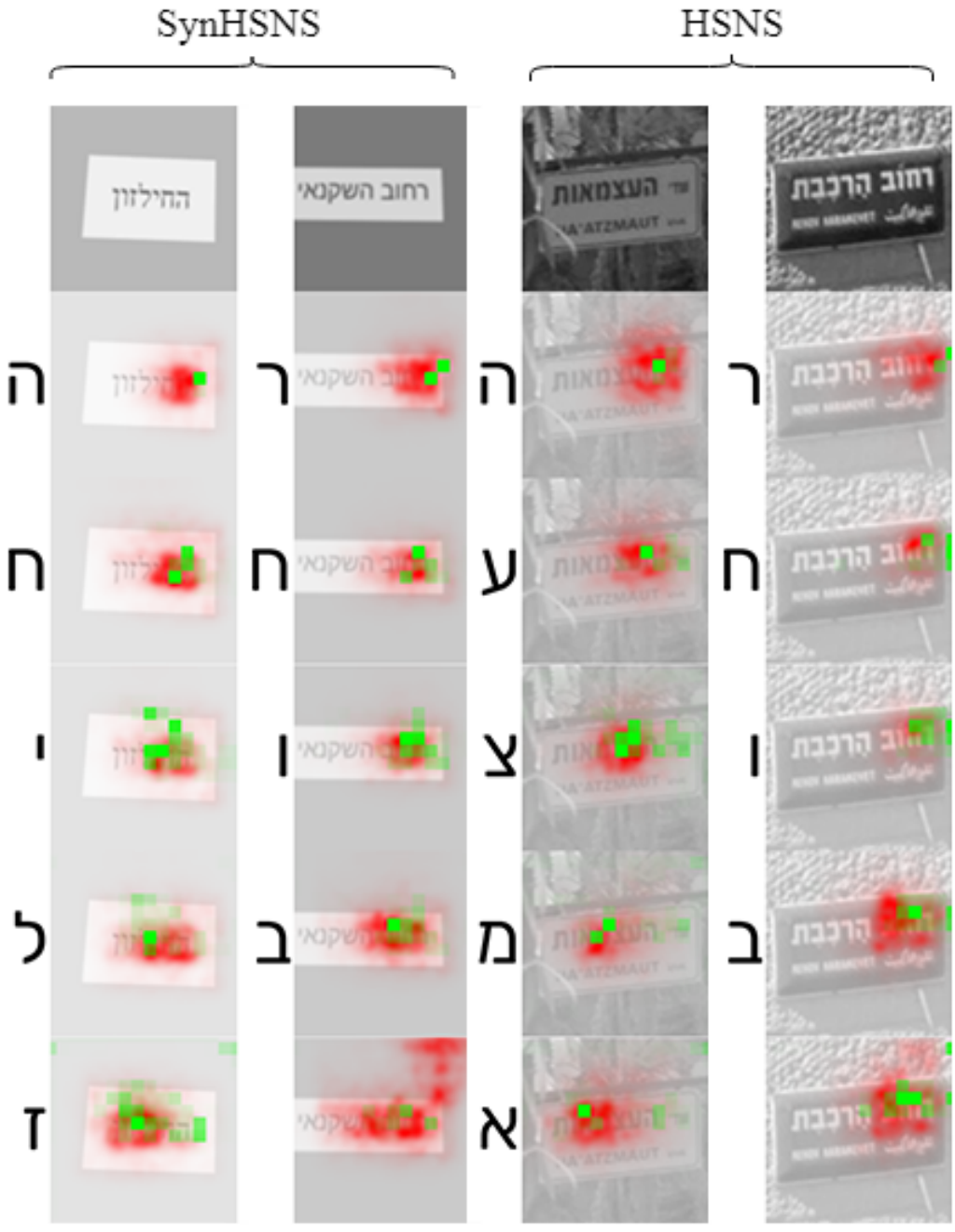


Figure 5.5: Visualization of attention maps (green) and saliency maps (red) for SynHSNS (left) and HSNS (right) under the "DA+MT" training protocol. Both maps are of comparable quality.

6. FINDING FACIAL FORGERY ARTIFACTS WITH PARTS-BASED DETECTORS

6.1 Introduction

The past few years have seen a sharp rise in the availability of technology for creating and distributing digital forgeries, specifically those where one individual's identity is replaced with that of a another. These forgeries, known as "Deepfakes," represent an important and growing threat to information integrity on the web. A variety of techniques have been proposed to combat the risks inherent in the widespread availability of this technology, many of which are known to perform very well on the datasets on which they are trained, but perform worse when they are applied to different datasets, particularly those which generate fakes using different algorithms.

For this reason, it is vital that video forgery detection systems be able to generalize effectively to novel datasets. Improving performance in this regard requires a strong understanding of both the detectors used and the data itself. To this end, a lot of approaches have been explored [28, 131, 133].

In this chapter, we seek to add one more approach to this list, analyzing the problem of digital forgery detection from another angle. Rather than focus on a single architecture for forgery detection, we break the problem down into smaller pieces, dividing the face into several distinct regions of interest and training separate classifiers for each one. Specifically, we divide the face into four main regions - the nose, mouth, eyes and chin - and learn separate detectors for the individual artifacts left behind in each region.

Despite the fact that these classifiers are designed to restrict their attention only to their specific assigned regions, some of them still generalize quite well. This suggests that the regions we are training on may contain powerful clues for forgery detection, powerful enough to let these restricted classifiers out-perform more conventional systems.

These parts-based detectors are also fertile ground for deeper empirical analysis. They are explainable because they make decisions based on limited, easily understood criteria, and we can

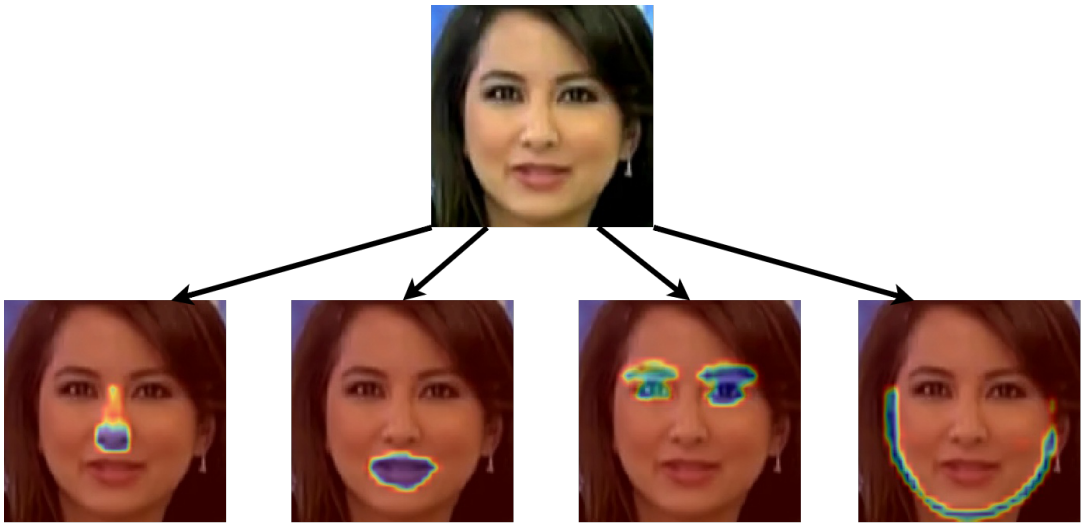


Figure 6.1: In order to detect manipulated images in an explainable way, we divide the face into four regions - nose, mouth, eyes and chin/jawline - and train separate classifiers for each region. The resultant classifiers generalize well and provide valuable insights into the image manipulation process.

use their strengths or weaknesses to make inferences about the underlying data they are trained and evaluated on. They thus allow us to gain further insights into the comparative structure of different forgery algorithms.

Using FaceForensics++ [188] as our main dataset, our empirical analysis is broken down as follows: first, we train and evaluate the generalizability of parts-based detectors for each of the four regions of the face that we designate - nose, mouth, eyes and chin/jawline. We then perform the same experiments with a combined parts-detector, which uses multiple branches within the same network to compute parts-based detection separately before recombining. Having trained these parts detectors, we perform comparative analysis of their strengths and weaknesses, looking at both the performance of our detectors and the statistics of altered regions in manipulated images in an effort to develop new insights into the problem of forgery detection. We finish with cross-dataset analysis, comparing generalization performance not just between different algorithms of the FaceForensics++ dataset but also across datasets, preforming transfer experiments on the Celeb-DF [132] and Facebook Deepfake Detection Challenge [47] datasets.

6.2 Related Works

6.2.1 Deepfake Algorithms

Computers have long been used to generate forged imagery, but until recently most quality digital forgeries would need to be carefully designed by hand. However, since the invention and widespread use of Generative Adversarial Networks (GANs) [82], it has become considerably easier to use computers to convincingly modify images. In this chapter, we are particularly concerned with digital forgery techniques that use deep learning or other modern techniques to transfer the face of one individual to another.

These forgeries, colloquially known as "Deepfakes," can be generated in a variety of ways. Some of these methods predate the invention of GANs, such as [100], which performs swaps by identifying landmarks on faces, and then warping one individual's face onto the other. Still, many Deepfake methods have GANs at their core. FSGAN [157] uses a GAN to reenact one person's pose with another's face, while the NTH model [276] produces Deepfakes by pre-training meta-variables and swapping faces in a few-shot setting. Similarly, techniques like StyleGAN [116] allow Deepfake generation by projecting one image into the latent space of another.

These techniques are especially easy to use because many of them have been made available as software packages. Software such as DeepFaceLab [170] and FaceSwap [1] are easy to download and use, making Deepfakes available to many people outside the research community.

6.2.2 Forgery/Deepfake Detection

As techniques for generating forged images have improved, so too have techniques for identifying them. Some techniques identified forgery by looking at metadata [102] or other low level artifacts. For instance, techniques such as [131, 133, 172, 173, 254] try to identify forgeries by looking for specific artifacts, e.g. by searching for repeated regions, or stitching or warping artifacts in transferring one face to another body. Patch-based techniques are also popular [179, 281]; for instance, [28] uses patch-based techniques to analyze and improve the generalizability of video forgery detection systems.

Many techniques take advantage of the strength of deep networks directly, by training image classifiers with various architectures to identify forgeries, either through RGB [4, 11, 40, 146] or

[1] https://github.com/deepfakes/faceswap

Figure 6.2: Mask generation pipeline. First parts are detected using dlib, then the masks are created from the convex hulls of the detected regions. Finally, masks are post-processed with morphological dilations and gaussian blur.

optical flow [7]. Our work continues in the same vein as many of these other works, taking key intuitions from other methods. In particular we take inspiration from the patch-based approach of [28], extending their patch-based approach to train specific parts-based detectors.

There have also been many new datasets introduced to evaluate forgery detection systems. These range from the smaller but varied Deepfake TIMIT [121] and FaceForensics++ [188] datasets, to the larger, more recent Facebook Deepfake Detection Challenge (DFDC) [47] and Celeb-DF [132] datasets, the last three of which we use in this work.

6.3 Method

We seek to develop an explainable infrastructure for detecting manipulated images and videos. Below, we explain our approach for building separate part-based classifiers for each region of the face. These classifiers - which must make their decisions by looking at only a limited portion of the image - allow us to get the explainable detectors we desire, which will be critical in performing our analysis.

6.3.1 Creating the Parts Masks

In order to train parts-specific detectors, we must create individual masks for each region of the face so that they can be used during training. Figure 6.2 illustrates this straight-forward process. We first acquire facial landmarks using the dlib [117] software library. These detected landmarks are subsequently grouped into four categories: nose, mouth, eyes, and chin/jawline. To convert these landmarks into masks, we simply take the convex hulls of each region separately, except for the chin which we keep as a series of line segments. We then perform 8 iterations of

morphological dilation and apply a gaussian filter to create the final high resolution masks. Since these masks will be used to train low-resolution filters within the network described below, the final step in our mask generation pipeline is to downsample the mask to a resolution that matches the maps produced by our network.

6.3.2 Parts Detection

Single Part-Based Classifier We first describe our detector in the context of a single fixed facial region R, such as the mouth or eyes, which we will refer to as an R-based detector. Since we are only interested in a small portion of the face at a time, we opt to use a patch-based detector with a small receptive field. Thus, as a backbone for our network we use a standard Xception [36] neural network, truncated after the second Xception block and given a single channel of output using a fully-connected layer. This is the same architecture used in [28], and we adopt it in part because the authors of [28] demonstrated that a truncated Xception network generalizes more effectively than the full, deeper network.

Unlike [28], however, we do not directly classify from the output of the truncated Xception network. Instead, in the case of fake images, we use these truncated outputs in order to learn a mask for the region R. Specifically, if x is our input image, $f(x)$ is the output of our truncated neural network, and M_R is the binary mask for region R constructed as described in Section 6.3.1, then we train an R-based classifier by minimizing standard binary cross-entropy loss:

$$\begin{aligned} \mathcal{L}_M(x, M_R) = &-\sum_{i,j} M_{R,ij} \log(\sigma(f(x)_{ij})) \\ &-\sum_{i,j} (1 - M_{R,ij}) \log(1 - \sigma(f(x)_{ij})) \end{aligned} \tag{6.1}$$

where i and j are the horizontal and vertical indices of a given map, and $\sigma(\cdot)$ is the sigmoid function.

If the image being classified is real, then M_R in equation 6.1 is set to all zeros, teaching the network not to activate when given a real image. Thus, $\mathcal{L}_M$ teaches the network to only look for artifacts of forgery in the specific region targeted by M_R.

Once $f(x)$ has been generated, we produce a final classification label $\hat{y}$ by performing average pooling over $f(x)$ and feeding the result through a classification layer. We denote this

operation as $\hat{y} = g(f(x))$. $\hat{y}$ is trained by minimizing binary cross entropy as well, this time over a single value only:

$$\mathcal{L}_C(x) = -y\log(\sigma(\hat{y}) - (1-y)\log(1-\sigma(\hat{y})) \tag{6.2}$$

where y is the ground truth label of the image. The fact that the network must make its prediction solely from the pooled results of the part detector ensures that only the activations of the part detector are used in the final classification.

Thus, for an R-based classifier over a single region, our final loss is:

$$\mathcal{L}_R(x) = \mathcal{L}_C(x) + \lambda\mathcal{L}_M(x, M_R) \tag{6.3}$$

where λ is a learning weight parameter set to 10 in all of our experiments.

We take a moment here to remark on the small receptive field of our classifier. [28] argued that this smaller receptive field helps with generalization, but it also provides another advantage in our analysis. Namely, it ensures that our network is only looking at the regions we want it to. If the receptive fields were large enough to see the whole image, then it would conceivably be possible that the network might perform classification in an implicit two step process that bypasses looking for artifacts in its assigned region: first identify if the image is real or manipulated, then detect the appropriate part. With a small receptive field, though, the network can make its decision only by looking in the area local to whichever region it is focused on classifying. Thus, in all our following analysis, we can be confident that the classifier is making its decision solely from observing the specific region we have trained it to examine.

Multiple Parts-Based Classifier In addition to performing experiments with single-parts based classifiers, we also perform analysis on detectors that combine multiple regions. Since we do not wish to train multiple networks from scratch in order to perform this recombination step, we seek a way to reuse as much computation as possible without having the networks interfere with each other.

Our proposed solution to this problem is illustrated in Figure 6.3. When using multiple parts, instead of simply truncating at the second block of the Xception architecture, we add an

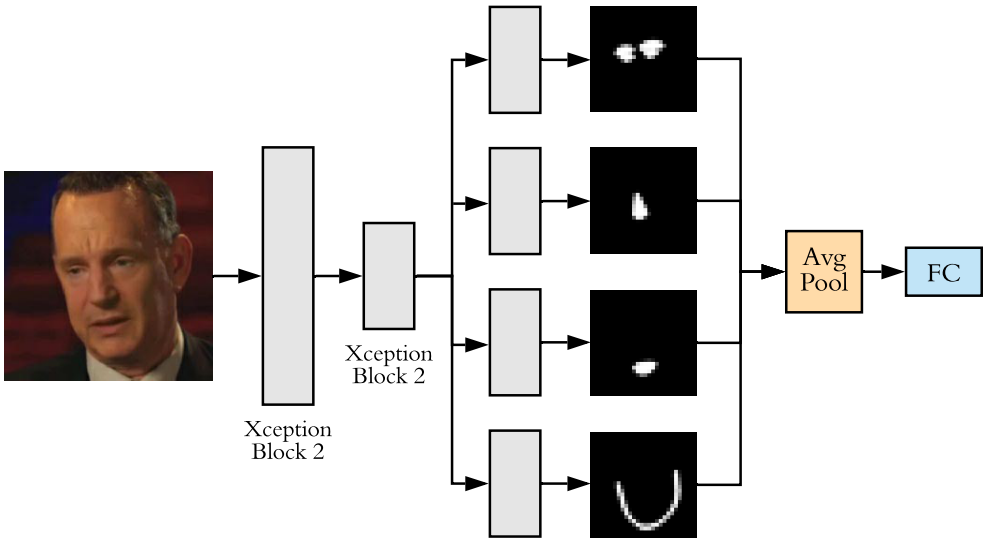


Figure 6.3: Overview of the architecture used in our experiments. We truncate the Xception architecture after two blocks, then divide the network into separate branches for each part of the face, learning a separate parts mask for each branch. We then concatenate and pool all of the masks into a single value. Finally, we feed this into a fully connected layer to make a prediction.

additional Xception block, identical to the previous one, for each of the four parts of the face we are detecting. None of these additional blocks share weights, thus ensuring that they are each performing their detections separately. We then average the results of all four maps to make our final predictions.

Our final loss for the multiple parts-based classifier is thus:

$$\mathcal{L}_{multi}(x) = \mathcal{L}_C(x) + \lambda \sum_R \mathcal{L}_M(x, M_R) \tag{6.4}$$

6.4 Experiments

6.4.1 Data

The main dataset we use in this chapter is FaceForensics++ [188]. This dataset consists of 5000 videos broken into three splits: a training split of 3600 videos (720 real and 2880 fake), a validation split of 700 videos (140 real and 560 fake), and a test split of 700 videos (140 real and 560 fake). FaceForensics++ labels the algorithms used to generate each manipulated video,

and for each real video fakes are designed using one of the the following algorithms: FaceSwap[2], Deepfakes[3], Face2Face [230], and NeuralTextures [229]. For the bulk of our experiments, we will train a system on one of these algorithmic splits, and evaluate on the other three.

For experiments measuring the transfer between datasets, there are two other datasets we use. The first of these is the Celeb-DF v2 dataset [132], which contains 6229 videos, of which 518 are in the test split that we use in our experiments. These videos were generated using an improved version of the standard Deepfake synthesis algorithm from 590 different YouTube videos of celebrities.

The second is the Facebook Deepfake Detection Challenge dataset [47], which at the time of this writing is the largest publicly-available Deepfake dataset collected. We will only be performing our evaluation on the publicly available test set of 5000 videos, which were created using the algorithms DFAE [170], MM/NN Face Swap [100], NTH [276], and FSGAN [157].

6.4.2 Implementation Details

For all of our experiments, we use an Xception backbone [36], pretrained on ImageNet [43]. The networks are trained on 2 Nvidia GeForce RTX 2080 Ti GPUs for 40,000 steps each with Adam [118] using a batch size of 128, a β_1 value of 0.928, weight decay of 0.00005 and an initial learning rate of 0.0001. Every 10000 steps of training, the learning rate is reduced by a factor of 10. All of our code is written in the Tensorflow [3] python library for Deep Learning.

For every video in each dataset, we randomly select 40 frames to use for training. For all images, we use the Dual Shot Face Detector (DSFD) [130] method to detect the faces within each frame. We then take a crop of this face and resize it to 288×288 using nearest neighbor interpolation for all of our Xception training. When training the ResNet50 baselines, we use images of size 224×224 instead. No augmentation is used during training or testing, and all images are compressed as high-quality JPEGs before being fed through the system.

6.4.3 FaceForensics Generalization

We first explore the effectiveness of each of our parts detectors individually. With this analysis, the purpose is to see what can be achieved with networks trained only to identify fake

[2] https://github.com/MarekKowalski/FaceSwap
[3] https://github.com/deepfakes/faceswap

Model		DF	F2F	FS	NT		DF	F2F	FS	NT
ResNet50	DF	0.98	0.51	0.48	0.57	FS	0.5	0.54	0.99	**0.49**
ResNet50 Block 2	DF	0.99	0.55	0.47	0.68	FS	**0.54**	0.65	1	0.42
Xception	DF	0.98	0.52	**0.49**	0.61	FS	0.51	0.58	0.99	0.5
Xception Block 2	DF	0.91	0.6	0.43	0.79	FS	0.48	0.62	0.92	0.29
Nose	DF	0.97	**0.63**	0.33	0.81	FS	0.52	0.54	0.99	0.41
Mouth	DF	0.94	0.56	0.48	0.76	FS	0.45	0.63	0.96	0.22
Eyes	DF	0.96	0.6	0.4	0.84	FS	0.5	0.62	0.97	0.38
Chin	DF	0.96	0.59	0.33	**0.85**	FS	0.44	**0.73**	0.99	0.33
Eyes+Chin	DF	0.97	0.58	0.42	0.76	FS	0.48	0.71	0.99	0.41
Combined	DF	0.97	0.62	0.39	0.83	FS	0.52	0.56	0.98	0.32
ResNet50	F2F	0.57	0.98	0.5	0.56	NT	0.56	0.51	0.48	0.94
ResNet50 Block 2	F2F	0.66	0.99	0.54	0.65	NT	0.67	0.52	0.43	0.98
Xception	F2F	0.58	0.98	0.52	0.54	NT	0.59	0.6	0.5	1
Xception Block 2	F2F	0.7	0.94	0.64	0.74	NT	0.69	0.55	0.42	0.98
Nose	F2F	0.65	0.99	0.52	0.63	NT	0.67	0.63	**0.55**	0.98
Mouth	F2F	0.53	0.98	0.65	0.52	NT	0.64	0.63	0.54	0.99
Eyes	F2F	**0.76**	0.98	**0.66**	0.73	NT	0.64	0.51	0.47	0.99
Chin	F2F	0.75	0.95	0.65	0.74	NT	**0.84**	**0.68**	0.38	0.99
Eyes+Chin	F2F	0.56	0.98	0.62	0.48	NT	0.77	0.61	0.47	0.98
Combined	F2F	**0.76**	0.95	0.53	**0.77**	NT	0.66	0.62	0.53	0.98

Table 6.1: AUC for the ROC curves of the parts-based detectors for each of the four parts of the face, as well as for the combined detector. The second column indicates which split of FaceForensics++ was used to train the model, while the other columns show the performance on each of those splits. Baselines are above the dotted lines and our parts-based detectors are below. Best results for each run are in bold, while models evaluated on the same split they were trained with are in grey.

imagery with respect to a specific location in the image, as described in Section 6.3.2.

Table 6.1 shows our performance on these methods. Here, our parts detectors are trained individually on each of the four algorithm-specific splits of the FaceForensics++ dataset, then evaluated on the other three algorithms. For each system we report the Area Under Curve (AUC) of the Receiver-Operating Characteristic (ROC) curve. We use AUC in part because we found that accuracy can be an unstable metric when measuring forgery detection generalization, since during transfer it is not uncommon for a system to be much better at classifying real images than fake images.

We compare our parts detector to 4 other baseline systems. Two of these systems are the standard Xception [36] and ResNet50 [92] architectures, each pretrained on ImageNet [44]. For each of these architectures, we also compare to a truncated variant, where we end the architecture after the second block (which is either a ResNet or Xception block) similar to [28], and compute

our prediction as the average of all the logits. This is effectively the same as training the parts-based classifier without $\mathcal{L}_C$, and assigning M_R to be either all zero or all one if the image is real or fake, respectively.

We find from Table 6.1 that certain parts detectors are superior to others. For instance, the mouth-based detector rarely out-preforms the Xception based detectors, generally performing much worse. This can be seen, for instance, when observing the performance in transferring from Face2Face to Deepfakes or NeuralTextures. As we will see in Section 6.4.3, this may be correlated to different patterns of artifacts in those regions.

The eyes and chin-based detectors, on the other hand, perform considerably better, often out-performing or at least matching the baselines. In some cases, such as with transfer from FaceSwap to Face2Face, or from NeuralTextures to Deepfakes, we see significant improvements in performance from using the chin detector alone. This strong performance suggests that important artifacts of the forgery process are left behind in these regions.

We observe as well that the nose-based detector falls somewhere in between the others. It often performs poorly, but on certain transfers it performs well, such as when transferring from Deepfakes to Face2Face, or from NeuralTextures to FaceSwap.

The rows marked "Combined" show the performance of our four-branch multi-headed parts detector using all branches. In many cases, we see that the combined detector falls somewhere in between the best and worst parts detectors for each given split.

Since the chin and eyes-based detectors tend to perform better than the other parts, we also include an "Eyes+Chin" detector, which uses two branches instead of the four used by the "Combined" method. Like the "Combined" method, however, we see that combining the detectors does not necessarily greatly improve performance.

Dataset Parts Breakdown

In order to better make sense of the performance discussed in Table 6.1, and in an effort to provide some intuition for certain results, we now perform pixel-level analysis on the FaceForensics++ dataset itself. To do this, we first compute, for one frame per video per algorithm split, the absolute difference between the manipulated image and its real counterpart. We then take the same masks M_R used to generate the ground truth for the R-based part detectors, and multiply them by this computed absolute difference. Specifically, for each part R, given real image x_r and

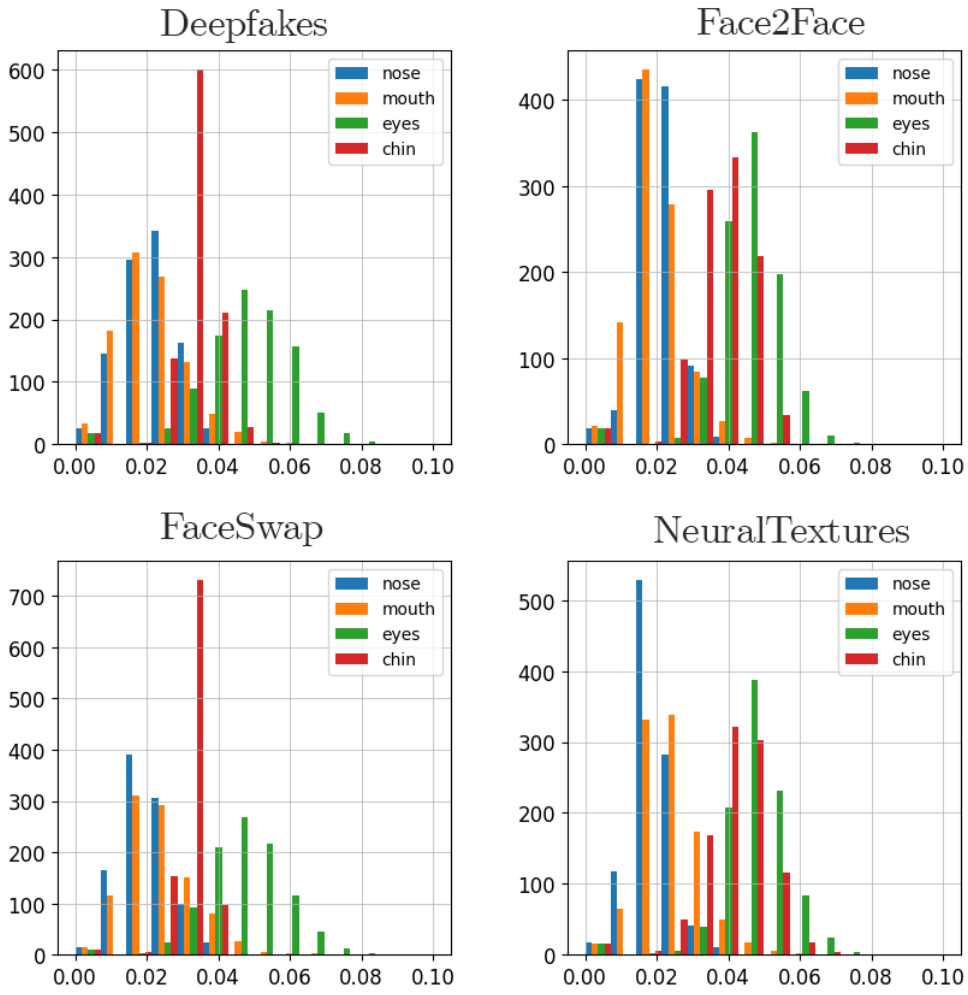


Figure 6.4: Histograms of absolute pixel difference in each of the four regions of the face used in our analysis, broken down by algorithmic split of the FaceForensics++ dataset.

manipulated image x_f, we compute a map D_R as

$$D_{R,ij} = M_{R,ij} \left| x_{r,ij} - x_{f,ij} \right| . \tag{6.5}$$

In Figure 6.4 and Table 6.2, we provide statistical summaries of the maps D_R for each part R and each data split.

We posit that these statistics tell us something about where to find artifacts left over from the manipulation process. Although these summary statistics leave out a lot of important low-level information, some patterns do emerge when comparing certain results from Table 6.1 to the histograms in Figure 6.4. For instance, the Nose detector performs well on the transfer from

	Nose	Mouth	Eyes	Chin
Deepfakes	0.0217	0.0218	0.0479	0.0325
Face2Face	0.0218	0.0206	0.0450	0.0371
FaceSwap	0.0206	0.0239	0.0470	0.0313
NeuralTextures	0.0196	0.0239	0.0469	0.0410

Table 6.2: Average normalized absolute difference between real and manipulated images, broken down by region of the face and manipulation algorithm used. We see that most algorithms have somewhat similar distributions, with the largest changes occurring in the eyes and chin.

NeuralTextures to FaceSwap (.55 vs a .42 truncated Xception baseline and a .5 Xception baseline), and we all see very similar histogram shapes for pixel differences in the nose region on those two datasets.

On the other hand, where distributions are very different, we see more discrepancy in performance. For instance, between Face2Face and Deepfakes we see very different distributions around the mouth regions, where Face2Face has a higher peak and Deepfakes is flatter. This may reflect the poor transfer performance of the mouth-based detector when trained on Face2Face and evaluated on Deepfakes (.53 vs a .7 baseline for truncated Xception).

Finally, when we look at Table 6.2 and observe the average absolute differences within regions, we see other distinct patterns. The most obvious pattern is that different parts have similar amounts of changes even between splits - there is far more variation between parts than between algorithms, with eyes changing the most while the nose and mouth change the least.

Beyond that, though, we observe other interesting patterns with respect to the performance of parts detectors. For instance, we see that in the NeuralTextures split the chin region changes far more than in the other splits, and in fact in Table 6.1 we see that the chin-based detector trained on NeuralTextures considerably out-performs the baseline in the Deepfake and Face2Face dataset splits (.84 and .68 vs .69 and .55, respectively). The one split where the chin-based detector consistently under-performs the baseline is FaceSwap (.38 vs .42), the split with the smallest changes - and therefore presumably the fewest artifacts - in the chin region.

Learned Masks

One advantage to using parts-based classifiers is their explainability. When a parts-based system identifies an image or video as real or fake, one need only look at the predicted masks for hints at what the system was used to make its prediction. In Figure 6.5 we provide two examples

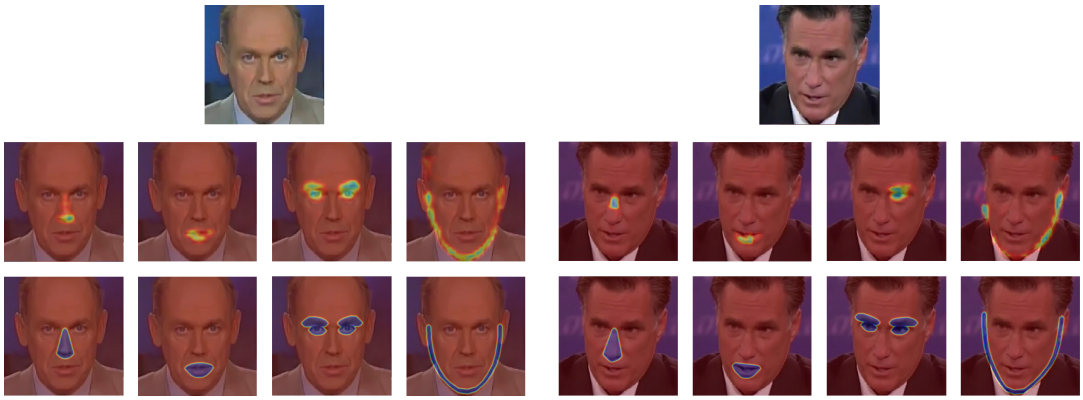


Figure 6.5: Masks learned by individual part detectors. The top row shows the masks learned by the various parts detectors, while the bottom row shows the ground truth masks constructed as described in Section 6.3.1 Left: a manipulated image from the Deepfakes FaceForensics++ split, evaluated by parts-based models trained on the Deepfakes split. Right: A manipulated image from the Face2Face split of FaceForensics++, evaluated on models trained on the Deepfakes split.

Model	DF	F2F	FS	NT
Mean	0.601	0.486	0.643	0.597
Max	0.645	0.529	0.58	0.571
FC	0.573	0.519	0.583	0.555
Ensemble	0.587	0.462	0.576	0.61

Table 6.3: Different aggregation methods for the composite parts-based model. "Mean" is the method used in the rest of the chapter which performs average pooling, "Max" performs max pooling, "FC" adds a fully connected layer, and "Ensemble" runs a separate network for each part, averaging the final logits.

of these masks for each of the four facial regions. On the left we have the relatively easy scenario of identifying a fake from the same distribution as the network was trained on, in this case the Deepfakes split of FaceForensics++. We see that the system easily identifies all of the relevant regions as fakes.

On the right side of Figure 6.5, we have a sample of masks generated by parts-based detectors trained on the Deepfakes split and evaluated on Face2Face. Even transferring between splits, we see that the network is still able to find enough artifacts of the forgery process to be effective. For instance, the detector is still able to successfully pick up the small discoloration on the individual's nose, even if the masks are less clean and more likely to miss certain regions, such as the left eye.

Model	DF	F2F	FS	NT
0 Blocks	0.528	0.446	0.523	0.56
1 Block	0.601	0.486	0.643	0.597
2 Blocks	0.575	0.497	0.582	0.565

Table 6.4: Performance of the aggregated parts-based model, trained with different numbers of Xception blocks included after truncation.

Architecture Design

Choice of Aggregation Function In all of the experiments we explored above, we chose to use an average pooling layer for aggregation in order to aggregate the results of our multi-headed parts detector. In Table 6.3, we explore the choice of other aggregation methods, including a max pooling layer and a trained fully connected layer. These layers are all applied after the individual parts-maps are aggregated separately in the spatial domain using an average pooling layer. We also include one additional method of aggregation labeled "Ensemble", which is attained by training each part detector with a completely different network, and then averaging the logits in an ensemble. Though this would of course be less practical since it involves training and running multiple networks, and the performance gains do not make it worthwhile, we do note that the final ensemble has only 1.6 million parameters and uses 14.6 billion FLOPS, which is still less than a full Xception architecture containing over 20 million parameters and using over 15 billion FLOPS.

In order to compute the values in Table 6.3, we train and evaluate each of our architectures on the same sixteen combinations of splits from the FaceForensics++ dataset that we used in Table 6.1. We then average all the results for a given split, ignoring results trained and evaluated on the same split. For example, for the Deepfakes split, we average the AUC results on the Deepfakes split for the architecture trained on Face2Face, FaceSwap, and NeuralTextures. This provides us with a good summary statistic for comparing the performance of all three architectural options. The full values obtained by all of these runs are included in the supplementary material.

We find that average and max pooling perform similarly, each out-performing the other in two out of the four categories. The fully-connected layer is generally an inferior form of aggregation, perhaps because it can encode biases for one part or another that do not transfer well between dataset splits.

Number of Additional Blocks Another axis of variation in our architecture is the choice of the number of additional Xception blocks used. Adding these extra blocks is an essential step,

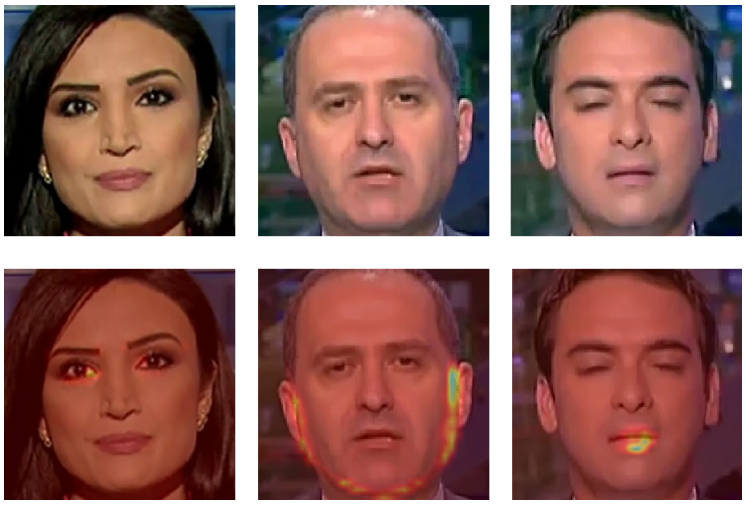


Figure 6.6: False positives for the eyes (left), chin (middle), and mouth (right) detectors. All three are incorrectly firing on regions within real images.

because they allow the different part detectors to operate separately. We explore this in Table 6.4, where we have trained the combined architecture using average-pooling aggregation over 0, 1, and 2 additional Xception blocks. Here, we aggregate values in the same manner as we did in Table 6.3, and once again note that the full experiments can be found in the supplementary material. From this analysis, we find that adding one additional Xception block after truncation is optimal in most cases, with only a small loss with respect to the Face2Face split (from .497 to .486), which is easily outweighed by the larger gains made in the other three splits. The poor performance of using zero blocks in particular confirms our hypothesis that it is necessary to have at least some degree of separate processing for the individual parts-based models, as trying to detect all parts in a single branch will cause the detectors to interfere with one another and performance will degrade.

Failure Case Analysis

We can also find new insights by looking at the failure cases for a given parts-detector. Figure 6.6 shows some examples of false positives detected by our system for different part-based detectors. More such false positives can also be found in the supplementary material. While we find that these failures look generally how we would expect them to, showing activations in areas where they should not exist, for combined parts detectors the ability to observe these maps when the algorithm fails would help an observer understand which regions of the image caused the network to trigger incorrectly.

Model	FF++	Celeb-DF	DFDC
Xception	0.965	0.629	**0.673**
Xception Block 2	0.754	0.622	0.593
Nose	0.909	**0.667**	0.611
Mouth	0.914	0.658	0.617
Eyes	0.847	0.63	0.586
Chin	0.92	0.644	0.618
Average	0.931	0.633	0.627

Table 6.5: Performance of various systems when trained on the entire FaceForensics++ dataset and evaluated on the Celeb-DF and Facebook DFDC datasets. Our parts-based detectors are below the dotted line. Best results are in bold.

6.4.4 Cross-Dataset Generalization

We have shown that parts-based detectors generalize well between splits of the FaceForensics++ datasets. In this section, we evaluate the generalization performance of parts-based detectors trained on the entirety of the FaceFroensics dataset and evaluated on two other datasets, Celeb-DF and Facebook DFDC. For both of these datasets, our evaluation is only on the publicly-available test splits.

The results of these experiments are shown in Table 6.5, using the same AUC metric used above. We observe that, with respect to the Celeb-DF dataset, parts-based detectors are generally superior to the Xception baselines, both truncated and otherwise. This shows that our method remains effective relative to other techniques even as the difference between training and evaluation grows.

However, for the DFDC dataset, we find that parts detectors are not sufficient to achieve state-of-the-art performance over the Xception baseline. However, we still observe that all parts-based methods out-perform the truncated Xception baseline, which itself outperformed the Xception baseline in the vast majority of other transfer tasks. Overall, this indicates that some of the data in the DFDC dataset might require a more global approach in order to perform proper detection, whereas more local approaches with smaller receptive fields, such as truncated Xception and parts-based classifiers, simply do not have sufficient receptive fields. This also opens up the possibility of future work into parts-based detectors that use much larger receptive fields, perhaps with variants of U-Net [187] or similar architectures.

6.5 Conclusion

In this work, we have shown that it is possible to use neural networks trained only to look in specific regions of the face to improve generalization performance between video manipulation algorithms. This suggests that these individual parts of the face may in some cases actually be more representative than the rest of the image as a whole, since restricting the classifier's attention to only these parts improves generalization. Having observed this, we used these parts-based detectors to perform extensive empirical analysis, analyzing which parts are most discriminative between datasets and examining the underlying distributions of our data.

6.6 Supplementary Material

6.6.1 Full Data

Here we include more complete data from our experiments. Table **??** shows complete data for Table 1 in the full chapter, Table **??** shows complete data for Table 3 in the full chapter, and Table 6.4 shows complete data for Table 4 in the full chapter.

For Table **??**, the key difference is in the addition of the accuracy metric, computed as the average accuracy of real and fake computation. Since for each split there are the same number of real and fake images, the metric does not need to be re-balanced. Tables **??** and 6.4 contain the accuracy metric as well, but also include detailed AUC numbers per split which were not included in the full chapter.

Model		DF Acc	F2F Acc	FS Acc	NT Acc	DF AUC	F2F AUC	FS AUC	NT AUC
ResNet50	DF	0.96	0.5	0.48	0.55	0.98	0.51	0.48	0.57
ResNet50 Block 2	DF	0.95	0.5	0.5	0.54	0.99	0.55	0.47	0.68
Xception	DF	0.97	0.51	0.5	0.58	0.98	0.52	0.49	0.61
Xception Block 2	DF	0.87	0.54	0.45	0.72	0.91	0.6	0.43	0.79
Nose	DF	0.87	0.57	0.41	0.74	0.97	0.63	0.33	0.81
Mouth	DF	0.82	0.55	0.44	0.7	0.94	0.56	0.48	0.76
Eyes	DF	0.88	0.54	0.49	0.74	0.96	0.6	0.4	0.84
Chin	DF	0.89	0.52	0.48	0.75	0.96	0.59	0.33	0.85
Eyes+Chin	DF	0.92	0.52	0.49	0.67	0.97	0.58	0.42	0.76
Combined	DF	0.9	0.51	0.5	0.67	0.97	0.62	0.39	0.83
ResNet50	F2F	0.56	0.97	0.51	0.53	0.57	0.98	0.5	0.56
ResNet50 Block 2	F2F	0.53	0.98	0.51	0.52	0.66	0.99	0.54	0.65
Xception	F2F	0.53	0.98	0.5	0.51	0.58	0.98	0.52	0.54
Xception Block 2	F2F	0.66	0.71	0.63	0.68	0.7	0.94	0.64	0.74
Nose	F2F	0.54	0.96	0.53	0.53	0.62	0.98	0.58	0.57
Mouth	F2F	0.51	0.94	0.55	0.49	0.53	0.98	0.65	0.52
Eyes	F2F	0.6	0.93	0.58	0.58	0.73	0.97	0.65	0.72
Chin	F2F	0.59	0.89	0.56	0.58	0.75	0.95	0.65	0.74
Eyes+Chin	F2F	0.5	0.94	0.53	0.48	0.56	0.98	0.62	0.48
Combined	F2F	0.68	0.87	0.52	0.71	0.76	0.95	0.53	0.77
ResNet50	FS	0.5	0.52	0.97	0.49	0.5	0.54	0.99	0.49
ResNet50 Block 2	FS	0.51	0.52	0.97	0.49	0.54	0.65	1	0.42
Xception	FS	0.5	0.54	0.98	0.5	0.51	0.58	0.99	0.5
Xception Block 2	FS	0.48	0.58	0.88	0.45	0.48	0.62	0.92	0.29
Nose	FS	0.51	0.53	0.96	0.49	0.53	0.56	0.98	0.47
Mouth	FS	0.48	0.59	0.8	0.34	0.45	0.63	0.96	0.22
Eyes	FS	0.51	0.52	0.96	0.49	0.53	0.64	0.98	0.45
Chin	FS	0.49	0.52	0.95	0.49	0.44	0.73	0.99	0.33
Eyes+Chin	FS	0.5	0.54	0.92	0.49	0.48	0.71	0.99	0.41
Combined	FS	0.54	0.55	0.93	0.44	0.52	0.56	0.98	0.32
ResNet50	NT	0.55	0.5	0.49	0.93	0.56	0.51	0.48	0.94
ResNet50 Block 2	NT	0.55	0.5	0.5	0.96	0.67	0.52	0.43	0.98
Xception	NT	0.55	0.57	0.5	0.98	0.59	0.6	0.5	1
Xception Block 2	NT	0.53	0.5	0.5	0.93	0.69	0.55	0.42	0.98
Nose	NT	0.52	0.51	0.5	0.98	0.54	0.51	0.5	0.98
Mouth	NT	0.52	0.51	0.5	0.95	0.64	0.63	0.54	0.99
Eyes	NT	0.54	0.5	0.5	0.97	0.64	0.53	0.47	0.99
Chin	NT	0.67	0.54	0.49	0.97	0.84	0.68	0.38	0.99
Eyes+Chin	NT	0.57	0.52	0.5	0.97	0.77	0.61	0.47	0.98
Combined	NT	0.52	0.5	0.5	0.96	0.66	0.62	0.53	0.98

Table 6.6: AUC for the ROC curves of the parts-based detectors for each of the four parts of the face, as well as for the combined detector. The second column indicates which split of FaceForensics++ was used to train the model, while the other columns show the performance on each of those splits. This is an extension of Table 1 in the full chapter. Here we include both standard accuracy (percentage of correct detections in each class) and AUC of the ROC curve.

Model		DF Acc	F2F Acc	FS Acc	NT Acc	DF AUC	F2F AUC	FS AUC	NT AUC
FC	DF	0.92	0.51	0.49	0.56	0.98	0.57	0.49	0.63
Mean	DF	0.9	0.51	0.5	0.67	0.97	0.62	0.39	0.83
Max	DF	0.89	0.54	0.47	0.67	0.96	0.58	0.45	0.73
Ensemble	DF	0.93	0.53	0.46	0.72	0.98	0.61	0.41	0.83
FC	F2F	0.63	0.83	0.58	0.67	0.66	0.97	0.59	0.69
Mean	F2F	0.68	0.87	0.52	0.71	0.76	0.95	0.53	0.77
Max	F2F	0.63	0.87	0.62	0.56	0.72	0.96	0.69	0.65
Ensemble	F2F	0.54	0.96	0.53	0.53	0.77	0.99	0.68	0.74
FC	FS	0.51	0.56	0.95	0.48	0.49	0.59	0.98	0.43
Mean	FS	0.54	0.55	0.93	0.44	0.52	0.56	0.98	0.32
Max	FS	0.5	0.55	0.96	0.49	0.5	0.73	0.99	0.36
Ensemble	FS	0.51	0.53	0.96	0.49	0.5	0.62	0.99	0.33
FC	NT	0.55	0.51	0.49	0.95	0.59	0.55	0.48	0.98
Mean	NT	0.52	0.5	0.5	0.96	0.66	0.62	0.53	0.98
Max	NT	0.64	0.53	0.5	0.97	0.77	0.63	0.45	0.98
Ensemble	NT	0.52	0.51	0.5	0.98	0.76	0.6	0.42	0.99

Table 6.7: Different aggregation methods for the composite parts-based model. "Mean" is the method used in the rest of the chapter which performs average pooling, "Max" performs max pooling, "FC" adds a fully connected layer, and "Ensemble" runs a separate network for each part, averaging the final logits. This is an extension of Table 3 in the full chapter. Here we include both standard accuracy (percentage of correct detections in each class) and AUC of the ROC curve.

Model		DF Acc	F2F Acc	FS Acc	NT Acc	DF AUC	F2F AUC	FS AUC	NT AUC
0 Blocks	DF	0.83	0.54	0.42	0.68	0.93	0.55	0.32	0.75
1 Block	DF	0.9	0.51	0.5	0.67	0.97	0.62	0.39	0.83
2 Blocks	DF	0.91	0.53	0.46	0.7	0.96	0.6	0.41	0.79
0 Blocks	F2F	0.52	0.53	0.5	0.55	0.52	0.53	0.5	0.55
1 Block	F2F	0.68	0.87	0.52	0.71	0.76	0.95	0.53	0.77
2 Blocks	F2F	0.52	0.94	0.52	0.5	0.6	0.99	0.6	0.63
0 Blocks	FS	0.52	0.5	0.87	0.44	0.49	0.5	0.93	0.27
1 Block	FS	0.54	0.55	0.93	0.44	0.52	0.56	0.98	0.32
2 Blocks	FS	0.52	0.55	0.91	0.43	0.49	0.58	0.99	0.32
0 Blocks	NT	0.52	0.51	0.52	0.84	0.55	0.53	0.52	0.92
1 Block	NT	0.52	0.5	0.5	0.96	0.66	0.62	0.53	0.98
2 Blocks	NT	0.51	0.5	0.5	0.95	0.58	0.54	0.48	0.98

Table 6.8: Performance of the aggregated parts-based model, trained with different numbers of Xception blocks included after truncation. This is an extension of Table 4 in the full chapter. Here we include both standard accuracy (percentage of correct detections in each class) and AUC of the ROC curve.

7. 3D HUMAN POSE ESTIMATION FROM DEEP MULTI-VIEW 2D POSE

7.1 Introduction

Articulated human pose estimation is a long-studied problem in the field of computer vision [9, 58] which involves estimating a parameterized 2D or 3D human body model from video or still imagery (see Figure 7.1). Pose estimation has many direct applications including motion capture for film and game production and may also be done as an pre-precessing step for higher-level vision problems such as action recognition [243, 244]. Most of the recent progress made in this area has leveraged advances in deep neural networks, particularly convolutional networks [127], to achieve impressive results in the areas of 2D and 3D pose estimation from single images [24, 154, 232, 233].

While significant breakthroughs, the utility of these approaches are limited by the single-view nature of the source data. 2D pose estimation techniques only detect joint and limb locations in image space, and cannot reveal anything directly about the positions and orientations of limbs in 3D. Approaches exist which perform 3D pose estimation on a single image [24, 154, 232, 233], but the range of 3D poses is heavily limited by the training dataset and limbs which are occluded must be inferred rather than directly measured. For applications which demand precise measurement of all 3D limbs, a synchronized network of overlapping calibrated cameras can be used to resolve single-view ambiguities and improve 3D triangulation accuracy.

To date, most published literature on multi-view pose estimation has relied on older 2D pose estimation techniques from before the deep learning revolution as a basis for their 3D fusion [8, 13]. Many techniques make no use of valuable temporal information when applied to video [169], instead handling each frame as an independent detection problem. Additionally, these techniques are usually focused on situations involving only one human actor of interest, and cannot handle multiple humans moving inside the scene [206, 207]

To accelerate research in multi-person, multi-camera 3D pose estimation, Amin *et al*. introduced the TUM Campus and Shelf datasets [8, 13], each of which contain video footage of several

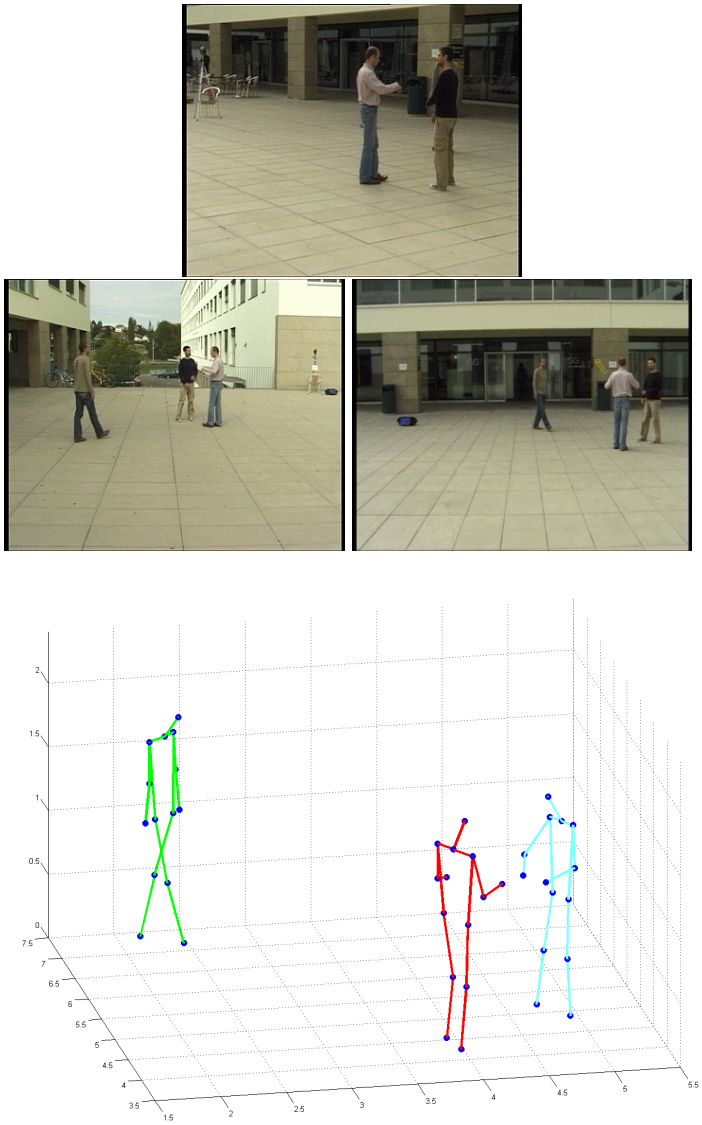


Figure 7.1: Three coincident sample frames (top) from the TUM Campus dataset [8] and a visualization of the 3D pose estimated for each person in the scene by our pipeline (bottom).

actors taken from 2-3 calibrated cameras. This dataset presents an opportunity to face challenges not present in single image pose estimation. For example, actors in the scene are frequently fully occluded by other actors in one view, and in this situation it becomes necessary to keep track of the actors through the occlusion event.

To address these challenges Amin *et al.*, and later Belgiannis *et al.*, use the sum-product belief propagation algorithm to optimize over the space of 3D pose configurations, which they discretize by triangulating the results of a pose detector. The factor graph they use features a large number of parameters for factors such as joint collision and temporal smoothing. For their system to achieve optimal results, however, they need to use a structured SVM solver to optimize over a series of hyper parameters [13].

The system we propose, by contrast, achieves significantly better results with fewer constraints on limb motion. First, we use a modern 2D pose detector [24] which leverages advances in deep learning to generate better 2D limb hypotheses. Next we use a factor graph optimization, where factors are all constructed to be readily interpretable, and as such it is very easy to identify reasonable values for each factor without the need to resort to a more complex hyperparameter optimization scheme. Our final factor graph uses only a three factors: limb location priors, collision terms, and temporal smoothing.

In this chapter, we present our approach: a full pipeline for multiple person, multiple camera 3D pose estimation. After presenting our technique, we empirically show the effectiveness of our technique on the Campus and Shelf datasets, where we in some cases significantly out-perform the previous state-of-art.

7.2 Related Work

7.2.1 2D Human Pose Estimation

Approaches to human pose estimation can be categorized broadly into two separate tasks based on goals: single person pose estimation and multiple person pose estimation. In single person pose detection, a tight bounding box is provided around the person in question. Even early approaches to this problem made use of graphical models, modeling the human body as a kinematic chain - a tree shaped graph connecting various parts [9, 58, 180, 267]. Some variants on these methods give the associated tree a hierarchical structure [221]. [207] handle occlusions

by modifying the tree structure with connections that create loops, and compensate by using an approximate algorithm for inference.

With the emergence of deep learning as a powerful technique for computer vision tasks [91, 124, 127, 226], several techniques were introduced that use deep learning to achieve superior performance in pose estimation tasks. [232] combine a deep network with a Markov Random Field, and [233] frame the problem as a deep network-based regression problem.

Recently, stacked hourglass networks [154] and their variants [38, 153] have emerged as the leading deep learning methods for single person pose estimation. A stacked hourglass network is essentially a fully-convolutional CNN architecture [138] with multiple "hourglass modules" which squeeze the image representation to a very small size, allowing for each unit in the CNN to have a very large receptive field.

Alternatively, multi-person pose estimation seeks a single architecture which, given a single image with multiple people present, can identify the individual people before or during the actual pose estimation process. To this end, a few recent techniques have been developed which all yield good performance [90, 166]. In Mask-RCNN [90], the authors use a deep network to simultaneously detect and segment objects within an image, and demonstrate that their technique can also be extended to effectively detect human poses.

The above multi-person pose estimation techniques represent a "top-down" approach, where people are first identified and then their poses are detected on an individual basis. In "bottom-up" pose detection, the joints or limbs are detected first, and then aggregated into human detections only afterwards. One example of this, among others [104], is [153], who present associative embeddings, which extend hourglass networks to operate on a single image, assigning a value to each identified joint which differentiates it from joints in different skeletons. Cao *et al*. present Part Affinity Fields [24], which, in addition to detecting individual joints, also detects a vector indicating the orientation of each joint. These joints can then be aggregated into separate skeletons, allowing for simultaneous person and part detection.

7.2.2 3D Human Pose Estimation - Single Image

Exploiting the success of 2D human pose estimation techniques, there has been a lot of work done in the area of 3D pose estimation, as inferred from a single image [168, 228, 282]. In particular, [169] uses a modified stacked hourglass to progressively produce 3D pose detections

with more and more depth. [52] uses existing 2D pose annotations as a form of weak supervision to train 3D pose estimation systems.

7.2.3 3D Human Pose Estimation - Multiple Images

While 3D human pose estimation techniques from a single image can generate impressive results, they still do not yield the accuracy that is possible when performing pose estimation with multiple calibrated cameras from different views.

[208] use a graphical model for 3D pose estimation and perform inference with a particle message passing scheme. [49] use a Sum of Gaussian based appearance model, which is later expanded in [48] to make use of deep learned features.

There exist multiple datasets for 3D human pose estimation in controlled indoor environments [106, 206, 207], where multiple cameras observe a single actor in a studio. However, there also exist many alternate datasets that address the problem of 3D human pose detection with multiple humans, or in outdoor environments [8, 49]. In particular, [8] introduce the TUM Campus and Shelf datasets, each of which feature three actors in a single video taken from at least three synchronized cameras. Following up on this, [12] and [13] use factor graphs optimized with belief propagation to perform inference on poses detected with 2D pose estimators.

7.3 Approach

The human body is represented as a graph with $N = 14$ nodes, where each node represents a joint in the human body (*e.g.* left hand, right shoulder, top of head, *etc.*), and edges represent adjacent joints. Our objective is to estimate, for each individual and timestep $t = 0...T$, the 3D joint locations $\mathbf{Y}^t = \{y_i^t | i = 1, 2, \ldots, N\}$ where each $y_i^t \in \mathbb{R}^3$. The input data comes in the form of video frames $\{I_c^t\}$ from each camera c, which have been pre-calibrated such that projection matrices have been computed and video frames have been aligned temporally.

Our processing pipeline for estimating the $\{\mathbf{Y}^t\}$ poses for each person from the video has multiple stages. First, 2D detections of joints are computed for each individual in the image using a state-of-art detector. When multiple people are present in the scene, these detections are used to identify specific individuals by triangulating 2D joint detections from different views. For each individual, a conditional random field (CRF) is constructed over 3D limb location variables to model the dependency on information sources including 2D detection strength, temporal smooth-

ness, and collision terms. Inference on each factor graph is performed using the sum-product belief propagation algorithm to obtain posterior probabilities on limb location, from which the maximum likelihood 3D location is selected. These steps are described in detail in the following subsections.

7.3.1 2D Pose Detection and Cross-frame Association

The first step in our pipeline is to obtain 2D limb detections in each frame of video from each camera. Many published approaches to this problem have made their software available and we choose to use the open source OpenPose library [24], which uses part-affinity fields to simultaneously extract human poses and detect individuals. Along with offering state-of-art 2D detection performance, this software can expose raw limb likelihood heat maps (see Figure 7.2) used to compute the final 2D limb position. These heat maps $\mathbf{H}_c^t = \{H_{c,1}^t, H_{c,2}^t, \ldots, H_{c,N}^t\}$ allow propagation of uncertainty in the 2D detection step to the 3D optimization, where a final decision can be made with all multi-view evidence. Each of these heat maps $H_{c,i}^t$ assigns a value in $[0,1]$ to each pixel in I_c^t indicating the likelihood that the given pixel is displaying the projection of joint i.

Given a collection of 2D poses within each frame, we must match skeletons belonging to the same person both across time and multiple views. Consider an individual $G_c^t = \{g_{c,1}^t, g_{c,2}^t, \ldots g_{c,N}^t\}$ detected in 2D at time t for camera c. Here, each $g_{c,i}^t$ represent the pixel locations of a 2D joint, *i.e.* a local maximum of the corresponding $H_{c,i}^t$. we determine that two detected individuals G_c^t and G_c^{t+1} from two consecutive frames are actually the same person is they have an IoU of at least 0.7. If a detection G_c^t does not have an IoU match with any bounding in the previous frame, then we determine it to be the first occurrence of that individual.

To determine matches between different cameras at the same time point, *i.e.* between two poses G_c^t and $G_{c'}^t$, we cast a ray into the scene for each pair of joints $g_{c,i}^t$ and $g_{c,i}^t$ in the pose, and measure the distance between the closest points of each ray (*e.g.*, the distance between the wrist detection in G_c^t and the same detection in $G_{c'}^t$). If the average distance between these points in less than 2cm, we determine that the two detections belong to the same person.

When we have completed the matchings, we remove from each frame any identities that only appear in a single camera, leaving behind only poses that can be fed into our multi-view system.

Figure 7.2: Heatmap for 2D detection

Figure 7.3: Sampling heatmap

Figure 7.4: (a) OpenPose outputs a heat map $H^t_{c,i}$ for each image, where intensity corresponds to the likelihood that a particular joint exists in that location. In the image above, we show the heat map for a left ankle detection. Notice that while most of the mass of the heat map is on the correct ankle, some of it falls on wrong ankle. (b) In order to generate a set of possible states in $\mathbb{R}^3$ for some joint y^t_i, we convert $H^t_{c,i}$ into a probability mass function and sample from it. We triangulate these samples to generate a set of hypotheses $\mathbf{S}^t_i$.

7.3.2 Conditional Random Field for 3D Pose

At this stage, the joint locations from matched detections in multiple views can be back-projected into 3D and triangulated to obtain a 3D pose estimate. The result is noisy, lacks temporal smoothness frame-to-frame, and may occasionally contain gross error for certain limbs if 2D detections are ambiguous. To improve the quality of the pose estimates, a CRF on 3D joint positions is constructed to allow introduction of additional consistency constraints. The CRF is represented by a factor graph, where the joint positions are the variables and constraints on these joints become factors:

$$p(\{\mathbf{Y}^t\}|\{I_c^t\}) \;=\; \frac{1}{Z}\prod_{t=1}^{T}\prod_{i=1}^{N} f_{data}(y_i^t,\{I_c^t\}) \;\cdot\; f_{temp}(y_i^t,y_i^{t-1},y_i^{t+1}) \;\cdot\; \prod_{t=1}^{T}\prod_{(i,j)\in E_{col}} f_{col}(y_i^t,y_j^t) \quad (7.1)$$

where Z is the partition function. The factors f are not necessarily true probability distributions although we do individually normalize all of our factors into probabilities, as we found that it leads to improved numerical stability.

The first of these factors, $f_{data}(y_i^t)$, represents the likelihood of a joint y_i^t given evidence aggregated from the detections in the source images $\{I_c^t\}$. The second factor is f_{temp}, which prefers the movement of joints between time steps to be temporally smooth. f_{col} constrains certain symmetric pairs of joints not to collide. The set of edges E_{col} represents the joints that cannot collide which in our implementation represent symmetric body parts, *e.g.* right and left wrist, right and left ankle, *etc*. An illustration of the dependencies of these factors on variables in Equation 7.1 is shown in Figure 7.5 for a subset of the full body network. Additional constraints, such as constant limb lengths, may be easily appended to the factorization if desired.

To make inference on the network tractable, the joint positions $\{y_i^t\}$ are limited to take on a discrete number of possible states. To discretize the state space for $\mathbf{Y}$, we use a method similar to that of [13]. Let $\mathbf{S}_i^t = \{s_{i,1}^t, s_{i,2}^t, \ldots, s_{i,M}^t\}$ be the discrete set of M possible states that y_i^t can take on. Our goal is to a select a small set of $s_{i,j}^t \in \mathbb{R}^3$ that well sample the 2D pose heat maps $\{H_{c,i}^t\}$ when projected into each image.

We iteratively select a subset of cameras of size C' with $2 \leq C' \leq C$ (a given joint i will generally be visible from either 2 or 3 cameras in the data we have). We then randomly sample

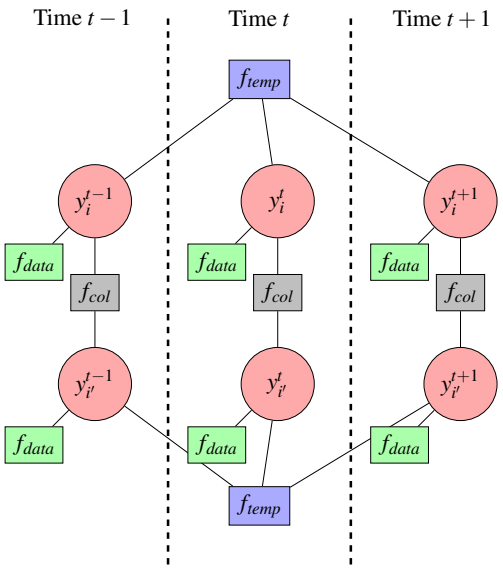


Figure 7.5: A factor graph diagram for a simplified network involving two symmetric joints y_i^t and $y_{i'}^t$ and three timesteps. The joint variables are denoted with circles and the data, temporal, and collision factors are denoted with green, purple, and gray rectangles, respectively.

from the 2D heat map $H_{c,i}^t$ in each camera and back-project the pixel coordinate into a ray. A 3D joint hypothesis $s_{i,j}^t$ is formed by computing the point in $\mathbb{R}^3$ closest to all C' rays. For our implementation, we draw 16 samples this way when $C = 3$ and 64 samples when $C = 2$. In both cases, the final result is $M = 64$ samples, because when $C = 3$, there are three subsets of cameras with $C' = 2$ and one with $C' = 3$. See Figure7.3.

In the subsections that follow, we provide detailed descriptions of the potential functions in (7.1) and our CRF optimization procedure using the sum-product belief propagation algorithm [125] to obtain posterior estimates on the joint positions.

Data Term

The data term $f_{data}(y_i^t)$ in the factor graph measures 2D pose confidence of the projected 3D joint hypothesis. This term is straight-forward to calculate: for each possible state $s_{i,j}^t$, we calculate $f_{data}(s_{i,j}^t)$ by projecting and $s_{i,j}^t$ from world coordinates onto each $H_{c,i}^t$ and averaging the results:

$$f_{data}(s_{i,j}^t) = \frac{1}{C}\sum_{c=1}^{C} H_{c,i}^t(\hat{s}_{i,j}^t) \tag{7.2}$$

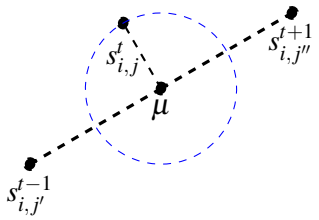


Figure 7.6: We calculate $f_{temp}(s^t_{i,j}, s^{t-1}_{i,j'}, s^{t+1}_{i,j''})$ by finding the midpoint μ between $s^{t-1}_{i,j'}$ and $s^{t+1}_{i,j''}$, then calculating the distance between μ and $s^t_{i,j}$ and modeling the distance with a Gaussian distribution.

where $\hat{s}$ is the projection of s in the image. Referring back to the 2D pose heat maps, as opposed to the single optimal 2D locations, allows this data term to fuse weak evidence in the case of ambiguity, such as bimodal distributions around left/right joints in close proximity to each other.

Temporal Term

The purpose of the temporal term f_{temp} is to bias joints to move smoothly frame-to-frame. Without some form of temporal smoothing, we found that animations of the estimated 3D body models contained significant jitter as result of each frame being calculated separately. Having information flow between time steps also helps the system to recover from short-term confusion events such as occlusion of limbs by the torso during walking.

We model temporal smoothing with a ternary term, involving 3D joint estimates from three consecutive time steps: y^t_i, y^{t-1}_i, and y^{t+1}_i. Specifically:

$$f_{temp}(s^t_{i,j}, s^{t-1}_{i,j'}, s^{t+1}_{i,j''}) \quad = \quad \exp\left[-\frac{1}{2\sigma^2_{temp}}\|s^t_{i,j} - \mu(s^{t-1}_{i,j'}, s^{t+1}_{i,j''})\|\right] \tag{7.3}$$

where $\mu(s^{t-1}_{i,j'}, s^{t+1}_{i,j''}) = \frac{s^{t-1}_{i,j'} + s^{t+1}_{i,j''}}{2}$ is the expected position of the joint using a constant velocity assumption and samples immediately before and immediately after the current time step. A state hypothesis $s^t_{i,j}$ is penalized if it veers too far from the constant-velocity prediction, as seen in Figure 7.6. We experimentally set $\sigma_{temp} = 2$cm, though we found that the results were not very sensitive to the exact choice of σ_{temp} so long as the choice was reasonably small.

Collision Term

The collision term f_{col} prevents symmetric left/right joints from colliding with each other in 3D. This common issue in pose detection is generally caused by the fact that symmetric joints, like the right and left feet, look very similar to each other and are often in close proximity. This often results in a bimodal distribution in the 2D pose heat maps which in turn can cause both symmetric joints in a pair to be assigned to the same location in 3D space.

In contrast to [13], who handle collision with a trained Gaussian on the distance between limbs, we instead use a modified sigmoid function that evaluates to 1 when the two points are reasonably far apart, and 0 when they are too close:

$$f_{col}(s_{i,j}^t, s_{k,j}^t) = \frac{1}{1 + e^{\theta_1 - \theta_2 \|s_{i,j}^t - s_{k,j}^t\|}} \tag{7.4}$$

where θ_1 is a hyperparameter that offsets the center of the sigmoid, and θ_2 is a hyperparameter that tightens the sigmoid. We empirically set $\theta_1 = 15$cm and $\theta_2 = 10$cm. As with σ_{temp}, we find that as long as the values chosen are reasonable, they don't have a huge effect on the results.

7.3.3 Belief Propagation Optimization

The factor graph defines the dependencies between joint variables and input video frames across time and space in the total probability distribution. With the graph constructed, the next objective is to perform inference to obtain posterior estimates of the 3D joint positions $\{y_i^t\}$. Belief propagation is a message-passing algorithm which estimates the posterior distribution in CRFs and has been used in many computer vision applications [177]. In the case where the graphs have loops, such as ours, belief propagation is not guaranteed to converge to the true posterior but may be used as an approximation regardless.

As belief propagation will iteratively optimize all frames of a video, an intelligent schedule for organizing the message passing between factors and variables is needed. Our implementation updates all factor-to-variable messages of the same type before moving on to the next type, i.e. all f_{temp} messages are updated, then all f_{col} messages, etc. The temporal factor update passes messages forward in time through the whole video clip, then back again, so that temporal information is rapidly propagated to distant frames. Future work could easily adapt this schedule to process

Figure 7.7: Three sample images from the Shelf Dataset.

live video by temporally optimizing over a sliding window of frames rather than the entire video clip. We run all of our tests with five iterations of belief propagation across the full network. We experimentally found that running more than five iterations did not change the values because by the fifth iteration the values had effectively converged.

7.4 Experiments

We evaluate our 3D pose estimation pipeline on the TUM Campus and TUM Shelf [8] datasets, which have labled 3D ground truth for multiple people moving in the scene. The inclusion of multiple people adds an extra dimension of challenge not present in many other 3D pose estimation datasets [106, 206], because as the people move and interact it is often the case that they obscure each other.

The TUM Campus and Shelf datasets both feature scenes shot from 3 calibrated cameras containing 2-3 people at any given time. We evaluated the Campus dataset on 220 frames of video with resolution 360×288 and the Shelf dataset on 300 frames with resolution 1032×776. During these test portions of the sequence, actors frequently come in and out of view, occlude each other, and stand stationary for long periods of time. Both datasets also contain ground truth for several hundred other frames which we ignore in our evaluation. See Figures 7.1 and 7.7.

To measure the performance of our algorithm, we use the PCP (percentage of correctly estimated parts) metric, following the example set by [8]. Using the PCP metric, originally described by [59], two adjacent joints are considered to be correctly estimated if their distances from their ground truth location is less than $\alpha = 0.5$ of the length of their limb in the ground truth.

Campus

		Head	Torso	Upper Arm	Forearm	Thigh	Shin	All	Average
Amin	Actor1	64.58	100.00	94.80	66.67	100.00	81.25	85.00	
et al. [8]	Actor2	78.84	100.00	84.66	27.25	98.15	83.33	76.56	76.61
	Actor3	38.52	100.00	83.71	55.19	90.00	70.37	73.70	
Belagiannis	Actor1	96.55	93.10	96.55	86.21	93.10	96.55	**93.45**	
et al. [13]	Actor2	98.24	48.82	97.35	42.94	75.00	89.41	75.61	81.08
	Actor3	93.20	85.44	89.81	74.76	91.75	76.21	84.37	
	Actor1	93.75	100.00	80.21	48.96	100.00	100.00	86.55	
Ours	Actor2	47.37	98.95	91.05	42.89	98.95	98.95	**82.54**	**85.15**
	Actor3	69.57	100.00	90.00	64.35	100.00	100.00	**88.14**	

Shelf

		Head	Torso	Upper Arm	Forearm	Upper Leg	Lower Leg	All	Average
Amin	Actor1	93.75	100.00	73.08	32.99	85.58	73.56	72.42	
et al. [8]	Actor2	100.00	100.00	73.53	2.94	97.06	73.53	69.41	77.3
	Actor3	85.23	100.00	86.62	60.31	97.89	88.73	85.23	
Belagiannis	Actor1	96.29	100.00	82.24	66.67	43.17	86.07	75.26	
et al. [13]	Actor2	78.95	100.00	82.58	47.37	50.00	78.95	69.68	79.00
	Actor3	98.00	100.00	93.15	92.30	56.50	97.00	87.59	
	Actor1	32.14	99.64	95.18	82.50	99.64	99.64	**88.34**	
Ours	Actor2	16.22	100.00	94.59	66.22	100.00	100.00	**85.26**	**88.92**
	Actor3	28.40	100.00	94.14	94.44	100.00	100.00	**91.30**	

Table 7.1: Performance breakdown by part for each part of the human body on the Campus and Shelf datasets. Best results are in bold. The "Average" column is weighted by how often each actor appears in the dataset.

7.4.1 Comparison to Other Methods

In this section, we compare the performance of our system to other state-of-the-art methods on the Shelf and Campus datasets, as broken down by individual joints. The results of these experiments are shown in Table 7.1.

With the exception of Actor 1 in the Campus dataset, our method out-performs all other methods on the six actors of the Campus and Shelf datasets. Within the Campus dataset, we find that the method from [13] performs slightly better on each actor's arms, but that our method makes up the difference by performing better on the legs and torso. We attribute this to differences in the output of the 2D part detectors.

On the other hand, our method performs almost unilaterally better on the Shelf dataset. This discrepancy is likely due to the resolution difference between the two datasets; at a native resolution of 360×288, images from the Campus dataset must be scaled up to be used by the OpenPose 2D pose detector, which processes input at a resolution of 656×368. The Shelf dataset's 1032×776 images, on the other hand, can make use of the full discriminative power OpenPose's part detector. It stands to reason then that the 2D detections made on the Shelf dataset may be

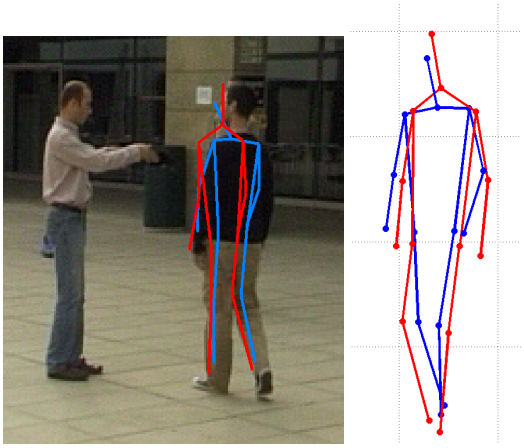


Figure 7.8: A superimposed comparison of our method's output (blue) with the ground truth pose (red) for one frame of the Campus dataset. We note that the head and neck joints of our output are systematically lower than on the ground truth, negatively affecting our accuracy.

more accurate than those made on the Campus dataset.

During computation of these metrics, the 3D head pose output by our system was manually adjusted by a fixed amount to account for a discrepancy between the OpenPose and ground truth body models. OpenPose is trained on the Microsoft COCO 2016 Dataset [136] to detect the head joint around the level of the nose, whereas the ground truth labels the head joint as the top of head. Similarly, OpenPose detects the neck as lying directly between the shoulder blades, as opposed to just above them. An example is shown in Figure 7.8. To perform a fair comparison with ground truth, the head and neck detections output by our system were translated upward in the z direction by 10cm for evaluation. This approximates the correct offset in these datasets since the range of actor pose is limited to upright poses. Longer term we expect that a community standardized human body model will emerge to eliminate these types of discrepancies in future datasets.

We also note that while Belagiannis *et al*. do out-perform our method on one actor out of six, our method always achieves accuracy of at least 82%, whereas the other methods we compare against score below 80% on at least half of the actors, making our method more consistent across sequences.

7.4.2 Analysis of Individual Factors

Table 7.1 breaks down the effect of each term in the factor graph on our final result accuracy. When the data term alone is used, belief propagation is not run and the 3D pose hypothesis with the highest average 2D pose likelihood is selected for the answer. As such, the data-only numbers

Campus

	Data	*Temp + Data*	*Col + Data*	*All*
Actor1	76.70	86.55	76.89	86.55
Actor2[1]	81.00	82.44	81.00	82.54
Actor3	87.27	88.06	87.35	88.14

Shelf

	Data	*Temp + Data*	*Col + Data*	*All*
Actor1	86.75	88.31	86.66	88.34
Actor2	84.52	85.26	84.77	85.26
Actor3	90.68	91.30	90.68	91.30

Table 7.2: The effects of each term on the results for the Campus and Shelf datasets. The first column shows the results using only the data term, the next two columns show the results of combining the data term with one other term, and the final column shows the results of all terms (data, temporal, and collision) together.

reflect the performance of a trivial triangulation of the 2D pose output without any additional reasoning. We note that performance with the data term is competitive with other state-of-the-art methods, and in the case of the Shelf dataset actually out-performs all other methods. This strong performance can be attributed to the highly effective 2D part detection of OpenPose built on deep networks.

The main contribution of the temporal term on metric performance is its ability to correct errors that exist in only a single isolated frame. Figure7.11 illustrates this effect; errors are corrected by information flow from prior and subsequent frames which do not exhibit the same flaws. Additionally, when viewing animated videos of the body-models estimated with and without temporal terms side-by-side, we observe that the temporal result is significantly less jittery. This smoothness is critical to applications such as motion capture for film and game animation, but is not well-captured by the PCP metric.

The collision term has a very minor effect in these results and corrects a small number of minor errors caused by joints being improperly disambiguated. Its effect may be more pronounced in datasets with more complex human motions not captured in the Campus and Shelf datasets. However, we conclude that for simple applications it could likely be removed for a modest improvement in runtime at a negligible accuracy penalty.

[1] For one frame of the video, our person detection step failed to identify Actor 2. For that frame, we supplied a PCP of 0 for all limbs.

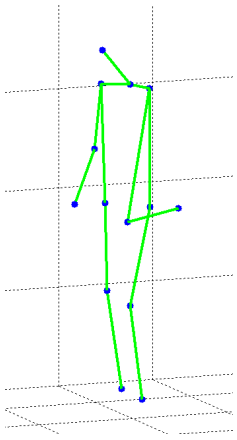


Figure 7.9: Data Only

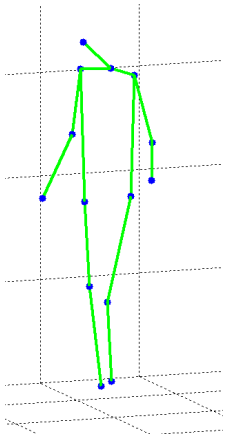


Figure 7.10: All Terms

Figure 7.11: An example of the benefits of temporal smoothing. (a) Using only the data term, it's possible for certain joints to be incorrectly placed on isolated frames. (b) The temporal term can propagate information from nearby frames to prevent these mistakes.

7.5 Conclusion

The multi-view, multi-person 3D human pose estimation pipeline described in this chapter advances the previous state-of-art by combining a modern 2D pose detector based on deep networks with a factor graph optimization to reason over multiple views. Our factor graph places fewer constraints on limb interactions than those proposed in previous multi-view 3D pose estimation papers, yet achieves higher metrics in the Campus and Shelf evaluation datasets. Future work will include adapting the factor graph to allow processing of live video by iterating over short sliding temporal windows.

Acknowledgements

The research was supported in part by the Office of Naval Research (N00014-16-P-2040).

8. SPIN: A HIGH SPEED, HIGH RESOLUTION VISION DATASET FOR TRACKING AND ACTION RECOGNITION IN PING PONG

8.1 Introduction

Sports datasets pose several major challenges for visual tasks which are rarely encountered elsewhere: highly dynamic scenes, viewpoint changes, and large occlusions, to name a few. There has been significant interest in using sports datasets for training vision models in these challenging settings, such as object detection, tracking, and action prediction. Vision models for basketball, baseball, ice-hockey and other sports have been actively explored in the past for predictive models [181, 191], or for generative models [220]. Most of these datasets have low-resolution images or slow frame rates, use single cameras, or focus on offline modeling.

In this chapter we introduce the SPIN dataset, a new high-resolution, high frame rate stereo video dataset for tracking and action recognition in the game of ping pong. The high resolution, and high frame rate, stereo videos allow the possibility of inferring 3D trajectories of ping-pong balls and 2D human pose in the videos at high speeds, in highly dynamic game scenarios. The dataset will be released at REDACTED-FOR-ANONYMITY .

The dataset consists of ping pong play with three main annotation streams that can be used to learn tracking and action recognition models – tracking of the ping pong ball, joint locations of a human pose, and spin of the ball. The training data consists of 53 hours of data, which is about 7.5 million frames of active play, recorded from 2 cameras simultaneously with annotations derived from previously trained models. The testing corpus contains 1 hour of data with the same information, albeit manually annotated via crowd-sourcing with quality control, for better precision.

Furthermore, we provide models for accurate detection, human pose and tracking which establish benchmarks for future use on these challenging tasks. In particular, for the case of high frequency, high-resolution images like those in SPIN, traditional models simply cannot perform

at the rate of image acquisition. To that end, we focus on models that perform inference online, at the speed of data generation, which is roughly 6.6 milliseconds per stereo pair (150 fps), of RGB images of size 1024 x 1280 [1]. Specifically, we introduce several baseline models, including recurrent models and single frame non-recurrent models, to address these challenges. We also observe interesting properties of ping pong ball trajectories, derived from the data. For instance, the dynamics of the movement of the ball gives rise to three distinct clusters, representing different levels of top spin applied to the ball.

Since the SPIN dataset is a high-resolution stereo video dataset, much richer information can be derived from it and it can be used for many other tasks. For example, the dataset can be used for 3-D human pose tracking, and for richer action prediction tasks such as forehand or back-hand shots, derived from the pose information. Thus the dataset is applicable to multi-task learning as well as predictive and generative modeling, and semi-supervised or unsupervised learning. We offer the dataset to the vision research community with the hope of spurring on interesting explorations in these and other related tasks.

8.2 Related Works

8.2.1 Action Recognition

The tasks afforded by the SPIN dataset are most related to action recognition. A large variety of datasets approach action recognition from different angles. This ranges from small, trimmed-video datasets, such as UCF-101 [215] or HMDB [107], to much larger datasets, such as Kinetics [1] or Moments-in-Time [147]. Other action recognition datasets, *e.g.* Charades [209] or ActivityNet [51], take longer videos and make multiple temporal predictions in an untrimmed setting. Still other datasets, such as AVA [84] or ActEV [76] require actions to be localized both spatially and temporally. When viewed as an action recognition dataset, SPIN is distinct from these, as it features highly dynamic scenes and its focus is on predicting the future outcome of a player's current actions, as opposed to the action itself. Though we perform our experiments with SPIN in a trimmed context, the SPIN data we release is open to be explored in other contexts, as well.

A variety of methods have been designed for action recognition [53, 57, 235, 255, 268]. One such method, i3D, fuses two streams of 3D convolutions; one taking in raw RGB frames and

[1] We train our models using Bayer patterns directly to limit the time it takes to load images on to a GPU.

the other taking computed optical flow [1]. Many approaches extend from this architecture, e.g., by attempting to make optical flow computation more efficient [46, 171, 222], or augmenting the architecture with attention based features [255].

The sequential nature of videos creates an opportunity to make use of many techniques originally designed for language modelling. In this vein, many action recognition models make use of self-attention [33], and similar techniques. This includes techniques such as second-order pooling [72, 139], or generalizations of the transformer [71, 241]. Still other techniques make use of recurrent models [200] or other gated structures [145, 263].

8.2.2 Tracking

Visual object tracking models have received significant attention in the community because of the crucial importance of tracking to downstream tasks, such as in-scene understanding and robotics. Tracking problems can be defined in several different ways [60, 141]. For example, tracking by detection relies on detecting a fixed subset of objects and then tracking them through videos sequences [141]. Alternatively, detection-free trackers can deal with arbitrary objects that are highlighted at the start of the videos as being objects of interest [57, 93].

Evaluation for tracking is challenging by itself, and there are many ways if can be formulated. For instance, evaluation methods may consider whether new objects can appear or disappear in the videos and how resets are measured in assessing the performance of the trackers. In our dataset, the subjects used multiple different ping pong balls (sometimes using two at the same time), and hence it is an instance of multi-instance video detection, with appearances of new instances and disappearances of old ones. We perform tracking by detection, building a model for ping pong ball detection in videos and using dynamics models to produce smoother trajectories.

The neural models we develop are video detection models, as opposed to tracking ones. Nevertheless, they can be incorporated into multi-object trackers that follow the more traditional tracking approach of associating detections that are close in a spatio-temporal sense, paired with approaches that allow for the appearance and disappearance of new tracks, similar to [262].

8.2.3 Dynamics of Ping Pong Balls

Dynamics models that describe the motion of ping pong balls through the air, or across table and paddle bounces, are surprisingly intricate. The modelling of the aerodynamic forces

acting on a flying spherical object in a game setting (*e.g.* golf, ping pong, baseball, etc) has been a classical topic among the physics community [19, 143]. Yet the effects of aerodynamic drag and the Magnus effect, which causes ping pong balls to curve, are still being actively researched [186].

The bounces of a ping pong ball are trivial to model when the relative motion between the ball and the surface is simplistic [150], but quickly become complicated when the angle of contact and the ball spin become variables [259]. Unfortunately, previous datasets and experiments are usually performed in controlled setups such as wind tunnels where variables can be isolated, which does not cover all of the variations in ball velocity and spin that would exist in a real game. Moreover, there has been no clear consensus on these modelling efforts [31]. To our knowledge this is the first dataset collected in real game conditions, accompanied with a benchmark for shared use.

8.3 Task Definition

8.3.1 Ball Tracking

The SPIN dataset affords the opportunity to explore many new tasks. The most fundamental of these tasks is ball tracking. For this task, we want to be able to identify the 2D image location of the ping pong ball as it moves through the air. Although we consider this a standalone task, it is key to other tasks, so we learn it also as a supplementary task to others.

8.3.2 Spin Prediction

When a highly skilled ping pong player hits a ball, they can often employ a technique to give the ball a topspin while it flies through the air. We empirically find that this gives rise to three different types of hits: "no spin", "light topspin", and "heavy topspin," all with significant differences in aerial dynamics. In theory, there exists another cluster for "backspin" that causes the ball to fall slower, but in practice the dataset did not include enough examples to parse out this cluster. Although we could train a model to learn which type of dynamics a ball has while it is flying in the air, this task would not be very challenging – it would follow too easily from a model that could effectively track a ball as it moved.

Rather, we address the much more challenging task of *predicting the future spin type of the ball* – that is, the value it will have only once the next player has actually hit it. We emphasize that this task relies on more than simple knowledge of the location of the ball; a successful model

must take into account and understand the movements of the ball and the players in order to guess what the player will do next.

8.3.3 Pose Detection

In addition to tracking the ball location, we explore the task of tracking human pose in 2D in highly dynamic games. Human pose is important to determine what action the human is most likely to do, which is key to predicting future events in the game. For instance, because spin prediction is a human-centric task (we are trying to determine what action the human is most likely to do next), it is likely that providing additional signal about the humans movements and location could improve performance.

8.4 Models

8.4.1 Tracking

To address the challenges of fast online inference we design the tracking models to be lightweight and efficient. Our recurrent architecture is fully-convolutional for more direct access to visual features and is inspired by standard RNN variants such as Gated Recurrent Units (GRUs) [34, 239] and LSTMs [95, 201]. However, we have simplified it to use a single stream of hidden states that are updated at each step. Given a previous hidden state h_{t-1} and an input x, we perform our recurrent step as follows:

$$z_t = \sigma(W_z * h_{t-1}) \tag{8.1}$$

$$c_t = \tanh(W_c * h_{t-1}) \tag{8.2}$$

$$h_t = z_t \odot c_t \tag{8.3}$$

where h_t is both the output and the new hidden state, $*$ represents application of a convolutional filter, σ is a sigmoid function, and $\odot$ represents element-wise multiplication. We chose a GRU-based architecture for its speed, effectiveness and ease of training. Additionally, each step is performed using convolution, instead of fully connected components, which allows spatial information to propagate through the recurrent portions of the network. In our experiments, we observe it performs better than its LSTM counterpart (Section 8.6).

To perform ball tracking, we feed in each 1280×1080 frame at full resolution, quickly

reducing dimensions using a series of convolutions. We perform recurrence on these reduced features, increase resolution using deconvolution layers, and ultimately generate a single channel 160×128 heatmap over the image by performing softmax over all spatial locations (Figure **??**). The resolution of this heatmap is less than that of the original image, so in order to maximize pixel-level accuracy we also learn a patch-based model which extracts the features in a window around the maximal detection and feeds them into a smaller network that uses deconvolutions to upsample the features into a heatmap which more precisely locates the ball within the patch. Figure **??** describes the main architecture for generating the heatmaps, and the supplementary material contains specifics for both the ball detection architecture and the patch model mentioned above. We train each heatmap using standard cross-entropy loss.

8.4.2 Spin Type

To learn which type of spin we expect the ball to have, we create a separate branch of the network for classification results. Specifically, we split off a new branch of our neural network after the recurrent layer (Figure **??**). In order to keep the number of parameters tractable, the resolution of the feature maps is reduced with an average pooling layer. This is followed by convolutional layers and two dense layers, the latter producing the predictions. This model is also trained with cross-entropy loss.

8.4.3 Pose

We perform pose prediction similarly to ball tracking, with structurally the same model, except now producing 30 heatmaps (15 joints for 2 people) at each frame in the same manner as the heatmaps for the ball location. We learn these prediction targets as heatmaps in the final layer, learned with the same cross entropy loss used for ball tracking.

8.5 The SPIN Dataset

The SPIN dataset is recorded in high speed at 150 frames per second with an image resolution of 1024×1280. In order to create variety, the walls behind the players are varied throughout the dataset (Figure 5.4). In particular, the eval dataset does not use any of the backgrounds present in the training set. For all experiments, we use a right-handed 3D coordinate system where x and y are parallel to the ground plane, and z is vertical. The players themselves stand along the y axis,

with one player on the positive side and the other one on the negative side.

8.5.1 2D Tracking

The original tracking data for the SPIN dataset is derived from a sequence of semi-supervised labelling of objects in the datasets. At the first stage, a simple color detector was used, along with an OpenCV [18] mixture of gaussians background detection algorithm to generate a candidate of moving ping pong balls in each frame of the stereo image pairs. Only stereo-pair candidates with low reprojection errors were kept for each frame. The remaining candidates were then extended from frame to frame using Kalman Filters both in 3D coordinates of the stereo pairs and 2D coordinates of the images of the stereo pair. Trajectories found in this way were used to generate training data for the next detection model, which was then used to generate the next round of 3D tracking results, and 2D tracking targets. Each subsequent round of semi-supervised data generation qualitatively improved the 2D detector over video images. A single detector was trained for all cameras. The detectors were trained with strong data augmentation with hue, saturation noise, and image flips (with the corresponding flipping of target locations).

8.5.2 Computing Ball Trajectories

Once we have initial frame-by-frame ball detections from at least 2 cameras, we use stereo matching between the cameras to generate possible 3D coordinates for the ball at each frame. To create more consistent trajectories from these detections, we first attempt to discover two different types of inflection points: ”bounce inflections” where the ball bounces off the table and ”return inflections” where the ball is hit by a player.

To find bounce inflections, we sweep a sliding window over the detections, taking 6 at a time and fitting each set to a polynomial that is first order in the *x*-*y* plane (parallel to the ground) and second order in the vertical *z* plane. From this polynomial we derive the velocity of the ball at the end of the window. We then compute the approximate velocity of the ball at the next 6 points, and record all of the locations where the signs of the vertical velocities differ. Finally, we perform non-maxima suppression to filter out any bounces that do not occur on the *x*-*y* bounds of the table, and mark all remaining inflections as bounce inflections.

When computing bounce inflections, we also keep track of the y velocity. Thus, though we compute return inflections using the same velocity-checking algorithm we used for bounce

inflections (albeit along the y axis instead of the z), we enforce that they can only occur between 2 bounce inflections that have opposite y velocities. This helps by drastically reducing false positives. This also causes hits that go out of bounds not to be detected, which we did not consider problematic for our purposes, as we wanted to focus our analysis on successful hits. If this is undesirable for another application, the procedure can easily be modified to make an exception for balls that go out of bounds by simply treating the end of the trajectory as a possible inflection point.

Finally, given a set of inflections, we smooth our final estimates using a form of statistical bootstrapping wherein we pass over the area between inflections with a sliding window of 20 points. In each window, we take 25 samples of 6 points, fit a polynomial to them, and average out the results to achieve our final estimate.

These trajectories provide a rich source of data. For instance, in addition to spin prediction, one could use the trajectory information to predict where on the table a ball will land after it is hit. The models we explore in this paper focus on predicting what type of hit the player will perform, but as the trajectories are in 3D coordinates they could also be used to learn other interesting tasks.

8.5.3 Measuring Ball Spin

One aspect of player actions that we wish to capture is the amount of spin that is placed on the ball when the player hits it. In order to do this, we show that it is possible to use information gleaned from ball trajectories to cluster different types of hits, as seen in Fig. **??**.

Given a ball trajectory immediately after a player hits the ball, we compute the change in x-y-velocity immediately before and after the ball bounces on the table, taken by measuring the movement 10 frames before and 10 frames after impact. Additionally, we compute the downward acceleration of the ball as it moves through the air by fitting a second-degree polynomial to the z coordinate of the ball through time. In other words, we approximate the z movement with a polynomial $ax^2 + bx + c$. Figure **??** plots $2a$ (the downward acceleration) on the z axis. To allow the trajectory to better approximate a parabola, we remove the first and last 5 frames of the trajectory.

We observe three clusters, as seen in Figure **??**. When we divide the hits by "professional" and "non-professional" players, we notice that balls hit by non-professionals are extremely consistent - they fall at a steady $9.8\frac{m}{s^2}$ and their hits have a very consistent change in velocity. When

	Top Cluster	**Middle Cluster**	**Bottom Cluster**	**No Cluster**
Train	10173	13849	7317	8456
Eval	413	832	228	412

Table 8.1: The number of hits in each cluster within the "SPIN-OR" data split described in Section 8.5.5.

professionals play, we see two new clusters emerge. Through observation, we determined that these clusters correspond to topspin placed on the ball. The extra spin causes the ball to accelerate when it hits the table, causing a sharper change in velocity when hitting the table. Furthermore, in light of the Magnus effect, balls with topspin actually accelerate towards the table faster.

From observation and discussions with the players, we determined that the two topspin clusters correspond to two distinct scenarios - the bottom cluster occurs when a player offensively hits the ball hard, and the middle cluster occurs when a player defends against an opponent's offensive hit. In the case of the middle cluster, the ball is still returned with a certain degree of topspin, but there is considerably less and thus the downward acceleration and coefficient of restitution are both reduced.

Using these plots, we manually select the approximate centroids of these clusters located at $[-0.64, -9.5], [-0.9, -17.5]$ and $[0.016, -24]$. When we process data for training, we assign each data point into a cluster based on which centroid it is closest to, and use this to assign a "spin type" to the hit. Table 8.1 shows the number of hits assigned to each of the three clusters. As a final note, we once again emphasize that this is annotation does *not* label which spin type the ball currently posses, but rather which spin type it will possess *after* it has been hit by the player.

8.5.4 Tracking Data

Once we have computed and smoothed the trajectories as described in Section 8.5.2, we create a training dataset out of the final trajectories. To do this, we use a stride of 15 and sample sets of 30 consecutive frames from the trajectory, meaning each frame of the trajectory is sampled twice. The ball detection is projected back into the image from its 3D coordinates, to be used as ground truth during training. Using this method, we generate 251,922 individual training examples over 212 videos (approximately 53 hours of game play) and 14,210 test examples over 13 videos. See Figure 5.4 for sample data. We refer to this dataset as "SPIN-ALL".

Human Annotation

While our automatically generated ground truth is reasonably high quality, it is far from perfect. In order to measure the quality of our trained models, we generate a set of human-annotated samples of ball trajectories, which we use for evaluation. For human annotation we chose to use a cloud-based crowd compute service.

The task was to annotate each track of a ball that was in play. A ball enters play when it is tossed up for a serve and exits play when it is no longer playable by a human player, e.g. on the floor or on the table. Each in-play ball received its own track. The tracks were comprised of sequential image-frame (x, y) center positions of the ball in each frame it was observed. Each center position is optionally annotated with a "Bounce" or "Hit" tag, indicating whether the ball bounced on the table or was hit by a player. Before the annotation, the questions were pre-annotated by a ball tracking model, to minimize the annotation time.

For annotation, we selected 11 three minute time segments of play that were collected with different backgrounds from training and with as much player diversity as possible. Both left and right camera images from the same time stamps were sent for annotation in order to allow association between cameras and stereo depth inference later on. The three minute clips of both cameras were further split into questions containing 200 frames each from a single camera. The 200-frames annotation was the unit of work for the human annotators. In order to associate tracks between questions, each question had a 10 frame overlap both with its temporal predecessor and successor to make it easier to associate tracks between consecutive questions.

8.5.5 Spin Type Data

To predict spin type, we create a separate dataset split we refer to as "SPIN-OR" (for "Only Return"). First, we limit our examples to those in which the ball is moving from left to right at the beginning of the example. Since our focus is primarily on the human who is preparing to hit the ball, only considering examples where the ball moves in a single direction allows the network to focus on a single individual, instead of needing to switch between individuals based on which direction the ball is moving. In order to avoid limiting the data unnecessarily, when the ball is moving right to left we flip the frame horizontally and reorder the human joints as necessary to make sure the example remains valid. Although one of our recorded players is already ambidextrous, using this flipped data has the added advantage of making the system robust to left-

Architecture	AUC @ 2	AUC @ 5
Gating (GRU)	81.5	96.7
LSTM	80.6	93.0
Single Frame	88.5	96.0

Table 8.2: Ball Tracking accuracy showing the AUC at distance 2 pixels and 5 pixels.

handed and right-handed play.

Next, we modify which frames we use. Instead of using all frames of play, we limit our examples to 25 frames surrounding the moment the ball is hit - 20 before the player hits the ball and 5 afterwards. We do this because we found empirically that using too many frames before the hit did not provide any additional useful signal - when the ball is on the opposite side of the table, it's too hard for the system to predict what the player will do. With this modified data generation, we create a dataset of 39,795 training samples and 1885 eval samples over the same 212 training and 13 eval videos used in Section 8.5.4.

8.5.6 2D Human Pose Data

We also supplement all of our data in both splits with annotations for human pose. To generate these annotations, we pass our data through the pose detector described in [165]. This produces for each individual 15 detections per frame corresponding to 15 different human joint locations.

In most cases, there are only two detections, corresponding to the two players in the game, however on rare occasions a bystander enters the frame and is detected. In these cases, we limit our annotations to only the two highest confidence detections. We further differentiate between the left and right player using the locations of their skeleton's detected bounding boxes, and add the final joint annotations to every frame in the data.

8.6 Experiments

8.6.1 Implementation

For all experiments in this section we train on the video snippets described in (Section 8.5.4), with 25 continuous frames per sample using a batch size of 8. We train for over 100K steps starting with a learning rate of 0.002 for the first thousand steps, then reducing to 0.00025 for the rest of training. We also change the optimizer over the course of training, starting with stochastic

Experiment	Spin	Spin + Pose	Spin + Ball	Spin + Pose + Ball
Dataset	OR	OR	OR+ALL	OR+ALL
Spin Accuracy	61.6	68.2	59.4	72.8

Table 8.3: Performance of our gated architecture on the spin classification task. Spin accuracy is shown under 4 different multi-task learning situations.

gradient descent for the first 5000 steps and switching to Adam [119] afterwards for the remainder of training. For the experiments below, the "SPIN-OR" split, described in Section 8.5.5, consists of samples that containing 20 frames leading up to the hit, and 5 frames after the hit occurs. We use all possible hits by flipping the image when the ball goes right to left. Similarly, when learning on the "SPIN-ALL" dataset split, we randomly flip images to provide additional signal. Finally, in the experiments we also use an "OR + ALL" split, wherein we combine the "SPIN-OR" and "SPIN-ALL" dataset splits by sampling in equal weights from each.

Experiment	AUC@16	AUC@40	Dataset
Pose	87.4	95.3	ALL
Pose + Ball	83.7	92.6	ALL
Pose + Spin	77.2	91.5	OR
Pose + Spin	83.0	92.3	OR + ALL

Table 8.4: Performance of our gated architecture on the 2D human pose detection task. The Pose task performance is paired with the ball detection and spin classification tasks. The "Pose + Spin" task is trained twice, once with the "OR" split centered around the moment the player hits the ball, and once with all examples.

8.6.2 Ball Detection

We begin with an analysis of the ball detection task. We evaluate the performance of ball detection using the main gated architecture presented in Section 8.4.1, LSTMs and a single frame prediction model. Table 8.2 explores these results.

For these experiments, we measure our performance by computing the area under curve (AUC) measurement for precision-recall at 2 and 5 pixels from the ground truth. Each of these curves was computed by sweeping through the log probability of detections from the model.

For comparison, we also experiment with a convolutional LSTM counterpart. The LSTM architecture is similar to the architecture of [95], with the notable exception that all fully connected layers are replaced with convolutional layers.

We find that the gated architecture performs better than alternative architectures (Table 8.2).

The LSTM architecture performs worse for the task than with gating, which may be attributed simply to the difficulty of training the LSTM architecture. Empirically, we observe it is much easier to find viable training parameters for the proposed gating architecture.

Single Frame

We also experiment in Table 8.2 with a single frame setting, replacing the recurrent connections of the previous experiment and replacing the entire recurrent unit with a single convolutional layer. Without a persistent state or gating, this system should be unable to learn dependencies or ball dynamics over time. However, We find that this model performs well at 2 pixel distance from ground-truth, and not as well at 5 pixel distance. We hypothesize that this is partly attributable to the easier training of the non-recurrent model, and that at increased tolerances, the recurrence is able to rule out false positives that the non-recurrent model finds.

8.6.3 Spin Classification

In this section we report the results on the spin classification task. We perform four experiments towards this end, shown in Table 8.3, where the first column shows the pose task learned on its own, the others show multi-task learning when pose detection is paired with other tasks. For all of these experiments, we use the standard gated architecture discussed in Section 8.4.1. As stated, earlier, all tasks use standard cross-entropy, either over a one-hot vector or a heat map, but performing multi-task learning we multiply the spin classification loss by 3.

Our first experiment with spin classification measures results when spin is learned in isolation. The second experiment, wherein human pose is learned as an auxiliary task, shows significant improvements in spin performance, likely due to the additional understanding of the humans playing.

The second set of experiments are conducted on "OR+ALL" dataset split, where we evaluate performance of spin detection when trained jointly with just ball detection or when trained jointly with both ball detection and pose. Here too, we observe significant jump in performance after including human pose as a multi-task output. For the experiments involving both spin and ball tracking, we combine the "SPIN-OR" and "SPIN-ALL" dataset splits, while only propagating losses for spin when the data comes from the "SPIN-OR" dataset. We find here that ball detection does not actually provide any advantage to spin classification. In all likelihood, this is because the

signal understanding from the player is far more informative than any signal from the ball.

8.6.4 Human Pose

Here we evaluate the human pose detection task. Results in Table 8.4 show AUC at 16 and 40 pixels, respectively, using the same top-1 and top-5 precision metrics used previously in Table 8.2. We observe that pose detection performs the best on its own, and adding, for example, ball detection is not helpful. This is not surprising as human pose contains more information and the model may not necessarily need the ball position.

The second set of experiments test the joint training of pose and spin, and demonstrate that additional data with more variety in human pose ("OR + ALL") is definitely more helpful than using the more restrictive samples from "OR" only.

8.7 Conclusion

We have presented the new SPIN dataset, a high-resolution, high frame rate stereo video dataset for the game of ping pong. SPIN is a rich computer vision dataset presenting several problems for computer vision researchers, of which we explore three: pose tracking, spin type prediction, and human pose detection. We present strong baselines for these challenging tasks in real game play settings.

8.8 Supplemental Material

8.8.1 Model architectures

We here visualize the model architectures used in detail. Figure **??** shows the main tracking model, whereas Figure **??** shows the patch model.

8.8.2 Human Annotation Details

The task of human annotation was to annotate each track of a ball that was in play. We here provide more details on the human annotation. A ball enters play when it is tossed up for a serve and exits play when it is no longer playable by a human player, e.g. on the floor or on the table. Each in-play ball received its own track. The tracks are comprised of sequential image-frame (x, y) center positions of the ball in each frame it was observed. Each center position is optionally

annotated with a ”Bounce” or ”Hit” tag, indicating whether the ball bounced on the table or was hit by a player.

We selected 11 three-minute time segments of play that were collected with different backgrounds from training and with as much player diversity as possible. Both left and right camera images from the same time stamps were sent for annotation in order to allow association between cameras and stereo depth inference later on. The three-minute clips of both cameras were further split into questions containing 200 frames each from a single camera. The 200-frames annotation was the unit of work for the human annotators. In order to associate tracks between questions, each question had a 10 frame overlap both with its temporal predecessor and successor to make it easier to associate tracks between consecutive questions. Before the annotation, the questions were pre-annotated by a ball tracking model. These annotations were used to minimize the time the annotators had to spend annotating each question.

8.8.3 Initial tracking model details

In order to avoid expansive annotations for large volumes of data, we build a self-supervised labeling pipeline which starts from an initial tracking model, which is then incrementally improved by the learning based models, presented in the chapter, and increasingly better quality annotations. We here describe the initial model for completeness. Alternative ones can also be used.

In the first stage, a simple color-based detector was used, along with an OpenCV [18] Mixture of Gaussians background detection algorithm to generate a candidate of moving ping pong balls in each frame of the stereo image pairs. Only stereo-pair candidates with low re-projection errors were kept for each frame. The remaining candidates were then extended from frame to frame using Kalman Filters both in 3D coordinates of the stereo pairs and 2D coordinates of the images of the stereo pair.

Subsequently, trajectories found in this way were used to generate training data for the next detection model, which was then used to generate the next round of 3D tracking results, and 2D tracking targets. Each subsequent round of semi-supervised data generation qualitatively improved the 2D detector over video images. A single detector was trained for all cameras. The detectors were trained with strong data augmentation with hue, saturation noise, and image flips (with the corresponding flipping of target locations).

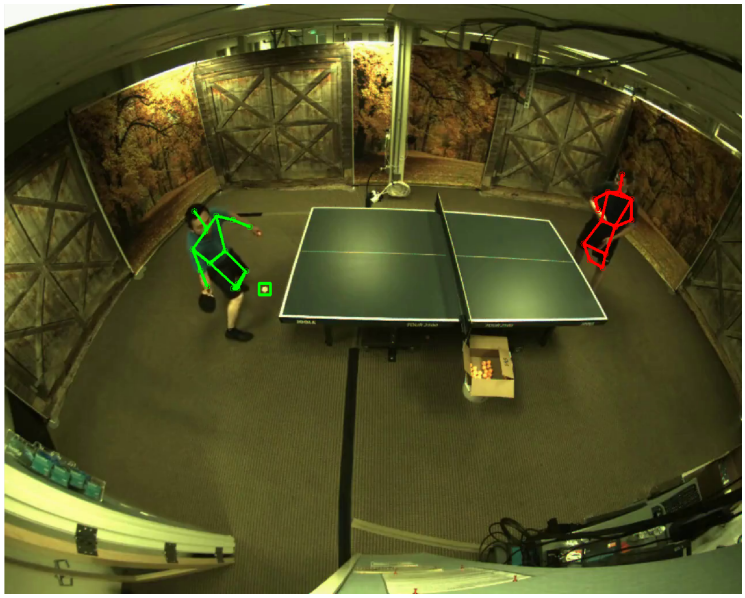


Figure 8.1: Ground truth

Figure 8.2: Detections

Figure 8.3: We present a dataset for analyzing ball dynamics and human actions in the game of ping pong. Top: Ground truth annotations for a frame from the SPIN dataset. SPIN contains annotations for the ball location and human pose at every frame, as well as information about future frames, such as how much the next hit will put on the ball and where the ball will bounce after it is hit. Bottom: A heatmap for ball location, predicted by our method.

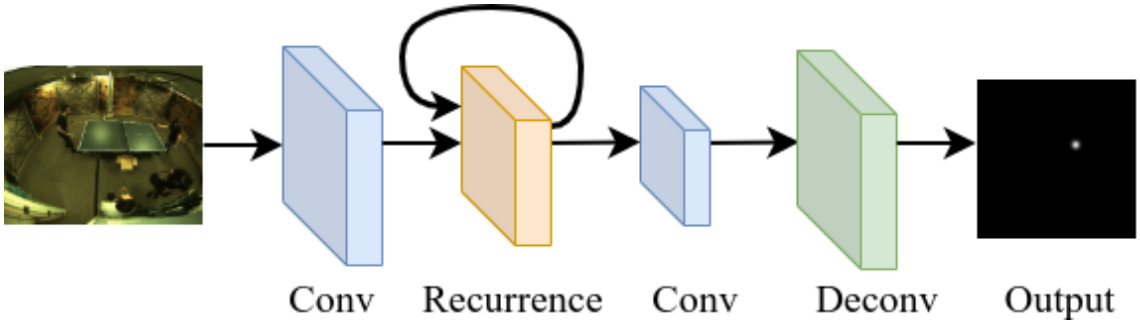


Figure 8.4: Tracking architecture

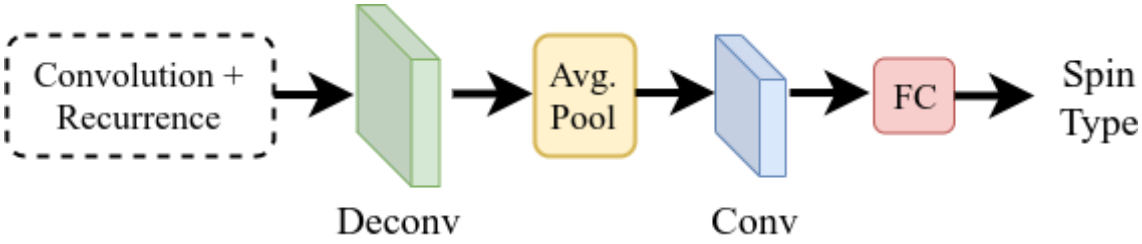


Figure 8.5: Spin classification architecture

Figure 8.6: A high-level view of the architectures we use. (a) The architecture we use for tracking uses convolution blocks to reduce resolution, a recurrent layer, and deconvolution layers to generate heatmaps for tracking. (b) When learning spin type, we branch off after the deconvolution before the last layer, sequentially performing average pooling, convolution, and flattening. We finish with a fully connected layer to predict the final spin type.

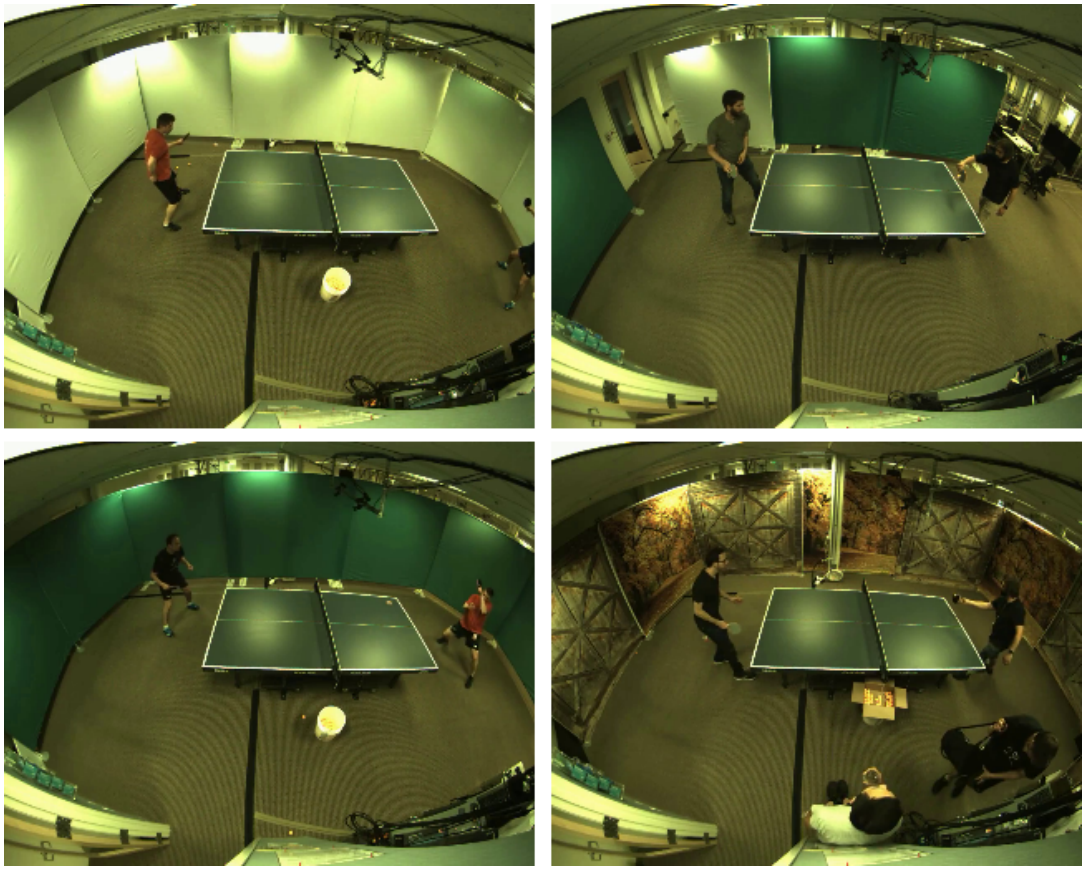


Figure 8.7: Sample frames from the SPIN dataset.

[b]0.235

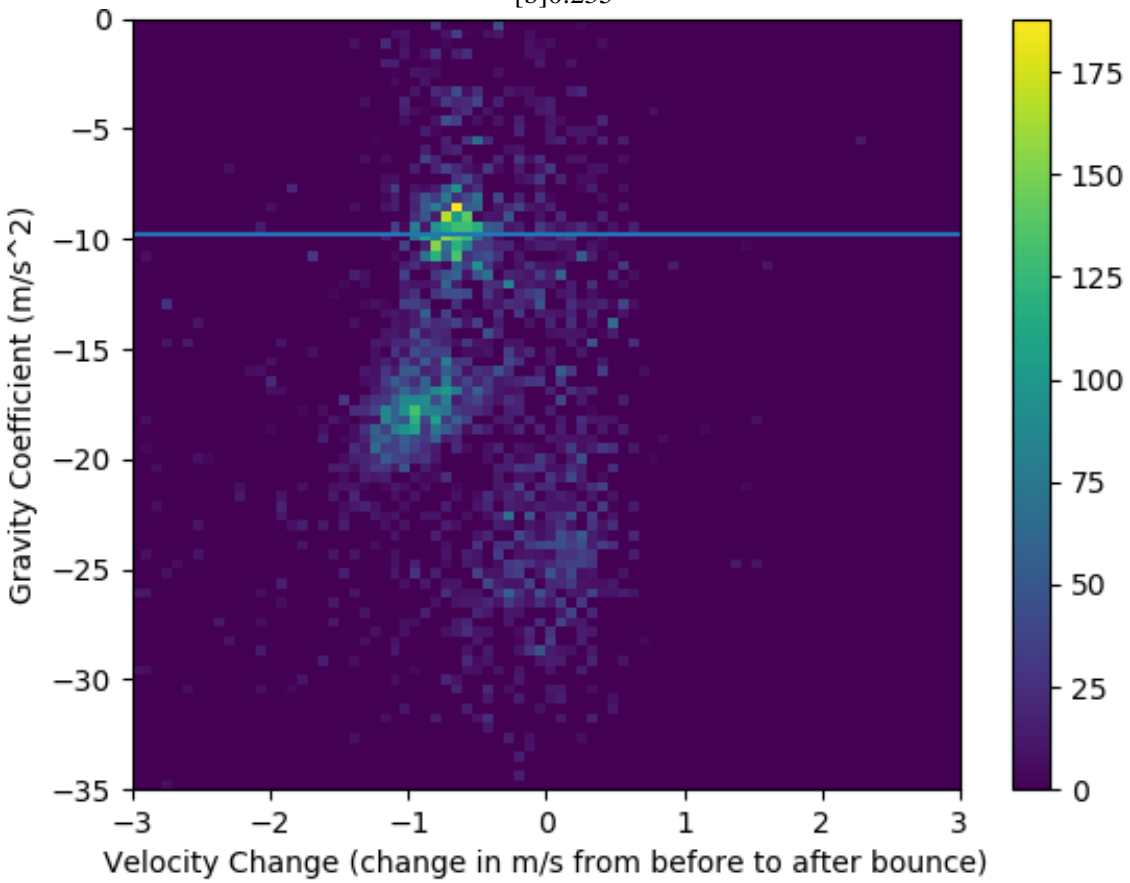


Figure 8.8: All players

[b]0.235

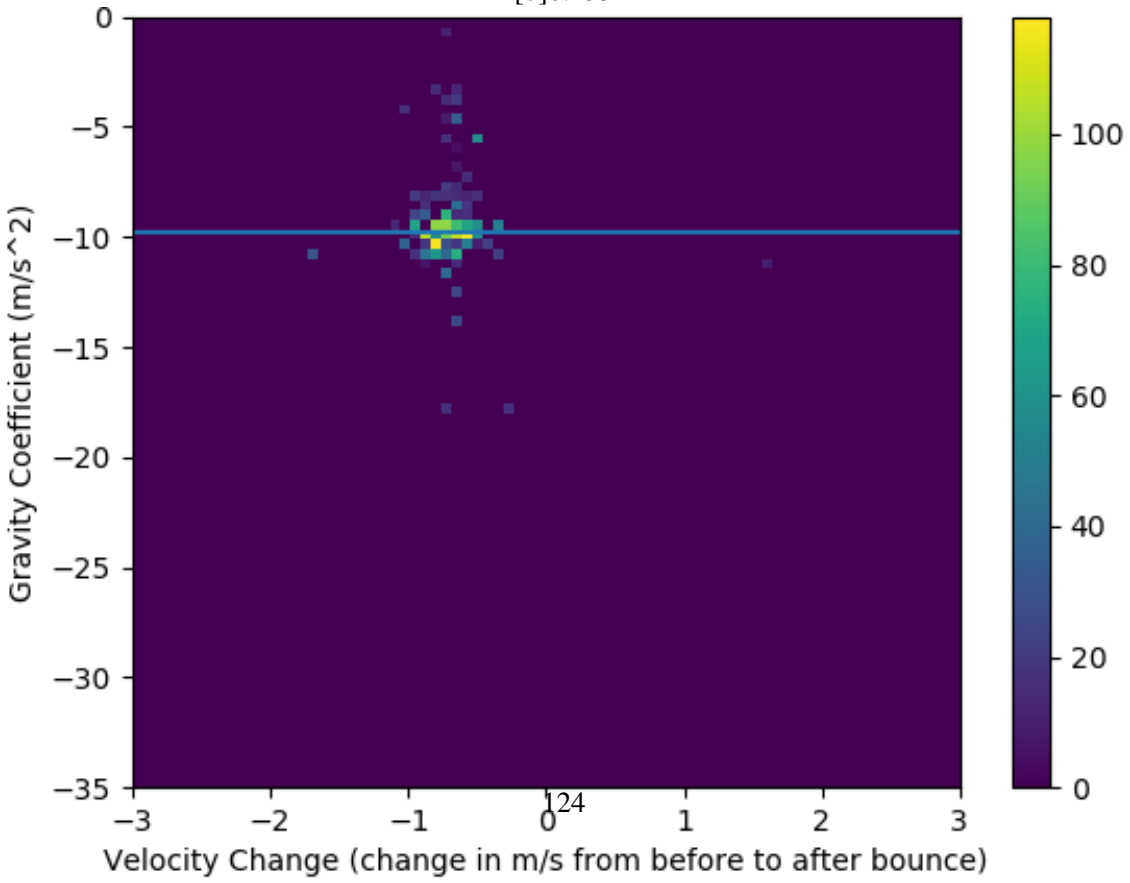


Figure 8.9: Amateurs

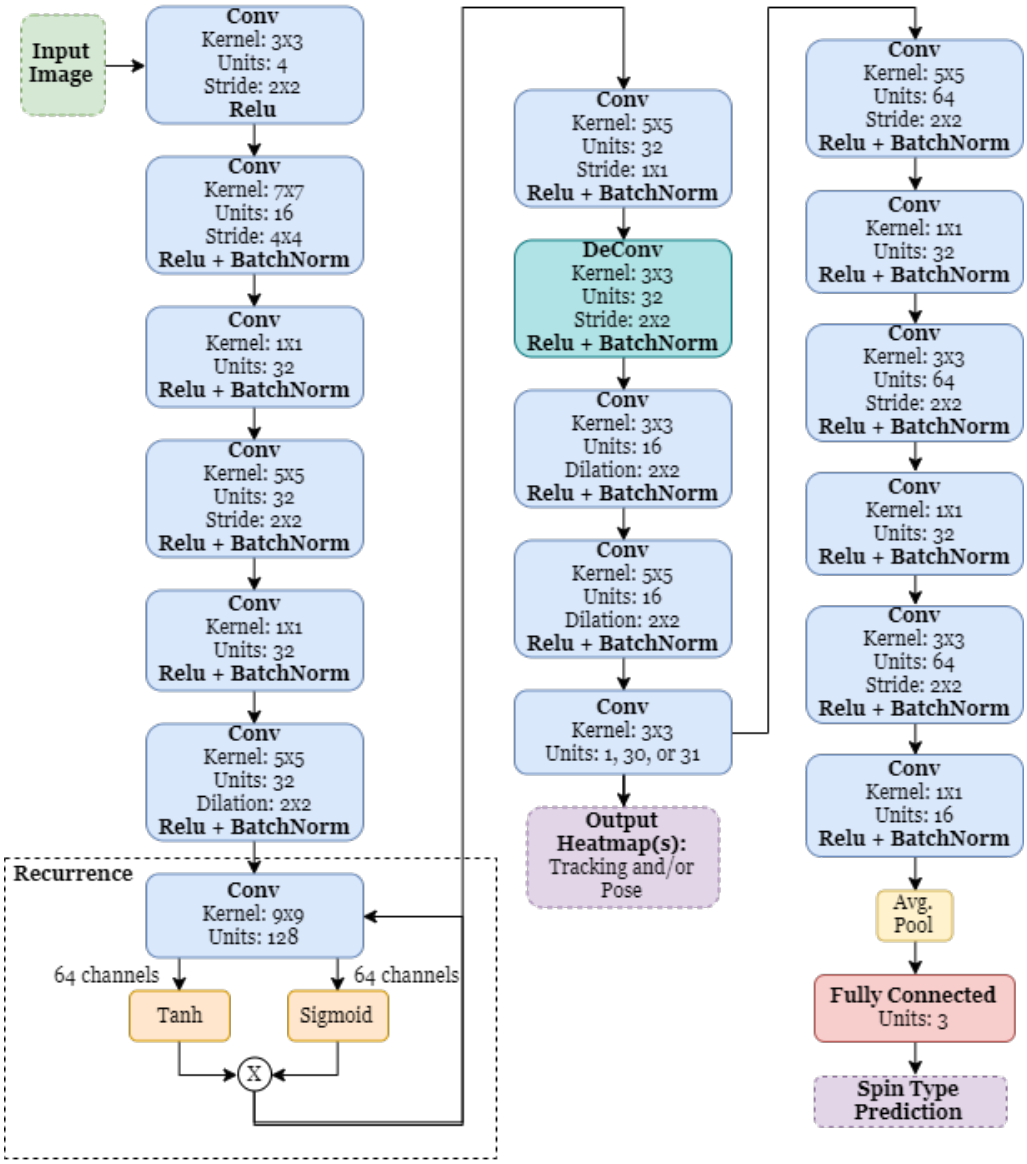


Figure 8.12: The complete architecture of our system, except for the patch prediction model. "**Deconv**" refers to the standard transpose convolution operation, "Dilation" refers to dilated convolutions [270], and "**BatchNorm**" refers to Batch normalization [105].

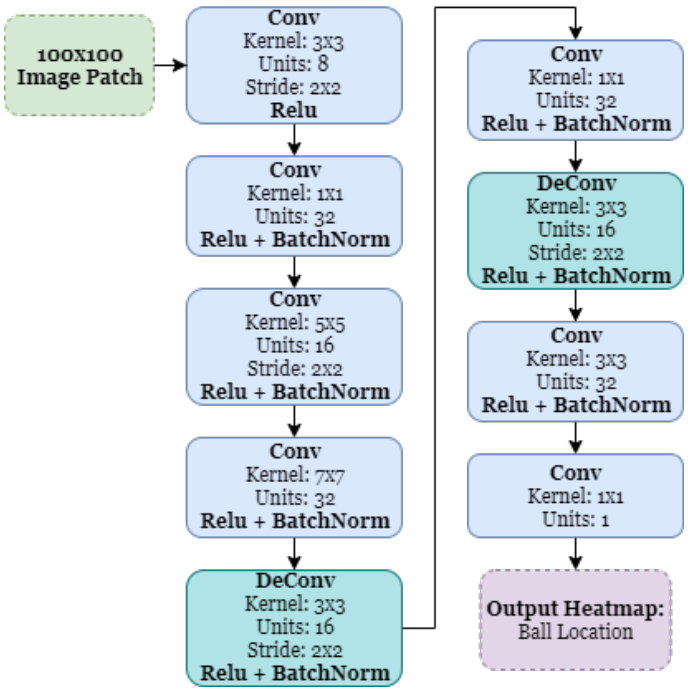


Figure 8.13: The patch prediction model. When performing tracking, after we find the maximum activation in the ball location heatmap, we crop out a 100×100 patch of the original image centered at that location, and feed it through the above network, creating a new heatmap which we then use for a more precise ball location. "**Deconv**" refers to the standard transpose convolution operation, "Dilation" refers to dilated convolutions [270], and "**BatchNorm**" refers to Batch normalization [105].

9. CONCLUSION

9.1 Summary

In this dissertation, we discussed a variety of techniques for analyzing Computer Vision problems involving videos of humans. We began this with proposed architectures for human activity detection in footage where the actions themselves are sparse, and for this we used a two-stage process that created high-recall proposals in the first stage and refined them in the second. We followed this up with few-shot learning problems that tried to perform the same task as the above human action detection algorithms, but did so with only very limited training examples per class. We also explored a completely different type of sequential processing in the form of optical character recognition. In this case we did not have access to labeled training data, and so we had to use synthetic data and labeled data from a different task in order to achieve high-quality performance.

We followed this up with a method for detecting if a video of an individual's face is a forgery that operated by breaking down the face of the individual into explainable regions, and used this analysis to develop a better understanding of how forgery datasets were visually structured.

We then focused on the problem of multi-camera, multi-person pose detection, where we needed to use graphical methods to optimize over many possible human joint locations in order to capture the pose of multiple actors within a scene. Finally, we developed a novel recurrent gated CNN architecture, and evaluated it on a new dataset for activity recognition, tracking, and pose detection in the game of Ping Pong.

9.2 Future Work

9.2.1 Improving Proposal Generation for Spatio-Temporal Activity Detection

One limitation of the activity detection system discussed in Chapter 2 is the large number of proposals generated. Future work should seek to reduce the number of proposals generated by the

system, while retaining the high recall that makes the system successful. One way to go about this is to develop a lightweight neural network that can quickly filter out proposals that are unlikely to be meaningful detections.

Towards this end, we propose a student-teacher framework for proposal filtering. At a high level, this would train a small network which mimics the existing TRI-3D architecture's outputs, but using considerably less processing power. The idea is that this network may be considerably less accurate, but as long as it can reliably determine when an action might be happening - even if it is rarely correct about which one, or has many false positives - it can still be used as a pre-filtering step to dramatically reduce the number of proposals fed into the activity detection system.

To give an example of how such a system might be trained, let f_t be the TRI-3D feature extractor and action classifier discussed in Chapter 2, which we will call the teacher network, and let f_s be a significantly smaller network that we will call the student. f_t is computationally heavy - it requires 64 frames of input, all at 224×224 resolution, uses 3D convolutions, and has many parameters. There are therefore many ways that f_s can be smaller and faster: taking only a few frames, having fewer layers, and operating at much lower resolution, among others. In addition to learning the ground truth annotations, f_s can be trained to mimic the output logits for f_t on all inputs, for instance using knowledge distillation [94]. This has been empirically shown to increase the amount of knowledge transferred beyond merely learning the ground truth [94].

Our hope is that the savings from using such a system as a filter for proposals would outweigh the cost of passing every proposal through a new network. If f_t is small enough and enough proposals are removed, we do not believe this will be difficult to accomplish.

9.2.2 Facial Forgery Detection Exploiting Paired Structure

While many labeled datasets for many domains simply contain examples from different classes, facial forgery datasets such as FaceForensics++ [188] or DFDC [47] go one step further by containing paired examples. Specifically, for any given video, these datasets almost always contain labeled examples from real and fake classes that are identical in every respect except that the face is different.

In other words, it is possible to organize the training datasets for facial forgery detection into pairs (x_r, x_f) where the difference between the real image x_r and the fake image x_f are the only contributing factors to the label that is ultimately assigned. It seems plausible, therefore,

that one could exploit this structure during training in order to produce more robust facial forgery detection systems, even when that pairing will not be available during inference time.

9.2.3 Forgery Detection for Mismatched Captions and Images

In addition to the facial forgery detections discussed in the previous section and Chapter 6, many other important forms of forgeries exists. One such area of concern is mismatched captions and images appearing in the wild, for instance in a falsified news article.

In order to address this issue, it is important to employ a model that can simultaneously process images and text, finding critical correlations between the two modalities. Recently, CLIP [178] was introduced as a powerful neural architecture for making exactly these correlations. CLIP is trained on a large set of images labeled with corresponding captions that were only lightly curated by humans, and learns using contrastive learning by creating separate feature embeddings for the image and caption. The final objective is a cross-entropy loss that is computed by scaling the cosine distance between the features and uses the results as logits.

CLIP thus shows a lot of promise as a viable method for performing forgery detection in this context. Indeed, there already exist datasets, such as NewsCLIPpings [140], which use CLIP to produce hard samples for exactly these problems. Other datasets for this problem can also be generated, for instance by using the existing VisualNews dataset [137] as a base.

Since these falsified image-caption pairs often appear in fake news articles, one possible approach for improving CLIP for forgery detection is to include the contents of the news article as additional context. On the surface, this would mean adding a large amount of additional information for the network to sift through, so in order to make this practical for a network, we propose an additional attention scheme that limits the network to only include relevant context from the article.

More specifically, suppose f_i, f_c, and $\mathbf{f}_a = [f_{a,1}, \ldots, f_{a,n}]$ are features extracted from CLIP for the image, caption, and article respectively, noting that there are multiple article features because the article is too long to be encoded in a single pass of the CLIP text encoder. Then there several options for ways to aggregate the features $\mathbf{f}_a$ and combine them with the features for images and captions. One could, for instance, use f_c as a query against the values in $\mathbf{f}_a$ and aggregate the results to modify the caption features:

$$\hat{f}_a = \sum_{i=1}^{n} (f_c^T f_{a,i}) f_{a,i} \hat{f}_c = f_c + \hat{f}_a \tag{9.1}$$

This would allow the most relevant article features to modify the caption features. Alternatively, the two-way contrastive loss of CLIP could be extended to a three-way loss, perhaps by taking all pairwise losses between f_i, f_c, and some aggregated form of $\mathbf{f}_a$. Many other possible methods of incorporating article information exist, and this is a potentially promising direction for using article information as an additional input in the problem of image-caption misalignment detection.

Bibliography

[1] Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[2] Sathyanarayanan Aakur, Daniel Sawyer, and Sudeep Sarkar. Fine-grained action detection in untrimmed surveillance videos. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 38–40. IEEE, 2019.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7, 2018.

[5] G. Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *J. Mach. Learn. Res.*, 15:3563–3593, 2014.

[6] Humam Alwassel, Fabian Caba Heilbron, and Bernard Ghanem. Action search: Spotting actions in videos and its application to temporal action localization. In *The European Conference on Computer Vision (ECCV)*, 2018.

[7] I. Amerini, L. Galteri, R. Caldelli, and A. Del Bimbo. Deepfake video detection through optical flow based cnn. In *2019 IEEE/CVF International Conference on Computer Vision*

Workshop (ICCVW), pages 1205–1207, 2019.

[8] Sikandar Amin, Mykhaylo Andriluka, Marcus Rohrbach, and Bernt Schiele. Multi-view Pictorial Structures for 3D Human Pose Estimation. In *Procedings of the British Machine Vision Conference 2013*, 2013.

[9] M. Andriluka, S. Roth, and B. Schiele. Pictorial structures revisited: People detection and articulated pose estimation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1014–1021, June 2009.

[10] George Awad, Asad Gov, Asad Butt, Keith Curtis, Yooyoung Lee, yooyoung@nist Gov, Jonathan Fiscus, David Joy, Andrew Delgado, Alan Smeaton, Yvette Graham, Wessel Kraaij, Georges Quénot, João Magalhães, and Saverio Blasi. Trecvid 2018: Benchmarking video activity detection, video captioning and matching, video storytelling linking and video search. 04 2019.

[11] Belhassen Bayar and Matthew C. Stamm. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, IHamp;MMSec '16, page 5–10, New York, NY, USA, 2016. Association for Computing Machinery.

[12] V. Belagiannis, S. Amin, M. Andriluka, B. Schiele, N. Navab, and S. Ilic. 3d pictorial structures for multiple human pose estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1669–1676, June 2014.

[13] Vasileios Belagiannis, Sikandar Amin, Mykhaylo Andriluka, Bernt Schiele, Nassir Navab, and Slobodan Ilic. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 1 3D Pictorial Structures Revisited: Multiple Human Pose Estimation.

[14] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 2010.

[15] Luca Bertinetto, João F. Henriques, P. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. *ArXiv*, abs/1805.08136, 2019.

[16] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 95–104, July 2017.

[17] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 343–351. Curran Associates, Inc., 2016.

[18] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[19] Lyman J Briggs. Effect of spin and speed on the lateral deflection (curve) of a baseball; and the magnus effect for smooth spheres. *American Journal of Physics*, 27(8):589–596, 1959.

[20] Denny Britz, Quoc Le, and Reid Pryzant. Effective domain mixing for neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 118–126. Association for Computational Linguistics, 2017.

[21] S Buch, V Escorcia, B Ghanem, L Fei-Fei, and JC Niebles. End-to-end, single-stream temporal action detection in untrimmed videos. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2017.

[22] Shyamal Buch, Victor Escorcia, Chuanqi Shen, Bernard Ghanem, and Juan Carlos Niebles. Sst: Single-stream temporal action proposals. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[23] Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. Fast temporal activity proposals for efficient detection of human actions in untrimmed videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[24] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. nov 2016.

[25] J. Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[26] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[27] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern*

Recognition, pages 6299–6308, 2017.

[28] Lucy Chai, David Bau, Ser-Nam Lim, and Phillip Isola. What makes fake images detectable? Understanding properties that generalize. aug 2020.

[29] Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A Ross, Jia Deng, and Rahul Sukthankar. Rethinking the faster r-cnn architecture for temporal action localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[30] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

[31] Xiaopeng Chen, Ye Tian, Qiang Huang, Weimin Zhang, and Zhangguo Yu. Dynamic model based ball trajectory prediction for a robot ping-pong player. In *2010 IEEE International Conference on Robotics and Biomimetics*, pages 603–608. IEEE, 2010.

[32] Yuhua Chen, Wen Li, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Domain adaptive faster r-cnn for object detection in the wild. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[33] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. A2-nets: Double attention networks. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 350–359, USA, 2018. Curran Associates Inc.

[34] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *ArXiv*, abs/1406.1078, 2014.

[35] Jinwoo Choi. Awesome action recognition. `https://github.com/jinwchoi/awesome-action-recognition`.

[36] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[37] Chenhui Chu and Rui Wang. A survey of domain adaptation for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages

1304–1319. Association for Computational Linguistics, 2018.

[38] Xiao Chu, Wei Yang, Wanli Ouyang, Cheng Ma, Alan L. Yuille, and Xiaogang Wang. Multi-Context Attention for Human Pose Estimation. feb 2017.

[39] Kellie Corona, Katie Osterdahl, Roderic Collins, and Anthony Hoogs. Meva: A large-scale multiview, multimodal video dataset for activity detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1060–1068, January 2021.

[40] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection. *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*, 2017.

[41] Xiyang Dai, Bharat Singh, Guyue Zhang, Larry S Davis, and Yan Qiu Chen. Temporal context network for activity localization in videos. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[42] Hal Daume III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263. Association for Computational Linguistics, 2007.

[43] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[44] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[45] Guneet S. Dhillon, P. Chaudhari, A. Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *ArXiv*, abs/1909.02729, 2020.

[46] Ali Diba, Vivek Sharma, Luc Van Gool, and Rainer Stiefelhagen. Dynamonet: Dynamic action and motion network. *ArXiv*, abs/1904.11407, 2019.

[47] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge dataset, 2020.

[48] A. Elhayek, E. de Aguiar, A. Jain, J. Thompson, L. Pishchulin, M. Andriluka, C. Bregler,

B. Schiele, and C. Theobalt. Marconi—convnet-based marker-less motion capture in outdoor and indoor scenes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(3):501–514, March 2017.

[49] A. Elhayek, E. de Aguiar, A. Jain, J. Tompson, L. Pishchulin, M. Andriluka, C. Bregler, B. Schiele, and C. Theobalt. Efficient convnet-based marker-less motion capture in general scenes with a low number of cameras. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3810–3818, June 2015.

[50] Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. Daps: Deep action proposals for action understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[51] Bernard Ghanem Fabian Caba Heilbron, Victor Escorcia and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[52] Haoshu Fang, Shuqin Xie, and Cewu Lu. RMPE: regional multi-person pose estimation. *CoRR*, abs/1612.00137, 2016.

[53] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. *ArXiv*, abs/1812.03982, 2018.

[54] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In *Advances in neural information processing systems*, pages 3468–3476, 2016.

[55] Christoph Feichtenhofer, Axel Pinz, and Richard P Wildes. Spatiotemporal multiplier networks for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4768–4777, 2017.

[56] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1933–1941, 2016.

[57] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3038–3046, 2017.

[58] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *Int. J. Comput. Vision*, 61(1):55–79, January 2005.

[59] V. Ferrari, M. Marin-Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. pages 1–8, June 2008.

[60] Mustansar Fiaz, Arif Mahmood, and Soon Ki Jung. Tracking noisy targets: A review of recent object tracking approaches. *arXiv preprint arXiv:1802.03098*, 2018.

[61] Chelsea Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[62] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1180–1189. JMLR.org, 2015.

[63] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, January 2016.

[64] Jiyang Gao, Kan Chen, and Ram Nevatia. Ctap: Complementary temporal action proposal generation. In *The European Conference on Computer Vision (ECCV)*, 2018.

[65] Jiyang Gao, Zhenheng Yang, and Ram Nevatia. Cascaded boundary regression for temporal action detection. 2017.

[66] Jiyang Gao, Zhenheng Yang, Chen Sun, Kan Chen, and Ram Nevatia. Turn tap: Temporal unit regression network for temporal action proposals. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[67] J. . Gauvain and Chin-Hui Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE Transactions on Speech and Audio Processing*, 2(2):291–298, April 1994.

[68] Spyros Gidaris, Andrei Bursuc, Nikos Komodakis, Patrick Perez, and Matthieu Cord. Boosting few-shot visual learning with self-supervision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[69] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages

4367–4375, 2018.

[70] Spyros Gidaris and Nikos Komodakis. Generating classification weights with gnn denoising autoencoders for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[71] Rohit Girdhar, João Carreira, Carl Doersch, and Andrew Zisserman. Video action transformer network. In *CVPR*, 2018.

[72] Rohit Girdhar and Deva Ramanan. Attentional pooling for action recognition. In *NIPS*, 2017.

[73] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[74] Georgia Gkioxari and Jitendra Malik. Finding action tubes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 759–768, 2015.

[75] Joshua Gleason, Rajeev Ranjan, Steven Schwarcz, Carlos Castillo, Jun-Cheng Chen, and R. Chellappa. A proposal-based solution to spatio-temporal action detection in untrimmed videos. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 141–150, 2019.

[76] Joshua Gleason, Rajeev Ranjan, Steven Schwarcz, Carlos Castillo, Jun-Cheng Chen, and Rama Chellappa. A proposal-based solution to spatio-temporal action detection in untrimmed videos. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 141–150, 2018.

[77] Joshua Gleason, Rajeev Ranjan, Steven Schwarcz, Carlos D. Castillo, Jun-Chen Cheng, and Rama Chellappa. A proposal-based solution to spatio-temporal action detection in untrimmed videos. *CoRR*, abs/1811.08496, 2018.

[78] Joshua Gleason, S. Schwarcz, R. Ranjan, Carlos D. Castillo, Jun-Cheng Chen, and Ramalingam Chellappa. Activity detection in untrimmed videos using chunk-based classifiers. *2020 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 107–116, 2020.

[79] Joshua Gleason, Steven Schwarcz, Rajeev Ranjan, Carlos D. Castillo, Jun-Cheng Chen, and Rama Chellappa. Activity detection in untrimmed videos using chunk-based classifiers. In

2020 IEEE Winter Applications of Computer Vision Workshops (WACVW), pages 107–116, 2020.

[80] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 513–520, USA, 2011. Omnipress.

[81] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[82] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[83] A. Gretton, AJ. Smola, J. Huang, M. Schmittfull, KM. Borgwardt, and B. Schölkopf. *Covariate shift and local learning by distribution matching*, pages 131–160. MIT Press, Cambridge, MA, USA, 2009.

[84] Chunhui Gu, Chen Sun, David A. Ross, Caroline Pantofaru, Yeqing Li, Sudheendra Vijayanarasimhan, George Toderici, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jagannath Malik. Ava: A video dataset of spatio-temporally localized atomic visual actions. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6047–6056, 2017.

[85] Chunhui Gu, Chen Sun, Sudheendra Vijayanarasimhan, Caroline Pantofaru, David A. Ross, George Toderici, Yeqing Li, Susanna Ricco, Rahul Sukthankar, Cordelia Schmid, and Jitendra Malik. AVA: A video dataset of spatio-temporally localized atomic visual actions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[86] P. Haeusser, T. Frerix, A. Mordvintsev, and D. Cremers. Associative domain adaptation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2784–2792, Oct 2017.

[87] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace

the history of 2d cnns and imagenet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[88] Bharath Hariharan and Ross B. Girshick. Low-shot visual recognition by shrinking and hallucinating features. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3037–3046, 2017.

[89] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[90] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. mar 2017.

[91] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[92] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (ICCV)*, pages 770–778, 2016.

[93] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference Computer Vision (ECCV)*, 2016.

[94] Geoffrey E. Hinton, Oriol Vinyals, and J. Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[95] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[96] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. CyCADA: Cycle-consistent adversarial domain adaptation. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1989–1998, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[97] Judy Hoffman, Dequan Wang, Fisher Yu, and Trevor Darrell. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *CoRR*, abs/1612.02649, 2016.

[98] Rui Hou, Chen Chen, and Mubarak Shah. Tube convolutional neural network (t-cnn) for action detection in videos. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[99] Rui Hou, Rahul Sukthankar, and Mubarak Shah. Real-time temporal action localization in untrimmed videos by sub-action discovery. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2017.

[100] Dong Huang and Fernando De La Torre. Facial action transfer with personalized bilinear regression. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 144–158, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[101] Jingjia Huang, Nannan Li, Tao Zhang, Ge Li, Tiejun Huang, and Wen] Gao. Sap: Self-adaptive proposal model for temporal action detection based on reinforcement learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[102] Minyoung Huh, Andrew Liu, Andrew Owens, and Alexei A. Efros. Fighting fake news: Image splice detection via learned self-consistency. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[103] Naoto Inoue, Ryosuke Furuta, Toshihiko Yamasaki, and Kiyoharu Aizawa. Cross-domain weakly-supervised object detection through progressive domain adaptation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[104] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model. may 2016.

[105] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[106] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014.

[107] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. In *International Conf. on Computer Vision (ICCV)*, pages 3192–3199, December 2013.

[108] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[109] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes, 2014.

[110] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. `http://crcv.ucf.edu/THUMOS14/`, 2014.

[111] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[112] David Joy. Actev scorer. `https://github.com/usnistgov/ActEV_Scorer/tree/v0.3.0`, 2018.

[113] Vicky Kalogeiton, Philippe Weinzaepfel, Vittorio Ferrari, and Cordelia Schmid. Action tubelet detector for spatio-temporal action localization. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[114] Svebor Karaman, Lorenzo Seidenari, and Alberto Del Bimbo. Fast saliency based pooling of fisher encoded dense trajectories. In *ECCV THUMOS Workshop*, 2014.

[115] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1725–1732, 2014.

[116] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019.

[117] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[118] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[119] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*,

abs/1412.6980, 2014.

[120] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[121] Pavel Korshunov and Sébastien Marcel. Deepfakes: a new threat to face recognition? assessment and detection, 12 2018.

[122] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[123] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[124] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[125] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, Feb 2001.

[126] B. Lake, R. Salakhutdinov, and J. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350:1332 – 1338, 2015.

[127] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[128] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[129] Kwonjoon Lee, Subhransu Maji, A. Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10649–10657, 2019.

[130] J. Li, Y. Wang, C. Wang, Y. Tai, J. Qian, J. Yang, C. Wang, J. Li, and F. Huang. Dsfd: Dual shot face detector. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5055–5064, 2019.

[131] Lingzhi Li, Jianmin Bao, Ting Zhang, Hao Yang, Dong Chen, Fang Wen, and B. Guo. Face x-ray for more general face forgery detection. *2020 IEEE/CVF Conference on Computer*

Vision and Pattern Recognition (CVPR), pages 5000–5009, 2020.

[132] Y. Li, X. Yang, P. Sun, H. Qi, and S. Lyu. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3204–3213, 2020.

[133] Yuezun Li and S. Lyu. Exposing deepfake videos by detecting face warping artifacts. In *CVPR Workshops*, 2019.

[134] Yann Lifchitz, Yannis Avrithis, S. Picard, and Andrei Bursuc. Dense classification and implanting for few-shot learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9250–9259, 2019.

[135] Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. Bsn: Boundary sensitive network for temporal action proposal generation. In *The European Conference on Computer Vision (ECCV)*, 2018.

[136] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[137] Fuxiao Liu, Yinghan Wang, Tianlu Wang, and Vicente Ordonez. Visualnews : A large multi-source news image dataset. *arXiv: Computer Vision and Pattern Recognition*, 2020.

[138] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[139] Xiang Long, Chuang Gan, Gerard de Melo, Jiajun Wu, Xiao Liu, and Shilei Wen. Attention clusters: Purely attention based local feature integration for video classification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7834–7843, 2017.

[140] Grace Luo, Trevor Darrell, and Anna Rohrbach. Newsclippings: Automatic generation of out-of-context multimodal media. *ArXiv*, abs/2104.05893, 2021.

[141] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Xiaowei Zhao, and Tae-Kyun Kim. Multiple object tracking: A literature review. *arXiv preprint arXiv:1409.7618*, 2014.

[142] Mihai Marian Puscas, Enver Sangineto, Dubravko Culibrk, and Nicu Sebe. Unsupervised tube extraction using transductive learning and dense trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 1653–1661, 2015.

[143] Rabindra D Mehta. Aerodynamics of sports balls. *Annual Review of Fluid Mechanics*, 17(1):151–189, 1985.

[144] Pascal Mettes, Jan C van Gemert, and Cees GM Snoek. Spot on: Action localization from pointly-supervised proposals. In *European Conference on Computer Vision*, pages 437–453. Springer, 2016.

[145] Antoine Miech, Ivan Laptev, and Josef Sivic. Learnable pooling with context gating for video classification. *ArXiv*, abs/1706.06905, 2017.

[146] Huaxiao Mo, Bolin Chen, and Weiqi Luo. Fake faces identification via convolutional neural network. In *Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security*, IHamp;MMSec '18, page 43–47, New York, NY, USA, 2018. Association for Computing Machinery.

[147] Mathew Monfort, Bolei Zhou, Sarah Adel Bargal, Alex Andonian, Tom Yan, Kandan Ramakrishnan, Lisa M. Brown, Quanfu Fan, Dan Gutfreund, Carl Vondrick, and Aude Oliva. Moments in time dataset: one million videos for event understanding. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[148] Fiza Murtaza, Muhammad Haroon Yousaf, Sergio Velastin, and Yu Qian. End-to-end temporal action detection using bag of discriminant snippets (bods). *IEEE Signal Processing Letters*, PP:1–1, 12 2018.

[149] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms, 2011.

[150] Mark L Nagurka. Aerodynamic effects in a dropped ping-pong ball experiment. *International Journal of Engineering Education*, 2003.

[151] Farhood Negin and François Bremond. Human action recognition in videos: A survey. *INRIA Technical Report*, 2016.

[152] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[153] Alejandro Newell, Zhiao Huang, and Jia Deng. Associative Embedding: End-to-End Learning for Joint Detection and Grouping. nov 2016.

[154] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked Hourglass Networks for Human Pose Estimation. mar 2016.

[155] Phuc Nguyen, Ting Liu, Gautam Prasad, and Bohyung Han. Weakly supervised action localization by sparse temporal pooling network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[156] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *ArXiv*, abs/1803.02999, 2018.

[157] Y. Nirkin, Y. Keller, and T. Hassner. Fsgan: Subject agnostic face swapping and reenactment. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7183–7192, 2019.

[158] NIST. Actev'21 sequestered data leaderboard, https://actev.nist.gov/sdl.

[159] NIST. Activity extended video (actev) prize challenge, https://actev.nist.gov/prizechallenge.

[160] NIST. Trecvid 2017 evaluation for surveillance event detection, https://www.nist.gov/itl/iad/mig/trecvid-2017-evaluation-surveillance-event-detection.

[161] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J.K. Aggarwal, Hyungtae Lee, Larry Davis, Eran Swears, Xiaoyang Wang, Qiang Ji, Kishore Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Anesco Fong Bi Song, Amit Roy-Chowdhury, and Mita Desai. A large-scale benchmark dataset for event recognition in surveillance video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[162] Dan Oneata, Jérôme Revaud, Jakob Verbeek, and Cordelia Schmid. Spatio-temporal object detection proposals. In *European conference on computer vision*, pages 737–752. Springer, 2014.

[163] Dan Oneata, Jakob Verbeek, and Cordelia Schmid. The lear submission at thumos 2014. 2014.

[164] Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent

adaptive metric for improved few-shot learning. In *NeurIPS*, 2018.

[165] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin Murphy. Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. In *ECCV*, 2018.

[166] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, and Kevin P. Murphy. Towards accurate multi-person pose estimation in the wild. *CoRR*, abs/1701.01779, 2017.

[167] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[168] Georgios Pavlakos, Xiaowei Zhou, Konstantinos G Derpanis, and Kostas Daniilidis. Harvesting Multiple Views for Marker-less 3D Human Pose Annotations.

[169] Georgios Pavlakos, Xiaowei Zhou, Konstantinos G. Derpanis, and Kostas Daniilidis. Coarse-to-fine volumetric prediction for single-image 3d human pose. *CoRR*, abs/1611.07828, 2016.

[170] Ivan Perov, Daiheng Gao, Nikolay Chervoniy, Kunlin Liu, Sugasa Marangonda, Chris Umé, Mr. Dpfks, Carl Shift Facenheim, Luis RP, Jian Jiang, Sheng Zhang, Pingyu Wu, Bo Zhou, and Weiming Zhang. Deepfacelab: A simple, flexible and extensible face swapping framework, 2020.

[171] A. J. Piergiovanni and Michael S. Ryoo. Representation flow for action recognition. In *CVPR*, 2018.

[172] A. C. Popescu and H. Farid. Exposing digital forgeries by detecting traces of resampling. *IEEE Transactions on Signal Processing*, 53(2):758–767, 2005.

[173] A. C. Popescu and H. Farid. Exposing digital forgeries in color filter array interpolated images. *IEEE Transactions on Signal Processing*, 53(10):3948–3959, 2005.

[174] H. Qi, M. Brown, and D. G. Lowe. Low-shot learning with imprinted weights. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5822–5830, 2018.

[175] L. Qiao, Y. Shi, Jia Li, Yaowei Wang, Tiejun Huang, and Yonghong Tian. Transductive episodic-wise adaptive metric for few-shot learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3602–3611, 2019.

[176] Siyuan Qiao, Chenxi Liu, Wei Shen, and A. Yuille. Few-shot image recognition by predicting parameters from activations. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7229–7238, 2018.

[177] Ariadna Quattoni, Michael Collins, and Trevor Darrell. Conditional random fields for object recognition. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1097–1104. MIT Press, 2005.

[178] Alec Radford, Jong Wook Kim, Chris Hallacy, A. Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *ArXiv*, abs/2103.00020, 2021.

[179] N. Rahmouni, V. Nozick, J. Yamagishi, and I. Echizen. Distinguishing computer graphics from natural images using convolution neural networks. In *2017 IEEE Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2017.

[180] D Ramanan, D A. Forsyth, and A Zisserman. Strike a pose: tracking people by finding stylized poses. 1:271– 278 vol. 1, 07 2005.

[181] Vignesh Ramanathan, Jonathan Huang, Sami Abu-El-Haija, Alexander Gorban, Kevin Murphy, and Li Fei-Fei. Detecting events and key actors in multi-person videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3043–3053, 2016.

[182] A. Ravichandran, Rahul Bhotika, and Stefano Soatto. Few-shot learning with embedded class models and shot-free meta training. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 331–339, 2019.

[183] Mengye Ren, E. Triantafillou, S. Ravi, J. Snell, Kevin Swersky, J. Tenenbaum, H. Larochelle, and R. Zemel. Meta-learning for semi-supervised few-shot classification.

ArXiv, abs/1803.00676, 2018.

[184] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[185] Alexander Richard and Juergen Gall. Temporal action detection using a statistical language model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[186] Garry Robinson and Ian Robinson. The motion of an arbitrarily rotating spherical projectile and its application to ball games. *Physica Scripta*, 88(1):018101, 2013.

[187] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[188] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. FaceForensics++: Learning to Detect Manipulated Facial Images. jan 2019.

[189] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[190] Andrei A. Rusu, D. Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *ArXiv*, abs/1807.05960, 2019.

[191] Seyed Morteza Safdarnejad, Xiaoming Liu, Lalita Udpa, Brooks Andrus, John Wood, and Dean Craven. Sports videos in the wild (svw): A video dataset for sports analysis. In *Proc. International Conference on Automatic Face and Gesture Recognition*, Ljubljana, Slovenia, May 2015.

[192] Suman Saha, Gurkirt Singh, and Fabio Cuzzolin. Amtnet: Action-micro-tube regression by end-to-end trainable deep architecture. 2017.

[193] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. Asymmetric tri-training for unsupervised domain adaptation. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the*

34th International Conference on Machine Learning, volume 70 of *Proceedings of Machine Learning Research*, pages 2988–2997, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[194] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum Classifier Discrepancy for Unsupervised Domain Adaptation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[195] Kuniaki Saito, Shohei Yamamoto, Yoshitaka Ushiku, and Tatsuya Harada. Open set domain adaptation by backpropagation. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[196] Swami Sankaranarayanan, Yogesh Balaji, Carlos D. Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[197] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser-Nam Lim, and Rama Chellappa. Unsupervised domain adaptation for semantic segmentation with gans. *CoRR*, abs/1711.06969, 2017.

[198] P. Sarkar and G. Nagy. Style-consistency in isogenous patterns. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 1169–1174, Sept 2001.

[199] Victor Garcia Satorras and Joan Bruna. Few-shot learning with graph neural networks. *ArXiv*, abs/1711.04043, 2018.

[200] Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. Action recognition using visual attention. *ArXiv*, abs/1511.04119, 2015.

[201] Xingjian Shi, Zhourong Chen, Hao Wang Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *NIPS*, 2015.

[202] Zheng Shou, Jonathan Chan, Alireza Zareian, Kazuyuki Miyazawa, and Shih-Fu Chang. Cdc: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1417–1426, 2017.

[203] Zheng Shou, Hang Gao, Lei Zhang, Kazuyuki Miyazawa, and Shih-Fu Chang. Autoloc: Weakly-supervised temporal action localization in untrimmed videos. In *The European Conference on Computer Vision (ECCV)*, 2018.

[204] Zheng Shou, Dongang Wang, and Shih-Fu Chang. Temporal action localization in untrimmed videos via multi-stage cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[205] Rui Shu, Hung Bui, Hirokazu Narui, and Stefano Ermon. A DIRT-t approach to unsupervised domain adaptation. In *International Conference on Learning Representations (ICLR)*, 2018.

[206] Leonid Sigal, Alexandru O. Balan, and Michael J. Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated humanmotion. *International Journal of Computer Vision*, 87(1):4, Aug 2009.

[207] Leonid Sigal and Michael J. Black. Measure locally, reason globally: Occlusion-sensitive articulated pose estimation. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2041–2048, Washington, DC, USA, 2006. IEEE Computer Society.

[208] Leonid Sigal, Michael Isard, Horst Haussecker, and Michael J. Black. Loose-limbed people: Estimating 3d human pose and motion using non-parametric belief propagation. *Int. J. Comput. Vision*, 98(1):15–48, May 2012.

[209] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. *ArXiv*, abs/1604.01753, 2016.

[210] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, 2014.

[211] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[212] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[213] Raymond Smith, Chunhui Gu, Dar-Shyang Lee, Huiyi Hu, Ranjith Unnikrishnan, Julian Ibarz, Sacha Arnoud, and Sophia Lin. End-to-end interpretation of the french street name signs dataset. In *ECCV Workshops*, 2016.

[214] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[215] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *ArXiv*, abs/1212.0402, 2012.

[216] Nitish Srivastava, Geoffrey E. Hinton, A. Krizhevsky, Ilya Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[217] Jong-Chyi Su, Subhransu Maji, and B. Hariharan. When does self-supervision improve few-shot learning? In *ECCV*, 2020.

[218] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2058–2065. AAAI Press, 2016.

[219] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In Gang Hua and Hervé Jégou, editors, *Computer Vision – ECCV 2016 Workshops*, pages 443–450, Cham, 2016. Springer International Publishing.

[220] Chen Sun, Per Karlsson, Jiajun Wu, Joshua B Tenenbaum, and Kevin Murphy. Stochastic prediction of multi-agent interactions from partial observations. *arXiv preprint arXiv:1902.09641*, 2019.

[221] Min Sun and Silvio Savarese. Articulated part-based model for joint object detection and pose estimation. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 723–730, Washington, DC, USA, 2011. IEEE Computer Society.

[222] Shuyang Sun, Zhanghui Kuang, Wanli Ouyang, Lu Sheng, and Wei Zhang. Optical flow guided feature: A fast and robust motion representation for video action recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1390–1399, 2017.

[223] Flood Sung, Yongxin Yang, L. Zhang, T. Xiang, P. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[224] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks.

[225] Christian Szegedy, W. Liu, Y. Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, V. Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[226] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[227] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[228] G. W. Taylor, L. Sigal, D. J. Fleet, and G. E. Hinton. Dynamical binary latent variable models for 3d human pose tracking. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 631–638, June 2010.

[229] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 38(4), July 2019.

[230] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. *Commun. ACM*, 62(1):96–104, December 2018.

[231] Yonglong Tian, Yue Wang, Dilip Krishnan, J. Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? *ArXiv*, abs/2003.11539, 2020.

[232] Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *CoRR*, abs/1406.2984, 2014.

[233] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neu-

ral networks. *CoRR*, abs/1312.4659, 2013.

[234] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[235] Du Tran, Jamie Ray, Zheng Shou, Shih-Fu Chang, and Manohar Paluri. Convnet architecture search for spatiotemporal feature learning. *arXiv preprint arXiv:1708.05038*, 2017.

[236] Hung-Yu Tseng, Hsin-Ying Lee, Jia-Bin Huang, and Ming-Hsuan Yang. Cross-domain few-shot classification via learned feature-wise transformation. *ArXiv*, abs/2001.08735, 2020.

[237] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2962–2971, 2017.

[238] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *CoRR*, abs/1412.3474, 2014.

[239] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, 2016.

[240] Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1510–1517, 2018.

[241] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[242] Sriharsha Veeramachaneni and George Nagy. Adaptive classifiers for multisource ocr. *Document Analysis and Recognition*, 6(3):154–166, Mar 2003.

[243] R. Vemulapalli, F. Arrate, and R. Chellappa. Human action recognition by representing 3d skeletons as points in a lie group. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–595, June 2014.

[244] R. Vemulapalli and R. Chellappa. Rolling rotations for recognizing human actions from 3d skeletal data. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4471–4479, June 2016.

[245] P. Vincent, H. Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[246] Pascal Vincent, H. Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML '08*, 2008.

[247] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[248] Riccardo Volpi, Pietro Morerio, Silvio Savarese, and Vittorio Murino. Adversarial feature augmentation for unsupervised domain adaptation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[249] Risto Vuorio, S. Sun, Hexiang Hu, and Joseph J. Lim. Multimodal model-agnostic meta-learning via task-aware modulation. In *NeurIPS*, 2019.

[250] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision*, 103(1):60–79, 2013.

[251] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition and detection by combining motion and appearance features. 2014.

[252] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4305–4314, 2015.

[253] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[254] Weihong Wang and Hany Farid. Exposing digital forgeries in video by detecting duplication. In *Proceedings of the 9th Workshop on Multimedia amp; Security*, MMamp;Sec '07, page 35–42, New York, NY, USA, 2007. Association for Computing Machinery.

[255] Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. Non-local neural

networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2017.

[256] Yu-Xiong Wang, Ross B. Girshick, M. Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7278–7286, 2018.

[257] Zhangyang Wang, Jianchao Yang, Hailin Jin, Eli Shechtman, Aseem Agarwala, Jonathan Brandt, and Thomas S. Huang. Real-world font recognition using deep network and domain adaptation. *CoRR*, abs/1504.00028, 2015.

[258] Zhangyang Wang, Jianchao Yang, Hailin Jin, Eli Shechtman, Jonathan Brandt Aseem Agarwala, and Thomas S. Huang. Decomposition-based domain adaptation for real-world font recognition.

[259] Ralf Widenhorn. The physics of juggling a spinning ping-pong ball. *American Journal of Physics*, 84(12):936–942, 2016.

[260] Zbigniew Wojna, Alexander N. Gorban, Dar-Shyang Lee, Kevin Murphy, Qian Yu, Yeqing Li, and Julian Ibarz. Attention-based extraction of structured information from street view imagery. *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 01:844–850, 2017.

[261] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6026–6035, 2018.

[262] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *Proceedings of the IEEE international conference on computer vision*, pages 4705–4713, 2015.

[263] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning for video understanding. *ArXiv*, abs/1712.04851, 2017.

[264] Huijuan Xu, Abir Das, and Kate Saenko. R-c3d: Region convolutional 3d network for temporal activity detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[265] Ke Yang, Peng Qiao, Dongsheng Li, Shaohe Lv, and Yong Dou. Exploring temporal preser-

vation networks for precise temporal action localization. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[266] Ke Yang, Peng Qiao, Dongsheng Li, Shaohe Lv, and Yong Dou. Exploring temporal preservation networks for precise temporal action localization. In *AAAI*, 2018.

[267] Y. Yang and D. Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR 2011*, pages 1385–1392, June 2011.

[268] Serena Yeung, Olga Russakovsky, Ning Jin, Mykhaylo Andriluka, Greg Mori, and Li Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. pages 1–15, 2015.

[269] Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[270] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.

[271] Jun Yuan, Bingbing Ni, Xiaokang Yang, and Ashraf A Kassim. Temporal action localization with pyramid of score distribution features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[272] Ze-Huan Yuan, Jonathan C Stroud, Tong Lu, and Jia Deng. Temporal action localization by structured maximal sums. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[273] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4694–4702, 2015.

[274] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*, pages 214–223. Springer, 2007.

[275] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *ArXiv*, abs/1605.07146, 2016.

[276] E. Zakharov, A. Shysheya, E. Burkov, and V. Lempitsky. Few-shot adversarial learning