ABSTRACT

Title of Dissertation: Closing the Gap Between Classification and Retrieval Models

Ahmed Taha, Doctor of Philosophy, 2021

Dissertation directed by: Professor Larry Davis, Abhinav Shrivastava
University of Maryland, College Park

Retrieval networks learn a feature embedding where similar samples are close together, and different samples are far apart. This feature embedding is essential for computer vision applications such as face/person recognition, zero-shot learning, and image retrieval. Despite these important applications, retrieval networks are less popular compared to classification networks due to multiple reasons: (1) The cross-entropy loss – used with classification networks – is stabler and converges faster compared to metric learning losses – used with retrieval networks. (2) The cross-entropy loss has a huge toolbox of utilities and extensions. For instance, both AdaCos and self-knowledge distillation have been proposed to tackle low sample complexity in classification networks; also, both CAM and Grad-CAM have been proposed to visualize attention in classification networks. To promote retrieval networks, it is important to equip them with an equally powerful toolbox. Accordingly, we propose an evolution-inspired approach to tackle low sample complexity in feature embedding. Then, we propose SVMax to regularize the feature embedding and avoid model collapse. Furthermore, we propose L2-CAF to visualize attention in

retrieval networks.

To tackle low sample complexity, we propose an evolution-inspired training approach to boost performance on relatively small datasets. The knowledge evolution (KE) approach splits a deep network into two hypotheses: the fit-hypothesis and the reset-hypothesis. We iteratively evolve the knowledge inside the fit-hypothesis by perturbing the reset-hypothesis for multiple generations. This approach not only boosts performance but also learns a slim (pruned) network with a smaller inference cost. KE reduces both overfitting and the burden for data collection.

To regularize the feature embedding and avoid model collapse, We propose singular value maximization (SVMax) to promote a uniform feature embedding. Our formulation mitigates model collapse and enables larger learning rates. SVMax is oblivious to both the input-class (labels) and the sampling strategy. Thus it promotes a uniform feature embedding in both supervised and unsupervised learning. Furthermore, we present a mathematical analysis of the mean singular value's lower and upper bounds. This analysis makes tuning the SVMax's balancing-hyperparameter easier when the feature embedding is normalized to the unit circle.

To support retrieval networks with a visualization tool, we formulate attention visualization as a constrained optimization problem. We leverage the unit L2-Norm constraint as an attention filter (L2-CAF) to localize attention in both classification and retrieval networks. This approach imposes no constraints on the network architecture besides having a convolution layer. The input can be a regular image or a pre-extracted convolutional feature. The network output can be logits trained with cross-entropy or a space embedding trained with a ranking loss. Furthermore,

this approach neither changes the original network weights nor requires fine-tuning. Thus, network performance remains intact. The visualization filter is applied only when an attention map is required. Thus, it poses no computational overhead during inference. L2-CAF visualizes the attention of the last convolutional layer of GoogLeNet within 0.3 seconds.

Finally, we propose a compromise between retrieval and classification networks. We propose a simple, yet effective, two-head architecture — a network with both logits and feature-embedding heads. The embedding head — trained with a ranking loss — limits the overfitting capabilities of the cross-entropy loss by promoting a smooth embedding space. In our work, we leverage the semi-hard triplet loss to allow a dynamic number of modes per class, which is vital when working with imbalanced data. Also, we refute a common assumption that training with a ranking loss is computationally expensive. By moving both the triplet loss sampling and computation to the GPU, the training time increases by just 2%.

Closing the Gap Between Classification and Retrieval Models

by

Ahmed Taha

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

Advisory Committee:
Professor Larry Davis, Chair/Advisor
Professor Abhinav Shrivastava, Co-Advisor
Professor David Jacobs
Professor Ramani Duraiswami
Professor Tom Goldstein
Professor Behtash Babadi

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Many computer vision applications can be formulated as a classification problem – *e.g.*, object recognition, semantic segmentation, and pose estimation. This drives a lot of attention to the cross-entropy loss. However, classification networks, with their logits output, has a number of limitations: (1) Classification networks suffer with unseen classes because cross-entropy assumes a fixed number of logits, *i.e.*, a closed-set recognition; (2) Classification networks suffers with nearest neighbor retrieval because cross-entropy promotes neither intra-class compactness nor inter-class margin-maximization. Retrieval networks can mitigate the aforementioned limitations. Unfortunately, retrieval networks have their own limitations.

During training, a retrieval network leverage a metric learning (ranking) loss. Compared to cross-entropy, a ranking loss converges slower and its performance is highly dependent on the mini-batch sampling procedure (*e.g.*, semi-hard vs hard mining). Proxy-based [65, 90] and Average-precision-based [6, 108] losses promise a faster convergence rate. Yet, retrieval networks still miss the extensive toolbox available for classification networks – tools for low sample complexity, regularization, visualization, and imbalance training dataset. In this thesis, we close the gap between classification and retrieval networks by proposing the following three tools:

1. A classification network has a toolbox for low sample complexity, *e.g.*, class-wise knowledge-distillation [153] and Label-smoothing [91]. For retrieval networks, we propose a knowledge evolution (KE) training approach. KE split a network into two hypotheses (subnetworks): he fit-hypothesis $H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$. These hypotheses are outlined by a binary mask M; 1 for $H^{\triangle}$ and 0 for $H^{\triangledown}$, *i.e.*, $H^{\triangle} = MN$ and $H^{\triangledown} = (1-M)N$. We iteratively evolve the knowledge inside the $H^{\triangle}$ by perturbing the $H^{\triangledown}$ for multiple generations. This approach not only boosts performance, but also reduces inference cost. On the CUB-200 retrieval dataset, KE boosts recall@1 by an absolute 5%; on the CAR-196 retrieval dataset, KE boosts recall@1 by an absolute 27%. This performance improvement is accompanied by a relative 35% reduction in inference cost. Furthermore, we show that KE supports classification networks as well.

2. A classification network has a toolbox for regularization, *e.g.*, AdaCos [162] and Virtual Softmax [13]. For retrieval networks, we propose SVMax as a feature embedding regular. We observe that singular values, from singular value decomposition, provide insights about the feature embedding structure. For a feature embedding in R^d, maximizing the mean singular value ($\sum_1^d s_i = s_\mu$) promotes data dispersion across all dimensions, *i.e.*, a uniform embedding. We further observe that s_μ has lower and upper bounds. Through these bounds, the hyperparameter tuning of SVMax is straightforward as it depends on the range of the loss function only. The SVMax regularizer supports both super-

vised and unsupervised learning.

3. A classification network has a toolbox for visualization, *e.g.*, CAM [164] and Grad-CAM [120]. For retrieval networks, we leverage the L2-Norm ($||.||_2$) constraint as an attention visualization filter (L2-CAF). The main idea is to consider the attention heatmap as a filter f that passes key image regions and blocks irrelevant regions. With this in mind, we apply the filter f to the last convolutional layer A, using Hadamard multiplication $A \bigodot f = A' \in R^{w \times h \times k}$, and optimize f using stochastic gradient descent. The optimization objective is to minimize the difference between the original network output and the filtered output, *i.e.*, $||NT(x) - FT(x, f)||^2$. To avoid a trivial solution and emphasize key image regions, the optimization problem is subject to the constraint $||f||_2 = 1$. Unlike related literature, this approach works for a large spectrum of neural networks. Furthermore, L2-CAF scales efficiently to 3D volumes – videos and medical images. It does not require fine-tuning or increase computational cost during inference. The proposed optimization problem is small (*e.g.*, $\in R^{7 \times 7}$), and the filter f converges on GoogLeNet within ≈ 0.3 seconds.

In the last part of this thesis, we propose a simple, yet efficient, modification to standard classification architectures. We introduce a two-head architecture that generates both class logits and feature embedding. The cross-entropy and triplet loss are applied to the logits and embedding head, respectively. The triplet loss promotes class compactness and inter-class margin maximization. These objectives

regularize the feature embedding learned by the logits head. The two-head architecture neither requires a fixed number of embedding modes nor computes explicit class representatives. Furthermore, the two-head architecture outperforms related baselines, yet, its key advantage is simplicity. This advantage is essential to promote the two-head architecture usability. Our proposal requires a *single* fully connected layer and increases the training time by $\approx 2\%$. With an embedding head, we enable compact feature embedding, boost classification performance and bring retrieval capabilities to standard architectures.

Chapter 2: Knowledge Evolution in Neural Networks

Deep learning relies on the availability of a large corpus of data (labeled or unlabeled). Thus, one challenging unsettled question is how to train a deep network on a relatively small dataset? To tackle this question, we propose an evolution-inspired training approach to boost performance on relatively small datasets. The knowledge evolution approach splits a deep network into two hypotheses: the fit-hypothesis and the reset-hypothesis. We iteratively evolve the knowledge inside the fit-hypothesis by perturbing the reset-hypothesis for multiple generations. This approach not only boosts performance, but also learns a slim (pruned) network with a smaller inference cost.

We evaluate the knowledge evolution (KE) approach on various network architectures and loss functions. We evaluate KE using relatively small datasets (*e.g.*, CUB-200) and randomly initialized deep networks. KE achieves an absolute 21% improvement margin on a state-of-the-art baseline. This performance improvement is accompanied by a relative 73% reduction in inference cost. KE achieves state-of-the-art results on classification and metric learning benchmarks. Code available at *http://bit.ly/3uLgwYb*

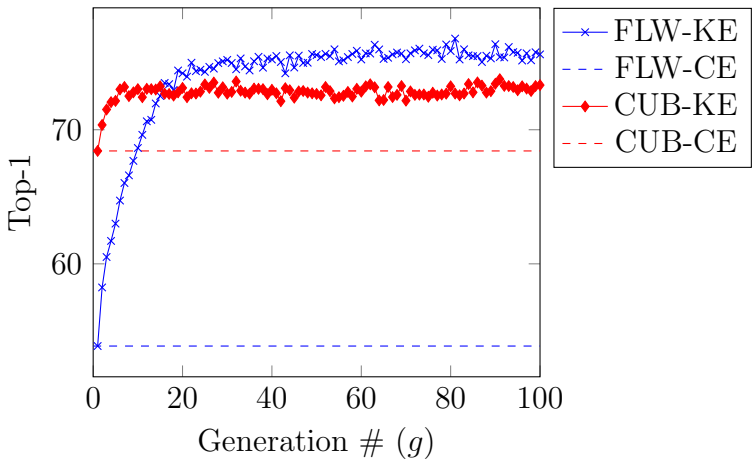


Figure 2.1: Classification performance on Flower-102 (FLW) and CUB-200 (CUB) datasets trained on a randomly initialized ResNet18. The horizontal dashed-lines denote a SOTA cross-entropy (CE) baseline [153]. The marked-curves show our approach (KE) performance across generations. The 100^{th} generation KE-N_{100} achieves absolute 21% and 5% improvement margins over the Flower-102 and CUB-200 baselines, respectively.

2.1 Introduction

Gene transfer is the transfer of genetic information from a parent to its offspring. Genes encode genetic instructions (knowledge) from ancestors to descendants. The ancestors do not necessarily have better knowledge; yet, the evolution of knowledge across generations promotes a better learning curve for the descendants. In this paper, we strive to replicate this process for deep networks. We encapsulate a deep network's knowledge inside a subnetwork, dubbed the fit-hypothesis $H^{\triangle}$. Then, we pass the fit-hypothesis's knowledge from a parent network to its offspring (next deep network generation). We repeat this process iteratively and demonstrate a significant performance improvement in the descendant networks as shown in Fig. 2.1.

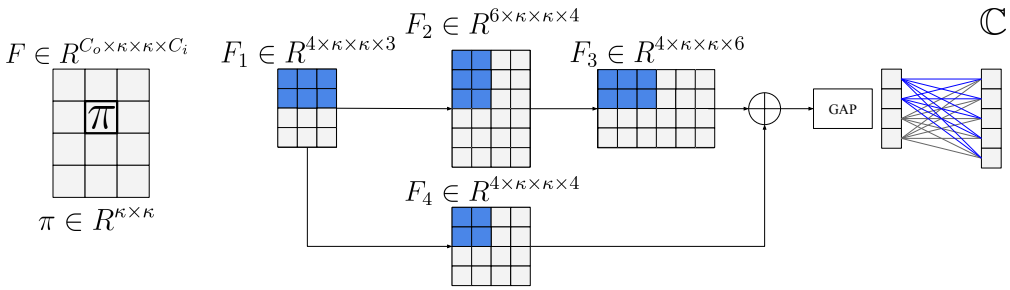


Figure 2.2: A split network illustration using a toy residual network. (Left) A convolutional filter F with $C_i = 3$ input, $C_o = 4$ output channels, and 2D kernels (*e.g.*, $\pi \in R^{3 \times 3}$). (Center-Right) A toy residual network N with a three-channel input (*e.g.*, RGB image) and a five-logit output ($\mathbb{C} = 5$). GAP denotes a global average pooling layer while $\bigoplus$ denotes the add operation. We split N into a fit-hypothesis $H^{\triangle}$ (dark-blue) and a reset-hypothesis $H^{\triangledown}$ (light-gray). The fit-hypothesis $H^{\triangle}$ is a slim network that can be extracted from the dense network N to perform inference efficiently.

The lottery ticket literature [31, 165, 89, 107, 35] regards a dense network as a set of hypotheses (subnetworks). Zhou *et al.* [165] propose a sampling-based approach, while Ramanujan *et al.* [107] propose an optimization-based approach, to identify the best *randomly-initialized* hypothesis. This hypothesis may be called the lottery ticket, but it is still limited by its random initialization. In this paper, we pick a random hypothesis, with inferior performance, and iteratively evolve its knowledge.

The main contribution of this paper is an evolution-inspired training approach. To evolve knowledge inside a deep network, we split the network into two hypotheses (subnetworks): the fit-hypothesis $H^{\triangle}$ and the reset hypothesis $H^{\triangledown}$ as shown in Fig. 2.2. We evolve the knowledge inside $H^{\triangle}$ by re-training the network for multiple generations. For every new generation, we perturb the weights inside $H^{\triangledown}$ to encourage the $H^{\triangle}$ to learn an independent representation. This knowledge evolution approach boosts performance on relatively small datasets and promotes a better

learning curve for descendant networks. Our intuitions are presented in Sec. 2.3.3 and empirically validated in Sec. 2.5.

The knowledge evolution (KE) approach requires network-splitting. If we split the weights of a neural network into two hypotheses ($H^{\triangle}$ and $H^{\triangledown}$) *randomly*, KE will boost performance. This emphasizes the generality of our approach. Furthermore, we propose a **ke**rnel-**l**evel convolutional-aware **s**plitting (KELS) technique to reduce inference cost. KELS is a splitting technique tailored for convolutional neural networks (CNNs). KELS splits a CNN such that the fit-hypothesis $H^{\triangle}$ is a slim independent network with a smaller inference cost as shown in Fig. 2.2. The KELS technique supports both vanilla CNNs (AlexNet and VGG) and modern residual networks.

KE supports various network architectures and loss functions. KE integrates seamlessly with other regularization techniques (*e.g.*, label smoothing). While KE increases the training time, the KELS technique reduces the inference cost significantly. Most importantly, KE mitigates overfitting on relatively small datasets, which in turn reduces the burden for data collection. Our community takes natural images for granted because they are available publicly. However, for certain applications, such as autonomous navigation and medical imaging, the data collection process is expensive even when labeling is not required.

In summary, the key contributions of this paper are:

1. A training approach, knowledge evolution (KE), that boosts the performance of deep networks on relatively small datasets (Sec. 2.3.1). We evaluate KE

using both classification (Sec. 2.4.1) and metric learning (Sec. 2.4.2) tasks. KE achieves SOTA results.

2. A network splitting technique, KELS, which learns a slim network automatically while training a deep network (Sec. 2.3.2). KELS supports a large spectrum of CNNs and introduces neither hyperparameters nor regularization terms. Our ablation studies (Sec. 2.5) demonstrate how KELS reduces inference cost significantly.

2.2 Related Work

This section compares knowledge evolution (KE) with two prominent training approaches: Born-Again Networks (BANs) [34] and Dense-Sparse-Dense (DSD) [42].

DSD [42] starts with a dense-phase to learn connections' weights and importance. Then, the sparse-phase prunes the unimportant connections and resumes training given a sparsity constraint. The final dense-phase removes the sparsity constraint, re-initializes the pruned connections, and trains the entire dense network. KE differs from DSD in multiple ways: (1) DSD masks (prunes) individual weights, while KE masks complete convolution kernels. Thus, DSD delivers dense networks, while KE delivers both dense and slim networks. (2) KE introduces the idea of a fit-hypothesis to encapsulate a network's knowledge and to evolve this knowledge across generations.

BANs [34] is a knowledge-distillation based approach. Similar to KE, BANs trains the same architecture iteratively. However, to transfer knowledge between

successive networks, BANs uses the class-logits distribution, while KE uses the networks' weights. This explains why BANs uses the teacher-student terminology while KE uses the parent-sibling terminology. This difference is important because (1) training a teacher network, which teaches future students, requires a large corpus of data (labeled or not). In contrast, KE acknowledges the deficiency of a parent network trained on a small dataset; (2) BANs randomly initializes student networks while KE leverages the knowledge of a parent network to initialize the next generation.

We distance our work from neural architecture search (NAS) literature [167, 79] such as Neural Rejuvenation [103] and MorphNet [37]. We assume the network's connections and the number of parameters are fixed.

2.3 Knowledge Evolution

In this section, we present (1) the knowledge evolution (KE) approach (Sec. 2.3.1), (2) various network-splitting techniques (Sec. 2.3.2), (3) intuitions behind KE (Sec. 2.3.3), and (4) how we evaluate KE (Sec. 2.3.4).

2.3.1 The Knowledge Evolution Training Approach

We first introduce our notation. We assume a deep network N with L layers. The network N has convolutional filters F, batch norm Z, and fully connected layers with weight W, bias B terms.

The Knowledge evolution (KE) approach starts by *conceptually* splitting the

deep network N into two exclusive hypotheses (subnetworks): the fit-hypothesis $H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$ as shown in Fig. 2.2. These hypotheses are outlined by a binary mask M; 1 for $H^{\triangle}$ and 0 for $H^{\triangledown}$, *i.e.*, $H^{\triangle} = MN$ and $H^{\triangledown} = (1-M)N$. We present various splitting techniques in Sec. 2.3.2. After outlining the hypotheses, the network N is initialized randomly, *i.e.*, both $H^{\triangle}$ and $H^{\triangledown}$ are initialized randomly. We train N for e epochs and refer to the trained network as the first generation N_1, where $H_1^{\triangle} = MN_1$ and $H_1^{\triangledown} = (1-M)N_1$.

To learn a better network (the next generation), we (1) **re-initialize** the network N using $H_1^{\triangle}$, then (2) **re-train** N to learn N_2. First, the network N is **re-initialized** using the convolutional filters F and weights W in the fit-hypothesis $H_1^{\triangle}$ from N_1, while the rest of the network ($H^{\triangledown}$) is initialized randomly. Formally, we re-initialize each layer l, using Hadamard product, as follows

$$F_l = M_l F_l + (1 - M_l)\, F_l^r, \tag{2.1}$$

where F_l is a convolutional filter at layer l, M_l is the corresponding binary mask and F_l^r is a randomly initialized tensor. These three tensors (F_l, F_l^r, and M_l) have the same size ($\in R^{C_o \times \kappa \times \kappa \times C_i}$). F_l^r is initialized using the default initialization distribution. For example, PyTorch uses Kaiming uniform [43] for convolution layers.

Similarly, we re-initialize the weight W_l and bias B_l through their corresponding binary masks. Modern architectures have bias terms in the single last fully connected layer only ($B \in R^{\mathbb{C}}$). Thus, for these architectures, all bias terms belong to the fit-hypothesis, *i.e.*, $B \subset H^{\triangle}$. We transfer the learned batch norm Z across

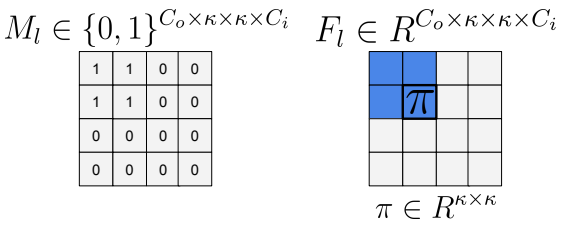


Figure 2.3: The KELS technique for CNNs. Given a split-rate s_r and a convolutional filter F_l at a layer l, the binary split-mask M_l outlines the first $\lceil s_r \times C_i \rceil$ kernels inside the first $\lceil s_r \times C_o \rceil$ filters. In this example, $C_o = C_i = 4$ and $s_r = 0.5$. Through KELS, the binary mask M outlines the fit-hypothesis $H^{\triangle}$ such that it is a slim network inside a dense network. The slim network $H^{\triangle}$ is equivalent to a dense network with $(1 - s_r^2)$ sparsity.

generations without randomization.

After re-initialization, we **re-train** N for e epochs to learn the second generation N_2. To learn better networks, we repeatedly **re-initialize** and **re-train** N for g generations. Basically, we transfer knowledge (convolutional filters and weights) from one generation to the next through the fit-hypothesis $H^{\triangle}$. It is important to note that (1) the contribution of a network-generation ends immediately-after initializing the next generation, *i.e.*, each generation is trained independently, (2) After training a new generation, the weights inside both hypotheses change, *i.e.*, $H_1^{\triangle} \neq H_2^{\triangle}$ and $H_1^{\triangledown} \neq H_2^{\triangledown}$, and (3) all network generations are trained using the exact hyperparameters, *i.e.*, same number of epochs, optimizer, etc.

2.3.2 Split-Networks

We support KE with two network-splitting techniques: (1) a simple technique to highlight the generality of KE, and (2) an efficient technique for CNNs.

The simple technique is the ***we**ight-**l**evel* **s**plitting (WELS) technique. For every layer l, a binary mask M_l splits l into two exclusive parts: the fit-hypothesis

Algorithm 1: The KE training approach splits a dense network N, with L layers, into fit and reset hypotheses using a split-rate s_r and a binary mask M. Then, KE trains N for g generations. The network N has convolutional filters F, weight W, bias B, and batch norm Z. We assume a single fully connected layer for simplicity.

Result: Both a dense network N_g and a slim network $H_g^{\triangle}$ outlined by the split mask M
/* Set the split masks M for conv and FC layers once and for all. */
for *layer* l **to** L **do**
  if is_conv(l) **then**
    C_o, κ, _, C_i = F_l.shape;
    M_l = zeros($(C_o, \kappa, \kappa, C_i)$);
    if C_i == *3* **then**
      $M_l[: C_o \times s_r, :, :, :] = 1$; // First conv
    else
      $M_l[: C_o \times s_r, :, :, : C_i \times s_r] = 1$;
    end
  else if is_fc(l) **then**
    C_o, C_i = W_l.shape; // C_o=$\mathbb{C}$
    M_l = zeros((C_o, C_i));
    $M_l[:, : C_i \times s_r] = 1$;
  end
end
W, B, Z, F are initialized randomly;
for *generation* i **to** g **do**
  $N_i \leftarrow$ Train N for e epochs; // Learn W,B,Z,F
  for *layer* l **to** L **do**
    if is_conv(l) **then**
      F_l^r = rand(F_l.shape);
      $F_l = M_l F_l + (1 - M_l) F_l^r$;
    else if is_fc(l) **then**
      W_l^r = rand(W_l.shape);
      $W_l = M_l W_l + (1 - M_l) W_l^r$;
    end
  end
end

$H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$. Given a split-rate $0 < s_r < 1$, we *randomly* split the weights $W_l \in R^{|W_l|}$ using the mask $M_l \in \{0, 1\}^{|W_l|}$, where $|W_l|$ is the number of weights inside layer l and $\text{sum}(M_l) = s_r \times |W_l|$. The WELS technique supports a large spectrum of layers – fully connected, convolution, recurrent, and graph convolution. This highlights the generality of KE.

Through WELS, KE boosts the network performance across generations. However, WELS does not benefit from the connectivity of CNNs. Thus, we propose a splitting technique that not only boosts performance but also reduces inference cost for relatively small datasets. We leverage the CNNs' connectivity and outline the

fit-hypothesis $H^{\triangle}$ such that it is a slim (pruned) network as shown in Fig. 2.2. Instead of masking individual weights, we mask kernels, *i.e.*, ***ke**rnel-**l**evel convolutional-aware* **s**plitting (KELS) technique. Given a split-rate s_r and a convolutional filter $F_l \in R^{C_o \times \kappa \times \kappa \times C_i}$, KELS outlines the fit-hypothesis to include the first $\lceil s_r \times C_i \rceil$ kernels inside the first $\lceil s_r \times C_o \rceil$ filters as shown in Fig. 2.3. KELS guarantees matching dimensions between consequence convolutional filters. Thus, KELS integrates seamlessly in both vanilla CNNs (AlexNet and VGG) and modern architectures with residual links.

For relatively small datasets, the performance of the slim fit-hypothesis $H^{\triangle}$ reaches the performance of the dense network N. In these cases, $H^{\triangle}$ not only delivers the dense network's performance but also reduces the inference cost. Through KELS, the slim $H^{\triangle}$ runs on general purpose hardware, *i.e.*, neither sparse BLAS libraries nor specialized hardware [41] is required. Given a split rate s_r, KELS delivers a slim $H^{\triangle}$ that is equivalent to a dense network N with *approximately* $(1-s_r^2)$ sparsity. It is *approximate* because the network's end-points have s_r sparsity. The first convolutional layer operates on all input channels (*e.g.*, RGB) and fully connected layers have s_r sparsity. Algorithm 1 summarizes KE while applying the KELS technique.

2.3.3 Knowledge Evolution Intuitions

To understand KE, we give two complementary intuitions. These intuitions do not require the KELS technique. We use KELS for visualization purpose only

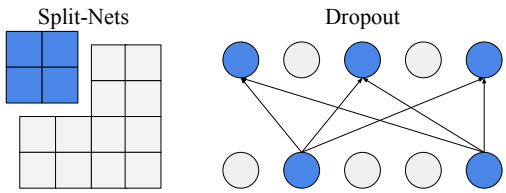


Figure 2.4: Split-Nets vs Dropout: The reset-hypothesis $H^{\triangledown}$ and dead neurons are highlighted in gray, while the fit-hypothesis $H^{\triangle}$ and "alive" neurons are highlighted in blue.

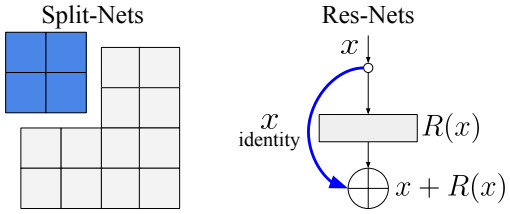


Figure 2.5: Split-Nets vs Res-Nets: Res-Nets split a network into an identity shortcut (blue) and a residual subnetwork $R(x)$. Split-Nets split a network into a fit-hypothesis $H^{\triangle}$ (blue) and a reset-hypothesis $H^{\triangledown}$. By splitting a network into two branches, Res-Net and Split-Net enable a zero-mapping in one of these branches ($R(x)$ and $H^{\triangledown}$) while keeping the network's depth intact.

(*e.g.*, Fig. 2.4). We empirically validate these intuitions in Sec. 2.5.

Intuition #1: Dropout

Dropout [126] randomly drops neurons during training as shown in Fig. 2.4. This encourages neurons to rely less on each other and to learn independent representations [21]. In KE, we drop the reset-hypothesis $H^{\triangledown}$ during re-initialization by randomly initializing $H^{\triangledown}$ before every generation. This encourages $H^{\triangle}$ to rely less on $H^{\triangledown}$ and to learn an independent representation. We validate this intuition by evaluating the performance of the slim $H^{\triangle}$ across generations. We observe that the performance of $H^{\triangle}$ increases as the number of generations increases.

Intuition #2: Residual Network

Res-Nets set the default mapping, between consecutive layers, to the identity as

shown in Fig. 2.5. Yet, from a different perspective, Res-Nets enable a zero-mapping in some subnetworks (residual links) without limiting the network's capacity [140, 147]. Similarly, KE enables a zero-mapping in the reset-hypothesis $H^{\triangledown}$ by re-using the fit-hypothesis $H^{\triangle}$ across generations. After the first generation N_1, $H^{\triangle}$ is always closer to convergence compared to $H^{\triangledown}$ that contains random values. Thus, KE encourages new generations to evolve the previous-generations' knowledge inside the fit-hypothesis $H^{\triangle}$ and suppress $H^{\triangledown}$.

We validate this intuition by measuring the mean absolute value inside both hypotheses. We observe that $H^{\triangle}$ and $H^{\triangledown}$ have comparable mean values at the first generation N_1. However, as the number of generations increases, the mean absolute value inside $H^{\triangle}$ increases and $H^{\triangledown}$ decreases. This supports our claim that KE promotes a zero-mapping inside the reset-hypothesis $H^{\triangledown}$.

Please note that Split-Nets have one degree of freedom that Res-Nets omit. Through the split-rate s_r, we control the size of the fit and reset hypotheses ($H^{\triangle}$ and $H^{\triangledown}$). If the training data is abundant, a large split-rate is better where a Split-Net reverts into a dense Res-Net. However, for relatively small datasets, a small split-rate s_r significantly reduces the inference cost while improving performance.

2.3.4 Evaluation Tasks

We evaluate KE using two supervised tasks: (1) classification and (2) metric learning. The performance of deep networks on small datasets is studied extensively using the classification task [132, 99, 26, 13, 91, 149, 156, 162, 153]. Thus, the

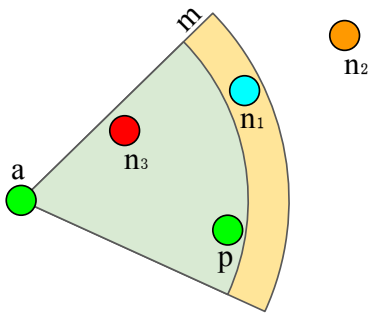


Figure 2.6: Triplet loss tuple (anchor, positive, negative) and margin m. The (h)ard, (s)emi-hard, and (e)asy negatives are highlighted in black, gray, and white.

classification task provides a rigorous performance benchmark. The metric learning evaluation highlights the flexibility of our approach and shows the generality of KE beyond mainstream literature that requires class logits.

We benchmark KE using both the cross-entropy and the triplet loss. We use these loss functions because most supervised tasks employ one of them.

Cross-Entropy (CE) Loss: We denote $x \in X$ as an input and $y \in Y = \{1, ..., \mathbb{C}\}$ as its ground-truth label. For a classification network N, CE is defined as follows

$$\text{CE}_{(x,y)} = -\log \frac{\exp\left(N(x;y)\right)}{\sum_{i=1}^{\mathbb{C}} \exp\left(N(x;i)\right)}, \tag{2.2}$$

where $N(x;y)$ denotes the output logit for class y given x.

Triplet Loss: A metric learning network learns an embedding where samples from the same class are close together, while samples from different classes are far apart. To train a metric learning network, we leverage triplet loss for its simplicity and efficiency. Triplet loss is defined as follows

$$\text{TL}_{(a,p,n)\in T} = \left[(D_{a,p} - D_{a,n} + m)\right]_+, \tag{2.3}$$

where $[\bullet]_+ = \max(0, \bullet)$, m is the margin between classes. $D_{x_1,x_2} = D(N(x_1), N(x_2))$; $N(\bullet)$ and $D(,)$ are the network's output-embedding and Euclidean distance, respectively. In Eq. 2.3, a, p, and n are the anchor, positive, and negative images in a triplet (a, p, n) from the triplets set T.

The performance of triplet loss relies heavily on the sampling strategy. Since we train randomly initialized networks, we leverage the semi-hard sampling strategy for its stability [118, 135]. In semi-hard negative sampling, instead of picking the hardest positive-negative samples, all anchor-positive pairs and their corresponding semi-hard negatives are considered. Semi-hard negatives are further away from the anchor than the positive exemplar yet within the banned margin m as shown in Fig. 2.6. Semi-hard negatives (n) satisfy Eq. 2.4

$$D_{a,p} < D_{a,n} < D_{a,p} + m. \tag{2.4}$$

2.4 Experiments

In this section, we evaluate KE using classification and metric learning tasks.

2.4.1 Knowledge Evolution on Classification

Datasets: We evaluate KE using five datasets: Flower-102 [92], CUB-200-2011 [142], FGVC-Aircraft [85], MIT67 [104], and Stanford-Dogs [63]. Table 2.1 summarizes the datasets' statistics.

Technical Details: We evaluate KE using two architectures: ResNet18 [44, 45] and

Table 2.1: Statistics of five classification datasets and their corresponding train, validation, and test splits.

	$\mathbb{C}$	Trn	Val	Tst	Total
Flower-102 [92]	102	1020	1020	6149	8189
CUB-200 [142]	200	5994	N/A	5794	11788
Aircraft [85]	100	3334	3333	3333	10000
MIT67 [104]	67	5360	N/A	1340	6700
Stanford-Dogs [63]	120	12000	N/A	8580	20580

DenseNet169 [58]. These architectures demonstrate the efficiency of KE on modern architectures. All networks are initialized randomly and optimized by stochastic gradient descent (SGD) with momentum 0.9 and weight decay 1e-4. We use cosine learning rate decay [82] with an initial learning rate $lr = 0.256$. We use batch size $b = 32$ and train N for $e = 200$ epochs. We use the standard data augmentation technique, *i.e.*, flipping and random cropping. For simplicity, we use the same training settings (lr, b, e) for all generations. We report the network accuracy at the last training epoch, *i.e.*, no early stopping.

Baselines: We benchmark KE using the cross-entropy (CE), label-smoothing (Smth) regularizer [91, 132], RePr [102], CS-KD [153], AdaCos [162], Dense-Sparse-Dense (DSD) [42], and Born Again Networks (BANs) [34] introduced in Sec. 2.2:

- **DSD** determines the duration of each training phase (# epochs) using the loss-convergence criterion. For small datasets, the loss converges rapidly to zero and some datasets do not have validation splits (see Table 2.1). So, we use $e = 200$, $e = 100$, and $e = 100$ epochs for the dense, sparse, dense phases, respectively. We prune each layer to the default 30% sparsity.
- **AdaCos** maximizes the inter-class angular margin by dynamically scaling the cosine similarities between training samples and their corresponding class

Table 2.2: Quantitative classification evaluation (Top-1 ↑) using ResNet18 with KELS. N_g denotes the performance of the g^{th} network generation. The first generation N_1 is both a baseline and a starting point for KE. As the number of generations increases, KE boosts performance.

Method	Flower	CUB	Aircraft	MIT	Dog
CE + AdaCos	**55.45**	**62.48**	**57.06**	56.25	**65.34**
CE + RePr	41.90	42.88	39.43	46.94	50.39
CE + DSD	51.39	53.00	57.24	53.21	63.58
CE + BANs-N_{10}	48.53	53.71	53.19	55.65	64.16
CE (N_1)	48.48	53.57	51.28	55.28	63.83
CE + KE-N_3 **(ours)**	52.53	56.73	52.53	57.44	64.28
CE + KE-N_{10} **(ours)**	56.15	58.11	53.21	**58.33**	64.56
Smth (N_1)	50.97	59.75	55.00	57.74	65.95
Smth + KE-N_3 **(ours)**	56.87	62.88	57.47	58.78	66.91
Smth + KE-N_{10} **(ours)**	**62.56**	**66.85**	**60.03**	**60.42**	**67.06**
CS-KD (N_1)	55.10	67.71	58.15	57.37	69.60
CS-KD + KE-N_3 **(ours)**	61.74	71.63	**59.97**	**58.41**	70.62
CS-KD + KE-N_{10} **(ours)**	**69.88**	**73.39**	59.08	57.96	**70.81**

center. Thus, AdaCos is a hyperparameter-free feature embedding regularizer.

- **CS-KD** is a knowledge distillation inspired approach that achieves state-of-the-art performance on small datasets. It distills the logits distribution between different samples from the same class. Thus, it mitigates overconfident predictions and reduces intra-class variations. We set CS-KD's hyperparameters $T = 4$ and $\lambda_{\text{cls}} = 3$ in all experiments.
- **RePr** is similar to DSD, but instead of pruning weights, RePr prunes *redundant* convolutional filters. Prakash *et al.* [102] recommend repeating the dense-sparse-dense phases three times. Since we train N for $e = 200$ epochs, we set RePr's hyperparameters $S1 = 50$ and $S2 = 10$. We use the default sparsity rate (prune rate) $p = 30\%$.

Results: Tables 2.2 and 2.3 present quantitative classification evaluation using ResNet18 and DenseNet169, respectively. For ResNet18, we use a split-rate $s_r = 0.8$

Table 2.3: Quantitative evaluation using DenseNet169 with WELS.

Method	Flower	CUB	Aircraft	MIT	Dog
CE + AdaCos	49.96	**62.20**	56.15	50.89	65.33
CE + RePr	39.75	47.01	36.04	49.77	55.63
CE + DSD	48.85	56.11	53.66	58.31	65.76
CE + BANs-N_{10}	44.92	57.30	52.56	57.66	65.49
CE (N_1)	45.85	55.16	51.73	56.62	64.82
CE + KE-N_3 **(ours)**	52.44	57.75	56.70	**59.67**	67.06
CE + KE-N_{10} **(ours)**	**60.15**	58.01	**59.73**	58.71	**67.75**
Smth (N_1)	46.34	59.93	57.74	57.81	65.12
Smth + KE-N_3 **(ours)**	55.46	**62.53**	62.86	**60.27**	**68.21**
Smth + KE-N_{10} **(ours)**	**64.18**	61.34	**65.86**	59.75	67.46
CS-KD (N_1)	46.97	67.32	58.87	56.62	69.83
CS-KD + KE-N_3 **(ours)**	59.36	69.77	59.91	**59.00**	**71.70**
CS-KD + KE-N_{10} **(ours)**	**65.27**	**70.36**	**61.22**	57.44	70.72

and KELS, *i.e.*, $\approx 36\%$ sparsity. For DenseNet169, we use $s_r = 0.7$ and WELS, *i.e.*, 30% sparsity. We report the performance of the dense network N because all baselines learn dense networks. In Sec. 2.5, we report the slim fit-hypothesis $H^{\triangle}$ performance and inference cost. Tables 2.2 and 2.3 present the performance of the first generation (N_1) as a baseline, the third generation (N_3) as the short-term benefit, and the tenth-generation (N_{10}) as the long-term benefit of KE.

A deeper network achieves higher accuracy when presented with enough training data. However, if the training data is scarce, a deeper network becomes vulnerable to overfitting. This explains why regularization techniques (*e.g.*, AdaCos) deliver competitive performance on the small ResNet18, but degrade on the large DenseNet169. Interestingly, KE remains resilient on the large DenseNet169 and delivers similar, if not superior, performance.

We applied KE on top of (1) the cross-entropy loss, (2) the label smoothing (Smth) regularizer with its hyperparameter [91] $\alpha = 0.1$, and (3) the CS-KD regularizer. KE is flexible and boosts performance on each baseline. N_3 outperforms N_1 on all datasets. After reaching a peak, KE's performance fluctuates. Thus, if N_3

outperforms N_{10} marginally, this indicates that KE reached its peak. In Fig. 2.1, KE reached its peak on CUB-200 after 20 generations, then KE fluctuates for 80 generations without degrading.

Even though RePr seems similar to KE, the following caveat explains RePr's inferior performance. RePr ranks the *redundant* filters across the entire network, *i.e.*, no per-layer ranking. Prakash *et al.* [102] report pruning more filters from deeper layers when training on large datasets. Yet, RePr prunes many filters from earlier layers when training on small datasets. The earlier layers get a small gradient compared to deeper layers; and with small datasets, the earlier filters remain close to their initialization, *i.e.*, no significant difference between earlier filters. Pruning earlier filters cripples the optimization process and achieves an inferior performance.

Another important difference between KE and RePr is how filters are re-initialized. KE re-initializes the reset-hypothesis randomly. Thus, KE makes no assumptions about the network architecture. In contrast, RePr is designed specifically for CNNs. RePr re-initializes the pruned filters to be orthogonal to both their values before being dropped and the current value of non-pruned filters. RePr uses the QR decomposition on the weights of the filters from the same layer to find the null-space, that is used to find an orthogonal initialization point. Basically, RePr stores the pruned filters to use them for re-initialization. This makes RePr more complex compared to KE.

Similar to KE, The BANs training approach trains a network for multiple generations. However, BANs transfers knowledge through the class-logits distribution. For small datasets, a teacher's logits distribution resembles the ground-truth labels

Table 2.4: Comparative evaluation between pretrained (CE + ImageNet) and randomly initialized (CS-KD + KE) networks. The performance of CE + ImageNet provides an upper-bound for KE.

Method	Flower	CUB	Aircraft	MIT	Dog
	ResNet18				
CE + ImageNet	**88.83**	**74.46**	**61.01**	**72.84**	**74.29**
CS-KD + KE-N_{10}	69.88	73.39	59.08	57.96	70.81
	DenseNet169				
CE + ImageNet	**93.46**	**80.73**	**69.85**	**77.90**	**79.92**
CS-KD + KE-N_{10}	65.27	70.36	61.22	57.44	70.72

(one-hot vector) when the loss converges to zero. Thus, BANs achieves regular cross-entropy performance even after training for 10 generations.

All previous experiments employ randomly initialized networks. Yet, pretrained networks achieve better performance on relatively small datasets. Table 2.4 highlights the performance gap between randomly initialized (CS-KD+KE) and ImageNet initialized (CE+ImageNet) networks. The CE+ImageNet baseline provides an upper bound. The CS-KD+KE baseline use KELS and $s_r = 0.8$ with ResNet18, and WELS and $s_r = 0.7$ with DenseNet169, *i.e.*, last rows in Tables 2.2 and 2.3. KE closes the performance gap between randomly initialized and ImageNet initialized networks significantly.

2.4.2 Knowledge Evolution on Metric Learning

Datasets: We evaluate KE using two standard metric learning datasets: CUB-200-2011 [142], Stanford Cars196 [69].

Evaluation Metrics: For quantitative evaluation, we use the Recall@K metric and Normalized Mutual Info (NMI) on the test split.

Table 2.5: Quantitative retrieval evaluation using standard metric learning datasets and architectures.

	ResNet50			GoogLeNet		
Datasets	NMI	R@1	R@4	NMI	R@1	R@4
CUB (N_1)	0.396	13.01	30.37	0.396	10.16	25.71
CUB + KE-N_3 **(ours)**	0.424	17.22	36.14	0.418	13.94	33.78
CUB + KE-N_{10} **(ours)**	**0.429**	**18.25**	**39.40**	**0.419**	**15.34**	**34.30**
Cars (N_1)	0.374	11.63	28.66	0.319	5.29	17.94
Cars + KE-N_3 **(ours)**	0.514	34.28	60.25	0.476	24.98	50.06
Cars + KE-N_{10} **(ours)**	**0.523**	**42.36**	**68.11**	**0.495**	**32.63**	**58.84**

Technical Details: We use the same hyperparameters (e, lr scheduler) and optimizer used in the classification experiments. However, the feature embedding $\in R^{d=128}$ is normalized to the unit circle and we use a batch size $b = 125$. Each mini-batch contains 25 different classes and 5 samples per class. We use a small learning rate $lr = 0.0256$ to avoid large fluctuations in the feature embedding during training.

Results: Table 2.5 presents a quantitative retrieval evaluation using two standard metric learning architectures: ResNet50 [44, 45] and GoogLeNet [131]. We use a split-rate $s_r = 0.8$ and KELS with both architectures. As the number of generations increases, the retrieval performance of the dense network increases. Through this experiment, we highlight how KE supports a large spectrum of network architectures and loss functions. Equipped with WELS, we expect KE to spread beyond CNNs. It is straight forward to tweak WELS and impose a regular sparsity, as in KELS, but for non CNNs.

Table 2.5 reports the retrieval performance using the dense network N. However, KELS delivers a slim $H^{\triangle}$ as well. Figures 2.7 and 2.8 present quantitative retrieval evaluation on CUB-200 and CARS196, respectively. Both figures leverage

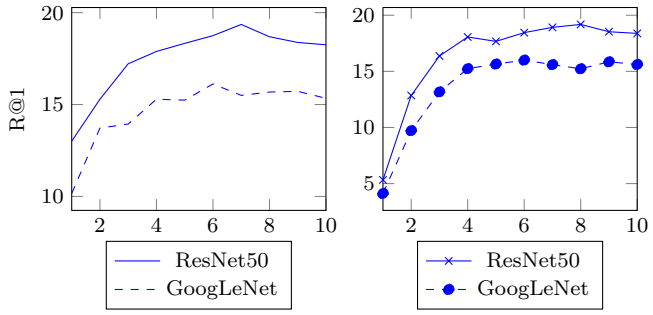


Figure 2.7: Quantitative retrieval evaluation using CUB-200 on both GoogLeNet and ResNet50. Both networks are trained for 10 generations. (Left) Recall@1 of the dense network N. (Right) Recall@1 of the slim fit-hypothesis $H^{\triangle}$.

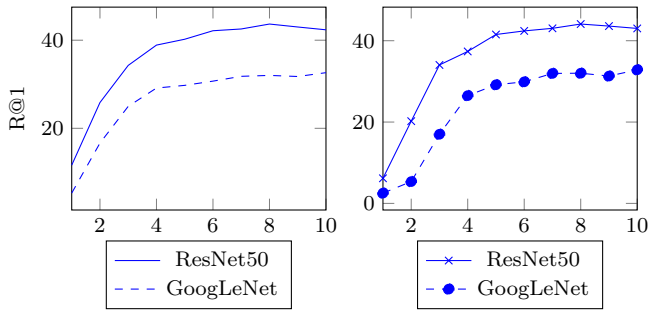


Figure 2.8: Quantitative retrieval evaluation using CARS196 on both GoogLeNet and ResNet50. (Left) Recall@1 of the dense network N. (Right) Recall@1 of the slim fit-hypothesis $H^{\triangle}$.

the R@1 metric for quantitative evaluation. We report the performance of both the dense network N and the slim fit-hypothesis $H^{\triangle}$. As the number of generations increases, the retrieval performance increases for both N and $H^{\triangle}$. Table 2.6 presents the fit-hypothesis $H^{\triangle}$ performance and inference cost. The fit-hypothesis $H^{\triangle}$ performance reaches the dense network N performance after $g = 10$ generations; yet, $H^{\triangle}$ achieves this performance at a significantly smaller inference cost.

Table 2.6: Quantitative evaluation for KELS using the number of both operations (G-Ops) and parameters (millions). $R1_g$ denotes the recall@1 performance at the g^{th} generation. $\blacktriangle_{\text{ops}}$ denotes the relative reduction in the number of operations. $\blacktriangle_{\text{r1}}$ denotes the absolute improvement margin on top of the dense baseline N_1.

	s_r	$R1_1$	$R1_{10}$	$\blacktriangle_{\text{r1}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
				CUB on GoogLeNet			
N_g	0.8	10.16	15.34	5.1%	3.00	-	11.44
$H_g^{\triangle}$		4.12	15.61	5.4%	1.98	34.0%	7.43
				CUB on ResNet50			
N_g	0.8	13.01	18.25	5.2%	8.19	-	47.48
$H_g^{\triangle}$		5.33	18.38	5.3%	5.32	35.0%	30.55
				CARS on GoogLeNet			
N_g	0.8	5.29	32.63	27.3%	3.00	-	11.44
$H_g^{\triangle}$		2.53	32.85	27.5%	1.98	34.0%	7.43
				CARS on ResNet50			
N_g	0.8	11.63	42.36	30.7%	8.19	-	47.48
$H_g^{\triangle}$		6.17	43.02	31.3%	5.32	35.0%	30.55

2.5 Ablation Study

This section presents six ablation studies: We (1) validate the dropout and Res-Net intuitions (from Sec. 2.3.3), (2) compare WELS and KELS techniques, (3) present the tradeoffs of the split-rate s_r, (4) evaluate the impact of changing the split-mask M across generations, (5) discuss why the improvement-margins of KE differ among datasets, and (6) evaluate KE on a large dataset, *i.e.*, ImageNet [22].

(1) Dropout and Res-Net intuitions' validation

To validate the dropout and Res-Net intuitions, we monitor the fit and reset hypotheses across generations. According to the dropout intuition, the fit-hypothesis should learn an independent representation. The KELS technique enables measuring the fit-hypothesis's performance. In this study, we use the CUB-200 dataset, VGG11_bn [121], and a split-rate $s_r = 0.5$. Fig. 2.9 (Top) shows the performance of the dense network N and the slim fit-hypothesis $H^{\triangle}$ for 10 generations. The hori-

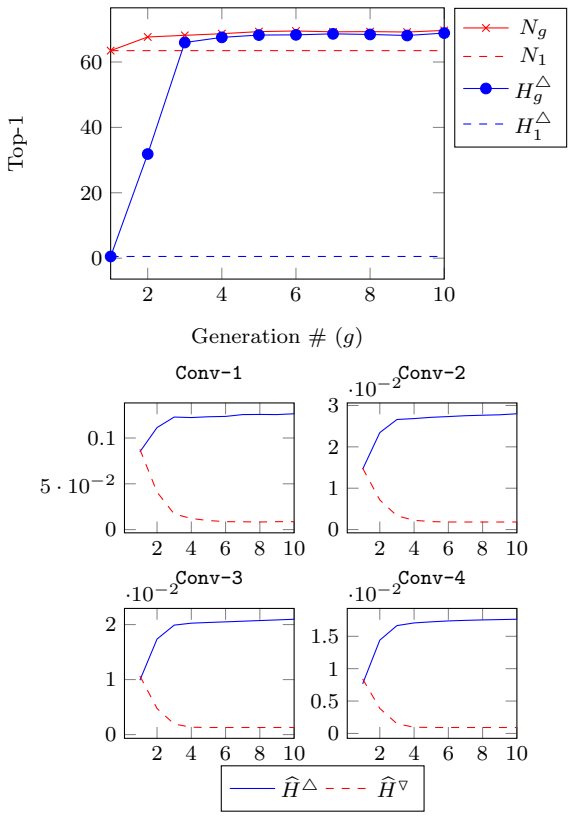


Figure 2.9: Quantitative classification evaluation using CUB-200 on VGG11_bn. The x-axis denotes the number of generations. The fit-hypothesis $H^{\triangle}$ achieves an inferior performance at $g = 1$, but its performance increases as the number of generations increases. $\widehat{H}^{\triangle}$ and $\widehat{H}^{\nabla}$ denote the mean absolute value inside $H^{\triangle}$ and H^{∇}.

zontal dashed lines denote the performance of the first generation (N_1 and $H_1^{\triangle}$). At the first generation, the fit-hypothesis's performance is inferior. Yet, as the number of generations increases, the fit-hypothesis performance increases. Table 2.7 (Top section) presents both the performance and inference cost of both N and $H^{\triangle}$.

According to the Res-Net intuition, the reset-hypothesis should converge to a zero-mapping because, after the first generation (N_1), the fit-hypothesis is always closer to convergence. Fig. 2.9 shows the mean absolute values ($\widehat{H}^{\triangle}$ and $\widehat{H}^{\nabla}$) inside the fit and reset hypotheses. We present these values inside the first four convolution

Table 2.7: Quantitative evaluation for KELS using the number of both operations (G-Ops) and parameters (millions). Acc_g denotes the classification accuracy at the g^{th} generation. $\blacktriangle_{\text{ops}}$ denotes the relative reduction in the number of operations. $\blacktriangle_{\text{acc}}$ denotes the absolute accuracy improvement on top of the dense baseline N_1.

	CUB on VGG11_bn						
	s_r	Acc_1	Acc_{10}	$\blacktriangle_{\text{acc}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
N_g	0.5	63.47	69.65	6.1%	15.22	-	259.16
$H_g^{\triangle}$		0.52	68.84	5.3%	3.85	74.7%	65.20
	FLW on ResNet18						
	s_r	Acc_1	Acc_{100}	$\blacktriangle_{\text{acc}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
N_g	0.8	53.87	75.62	21.7%	3.63	-	22.44
$H_g^{\triangle}$		6.41	75.62	21.7%	2.39	34.1%	14.43
N_g	0.5	52.62	74.60	21.9%	3.63	-	22.44
$H_g^{\triangle}$		0.37	74.60	21.9%	0.96	73.5%	5.64
	CUB on GoogLeNet						
	s_r	Acc_1	Acc_{10}	$\blacktriangle_{\text{acc}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
N_g	0.8	64.76	72.93	8.1%	3.00		11.59
$H_g^{\triangle}$		0.64	71.67	6.9%	1.98	34.0%	7.54
N_g	0.5	65.18	72.44	7.2%	3.00		11.59
$H_g^{\triangle}$		0.50	57.23	-7.9%	0.81	73.0%	3.00

layers of VGG11_bn. $\widehat{H}_1^{\triangle}$ and $\widehat{H}_1^{\triangledown}$ are comparable at N_1. However, as the number of generations increases, $\widehat{H}^{\triangle}$ increases while $\widehat{H}^{\triangledown}$ decreases.

(2) WELS vs KELS techniques

KE requires a network-splitting technique. WELS delivers a dense network N only. Thus, we compare WELS and KELS using N. Fig. 2.10 (Left) compares WELS and KELS using Flower-102, cross-entropy with the CS-KD regularizer [153], ResNet18, and two split-rates ($s_r = \{0.5, 0.8\}$). KELS and WELS achieve comparable performance. This is promising because WELS can be applied to any neural network. Fig. 2.10 (Right) re-assures that KELS delivers high performance while reducing inference cost as shown in Table 2.7 (middle section). The performance of $H_{100}^{\triangle}$ matches N_{100} because $H^{\triangle}$ has enough capacity for the small Flower-102. With $s_r = 0.5$, KELS achieves an absolute 21% improvement margin while reducing

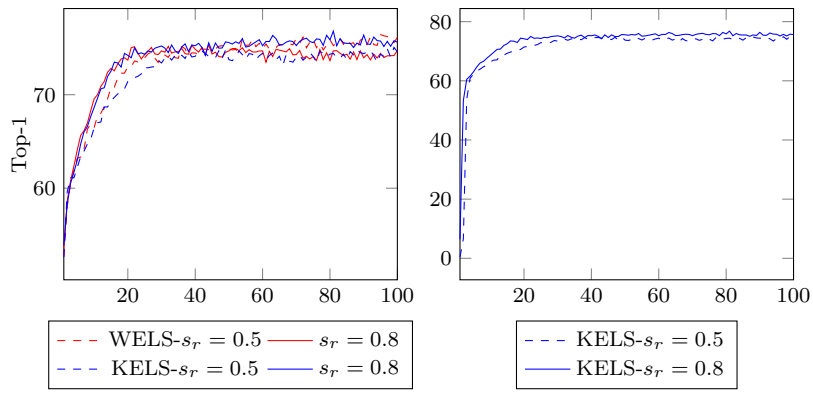


Figure 2.10: Quantitative evaluation for both KELS and WELS using Flower-102 on ResNet18 for 100 generations. The x and y axes denote the number of generations and the top-1 accuracy, respectively. (Left) The classification performance of the dense network N. (Right) The performance of the slim fit-hypothesis $H^{\triangle}$.

inference cost by 73%.

(3) The split-rate s_r tradeoffs

The split-rate s_r controls the size of the fit-hypothesis; a small s_r reduces the inference cost. Yet, a small s_r reduces the capacity of $H^{\triangle}$. Fig. 2.11 (Left) compares two split-rate ($s_r = \{0.5, 0.8\}$) using CUB-200 and GoogLeNet for 10 generations. Both split-rates achieve significant improvement margins on the dense network N. However, Fig. 2.11 (Right) shows that the large split-rate $s_r = 0.8$ helps the fit-hypothesis $H^{\triangle}$ to converge faster and to achieve better performance. Table 2.7 (third section) highlights this performance and inference-cost tradeoff. For a large dataset, a large split-rate is required to deliver a slim fit-hypothesis $H^{\triangle}$ with competitive performance.

(4) Changing the split-mask M across generations

In all previous experiments, we split the network using a split-mark M. The *same* mask is used to re-initialize every generation. However, we also highlighted the similarity between KE and dropout. Dropout does not drop the *same* neurons

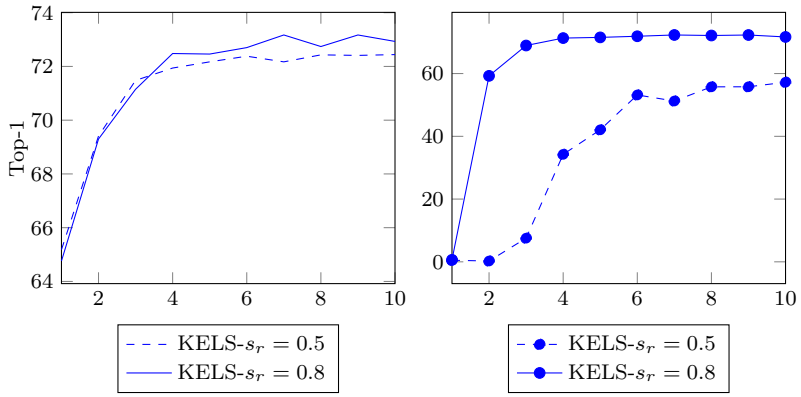


Figure 2.11: Quantitative evaluation for different split-rates using CUB-200 on GoogLeNet for 10 generations. (Left) The classification performance of the dense network N. (Right) The performance of the slim fit-hypothesis $H^{\triangle}$.

during training. Thus, we investigate the impact of changing the split-mask M across generations. This is possible with the WELS technique. In this experiment, We use CUB-200, ResNet18, label smoothing regularizer, the WELS technique, and four split-rates $s_r = \{0.2, 0.3, 0.5, 0.8\}$. We train N for 10 generations. After each generation, we re-initialize M randomly, *i.e.*, as if we initialize it for the first time. We refer to this WELS variant as WELS-Rand.

Fig. 2.12 compares WELS against WELS-Rand. With small split-rates ($s_r = \{0.2, 0.3\}$), WELS is significantly superior to WELS-Rand. However, as the split-rate increases ($s_r = \{0.5, 0.8\}$), both WELS and WELS-Rand become comparable. This happens because different fit-hypotheses, in WELS+Rand, overlap partially. Given a split-rate s_r, a network-weight belongs to two consecutive fit-hypotheses with probability s_r^2. Accordingly, WELS-Rand with a small s_r flushes the entire knowledge of a parent network. In contrast, WELS-Rand with a large split-rate retains the parent-network's knowledge at least partially.

(5) Why the improvement margins $\blacktriangle_{\mathbf{acc}}$ of KE differ?

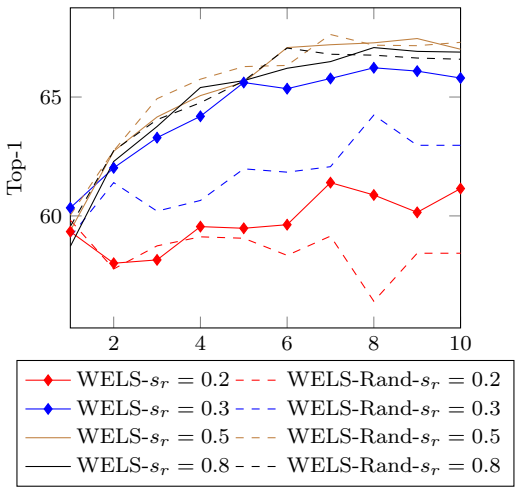


Figure 2.12: Comparative evaluation between WELS and WELS-Rand. WELS uses the same binary mask M across all generations. In contrast, WELS-Rand randomly re-initialize M after every generation. With a small split-rate, WELS-Rand flushes the parent-networks' knowledge.

Table 2.8: The KE's improvement margins $\blacktriangle_{\text{acc}}$ versus the FCAMD accuracies on each dataset. There is a strong positive Pearson correlation ($r = 0.9529$) between $\blacktriangle_{\text{acc}}$ and the datasets' simplicity (FCAMD's accuracies).

Datasets	$\blacktriangle_{\text{acc}}$	FCAMD Acc
Flower	14.78	63.06
CUB	5.68	19.60
Aircraft	0.93	15.80
MIT	0.59	19.20
Stanford Dogs	1.21	13.20
		$r = 0.952$

In deep learning, we assume that more training data leads to better accuracy. However, the KE's improvement margins $\blacktriangle_{\text{acc}}$ contradict this assumption. For instance, Table 2.2 shows that $\blacktriangle_{\text{acc}}$ on Flower-102 is bigger than $\blacktriangle_{\text{acc}}$ on CUB-200, *i.e.*, 14.78 vs 5.68 after 10 generations with the CS-KD regularizer. Fig. 2.1 also emphasizes this behavior; Flower-102 is a much smaller dataset compared to CUB-200, yet $\blacktriangle_{\text{acc}}$ is over 20% for Flower-102 but less than 10% for CUB-200. We posit that $\blacktriangle_{\text{acc}}$ depends not only on the dataset size, but also on the dataset simplicity.

To evaluate our postulate, we quantify the simplicity of our five datasets

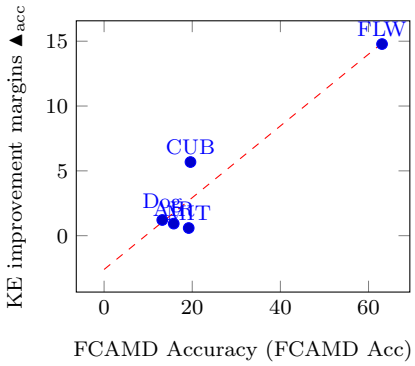


Figure 2.13: The average accuracy of the Flower (FLW), CUB, Aircraft (AIR), MIT, and Dog datasets inside the FCAMD dataset. The five datasets are equally represented inside FCAMD, *i.e.*, 50 classes each and 10 images per class. The accuracy metric reflects the simplicity of each dataset. The x-axis denotes the accuracy of a dataset inside FCAMD and the y-axis denotes the KE improvement margins. There is a strong positive correlation between the datasets' simplicity and the KE improvement margins.

(**F**lower, **C**UB, **A**ircraft, **MI**T, and **D**og). We create a new dataset, dubbed FCAMD, using the five datasets. We randomly sample 50 classes from each dataset. For each class, we randomly sample 10 training and 10 testing images. Thus, FCAMD has 2500 training and 2500 testing images, *i.e.*, 250 classes, 10 training images per class. We train a ResNet18 from scratch on FCAMD. To quantify the simplicity of each dataset, we measure the average accuracy of its 50 classes. Higher accuracy indicates a simpler dataset. There is a strong positive Pearson correlation ($r = 0.9529$) between the datasets' simplicity (from FCAMD's accuracies) and the KE improvement margins $\blacktriangle_{\text{acc}}$ as shown in Fig. 2.13 and Table 2.8. To compute the Pearson correlation, we use the KE improvement margins $\blacktriangle_{\text{acc}}$ achieved after 10 generations on top of the CS-KD [153] baseline, *i.e.*, $\blacktriangle_{\text{acc}}$ from the last section of Table 2.2. Even if we dismissed Flower-102 as an outlier, the correlation would become $r = 0.494$ for the remaining four datasets (CUB, AIR, MIT, and Dog).

Table 2.9: The KE's improvement margins ▲$_{acc}$ versus the accuracies of a *fine-tuned* ResNet18. There is a strong positive Pearson correlation ($r = 0.850$) between ▲$_{acc}$ and the datasets' simplicity (fine-tuned ResNet18 accuracies).

Datasets	▲$_{acc}$	Fine-tuned ResNet18
Flower	14.78	88.83
CUB	5.68	74.46
Aircraft	0.93	61.01
MIT	0.59	72.84
Stanford Dogs	1.21	74.29
		$r = 0.850$

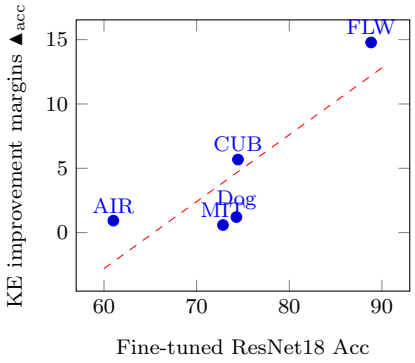


Figure 2.14: The accuracy of the Flower (FLW), CUB, Aircraft (AIR), MIT, and Dog datasets on a *fine-tuned* ResNet18. The accuracy metric reflects the simplicity of each dataset. The x-axis denotes the accuracy of a dataset on a *fine-tuned* ResNet18 and the y-axis denotes the KE improvement margins. There is a strong positive correlation between the datasets' simplicity and the KE improvement margins.

Another way to quantify the simplicity of a dataset is through a pretrained network. A pretrained network contains the ImageNet's knowledge. This large knowledge mitigates the impact of both a small dataset size and a small number of samples per class. Thus, we fine-tune a pretrained ResNet18 on the five datasets as shown in Table 2.4. The accuracy of the fine-tuned ResNet18 reflects the simplicity of each dataset. Higher accuracy indicates a simpler dataset. Again, there is a strong positive Pearson correlation ($r = 0.850$) between ▲$_{acc}$ and the fine-tuned ResNet18 accuracies as shown in Fig. 2.14 and Table 2.9.

The FCAMD and fine-tuned ResNet18 experiments present an interesting find-

ing. It seems that the dataset size is no longer the dominant factor that controls the performance of a randomly initialized network on relatively small datasets.

(6) Evaluate KE on ImageNet

Our paper tackles the following question: how to train a deep network on a relatively small dataset? Answering this question will have a significant impact on both academia and industry. However, it is important to understand how KE behaves on a large dataset, *i.e.*, ImageNet. The goal of this experiment is *not* to boost performance on ImageNet; Stock *et al.* [127] and Beyer *et al.* [5] deliver strong arguments why boosting performance on ImageNet should no longer be an ultimate goal. While KE boosts performance on ImageNet, our goal is to monitor the performance of the fit-hypothesis. We want to answer the following question: can KE evolve knowledge inside the fit-hypothesis even when presented with a large dataset?

Technical Details: We train a ResNet18 for 5 generations using KELS and a split-rate $s_r = 0.8$, *i.e.*, $\approx 36\%$ sparsity. Our implementation for ImageNet follows the practice in [44]. We use a batch size $b = 128$, and a step learning rate scheduler with a starting $lr = 0.1$. We train for $e = 150$ epochs per generation. Other parameters (*e.g.*, momentum, optimizer) are the same as those reported in Sec. 2.4.1.

Results: Fig. 2.15 presents a quantitative classification evaluation using ImageNet. KE boosts performance for both the dense network N and the slim fit-hypothesis $H^{\triangle}$. In Sec. 2.4.1, we evaluate KE using relatively small datasets and large architectures. In contrast, this experiment evaluates KE using a large dataset and a small architecture. Accordingly, these improvement margins on ImageNet are a

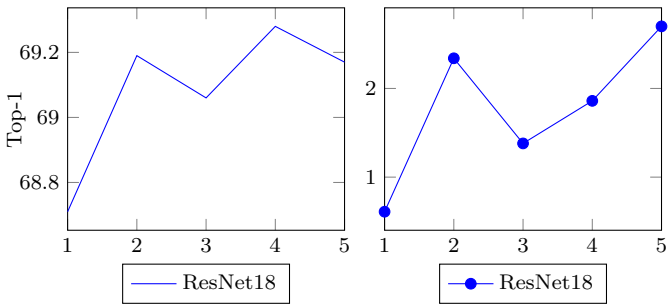


Figure 2.15: Quantitative classification evaluation using ImageNet on ResNet18 for 5 generations. (Left) The accuracy performance (Top-1 ↑) of the dense network N. (Right) The performance of the slim fit-hypothesis $H^{\triangle}$.

Table 2.10: Quantitative classification evaluation using both ResNet34 and ResNet50. N_g and $H_g^{\triangle}$ denote the performance of the dense network N and the fit-hypothesis $H^{\triangle}$ at the g^{th} generation. $\blacktriangle_H$ denotes the absolute improvement margin in the fit-hypothesis relative to the baseline $H_1^{\triangle}$

	ResNet34			ResNet50		
g	N_g	$H_g^{\triangle}$	$\blacktriangle_H$	N_g	$H_g^{\triangle}$	$\blacktriangle_H$
1	72.51	0.28	-	74.54	0.20	-
2 **(ours)**	**72.86**	1.25	0.97	74.78	3.44	3.24
3 **(ours)**	72.78	2.27	1.99	75.01	6.71	6.51
4 **(ours)**	**72.86**	1.96	1.68	75.15	4.63	4.43
5 **(ours)**	**72.86**	**4.49**	4.21	**75.27**	**13.81**	13.61

lower-bound on the potential of KE. As the architecture gets bigger, these improvement margins will increase. Accordingly, we conclude that KE can evolve knowledge inside the fit-hypothesis.

We further evaluate KE on two larger architectures. Table 2.10 presents quantitative classification evaluation using ResNet34 and ResNet50. We use the same technical details from the ResNet18 experiment. KE boosts performance on the fit-hypothesis $H^{\triangle}$ consistently. This confirms our finding that KE evolves knowledge in the fit-hypothesis $H^{\triangle}$.

2.5.1 Discussion

ImageNet [22] will eventually become a toy dataset given the increasing size of deep networks [20, 129, 84, 7] (*e.g.*, GPT-3). To train these large networks, unsupervised [61, 10] and self-supervised [144, 97, 46, 136] learning mitigate the burden of data annotation. However, these learning approaches still require storing and maintaining a large corpus of data. This is (1) expensive even if neither labeling nor curating is required, (2) impractical for applications with privacy concerns like medical imaging. KE tackles the problem of training deep networks on relatively small datasets. KE's main limitation is the training time. It takes $\approx$ 8 hours to train 100 generations, 200 epochs each, on Flower-102 using GTX1080Ti GPU. This long training time can be reduced by monitoring the performance on a validation split.

2.6 Conclusion

We have proposed knowledge evolution (KE) to train deep networks on relatively small datasets. KE picks a random subnetwork (fit-hypothesis), with inferior performance, and evolves its knowledge. We have equipped KE with a kernel-level convolution-aware splitting (KELS) technique to learn a slim network automatically while training a dense network. Through KELS, KE reduces the inference cost while boosting performance. Through the weight-level splitting (WELS) technique, KE supports a large spectrum of architectures. We evaluated KE using classification and metric learning tasks. KE achieves SOTA results.

Chapter 3: SVMax: A Feature Embedding Regularizer

A neural network regularizer (*e.g.*, weight decay) boosts performance by *explicitly* penalizing the complexity of a network. In this paper, we penalize inferior network activations – feature embeddings – which in turn regularize the network's weights *implicitly*. We propose singular value maximization (SVMax) to learn a more uniform feature embedding. The SVMax regularizer supports both supervised and unsupervised learning. Our formulation mitigates model collapse and enables larger learning rates. We evaluate the SVMax regularizer using both retrieval and generative adversarial networks. We leverage a synthetic mixture of Gaussians dataset to evaluate SVMax in an unsupervised setting. For retrieval networks, SVMax achieves significant improvement margins across various ranking losses. Code available at *https://bit.ly/3jNkgDt*

3.1 Introduction

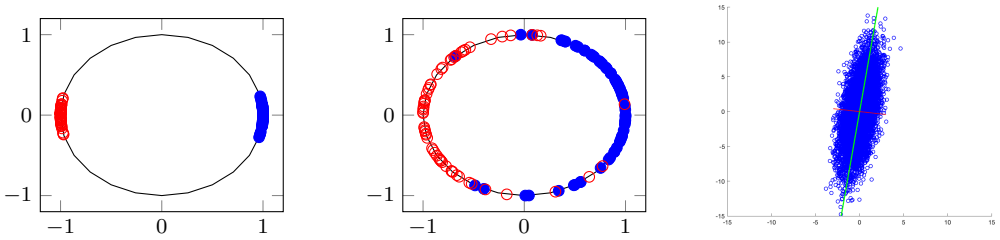
A neural network's knowledge is embodied in *both* its weights and activations. This difference manifests in how network pruning and knowledge distillation tackle the model compression problem. While pruning literature [73, 83, 151] compresses models by removing less significant weights, knowledge distillation [49] reduces com-

putational complexity by matching a cumbersome network's last layer activations (logits). This perspective, of weight-knowledge versus activation-knowledge, emphasizes how neural network literature is dominated by explicit weight regularizers. In contrast, this paper leverages singular value decomposition (SVD) to regularize a network through its last layer activations – its feature embedding.

Our formulation is inspired by principal component analysis (PCA). Given a set of points and their covariance, PCA yields the set of orthogonal eigenvectors sorted by their eigenvalues. The principal component (first eigenvector) is the axis with the highest variation (largest eigenvalue) as shown in Figure 3.1c. The eigenvalues from PCA, and similarly the singular values from SVD, provide insights about the feature embedding structure. As such, by regularizing the singular values, we reshape the feature embedding.

The main contribution of this paper is to leverage the singular value decomposition of a network's activations to regularize the feature embedding. We achieve this objective through singular value maximization (SVMax). The SVMax regularizer is oblivious to both the input-class (labels) and the sampling strategy. Thus it promotes a uniform feature embedding in both supervised and unsupervised learning. Furthermore, we present a mathematical analysis of the mean singular value's lower and upper bounds. This analysis makes tuning the SVMax's balancing-hyperparameter easier, when the feature embedding is normalized to the unit circle.

SVMax promotes a uniform feature embedding. During training, SVMax speeds up convergence by enabling large learning rates. The SVMax regularizer



(a) Singular values: (11.25, 1.14) (b) Singular values: (8.99, 6.86) (c) PCA of a toy dataset

Figure 3.1: Feature embeddings scattered over the $2D$ unit circle. In (a), the features are polarized across a single axis; the singular value of the principal (horizontal) axis is large while the singular value of the secondary (vertical) axis is small, respectively. In (b), the features spread uniformly across both dimensions; both singular values are comparably large. (c) depicts the PCA analysis of a toy 2D Gaussian dataset to demonstrate our intuition. The principal component (green) has the highest eigenvalue, *i.e.*, the axis with the highest variation, while the second component (red) has a smaller eigenvalue. Maximizing all eigenvalues promotes data dispersion across all dimensions. In this paper, we maximize the mean singular value to regularize the feature embedding and avoid a model collapse.

integrates seamlessly with various ranking losses. We apply the SVMax regularizer to the last feature embedding layer, but the same formulation can be applied to intermediate layers. The SVMax regularizer mitigates model collapse in both retrieval networks and generative adversarial networks (GANs) [36, 125, 88]. Furthermore, the SVMax regularizer is useful when training self/un-supervised feature embedding networks with a contrastive loss (*e.g.*, CPC) [94, 97, 46, 136].

In summary, we propose singular value maximization to regularize the feature embedding. In addition, we present a mathematical analysis of the mean singular value's lower and upper bounds to reduce hyperparameter tuning (Sec. 3.3). We quantitatively evaluate how SVMax significantly boosts the performance of ranking losses (Sec. 3.4.1). And we provide a qualitative evaluation of using SVMax in the unsupervised learning setting via GAN training (Sec. 3.4.3).

3.2 Related Work

Network weight regularizers dominate the deep learning regularizer literature because they support a large spectrum of tasks and architectures. Singular value decomposition (SVD) has been applied as a weight regularizer in several recent works [155, 119, 39]. Zhang et al. [155] employ SVD to avoid vanishing and exploding gradients in recurrent neural networks. Similarly, Guo and Ye [39] bound the singular values of the convolutional layer around 1 to preserve the layer's input and output norms. A bounded output norm mitigates the exploding/vanishing gradient problem. Weight regularizers share the common limitation that they do not enforce an explicit feature embedding objective and are thus ineffective against model collapse.

Feature embedding regularizers have also been extensively studied, especially for classification networks [110, 145, 47, 52, 135]. These regularizers aim to maximize class margins, class compactness, or both simultaneously. For instance, Wen et al. [145] propose center loss to explicitly learn class representatives and thus promote class compactness. In classification tasks, test samples are assumed to lie within the same classes of the training set, *i.e.*, closed-set identification. However, retrieval tasks, such as product re-identification, assume an open-set setting. Because of this, a retrieval network regularizer should aim to spread features across many dimensions to fully utilize the expressive power of the embedding space.

Recent literature [113, 159] has recognized the importance of a spread-out feature embedding. However, this literature is tailored to triplet loss and therefore assumes a particular sampling procedure. In this paper, we leverage SVD as a reg-

ularizer because it is simple, differentiable [60], and class oblivious. SVD has been used to promote *low* rank models to learn compact intermediate layer representations [67, 117]. This helps compress the network and speed up matrix multiplications on embedded devices (iPhone and Raspberry Pi). In contrast, we regularize the embedding space through a *high* rank objective. By maximizing the mean singular value, we promote a higher rank representation – a spread-out embedding.

3.3 Singular Value Maximization (SVMax)

We first introduce our mathematical notation. Let $\mathcal{I}$ denote the image space and $E_{\mathcal{I}} \in R^d$ denote the feature embeddings space, where d is the dimension of the features. A feature embedding network is a function $F_\theta : \mathcal{I} \rightarrow E_{\mathcal{I}}$, parameterized by the network's weights θ. We quantify similarity between an image pair $(\mathcal{I}_1, \mathcal{I}_2)$ via the Euclidean distance in feature space, *i.e.*, $\|E_{\mathcal{I}_1} - E_{\mathcal{I}_2}\|_2$.

During training, a $2D$ matrix $E \in R^{b \times d}$ stores b samples' embeddings, where b is the mini-batch size. Assuming $b \geq d$, the singular value decomposition (SVD) of E provides the singular values $S = [s_1, ., s_i, ., s_d]$, where s_1 and s_d are the largest and smallest singular values, respectively. We maximize the mean singular value, $s_\mu = \frac{1}{d} \sum_{i=1}^{d} s_i$, to regularize the network's last layer activations – the feature embedding. By maximizing the mean singular value, the deep network spreads out its embeddings. This has the added benefit of implicitly regularizing the network's weights θ. The proposed SVMax regularizer integrates with both supervised and

unsupervised feature embedding networks as follows

$$L_{\text{NN}} = L_r - \lambda \frac{1}{d} \sum_{i=1}^{d} s_i = L_r - \lambda s_\mu, \tag{3.1}$$

where L_r is the original loss and λ is a hyperparameter.

Lower and Upper Bounds of the Mean Singular Value: One caveat to equation 3.1 is the hyperparameter λ. It is difficult to tune since the mean singular value s_μ depends on the range of values inside E and its dimensions (b, d). Thus, changing the batch size or embedding dimension requires a different λ. To address this, we constrain the embeddings to lie on the unit circle (L2-normalized) – a common assumption in metric learning. This provides *both* lower and upper bounds on ranking losses. This will also allow us to impose lower and upper bounds on s_μ.

For an L2-normalized embedding E, the largest singular value s_1 is maximum when the matrix-rank of E equals one, *i.e.*, $rank(E) = 1$, and $s_i = 0$ for $i \in [2, d]$. Horn and Johnson [53] provide an upper bound on this largest singular value s_1 as $s^*(E) \leq \sqrt{||E||_1 ||E||_\infty}$. This holds in equality for all L2-normalized $E \in R^{b \times d}$ with $rank(E) = 1$. For an L2-normalized matrix E with $||E||_1 = b$, and $||E||_\infty = 1$, this gives:

$$s^*(E) = \sqrt{||E||_1 ||E||_\infty} = \sqrt{b}. \tag{3.2}$$

Thus, the lower bound L on s_μ is $L = \frac{s^*(E)}{d} = \frac{\sqrt{b}}{d}$.

Similarly, an upper bound is defined on the sum of the singular values [137, 68,

32]. This summation is formally known as the nuclear norm of a matrix $||E||_*$. Hu [55] established an upper bound on this summation using the Frobenius Norm $||E||_F$ as follows

$$||E||_* \leq \sqrt{\frac{b \times d}{max(b,d)}}||E||_F, \tag{3.3}$$

where $||E||_F = \left(\sum_{i=1}^{rows}\sum_{j=1}^{cols}|E_{ij}|^2\right)^{\frac{1}{2}} = \sqrt{b}$ because of the unit-circle assumption.

Accordingly, the lower and upper bounds of s_μ are $[L,U] = [\frac{s^*(E)}{d}, \frac{||E||_*}{d}]$. With these bounds, we rewrite our final loss function as follows

$$L_{\text{NN}} = L_r + \lambda \exp\left(\frac{U - s_\mu}{U - L}\right). \tag{3.4}$$

The SVMax regularizer grows exponentially $\in [1, e]$. We employ this loss function in all our retrieval experiments. It is important to note that the L2-normalized assumption makes λ tuning easier, but it is not required. Equation 3.4 makes the hyperparameter λ only dependent on the range of L_r which is also bounded for ranking losses.

Lower and Upper Bounds of Ranking Losses: We briefly show that ranking losses are bounded when assuming an L2-normalized embedding. Equations 3.5 and 3.6 show triplet and contrastive losses, respectively, and their corresponding bounds $[L,U]$.

$$\text{TL}_{(a,p,n)\in T} = [(D_{a,p} - D_{a,n} + m)]_+ \xrightarrow{[L,U]} \quad [0, 2+m], \tag{3.5}$$

$$\text{CL}_{(x,y)\in P} = \delta_{x,y}D_{x,y} + (1-\delta_{x,y})[m - D_{x,y}]_+ \xrightarrow{[L,U]} \quad [0,2], \tag{3.6}$$

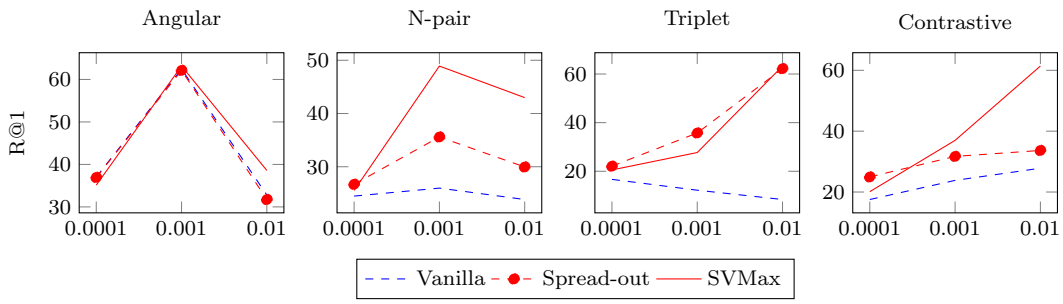


Figure 3.2: Quantitative evaluation on Stanford CARS196. X and Y-axis denote the learning rate lr and recall@1 performance, respectively.

where $[\bullet]_+ = max(0, \bullet)$, $m < 2$ is the margin between classes, since 2 is the maximum distance on the unit circle. $D_{x_1,x_2} = D(N(x_1), N(x_2))$; $N(\bullet)$ and $D(\bullet, \bullet)$ are the network's output-embedding and Euclidean distance, respectively. In equation 3.5, a, p, and n are the anchor, positive, and negative images in a single triplet (a, p, n) from the triplets set T.

3.4 Experiments

In this section, we evaluate SVMax using both supervised and unsupervised learning. We leverage retrieval (Sec. 3.4.1) and self-supervised learning (Sec. 3.4.2) for quantitative evaluation. Then, we leverage generative adversarial networks (Sec. 3.4.3) for qualitative evaluation.

3.4.1 Retrieval Networks

Technical Details: We evaluate SVMax quantitatively using three datasets: CUB-200-2011 [142], Stanford CARS196 [69], and Stanford Online Products [95]. We use GoogLeNet [131] and ResNet50 [44]; both pretrained on ImageNet [22] and fine-

Table 3.1: Quantitative evaluation on CUB-200-2011 with batch size $b = 144$, embedding dimension $d = 128$ and multiple learning rates $lr = \{0.01, 0.001, 0.0001\}$. $\triangle_{R@1}$ column indicates the R@1 improvement margin relative to the vanilla ranking loss. A large learning rate lr increases the chance of model collapse, while a small lr slows convergence. λ is dependent on the ranking loss.

	$lr = 0.01$				$lr = 0.001$				$lr = 0.0001$			
Method	NMI	R@1	R@8	$\triangle_{R@1}$	NMI	R@1	R@8	$\triangle_{R@1}$	NMI	R@1	R@8	$\triangle_{R@1}$
						Contrastive						
Vanilla	0.435	25.73	58.88	-	0.443	28.68	64.70	-	0.413	24.49	59.54	-
Spread-out	0.440	24.54	57.16	-1.18	0.479	32.12	66.83	3.44	**0.458**	**31.85**	**67.45**	7.36
SVMax (Ours)	**0.527**	**41.26**	**75.24**	15.53	**0.547**	**43.11**	**77.26**	14.43	0.449	29.56	65.50	5.06
						Triplet Loss						
Vanilla	0.496	29.34	67.96	-	0.477	28.88	64.60	-	0.449	24.86	61.14	-
Spread-out	0.545	43.60	76.98	14.26	**0.557**	**44.02**	**78.54**	15.14	0.435	**28.33**	64.33	3.46
SVMax $\lambda = 1$ (Ours)	**0.556**	43.21	77.43	13.88	0.527	39.13	74.17	10.25	0.401	25.07	60.01	0.20
SVMax $\lambda = 0.1$ (Ours)	0.547	**43.80**	**77.97**	14.47	**0.557**	43.89	78.44	15.01	**0.436**	28.22	**64.40**	3.36
						N-pair						
Vanilla	0.402	18.96	50.32	-	0.452	27.65	63.10	-	0.455	31.41	66.95	-
Spread-out	0.416	20.64	52.80	1.69	0.483	32.46	66.41	4.81	0.474	33.39	68.80	1.98
SVMax (Ours)	**0.483**	**34.62**	**68.11**	15.67	**0.547**	**43.79**	**77.31**	16.14	**0.488**	**34.13**	**69.92**	2.72
						Angular						
Vanilla	0.470	28.54	60.03	-	0.508	38.94	72.82	-	**0.538**	41.80	76.18	-
Spread-out	.471	28.29	60.26	-0.25	0.508	38.96	72.86	0.02	**0.538**	41.81	76.23	0.02
SVMax (Ours)	**0.487**	**32.88**	**66.27**	4.34	**0.523**	**41.29**	**74.71**	2.35	0.531	**42.00**	**76.30**	0.20

tuned for K iterations. These are standard retrieval datasets and architectures. By default, the embedding $\in R^{d=128}$ is normalized to the unit circle. In all experiments, a batch size $b = 144$ is employed, the learning rate lr is fixed for $K/2$ iterations then decayed polynomially to $1e-7$ at iteration K. We use the SGD optimizer with 0.9 momentum. Each batch contains p different classes and l different samples per class. For example, triplet loss employs $p = 24$ different classes and $l = 6$ instances per class. The mini-batch of N-pair loss contains 72 classes and a single positive pair per class, *i.e.*, $p = 72$ and $l = 2$. This same mini-batch setting is used for angular loss. For contrastive loss, $p = 36$ and $l = 4$ are divided into 72 positive and 72 negative pairs. For both CUB-200 and CARS196, $K = 5,000$ iterations; for Stanford Online Products, $K = 20,000$.

Table 3.2: Quantitative evaluation on Stanford Online Products.

Method	$lr = 0.01$				$lr = 0.001$				$lr = 0.0001$			
	NMI	R@1	R@8	$\triangle_{R@1}$	NMI	R@1	R@8	$\triangle_{R@1}$	NMI	R@1	R@8	$\triangle_{R@1}$
						Contrastive						
Vanilla	0.816	18.23	34.07	-	0.820	28.70	43.27	-	0.813	34.30	48.49	-
Spread-out	0.811	18.87	35.74	0.64	0.822	29.97	46.69	1.27	0.824	36.15	51.22	1.85
SVMax (Ours)	**0.875**	**61.82**	**78.90**	43.59	**0.854**	**53.94**	**70.92**	25.25	**0.832**	**41.96**	**57.44**	7.66
						Triplet Loss						
Vanilla	**0.891**	**71.96**	**86.24**	-	**0.873**	64.09	80.07	-	**0.840**	46.29	62.57	-
Spread-out	0.890	71.60	85.73	-0.36	0.872	**64.23**	80.10	0.14	**0.840**	**46.68**	**63.04**	0.39
SVMax $\lambda = 1$ (Ours)	0.868	63.82	80.95	-8.15	0.857	58.04	75.14	-6.04	0.836	44.62	60.76	-1.67
SVMax $\lambda = 0.1$ (Ours)	0.889	71.48	85.97	-0.49	0.872	**64.23**	**80.14**	0.14	**0.840**	46.64	62.95	0.35
						N-pair						
Vanilla	0.798	12.86	24.53	-	0.815	23.83	38.97	-	0.818	33.98	48.56	-
Spread-out	0.803	16.58	31.91	3.72	0.824	32.88	50.34	9.05	0.825	37.39	52.55	3.40
SVMax (Ours)	**0.871**	**57.76**	**76.05**	44.90	**0.858**	**54.70**	**71.57**	30.87	**0.835**	**43.04**	**58.78**	9.06
						Angular						
Vanilla	0.883	62.83	80.13	-	**0.885**	66.93	82.12	-	**0.856**	54.29	71.14	-
Spread-out	0.883	62.73	79.96	-0.10	**0.885**	66.91	82.09	-0.02	**0.856**	54.30	71.10	0.02
SVMax (Ours)	**0.885**	**65.44**	**81.73**	2.61	0.884	**67.28**	**82.47**	0.35	0.855	**54.88**	**71.47**	0.59

Baselines: We evaluate SVMax using contrastive [40], hard triplet [51, 48], N-pair [124] and angular [143] losses. We use the margin $m = 1$ for contrastive loss, $m = 0.2$ for triplet loss, and the angle bound $\alpha = 45°$ for angular loss. Similar to SVMax, multiple regularizers [72, 159, 117, 12] promote a uniform embedding space. Unlike SVMax, these regularizers require a supervised setting to push anchor-negative pairs apart. We employ the spread-out regularizer [159] as a baseline for its simplicity, with default hyperparameter $\alpha = 1$. To enable the spread-out regularizer on non-triplet ranking losses, we pair every anchor with a random negative sample from the training mini-batch.

Evaluation Metrics: For quantitative evaluation, we use the Recall@K metric and Normalized Mutual Info (NMI) on the test split.

The hyperparameter: $\lambda = 1$ for both contrastive and N-pair losses, $\lambda = 0.1$ for triplet loss, and $\lambda = 2$ for angular loss. We fix λ across datasets, architectures, and

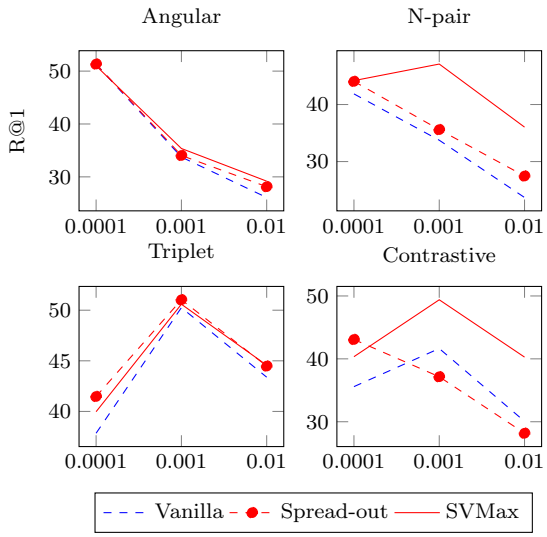


Figure 3.3: Quantitative evaluation on CUB-200-2011 using ResNet50. The X axis denotes the learning rate lr and the Y-axis denotes recall@1 performance.

other hyperparameters (b, d).

Results: Tables 3.1 and 3.2 present quantitative retrieval evaluation on CUB-200 and Stanford Online Products datasets – both using GoogLeNet. These tables provide in depth analysis and emphasize our improvement margins on a small and large dataset. Figure 3.2 provides quantitative evaluation on Stanford CARS196. Figures 3.3 and 3.4 present quantitative evaluation using ResNet50 on CUB-200-2011 and Stanford CARS196, respectively. Our training hyperparameters – learning rate lr and number of iterations K – do not favor a particular ranking loss in these experiments.

We evaluate SVMax on various learning rates. A large learning rate, *e.g.*, $lr = 0.01$, speeds up convergence, but increases the chance of model collapse. In contrast, a small rate, *e.g.*, $lr = 0.0001$, is likely to avoid model collapse but is slow to converge. This undesirable effect is tolerable for small datasets – where increasing

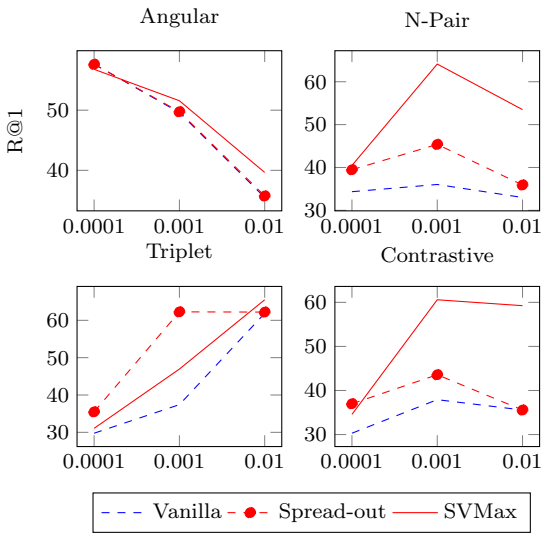


Figure 3.4: Quantitative evaluation on Stanford CARS196 using ResNet50.

the number of training iterations K does not drastically increase the overall training time – but it is infeasible for large datasets. For contrastive and N-pair losses, SVMax significantly outperforms both the vanilla and spread-out baselines with larger learning rates. A small lr slows convergence and all approaches are roughly equivalent. The spread-out regularizer [159] and its hyperparameters are tuned for triplet loss. Thus, for this particular ranking loss, the SVMax and spread-out regularizers are on par.

In our experiments, we employ a large learning rate because it is the simplest factor to induce model collapse. However, the learning rate is not the only factor. Another factor is the training dataset size and its intra-class variations. A small dataset with large intra-class variations increases the chances of a model collapse. For example, a pair of dissimilar birds from the same class justifies a model collapse when coupled with a large learning rate. The hard triplet loss experiments emphasize this point because every anchor is paired with the hardest positive and negative

samples. On small fine-grained datasets like CUB-200 or CARS196, the vanilla hard triplet loss suffers significantly. Yet, the same implementation is superior on a big dataset like Stanford Online Products. By carefully tuning the training hyperparameter on CUB-200, it is possible to avoid a degenerate solution. However, this tedious tuning process is unnecessary when using either the spread-out or the SVMax regularizer.

The vanilla N-pair loss underperforms because it does not support feature embedding on the unit circle. Both spread-out and SVMax mitigate this limitation. For angular loss, a bigger $\lambda = 2$ is employed to cope with the angular loss range. SVMax is a class oblivious regularizer. Thus, λ should be significant enough to contribute to the loss function without dominating the ranking loss.

Wu et al. [146] show that the distance between any anchor-negative pair, which is randomly sampled from an n-dimensional unit sphere, follows the normal distribution $N(\sqrt{2}, \frac{1}{2n})$. This mean distance $\sqrt{2}$ is large relative to the triplet loss margin $m = 0.2$, but comparable to the contrastive loss margin $m = 1$. Accordingly, triplet loss converges to zero after a few iterations, because most triplets satisfy the margin $m = 0.2$ constraint. When triplet loss equals zero, the SVMax regularizer with $\lambda = 1$ becomes the dominant term. However, SVMax should not dominate because it is oblivious to data annotations; it equally pushes anchor-positive and anchor-negative pairs apart. Reducing λ to 0.1 solves this problem.

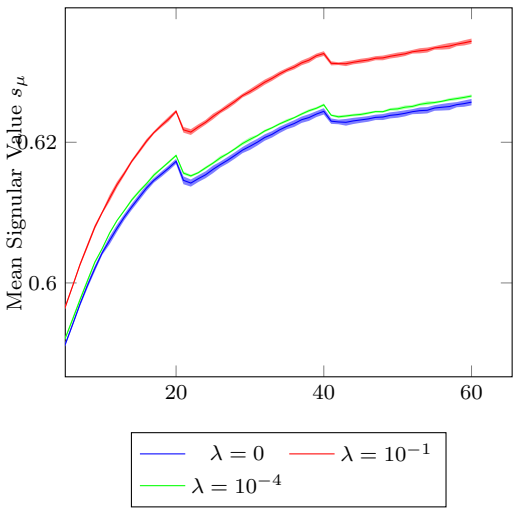
A less aggressive triplet loss [118, 150] is another way to avoid model collapse. For instance, Schroff et al. [118] have proposed a triplet loss variant that employs semi-hard negatives. The semi-hard triplet loss is more stable than the aggressive

Table 3.3: Quantitative retrieval evaluation using Proxy-Anchor loss on three datasets: CUB-200, Stanford Cars, and Stanford Online Products (SOP). The performance is reported on both Inception-BN and ResNet50 architectures using Recall@1.

	CUB	CARS	SOP
	Inception-BN		
Vanilla	66.92±0.002	84.67±0.002	79.02±0.001
SVMax $\lambda = 10^{-3}$ (Ours)	67.00±0.008	84.73±0.002	**79.04±0.000**
SVMax $\lambda = 10^{-4}$ (Ours)	**67.16±0.003**	**84.74±0.002**	**79.04±0.001**
	ResNet50		
Vanilla	68.37±0.003	87.30±0.001	**80.06±0.007**
SVMax $\lambda = 10^{-3}$ (Ours)	**68.67±0.004**	87.39±0.002	79.26±0.007
SVMax $\lambda = 10^{-4}$ (Ours)	68.45±0.005	**87.64±0.003**	79.52±0.004

hard triplet and lifted structured losses [95]. Unfortunately, the semi-hard triplet loss assumes a large mini-batch ($b = 1,800$ in Schroff et al. [118]), which is impractical. Furthermore, when model collapse is avoided, aggressive triplet loss variants achieve superior performance [48]. In contrast, SVMax only requires a larger mini-batch than the embedding dimension, *i.e.*, $b \geq d$, a natural constraint for retrieval networks which favor compact embedding dimensions. Additionally, SVMax makes no assumptions about the sampling procedure. Thus, unlike [113, 159], SVMax supports various supervised ranking losses.

SVMax and SOTA results: In the previous experiments (tables 3.1 and 3.2), we did not tune our hyperparameters to a particular ranking loss. The best hyperparameters for angular loss achieve inferior performance on contrastive loss and vice versa. Thus, we choose learning rates uniformly [0.01, 0.001, 0.0001]. In order to achieve SOTA results, it is important to tune hyperparameters (lr, # iterations), leverage a large embedding dimension ($d = 512$), and freeze certain layers (*e.g.*, batch norm). In the following experiment, we evaluate SVMax on the Proxy-Anchor



λ	Recall@1
0	79.02±0.001
10^{-1}	79.00±0.001
10^{-4}	**79.04±0.001**

Figure 3.5: Stanford Online Products experiment. Left: Quantitative evaluation of s_μ while training for 60 epochs. Right: Quantitative retrieval evaluation using recall@1 metric.

loss [65].

Table 3.3 presents a quantitative retrieval evaluation using Proxy-Anchor. For these experiments, we set the embedding dimension $d = 512$. We use a mini-batch $b = 512$ with Inception-BN [59] and $b = 256$ with ResNet50 [44]. To achieve SOTA results on different datasets and architectures, Proxy-Anchor tunes *five* hyperparameters[1]: number of warm-up epochs, learning rate, whether to freeze batch norm or not, learning rate decay step, and learning rate decay gamma. This intense hyperparameter tuning explains why SVMax has marginal effect on Proxy-Anchor. We recommend a single $\lambda = 10^{-4}$ across all architectures and datasets. We report performance using both $\lambda = \{10^{-3}, 10^{-4}\}$ to highlight SVMax's stability.

The bounds of the mean singular value s_μ enable us to evaluate the feature embedding quantitatively. In the Inception-BN experiment ($b = d = 512$), the upper and lower bounds of s_μ are $[L, U] = [0.044, 1]$. Figure 3.5 depicts s_μ during

[1]https://github.com/tjddus9597/Proxy-Anchor-CVPR2020

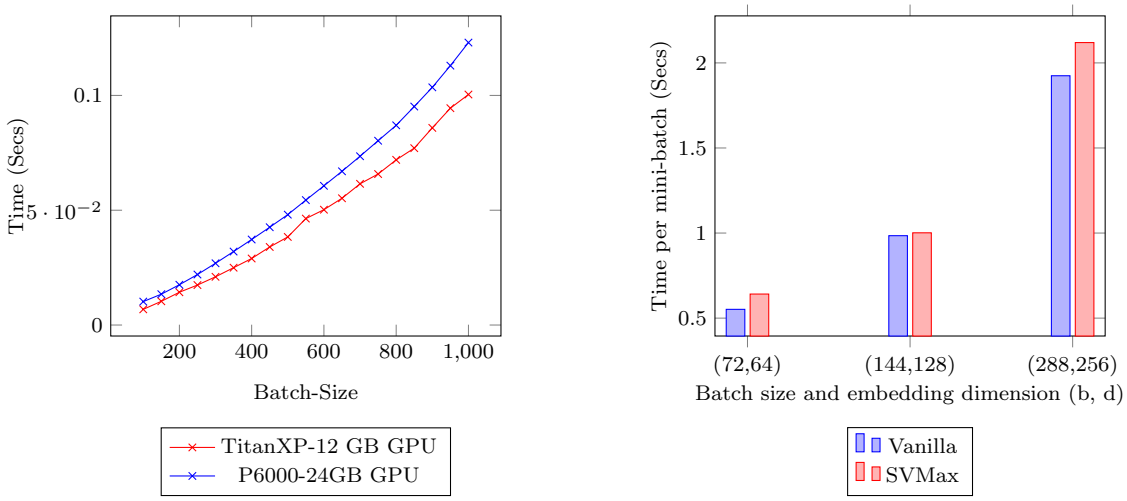


Figure 3.6: (Left) Timing analysis for the Tensorflow (TF) `tf.linalg.svd` function. The x-axis denotes the batch size b, and the y-axis denotes the running time in seconds. We time this TF function using two different GPUs. (Right) Timing analysis for a mini-batch training time using MobileNet. The x-axis denotes both the batch size b and the embedding dimension d. The y-axis denotes the batch training time in seconds.

training on Stanford Online Products for 60 epochs, and the learning rate decays every 20 epochs. The figure depicts s_μ using three different λ values. We repeat each experiment five times and report the mean s_μ (solid line) and standard deviation (filled-area). The vanilla Proxy-Anchor loss maximizes s_μ without SVMax ($\lambda = 0$). With $\lambda = 10^{-1}$, SVMax regularizes the feature embedding and maximizes s_μ, but recall@1 (R@1) decreases by 0.02%. SVMax with $\lambda = 10^{-4}$ maximizes s_μ while boosting R@1.

Weight decay imposes a prior on the network weights. Similarly, SVMax imposes a prior on the network activations. Both priors are important but they should never dominate the loss function. Thus, a large λ (*e.g.*, 1 or 0.1) is undesirable if the hyperparameters are already tuned to achieve state-of-the-art results.

SVMax's Computational Complexity: We compute the singular values $S =$

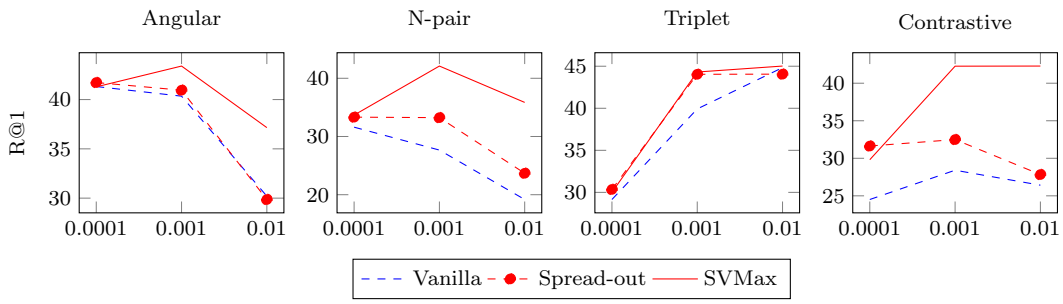


Figure 3.7: Quantitative evaluation on CUB-200-2011 using GoogLeNet with $b = 72$ and $d = 128$, *i.e.*, $b < d$.

$[s_1, ., s_i, ., s_d]$ using TensorFlow (TF) `tf.linalg.svd`. This function runs on the GPU. We did not notice any computational overhead or numerical instability during training. Figure 3.6 (Left) provides a timing analysis of the TF function using square matrices. For a typical mini-batch size ($b < 256$), the function takes around 0.01 seconds. This speed depends on the GPU specification and recent GPUs would perform faster. Figure 3.6 (Right) provides a timing analysis for the mini-batch training time using MobileNet. SVMax adds minimal overhead compared to the overhead of performing gradient descent on a deep network. We conclude that for a reasonable batch size b and embedding dimension d, SVMax adds minimal computational complexity to the training process.

SVMax with Small Batches: In the approach section, we assumed $b \geq d$ to deliver a rigorous mathematical foundation for SVMax. In this section, we present empirical evidence to support SVMax with small mini-batches. When $b < d$, there will be at most b singular values, instead of d. The lower and upper bounds of SVMax, per mini-batch, become $[L, U] = [\frac{\sqrt{b}}{b}, \sqrt{\frac{b \times d}{max(b,d)}} \frac{\sqrt{b}}{b}]$. It is possible that SVMax will utilize only b dimensions of the feature embedding space. We argue against this possibility using a toy example. Consider the following two mini-batches

$(m_1, m_2) \in R^{3\times d}$

$$m_1 = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & .. & 0 \\ 0 & 1 & 0 & 0 & .. & 0 \\ 0 & 0 & 1 & 0 & .. & 0 \end{bmatrix}, \tag{3.7}$$

$$m_2 = \begin{bmatrix} m_{21} \\ m_{22} \\ m_{23} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & .. & 0 \\ 0 & 1 & 0 & 0 & .. & 0 \\ 0 & 0 & 1 & 0 & .. & 0 \end{bmatrix}, \tag{3.8}$$

where the mini-batch size $b = 3$. Each individual mini-batch utilizes only the first three dimensions, *i.e.*, $rank(m_1) = rank(m_2) = 3$. While all other dimensions $[4, d]$ contain zeros, the maximum mean singular value is feasible with only the first three dimensions. However, due to the random sampling procedure, a future mini-batch m_3 will contain elements from both m_1 and m_2. For instance, $m_3 = [m_{11}\ m_{21}\ m_{22}]^T$ will have a $rank(m_3) = 2$. For the mini-batch m_3, the mean singular value is not maximum. To maximize s_μ, one feasible solution is to keep utilizing only the first three dimensions. However, this solution is like tossing a coin N times and expecting N heads. It is a feasible solution but unlikely.

Figure 3.7 presents a quantitative evaluation using CUB-200 on GoogLeNet with $b = 72$ and $d = 128$. Similarly, Figure 3.8 presents a quantitative evaluation using Stanford Online Products. SVMax consistently outperforms the vanilla and spread-out baselines even when $b < d$.

Finally, Figure 3.9 depicts the mean singular value on the test split of CUB-

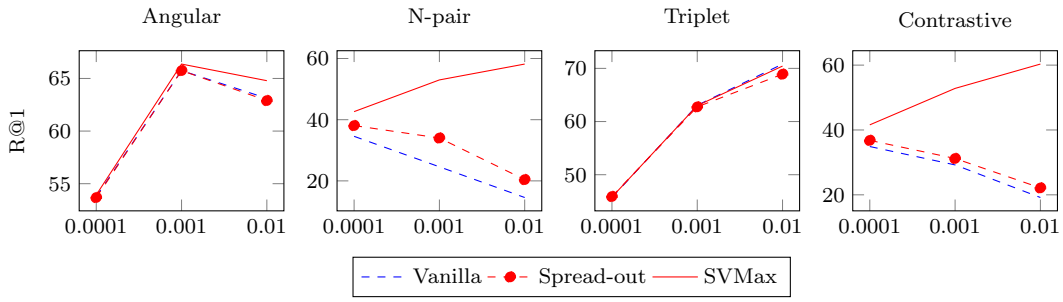


Figure 3.8: Quantitative evaluation on Stanford Online Products using GoogLeNet with $b = 72$ and $d = 128$, *i.e.*, $b < d$.

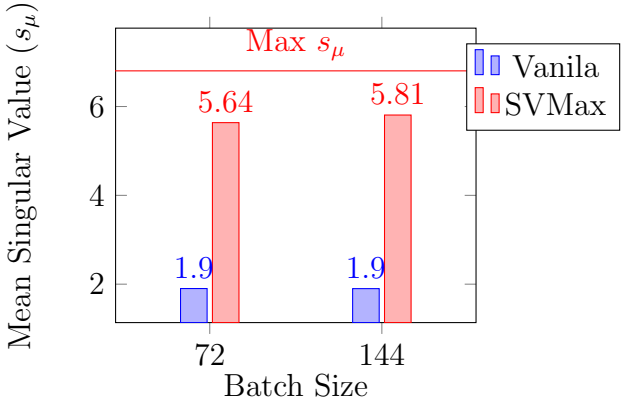


Figure 3.9: The mean singular values s_μ for networks trained with an embedding dimension $d = 128$. The X and Y-axes denote the mini-batch size b and the s_μ of the feature embedding of CUB-200's test split. The feature embedding is learned using a contrastive loss with and without SVMax. The horizontal red line denotes the upper bound on s_μ. With SVMax, s_μ decreases marginally with $b = 72$ compared to $b = 144$. Thus, the SVMax still promotes a uniform feature embedding even when $b < d$.

200. We train our network using (1) contrastive loss with and without SVMax, and (2) different mini-batch sizes $b = \{72, 144\}$. We fix the embedding dimension $d = 128$ to study the batch size's impact, *i.e.*, $b < d$ versus $b \geq d$. The test split of CUB-200 has 5924 test images. Thus, the upper bound of the mean singular value $U = \sqrt{\frac{b \times d}{max(b,d)}} \frac{\sqrt{b}}{d} = 6.80$, where $b = 5924$ and $d = 128$ for the *whole* test split. After training our network, the actual mean singular value $s_\mu = 5.64$ with batch size $b = 72$, and $s_\mu = 5.81$ with $b = 144$. These mean singular values significantly

outperform their vanilla contrastive loss counterparts ($s_\mu = 1.9$). Compared to $b = 144$, s_μ is smaller when using the mini-batch size $b = 72$. At a mini-batch level, SVMax spreads the feature embedding across $d = 128$ dimensions when $b = 144$, while SVMax spreads the feature embedding across $d = 72$ dimensions when $b = 72$. Yet, the comparable s_μ (5.64 versus 5.81) indicates that SVMax supports $b < d$.

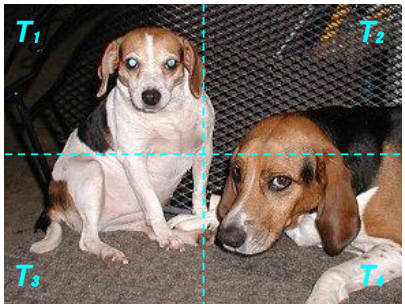
3.4.2 Self-Supervised Learning

Another form of model collapse is a loss function with a trivial solution. This form manifests in the two terms of contrastive loss Eq. 3.9 as follows

$$\mathrm{CL}_{(x,y)\in P} = \delta_{x,y} D_{x,y} + (1 - \delta_{x,y}) \left[m - D_{x,y}\right]_+, \tag{3.9}$$

where $D_{x_1,x_2} = D(N(x_1), N(x_2))$; $N(\bullet)$ and $D(\bullet, \bullet)$ are the network's output-embedding and Euclidean distance, respectively. While the first term pulls similar points together, the second term pushes different points apart. Without the second term, contrastive loss suffers a model collapse, *i.e.*, the trivial solution $N(x) = 0,\ \forall x$. In supervised metric learning, similar and different classes are labeled. Thus, it is trivial to leverage contrastive loss with both terms. However, it is non-trivial to leverage contrastive loss in un/self-supervised learning.

To avoid this trivial solution, different methods have been proposed for un/self-supervised learning. For example, SimCLR [15] repels random images – assuming they belong to different classes. Other approaches like SwAV [11] leverages online



Tile	Nose	Eye	Ear	Paws
T_1	1	2	2	0
T_2	0	0	0	0
T_3	0	0	0	3
T_4	1	2	2	1
I (Total)	2	4	4	4

Figure 3.10: An image I is split into four tiles T_i, where $i \in \{1, 2, 3, 4\}$. To learn an image representation, Rep-Cnt trains a network N such that counts of visual primitives in I equals the total visual primitives in T_i as shown in the table – $N(I) = \sum_{i=1}^{4} N(T_i)$.

clustering, while BYOL [38] leverages a momentum encoder. Recently, SimSiam [17] utilize a stop-gradient operation to avoid model collapse. All these methods deliver SOTA results, but they make assumptions about the problem formulation. For example, both SimCLR and SwAV require a large batch (e.g., 4096) to work well. In the following experiment, we show that SVMax avoids model collapse without making assumptions about the problem formulation.

To evaluate SVMax in self-supervised learning, we first introduce representation counting (Rep-Cnt) [94]. Rep-Cnt is a simple self-supervised approach that counts visual primitives. Given an input image I, Rep-Cnt splits I into four tiles T_i, where $i \in \{1, 2, 3, 4\}$ as shown in Fig. 3.10. Rep-Cnt trains a neural network N to count visual primitives in I and T_i. The network is trained to maintain the equivariance relation. The equivariance relation requires that the count of visual primitives in I equals the total visual primitives from the four tiles, *i.e.*,

Table 3.4: Quantitative SVMax evaluation using self-supervised learning. We evaluate the pretrained network N through ImageNet classification with a linear classifier on top of frozen convolutional layers. For every layer, the convolutional features are spatially resized until there are fewer than 10K dimensions left. A fully connected layer followed by softmax is trained on a 1000-way object classification task. * denotes our implementation of the baseline.

Method	conv1	conv2	conv3	conv4	conv5
Supervised	19.3	36.3	44.2	48.3	50.5
Random	11.6	17.1	16.9	16.3	14.1
Context [23]	16.2	23.3	30.2	31.7	29.6
Jigsaw [93]	18.2	28.8	34.0	33.9	27.1
ContextEncoder [98]	14.1	20.7	21.0	19.8	15.5
Adversarial [24]	17.7	24.5	31.0	29.9	28.0
Colorization [157]	12.5	24.5	30.4	31.5	30.3
Split-Brain [158]	17.7	29.3	**35.4**	**35.2**	**32.8**
Rep-Cnt* [94]	18.9	**30.7**	33.9	30.6	26.0
Rep-Cnt+SVMax″ (Eq. 3.11 $\lambda = 10$)	**19.4**	29.3	31.7	28.9	24.5
Rep-Cnt+SVMax″ (Eq. 3.11 $\lambda = 100$)	19.2	29.4	31.8	29.3	25.3

$N(I) = \sum_{i=1}^{4} N(T_i)$. Unfortunately, this simple self-supervised signal has a trivial solution, *i.e.*, $N(x) = 0,\ \forall x$. To avoid the trivial solution, Rep-Cnt is formulated as follows

$$L = \mathbb{D}_{I,T_i} + \underbrace{\left[m - \mathbb{D}_{\hat{I},T_i}\right]_+}_{Problem-specific} \tag{3.10}$$

where $\mathbb{D}_{I,T_i} = D\left(N(I), \sum_{i=1}^{4} N(T_i)\right)$. The second term pushes the representation of a random image $\hat{I}$ from $\left(\sum_{i=1}^{4} N(T_i)\right)$, *i.e.*, the representation of I. This loss formulation explains how a problem-specific loss term is always required to avoid model collapse. Similarly, a different self-supervised pretext requires a different problem-specific formulation. Instead, we propose SVMax, a generic prior, to promote a uniform feature embedding.

To integrate SVMax in Rep-Cnt, we replace the problem-specific term with

our generic prior as follows

$$L = \mathbb{D}_{I,T_i} - \lambda \underbrace{s_\mu}_{Generic-prior} \tag{3.11}$$

where s_μ is the mean singular value of the mini-batch embeddings $N(I)$. We leverage the unbounded SVMax formulation ($-\lambda s_\mu$) instead of the bounded SVMax $\left(\lambda \exp\left(\frac{U-s_\mu}{U-L}\right)\right)$ because vector-norms satisfy the triangle inequality property, *i.e.*, $||x+y||^2 \leq ||x||^2 + ||y||^2$. If we normalize the output embedding, the $N(I) = \sum_{i=1}^{4} N(T_i)$ objective becomes infeasible.

To evaluate SVMax quantitatively, we follow Rep-Cnt technical details. We use AlexNet with three fully connected layers. The last fully connected layer provides a feature embedding. Thus, we reduce the layer's dimension from 1000 to 128. This reduces the computational cost of SVMax. We set $m = 10$ in Eq. 3.10 as in [94]. We pretrain the AlexNet network using both Eq. 3.10 and Eq. 3.11 as a self-supervision signal. For each pretrained network, we train a linear classifier on top of the frozen CNN layers using ImageNet [22]. This evaluation configuration is proposed by [157].

Table 3.4 presents a quantitative SVMax evaluation in self-supervised learning. We applied SVMax on Rep-Cnt because it is a simple baseline. Rep-Cnt + SVMax does not achieve state-of-the-art results. However, SVMax can be applied on top of various self-supervised pretexts. Through this experiment, we demonstrate how SVMax avoids a model collapse in self-supervised learning. SVMax avoids the trivial solution without the need for a problem-specific repulsion term, an input-

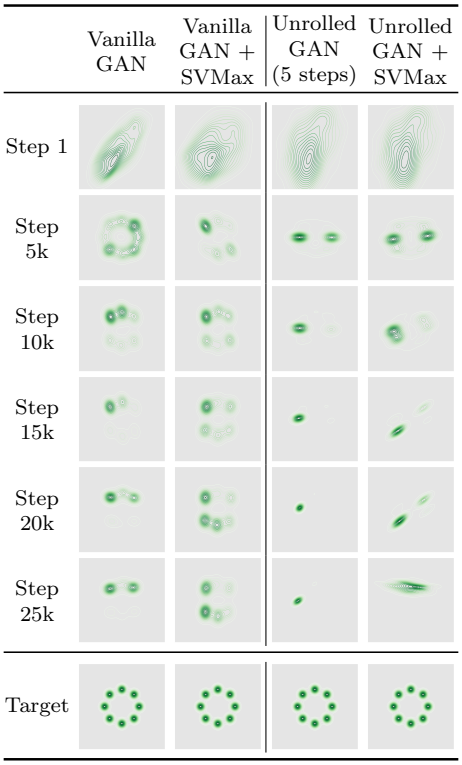


Figure 3.11: SVMax mitigates model collapse in a GAN trained on a toy 2D mixture of Gaussians dataset. Rows show heatmaps of the generator distributions at different training steps. The final row shows the groundtruth distribution. The first column shows the distributions generated by training a vanilla GAN suffering a model collapse. The second column shows the generated distribution when penalizing the generator's fake embedding with SVMax. The third and fourth columns show two distributions generated using an unrolled-GAN without and with SVMax, respectively. This high resolution figure is best viewed on a screen with zoom capabilities.

reconstruction term, or an adversarial loss term.

3.4.3 Generative Adversarial Networks

Model collapse is one of the main challenges of training generative adversarial networks (GANs) [88, 125, 87, 114]. To tackle this challenge, Metz et al. [88] propose an unrolled-GAN to prevent the generator from overfitting to the discriminator. In an unrolled-GAN, the generator observes the discriminator for l steps before updat-

ing the generator's parameters using the gradient from the final step. Alternatively, we leverage the simpler SVMax regularizer to avoid model collapse. We evaluate our regularizer using a simple GAN on a 2D mixture of 8 Gaussians arranged in a circle. This 2D baseline [88, 125, 4] provides a simple qualitative evaluation and demonstrates SVMax's potential in unsupervised learning. We leverage this simple baseline because we assume $b \geq d$, which does not hold for images.

Figure 3.11 shows the dynamics of the GAN generator through time. We use a public PyTorch implementation[2] of [88]. We made a single modification to the code to use a relatively large learning rate, *i.e.*, $lr = 0.025$ for both the generator and discriminator. This single modification is a simple and fast way to induce model collapse. The mixture of Gaussians circle has a radius $r = 2$, *i.e.*, the generated fake embedding is neither L2-normalized nor strictly bounded by a network layer. We kept the radius parameter unchanged to emphasize that neither L2-normalization nor strict-bounds are required. To mitigate the impact of lurking variables (*e.g.*, random network initialization and mini-batch sampling), we fix the random generator's seed for all experiments. We apply SVMax to a vanilla and an unrolled GAN for five steps. We apply the unbounded SVMax regularizer (Eq. 3.1), *i.e.*, $L_{\text{NN}} = L_{\text{GAN}} - \lambda s_{\mu}$, where $\lambda = 0.01$ and s_{μ} is mean singular value of the generator fake embedding.

GANs are typically used to generate high resolution images. This high-resolution output is the main limitation of the SVMax regularizer. The *current* formulation assumes the batch size is bigger than the embedding dimension, *i.e.*, $b \geq d$. This con-

[2]https://github.com/andrewliao11/unrolled-gans

straint is trivial for the Gaussians mixture 2D dataset and retrieval networks with a compact embedding dimensionality (*e.g.*, $d = \{128, 256\}$). However, this constraint hinders high resolution image generators because the mini-batch size constraint becomes $b \geq W \times H \times C$, where W, H, and C are the generated image's width, height, and number of channels, respectively. Nevertheless, this GAN experiment emphasizes the potential of the SVMax regularizer in unsupervised learning.

If the batch-size limitation is set aside, the following points are worth noting: (I) Image-synthesis GANs have bounded outputs [0, 255]; White images will not fool the discriminator. Thus, s_μ remains bounded but with different bounds from those presented in the approach section. (II) Alain and Bengio [3] (§3.4) address practical concerns when working with high dimensional features. (III) GANs have synthesized not only high quality images, but also feature embeddings [166].

3.4.4 Ablation Study

In this section, we evaluate two hypotheses: (1) the same SVMax hyperparameter λ supports different embedding dimensions and batch sizes – the main objective of the mean singular value's bounds analysis, (2) the SVMax regularizer boosts retrieval performance because it learns a uniform feature embedding.

The mean singular value bound analysis makes tuning the hyperparameter λ easier. This hyperparameter becomes only dependent on the ranking loss's range and independent of both the batch size and the embedding dimension. Figure 3.12 presents a quantitative evaluation using the CUB-200 dataset. We explore various

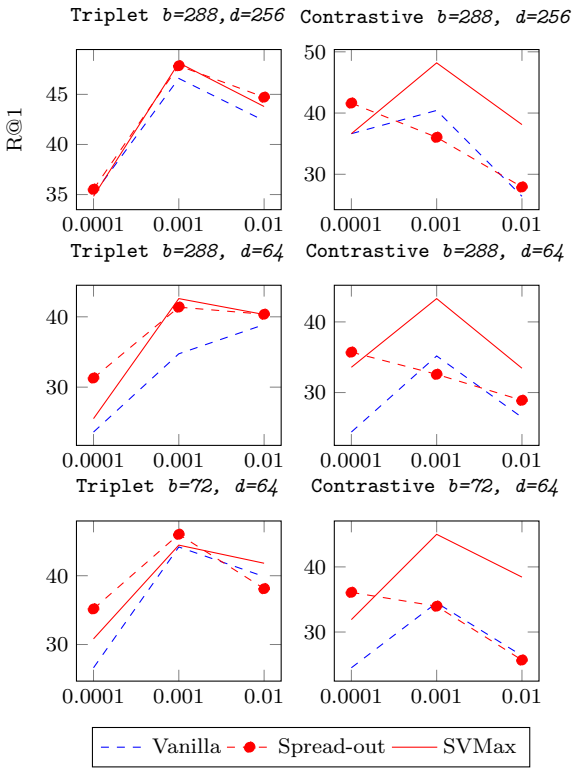


Figure 3.12: Quantitative evaluation on CUB-200-2011 with various batch sizes $b = \{288, 72\}$ and embedding dimensions $d = \{256, 64\}$ to demonstrate the stability of our hyperparameter. $\lambda = 1$ for contrastive loss and $\lambda = 0.1$ for triplet loss.

batch sizes $b = \{288, 72\}$ and embedding dimensions $d = \{256, 64\}$. We employ a MobileNetV2 [115] to fit the big batch $b = 288$ on a 24GB GPU.

To evaluate SVMax's impact on feature embeddings, we embed the MNIST dataset onto the 2D unit circle. In this experiment, we use a tiny CNN (one convolutional layer and one hidden layer). Figure 3.13 shows the feature embedding after training for t epochs. With SVMax, the feature embeddings spread out more uniformly and rapidly than the vanilla contrastive loss.

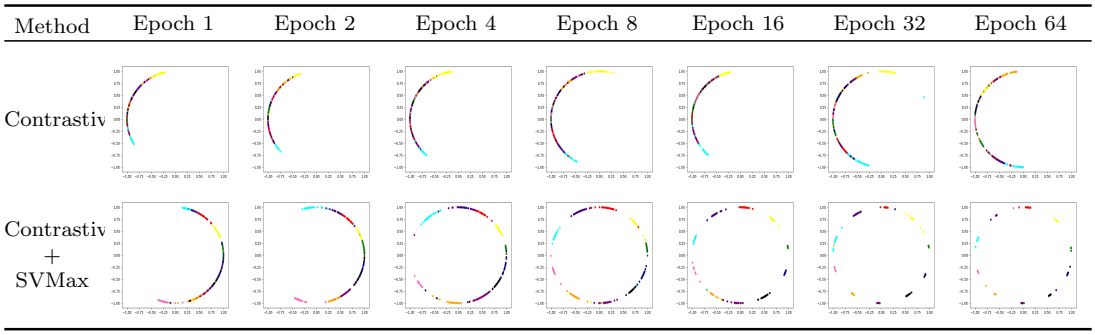


Figure 3.13: Qualitative feature embedding evaluation using the MNIST dataset projected onto the 2D unit circle. The first row shows the feature embedding learned using a vanilla contrastive loss and the second row applies the SVMax regularizer. A random subset of the test split is projected for visualization purpose. Different colors denote different classes. The regularized feature embedding spreads out uniformly and rapidly.

3.5 Conclusion

We have proposed singular value maximization (SVMax) as a feature embedding regularizer. SVMax promotes a uniform embedding, mitigates model collapse, and enables large learning rates. Unlike other embedding regularizers, SVMax supports a large spectrum of ranking losses. Moreover, it is oblivious to data annotation and, as such, supports both supervised and unsupervised learning. Qualitative evaluation using a generative adversarial network demonstrates SVMax's potential in unsupervised learning. Quantitative retrieval evaluation highlight significant performance improvements due to the SVMax regularizer.

Chapter 4: A Generic Visualization Approach for Convolutional Neural Networks

Retrieval networks are essential for searching and indexing. Compared to classification networks, attention visualization for retrieval networks is hardly studied. We formulate attention visualization as a constrained optimization problem. We leverage the unit L2-Norm constraint as an attention filter (L2-CAF) to localize attention in both classification and retrieval networks. Unlike recent literature, our approach requires neither architectural changes nor fine-tuning. Thus, a pre-trained network's performance is never undermined

L2-CAF is quantitatively evaluated using weakly supervised object localization. State-of-the-art results are achieved on classification networks. For retrieval networks, significant improvement margins are achieved over a Grad-CAM baseline. Qualitative evaluation demonstrates how the L2-CAF visualizes attention per frame for a recurrent retrieval network. Further ablation studies highlight the computational cost of our approach and compare L2-CAF with other feasible alternatives. Code available at *https://bit.ly/3iDBLFv*

Figure 4.1: L2-CAF enables both class-oblivious and class-specific visualizations. This separates our work from dominant literature that targets classification networks only.

4.1 Introduction

Both classification and retrieval neural networks need attention visualization tools. These tools are important in medical and autonomous navigation to understand and interpret networks' decisions. Moreover, attention visualization enables weakly supervised object localization (WSOL) which reduces the cost of data annotation. WSOL avoids bounding-box labeling required by fully supervised approaches. Attention visualization and WSOL have been intensively studied for classification architectures [154, 164, 122, 120, 160, 19]. However, these approaches do not address retrieval networks. In this paper, we leverage the unit L2-Norm constraint as an attention filter (L2-CAF) that works for both classification and retrieval neural networks, as shown in Figure 4.1.

For classification networks, Zhou *et al.* [164] propose class activation maps (CAM) for attention visualization and WSOL. Further research [122, 160, 161, 19] improved WSOL by augmenting the most discriminative region with other less discriminative parts, *e.g.*, augment a cat's head with its legs. This improvement comes at the cost of few drawbacks: (1) They impose architectural constraints, *e.g.*, global

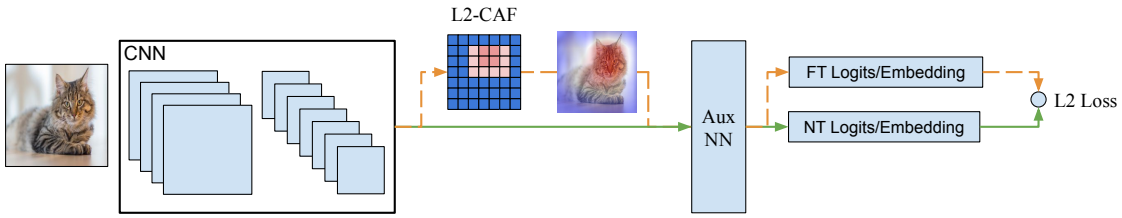


Figure 4.2: An overview of the proposed unit L2-Norm constrained attention filter (L2-CAF). Given a pre-trained CNN with an auxiliary head (Aux NN), feed an input frame through a normal feed-forward pass (green solid path) to generate the network output logits/embedding $NT(x)$. Then, feed the same input again but multiply the last convolutional feature map by a constrained attention filter f (orange dashed path) to generate a new filtered output $FT(x, f)$. Optimize the filter's weights through gradient descent to minimize the difference between $NT(x)$ and $FT(x, f)$. In standard CNN architectures, the L2-CAF is typically 7×7, *i.e.*, a cheap optimization problem $\in R^{49}$.

average pooling (GAP) layer; (2) While fine-tuning boosts localization efficiency, it degrades classification accuracy. Grad-CAM [120] avoids these limitations, but it is originally formulated for classification networks.

Retrieval networks are essential for visual search [96, 64], zero-shot learning [8, 152, 163], and fine-grained retrieval [124, 95]. The large metric learning [95, 143, 16] and product quantization [9, 75, 27] literature reflect their importance. Despite that, attention visualization for retrieval networks has not been evaluated quantitatively. It is more challenging compared to classification due to the network's output – a class-oblivious embedding.

The main contribution of this paper is to leverage the L2-CAF as a visualization filter to identify key features of both classification and retrieval networks' output. Figure 4.2 illustrates the approach. Given a pre-trained CNN, feeding the same input x through the network (green solid path) will always generate the same output $NT(x)$. If the final convolutional feature map is multiplied by a constrained

attention filter f in an element-wise manner (orange dashed path), the network generates a filtered output $FT(x, f)$. Through gradient descent, we optimize f to minimize the L2 loss $L = ||NT(x) - FT(x, f)||^2$. The optimized filter f reveals key spatial regions, *e.g.*, the cat's head. The filter size (f_w, f_h) depends on the convolution layer size, *e.g.*, the last convolution layer in standard CNNs $\in R^{7\times7}$.

This approach imposes no constraints on the network architecture besides having a convolution layer. The input can be a regular image or a pre-extracted convolutional feature. The network output can be logits trained with softmax or a feature embedding trained with a ranking loss. Furthermore, this approach neither changes the original network weights nor requires fine-tuning. Thus, network performance remains intact. The visualization filter is applied only when an attention map is required. Thus, it poses no computational overhead during inference. L2-CAF visualizes the attention of the last convolutional layer of GoogLeNet within 0.3 seconds.

Section 4.3 describes two variants of the L2-CAF and their mathematical optimization details. The first is the class-oblivious variant illustrated in Figure 4.2. The second is the class-specific variant for classification networks to localize objects of a specific class. We also present a technique to reduce the computational cost of the L2-CAF's optimization formulation. We benchmark our approach quantitatively using WSOL for both classification and retrieval architectures.

In summary, the key contributions of this paper are:

1. A novel attention visualization approach for *both* classification and retrieval

networks (Sec. 4.3). This approach achieves state-of-the-art WSOL results using classification architectures (Sec. 4.4.1).

2. A modified Grad-CAM to better support WSOL on retrieval networks (Sec. 4.4.2); L2-CAF achieves significant localization improvement margins, up to an absolute 36%, compared to the vanilla Grad-CAM.
3. A method to visualize attention for video frames that are temporally fused using a recurrent network (Sec. 4.4.3).

4.2 Related Work

This section briefly reviews weakly supervised object localization (WSOL) for classification networks. Grad-CAM is reviewed in the WSOL retrieval evaluation Section 4.4.2. Figure 4.3 presents a high-level categorization of WSOL approaches in terms of (1) supported architectures; (2) whether fine-tuning is required or not? The experiment section provides further one-to-one comparisons.

Classification networks' attention visualization increases interpretability and enables WSOL. CAM [164] and Grad-CAM [120] identify the most discriminative spatial region. To boost WSOL performance, [122, 160, 161, 19] propose architectural modifications to augment the most discriminative region with less-discriminative object regions. This is achieved by fine-tuning a pre-trained network while hiding the most discriminative region stochastically. This forces the network to recognize other informative regions and thus improve WSOL. To detect and hide the most discriminative region while fine-tuning, a network is assumed to use

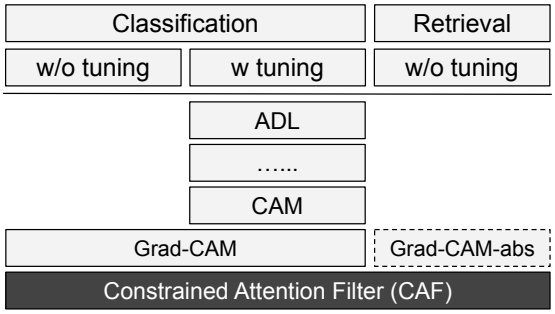


Figure 4.3: An overview of weakly supervised object localization (WSOL) approaches for classification and retrieval networks. Some approaches impose architectural constraints and require fine-tuning, *e.g.*, CAM and ADL.

global average pooling (GAP) [164, 122, 19] or an equivalent 1×1 feature reduction convolution layer [160]. This fine-tuning paradigm tends to degrade classification performance.

4.3 Constrained Attention Filter (CAF)

This section presents two variants for optimizing the proposed L2-CAF. The first variant, *class-oblivious*, works for both classification and retrieval CNNs. It generates a single heatmap per frame. The second variant, *class-specific*, works for classification CNNs and generates class-specific heatmaps per frame. Both variants impose no architectural constraints in terms of spatial pooling (GAP, FCN) or temporal fusing components (RNN, LSTM).

4.3.1 Class-Oblivious Variant

Given a pre-trained network and an input $x \in R^{W \times H \times 3}$, the last convolution layer provides a feature map $A \in R^{w \times h \times k}$, with size $w \times h$ and k channels. The network's output $NT(x)$, logits or embedding, depends on discriminative

features in A. We optimize an L2 normalized filter f to identify the discriminative features to the network's output $NT(x)$. After multiplying A by the filter f $\left(A \bigodot f = A' \in R^{w \times h \times k}\right)$, the network generates a filtered output $FT(x, f)$. While fixing the network's weights and input, we optimize f to minimize

$$L = ||NT(x) - FT(x, f)||^2, \quad \text{subject to} \quad ||f||_2 = 1, \tag{4.1}$$

$FT(x, f)$ equals $NT(x)$ if and only if $f = f_I = \{1\}^{w \times h}$ which is infeasible due to the unit L2-Norm constraint.

Intuition: An ideal heatmap can be regarded as a filter that approximates $NT(x)$ by blocking irrelevant features in A. Accordingly, we seek a filter f that *spatially* prioritizes convolutional features and *flexibly* captures irregular (*e.g.*, discontinuous) shapes or multiple different agents in a frame. The L2-Norm, a simple *multi-mode* differentiable filter, satisfies these requirements. On account of irrelevant features, the $||f||_2 = 1$ constraint assigns higher weights to relevant features. Figure 4.11 qualitatively emphasizes the intuition behind the L2-Norm constraint.

This formulation (Eq. 4.1) is oblivious to the nature of the network's output (logits or embedding), architecture, and input format (RGB image or pre-extracted features). For a given input x, the class-oblivious formulation generates a single heatmap. This can be a limitation if the input x contains objects from different classes. The next subsection tackles this limitation by offering an alternative class-specific optimization formulation.

4.3.2 Class-Specific Variant

To support class-specific heatmaps per input, we first assume a classification CNN architecture with class-specific logits. We learn the attention for class c by optimizing the L2-CAF f using the following loss function

$$L_c = -FT_c(x, f) + \sum_{i=0,\ i\neq c}^{N} FT_i(x, f), \quad \text{subject to} \quad ||f||_2 = 1, \tag{4.2}$$

where $FT_c(x, f)$ is the filtered output's logit for class c and N is the total number of classes. This loss maximizes the output logit for the intended class c while minimizing the output logits for all other classes.

Figure 4.1 presents a qualitative comparison between the class-oblivious and class-specific variants. For example, the first example shows an image of a butterfly standing on a mastiff's nose. The first image shows the resulting heatmap from optimizing Eq. 4.1. The following two images show the result heatmaps from optimizing Eq. 4.2 for the mastiff and butterfly classes, respectively. In these examples, the L2-CAF is applied to the last convolutional layer.

Technical Details: To compute the class-oblivious heatmap for an input x, we utilize gradient descent for l iterations. At iteration i, L^i is computed using the filter $\frac{f^i}{||f^i||_2}$. The filter f is initialized randomly, *i.e.*, $f^1 \in [0, 1]^{w\times h}$. Gradient descent iteratively updates f to minimize L. We terminate when L converges and remains approximately the same for d iterations. Concretely, we terminate the gradient descent at $i = l$ when $\left|L^l - L^{l-d}\right| < \epsilon$ where $\epsilon = 10^{-5}$ and $d = 50$. This constrained

minimization formulation is non-convex, so we also impose a maximum number of iterations L_{max} to avoid oscillating between local minima. After termination, the heatmap is generated by resizing $\frac{|f|^l}{||f^l||_2}$ to the input's size. The same procedure is used for class-specific heatmaps with L_c (Eq. 4.2). For more details, please refer to our released code.

Timing: To optimize the small (*e.g.*, 7×7) L2-CAF using gradient descent, the vanilla L2-CAF requires multiple feed-forward and backpropagation passes through the network. This is affordable for lightweight networks like MobileNet [54] and GoogLeNet (InceptionV1) [131] but computationally expensive for bulky networks like VGG [121] and DenseNet [58]. We propose a technique to reduce this cost through (1) making a single feed-forward pass through the whole network to compute the network's output at every layer, (2) optimizing the L2-CAF f using a small subset of layers.

Figure 4.4 illustrates this technique. Instead of optimizing the filter f through the network's endpoints (x, $NT(x)$), it is equivalent to use the outputs of the direct pre and post layers (V, V') to the attention filter. For a given input x, these layers' outputs ($V(x)$, $V'(x)$) require a single feedforward pass through the whole network. Once computed, the loss function from Eq. 4.1 becomes

$$L = ||V'(x) - FT(V(x), f)||^2, \quad \text{subject to} \quad ||f||_2 = 1. \tag{4.3}$$

To generate class-specific heatmaps for a classification network, $V'(x)$ must be the network's logits $NT(x)$. Since it is typical to visualize the attention of the

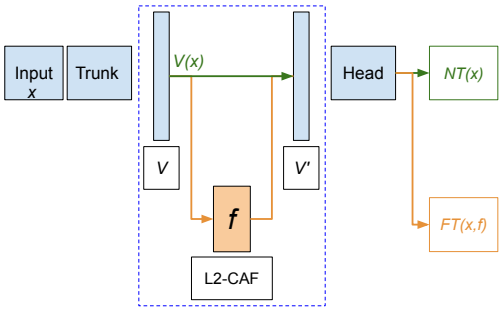


Figure 4.4: Reduce the computational optimization cost of the L2-CAF f by solving an equivalent sub-problem (blue-dashed). Instead of using the network's endpoints $(x,\ NT(x))$, use $(V(x),\ V'(x))$ to optimize f.

last convolution layer, this formulation skips the overhead of a network's trunk and significantly reduces the computational cost. The speed-up of this technique is quantified through an ablation study.

The fast L2-CAF approach is a computationally cheaper alternative to sampling [100, 123] and masking [30, 29] approaches. In addition, the L2-CAF has a smaller set of hyper-parameters. For instance, while both L2-CAF and masking [30, 29] approaches require a stopping criterion for an optimization problem, Fong *et al.* [29] evaluate multiple mask-sizes per image. Furthermore, the fast L2-CAF works on a small subset of network layers, *i.e.*, independent of the network backbone. Thus, it compares favorably for video processing. For $3D$ volumes (*e.g.*, medical images), our optimization problem remains independent of the network size, *i.e.*, $\in R^{7\times7\times7}$.

4.4 Experiments

The next two subsections present L2-CAF's quantitative evaluation using classification and retrieval networks, respectively. Then, a recurrent retrieval net-

work qualitatively illustrates L2-CAF's potential for video applications. Finally, we present our ablation studies.

4.4.1 WSOL Using Classification Networks

The L2-CAF is quantitatively evaluated using WSOL on both *standard* and *fine-tuned* classification architectures. We leverage the ImageNet validation set [22] for evaluation on standard architectures. For fined-tuned architectures, we follow ADL [19] evaluation procedure and utilize both ImageNet [22] and CUB-200-2011 [142] datasets. In all experiments, we use the fast L2-CAF technique. The loss in Eq. 4.2 is minimized using the last convolution layer and the network's logits (before softmax) as endpoints.

Evaluation using standard architectures is performed using both the top-1 and top-5 predictions. Similar to [120], we obtain the top predictions for every image, then, optimize our filter f to learn the corresponding heatmap for every prediction. Following Zhou *et al.* [164], we segment the heatmap using a simple thresholding technique. This generates connected segments of pixels; we draw a bounding box around the largest segment. Localization is correct if the predicted class is correct and the intersection over union (IoU) between the ground truth and estimated bounding boxes is $\geq$50%. Table 4.1 compares L2-CAF and Grad-CAM using three architectures. Both approaches are applied to the last 7×7 convolution layer. We fix the architecture and evaluate different localization approaches – same classification but different localization performance.

Table 4.1: Classification and localization accuracies on the ImageNet (ILSVRC) validation set using standard architectures – no fine-tuning required.

		Classification		Localization	
Method	Backbone	Top 1↑	Top 5↑	Top 1↑	Top 5↑
Grad-CAM	GoogLeNet [131]	71.17	86.39	44.43	57.50
L2-CAF (ours)	GoogLeNet [131]	71.17	86.39	**45.48**	**59.32**
Grad-CAM	ResNetV2-50 [45]	71.51	86.56	46.57	59.96
L2-CAF (ours)	ResNetV2-50 [45]	71.51	86.56	**48.18**	**62.38**
Grad-CAM	DenseNet-161 [58]	78.20	91.39	49.28	**66.57**
L2-CAF (ours)	DenseNet-161 [58]	78.20	91.39	**49.68**	65.28

Table 4.2: Classification and localization accuracies on the CUB-200-2011 test and ImageNet validation split using fine-tuned architectures. The accuracy with an asterisk* indicates that the score is from the original paper.

			CUB-200-2011		ImageNet	
Method	Backbone	Tuning	CLS ↑	LOC ↑	CLS ↑	LOC ↑
CAM	VGG-GAP	GAP	68.53	45.66	69.96	43.46
L2-CAF (ours)	VGG-GAP	GAP	68.53	46.01	69.96	44.09
Fuse 2 CAMs	VGG-GAP	ACoL [160]	71.90	45.90*	67.50	**45.83***
CAM	VGG-GAP	ADL	64.16	48.27	69.58	42.93
L2-CAF (ours)	VGG-GAP	ADL	64.16	**48.55**	69.58	43.27
CAM	ResNet50-SE	ADL	78.94	**61.71**	76.218	49.90
L2-CAF (ours)	ResNet50-SE	ADL	78.94	61.16	76.218	**50.49**

L2-CAF versus Grad-CAM: Grad-CAM is 5 times faster than L2-CAF on DenseNet-161 (7 times on GoogLeNet). Both approaches support a large variety of architectures. In terms of localization accuracy, L2-CAF compares favorably to Grad-CAM. Fong and Vedaldi [30] explain why gradient-based approaches like Grad-CAM are not optimal for visualization. They show that neural networks' gradients $\frac{\partial y}{\partial x}$ are independent of the input image x for linear classifiers ($y = wx + b; \frac{\partial y}{\partial x} = w$). For non-linear architectures, this problem is reduced but not eliminated. They also show qualitatively that gradient saliency maps contain strong responses in irrelevant image regions. We hypothesize that DenseNet-161's better classification accuracy and, accordingly, better gradient closes the localization performance gap.

Evaluation using fine-tuned architectures is performed using the top-1

accuracy on fine-tuned architectures (*e.g.*, VGG-GAP [164]); this follows the evaluation procedure in attention-based dropout layer (ADL) [19]. ADL is the current state-of-the-art method for WSOL. During fine-tuning, ADL applies dropout at multiple network stages. It is not straightforward to determine where to plug these extra dropout layers – it is network dependent. Therefore, we leverage their publicly released VGG-GAP and ResNet50-SE implementations to evaluate our approach. The ACoL performance is reported from the original paper.

Table 4.2 presents a quantitative evaluation using CUB-200-2011 and ImageNet datasets. The first column denotes the object localization approach, *e.g.*, CAM versus L2-CAF. Grad-CAM is dropped because it is equivalent to CAM when a GAP layer is utilized [120]. In the second column (backbone), all the architectures utilize a global average pooling or an equivalent surrogate [160]. The third column denotes the fine-tuning approaches considered: GAP [164], ACol [160], ADL [19]. We fine-tune the VGG-GAP architecture with both GAP and ADL. L2-CAF consistently outperforms CAM's localization on the ImageNet validation set.

Relation with WSOL approaches (*e.g.*, ADL): To generate class activation maps (CAMs), WSOL approaches employ a global average pooling layer (GAP) [164, 122, 161, 19], or equivalent [160]. L2-CAF relaxes this architectural requirement. Thus, while supporting previous WSOL approaches, L2-CAF introduces a new degree of freedom. This flexibility is vital to explore attention visualization and WSOL beyond standard supervised classification networks.

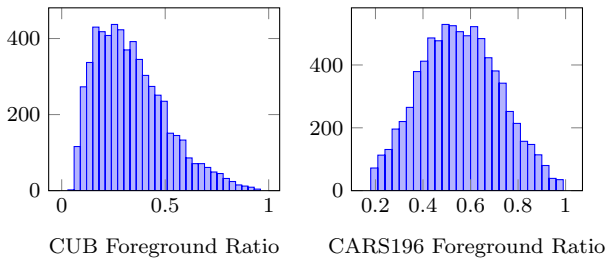


Figure 4.5: Histogram of the foreground objects' bounding box size relative to the whole image in CUB and CARS196 datasets. CUB birds tend to occupy less than 50% of the whole image (left-skewed), while the Stanford cars are normally distributed.

4.4.2 WSOL Using Retrieval Networks

Weakly supervised object localization provides a quantitative evaluation metric for attention visualization approaches. The ability to localize attention for various architectures is a core advantage of L2-CAF. In this subsection, we quantitatively evaluate L2-CAF against Grad-CAM. We employ the class oblivious formulation (Eq. 4.1) using the last convolution layer and the raw embedding (before unit-circle normalized) as endpoints.

Datasets: We employ CUB-200-2011 birds [142] and Stanford CARS196 [69] retrieval datasets, *i.e.*, standard retrieval datasets [95, 143, 16, 146]. Both datasets provide the ground truth bounding box annotation. They pose several challenges for foreground objects' localization. Birds are not naturally rectangular; discriminative parts (*e.g.*, head [14]) occupy a small part of the body. Cars pose a similar challenge in terms of relatively smaller discriminative parts (*e.g.*, wheel) relative to the whole body. Figure 4.5 depicts the ratio of the ground truth bounding box to the whole image size for both datasets.

Evaluation metrics: For retrieval, we utilize both Recall@1 (R@1) and the Nor-

Table 4.3: Triplet (TL) and N-pair (NP) losses' quantitative retrieval evaluation using NMI and Recall@1 on CUB-200-2011 and CARS196. Quantitative localization accuracy evaluation using the 0.5 intersection over union (IoU) criterion. △ column indicates the absolute localization improvement margin relative to the vanilla Grad-CAM.

			CUB-200-2011				CARS196			
			Retrieval		Localization		Retrieval		Localization	
Method	Backbone	Loss	NMI↑	R@1↑	LOC↑	△	NMI↑	R@1↑	LOC↑	△
Grad-CAM	GoogLeNet	TL	0.582	47.75	20.63	-	0.532	54.55	32.38	-
Grad-CAM-abs	GoogLeNet	TL	0.582	47.75	22.96	+2.33	0.532	54.55	46.21	+13.82
L2-CAF (ours)	GoogLeNet	TL	0.582	47.75	**29.63**	+9.00	0.532	54.55	**54.10**	+21.72
Grad-CAM	ResNet	TL	0.601	50.06	16.15	-	0.565	61.55	31.77	-
Grad-CAM-abs	ResNet	TL	0.601	50.06	29.49	+13.34	0.565	61.55	56.32	+24.55
L2-CAF (ours)	ResNet	TL	0.601	50.06	**39.28**	+23.13	0.565	61.55	**61.27**	+29.50
Grad-CAM	GoogLeNet	NP	0.583	48.95	14.13	-	0.597	65.23	28.29	-
Grad-CAM-abs	GoogLeNet	NP	0.583	48.95	19.87	+5.74	0.597	65.23	55.01	+26.72
L2-CAF (ours)	GoogLeNet	NP	0.583	48.95	**30.50**	+16.37	0.593	65.23	**64.85**	+36.56
Grad-CAM	ResNet	NP	0.580	47.92	11.92	-	0.609	67.61	32.42	-
Grad-CAM-abs	ResNet	NP	0.580	47.92	26.67	+14.75	0.609	67.61	61.62	+29.20
L2-CAF (ours)	ResNet	NP	0.580	47.92	**38.69**	+26.77	0.609	67.61	**67.35**	+34.93

malized Mutual Information (NMI) metrics. For localization, we follow the same evaluation procedure in [164, 120] for classification networks. We replace the top-1 by R@1 metric to decide if the network's output is correct or not. The same IoU $> 50\%$ criterion is used to evaluate localization.

Vanilla Grad-CAM baseline: To evaluate L2-CAF quantitatively, we extend the classification Grad-CAM to deal with retrieval networks. The Grad-CAM class-discriminative localization map M^c has been proposed as follows

$$M^c = RELU\left(\sum_k \alpha_k^c A^k\right) \tag{4.4}$$

$$\alpha_k^c = \frac{1}{w \times h} \sum_{i=0}^{w} \sum_{j=0}^{h} \frac{\partial y^c}{\partial A_{i,j}^k}, \tag{4.5}$$

where $M^c \in R^{w \times h}$ for any class c, $A \in R^{w \times h \times k}$ is a convolutional feature map with k channels. $\alpha^c \in R^k$ quantifies the k^{th} channel's importance for a target class c.

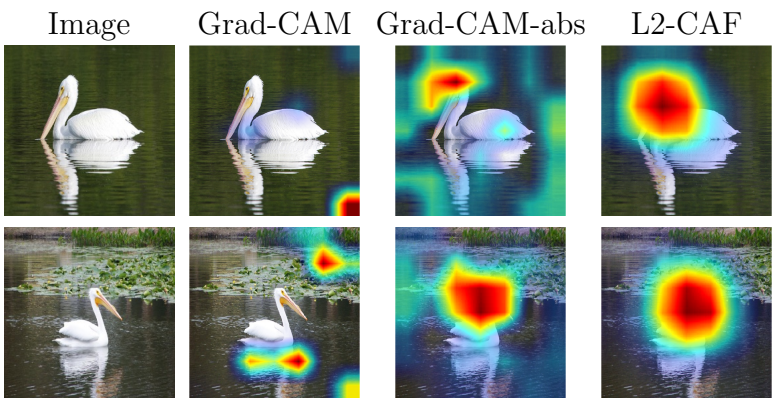


Figure 4.6: Qualitative attention evaluation for different visualization approaches on retrieval networks. Both Grad-CAM variants suffer near images' corners.

Basically, $\sum_k \alpha_k^c A^k$ provides a weighted sum of the feature maps (A) for class c. α_k^c is computed using the gradient of the score for class y^c with respect to the feature maps A^k.

To support a retrieval network, we utilize the gradient of the output embedding $\frac{\partial y}{\partial A_{i,j}^k}$ instead of the class score $\frac{\partial y^c}{\partial A_{i,j}^k}$ as follows

$$\alpha_k^y = \frac{1}{w \times h} \sum_{i=0}^{w} \sum_{j=0}^{h} \frac{\partial y}{\partial A_{i,j}^k}, \tag{4.6}$$

we denote this formulation as *vanilla Grad-CAM* for retrieval. We compute $\frac{\partial y}{\partial A_{i,j}^k}$ using tf.gradients [1].

Grad-CAM-abs baseline: The Vanilla Grad-CAM is largely inferior for retrieval networks because of the $RELU$ in Eq. 4.4. $RELU$ is introduced for classification networks to emphasize feature maps that have a positive influence on the class of interest y^c, assuming pixels with negative gradient belong to other classes. This assumption is valid for classification but invalid for retrieval. Therefore, we further modify the Grad-CAM formulation by replacing the $RELU$ with the absolute

function *abs*. This Grad-CAM-abs baseline is defined as follows

$$M^y_{abs} = abs\left(\sum_k \alpha^y_k A^k\right). \tag{4.7}$$

Implementation Details For Retrieval Networks: To evaluate the localization performance quantitatively, we leverage both triplet [118] and N-pair [124] ranking losses. We use the default settings for each loss; the N-pair's embedding is unnormalized while the triplet loss's embedding is normalized to the unit-circle and a margin $m = 0.2$ is utilized. We employ ResNet-50 [44] and GoogLeNet [131] as backbones. These are standard architectures for evaluating ranking losses [124, 143, 95]. Both architectures are trained for 5K iterations. VGG architecture is omitted because it overfits on these datasets. Similar to Hermans *et al.* [48], the last convolution layer is followed by a global average pooling layer then a single fully connected layer, *i.e.*, a feature embedding $\in R^{128}$.

Results: Table 4.3 presents a quantitative evaluation for both retrieval and localization performance. ResNet-50 has more parameters than GoogLeNet; and is marginally better in terms of retrieval. Generally, N-pair loss outperforms triplet loss. Cars are rectangular and thus simpler than CUB birds for bounding box localization. The localization error is highly correlated and upper-bounded by the retrieval performance (R@1). Grad-CAM-abs outperforms the vanilla Grad-CAM for retrieval. L2-CAF brings further localization improvement.

Figure 4.6 qualitatively compares different localization approaches. We found that feature maps at the images' corners can have a high positive gradient, while

Figure 4.7: Qualitative localization evaluation on CUB-200-2011 and CARS196 using a retrieval network trained with a triplet loss. The green and blue bounding boxes indicate the ground truth and the L2-CAF bounding boxes, respectively.

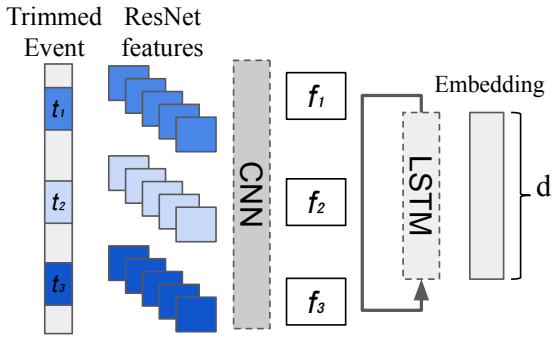


Figure 4.8: A convolution architecture to embed autonomous navigation videos. This network employs a ranking loss to learn a feature embedding and a recurrent layer for temporal modeling. The CNN layer is shared across the three frames. The attention filters (f_1, f_2, f_3) are used during attention visualization only.

the feature maps at the foreground object can have a high negative gradient. It is a common practice to embed images into the unit-circle, *i.e.*, some images are embedded in the negative space. When this happens, the vanilla Grad-CAM ignores the foreground objects. Grad-CAM-abs handles negative gradient better but still suffers around the corners. Grad-CAM inferior behavior around the corner is qualitatively reported in [29]. This undesirable behavior degrades Grad-CAM's WSOL performance. Figure 4.7 shows a qualitative localization evaluation on both datasets. For CUB-200-2011, our estimated bounding box (blue) tends to be centered around the birds' heads.

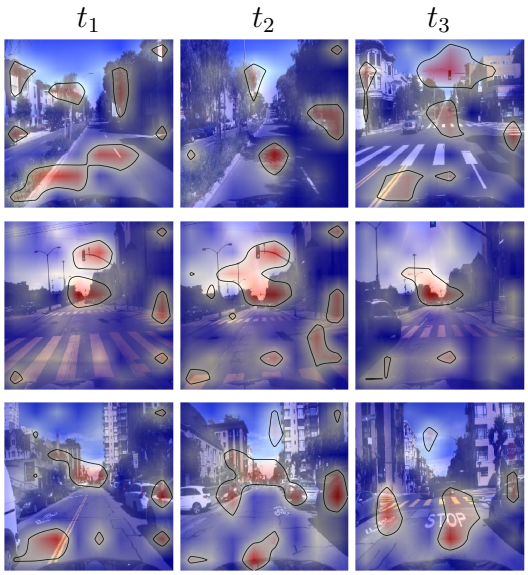


Figure 4.9: The recurrent network's attention visualization at different time steps using heatmaps. Each row depicts three frames sampled from an action video. Contours highlight regions with higher attention. The network attends to the spatial locations of traffic lights and road signs. This figure is best viewed on a screen (color and zoom).

4.4.3 Recurrent Networks' Attention

This subsection illustrates how to visualize attention for temporally fused video frames through the Honda driving dataset (HDD) [106]. HDD is a video dataset for reasoning about drivers' actions (events) like crossing intersections, making left and right turns. A key objective is modeling the subtle intra-action (events) variations without explicit fine-grained labeling. For instance, an autonomous navigation application with a left-turn query video should differentiate smooth left-turns maneuvers from those interrupted by crossing pedestrians. A retrieval network models these intra-action variations through a feature embedding.

Figure 4.8 presents a recurrent retrieval network for video embedding. Given a trimmed video event, three frames are sampled at t_1, t_2, and t_3. To enable a large

training mini-batch for triplet loss, a pre-trained ResNet is employed to extract convolutional features for every frame. The extracted ResNet features are fed into a trainable shallow CNN. The resulting convolutional features are temporally fused using an LSTM [33, 50].

After training, we employ three L2-CAF filters (f_1, f_2, and f_3) to visualize attention, *i.e.*, one filter per frame. These filters are inserted between the shallow CNN and the LSTM layers *during inference only*. To ground attention in each frame, we optimize each filter independently. Concretely, we pass the first frame's features through f_1 and optimize f_1 while feeding the second and third frames' features normally, *i.e.*, f_2 and f_3 are inactive. After f_1 converges, we deactivate it and optimize the next filter f_2 and so on. After optimizing all filters, each filter provides an attention map for the corresponding frame.

Figure 4.9 presents our qualitative evaluation. In the first row at t_3, the network attention is drawn to the traffic lights and double yellow lane marks. Similarly, the second row shows attention drawn toward the traffic light at $t_{1,2}$. The final row shows an interesting case at t_3 where the attention is drawn to the stop sign and also to the frame's top center, which is the typical location for a traffic light. Through visualization, we can see that the network uses traffic lights, signs, and road signs as discriminative features. We found that a Mast Arm (L-shaped) traffic light is easier to detect by a neural network compared to a straight pole traffic light. The variable height of a straight pole traffic light poses a challenge for neural networks. For instance, the network attends to the right side of the frame multiple times at different heights in the second row at $t_{1,2}$.

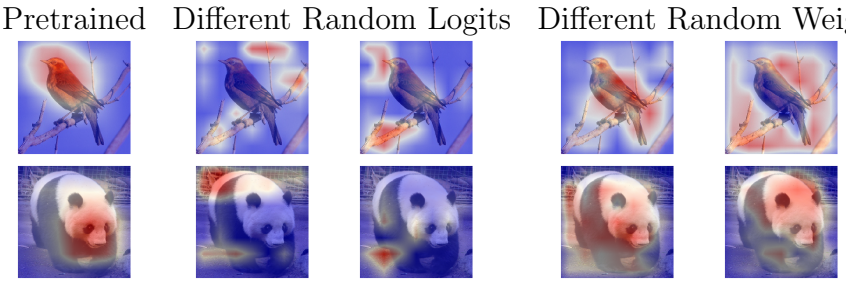


Figure 4.10: Sanity checks [2]. First column visualizes attention using a pretrained network–nothing random. Columns two to five visualize attention when logits and weights (all-layers) are randomized. Different random initializations generate different heatmaps.

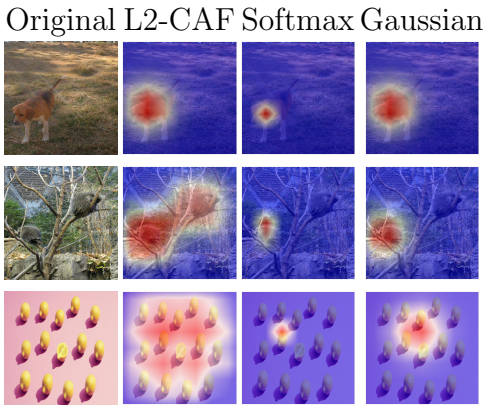


Figure 4.11: Qualitative evaluation for alternative constraints. Softmax offers a sparse result while the Gaussian filter assumes a single mode. L2-CAF supports multi-mode.

4.4.4 Ablation Study

This subsection provides sanity checks for saliency maps [2], then evaluates alternative attention constraints, and finally presents a timing analysis.

Sanity Checks: Figure 4.10 shows how L2-CAF is affected by randomly initialized logits-layer or weights (all layers). Sanity checks [2] emphasize a high dependency between the optimized L2-CAF (heatmap) and the network's weights.

Alternative Attention Constraints: We qualitatively compare the L2-CAF with

both softmax and Gaussian constraints. These are selected for their differentiability, simplicity, and usability in recent literature. Other filtering alternatives (*e.g.*, L1-Norm) are also feasible. Softmax is a typical attention mechanism for image captioning [148, 62] and machine translation [139]. In these problems, the softmax attention module is employed *recurrently* on a single image frame or an input sentence for every output word. This fits the softmax's sparse nature. Gaussian filters have been utilized for temporal action localization [81, 101]. They are denser (more relaxed) compared to softmax but also assume a single mode (elliptical shape). To localize objects in images using a Gaussian constraint, we optimize the filter's mean $\mu \in R^2$ while fixing the covariance matrix $\sigma \in R^{2\times2}$ to the identity matrix. σ must be constrained to avoid a degenerate solution where the Gaussian becomes a uniform distribution, *i.e.*, $\sigma \rightarrow \infty$. All filters are optimized using the class-oblivious formulation (Sec 4.3.1).

Figure 4.11 provides a qualitative evaluation using GoogLeNet architecture and three attention constraints. The L2-CAF identifies key region(s) for a network's output with a single glimpse. The filter prioritizes these regions quantitatively. L2-CAF supports a large spectrum of neural networks as a post-training inspection tool. It supports complex architectures including, but not limited to, encoder-decoder [66, 109], generative [36], and U-shaped architectures [112, 74]. It neither undermines the performance nor raises the inference cost. L2-CAF is not the fastest attention visualization approach but is computationally cheap.

Timing Analysis: Speed is the main limitation of our iterative formulation. Figure 4.12 presents a timing analysis for L2-CAF. The y-axis denotes the processing

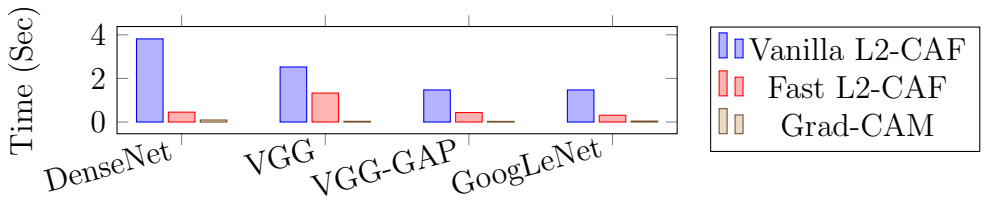


Figure 4.12: Time analysis for the L2-CAF. The fast L2-CAF brings a significant speed-up while solving the same optimization problem.

time per frame in seconds. The vanilla L2-CAF uses the default endpoints (x, $NT(x)$), while the fast L2-CAF uses the output of the last convolution layer and the logits $NT(x)$ as endpoints. The vanilla and fast L2-CAF are equivalent optimization problems but the fast L2-CAF provides a significant speed-up. The three fully connected layers in VGG, between the last convolution layer and the logits, limit the fast optimization technique. VGG-GAP replaces these fully connected layers with an average pooling layer, so its speed is similar to GoogLeNet. Fast L2-CAF takes ≈ 0.4 and 0.3 seconds per frame on VGG-GAP and GoogLeNet, respectively. The DenseNet-161's speed-up is maximum because the fast L2-CAF skips all dense blocks before the last convolution layer.

4.5 Conclusion

We have introduced the unit L2-Norm constrained attention filter (L2-CAF) as a visualization tool that works for a large spectrum of neural networks. L2-CAF neither requires fine-tuning nor imposes architectural constraints. Weakly supervised object localization is utilized for quantitative evaluation. State-of-the-art results are achieved on both standard and fine-tuned classification architectures. For retrieval

networks, L2-CAF significantly outperforms Grad-CAM baselines. Ablation studies highlight L2-CAF's superiority to alternative constraints and analyze L2-CAF's computational cost.

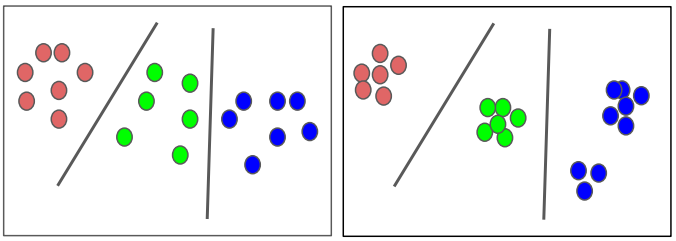
Chapter 5: Boosting Standard Classification Architectures Through a Ranking Regularizer

We employ triplet loss as a feature embedding regularizer to boost classification performance. Standard architectures, like ResNet and Inception, are extended to support both losses with minimal hyper-parameter tuning. This promotes generality while fine-tuning pretrained networks. Triplet loss is a powerful surrogate for recently proposed embedding regularizers. Yet, it is avoided due to large batch-size requirement and high computational cost. Through our experiments, we re-assess these assumptions.

During inference, our network supports both classification and embedding tasks without any computational overhead. Quantitative evaluation highlights a steady improvement on five fine-grained recognition datasets. Further evaluation on an imbalanced video dataset achieves significant improvement. Triplet loss brings feature embedding capabilities like nearest neighbor to classification models.

5.1 Introduction

Standard convolutional architectures [44, 131] learn powerful representation for classification. Pretrained ImageNet [22] weights scale their strength through fine



(a) Softmax Loss (b) Triplet Loss Regularizer

Figure 5.1: Softmax learns powerful representations with limited embedding regularization. Triplet loss promotes better embedding without an explicit number of class centers.

tuning to novel domains and relax the large labeled dataset requirement. Yet, the learned representation through softmax attains limited intra-class compactness and inter-class separation. To advocate for a better embedding quality, we propose a two-head architecture. We leverage triplet loss [118] as a classification regularizer. It promotes a better feature embedding by attracting similar and repelling different classes as shown in Figure 5.1. This embedding also raises classification model interpretability by enabling nearest neighbor retrieval.

Embedding losses have been successfully applied in conjunction with softmax loss as regularizers. For example, center loss [145] was proposed for better face recognition efficiency. Magnet loss [110] generalizes the unimodality assumption of center loss. A recent triplet-center loss (TCL) [47] uses only a unimodal embedding but introduced a repelling force between class centers, *i.e.*, inter-class margin maximization. All these methods assume a fixed number of class centers (embedding modes) for all classes.

Unlike the aforementioned approaches, the standard triplet loss requires no explicit number of embedding modes. Thus, it avoids computing class centers while

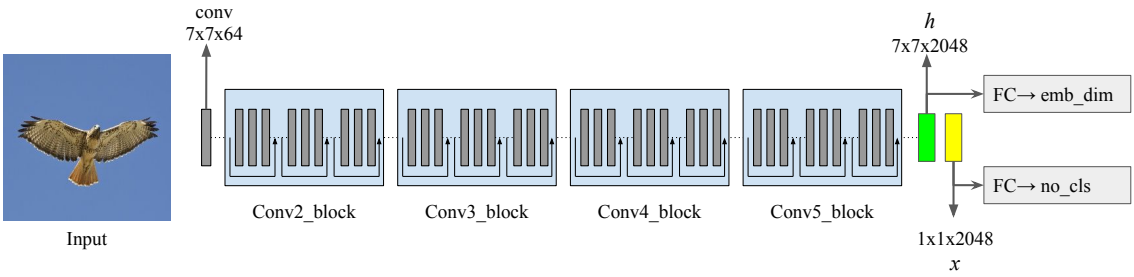


Figure 5.2: Our proposed two-head architecture builds on standard networks – ResNet used for visualization, $x_{input} = pool(h_{input})$. Besides computing classification logits, the pre-logits layer supports the embedding head. Softmax and triplet losses are applied to the classification logits and embedding features, respectively.

promoting intra-class compactness and inter-class margin maximization. Surprisingly, recent papers [145, 47] do *not* report the softmax+triplet loss quantitative evaluation. Assumptions about large training batch requirement [118] for faster convergence or high batch-processing complexity, to compute pairwise distance matrix, have hindered triplet loss's adoption. Our experiments reassess these assumptions through multiple triplet loss sampling strategies.

To incorporate embedding losses, previous approaches employ loss-specific architectures. This custom setting is imperfect for the softmax baseline as it omits the pre-trained ImageNet weights. Through our proposed seamless integration into standard CNNs, we push our baselines' limits. We introduce an embedding head similar to the classification head. Each head applies a single fully connected (FC) layer on the pre-logit convolutional layer features. Figure 5.2 shows our two head architecture where the pre-logit convolutional features support both softmax and triplet losses for classification and embedding respectively. This integration boosts classification performance while promoting better embedding.

We evaluate our approach on various classification domains. The first is a fine-grained visual recognition (FGVR) across five datasets. The second domain is an ego-motion action recognition task with high class imbalance. Large improvements (1-4%) are achieved in both domains. Evaluation on multiple architectures with the same hyper-parameters highlights our approach's generality. The large batch size requirement represents a key challenge for triplet loss adoption; Schroff *et al.* [118] use a batch-size $b = 1800$ and trained on a CPU cluster for 1,000 to 2,000 hours. In our experiments, we show that using a small batch size $b = 32$ still improves performance. A further qualitative evaluation highlights beneficial qualities like nearest neighbor retrieval added to standard classification architectures. In summary, the key contributions of this paper are:

1. A two-head architecture proposal that uses triplet loss as a regularizer to boost standard architectures' performance through promoting a better feature embedding.
2. A re-evaluation of the large batch size requirement and high computational cost assumptions for triplet loss,
3. Enable better nearest neighbor retrieval on classification architectures.

5.2 Related Work

Visual recognition deep networks employ softmax loss as follows

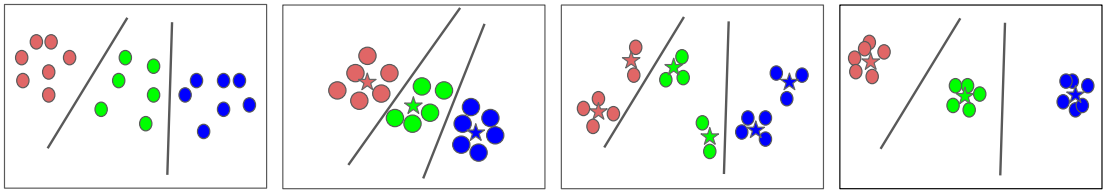
$$L_{\text{soft}} = -\sum_{i=1}^{b} \log \frac{e^{W_{y_i}^T x_i}}{\sum_{j=1}^{n} e^{W_j^T x_i}}, \tag{5.1}$$

where $x_i \in R^d$ denotes the ith deep feature, belonging to the y_ith class. In standard architectures, x_i is the pre-logit layer; the result of flattening the pooled convolutional features as shown in Figure 5.2. $W_j \in R^d$ denotes the jth column of the weights $W \in R^{d \times n}$ in the last fully connected layer. b and n are the batch size and the number of class respectively. The softmax loss only cares about separating samples from different class. It disregards properties like intra-class compactness and inter-class margin maximization. Embedding regularization is one way to tackle this limitation. Figure 5.3 depicts different embedding regularizers; all require an explicit number of embedding modes.

5.2.1 Center Loss

Wen *et al.* [145] propose center loss to minimize intra-class variations. By maintaining a per class representative feature vector $c_{yi} \in R^d$, the novel loss term in equation 5.2 is proposed. The class centers are computed by averaging corresponding class features. They are updated after every training mini-batch. To avoid perturbations caused by noisy samples, a hyper-parameter α controls the learning rate of the centers, *i.e.*, moving average.

$$L_{\text{cen}} = \frac{1}{2} \sum_{i=1}^{b} \parallel x_i - c_{yi} \parallel_2^2. \tag{5.2}$$



(a) Softmax Loss (b) Center Regularizer (c) Magnet Regularizer (d) TCL Regularizer

Figure 5.3: Visualization of softmax and feature embedding regularizers. Softmax separates samples with neither class compactness nor margin maximization considerations. Center loss promotes unimodal compact class while magnet loss supports multi-modal embedding. Triplet center loss (TCL) strives for unimodal, margin maximization and class compactness. The computed classes' centers are depicted using a star symbol

5.2.2 Magnet Loss

Rippel *et al.* [110] propose a center loss term supporting multi-modal embedding, dubbed magnet loss. It computes K class representatives, *i.e.*, K-clusters per class. Each sample is iteratively assigned to one of the K clusters and pushed towards its center. The magnet loss adaptively sculpts the representation space by identifying and enforcing intra-class variation and inter-class similarity. This is formulated as follows

$$L_M = \frac{1}{N}\sum_{i=1}^{N} -\log \frac{\exp(\frac{-1}{2\sigma^2}\| x_i^k - \mu_k^c \|_2^2 - \alpha)}{\sum_{c \neq C(x_i^k)} \sum_{k=1}^{K} \exp(\frac{-1}{2\sigma^2}\| x_i^k - \mu_k^c \|_2^2 - \alpha)}, \tag{5.3}$$

where N and K are the number of samples and clusters per class respectively. $x_i^k \in R^d$ denotes the ith deep feature, belonging to cluster k in the y_ith class, $\mu_k^c \in R^d$ is the kth cluster center belonging to class c. Finally $\sigma^2 = \frac{1}{N-1}\sum \| x_i^k - \mu_k^c \|_2^2$ is the variance of all samples from their respective centers. One criticism of magnet loss is the complexity overhead to maintain multiple clusters per class and their

assigned samples. Moreover, the constant number of clusters per-class disputes with imbalanced data distributions.

5.2.3 Triplet Center Loss

While promoting class compactness, the center loss depends on the softmax loss supervision signal to push different classes apart. The learned features optimized with the softmax loss supervision signal are not discriminative enough, *i.e.*, no explicit repelling force pushes different classes apart. Inter-class clusters can overlap due to missing an explicit inter-class repelling incentive. He *et al.* [47] propose triplet center loss (TCL) to avoid this limitation. By maintaining a per class center $c_{yi} \in R^d$ similar to [145], TCL is formulated as follows

$$L_{\text{tcl}} = \sum_{i=1}^{b} \left[(D(x_i, c_{yi}) - \min_{j \neq i} D(x_i, c_{yj}) + m) \right]_+, \tag{5.4}$$

where m is a separating margin, $[.]_+ = max(0, .)$ and $D(.)$ represents the squared Euclidean distance function.

Triplet loss is a well-established surrogate for TCL. It achieves the intra and inter-class embedding objectives without computing class centers. Yet, it is largely avoided for its computational complexity and large training batch requirement assumptions. In the experiment section, we address these concerns and evaluate the utility of triplet loss as a regularizer. Our approach is evaluated on the challenging FGVR task where intra-class overwhelm inter-class variations. Further evaluation on the Honda driving dataset (HDD) demonstrates our approach's competence on

an imbalanced video dataset. Triplet loss regularization not only lead to higher classification accuracy but also enables better feature embedding.

5.3 The Triplet Loss Regularizer

The next subsection introduces triplet loss [118] as a softmax loss regularizer. Then, we explain our standard architectural extension to integrate a ranking loss.

5.3.1 Triplet Loss

Triplet loss [118] has been successfully applied in face recognition [118, 116] and person re-identification [18, 128, 111]. In both domains, it is used as a feature embedding tool to measure similarity between objects and provide a metric for clustering. In this work, we utilize triplet loss as a classification regularizer. It is more efficient than contrastive loss [40, 76], and less computationally expensive than quadruplet [57, 16] and quintuplet [56] losses. While the pre-logits layer learns better representations for classification using the softmax loss, triplet loss promotes a better feature embedding. Equation 5.5 shows the triplet loss formulation

$$L_{\text{tri}} = \frac{1}{b} \sum_{i=1}^{b} [(D(a_i, p_i) - D(a_i, n_i) + m)]_+, \tag{5.5}$$

where an anchor image's embedding a of a specific class is pushed closer to a positive image's embedding p from the same class than it is to a negative image's embedding n of a different class. Equation 5.6 is our loss function with a balancing hyper-parameter λ.

$$L = L_{\text{soft}} + \lambda L_{\text{tri}}. \tag{5.6}$$

Sampling: Triplet loss performance is dependent on its sampling strategy. We evaluate both the hard [48] and semi-hard [118] sampling strategies. In semi-hard negative sampling, instead of picking the hardest positive-negative samples, all anchor-positive pairs and their corresponding semi-hard negatives are considered. Semi-hard negatives satisfy equation 5.7. They are further away from the anchor than the positive exemplar, yet within the banned margin m.

$$D(a,p) < D(a,n) < D(a,p) + m. \tag{5.7}$$

Figure 5.4 shows a triplet loss tuple and highlights the different types of negative exemplars: easy (n_2), semi-hard (n_1) and hard (n_3) negatives. An easy negative satisfies the margin constraint and suffers a zero loss. Unlike hard-sampling, semi-hard sampling supports a multi-modal embedding. Hard sampling picks the farthest positive and nearest negative without any consideration for the margin. In contrast, Figures 5.5 illustrates how semi-hard sampling ignores hard negatives. Two classes, red and green, are embedded into one and two clusters respectively. A hard sampling strategy pulls the farthest positive from one cluster to the anchor in the other cluster, i.e. promotes a merge. The semi-hard sampling strategy omits this tuple because the negative sample is nearer than the positive.

The existence of a semi-hard negative is not guaranteed in small batches,

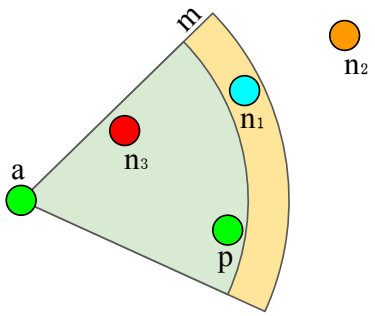


Figure 5.4: Triplet loss tuple (anchor, positive, negative) and margin m. Hard, semi-hard and easy negatives highlighted in red, cyan and orange, respectively.

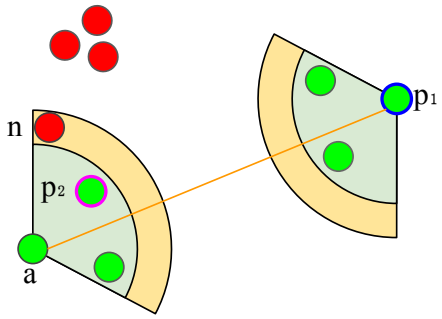


Figure 5.5: Hard sampling promotes unimodal embedding by picking the farthest positive and nearest negative $(a, p1, n)$. Semi-hard sampling picks $(a, p2, n)$ and avoids any tuple (a, p, n) where n lies between a and p.

especially near convergence. Thus, we prioritize negative exemplars as illustrated in Figure 5.4. First priority is given to semi-hard (n_1), then easy (n_2) and finally hard negatives (n_3).

5.3.2 Two-Head Architecture

Standard convolutional architectures, with ImageNet [22] weights, are employed in various applications for their powerful representation. We seek to leverage pre-trained standard networks for their advantages in tasks like fine-grained visual recognition [78, 71, 70]. This key integration promotes the generality of our ap-

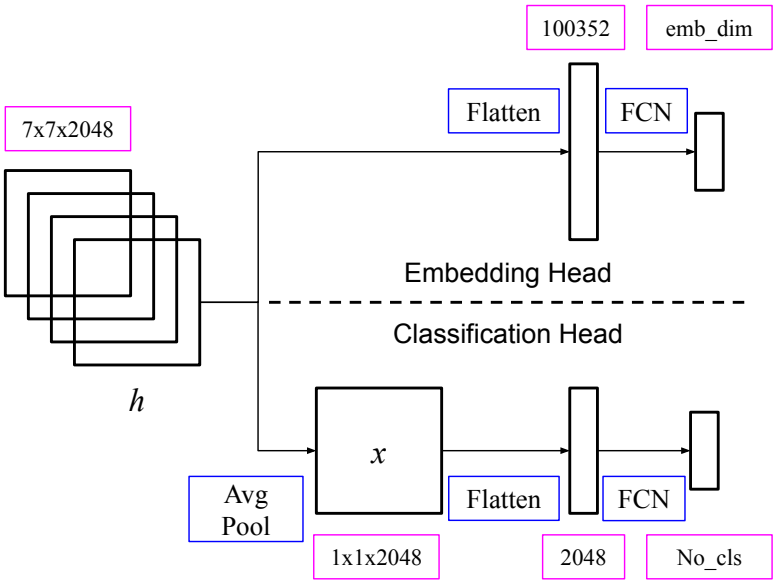


Figure 5.6: Our proposed two-head architecture. The last convolutional feature map (h) supports both embedding and classification heads. Operations and dimensions are highlighted with blue and pink colors, respectively. ResNet-50 dimensions used for illustration.

proach and distances our work from [145, 47, 130] which use custom architectures. Through experiments, we demonstrate how triplet loss achieves superior classification efficiency compared to center loss.

Unlike VGG [121], recent architectures [44, 132, 58] employ a convolutional layer before the classification head. To generate logits, the classification head pools the convolutional layer features, flatten them, then utilize a customizable fully connected layer to support various numbers of classes. Similarly, we integrate triplet loss to regularize embedding as shown in Figure 5.6. Before pooling, we flatten the convolutional layer features then apply another fully connected layer W_{emb} to

	Flowers-102 [92]	Aircrafts [86]	NABirds [138]	Stanford Cars [69]	Stanford Dogs [63]
Num Classes	102	100	550	196	120
Avg samples Per Class	10	100	43.5	41.55	100
Train Size	1020	3334	23929	8144	12000
Val Size	1020	3333	N/A	N/A	N/A
Test Size	6149	3333	24633	8041	8580
Total Size	8189	10000	48562	16185	20580

Table 5.1: Statistics of five FGVR datasets and their corresponding train, validation and test splits.

generate embeddings as illustrated in equation 5.9.

$$\text{Logits} = W_{\text{logits}} * \text{flatten}(x) \tag{5.8}$$

$$\text{Embedding} = W_{\text{emb}} * \text{flatten}(h), \tag{5.9}$$

where $x = \text{pool}(h)$. Orderless pooling, like averaging, disregard spatial information. Thus, a fully connected layer W_{emb} applied on h has a better representation power. The final embedding is normalized to the unit-circle and the square Euclidean distance metric is employed. During inference, the two-head architecture enables both classification and retrieval with negligible overhead.

5.4 Experiments

5.4.1 Evaluation on FGVR

Datasets: We evaluate our approach on five FGVR datasets. These datasets comprise both make/model classification and wildlife species. The **Aircrafts** dataset

contains 10,000 images of aircraft spanning 100 aircraft-models. The finer level differences between models makes visual recognition challenging. The **NABirds** dataset contains 48,562 images across 550 visual categories of North American birds. The **Flower-102** dataset contains 8189 images across 102 classes. The **Stanford Cars** dataset contains 16185 images across 196 car classes that represent variations in car make, model, and year. Finally, the **Stanford Dogs** dataset has 20,580 images across 120 breeds of dogs. These datasets provide challenges in terms of large intra-class but small inter-class variations. Table 5.1 summarizes the datasets' size, number of classes and splits.

Baselines: We evaluate our approach against two baselines: (1) Single head softmax; (2) Two-head leveraging center loss [145] with it's proposed hyper-parameters $\lambda = 0.003$ and $\alpha = 0.5$. We found Magnet loss [110] implementation computationally expensive. It applies k-means to cluster all training samples after each epoch, *i.e.*, $O(N^2)$ where N is the train split size. For triplet loss, both hard [48] and semi-hard [118] sampling variants are evaluated. By default, our hyper-parameter $\lambda = 1$ and embedding normalized to the unit circle with dimensionality $d_{\text{emb}} = 256$. With triplet hard sampling, a soft margin between classes is imposed by the softplus function $\ln(1 + \exp(\bullet))$. It is similar to the hinge function $\max(\bullet, 0)$ but it decays exponentially instead of a hard cut-off. With triplet semi-hard sampling, we employ the hard margin $m = 0.2$ as proposed by [118]

All experiments are conducted on Titan Xp 12GB GPU with batch-size $b = 32$. All networks are initialized with ImageNet weights, and then fine-tuned. Momentum optimizer is utilized with momentum 0.9 and a polynomial decaying learning

rate $lr = 0.01$. We quantitatively evaluate our approach on three architectures: (1) ResNet-50 [44] and (2) DenseNet-161 [58] both trained for 40K iterations, and (3) Inception-V4 [133] trained for 80K iterations. While early stopping is a valid regularization form to avoid a fixed number of training iteration, not all datasets provide a validation split as illustrated in table 5.1. The chosen number of training iterations achieve comparable results with recent FGVR softmax baselines [77, 71, 25].

To evaluate our approach, our training batches contain both positive and negative samples. We follow the batch construction procedure proposed by Hermans *et al.* [48]. A class is uniformly sampled then $K = 4$ sample images, with resolution 224×224, are randomly drawn. Training images are augmented online with random cropping and horizontal flipping. This process iterates until a batch is complete. Table 5.2 presents our fine-tuning quantitative evaluation on the five datasets. Our two-head architecture with hard triplet loss achieves large steady (1-4%) improvement on ResNet-50. Similar trend appears with Inception-V4 but suffers an interesting fluctuation between hard and semi-hard triplet loss. Section 5.4.3 reflects on this phenomena through a quantitative embedding analysis. Vanilla DenseNet-161 achieves comparable state-of-the-art results on all FGVR datasets, yet triplet loss regularizer maintains a steady trend of performance improvement.

Center loss achieves an inferior classification performance especially on the Dogs dataset – a lag $\approx 4\%$ behind vanilla softmax on Inception-V4 and DenseNet-161. The single mode embedding assumption is valid for face recognition [145] and vehicle re-identification [80] because different images for the same identify belong to a single cluster. However, when working with categories of high intra-class variations,

Database	Cars	Flowers	Dogs	Aircrafts	Birds
			ResNet-50		
Softmax	85.85	85.68	69.76	83.22	64.23
Two-Head (Center)	88.23	85.00	70.45	84.48	65.5
Two-Head (Semi)	88.22	85.52	70.69	85.08	65.20
Two-Head (Hard)	**89.44**	**86.61**	**72.70**	**87.33**	**66.19**
			Inception-V4		
Softmax	88.42	88.22	77.20	86.76	74.90
Two-Head (Center)	89.50	88.35	70.83	87.78	76.86
Two-Head (Semi)	**89.72**	88.69	**77.71**	88.59	**76.99**
Two-Head (Hard)	89.06	**90.66**	75.97	**89.04**	76.57
			DenseNet-161		
Softmax	91.64	92.56	**81.58**	89.13	78.69
Two-Head (Center)	89.08	92.58	77.02	**89.97**	79.05
Two-Head (Semi)	**92.36**	**93.65**	80.89	89.64	**79.57**
Two-Head (Hard)	92.41	93.25	81.16	89.34	79.47

Table 5.2: Quantitative evaluation on the five FGVR datasets using ResNet-50, Inception-V4, and DenseNet-161.

this assumption degenerate the feature embedding quality. Our feature embedding evaluation (Sec 5.4.3) highlights the consequence of using a single mode/cluster, for general classification problems, in terms of feature embedding instability or collapse.

Our simple but vital integration into standard architectures distance our approach from similar softmax+clustering formulations. In addition, all recent convolutional architectures share similar ending structure; the last convolutional layer is followed by an average pooling, and then a single fully connected layer. Thus, apart from the studied architectures, our secondary embedding head proposal can be applied to other architectures, *e.g.*, MobileNet [54].

5.4.2 Task Generalization

For further evaluation, we leverage the Honda Research Institute Driving Dataset (HDD) [105] for action recognition. HDD is an ego-motion video dataset for driver behavior understanding and causal reasoning. It contains 10,833 events spanning eleven event classes. Moreover, the HDD event class distribution is long-

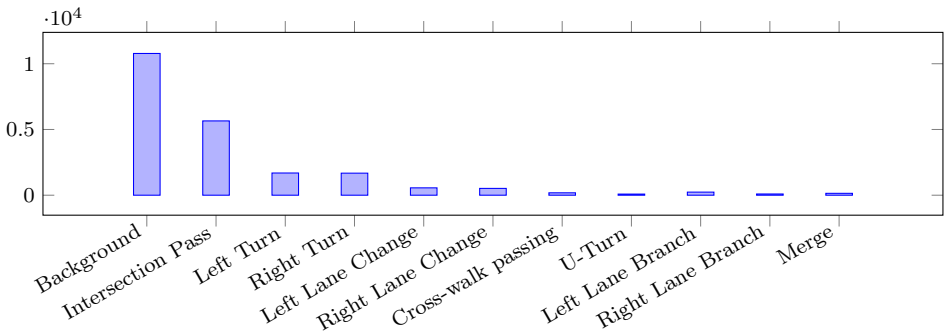


Figure 5.7: Honda driving dataset long tail class distribution

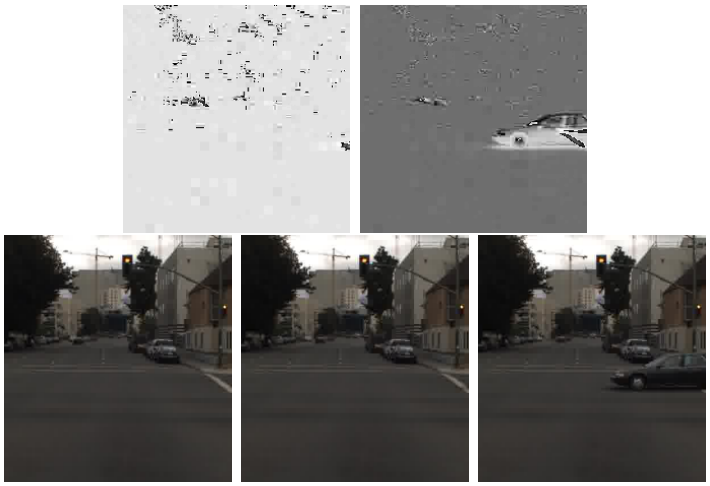


Figure 5.8: Stack of difference motion encoding. Instead of six frames, three are used for visualization purpose. The first row shows a stack of two difference frames constructed by subtracting consecutive pairs of grayscale frames in the second row. These images are best viewed in color/screen.

tailed which poses an imbalance data challenge. Figure 5.7 shows the eleven event classes with their distributions. To reduce video frames' redundancy, three frames are sampled per second, and events shorter than 2 seconds are omitted.

To leverage standard architecture for action recognition, stack of difference (SOD) motion encoding proposed by Fernando et al. [28] is adopted. While better motion encoding like optical-flow exists, the SOD is utilized for its simplicity and ability to achieve competitive results [28, 134]. Given a sequence of frames representing an event, six consecutive frames spanning 2 seconds are randomly sampled.

	Micro Acc	Macro Acc
Softmax ($b = 33$)	84.43	47.66
Two-head (Semi) ($b = 33$)	**84.93**	**53.70**
Softmax ($b = 63$)	84.45	46.53
Two-head (Semi) ($b = 63$)	**84.85**	**54.08**

Table 5.3: Action recognition quantitative evaluation on the Honda dataset. b indicates the batch-size used. Macro average accuracy highlights performance on minority classes.

They are converted to grayscale, and then every consecutive pair is subtracted to create a stack of difference $\in Z^{W \times H \times 5}$ as depicted in Figure 5.8. Standard architectures are easily adapted to this input representation by treating the SOD input as a five-channel image instead of three.

Unlike FGVR input $\in [0, 255]$, SOD $\in [-255, 255]$. Thus, we employ a randomly initialized ResNet-50 [44] architecture. It is trained for 10K iterations with $\lambda = 1$ and a polynomial decaying learning rate $lr = 0.01$. Batch sizes 33 and 63 are used to compare the vanilla softmax against our approach. To highlight performance on minority classes, both micro and macro average accuracies are reported in Table 5.3. Macro-average computes the metric for each class independently before taking the average. Micro-average is the traditional mean for all samples. Macro-average treats all classes equally while micro-averaging favors majority classes. Table 5.4 highlights the efficiency of our approach on minority classes.

	Softmax	**Two-Head**	Softmax	**Two-Head**
Event	Batch-size 33		Batch-size 63	
Background	**96.28**	95.29	**97.32**	96.28
Intersection Passing	74.61	**75.86**	74.26	**74.68**
Left Turn	**85.49**	84.87	85.18	**86.11**
Right Turn	**88.47**	87.22	**86.91**	86.60
Left Lane Change	59.40	**66.33**	55.44	**62.37**
Right Lane Change	44.79	**61.45**	40.62	**51.04**
Cross-walk Passing	**18.18**	**18.18**	**12.12**	**12.12**
U-Turn	0.00	**11.76**	0.00	**23.52**
Left Lane Branch	53.84	**64.10**	41.02	**64.10**
Right Lane Branch	0.00	**6.24**	12.49	**18.74**
Merge	3.22	**19.35**	6.45	**19.35**
Macro Accuracy	47.66	**53.70**	46.53	**54.08**

Table 5.4: Detailed evaluation on the Honda driving dataset. Our two-head architecture using semi-hard triplet loss achieves better performance on minority classes.

		NMI	R@1	R@4	R@8	R@16
Car - ResNet	CNTR	0.549	67.73	75.36	81.91	87.28
	SEMI	0.879	89.45	93.14	95.24	96.62
	HARD	**0.900**	**91.95**	**94.22**	**95.70**	**96.78**
Flowers - ResNet	CNTR	0.723	74.53	86.78	90.94	94.06
	SEMI	0.822	87.56	**94.29**	**96.39**	**97.89**
	HARD	**0.856**	**90.40**	94.00	94.84	95.64
Dogs - ResNet	CNTR	0.419	30.41	40.69	63.96	75.14
	SEMI	0.708	60.70	79.55	85.84	90.15
	HARD	**0.740**	**64.01**	**81.60**	**86.41**	**89.97**
Aircrafts - ResNet	CNTR	0.645	64.36	80.32	85.57	89.41
	SEMI	0.846	82.15	90.01	92.38	**94.45**
	HARD	**0.879**	**85.84**	**91.63**	**92.89**	93.94
NABirds - ResNet	CNTR	0.517	32.16	50.89	60.03	68.70
	SEMI	0.749	56.30	76.08	82.99	**88.30**
	HARD	**0.769**	**59.09**	**77.35**	**83.49**	88.12
Cars - Inc-V4	CNTR	0.120	2.98	5.96	8.84	13.87
	SEMI	**0.880**	**85.45**	**93.56**	**95.66**	**97.15**
	HARD	0.652	46.97	71.14	80.87	87.90
Flowers - Inc-V4	CNTR	0.183	9.01	11.97	13.82	16.13
	SEMI	0.828	88.70	94.70	96.47	**97.89**
	HARD	**0.885**	**93.66**	**96.13**	**96.96**	97.59
Dogs - Inc-V4	CNTR	0.726	65.47	76.62	79.01	81.04
	SEMI	**0.760**	**68.48**	**85.10**	**90.26**	**93.83**
	HARD	0.458	19.52	41.41	55.63	70.63
Aircrafts - Inc-V4	CNTR	0.333	27.21	36.75	42.81	49.62
	SEMI	0.872	86.53	92.35	**93.88**	**95.08**
	HARD	**0.887**	**87.79**	**92.47**	93.67	94.42
NABirds - Inc-V4	CNTR	0.209	3.77	6.26	8.29	11.50
	SEMI	**0.808**	**67.30**	**83.81**	**88.96**	**92.79**
	HARD	0.503	15.92	31.84	42.66	54.64
Cars - Dense	CNTR	**0.914**	88.93	93.97	95.01	95.65
	SEMI	0.905	88.77	**95.72**	**97.08**	**98.30**
	HARD	0.913	**89.40**	95.57	96.99	98.15
Flowers - Dense	CNTR	**0.910**	**95.23**	97.19	97.61	98.13
	SEMI	0.869	94.52	**97.90**	**98.68**	**99.14**
	HARD	0.898	87.73	91.87	92.32	92.65
Dogs - Dense	CNTR	0.795	72.03	84.11	86.55	88.39
	SEMI	0.802	73.33	88.24	92.21	**95.02**
	HARD	**0.807**	**73.99**	**88.66**	**92.44**	94.99
Aircrafts - Dense	CNTR	**0.898**	87.73	91.87	92.32	92.65
	SEMI	0.883	86.98	93.49	95.11	**96.28**
	HARD	0.889	**87.82**	**94.27**	**95.38**	96.07
NABirds - Dense	CNTR	**0.847**	**76.90**	85.37	88.03	90.57
	SEMI	0.829	72.09	86.90	**91.24**	94.35
	HARD	0.829	72.02	**87.11**	91.61	**94.70**

Table 5.5: Detailed feature embedding quantitative analysis across the five datasets using ResNet-50, Inception-V4 and DenseNet-161. Triplet with hard mining achieves superior embedding with ResNet-50 trained for 40K iterations. Semi-hard triplet is competitive and stable with Inception-V4 trained for 80K iterations. Center loss learns an inferior embedding while suffering the highest instability.

5.4.3 Retrieval Evaluation on FGVR

In the two-head architecture, the secondary embedding head brings values like an enhanced feature embedding, nearest neighbor retrieval and interpretability. Following Song *et al.* [95], we evaluate the quality of feature embedding using Recall@K metric on the test split. We also leverage the Normalized Mutual Info (NMI) score to evaluate the quality of cluster alignments. NMI $= \frac{I(\Omega,C)}{\sqrt{H(\Omega)H(C)}}$, where $\Omega = \{\omega_1, .., \omega_n\}$ is the ground-truth clustering while $C = \{c_1, ...c_n\}$ is a clustering assignment for the learned embedding. $I(\bullet, \bullet)$ and $H(\bullet)$ denotes mutual information and entropy respectively. We use K-means to compute C.

Table 5.5 presents a detailed feature embedding quantitative analysis. Triplet loss with hard-mining consistently learns the best embedding on ResNet-50. However, semi-hard sampling, on Inception-V4 and DenseNet, is stabler. Despite having an explicit rebelling force pushing negative samples away from their anchors, hard triplet mining can in practice lead to bad local minima (as can be seen in inception-V4). It can result in a collapsed mode (*i.e.*, $f(x) = 0$) [118]. Center loss suffers the same model collapse problem. It is a more vulnerable variant of hard-triplet loss, i.e., missing the repelling force. It learns an inferior embedding while suffering the highest instability. It often degenerates with Inception-V4. These conclusions follow Schroff *et al.* [118] semi-hard mining findings.

Table 5.6 compares classification and retrieval performance quantitatively. The reported classification accuracy provides an upper bound for retrieval. Retrieval and classification top 1 accuracies are comparable. Recall@4 is superior to the classifi-

	Cars	Flowers-102	Dogs	Aircrafts	NABirds
			ResNet-50		
Classification Top 1	89.44	86.61	72.70	87.33	66.19
Retrieval Top 1	91.95	90.40	64.01	85.84	59.09
Retrieval Top 4	94.22	94.00	81.60	91.63	77.35
			Inception-V4		
Classification Top 1	89.72	90.66	77.71	89.04	76.99
Retrieval Top 1	85.45	93.66	68.48	87.79	67.30
Retrieval Top 4	93.56	96.13	85.10	92.47	83.81
			DenseNet-161		
Classification Top 1	92.36	93.65	81.58	89.97	76.57
Retrieval Top 1	89.40	95.23	73.99	87.82	76.90
Retrieval Top 4	95.72	97.90	88.66	94.27	87.11

Table 5.6: Comparative quantitative evaluation between retrieval and classification as an upper bound. Both retrieval and classification accuracies are comparable. Retrieval top 4 is superior to classification top 1.

Figure 5.9: Retrieval qualitative evaluation on three FGVR datasets: Flowers-102, Aircrafts and Cars. Given a query image, the three nearest neighbors are depicted. The three consecutive rows show search results using center loss, semi-hard and hard triplet regularizers. Green and red outlines denote match and mismatch between the query and it's result respectively.

cation top 1 on all datasets. Figure 5.9 presents a qualitative retrieval evaluation across center loss, triplet semi-hard, and triplet hard regularizers.

It is challenging, for the current classification architectures, to interpret a test image misclassification. By learning image embedding through a secondary head, it becomes trivial to investigate an image's test and train splits neighborhood. Figure 5.10 shows nine (three images per odd column) misclassified test images and their corresponding nearest neighbor from the train split. The resembles between a misclassified test image and a particular training image can reveal corner cases

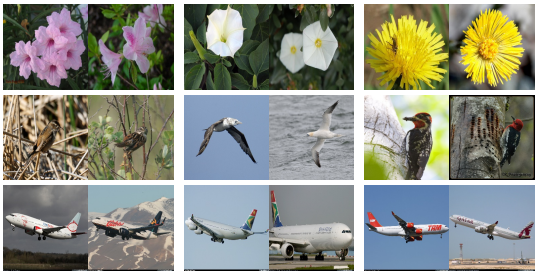


Figure 5.10: Qualitative misclassification interpretation. The odd columns show a misclassified test image while the even columns show the nearest neighbor from the *training* split.

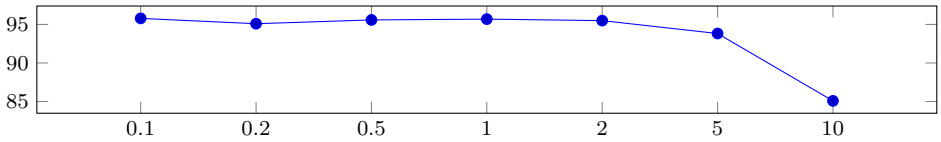


Figure 5.11: Hyper-parameter λ tuning on the Flowers-102 dataset.

omitted while collecting the data. One interesting statistic is that 79.34% of misclassified predictions, from Flowers-102 test split, match the label of their nearest training neighbor. This emphasizes the classification complexity level of FGVR.

5.4.4 Ablation Analysis

Hyper-Parameter Stability: Our approach has two hyper-parameters: λ and the embedding dimensionality d_{emb}. λ is tuned on the Flowers-102 dataset through the validation split. All hyper-parameter tuning experiments are executed for 2000 iterations. Figure 5.11 highlights λ stability within $[0.1, 2]$. A larger λ making triplet loss dominant is discouraged. Intuitively, further hyper-parameters tuning can achieves better performance.

Two-Head Time Complexity: The computational cost of the embedding head is negligible. Both sampling and backpropagation are implemented on GPU. Training

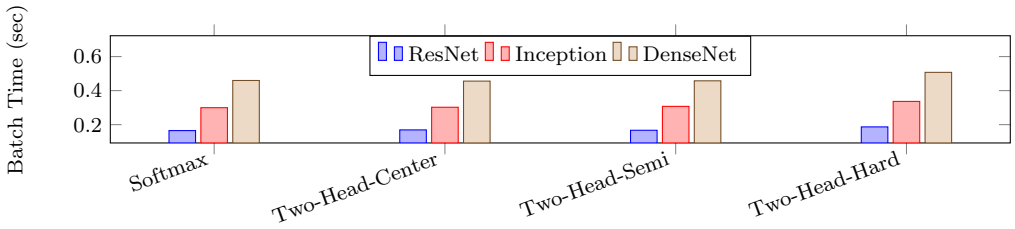


Figure 5.12: Two-head time complexity analysis on ResNet-50, Inception-V4 and DenseNet-161 using Flowers-102 dataset.

time increases by $1\%, 3\%$, and 2% for semi-hard, hard and center losses on Titan XP GPU, respectively. Figure 5.12 shows a time complexity analysis in terms of batch processing time (secs). Please note that triplet loss approaches retain from computing classes centers or enforcing a specific number of modes.

5.4.5 Discussion

Our experiments demonstrate how a two-head architecture with triplet loss outperforms a vanilla single-head softmax network. Triplet loss attains the center loss, triplet center loss and magnet loss objectives without enforcing explicit class representatives. It promotes both intra-class compactness and inter-class margin maximization. Semi-hard triplet loss relaxes the unimodal embedding constraint while maintaining stabler learning curve. Hard triplet loss achieves larger improvement margins but can suffer model collapse. Triplet loss effectively regularizes softmax and promote better feature embedding.

The two-head architecture with triplet loss is the main scope of this paper. Investigating other recent ranking losses, *e.g.*, Margin loss [146], and comparing their benefits to softmax remains an open question.

5.5 Conclusion

We propose a seamless integration of triplet loss as an embedding regularizer into standard classification architectures. The regularizer competence is illustrated on multiple datasets, architectures and recognition tasks. Triplet loss, without the large batch requirement, boosts standard architectures' performance. With minimal hyper-parameter tuning and a *single* fully connected layer on top of pretrained standard architectures, we promote generality to novel domains. Promising results are achieved on an imbalanced dataset. We incur a minimal computational overhead during training, but raise classification model efficiency and interpretability. Our architectural extension enables both retrieval and classification tasks during inference.

Chapter 6: Conclusion and Future Directions

We have proposed a set of tools to promote retrieval (feature embedding) networks. Our goal is to boost feature embedding networks' (1) performance, (2) stability, and (3) interpretability. To boost *performance* on small datasets, we re-train a network for multiple generations. We leverage the knowledge of a parent-network to boost the performance of its descendants. To boost feature embedding *stability* and avoid model collapse, we propose SVMax – a feature embedding regularizer. SVMax quantifies the uniformity of a feature embedding through its singular values. To boost retrieval network *interpretability*, we formulate attention visualization as a constrained optimization problem. We leverage the L2-Norm constraint to visualize attention in both classification and retrieval networks.

Furthermore, we propose a two-head architecture as a compromise between classification and retrieval networks. The two-head architecture is trained with both the cross-entropy loss and a ranking regularizer. This architecture converges fast during training – thanks to the classification head. In addition, it enables nearest neighbor search – thanks to the retrieval head.

Despite our effort to promote retrieval networks, we acknowledge that these networks still face multiple challenges. For instance, intense hyper-parameter tuning

is required to achieve state-of-the-art (SOTA) performance on different datasets and architectures. Another important challenge, for retrieval networks, is representation disentanglement. SOTA metric learning methods learn a feature embedding (*e.g.*, R^{128}) where it is impossible to embed different semantics in distinct subspaces (*e.g.*, break R^{128} into four R^{32} embeddings). This representation disentanglement problem has been tackled recently [141] in a supervised setting where different semantic (similarity notion) are annotated in the training dataset. However, this supervised setting is not sustainable.

Bibliography

[1] tf.gradients. URL `https://www.tensorflow.org/api_docs/python/tf/gradients`.

[2] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. Sanity checks for saliency maps. In *NIPS*, 2018.

[3] G. Alain and Y. Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[4] D. Bang and H. Shim. Mggan: Solving mode collapse using manifold guided training. *arXiv preprint arXiv:1804.04391*, 2018.

[5] L. Beyer, O. J. Hénaff, A. Kolesnikov, X. Zhai, and A. v. d. Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

[6] A. Brown, W. Xie, V. Kalogeiton, and A. Zisserman. Smooth-ap: Smoothing the path towards large-scale image retrieval. In *ECCV*, 2020.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[8] M. Bucher, S. Herbin, and F. Jurie. Improving semantic embedding consistency by metric learning for zero-shot classiffication. In *ECCV*, 2016.

[9] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen. Deep quantization network for efficient image retrieval. In *AAAI*, 2016.

[10] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[11] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv preprint arXiv:2006.09882*, 2020.

[12] B. Chen and W. Deng. Energy confused adversarial metric learning for zero-shot image retrieval and clustering. In *AAAI*, 2019.

[13] B. Chen, W. Deng, and H. Shen. Virtual class enhanced discriminative embedding learning. In *NeurIPS*, 2018.

[14] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin. This looks like that: deep learning for interpretable image recognition. *NeurIPS*, 2018.

[15] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[16] W. Chen, X. Chen, J. Zhang, and K. Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. In *CVPR*, 2017.

[17] X. Chen and K. He. Exploring simple siamese representation learning. *arXiv preprint arXiv:2011.10566*, 2020.

[18] D. Cheng, Y. Gong, S. Zhou, J. Wang, and N. Zheng. Person re-identification by multi-channel parts-based cnn with improved triplet loss function. In *CVPR*, 2016.

[19] J. Choe and H. Shim. Attention-based dropout layer for weakly supervised object localization. In *CVPR*, 2019.

[20] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew. Deep learning with cots hpc systems. In *ICML*, 2013.

[21] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[23] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015.

[24] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[25] A. Dubey, O. Gupta, P. Guo, R. Raskar, R. Farrell, and N. Naik. Pairwise confusion for fine-grained visual classification. In *ECCV*, 2018.

[26] A. Dubey, O. Gupta, R. Raskar, and N. Naik. Maximum-entropy fine grained classification. In *NeurIPS*, 2018.

[27] S. Eghbali and L. Tahvildari. Deep spherical quantization for image search. In *CVPR*, 2019.

[28] B. Fernando, H. Bilen, E. Gavves, and S. Gould. Self-supervised video representation learning with odd-one-out networks. In *CVPR*, 2017.

[29] R. Fong, M. Patrick, and A. Vedaldi. Understanding deep networks via extremal perturbations and smooth masks. In *ICCV*, 2019.

[30] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *ICCV*, 2017.

[31] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[32] S. Friedland and L.-H. Lim. The computational complexity of duality. *SIAM Journal on Optimization*, 26(4):2378–2393, 2016.

[33] K.-i. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 1993.

[34] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. Born again neural networks. *arXiv preprint arXiv:1805.04770*, 2018.

[35] S. Girish, S. R. Maiya, K. Gupta, H. Chen, L. Davis, and A. Shrivastava. The lottery ticket hypothesis for object recognition. *arXiv preprint arXiv:2012.04643*, 2020.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

[37] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, 2018.

[38] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

[39] P. Guo and Q. Ye. On regularization for a convolutional kernel in neural networks. *arXiv preprint arXiv:1906.04866*, 2019.

[40] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

[41] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 2016.

[42] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[43] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*, 2015.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[46] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[47] X. He, Y. Zhou, Z. Zhou, S. Bai, and X. Bai. Triplet-center loss for multi-view 3d object retrieval. *arXiv preprint arXiv:1803.06189*, 2018.

[48] A. Hermans, L. Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[49] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[50] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[51] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, 2015.

[52] J. Hoffman, D. A. Roberts, and S. Yaida. Robust learning with jacobian regularization. *arXiv preprint arXiv:1908.02729*, 2019.

[53] R. A. Horn and C. R. Johnson. Topics in matrix analysis cambridge university press. *Cambridge, UK*, 1991.

[54] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[55] S. Hu. Relations of the nuclear norm of a tensor and its matrix flattenings. *Linear Algebra and its Applications*, 478:188–199, 2015.

[56] C. Huang, Y. Li, C. Change Loy, and X. Tang. Learning deep representation for imbalanced classification. In *CVPR*, 2016.

[57] C. Huang, C. C. Loy, and X. Tang. Local similarity-aware deep feature embedding. In *NIPS*, 2016.

[58] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[59] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[60] C. Ionescu, O. Vantzos, and C. Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015.

[61] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum. Mining on manifolds: Metric learning without labels. In *CVPR*, 2018.

[62] V. Kazemi and A. Elqursh. Show, ask, attend, and answer: A strong baseline for visual question answering. *arXiv preprint arXiv:1704.03162*, 2017.

[63] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *CVPRW*, 2011.

[64] S. Kim, M. Seo, I. Laptev, M. Cho, and S. Kwak. Deep metric learning beyond binary supervision. In *CVPR*, 2019.

[65] S. Kim, D. Kim, M. Cho, and S. Kwak. Proxy anchor loss for deep metric learning. In *CVPR*, 2020.

[66] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[67] M. Kliegl, S. Goyal, K. Zhao, K. Srinet, and M. Shoeybi. Trace norm regularization and faster inference for embedded speech recognition rnns. *arXiv preprint arXiv:1710.09026*, 2017.

[68] X. Kong, J. Li, and X. Wang. New estimations on the upper bounds for the nuclear norm of a tensor. *Journal of inequalities and applications*, 2018(1): 282, 2018.

[69] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, 2013.

[70] J. Krause, H. Jin, J. Yang, and L. Fei-Fei. Fine-grained recognition without part annotations. In *CVPR*, 2015.

[71] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and L. Fei-Fei. The unreasonable effectiveness of noisy data for fine-grained recognition. In *ECCV*, 2016.

[72] B. Kumar, G. Carneiro, I. Reid, et al. Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5385–5394, 2016.

[73] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[74] J. Li, P. Lo, A. Taha, H. Wu, and T. Zhao. Segmentation of renal structures for image-guided surgery. In *MICCAI*, 2018.

[75] Q. Li, Z. Sun, R. He, and T. Tan. Deep supervised discrete hashing. In *NIPS*, 2017.

[76] Y. Li, Y. Song, and J. Luo. Improving pairwise ranking for multi-label image classification. In *CVPR*, 2017.

[77] T.-Y. Lin and S. Maji. Improved bilinear pooling with cnns. *arXiv preprint arXiv:1707.06772*, 2017.

[78] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear cnn models for fine-grained visual recognition. In *ICCV*, 2015.

[79] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[80] H. Liu, Y. Tian, Y. Yang, L. Pang, and T. Huang. Deep relative distance learning: Tell the difference between similar vehicles. In *CVPR*, 2016.

[81] F. Long, T. Yao, Z. Qiu, X. Tian, J. Luo, and T. Mei. Gaussian temporal awareness networks for action localization. In *CVPR*, 2019.

[82] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[83] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[84] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, 2018.

[85] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[86] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[87] Q. Mao, H.-Y. Lee, H.-Y. Tseng, S. Ma, and M.-H. Yang. Mode seeking generative adversarial networks for diverse image synthesis. In *CVPR*, 2019.

[88] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In *ICLR*, 2017.

[89] A. Morcos, H. Yu, M. Paganini, and Y. Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *NeurIPS*, 2019.

[90] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No fuss distance metric learning using proxies. In *ICCV*, 2017.

[91] R. Müller, S. Kornblith, and G. E. Hinton. When does label smoothing help? In *NeurIPS*, 2019.

[92] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *ICVGIP*, 2008.

[93] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, 2016.

[94] M. Noroozi, H. Pirsiavash, and P. Favaro. Representation learning by learning to count. In *ICCV*, 2017.

[95] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016.

[96] H. Oh Song, S. Jegelka, V. Rathod, and K. Murphy. Deep metric learning via facility location. In *CVPR*, 2017.

[97] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[98] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

[99] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[100] V. Petsiuk, A. Das, and K. Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

[101] A. Piergiovanni and M. S. Ryoo. Learning latent super-events to detect multiple activities in videos. In *CVPR*, 2018.

[102] A. Prakash, J. Storer, D. Florencio, and C. Zhang. Repr: Improved training of convolutional filters. In *CVPR*, 2019.

[103] S. Qiao, Z. Lin, J. Zhang, and A. L. Yuille. Neural rejuvenation: Improving deep network training by enhancing computational resource utilization. In *CVPR*, 2019.

[104] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, 2009.

[105] V. Ramanishka, Y.-T. Chen, T. Misu, and K. Saenko. Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning. In *CVPR*, 2018.

[106] V. Ramanishka, Y.-T. Chen, T. Misu, and K. Saenko. Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning. In *CVPR*, 2018.

[107] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? In *CVPR*, 2020.

[108] J. Revaud, J. Almazán, R. S. Rezende, and C. R. d. Souza. Learning with average precision: Training image retrieval with a listwise loss. In *ICCV*, 2019.

[109] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *PMLR*, 2014.

[110] O. Rippel, M. Paluri, P. Dollar, and L. Bourdev. Metric learning with adaptive density discrimination. *arXiv preprint arXiv:1511.05939*, 2015.

[111] E. Ristani and C. Tomasi. Features for multi-target multi-camera tracking and re-identification. *arXiv preprint arXiv:1803.10859*, 2018.

[112] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[113] A. Sablayrolles, M. Douze, C. Schmid, and H. Jégou. Spreading vectors for similarity search. *arXiv preprint arXiv:1806.03198*, 2018.

[114] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, 2016.

[115] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[116] S. Sankaranarayanan, A. Alavi, C. Castillo, and R. Chellappa. Triplet probabilistic embedding for face verification and clustering. *arXiv preprint arXiv:1604.05417*, 2016.

[117] A. Sanyal, V. Kanade, and P. H. S. Torr. Learning low-rank representations. *arXiv preprint arXiv:1804.07090*, 2019.

[118] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

[119] H. Sedghi, V. Gupta, and P. M. Long. The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*, 2018.

[120] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

[121] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[122] K. K. Singh and Y. J. Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *ICCV*, 2017.

[123] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[124] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *NIPS*, 2016.

[125] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In *NSIP*, 2017.

[126] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

[127] P. Stock and M. Cisse. Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases. In *ECCV*, 2018.

[128] C. Su, S. Zhang, J. Xing, W. Gao, and Q. Tian. Deep attributes driven multi-camera person re-identification. In *ECCV*, 2016.

[129] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017.

[130] Y. Sun, X. Wang, and X. Tang. Deeply learned face representations are sparse, selective, and robust. In *CVPR*, 2015.

[131] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[132] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[133] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

[134] A. Taha, M. Meshry, X. Yang, Y.-T. Chen, and L. Davis. Two stream self-supervised learning for action recognition. *DeepVision CVPRW*, 2018.

[135] A. Taha, Y.-T. Chen, T. Misu, A. Shrivastava, and L. Davis. Boosting standard classification architectures through a ranking regularizer. In *WACV*, 2020.

[136] Y. Tian, D. Krishnan, and P. Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

[137] R. Turkmen and H. Civciv. Some bounds for the singular values of matrices. *Applied Mathematical Sciences*, 1(49):2443–2449, 2007.

[138] G. Van Horn, S. Branson, R. Farrell, S. Haber, J. Barry, P. Ipeirotis, P. Perona, and S. Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *CVPR*, 2015.

[139] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[140] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NeurIPS*, 2016.

[141] A. Veit, S. Belongie, and T. Karaletsos. Conditional similarity networks. In *CVPR*, 2017.

[142] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[143] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin. Deep metric learning with angular loss. In *ICCV*, 2017.

[144] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015.

[145] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *ECCV*, 2016.

[146] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl. Sampling matters in deep embedding learning. In *ICCV*, 2017.

[147] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, 2018.

[148] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[149] T.-B. Xu and C.-L. Liu. Data-distortion guided self-distillation for deep neural networks. In *AAAI*, 2019.

[150] H. Xuan, A. Stylianou, and R. Pless. Improved embeddings with easy positive triplet mining. In *WACV*, 2020.

[151] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

[152] Y. Yuan, K. Yang, and C. Zhang. Hard-aware deeply cascaded embedding. In *ICCV*, 2017.

[153] S. Yun, J. Park, K. Lee, and J. Shin. Regularizing class-wise predictions via self-knowledge distillation. In *CVPR*, 2020.

[154] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[155] J. Zhang, Q. Lei, and I. S. Dhillon. Stabilizing gradients for deep neural networks via efficient svd parameterization. *arXiv preprint arXiv:1803.09327*, 2018.

[156] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *ICCV*, 2019.

[157] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *ECCV*, 2016.

[158] R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *CVPR*, 2017.

[159] X. Zhang, F. X. Yu, S. Kumar, and S.-F. Chang. Learning spread-out local feature descriptors. In *ICCV*, 2017.

[160] X. Zhang, Y. Wei, J. Feng, Y. Yang, and T. S. Huang. Adversarial complementary learning for weakly supervised object localization. In *CVPR*, 2018.

[161] X. Zhang, Y. Wei, G. Kang, Y. Yang, and T. Huang. Self-produced guidance for weakly-supervised object localization. In *ECCV*, 2018.

[162] X. Zhang, R. Zhao, Y. Qiao, X. Wang, and H. Li. Adacos: Adaptively scaling cosine logits for effectively learning deep face representations. In *CVPR*, 2019.

[163] Z. Zhang and V. Saligrama. Zero-shot learning via joint latent similarity embedding. In *CVPR*, 2016.

[164] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *CVPR*, 2016.

[165] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *NeurIPS*, 2019.

[166] Y. Zhu, M. Elhoseiny, B. Liu, X. Peng, and A. Elgammal. A generative adversarial approach for zero-shot learning from noisy texts. In *CVPR*, 2018.

[167] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.