

*LOS ANGELES COMPUTER VISION MEETUP*

# The State of the Art in Similarity Learning

Jacob Richeimer

GumGum

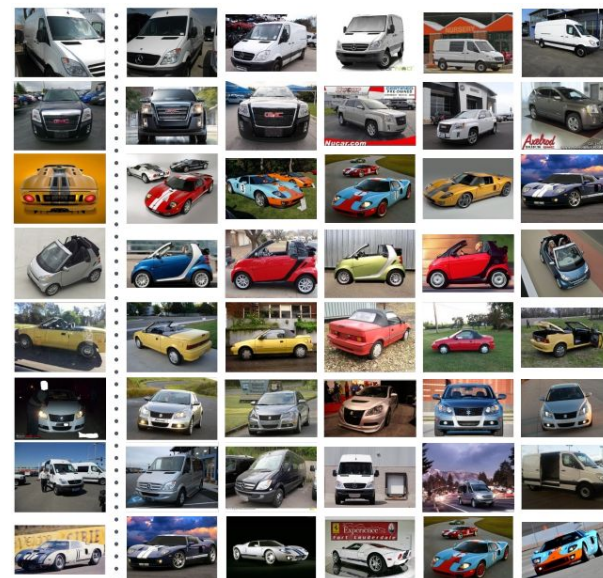
# The Problem

- Given a pair of images, determine the similarity between them.
- “Similarity” must be defined by the dataset.
- In Face Recognition/Verification, “similarity” means that the two face images are of the same person.



# Applications: Retrieval

Given an image, find the image(s) most similar to it in a database.

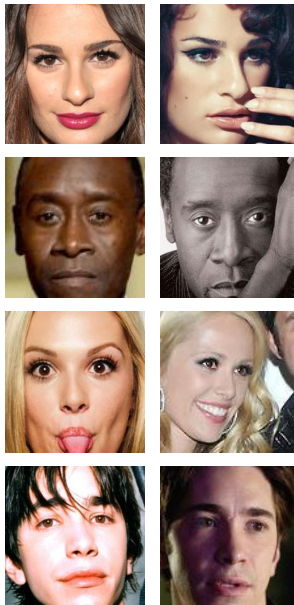


# Applications: Verification

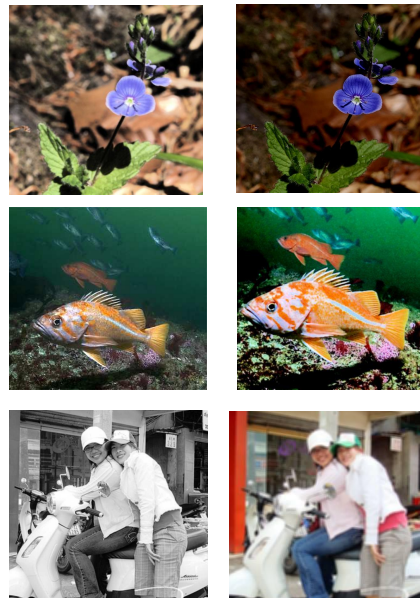
Given an image, determine if it belongs to the same class as a template or reference image.



person re-identification



face verification



duplicate "lookup"

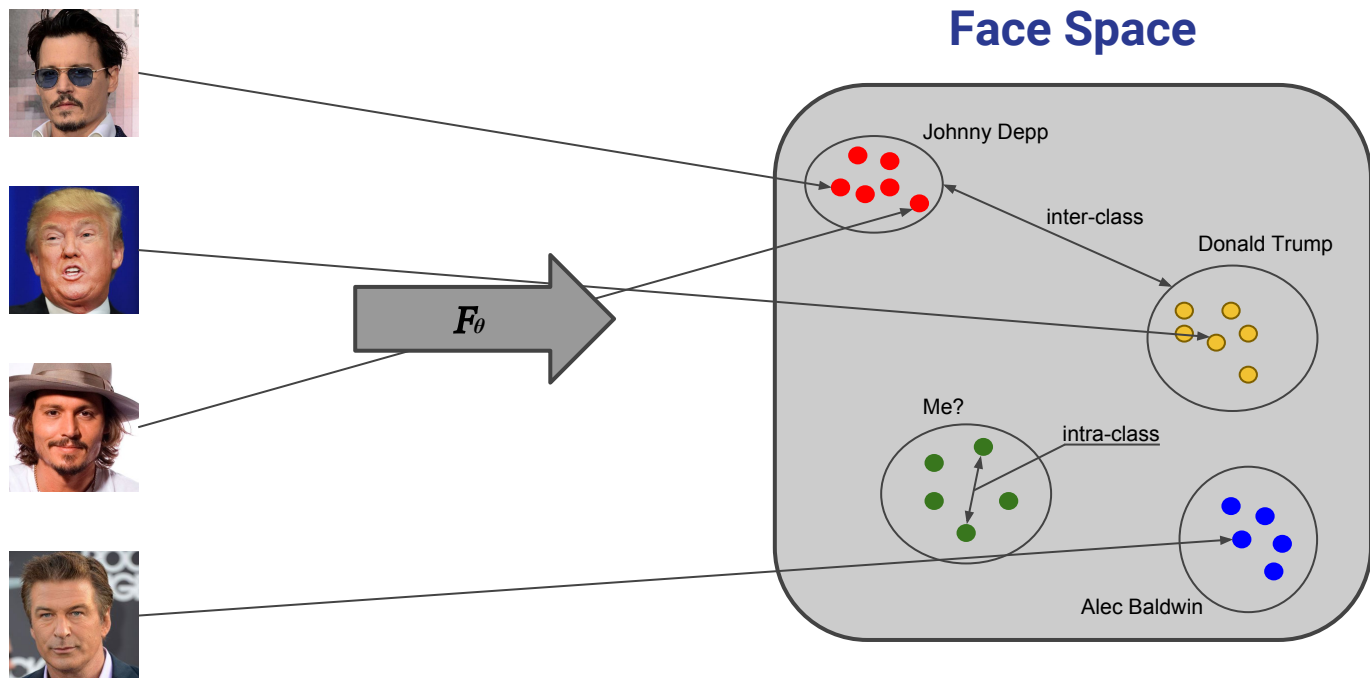
# The Training Data

We consider only data for which we have pairwise binary labels of similarity (similar/dissimilar, same-person/not-same-person)



Typically, we have a classification-style dataset, and label pairs-wise similarity based on the class labels.

# Fundamental Idea



# The Function

- Neural networks have become unmatched at feature learning for images.  
This motivates us to represent  $F_\theta$  as a neural network.
- How do we train?
  - What is the objective function (the loss) that we are trying to minimize?
  - How do we represent the image samples and labels?



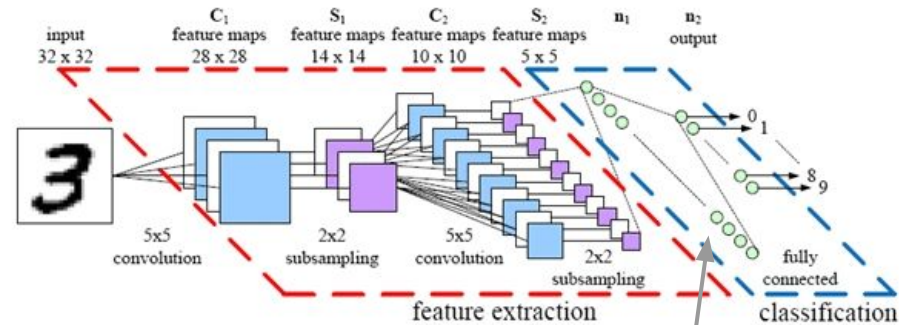


# Let's Review Deep Learning for Classification



# Classification

- A network for classification can be seen as consisting of two parts:
  - The *feature extractor*
  - The *classifier*
- The feature extractor takes the image as input and returns a (typically) one-dimensional vector as output. It consists of convolutional layers and often one or more fully connected layers.
- The classifier is a fully connected layer which outputs a vector with dimensionality equal to the number of classes. It is followed by a “softmax” activation.



Output of feature extractor

# The Softmax function

- The softmax function is, as its name implies, a “smoothened” version of the argmax function across a vector of values.

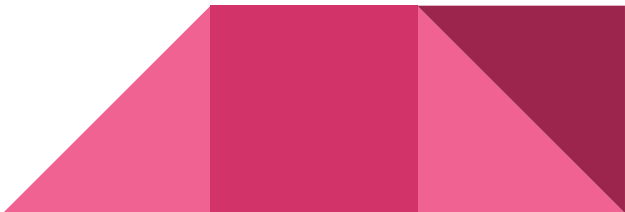
$$\text{Argmax} (v)_i = \begin{cases} 1 & \text{if } v_i > v_j \quad \forall j \neq i \\ 0 & \text{otherwise} \end{cases}$$

- The softmax allows a network to predict a probability distribution across the number of classes.

$$\text{Softmax} (v)_i = \frac{\exp(v_i)}{\sum_k \exp(v_k)}$$

exponentiation

normalization



# Cross-Entropy

When we train a classification model as described, the objective is to minimize the *cross-entropy* between the label and the predicted output.

$$\text{Cross-entropy} (y, y^*) = - \sum_k [y_k^* \times \log(y_k)]$$

Example:

$$- \log( [0.01, 0.21, 0.74, 0.04] ) \times [0, 0, 1, 0]^T$$

prediction

label





# Two Approaches to Solving Our Problem

# Two Approaches

1. Train a regular classifier (perhaps with extra restrictions), then use the feature extractor component as our mapping  $F_\theta$ .
2. Replace the classification layer with another layer designed to directly minimize intra-class distance and maximize inter-class distance between the feature vectors.



# Approach #1

Taking Advantage of Classifiers

---

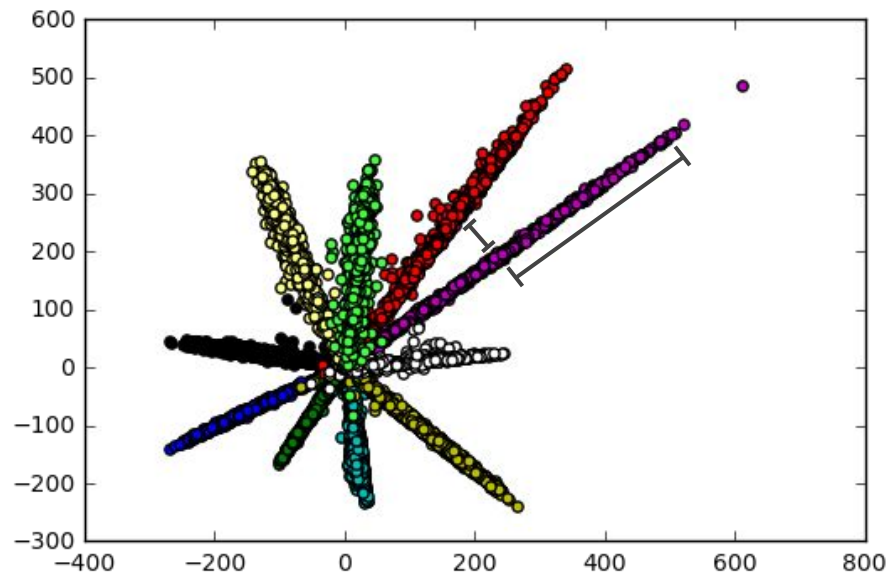
# Vanilla Classification

What happens when we train a classifier with a regular softmax layer?

Since the softmax activation normalizes its output, it is invariant to scaling of the input (the feature vector).

Therefore, Euclidean distance as a metric of similarity on the resulting feature space is a bad idea.

Good idea: Use cosine similarity (equivalently: Euclidean distance on normalized features) as a metric.



MNIST feature extraction (d=2) after classification training

# Beware of Bias

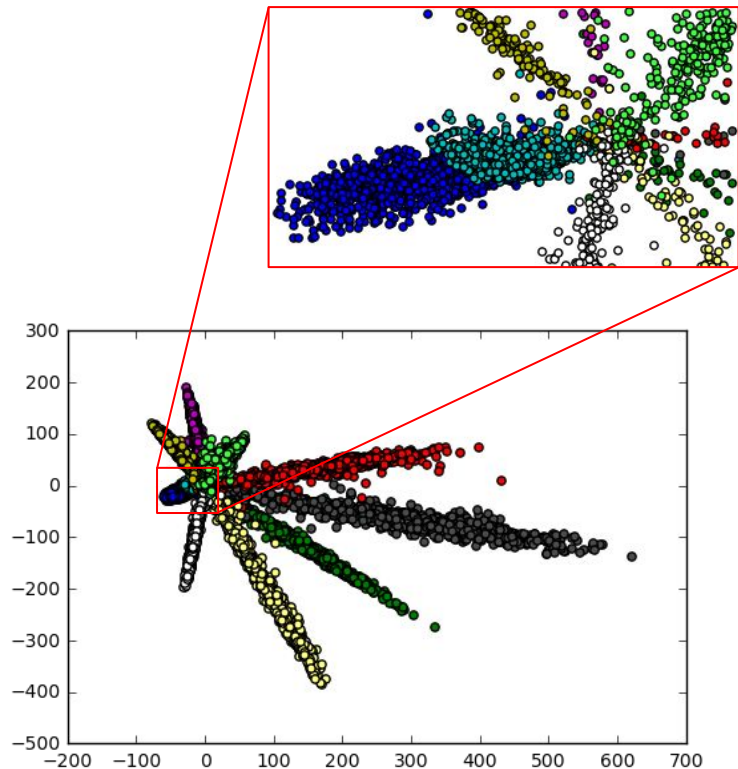
This new metric only makes sense if the softmax layer used in training includes no bias term.

We want this:

$$\text{pred} = \text{Softmax}(W * f)$$

Not this:

$$\text{pred} = \text{Softmax}(W * f + b)$$



MNIST feature extraction (d=2) after classification training with bias. [Notice the blue clusters.]



# How Can We Improve this Approach?

This cosine-similarity on top of a trained classifier approach really just takes advantage of a convenience in the structure of the feature space that a classifier learns via the softmax layer.

But perhaps we can make some restrictions on the structure of this space by altering or adding to the classification loss.

We will discuss two methods of doing this.



# Adding a Cluster Loss

- In addition to the regular classification loss, we can add an extra loss on the intermediate feature vector, to minimize its distance from its class center.
- This requires computing approximate class centers between each batch. This is done by updating the class centers at each iteration by adding (a proportion of) the average difference between the feature vectors of a class and its previous center.

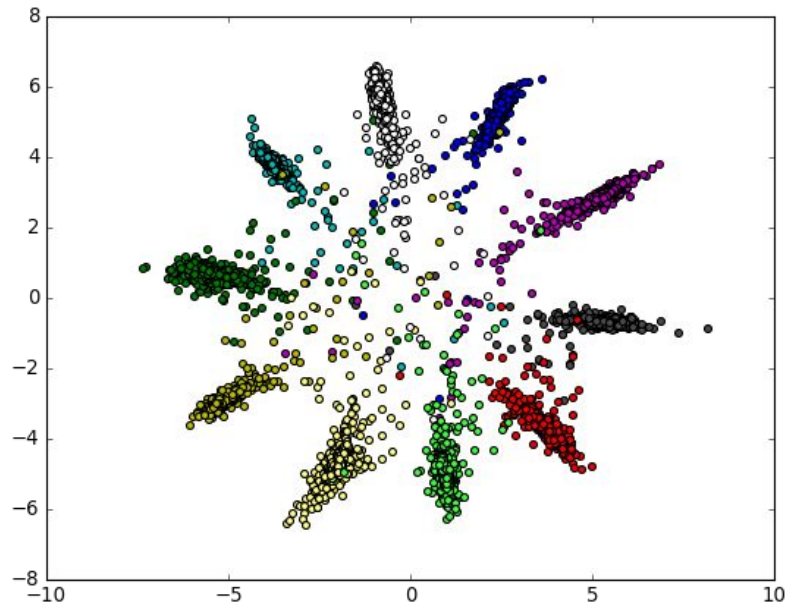
# Adding a Cluster Loss

$$\text{Loss}(f, y^*) = \text{cross-entropy}(y^*, \text{softmax}(W * f)) \\ + \lambda [0.5 * \sum \|f_i - c_{\text{class}(i)}\|^2]$$

The class centers  $\{c_i\}$  are initialized before the learning process and then are updated at each iteration:

$$\Delta c_k = \frac{\sum_i \delta(y_i = k) (c_k - f_i)}{1 + \sum_i \delta(y_i = k)}$$

$$c_{k+1} \leftarrow c_k + \alpha \Delta c_k$$

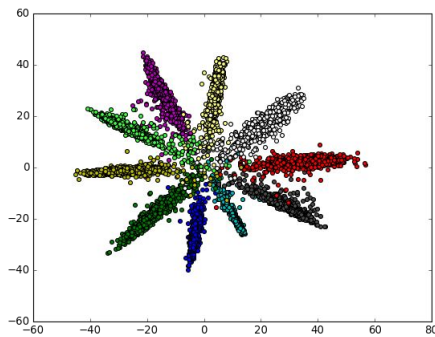


$\lambda=0.1, \alpha=0.5$

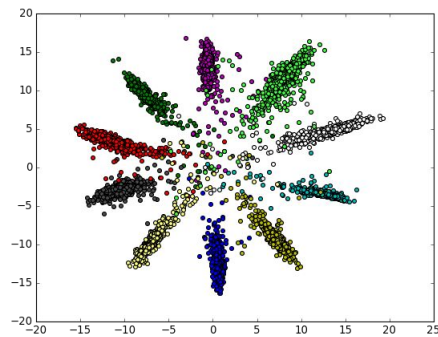
MNIST feature extraction (d=2) with classification and cluster loss.

# Adding a Cluster Loss

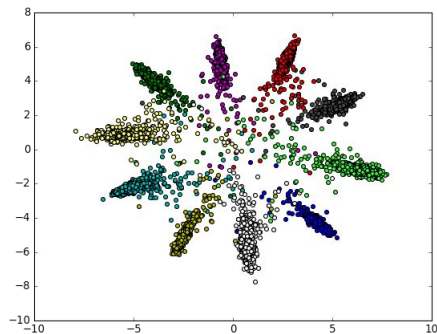
The parameter  $\lambda$  controls how much the cluster distance loss is considered in relation to the classification loss.



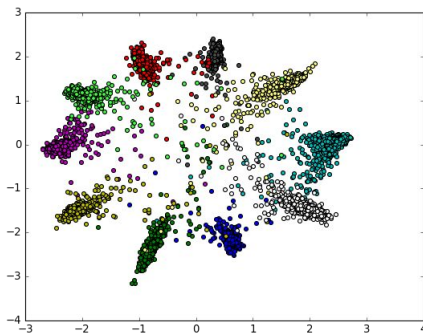
$\lambda=0.0001$



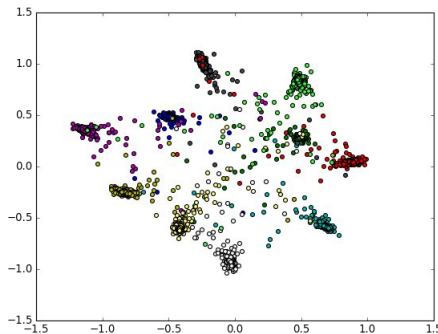
$\lambda=0.001$



$\lambda=0.01$



$\lambda=0.1$



$\lambda=1$

# Softmax with Cosine Distance

- The downside of the cluster loss is the extra step of updating the cluster centers between each iteration, and the added hyperparameters associated with that step.
- We can build in the idea of the cluster loss into the regular softmax layer (fully-connected with softmax activation) by considering the rows of the weight matrix in the layer as the cluster centers.



# Softmax with Cosine Distance

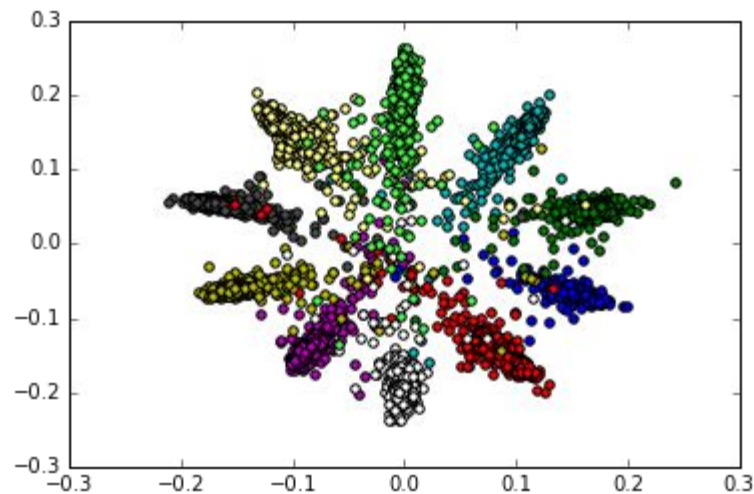
By normalizing both the feature vectors and the rows of the weight matrix  $W$  in the softmax layer, we can better optimize the distances between samples.

So instead of this:  $\text{pred} = \text{Softmax}(W * f)$

We'll have:  $\text{pred} = \text{Softmax}(s * \tilde{W} * \tilde{f})$

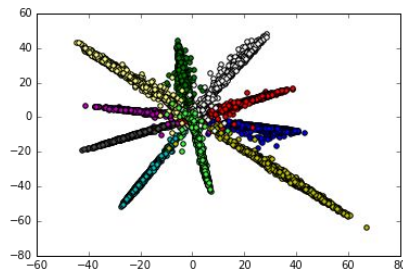
where  $\tilde{f}$  is normalized  $f$  and  $\tilde{W}$  is  $W$  with its rows unit-normalized.

The scale factor  $s$  is to “enlarge” the embedded hypersphere depending on the number of classes in the dataset.

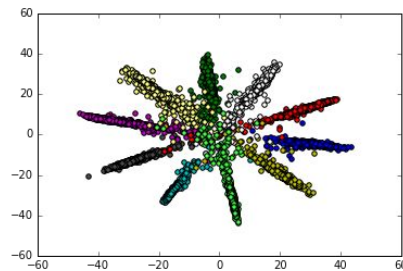


# Batch Size Matters

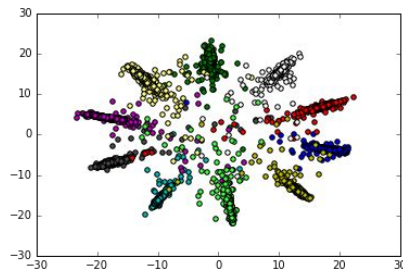
The “softmax-with-cosine” loss minimizes the angular distance between class samples and the corresponding rows of the weight matrix. Since the weight matrix is updated after each batch, the batch size directly affects the results.



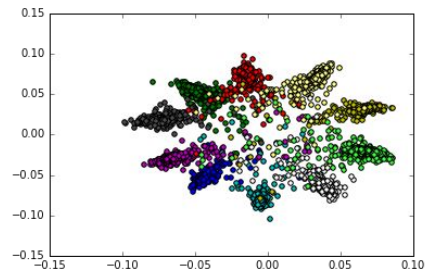
batch\_size = 16



batch\_size = 64



batch\_size = 256



batch\_size = 1024

# Feature Re-weighting

- When using classification training for similarity learning, it could be that some dimensions of the latent space matter more for the similarity metric than others.
- During classification training, the features can be “re-weighted” (element-wise multiplied) as part of the softmax layer. But the softmax layer is discarded after training.
- This might be rectified by replacing the typical softmax layer (fully-connected→softmax) into a *feature re-weighting layer* and then a softmax layer (multiplication→fully-connected →softmax).

$$\text{pred} = \text{softmax}(W^*(w_{frw} \odot f))$$



# Approach #2

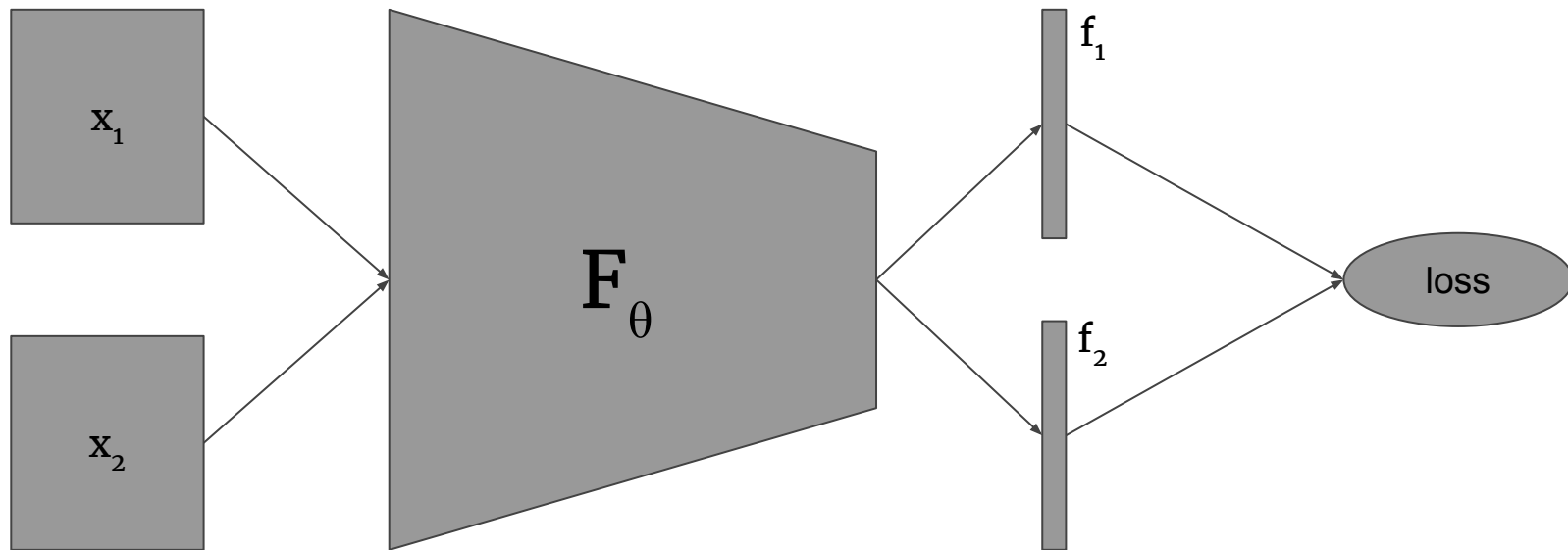
A new path...

---

# Siamese Networks

- Siamese Networks were originally proposed (in 1994) for signature verification.
- The idea is to have one feature extractor, and to have a sample be a pair of images rather than a single image. The feature extractor will output two feature vectors, which can then be compared by the loss layer.

# Siamese Networks

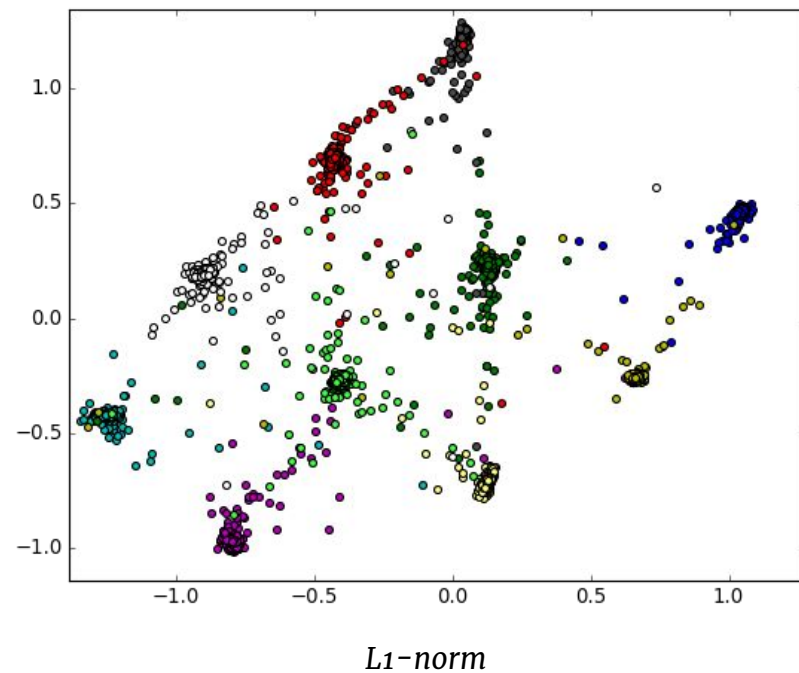
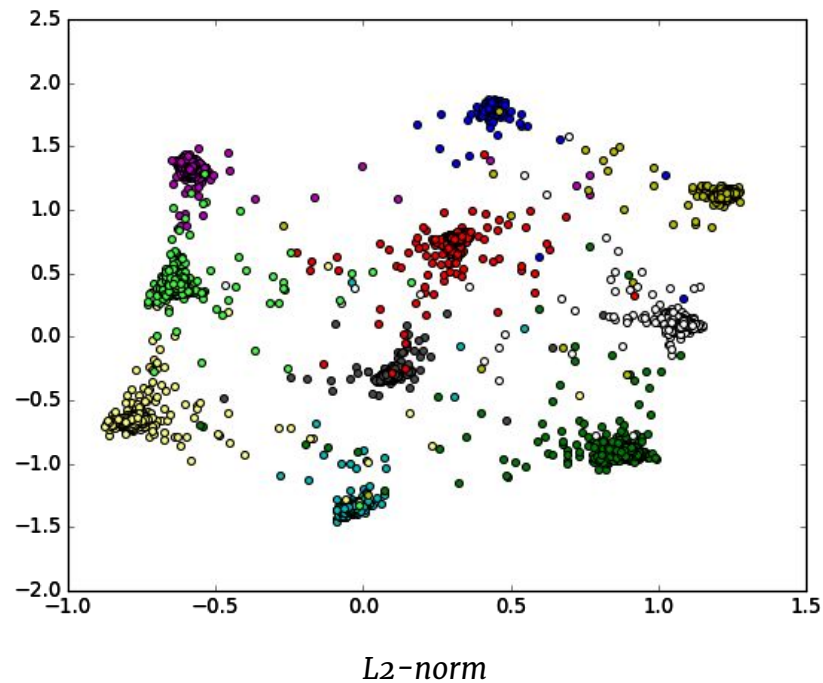


# Contrastive Loss

- Our pair-samples that will be fed into the network will have binary labels: 1 if they are similar, i.e. belong to the same class, and 0 if they are dissimilar.
- The *contrastive loss* is a way of minimizing the distance between the feature vectors of similar samples and maximizing the distance between feature vectors of dissimilar samples.

$$\text{contrastive}(\mathbf{f}_1, \mathbf{f}_2) = \begin{cases} \|\mathbf{f}_1 - \mathbf{f}_2\| & \text{if similar} \\ \max(0, m - \|\mathbf{f}_1 - \mathbf{f}_2\|) & \text{if dissimilar} \end{cases}$$

# Contrastive Loss



# Triplet Networks

- The intuition: the goal is not necessarily to maximize inter-class distances and minimize the intra-class distances. The goal is really to make sure the inter-class distances are **greater than** the intra-class distances.
- This motivates the *triplet network*. The structure is essentially the same as the Siamese network, except that the samples on which the loss is computed are composed of “triplets” of images rather than pairs.



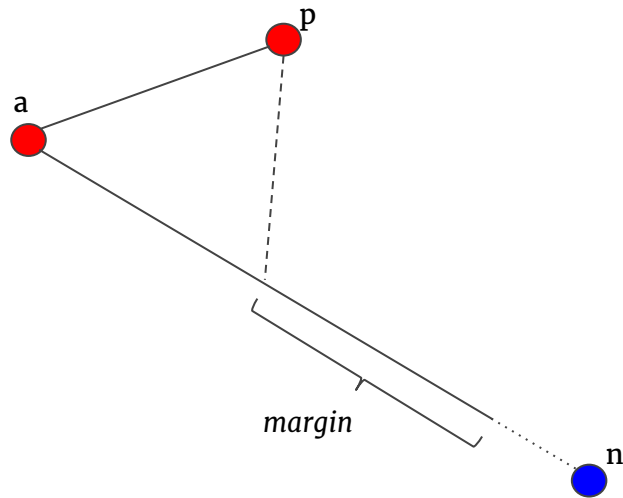
# Triplet Loss

During training, the loss is computed per triplet. One sample acts as the *anchor* (or *reference*), another a positive sample which is “similar” to the anchor, and the third is a negative sample, dissimilar to the anchor.

$$\text{triplet-loss}(\mathbf{f}_a, \mathbf{f}_p, \mathbf{f}_n) = \left[ \|\mathbf{f}_a - \mathbf{f}_p\| - \|\mathbf{f}_a - \mathbf{f}_n\| + m \right]_+$$

Remember, our goal is:

$$\|\mathbf{f}_a - \mathbf{f}_p\| + m < \|\mathbf{f}_a - \mathbf{f}_n\|$$



example of feature vectors of a triplet

# Triplet Mining

- For a dataset of size  $N$ , the number of triplet combinations is  $O(N^3)$ .
- Most of the triplet combinations soon into training already satisfy:

$$\|f_a - f_p\| + m < \|f_a - f_n\|$$

which means that the loss on those triplets is zero and contributes nothing to the training.

- This problem has motivated many strategies of “hard triplet mining” - finding triplet samples that are “hard” (do not satisfy the above inequality) on which to train the model.





# Triplet Mining

For Google's "Facenet", the triplet sampling procedure was described as follows:

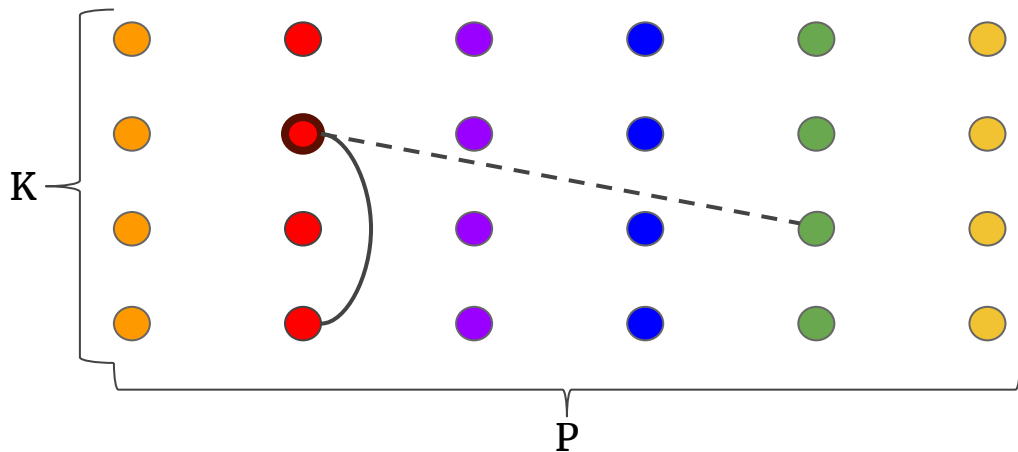
Take a large batch (`batch_size = 1800`) with at least 40 samples from each of  $k$  classes. Find all the anchor-positive pairs within those  $k$  classes. For each anchor-positive pair, compute the anchor-negative distance for all the available negative samples in the batch. Discard all the triplets except those which are *semi-hard* — meaning that the difference between the anchor-positive and anchor-negative distances is within the margin. Use only those triplets to compute the loss.

# Triplet Mining

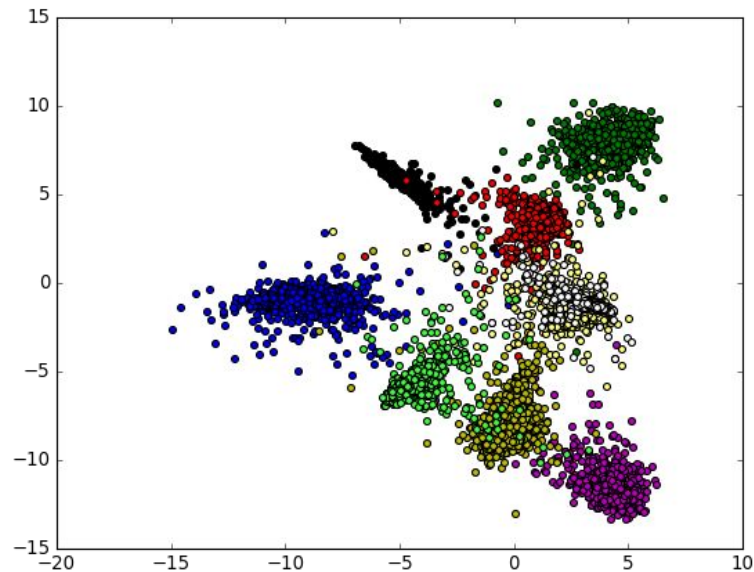
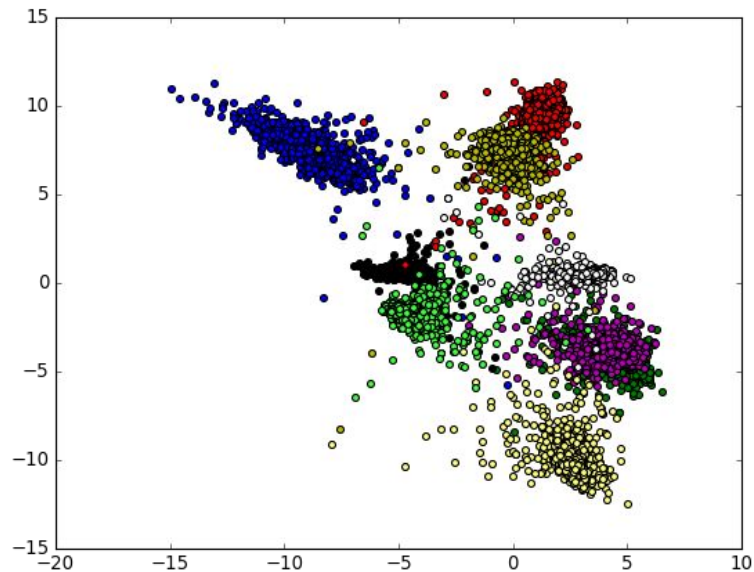
In a recent paper comparing several triplet mining techniques, the following *batch-hard* sampling was recommended:

In each batch,  $P$  classes are sampled randomly, and  $K$  samples randomly drawn from those classes.

For each sample in the batch, the farthest intra-class sample is selected as the positive and the closest inter-class sample is selected as the negative.

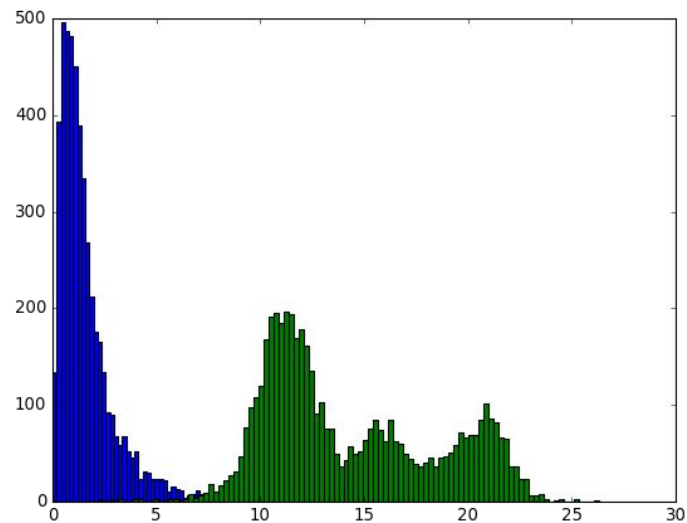
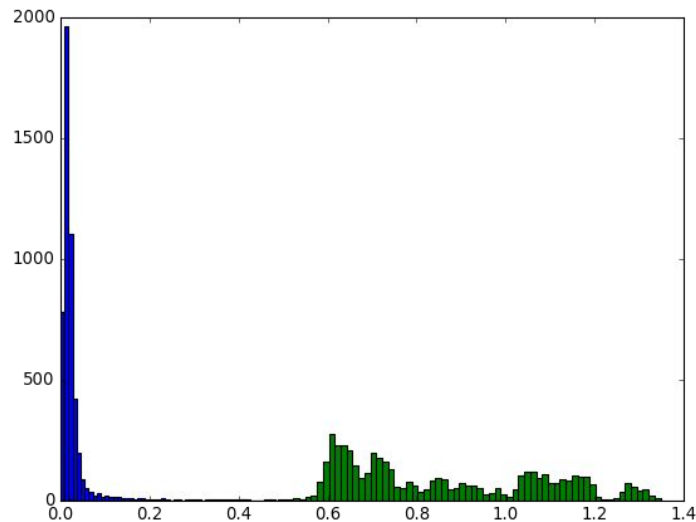


# Triplet Mining



3-dimensional latent space learned for MNIST with the triplet loss, pictured above from two orthogonal perspectives. Batch-hard triplet mining was used ( $P = 10$  ,  $K = 10$ ).

# Distance Distributions



Histograms of distances between randomly selected positive (blue) and negative (green) pairs in MNIST trained with contrastive loss (left) and triplet loss (right).

# Softening the Terms

Techniques to smoothen the operations applied in triplet learning are suggested by a few implementations on the following steps.

- Instead of taking the max of the positive distances per batch sample, take the *log-sum-exp* – the smooth upper bound of the max.

$$\text{log-sum-exp}(x) = \log(\sum_i e^x)$$

- Instead of taking the rectified-linear of the distance difference, take the *softplus*.

$$\text{softplus}(x) = \log(1 + e^x)$$



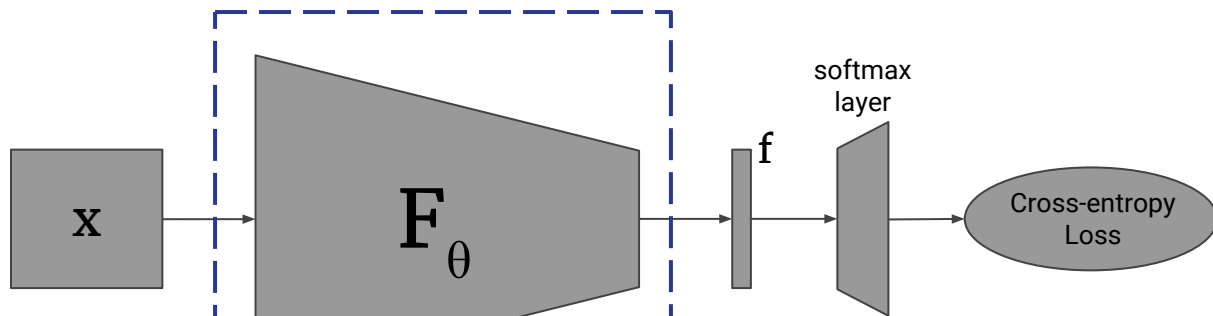
# Pre-training the Weights

Often, networks trained with the triplet loss have trouble converging. Sometimes, the weights collapse to zero. The severity of this problem depends a lot on the triplet mining technique.

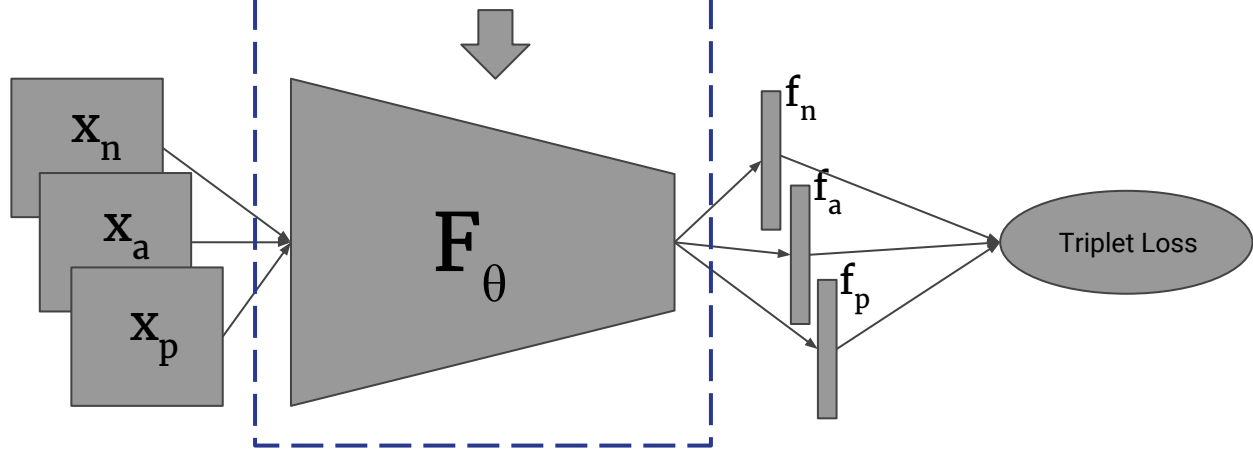
To solve this, it's common to pre-train the net with a classification loss, then discard the last layer (classifier) and replace with the triplet loss. This lets the weights start from a good initialization in which the feature space already tentatively clusters the classes.

# Pre-training the Weights

1



2



# Combining with Classification

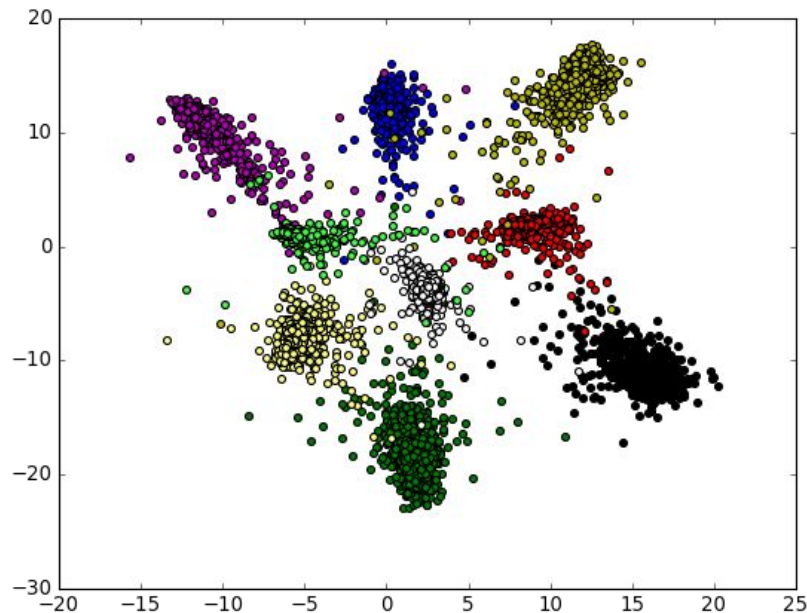
Instead of pre-training with a classification loss, we can also train with both the triplet and classification losses simultaneously.

We add an extra softmax layer after the feature extraction, which outputs the class prediction  $y$ .

$$y = \text{softmax}(W*f+b)$$

$$\text{loss}(f_a, f_p, f_n) = \left[ \|f_a - f_p\| - \|f_a - f_n\| + m \right] + \lambda \sum_{i=\{a,p,n\}} \text{x-entropy}(y_i, y_i^*)$$

cf. Geng et. al., “Deep Transfer Learning for Person Re-identification”



2D MNIST clusters trained with combined triplet and classification losses. ( $\lambda=2.3$ )

[Batch-hard triplet mining with  $P=5$ ,  $K=20$ ]



# Quadruplet Nets

- While triplet loss is the most popular objective for metric learning, some papers have attempted a quadruplet loss.
- The argument for its superiority: the positive pair distance and negative pair distance that are being compared per triplet are in reference to the same anchor. Ideally, we want the maximum intra-class distance to be smaller than the maximum inter-class distance regardless of the class.

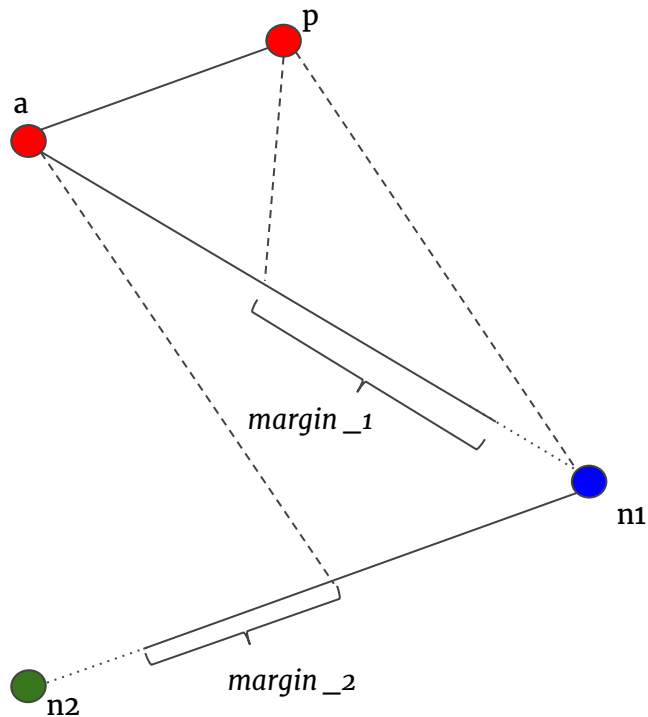


# Quadruplet Loss

The quadruplet loss adds a term to the triplet loss which enforces the familiar inequality, but with respect to an inter-class distance between samples of other classes.

$$\text{quad-loss}(\mathbf{f}_a, \mathbf{f}_p, \mathbf{f}_{n1}, \mathbf{f}_{n2}) = \overbrace{\left[ \|\mathbf{f}_a - \mathbf{f}_p\| - \|\mathbf{f}_a - \mathbf{f}_{n1}\| + m_1 \right]_+}^{\text{triplet loss term}} + \left[ \|\mathbf{f}_a - \mathbf{f}_p\| - \|\mathbf{f}_{n1} - \mathbf{f}_{n2}\| + m_2 \right]_+$$

The added difficulty of “hard-quadruplet” mining, and the extra hyperparameter makes this difficult to implement in practice.



# Histogram Loss

Another approach that has been proposed is to compute the histogram of distances between all positive pairs in a batch and all negative pairs. Given these two distributions, we can estimate the probability of a positive pair distance being less than a negative pair distance as:

$$\int_{-1}^1 p^-(x) \left( \int_{-1}^x p^+(y) dy \right) dx$$

This is proposed, in its discretized form (with number of bins as a hyperparameter), as the *histogram loss*.

# Datasets

*Open Test Set:* The test set will contain classes (identities) that were not seen in training. A large number of training classes is beneficial for the feature extractor to begin to generalize (rather than look for particular discriminating features for the training classes).

Dataset	Celebrity?	Identities	Size
LFW	Yes	5K	13K
FaceScrub	Yes	530	106K
YFD	Yes	1.5K	3.4K Videos
CelebFaces	Yes	10K	202K
UMDFaces	Yes	8.5K	367K
CASIA-WebFace	Yes	10K	500K
MS-Celeb-1M	Yes	100K	10M
VGG-Face	Yes	2.6K	2.5M
<b>Ours</b>	<b>No</b>	<b>672K</b>	<b>4.7M</b>
Facebook†	No	4K	4.4M
Google †	No	8M	200M+
Adience	No	2.2K	26K

*The size of some popular face recognition datasets.  
(From the MegaFace paper.)*



# MegaFace Challenge

The University of Washington has compiled a dataset of face identities as a benchmark of large-scale real-world face identification and verification

- **Identification:** Given  $M$  images of a person, insert one image into a “gallery” of up to one million *distractors*. Then test the probability that a random image from the  $M-1$  other images is closest to that image.
- **Verification:** Test accuracy (recall/precision) on billions of positive and negative face-image pairs.

Kemelmacher-Shlizerman et al., “The MegaFace Benchmark: 1 Million Faces for Recognition at Scale” (2016)

Nech et. al., “Level Playing Field for Million Scale Face Recognition” (2017)

<<http://megaface.cs.washington.edu/>>





# Thank You

Questions?