# Image Recognition

# Applications of Face Recognition Technology

FRAUD DETECTION
PAYMENTS
HEALTHCARE
ADVERTISING

- Prevent Retail Crime. ...
- Unlock Phones. ...
- Smarter Advertising. ...
- Find Missing Persons. ...
- Protect Law Enforcement. ...
- Aid Forensic Investigations. ...
- Identify People on Social Media Platforms.
- CONTROL ACCESS TO SENSITIVE AREAS

# METRIC LEARNING

Metric learning learns a **metric function from training data to calculate the similarity or distance between samples**

## What is a Metric ?

- ❑ A metric is a pairwise distance function over a set ***X***.

$$d : X \times X \rightarrow \mathbb{R}^+$$

- ❑ For any points (x,y,z ϵ X), the following conditions must be satisfied.
  - ✓ Non negativity
  - ✓ Symmetry
  - ✓ Identity
  - ✓ triangle inequality

## Commonly used Metrics

- ❑ Manhattan Distance

$$d_1(x,x') = \sum_{i=1}^{p} |x_i - x_i'|$$

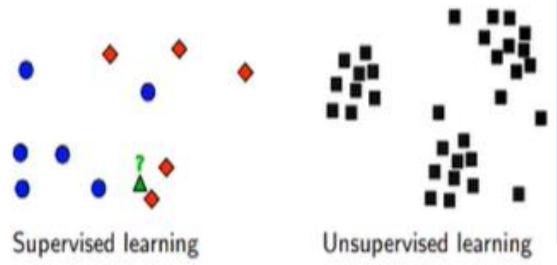
- ❑ Euclidean Distance

$$d_2(x,x') = \sqrt{\sum_i (x_i - x_i')^2}$$

## Importance of Metrics

Metrics (distance functions) are used in following domains:

- ❑ Classification
- ❑ Regression
- ❑ Clustering
- ❑ Ranking
- ❑ Information Retrieval



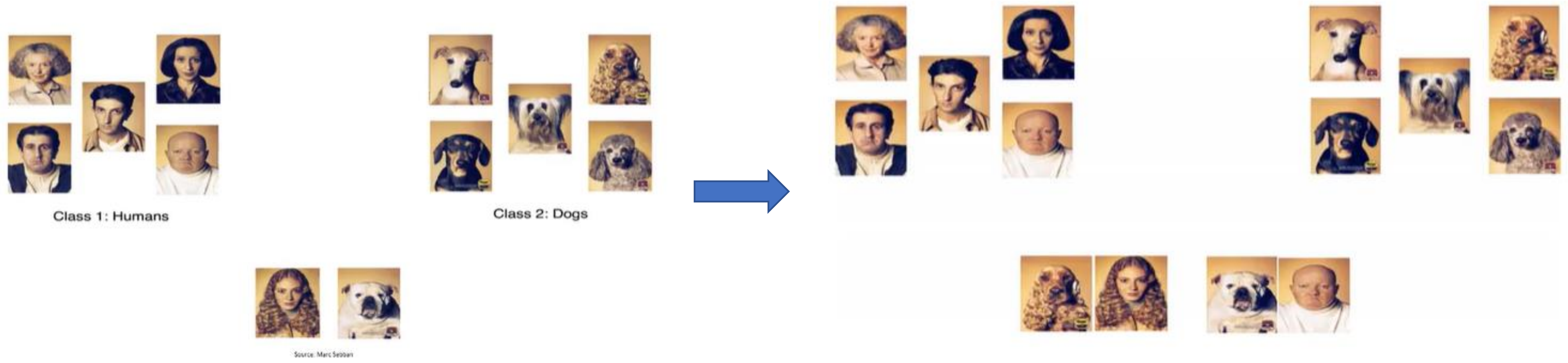
Euclidean based distances metrics are commonly used

- ➢ These distance **metrics are not parametrized**, which means Euclidian distance metrics do not capture the **specificities of the data** suppose if data has outliers it wont considers the effect of it while calculating distance metric

## HOW TO CHOOSE A GOOD METRIC?

*"Rather than choosing a specific metric from standard metrics, we can design a specific metric according to the problem at hand"*

**Limitations of standard Metric functions :**
Lets consider below example of **dogs and humans we have binary data**, lets assume that we have extracted feature maps and **used the standard distance metric to calculate the similarity** and used **1 nearest neighbor algorithm we get wrong classification** as we see below, because the **quality of standard metrics** (what ever distance metric) are not very well suited for this kind of data due to **limitations of standard metrics like they are not parametrized hence they are not allowed to capture the specificities of the data.**

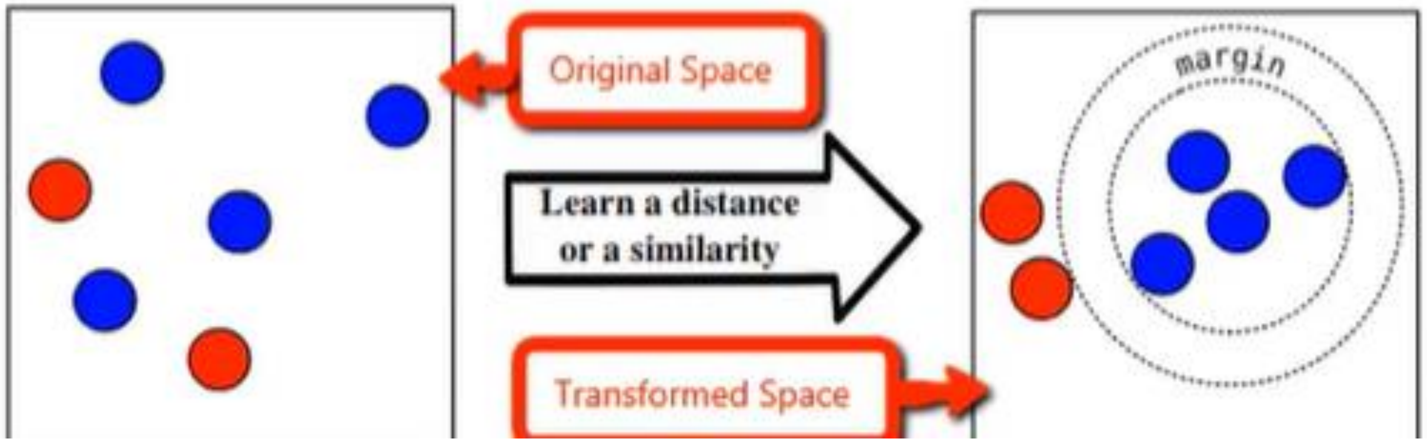


We clearly see that **hair comes in to the role of** metric learning **which is to push one of them far away,** that is the objective of metric leering to **represents these features in another space** where the **data classes are same they are close each other and where the data classes are different they are far away** so that discrimination made easily

# Definition of Metric Learning

*"The main idea in metric learning is to learn a metric that assigns small distance to pairs of examples that are semantically similar and large distance to pair of examples that are semantically dissimilar"*

*"Data is projected from original space to a new space where some constraints are satisfied"*

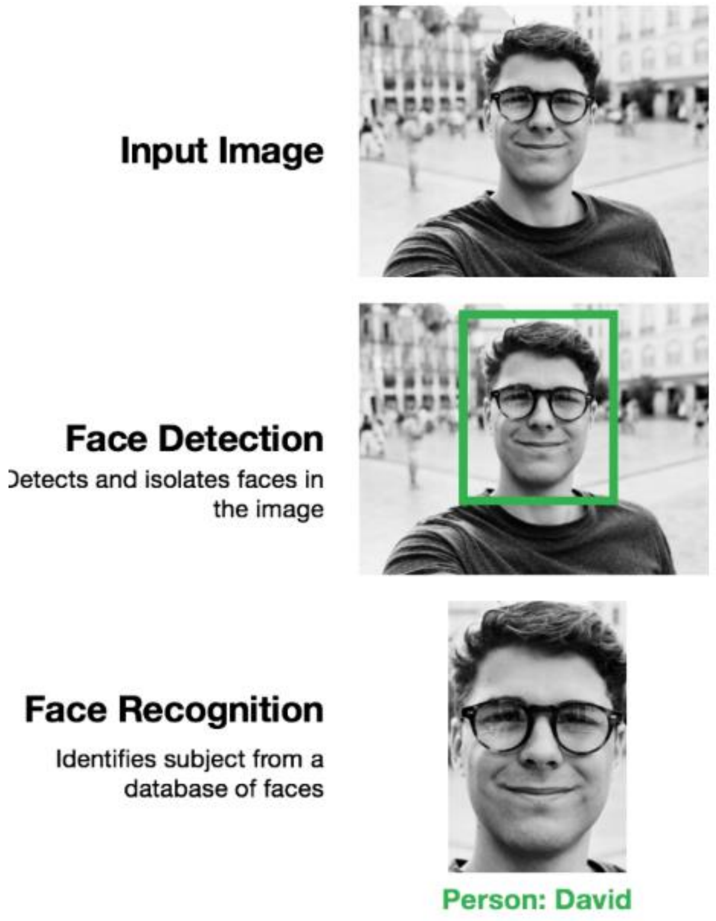


## IMAGE RECOGNITION

Face recognition problem **into smaller steps and subproblems**. That way, we can better understand what's going on under the hood of a facial recognition system. A face recognition problem can be broken down into the following smaller subproblems:

•**Face detection**: **Detect and isolate faces in the image**. In an image with multiple faces, we need to detect each of them separately. In this step, we should also crop the detected faces from the original input image, to identify them separately.

•**Face recognition**: For each detected face in the image, we run it through a neural network to classify the subject. Note that we need to repeat this step for each detected face.


Input Image
Face Detection
Detects and isolates faces in the image
Face Recognition
Identifies subject from a database of faces
Person: David

Intuitively, this process makes a lot of sense. If we think of how humans recognize faces, we see that it is very similar to the process that we described. Given an image, our eyes immediately **zoom into each face (face detection), and we recognize the faces individually (face recognition).**

Source -Neural Network Projects with Pythonby James Loy*Published*

# REQUIREMENTS OF FACE RECOGNITION SYSTEMS

## 1. Speed

The first requirement of a facial recognition system is that **they need to be fast**. If we look at the **onboarding process of the facial recognition systems in our smartphones**, we usually need to use the front-facing camera in the phone to scan our face at various angles for a few seconds. During this short process, our phone captures images of our face, and uses an image to train a neural network to recognize us. This process needs to be fast.

The following picture shows the typical onboarding process for a facial recognition system in smartphones:

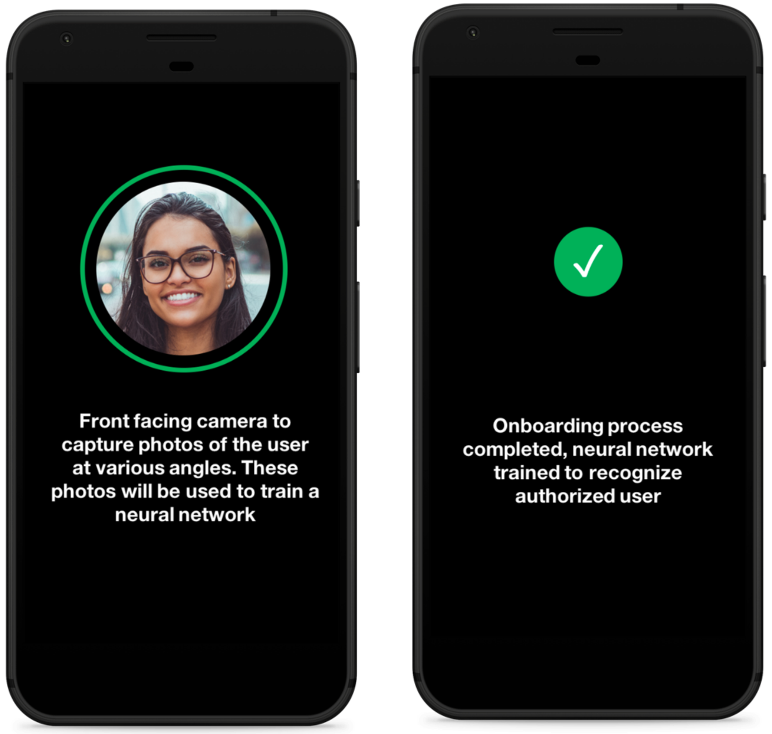

Front facing camera to capture photos of the user at various angles. These photos will be used to train a neural network
Onboarding process completed, neural network trained to recognize authorized user

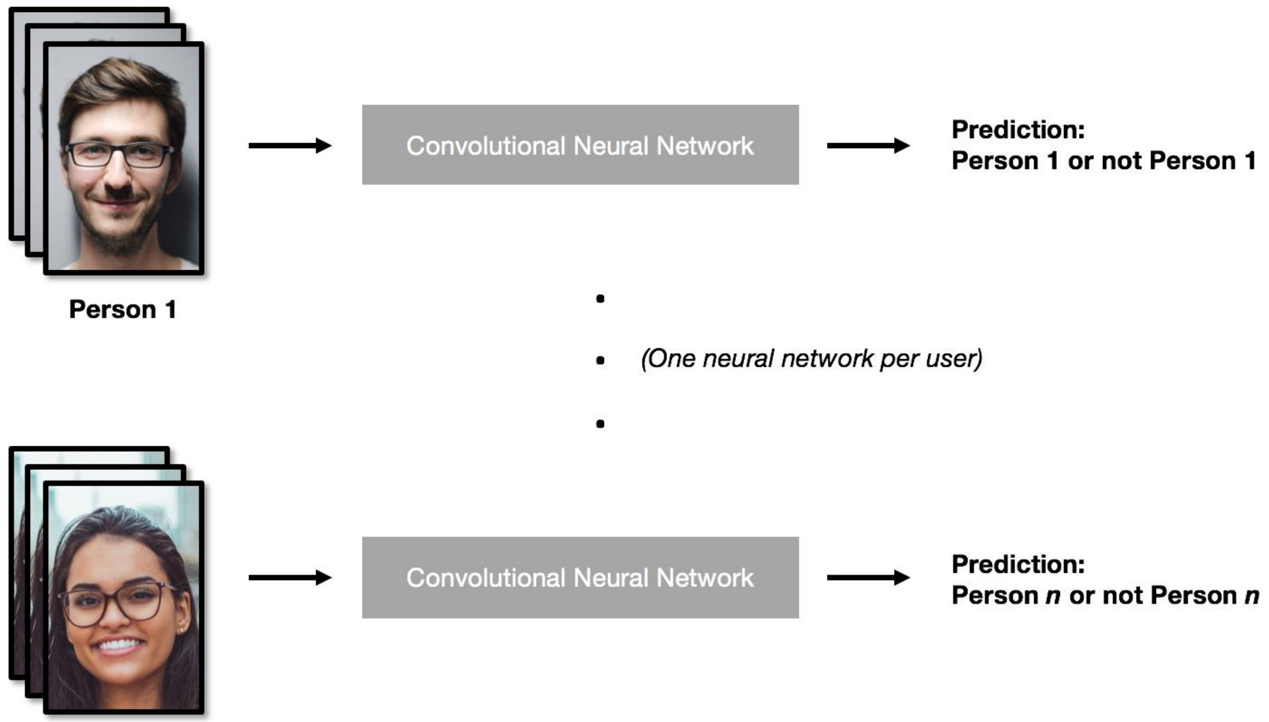
***Image Classification Using CNNs:*** we saw how slow it is to train a CNN to identify images of cats and dogs. Even with powerful GPUs, training a CNN can sometimes take hours (or even days!).

From a user experience point of view, it is not practical for the onboarding process of facial recognition systems to take this long. Therefore, **CNNs do not satisfy the speed requirement of facial recognition systems**.

# REQUIREMENTS OF FACE RECOGNITION SYSTEMS

## 2. Scalability

The second requirement of facial recognition systems is **that it needs to be scalable.** The model that we train must **ultimately be able to scale to millions of different users**, each with a unique face. Again, this is **where CNNs fall short**.

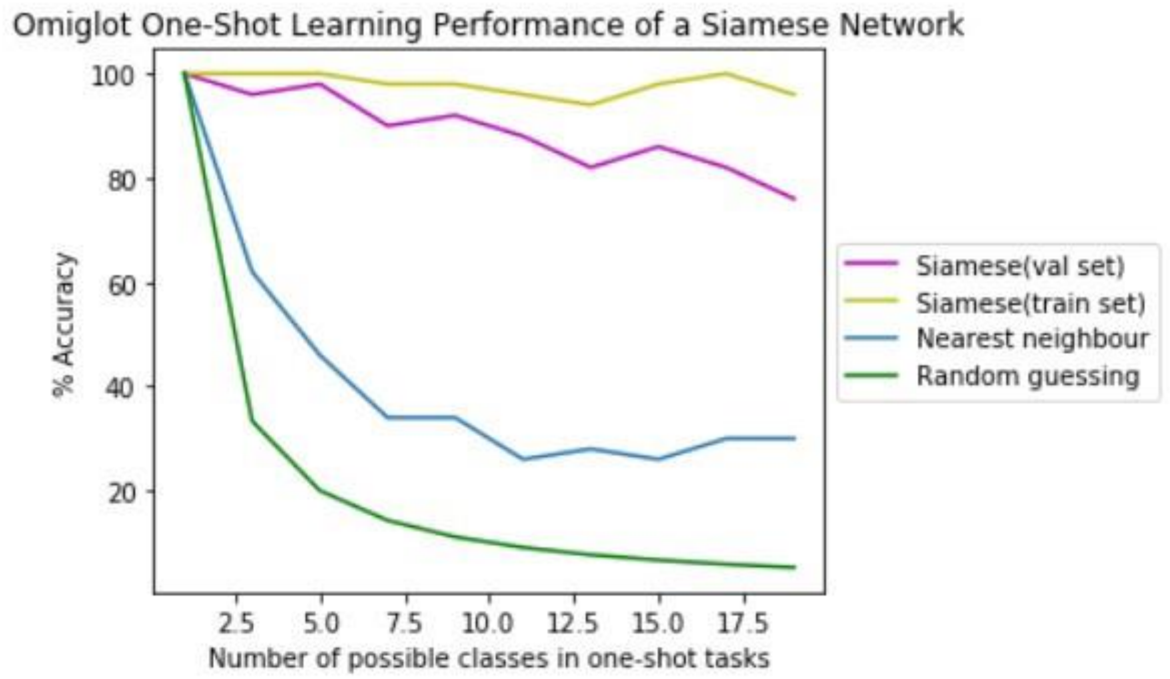

Convolutional Neural Network
Prediction:
Person 1 or not Person 1
Person 1
(One neural network per user)
Convolutional Neural Network
Prediction:
Person *n* or not Person *n*

- *For example W*e trained a CNN to differentiate cats from dogs, this neural network is only able to identify and classify **images of cats and dogs**, and not of other animals, which it was not trained on.
- This means that if we were to **use CNNs for facial recognition, we would have to train a separate neural network for each individual user**. This would simply be unworkable from a scalability point of view!

# REQUIREMENTS OF FACE RECOGNITION SYSTEMS

**3. High accuracy with small data**

- The third requirement of a facial recognition system is that it **needs to be sufficiently accurate** (hence secure) while working with a small amount of training data.
- Going back to the example of the onboarding process for facial recognition in smartphones, we can see that only a handful of photos are taken, and we need to be able to train our model, using this limited dataset.



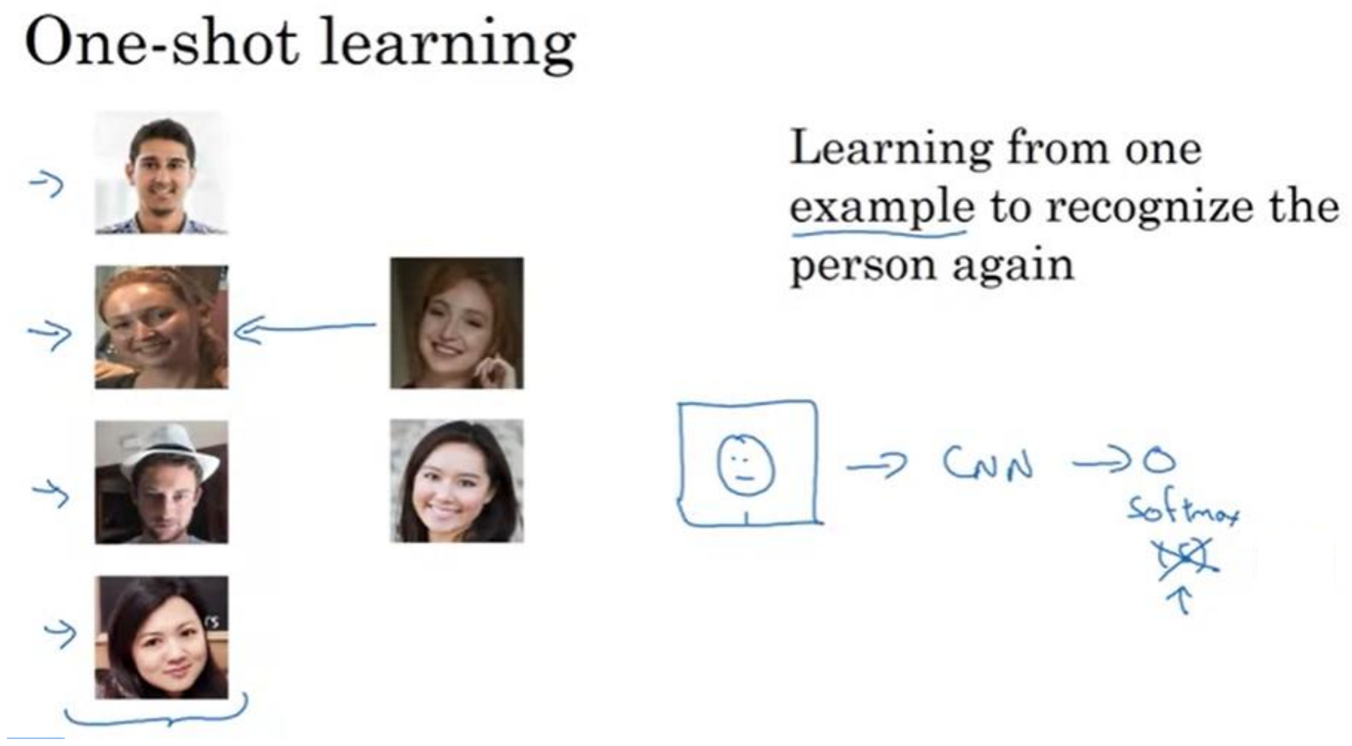
- Once again, CNNs do not satisfy this requirement, because we **need lots of images to train a CNN.** While CNNs are fairly accurate at image classification tasks, this comes at the expense of requiring a huge training set.

- Imagine having to take **thousands of selfies** with our **smartphones before we can start using the facial recognition** systems in our phones! This would simply not work for most facial recognition systems.

# ONE-SHOT LEARNING

- One of the challenges of face recognition is that you need to solve **the one-shot learning problem.** What that means is that for most face recognition applications you need to be able to recognize a person **given just one single image**, or given just one example of that person's face

- Historically, **deep learning algorithms don't work well if you have only one training example**. Let's see an example of what this means and talk about how to address this problem. Let's say you have a database of four pictures of employees in you're organization

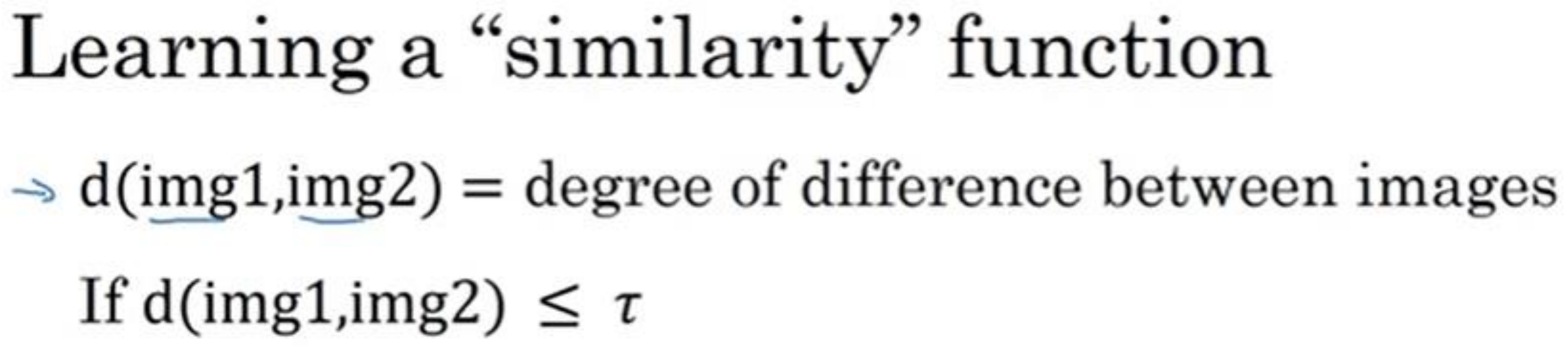


One-shot learning

Learning from one example to recognize the person again

## ONE-SHOT LEARNING

This can be achieved by



Learning a “similarity” function

→ d($\underline{\text{img1}}$,$\underline{\text{img2}}$) = degree of difference between images

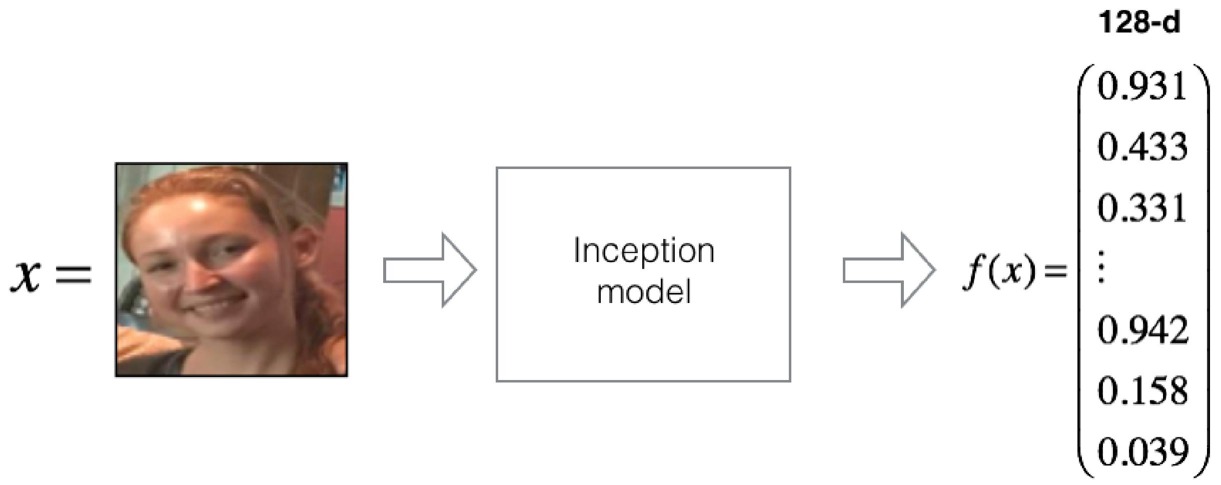
If d(img1,img2) $\leq \tau$

- In particular, we want a neural network to learn a function which going to denote **d**
- which inputs two images and outputs **the degree of difference between the two images**.
- If the two images are of the **same person, you want this to output a small number** and if the two images are of two very **different** people you want **it to output a large number**.
- During recognition time, if the degree of difference between them is less than some threshold called tau, which is a hyperparameter. Then you would predict that these two pictures are the same person and if it is greater than tau, you would predict that these are different persons.

# IMAGE EMBEDDINGS

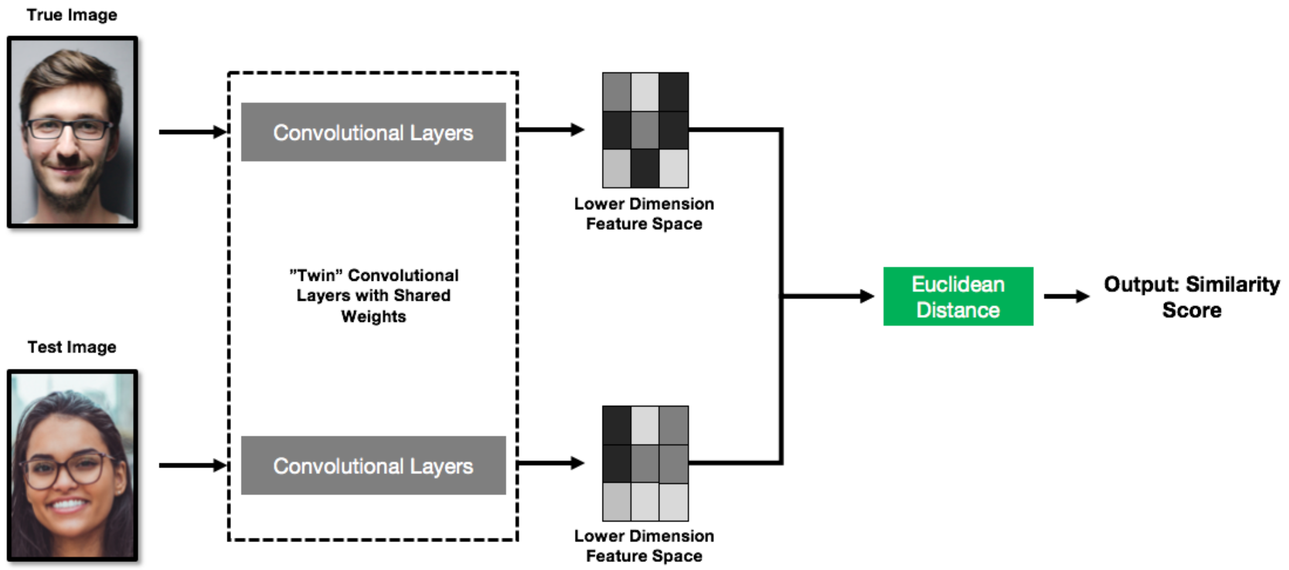
## Embeddings

- To analyse the similarity between two pictures, we need to be able to transform our input picture into a smaller representation, **say a single vector**. This **representation is usually called an embedding**. We need to build theses embeddings so that they have the following properties:
  - Two similar images produce two embeddings so that the mathematical distance between them is small
  - Two very different images produce two embeddings so that the mathematical distance between them is large
  - The embedding is L2-normalized, ie each embedding is forced to be on the unit hypersphere



# SIAMESE NEURAL NETWORKS

- Intuitively, humans recognize faces by **comparing their key features.** For example, humans use features **such as the shape of the eyes, the thickness of the eyebrows, the size of the nose, the overall shape of the face, and so on to recognize a person**.
- This ability comes naturally to us, and we are rarely affected by variations in angles and lighting. **Could we somehow teach a neural network to identify these features from images of faces**, before using the Euclidean distance to measure the similarity between the identified features?
- This should sound familiar to you! As we have seen in the previous chapters, **convolutional layers excel in finding such identifying features automatically.** For facial recognition, researchers have found that when convolutional layers are applied to human faces, **they extract spatial features, such as eyes and noses.**
- Use **convolutional layers to extract identifying features** from faces..
  - Using the Euclidean distance, measure the difference of the two lower-dimension vectors output from the convolutional layers.
  - One last thing to note is that, since we are feeding two images into our neural network simultaneously, however, we require the two separate sets of convolutional layers **to share the same weights**, because we want **similar faces to be mapped to the same point in the lower-dimension feature space.**


True Image
Convolutional Layers
Lower Dimension Feature Space
"Twin" Convolutional Layers with Shared Weights
Euclidean Distance
Output: Similarity Score
Test Image
Convolutional Layers
Lower Dimension Feature Space

We can thus think of these two sets of convolutional layers as twins, as they share the same weights.

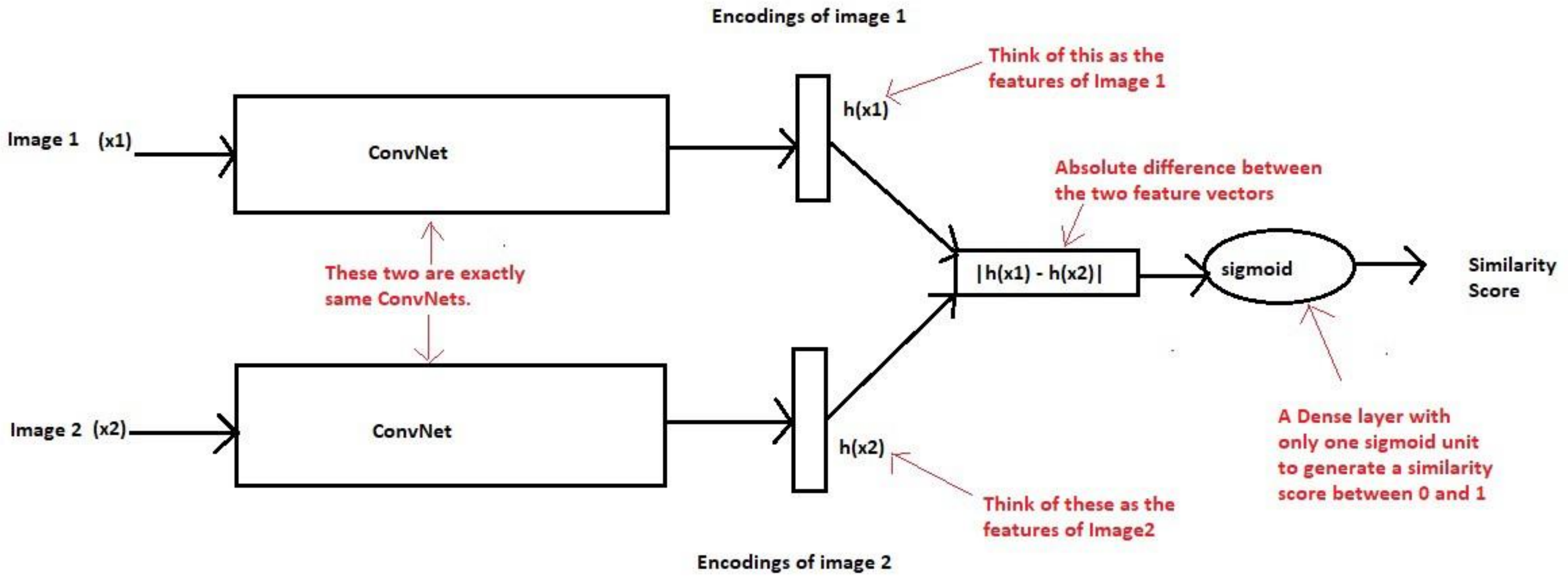
# Why they work?

Siamese networks work well in these tasks because,

- **Sharing weights across subnetworks** means fewer parameters to train for, which in turn means less data required and **less tendency to overfit**.
- Each subnetwork essentially produces a representation of its input. (“Signature Feature Vector”  or “Face Feature Vector”)

If your inputs are of the same kind, like matching two pictures, it makes sense to use similar model to process similar inputs. This way you have representation vectors with the same semantics, making them easier to compare.

# Duplet Siamese Training


Encodings of image 1
Think of this as the features of Image 1
h(x1)
Image 1 (x1)
ConvNet
Absolute difference between the two feature vectors
These two are exactly same ConvNets.
|h(x1) - h(x2)|
sigmoid
Similarity Score
Image 2 (x2)
ConvNet
h(x2)
A Dense layer with only one sigmoid unit to generate a similarity score between 0 and 1
Think of these as the features of Image2
Encodings of image 2

Source: [https://sorenbouma.github.io/blog/oneshot/](https://sorenbouma.github.io/blog/oneshot/)

# SIAMESE NEURAL NETWORKS – CONTRASTIVE LOSS

- It is a **Distance-based Loss function** (as opposed to **prediction error-based** Loss functions like Logistic loss or Hinge loss used in Classification).

- Like any distance-based loss, it tries to ensure that **semantically similar examples are embedded close together. It is calculated on Pairs (other popular distance-based Loss functions are Triplet & Centre Loss, calculated on *Triplets* and *Pointwise* respectively)**

Take a look at the following variables:

- $Y_{true}$: Let $Y_{true}$ be *1* if the two input images are from the same subject (same face) and 0 if the two input images are from different subjects (different faces)
- $D$: The predicted distance output from the neural network

So, the *Contrastive Loss* is defined as follows:

$$Contrastive\ Loss = Y_{true} * D^2 + (1 - Y_{true}) * max(margin - D, 0)$$

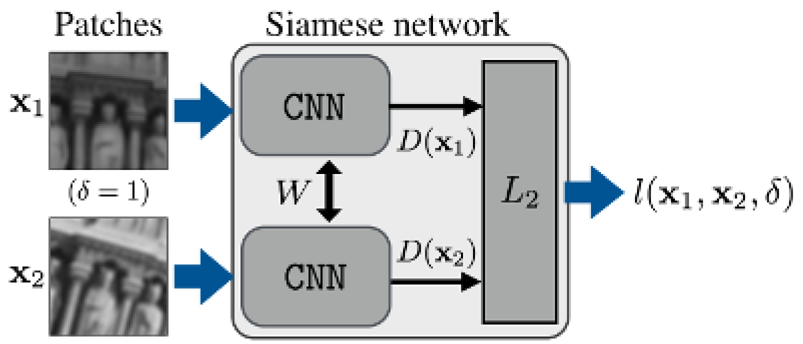
Here margin is hyper parameter which maximizes the margin between the **true image vs Anchor image(test image)**
All it does is simply produce a high loss( that is, a penalty) when predicted distance is large when the faces are similar, and a low loss when the predicted distance is small, and vice versa for the case when the faces are different

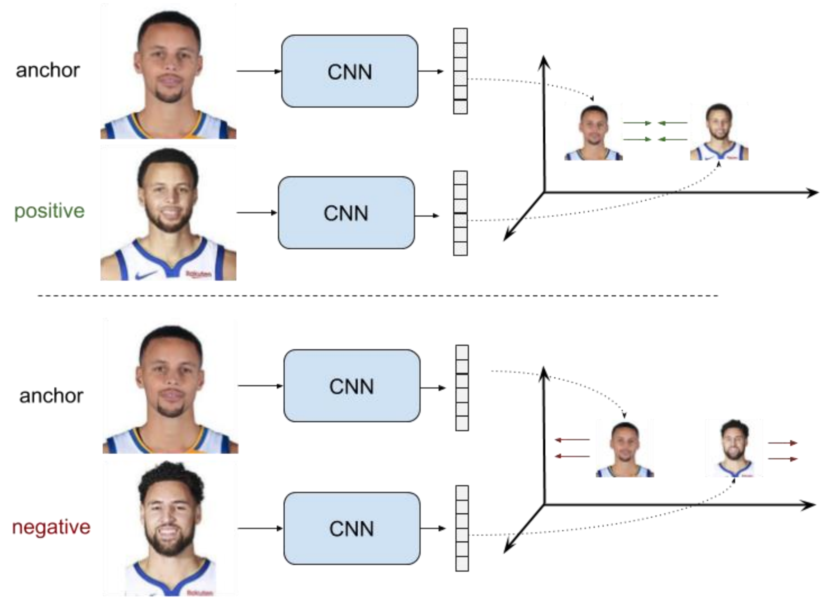
## How to train a Siamese Network?

Two commonly used metric learning Loss functions

- Regular or Duplet Siamese Training
- Triplet Siamese Training

# Regular or Duplet Siamese Training





**Is this setup positive and negative pairs of training data points are used. Positive pairs are composed by an anchor sample $x_a$ and a positive sample $x_p$, which is similar to $x_a$ in the metric we aim to learn, and negative pairs composed by an anchor sample $x_a$ and a negative sample $x_n$, which is dissimilar to $x_a$ in that metric.**

**The objective is to learn representations with a small distance $d$ between them for positive pairs, and greater distance than some margin value $m$ for negative pairs**. Pairwise Ranking Loss forces representations to have 0 distance for positive pairs, and a distance greater than a margin for negative pairs. Being $r_a$, $r_p$ and $r_n$ the samples representations and $d$ a distance function, we can write:

$$L = \begin{cases} d(r_a, r_p) & if \quad PositivePair \\ max(0, m - d(r_a, r_n)) & if \quad NegativePair \end{cases}$$

# TRIPLET NETWORK

- **A query picture, called the Anchor**
- **A picture from the same class as the anchor, called the Positive**
- **A picture from a different class to the anchor, called the Negative**

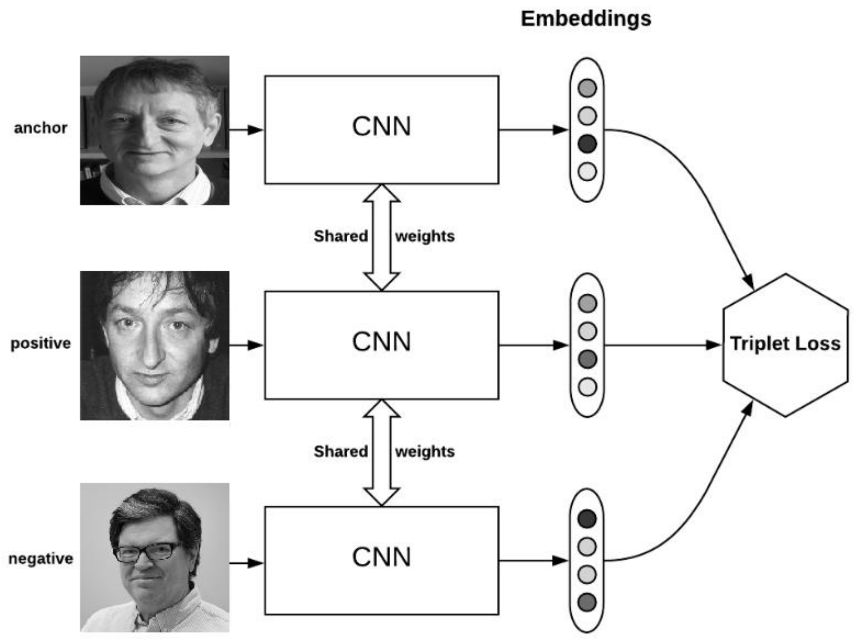
With these triplets of three images, **(let’s call these embeddings A, P and N respectively)**, we want
***distance(A,P) < distance(A,N)***
Let’s rewrite it differently
***distance(A,P) - distance(A,N) < 0***
Prevent our network from learning an easy solution that satisfies the equation **by outputting zeros for everything**, let’s force a margin between AP and AN with a margin parameter:
*distance(A,P) — distance(A,N) + margin < 0*
Our Loss function will be based on this entity and will then have the following form:

$$L=\max(d(A,P)-d(A,N)+\text{margin},0)$$

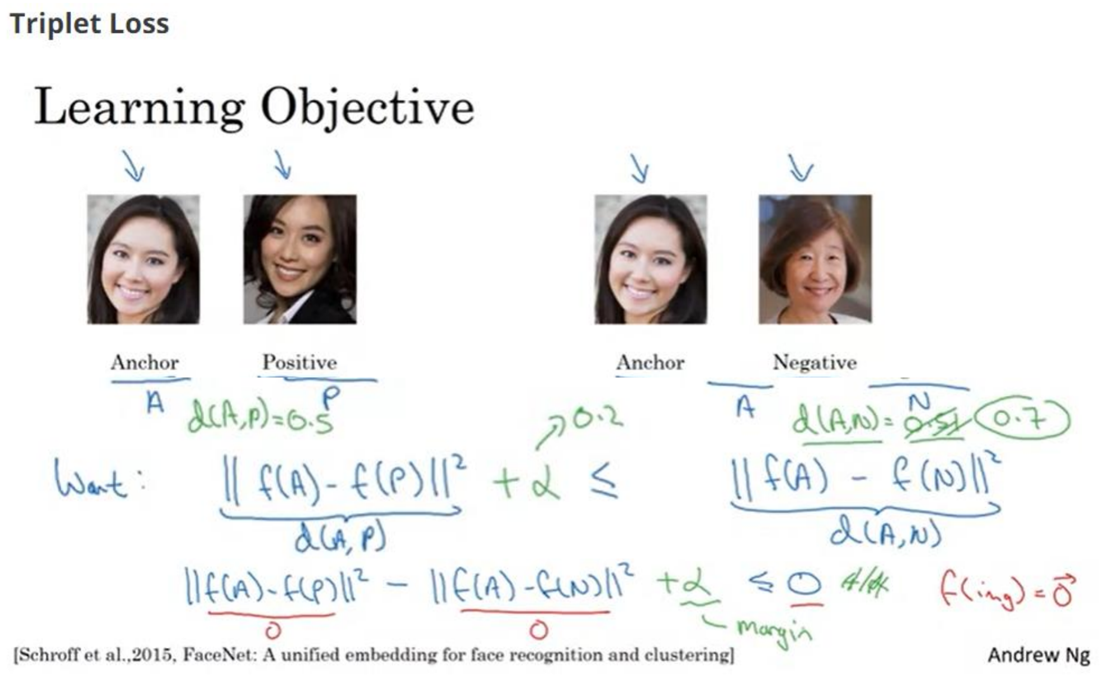
# SIAMESE NEURAL NETWORKS – TRIPLET LOSS FUNCTION

- Very often, learning in Siamese networks is done by the triplet loss function. Such models **require three input images on training phase, and the loss is calculated as below**

1. Extract embeddings from an **anchor input image *a*.**
2. Extract embeddings from a **positive input image *p* (same class as the anchor).**
3. Extract embeddings from a negative input image *n* (different class from the anchor).
4. Calculate the Euclidean distances ***d(a, p)* and *d(a, n).*** Ideally, the first distance should be as small as possible, while the latter should be as big as possible.
5. **The loss function is defined as: *L = max(d(a, p)- d(a,n) + α, 0)*, where *α or margin* is a parameter that defines how far away the dissimilarities should be, and enforces a distinction between the positive and the negative image.**



## Margin


Triplet Loss

Learning Objective

Anchor Positive Anchor Negative

$A$ $d(A,P)=0.5$ $P$ $\nearrow 0.2$ $A$ $d(A,N)=$ ~~0.51~~ $\boxed{0.7}$ $N$

$$\text{Want: } \underbrace{\| f(A) - f(P) \|^2}_{d(A,P)} + \alpha \le \underbrace{\| f(A) - f(N) \|^2}_{d(A,N)}$$

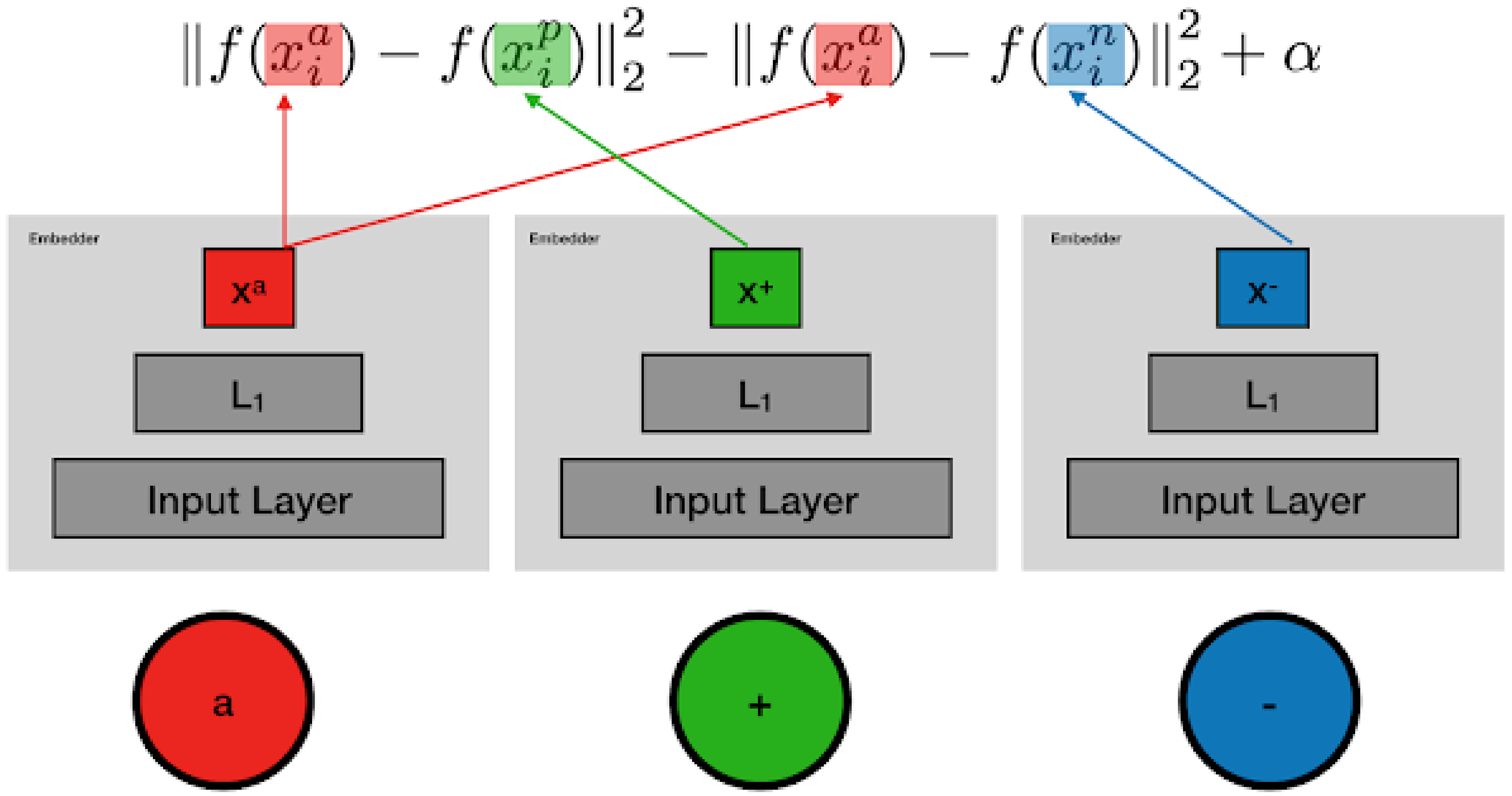
$$\underbrace{\| f(A) - f(P) \|^2}_{0} - \underbrace{\| f(A) - f(N) \|^2}_{0} + \underbrace{\alpha}_{\text{margin}} \le 0 \qquad f(\text{img}) = \vec{0}$$

[Schroff et al.,2015, FaceNet: A unified embedding for face recognition and clustering]

Andrew Ng

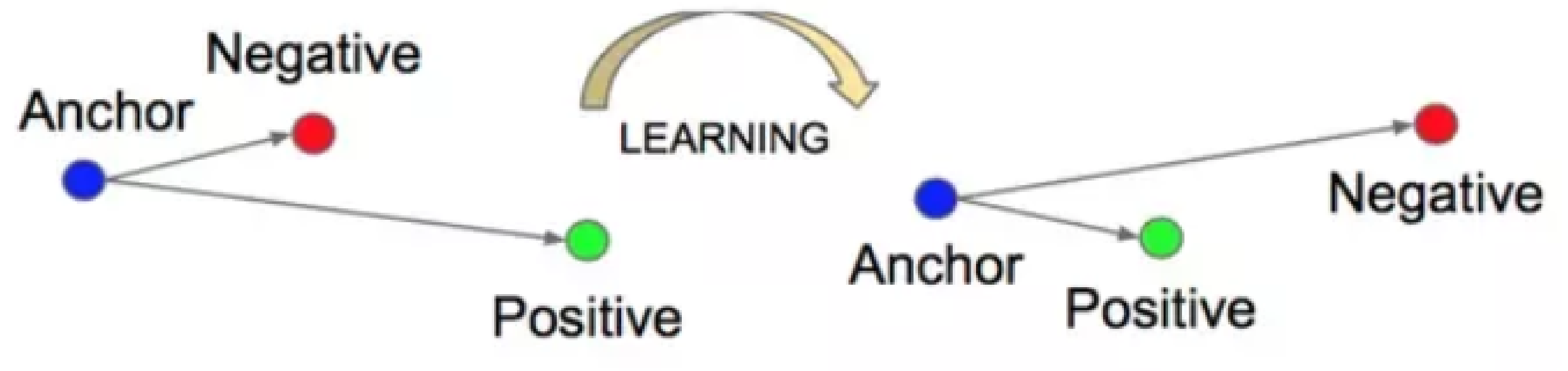
- ➢ If we see d(A,P)=0.5 and d(A,N) is 0.51, here it wont be satisfy the condition , that means this is not good enough to say that they are different we want d(A,N) is much bigger
- ➢ Assume that margin is set to 0.2, algo find the parameters to push the margin d(A,P) tow be lower or d(A, N) to be higher to say they are different
- ➢ By adding this margin we want distance between positive and negative sample should be more not very small

## Triplet Loss function



# CONTRASTIVE LOSS VS TRIPLET LOSS

One advantage of the triplet loss is that it tries to be less "greedy" than the contrastive loss (which considers pairwise examples). This is because the triplet loss takes an anchor example and tries to bring positive examples closer while also pushing away negative example. From the paper by Schroff and colleagues, this looks like this:



The contrastive loss, on the other hand, only considers pairwise examples at a time, so in a sense it is more "greedy." The triplet loss is still too greedy however, since it heavily depends on the selection of the anchor, negative, and positive examples. The magnet loss introduced by Rippel and colleagues tries to mitigate this issue by consider the distribution of positive and negative examples:

For a given triplet of Reference point (A) and a point similar (P) and dissimilar (N) to it:

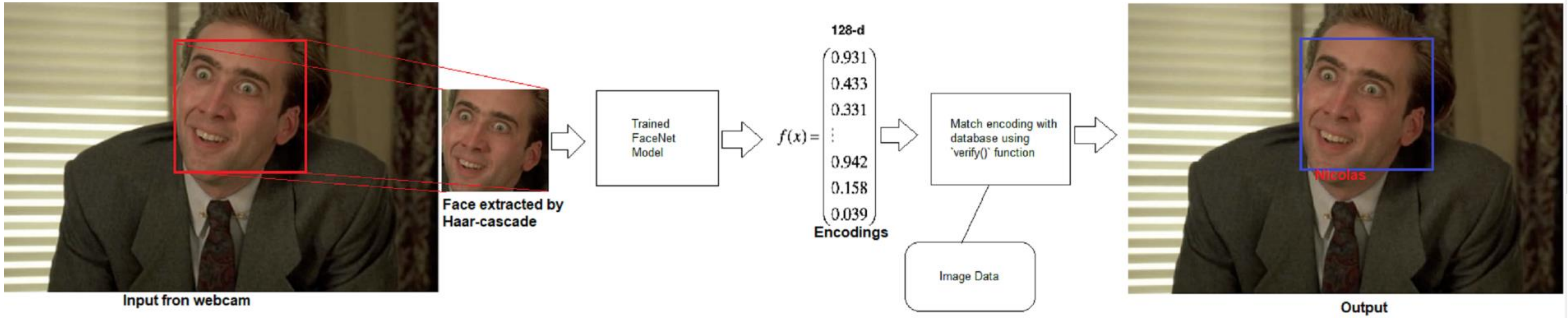
- ➢ Triplet Loss maximizes the difference between A-N & A-P distance (with margin).
- ➢ But Contrastive loss looks to maximize A-N and minimize A-P separately (again with margins).

## Using trained model with webcam for real time Face Recognition:

55

Run `webcamFaceRecoMulti.py` to recognize faces using webcam. The pipeline that I have built is very simple. I used openCV to read webcam feed. Then the webcam feed is passed through another openCV program named ***Haar-Cascade.*** Haar-Cascade will detect all the faces and it will provide the co-ordinates of the bounding box of all the faces.

The detected faces are then passed through our trained FaceNet model which will provide us face encoding. The face encoding is then compared with the encoding of the persons available in our database. Verification is done using `verify(...)` function.

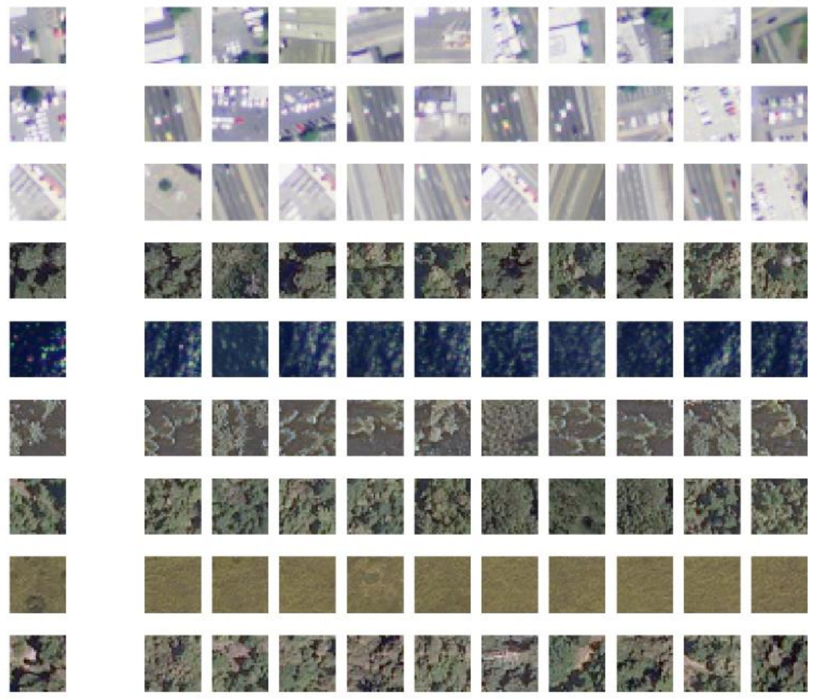


Pipeline

## MODEL LEARNING:

- The model not only learned **to formulate clusters for different classes at the same time, but it's also successful in projecting similar-looking images into their neighborhood region**.
- In the case of **classification architecture, the model tries to learn a decision boundary between a pair of classes**, but the model doesn't take care of the integrity between similar and dissimilar images within a class.

Following is a snapshot of how the model is performing. I have randomly chosen few query images from the corpus of test images. And for each query image, plotted top 10 most probable images which are similar in terms of cosine similarity on high dimensional vector space representation

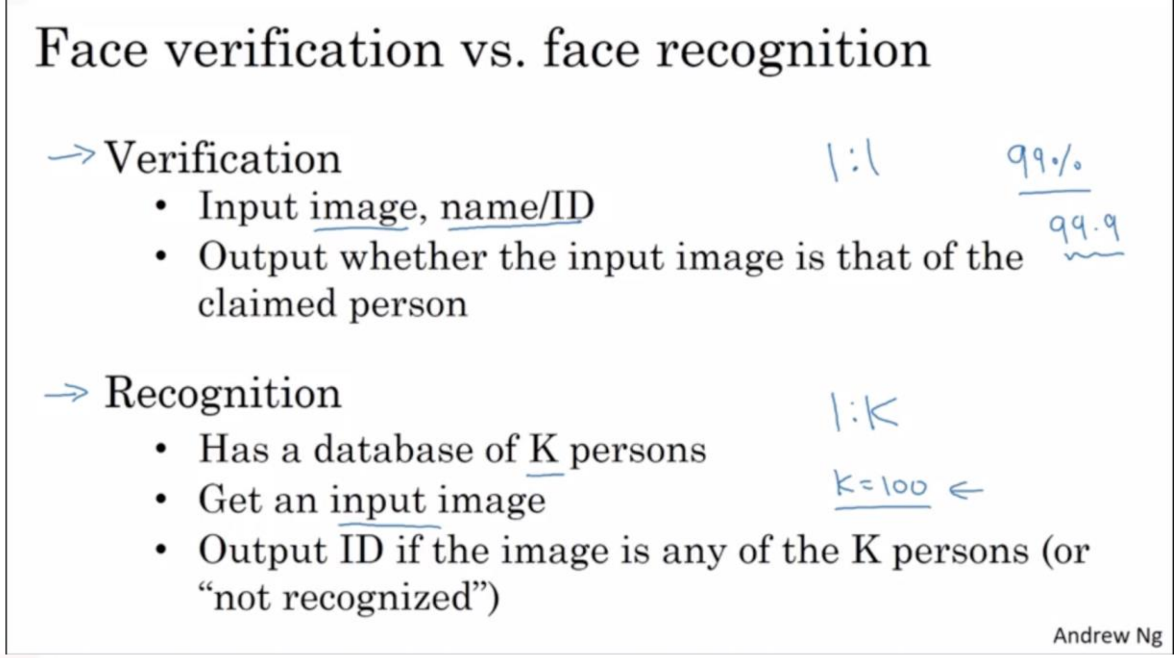


Triplet Loss architecture helps us to solve several problems having a very high number of classes. Let's say you want to build a Face recognition system, where you have a database of 1 million human faces, pre-compute D dimensional vectors for each face. Now, given a human face image [as test image] compute the cosine similarity with all 1 million pre-computed vectors and whatever image has the highest similarity will be the selected candidate.

## APPENDIX

- Face Authentication/Verification (1:1 matching)

- Face Identification/Recognition (1:N matching)



## Face verification vs. face recognition

→ Verification 1:1 99%
- Input image, name/ID
- Output whether the input image is that of the claimed person 99.9

→ Recognition 1:K
- Has a database of K persons K=100 ←
- Get an input image
- Output ID if the image is any of the K persons (or "not recognized")

Andrew Ng