Metric Learning for Computer Vision

Advanced Computer Vision

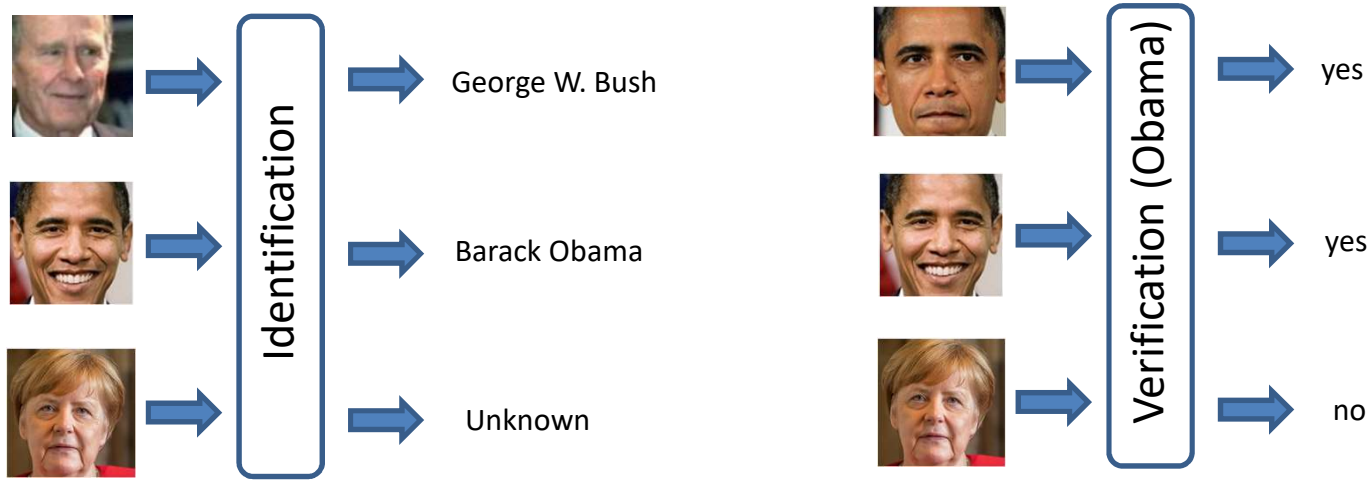
Niels Landwehr

Overview

- Introduction: Computer Vision
- Data, Models, Optimization
- Neural Networks and Automatic Differentiation
- Convolutional Architectures For Image Classification
- Visualization and Transfer Learning
- Metric Learning for Computer Vision

A Motivating Example: Face Recognition

- **Motivating example: face recognition**
  - Problem setting identification: map an input image to a set of known identities or „unknown“
  - Problem setting face verification: decide whether an input image shows a particular person



- Many applications: unlocking a mobile phone, tagging persons in photos, ...

Face Recognition: A Classification Problem?

- **Setting up face recognition as classification?**

- Identification setting:
  - Fixed set of identities $c_1,...,c_K$ could be the different classes?
  - But what about the „unknown“ class?
  - Would have to collect a lot of training data for this group of people
  - Would need to retrain the model every time a new identity is added
- Verification setting:
  - binary problem, train a classifier for a specific identity
  - would need a lot of training data for that person, unrealistic for most applications
  - have to train a separate model for each identity (expensive)

- **We need a different way of doing this...**

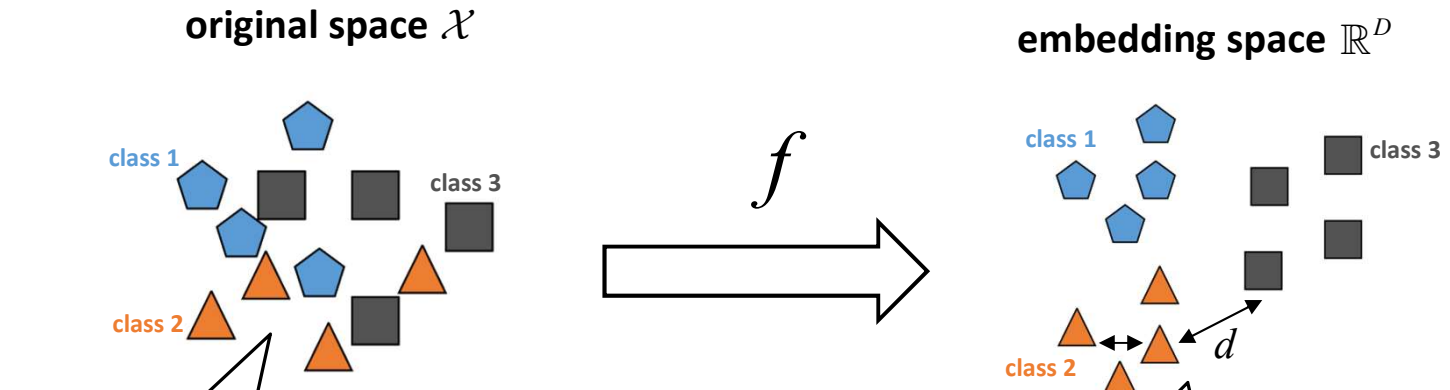
Problem Setting Metric Learning

- **Problem Setting: Metric Learning (not just for face recognition)**

- **Given:**
  - a set of instances $\mathbf{x}_1,...,\mathbf{x}_n \in \mathcal{X}$
  - a set of class labels $y_1,...,y_n \in \{c_1,...,c_k\}$, where we assume for simplicity that $\{c_1,...,c_k\} = \{1,...,k\}$
- **Find**:
  - a model $f : \mathcal{X} \to \mathbb{R}^D$ mapping instances to a relatively low-dimensional vector space (D is a hyperparameter of the model)
  - such that for instances $\mathbf{x}, \mathbf{x}'$ from the same class $d(f(\mathbf{x}), f(\mathbf{x}'))$ is small, and for instances $\mathbf{x}, \overline{\mathbf{x}}$ from different classes $d(f(\mathbf{x}), f(\overline{\mathbf{x}}))$ is large
  - Here, $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a distance measure (e.g. $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2$)
- Can also be formulated using similarities instead of distances

Visualization Metric Learning

- **Visualization:** metric learning means learning a mapping that clusters classes in the embedding space $\mathbb{R}^D$

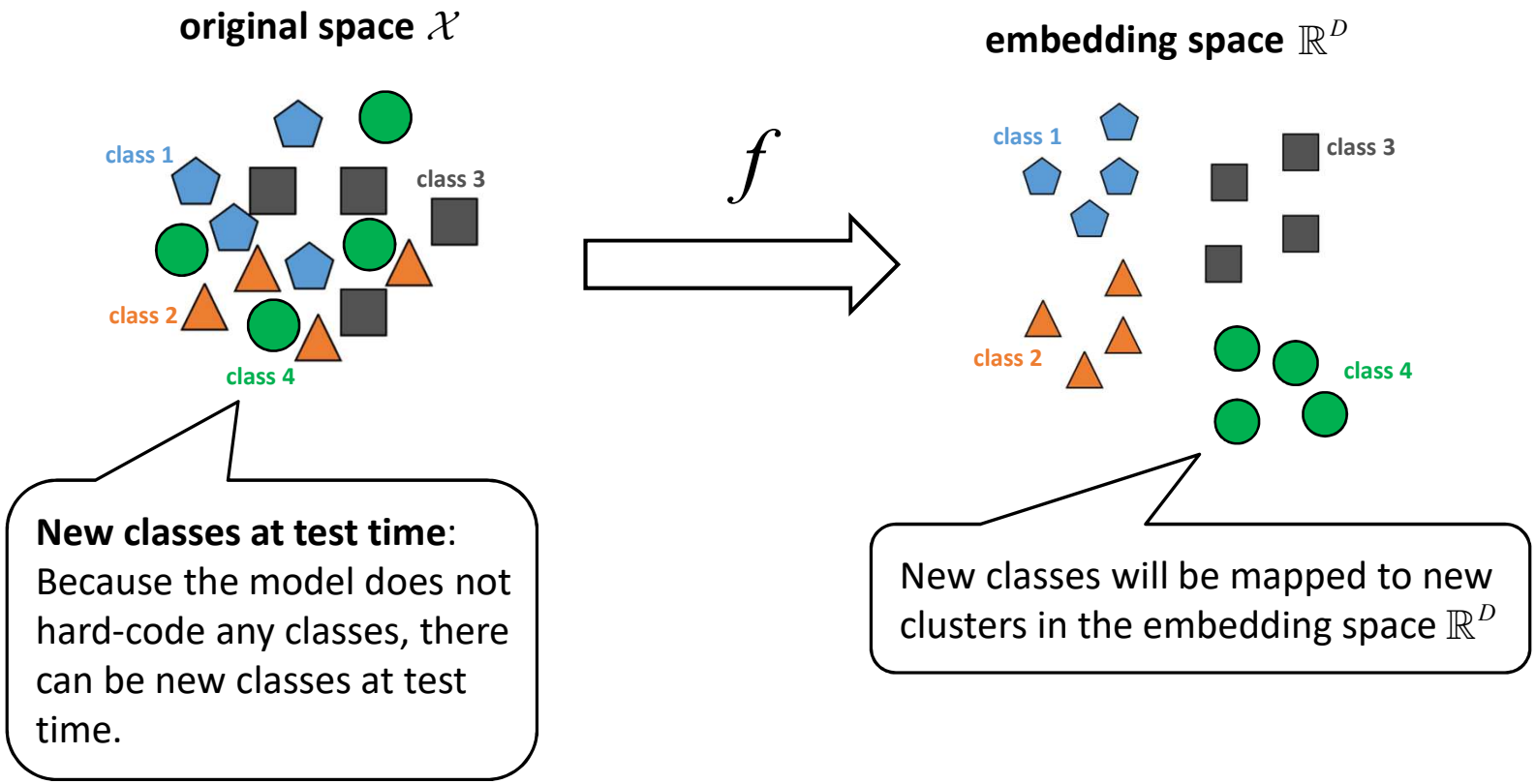


- Original space $\mathcal{X}$: classes are not well separated
- E.g. images of persons in face recognition: simple distance in pixel space not informative

- Trained model maps instances into embedding space $\mathbb{R}^D$
- In this space, classes cluster according to the distance or similarity measure
- Note that the model has no concept of classes, it only defines a mapping

Visualization Metric Learning

- **Visualization:** metric learning means learning a mapping that clusters classes in the embedding space $\mathbb{R}^D$

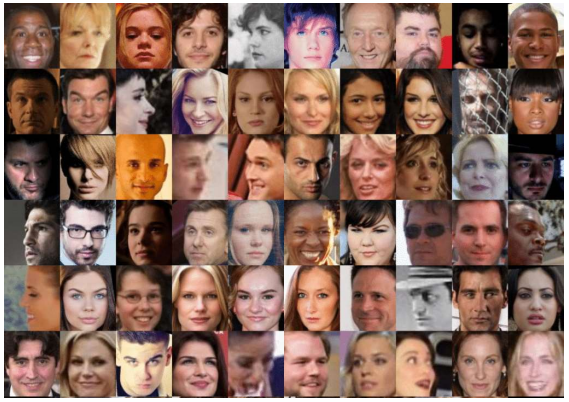


Metric Learning for Face Recognition

- **Advantage of using metric learning: example face recognition**
  - We no longer have to train a model on a particular person or group of persons, for which we would often have too little data
  - Instead, we train **one** general embedding model on any large face data set, and use that model to embed any new face image

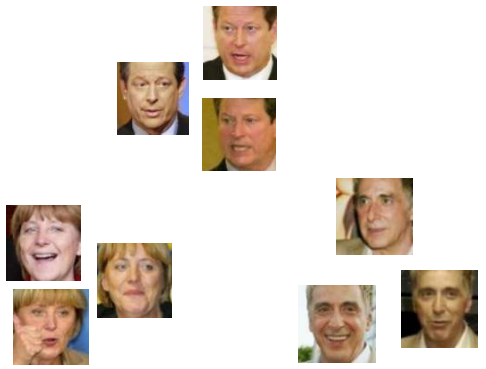
Training embedding model

original space $\mathcal{X}$



f

embedding space $\mathbb{R}^D$

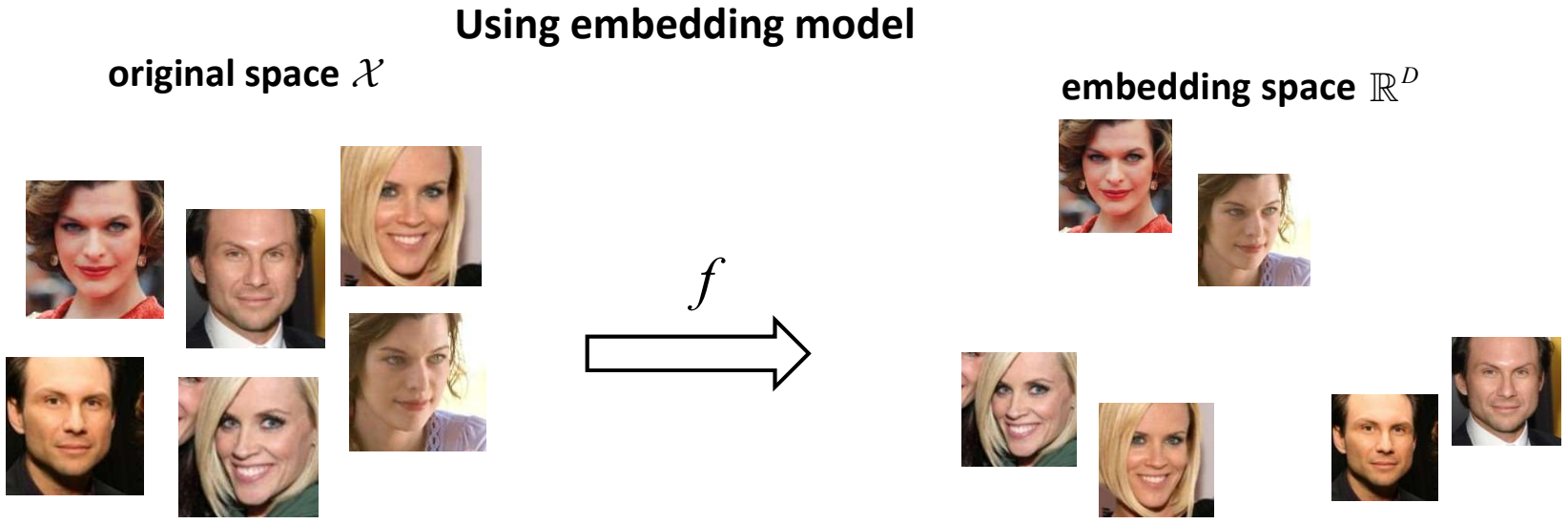


Large-scale data set: e.g. Microsoft Celeb (public, 10 million images) or proprietory data sets (Facebook, Google, ...) of hundreds of million images

Learned embedding space separates identities in training data set

Metric Learning for Face Recognition

- **Advantage of using metric learning: example face recognition**
  - We no longer have to train a model on a particular person or group of persons, for which we would often have too little data
  - Instead, we train **one** general embedding model on any large face data set, and use that model to embed any new face image



New identities (with litte data) at test time

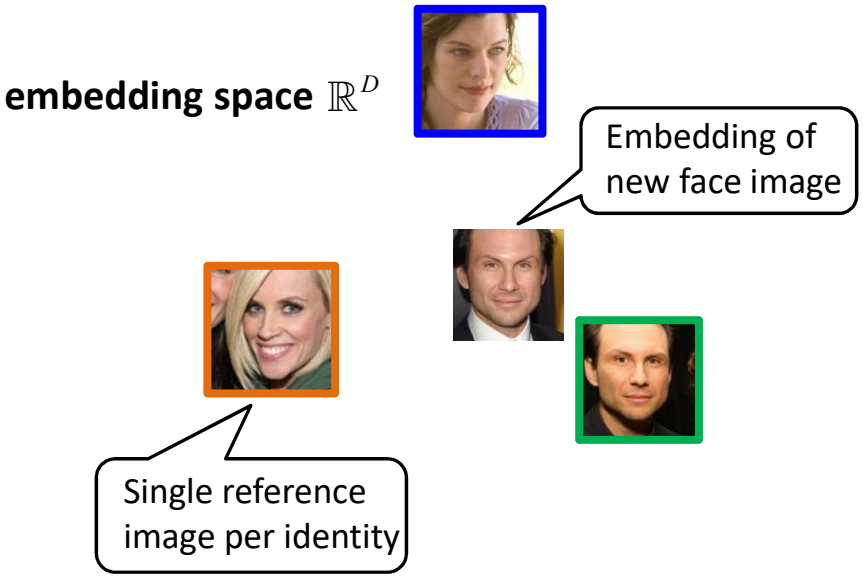
Learned embedding also separates new identities

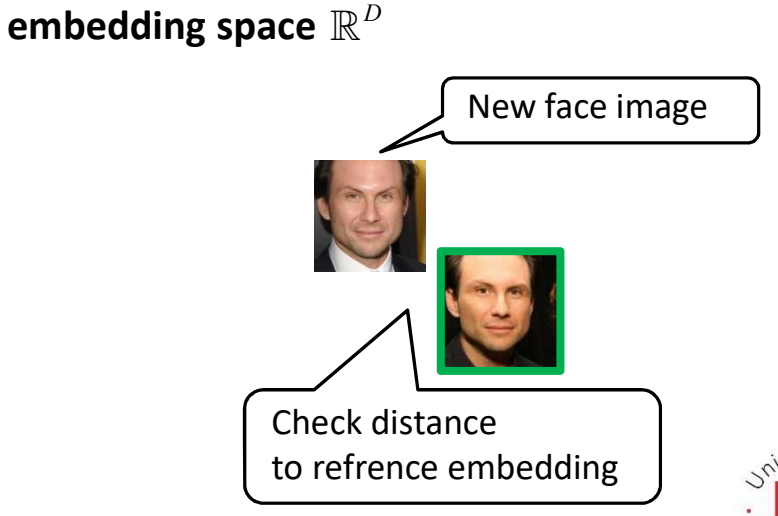
Metric Learning for Face Recognition

- Once we have a good embedding, face recognition becomes easy

Identification:
- Map embedding of new face image to closest reference image embedding
- If distance to all reference images is large, say „unknown“
- Can also have multiple reference images, using minimum or average distance

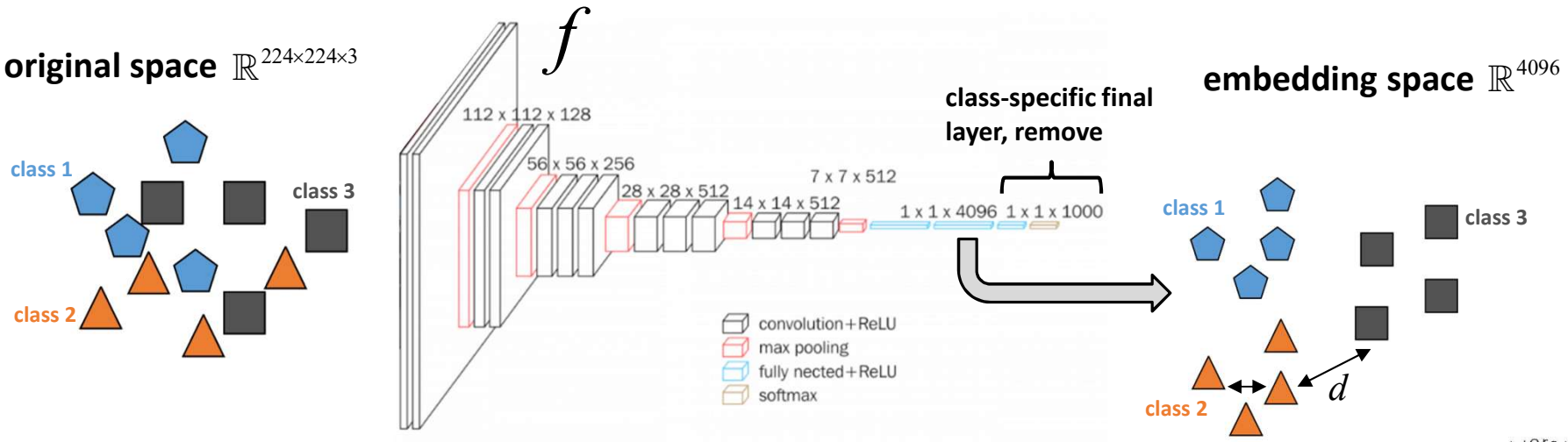
Verification:
- Compute distance of embedding of new face image to reference embedding
- If distance is below threshold, return „same“, else return „different“
- Can also have multiple reference images, using minimum or average distance





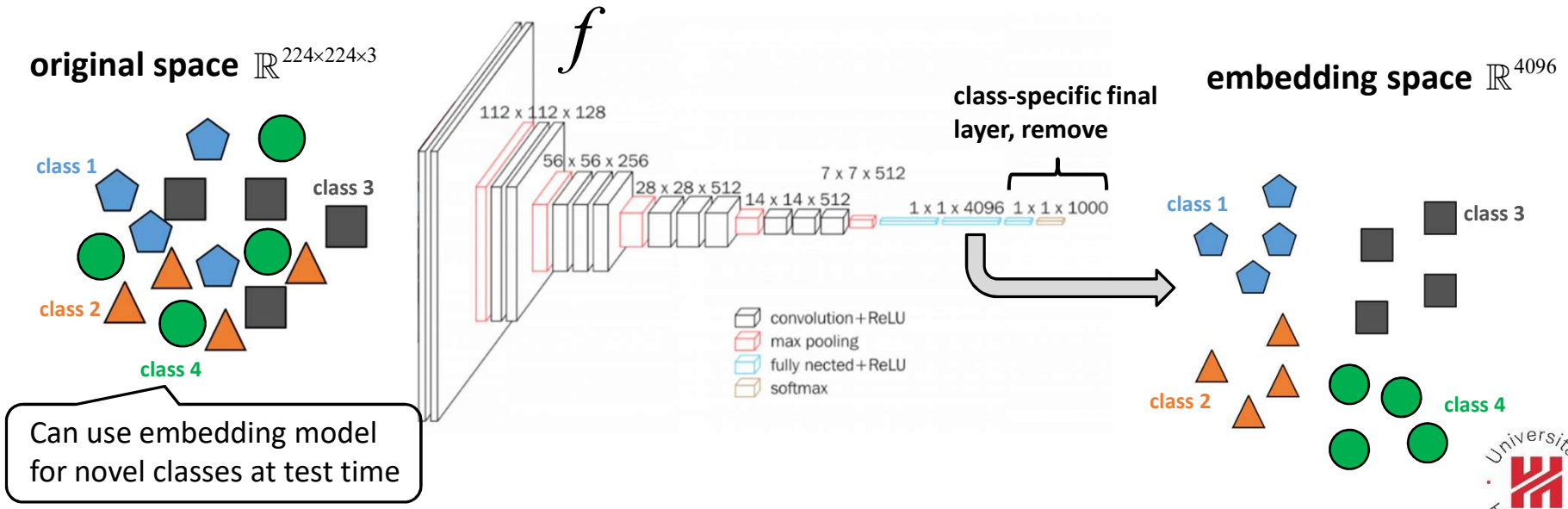
Metric Learning With Classification Model

- How do we train a metric learning model?
- **Approach 1: use embedding from standard classification model**
  - Use any standard classification model (e.g. VGG, ResNet, ...)
  - Set up training as multiclass classification with cross-entropy loss
  - After training, use the features at the last layer before the class-specific class score (and possibly softmax) layer as the embedding:



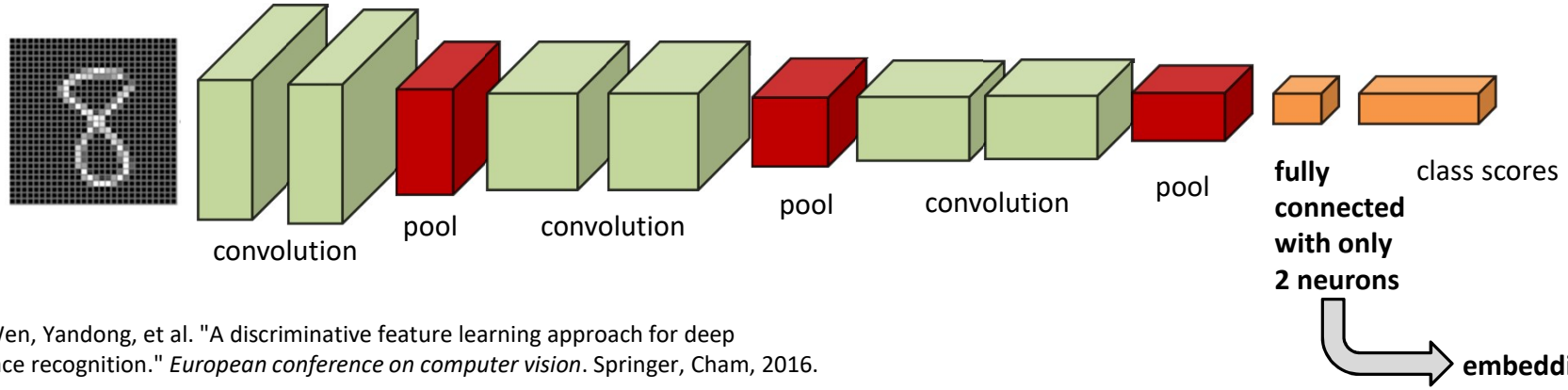
Metric Learning With Classification Model

- How do we train a metric learning model?
- **Approach 1: use embedding from standard classification model**
  - Use any standard classification model (e.g. VGG, ResNet, ...)
  - Set up training as multiclass classification with cross-entropy loss
  - After training, use the features at the last layer before the class-specific class score (and possibly softmax) layer as the embedding:



Why Does This Work?

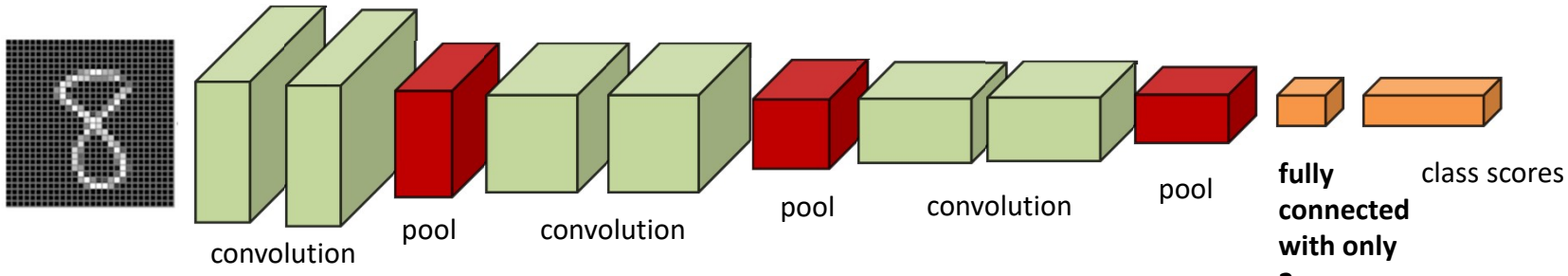
- **Why does Approach 1 work?**
- Idea:
  - the final layer of the classification CNN is simply a linear classifier
  - the feature representation learned in the layer before must already separate the classes quite well for the overall model to work
- **Toy example** [Wen et al., 2016]:
  - MNist digit classification data set (10-class problem)
  - Simple convolutional model similar to the LeNet architecture
  - To simplify visualization, final layer before class scores has only 2 neurons



Wen, Yandong, et al. "A discriminative feature learning approach for deep face recognition." *European conference on computer vision*. Springer, Cham, 2016.

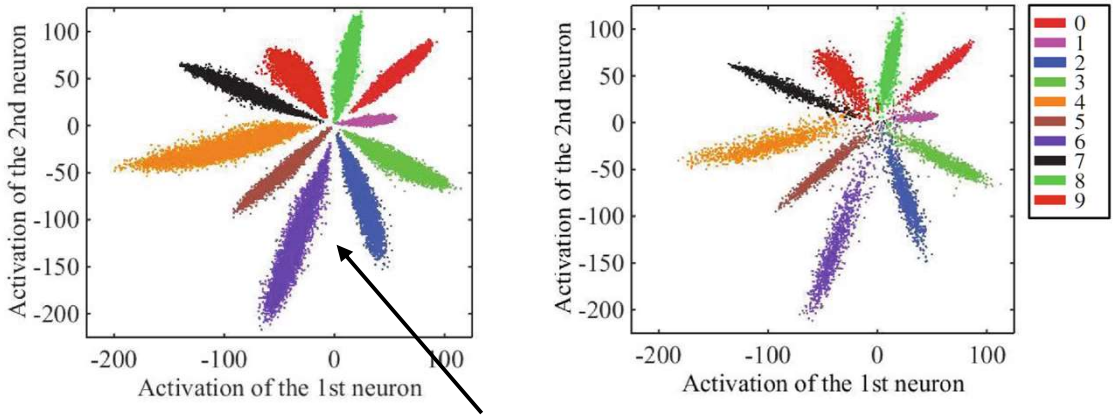
Visualization: Toy Example For Approach 1

- **Toy example** [Wen et al., 2016]:



Resulting embedding: not bad, but classes very close at center

On training data



On test data

embedded classes mostly linearly separable by final class score layer

Wen, Yandong, et al. "A discriminative feature learning approach for deep face recognition." *European conference on computer vision*. Springer, Cham, 2016.

Approach 2: Center Loss

- **Approach 2: Center Loss** [Wen et al., 2016]
- Idea: extend the crossentropy loss by a loss component that leads to more compact representations of classes in the embedding

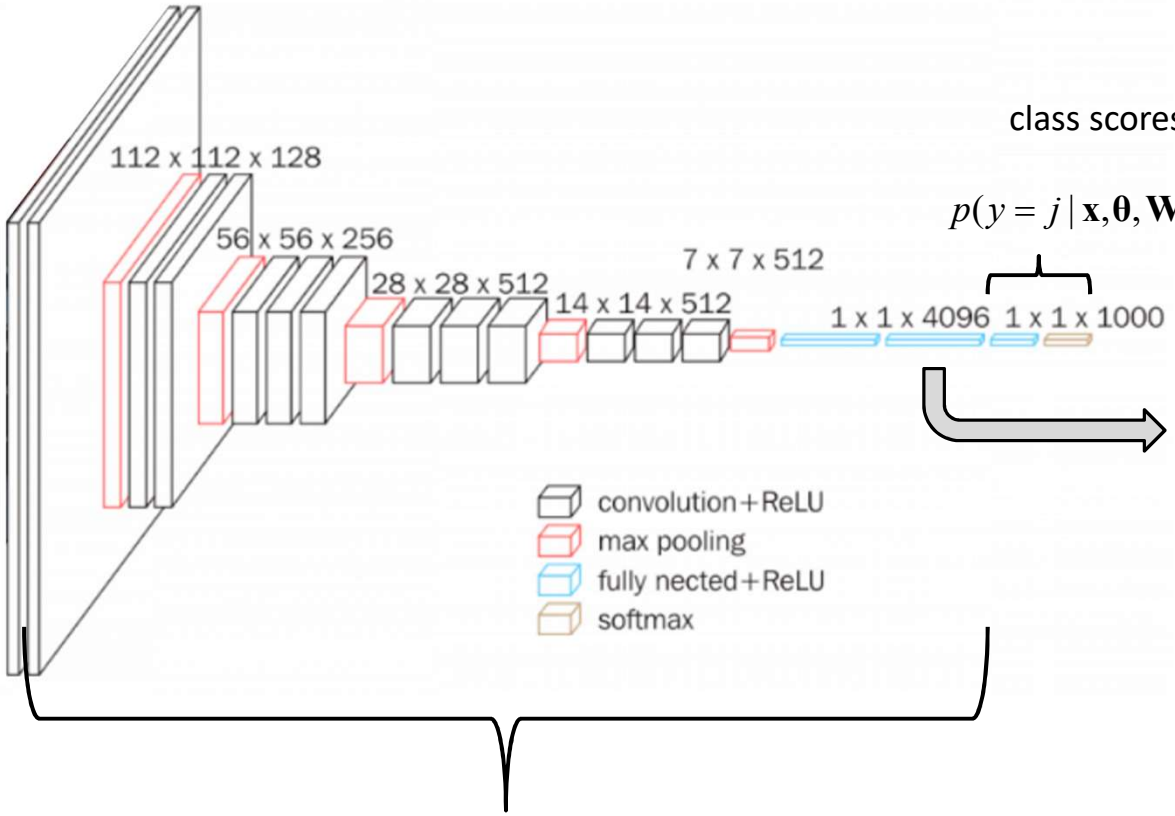
- Let $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}^D$ denote a deep neural network that produces an embedding: e.g. a normal classification network with the last layer (class scores) removed
- Let $\mathbf{z}_i = f_{\boldsymbol{\theta}}(\mathbf{x}_i) \in \mathbb{R}^D$ denote the computed embedding of input $\mathbf{x}_i$
- We can extend $f_{\boldsymbol{\theta}}$ to a classification network with a final fully connected layer producing class scores
- For simplicity, assume that this layer has no bias terms, and is parameterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{k \times D}$:

$$p(y = j \mid \mathbf{x}_i, \boldsymbol{\theta}, \mathbf{W}) = \frac{\exp(\mathbf{w}_j^{\mathrm{T}} \mathbf{z}_i)}{\sum_{l=1}^{k} \exp(\mathbf{w}_l^{\mathrm{T}} \mathbf{z}_i)}$$

$\mathbf{w}_1, \ldots, \mathbf{w}_k$ rows of weight matrix $\mathbf{W} \in \mathbb{R}^{k \times D}$ that parameterizes the final class score layer. Assuming no bias terms for simplicity.

Approach 2: Center Loss

- Visualization of model architecture and parameters (VGG example):



class scores/probabilites

$$p(y = j \mid \mathbf{x}, \boldsymbol{\theta}, \mathbf{W}) = \frac{\exp(\mathbf{w}_j^{\mathrm{T}} \mathbf{z}_i)}{\sum_{l=1}^{k} \exp(\mathbf{w}_l^{\mathrm{T}} \mathbf{z}_i)}$$

$\mathbf{z}_i$ embedding

$$f_{\boldsymbol{\theta}}(\mathbf{x}_i) = \mathbf{z}_i$$

Approach 2: Center Loss

- Assume training data $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathcal{X}$ with labels $\mathbf{y} = \{y_1, ..., y_n\}$

- The cross-entropy loss can be expressed as

class score of ground truth class

$$L_S = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{\exp(\mathbf{w}_{y_i}^{\mathrm{T}}\mathbf{z}_i)}{\sum_{j=1}^{k}\exp(\mathbf{w}_j^{\mathrm{T}}\mathbf{z}_i)}$$

cross-entropy loss is also called „softmax“ loss, after the way class probabilities are calculated as normalized exponentiated scores (so-called „softmax“ function)

- We could train the model simply with cross-entropy loss and remove the final classification layer
- But we can do better by extending the loss function to encourage the model to learn better embeddings

Center Loss Function

- In addition to the cross-entropy loss, we define a so-called **center loss**

$$L_C = \frac{1}{2n}\sum_{i=1}^{n}\left\|\mathbf{z}_i - \mathbf{c}_{y_i}\right\|_2^2$$

embedding should be close to center of its ground-truth class

where $\mathbf{c}_1,\ldots,\mathbf{c}_k \in \mathbb{R}^D$ are trainable parameters that act as „centers“ for the embeddings of the classes *1,…,k*

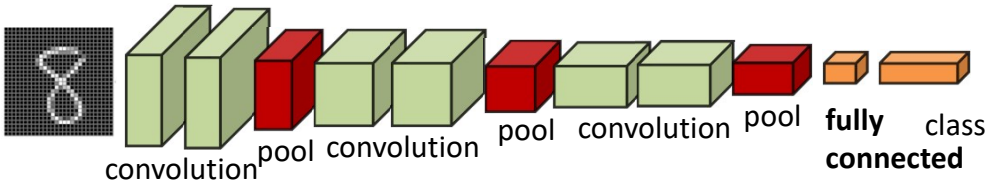
- We then train the model with a combined loss function

$$\begin{aligned} L &= L_S + \lambda L_C \\ &= -\frac{1}{n}\sum_{i=1}^{n}\log\frac{\exp(\mathbf{w}_{y_i}^{\mathrm{T}}\mathbf{z}_i)}{\sum_{j=1}^{k}\exp(\mathbf{w}_j^{\mathrm{T}}\mathbf{z}_i)} + \frac{\lambda}{2n}\sum_{i=1}^{n}\left\|\mathbf{z}_i - \mathbf{c}_{y_i}\right\|_2^2 \end{aligned}$$

where we have introduced a hyperparameter λ that controls the relative weight of the center loss in comparison to the cross-entropy loss

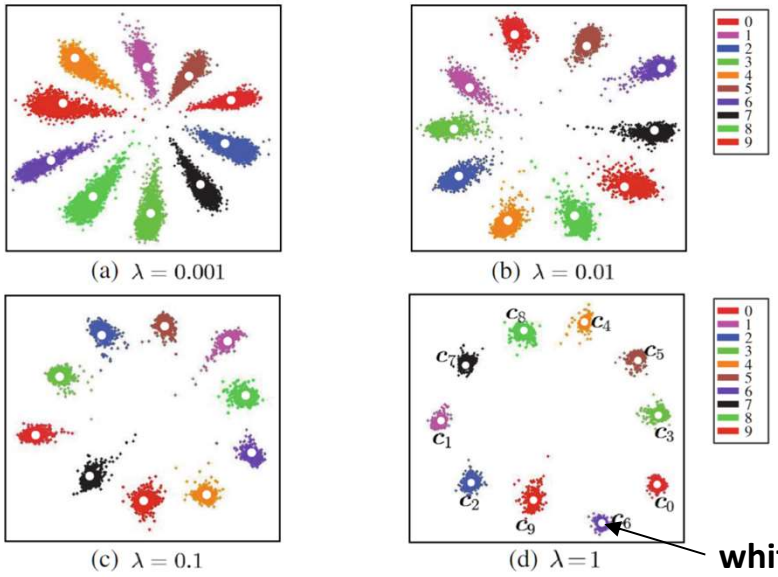
Toy Example With Combined Loss Function

- Look again at toy example, but now trained with combined loss function *L*



embedding

Resulting embeddings for different λ



white dots: estimated class centers $\mathbf{c}_j$

The addition of center loss leads to more compact class clusters

Better satisfies metric learning criterion: distances within class are small, between classes are large

More suitable to assigning instances based on distances

Wen, Yandong, et al. "A discriminative feature learning approach for deep face recognition." *European conference on computer vision*. Springer, Cham, 2016.

Estimating Class Centers: Gradient in Centers

- How do we estimate the class centers?
- The center loss is differentiable with respect to the class centers $\mathbf{c}_j$, which are learnable model parameters:

$$\frac{\partial L_C}{\partial \mathbf{c}_j} = \frac{\partial}{\partial \mathbf{c}_j} \frac{1}{2n} \sum_{i=1}^{n} \left\| \mathbf{z}_i - \mathbf{c}_{y_i} \right\|_2^2$$

$$= \frac{\partial}{\partial \mathbf{c}_j} \frac{1}{2n} \sum_{i=1}^{n} I(y_i = j) \left\| \mathbf{z}_i - \mathbf{c}_j \right\|_2^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} I(y_i = j)(\mathbf{c}_j - \mathbf{z}_i)$$

for summands with $y_i \neq j$, the summand contains no $\mathbf{c}_j$ and therefore the derivative is zero

- We can estimate the class centers based on this gradient

Estimating Class Centers: Gradient in Embedding

- How do we estimate the other model parameters $\boldsymbol{\theta}$?
- The center loss is also differentiable with respect to the embeddings $\mathbf{z}_i$: the differentiation with respect to the embedding of a single example is

$$\frac{\partial L_C}{\partial \mathbf{z}_i} = \frac{\partial}{\partial \mathbf{z}_i} \sum_{j=1}^{n} \frac{1}{2n} \left\| \mathbf{z}_j - \mathbf{c}_{y_j} \right\|_2^2$$

$$= \frac{\partial}{\partial \mathbf{z}_i} \frac{1}{2n} \left\| \mathbf{z}_i - \mathbf{c}_{y_i} \right\|_2^2$$

Derivative of summands for which $i \neq j$ is zero

$$= \frac{1}{n} (\mathbf{z}_i - \mathbf{c}_{y_i})$$

For single example $\mathbf{x}_i$, this is the upstream gradient that flows into the lower layers from the part of the loss function given by L_C

Algorithm: Training Model and Class Centers

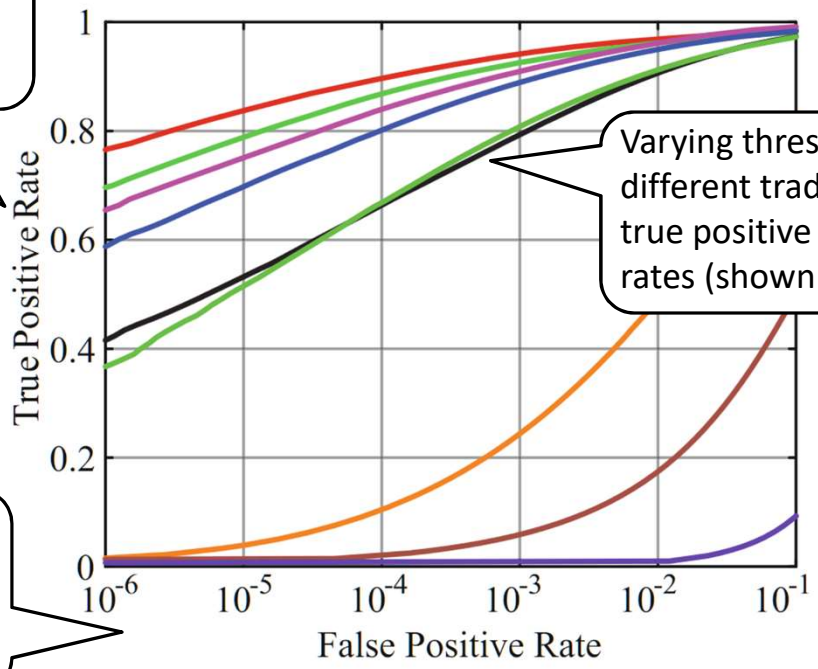
- Model parameters and class centers can now be jointly estimated by a stochastic gradient descent algorithm
  - In principle, standard stochastic gradient descent in all model parameters ($\boldsymbol{\theta}, \mathbf{W}, \mathbf{c}_1, \ldots, \mathbf{c}_k$)
  - In the [Wen et al., 2016] paper, a slightly modified custom gradient algorithm is proposed that uses a different learning rate for the update of the centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$ compared to the remaining parameters
  - Also uses a custom normalization factor for the gradient when updating the class centers
  - For details, see the paper

Wen, Yandong, et al. "A discriminative feature learning approach for deep face recognition. " *European conference on computer vision*. Springer, Cham, 2016.

Example: Center Loss in Face Recognition

- Face recognition: training a model on a subset of 490,000 images from the MegaFace data set [Miller et al., 2015]
- Experiments in face verification: decide if two face images show same person
- Say „match“ if distance in embedding space is lower than a threshold

True positive rate: for two face images showing the same person, how often does the model say „match“?



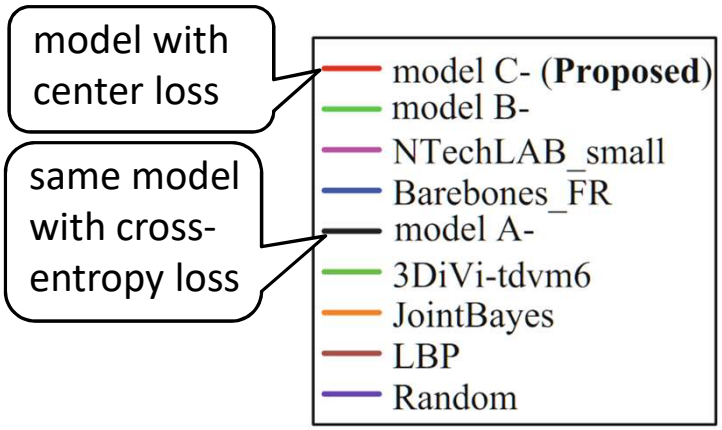
Varying threshold generates different trade-offs between true positive and false positive rates (shown as curve)

False positive rate: for two face images showing different persons, how often does the model say „match“?

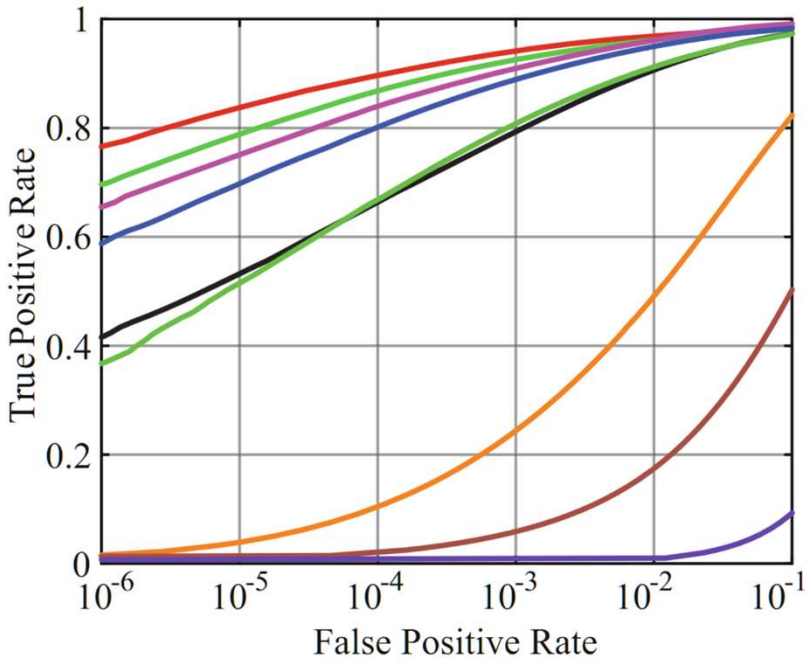
Wen, Yandong, et al. "A discriminative feature learning approach for deep face recognition." *European conference on computer vision*. Springer, Cham, 2016.

Example: Center Loss in Face Recognition

- Face recognition: training a model on a subset of 490,000 images from the MegaFace data set [Miller et al., 2015]
- Experiments in face verification: decide if two face images show same person
- Say „match“ if distance in embedding space is lower than a threshold



At FPR of 10^{-6}, center loss improves TPR from ≈ 0.41 to ≈ 0.78 (almost two-third error reduction)



Wen, Yandong, et al. "A discriminative feature learning approach for deep face recognition." *European conference on computer vision*. Springer, Cham, 2016.

Triplet Loss: Optimizing Distances Directly

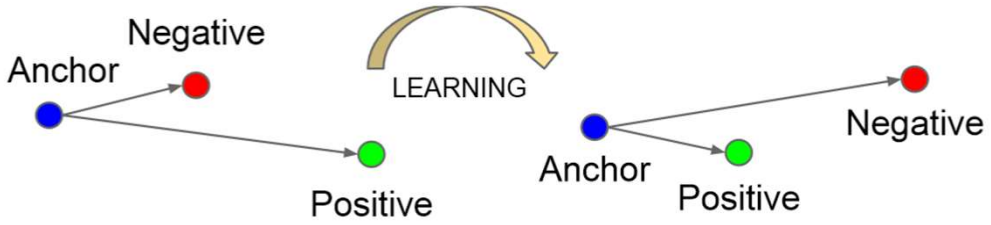
- Center loss optimizes distances by learning a center for each class in embedding space and enforcing that instances are close to their class center
- Alternative approach: look at pairs of instances and enforce that distance for pair of same class is smaller than distance for pair of different classes

- **Triplet loss [Schroff et al., 2015]: enforce that distance for a positive pair (same class) is smaller than distance for negative pair (different class) by at least a fixed margin**
- Specifically, triplet loss always looks at triplets of instances:

„Anchor“ and „Positive“ are of the same class.

„Negative“ is an instance from a different class

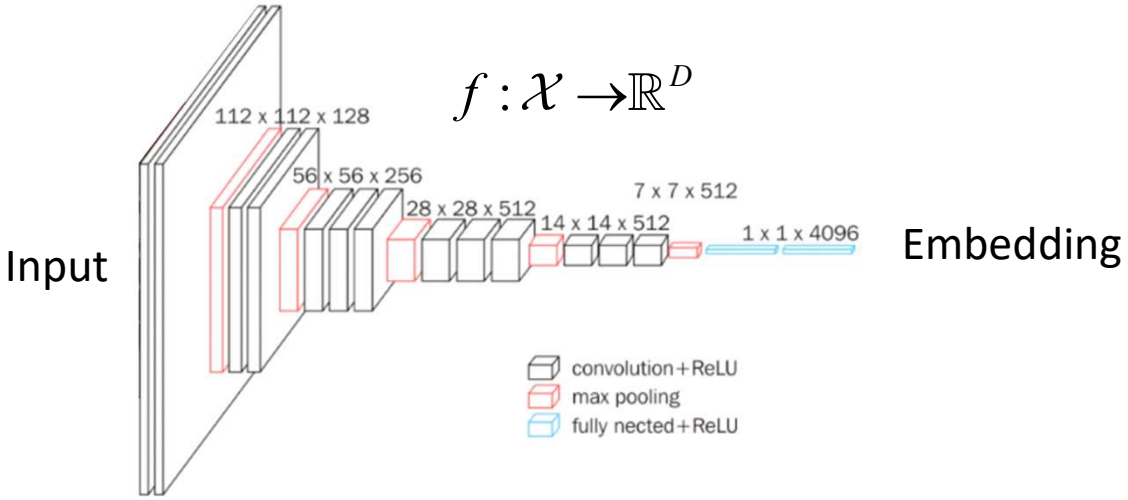
d(Anchor,Positive) should be smaller than d(Anchor, Negative)



Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

Triplet Loss: Embedding Model

- When training with triplet loss, we only need an embedding model, without a final classification layer (can be any architecture)



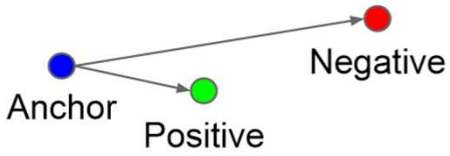
- Embedding model f maps instances $\mathbf{x} \in \mathcal{X}$ to embeddings $f(\mathbf{x}) \in \mathbb{R}^D$
- Loss function directly works at the level of the embeddings

Triplet Loss Formally

- Triplet loss looks at triplets of three instances, two of which are in same class

Set of all such triplets in training data:

$$\mathcal{T} = \{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \,|\, i, j, k \in \{1, \ldots, n\}, y_i = y_j, y_i \neq y_k\}$$



- We enumerate the set $\mathcal{T}$ by

$$\mathcal{T} = \{(\mathbf{x}_1^a, \mathbf{x}_1^p, \mathbf{x}_1^n), \ldots, (\mathbf{x}_M^a, \mathbf{x}_M^p, \mathbf{x}_M^n)\} \qquad M >> n$$

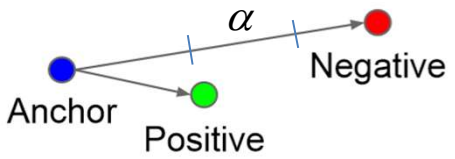
- Loss operates at triplet level:

$$L = \sum_{i=1}^{M} \ell(\mathbf{x}_i^a, \mathbf{x}_i^p, \mathbf{x}_i^n)$$

Triplet Loss Formally

- The objective is that the distance for the positive pair (Anchor, Positive) is smaller by margin α than distance for the negative pair (Anchor, Negative):

$$\left\| f(\mathbf{x}_i^a) - f(\mathbf{x}_i^p) \right\|_2^2 + \alpha < \left\| f(\mathbf{x}_i^a) - f(\mathbf{x}_i^n) \right\|_2^2 \qquad (1)$$



- Loss function:

$$\ell(\mathbf{x}_i^a, \mathbf{x}_i^p, \mathbf{x}_i^n) = \max(0, \left\| f(\mathbf{x}_i^a) - f(\mathbf{x}_i^p) \right\|_2^2 - \left\| f(\mathbf{x}_i^a) - f(\mathbf{x}_i^n) \right\|_2^2 + \alpha)$$

Loss is zero if condition (1) above is fulfilled.

Otherwise, loss is positive and scales linearly with

$$\left\| f(\mathbf{x}_i^a) - f(\mathbf{x}_i^p) \right\|_2^2 - \left\| f(\mathbf{x}_i^a) - f(\mathbf{x}_i^n) \right\|_2^2 .$$

Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

Hard Negative Mining

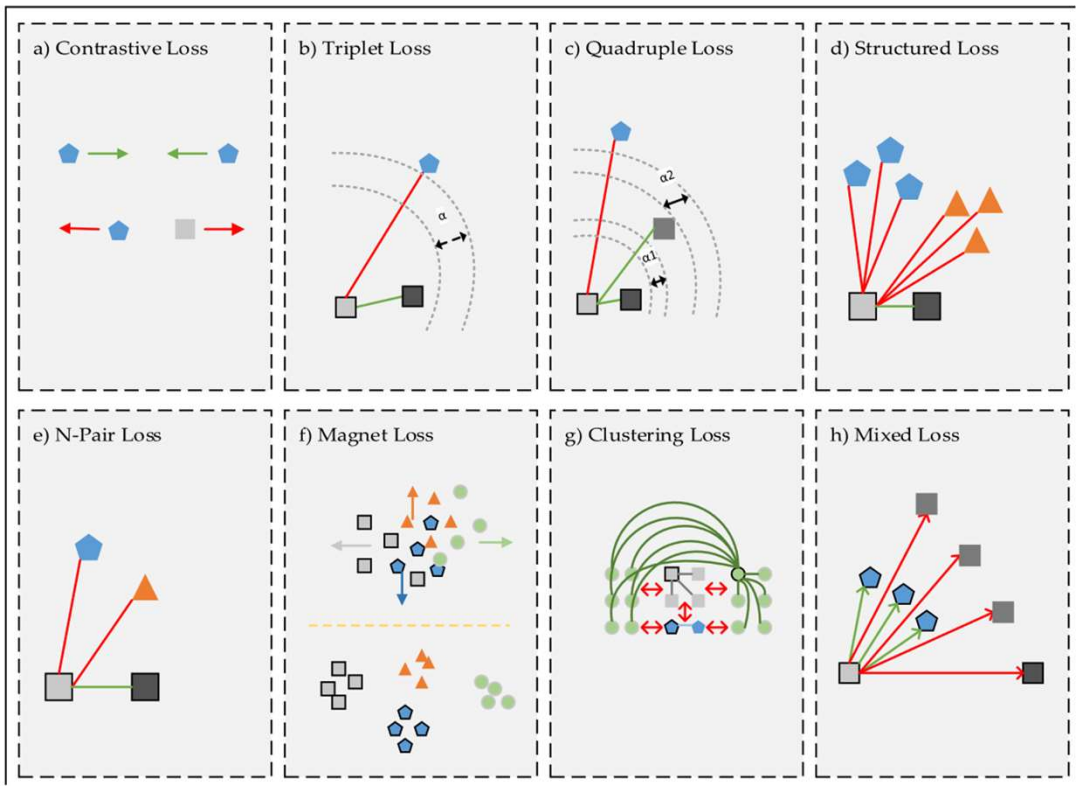
- In principle, the loss is defined over all triplets: $L = \sum_{i=1}^{M} \ell(\mathbf{x}_i^a, \mathbf{x}_i^p, \mathbf{x}_i^n)$
- Mini-batch training:
  - Use relatively large minibatches (approx. 1000-2000 instances), with a certain balance of positive and negative pairs in batch (no details)
  - Sum loss over triplets that can be formed from instances in the mini-batch

- Once a model has been partially trained, it is often the case that a majority of triplets already fulfill condition (1) on last slide.
- Those triplets do not contribute to loss function or gradient, but contribute to computational costs in backpropagation

- Need to perform **hard-negative mining**: for an anchor example, select one instance of same class and then find an instance of other class such that condition (1) is violated
- Different strategies possible, see paper for details

Other Pair-based Metric Learning Loss Functions

- Triplet loss is only one example for a pair-based loss for metric learning
- Many other geometrically motivated loss functions have been proposed



Kaya, Mahmut, and Hasan Şakir Bilge. "Deep metric learning: A survey." *Symmetry* 11.9 (2019): 1066.

Summary Metric Learning

- In applications such as face recognition, the goal is to learn an embedding of instances in a space such that instances are clustered according to classes
- This allows to have new classes (e.g., new identities in face recognition) at test time
- The **center loss** accomplishes this by learning for each class a „class center“ in the embedding space, and enforcing that instances are close to their class center
- The **triplet loss** directly optimizes a criterion that enforces that in the embedding space, instances from the same class are closer to each other than to instances from other classes