# Face Classification and Verification

HW2P2 Boot Camp

Shayeree Sarkar & Shriti Priya

# Today we will talk about

**Problem Statement for Part 2**

Classification and Verification

Transfer Learning

**Model Architectures**

**Types of Losses**

**Dataset and custom Dataloader for PyTorch(in the Notebook)**

## Problem Statement

- Face Classification:
  - Classifying the person(ID) based on the image of the person’s face
- Face Verification:
  - How would you use the same network you trained for Classification to do face verification, Ideas??
  - You identify the most important features in the image which capture the identity of the face
  - The extracted features will be represented by a fixed length vector, known as an embedding
  - In order to do verification, we need to identify if a given embedding is similar to a reference embedding of that person using a distance metric like the Cosine Distance

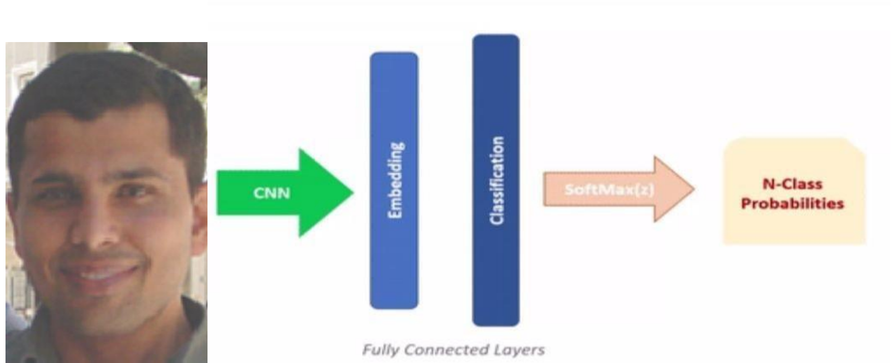
## Classification vs Verification

**Classification**

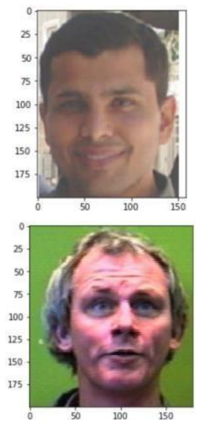
- An N way classification task, predicting from a fixed set of possible output classes

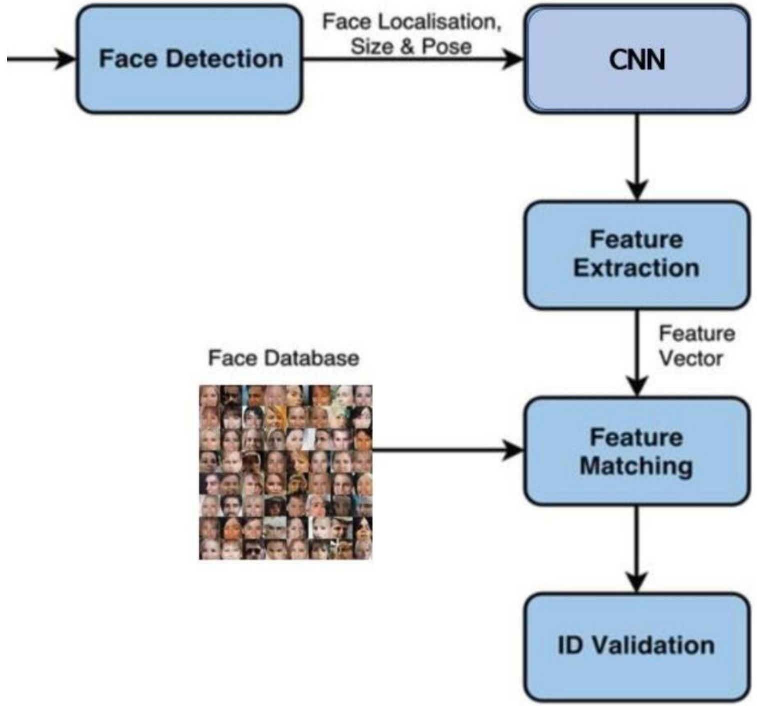
**Verification**

- It is a matching operation, where you match the given sample to the closest sample from a
- reference of N other samples
- Can also be a 1 to 1 task, where we want to verify if the two embeddings are similar (belong to the same class)
- N way classification using cross entropy loss



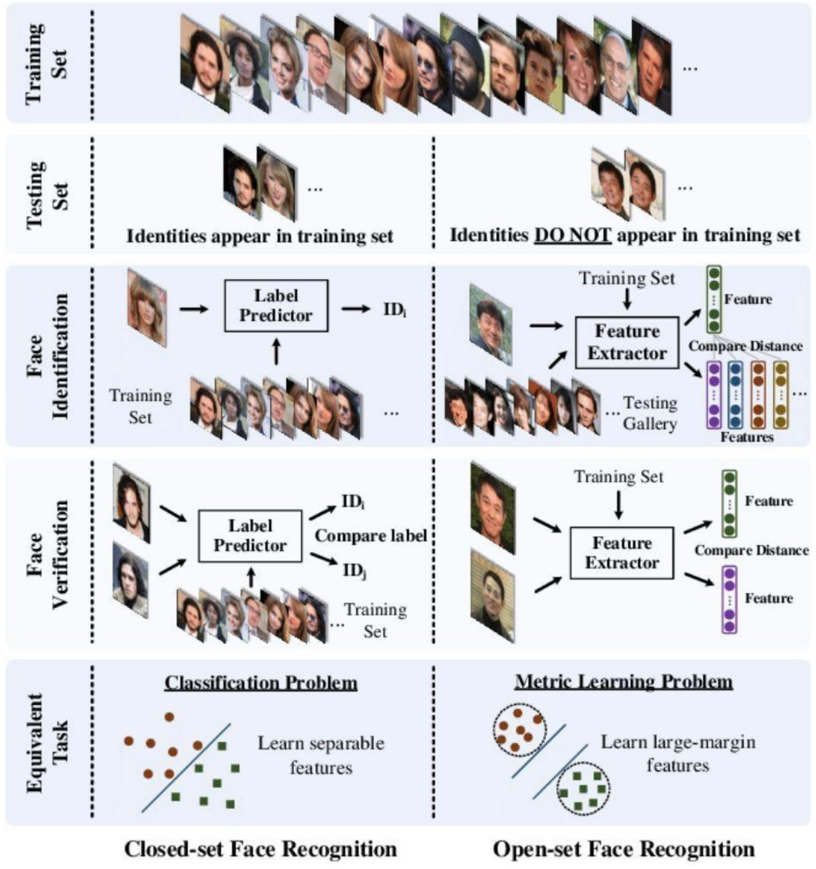
## Verification





# Open set vs Closed Set

Classification versus Verification & Open versus Closed set for the facial recognition problem.



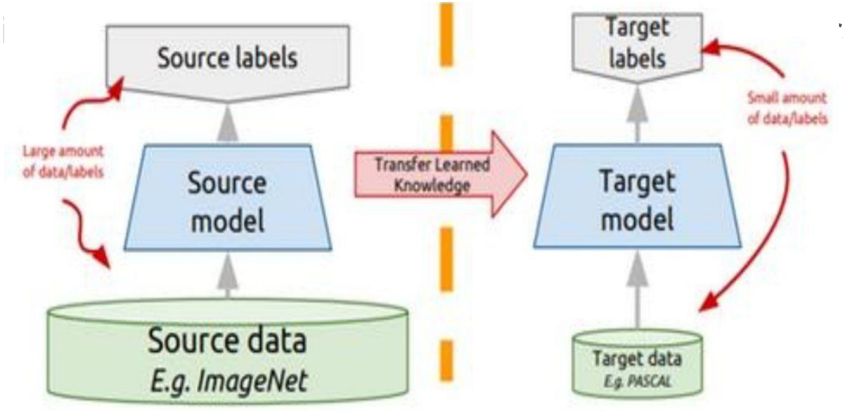
- **CLASSIFICATION :**
  - The is called a “**Closed-set**” task.
  - This is basically a straightforward **classification problem**, where we need to map the input face image to one of the N classes (people).
  - You can’t add new people and you can’t exclude existing onesThe model here is trying to learn **separable features** in this case — i.e. features, that would allow to assign a label from a predefined set to a given image.
  - **The model is trying to find a hyperplane, a rule that separates given classes in space.**

- **VERIFICATION :**
  - This is called an “Open-set” task.
  - This one means that we do indeed have some predefined set of people for training, but the model can be applied to any unseen data and it **should generalize**.
  - In this case the model is trying to solve a **metric-learning problem**: to learn some sort of **similarity metric**, and for that it needs to extract **discriminative features** — features that can be used to distinguish between different people on any two (or more) images.
  - The model is trying **not to separate images** with a hyperplane, but rather reorganize the input space**, pull the similar images together** in some form of a cluster while pushing dissimilar images away.

## Transfer Learning

- Transfer Learning is a method where a model developed for a task is reused
  as the starting point for a model on another task
- Two ways to do Transfer Learning
  - Use trained model weights to initialize the network and train the entire network again
  - Freeze the wei

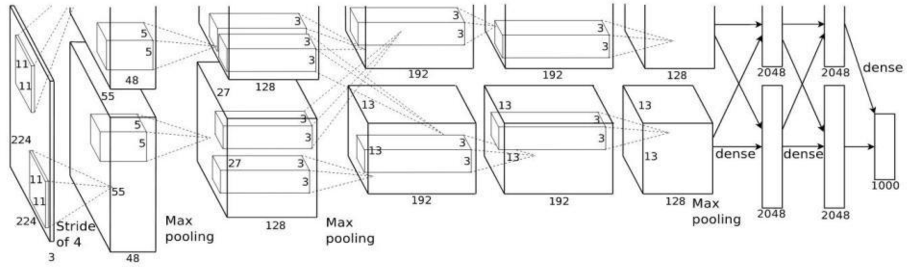
's on the target dataset



## CNN Architectures: AlexNet

Published in 2012 and was the winner of the Imagenet LSVRC-2012

- Uses ReLU as it makes training faster
- Implements dropout between layers
- Uses data augmentation methods (Random Crop, Random Flip in PyTorch)
- The network is

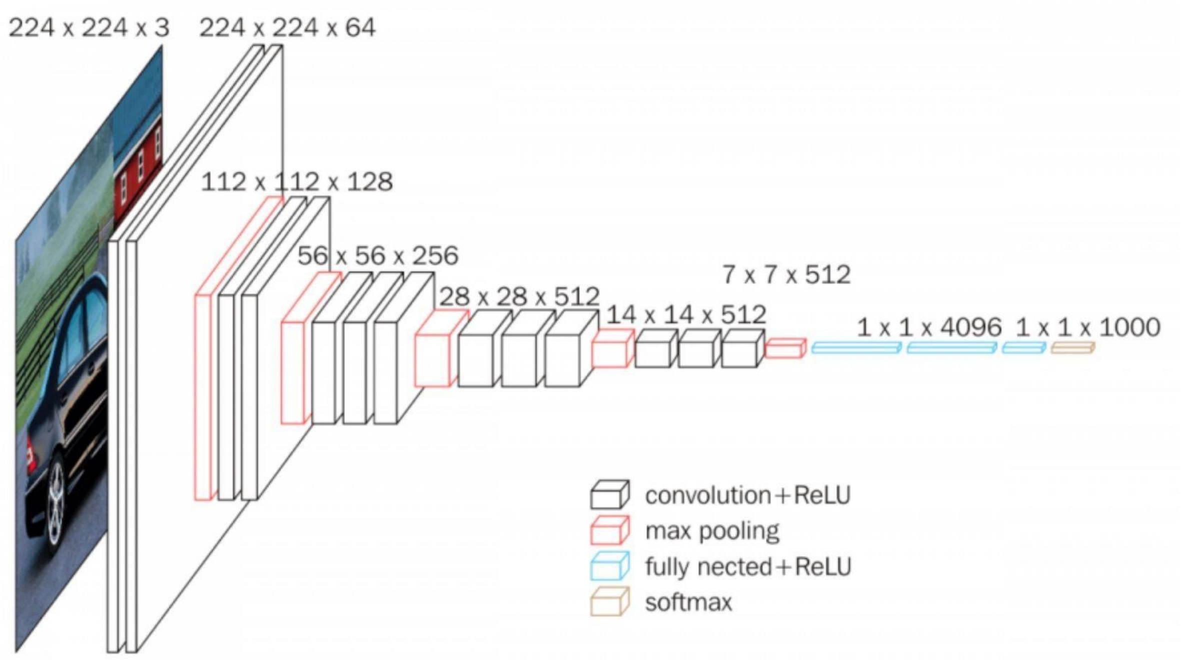


https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## VGG Net

VGG (2014) architecture reduces the size of each layer yet increases the overall depth of it. reinforces the idea that CNNs must be deep in order to work well on visual data.

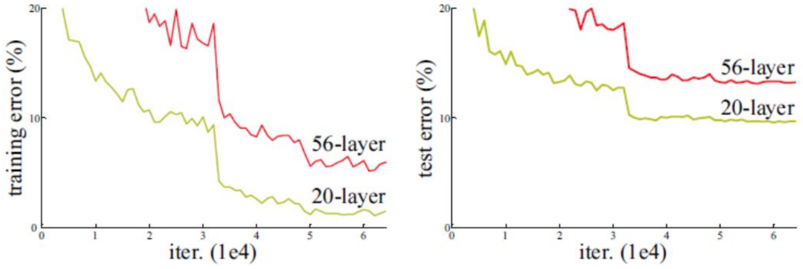
- Conv filters (3*3)
- Max Pool (2*2)



https://arxiv.org/pdf/1409.1556.pdf

# Vanishing / Exploding Gradients

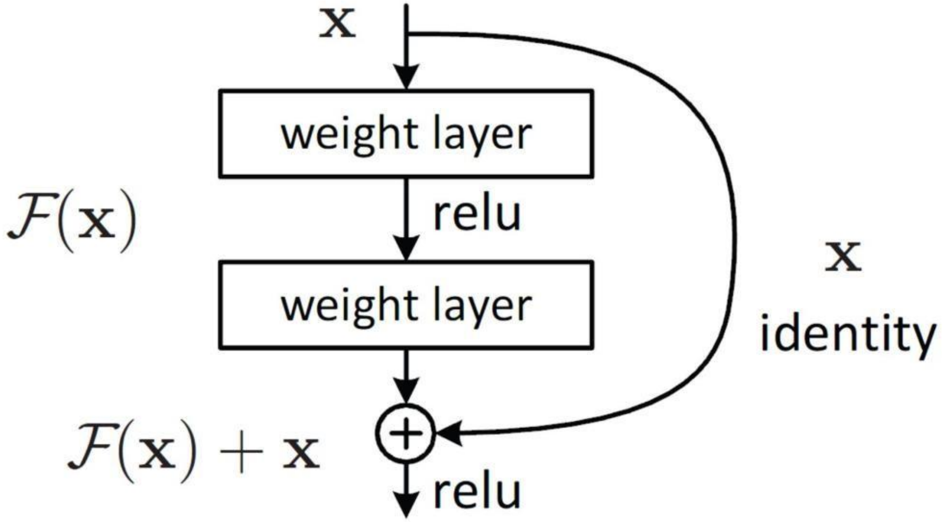
- During backpropagation, when partial derivative of the error function with respect to the current weight in each iteration of training, this has the effect of **multiplying *n* of these small / large numbers to compute gradients** of the "front" layers in an *n*-layer network
- When the network is deep, and multiplying *n* of these small numbers will become zero (vanished).
- When the network is deep, and multiplying *n* of these large numbers will become too large (exploded).
- We expect deeper network will have more accurate prediction. However, below shows an example, **20-layer plain network got lower training error and test error than 56-layer plain network**, a degradation problem occurs due to vanishing gradients.

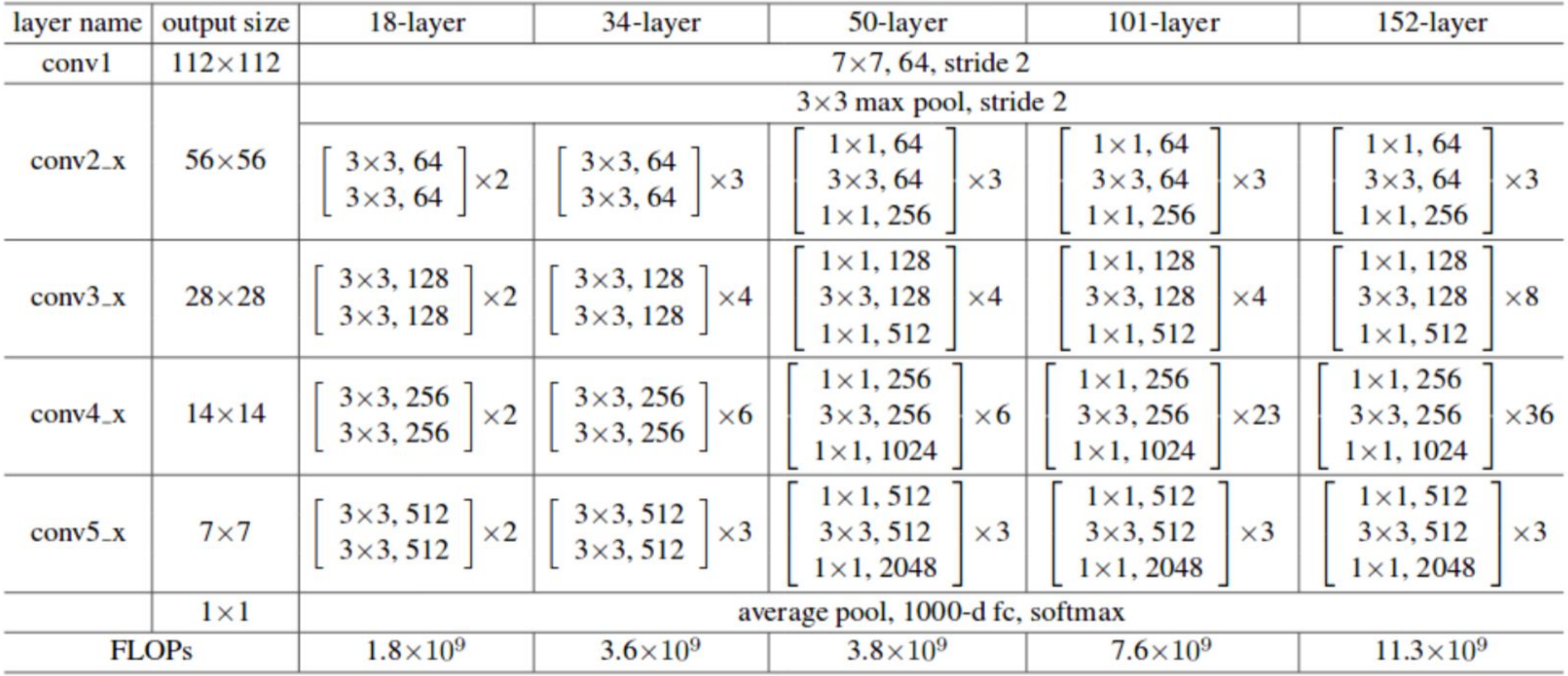


## ResNet

- Introduced in 2015, utilizes bottleneck architectures efficiently and learns them as residual functions
- Easier to optimize and can gain accuracy from increased depth due to skip connections

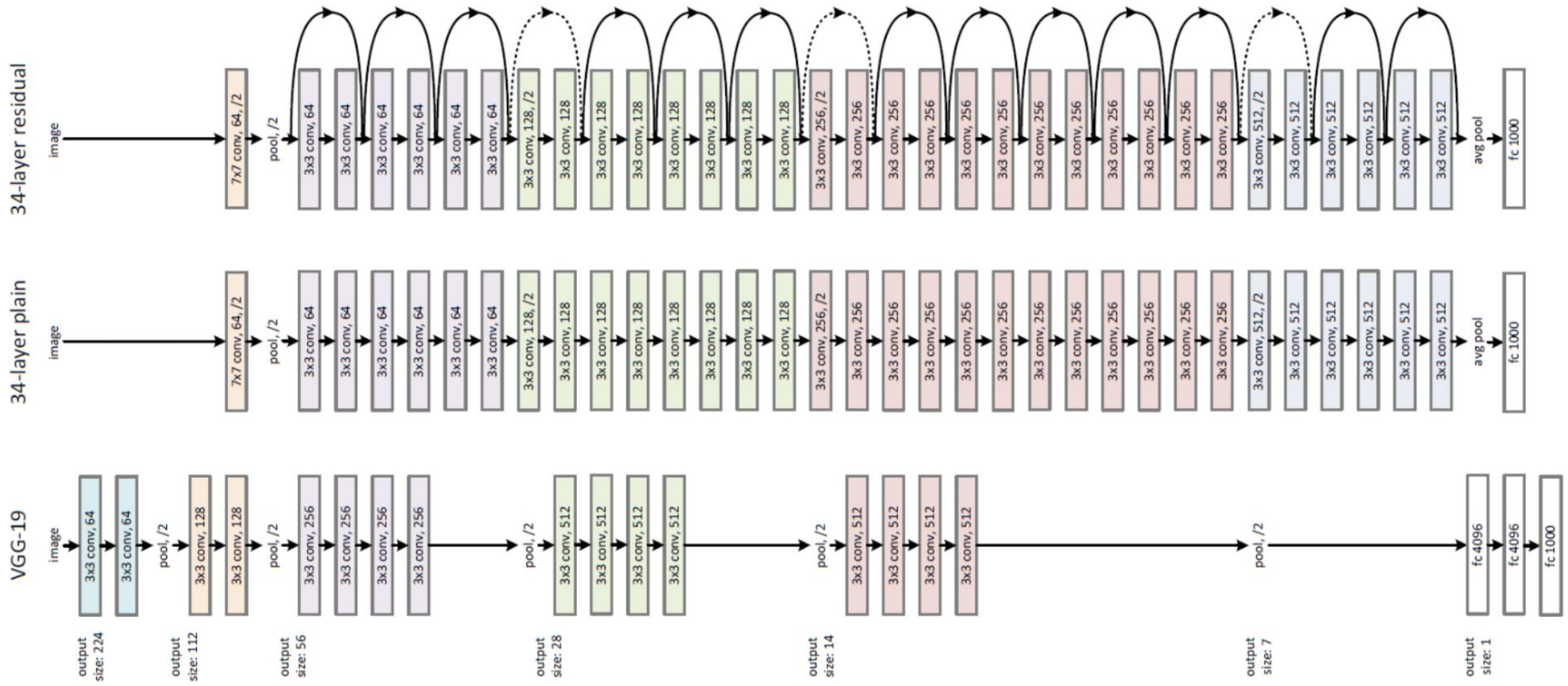
https://arxiv.org/pdf/1512.03385.pdf





| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$ | $\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$ | $\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$ | $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$ | $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$ | $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$ | $\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$ | $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$ | $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$ | $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$ | $\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

ResNet Architectures

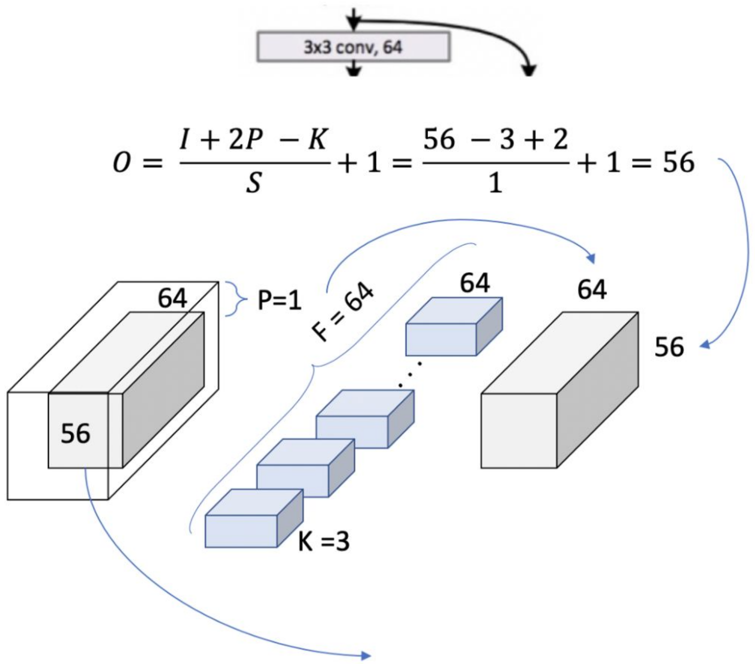


34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom)

## Block 1

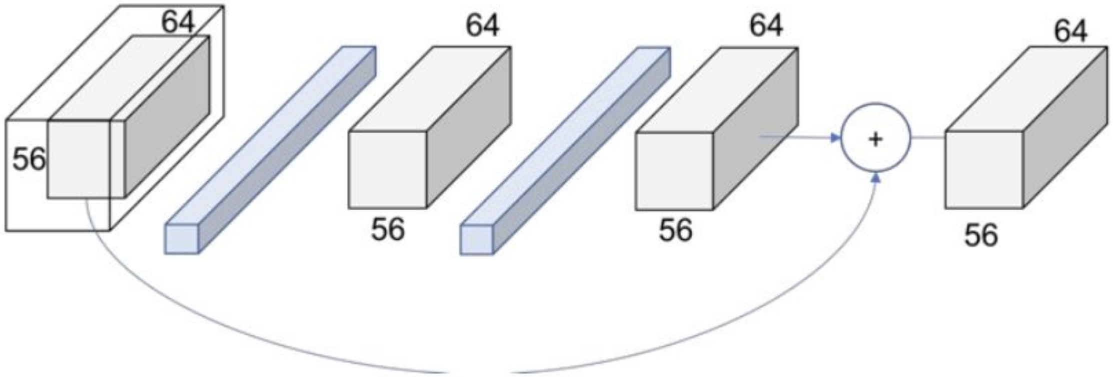
### 1 convolution

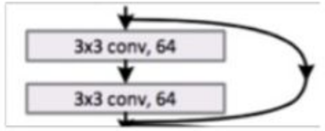
We are replicating the simplified operation for every layer on the paper.



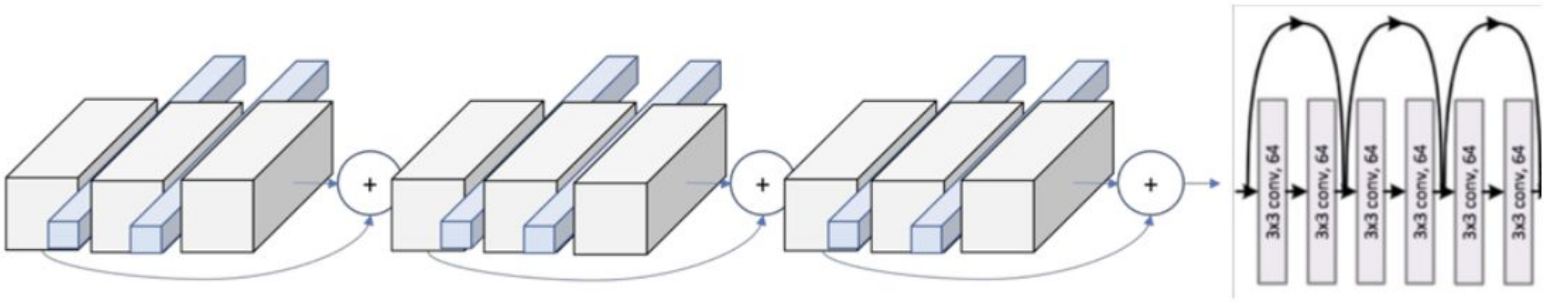
$$O = \frac{I + 2P\ - K}{S} + 1 = \frac{56\ - 3 + 2}{1} + 1 = 56$$

Figure 6. Layer 1, block 1, operation 1

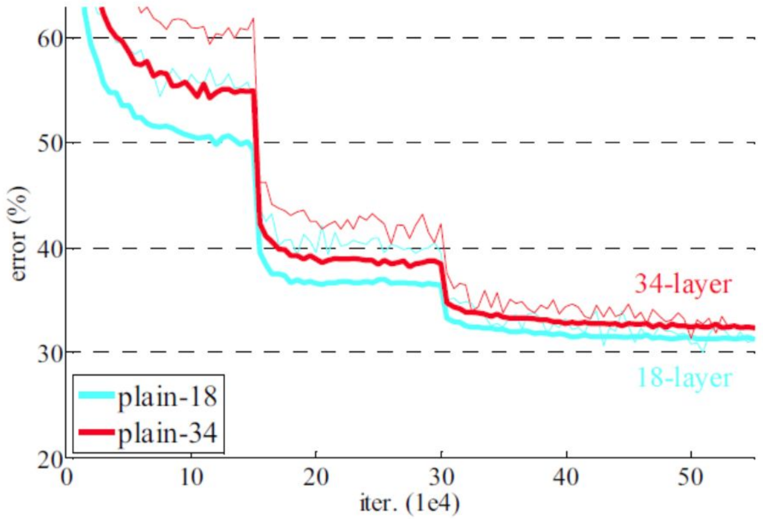

64
56
64
56
64
56
+
64
56

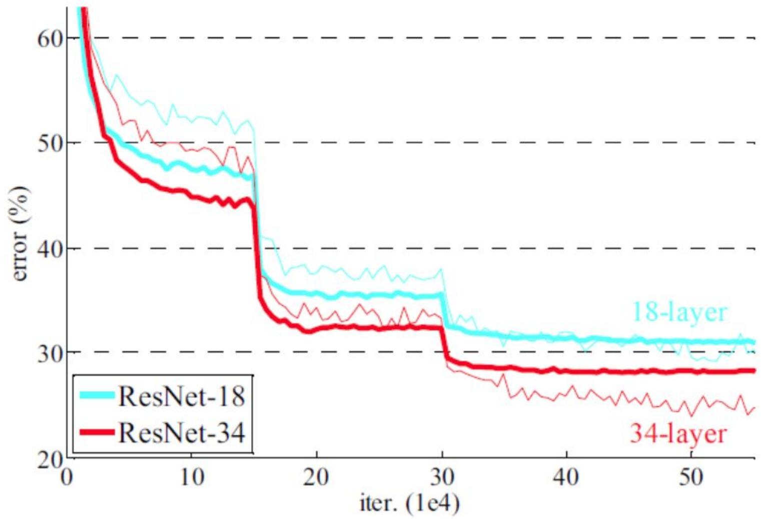

3x3 conv, 64
3x3 conv, 64

We can see how we have the ***[3x3, 64] x 3 times within the layer***.



## 5.1 Plain Network VS ResNet



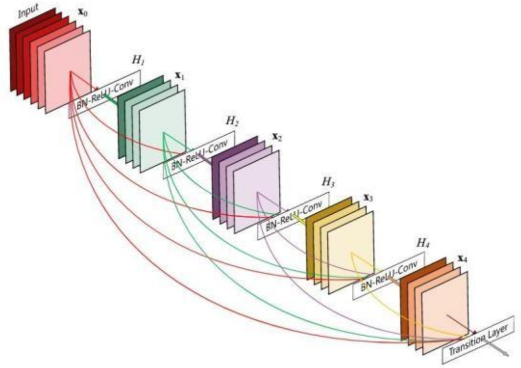


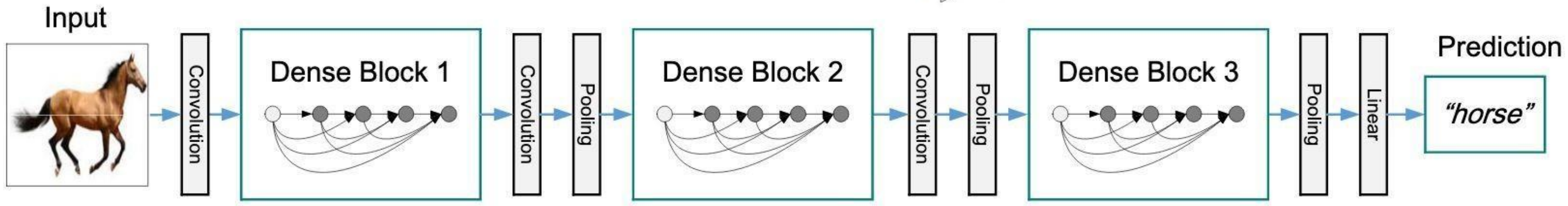
Validation Error: 18-Layer and 34-Layer Plain Network (Left), 18-Layer and 34-Layer ResNet (right)

# Dense Nets

- It includes a stack of dense blocks followed by a transition layers
- Strengthen feature propagation, encourage feature reuse and decrease the

  number of parameters





I
https://arxiv.org/pdf/1608.06993.pd

## Grouped Convolutions

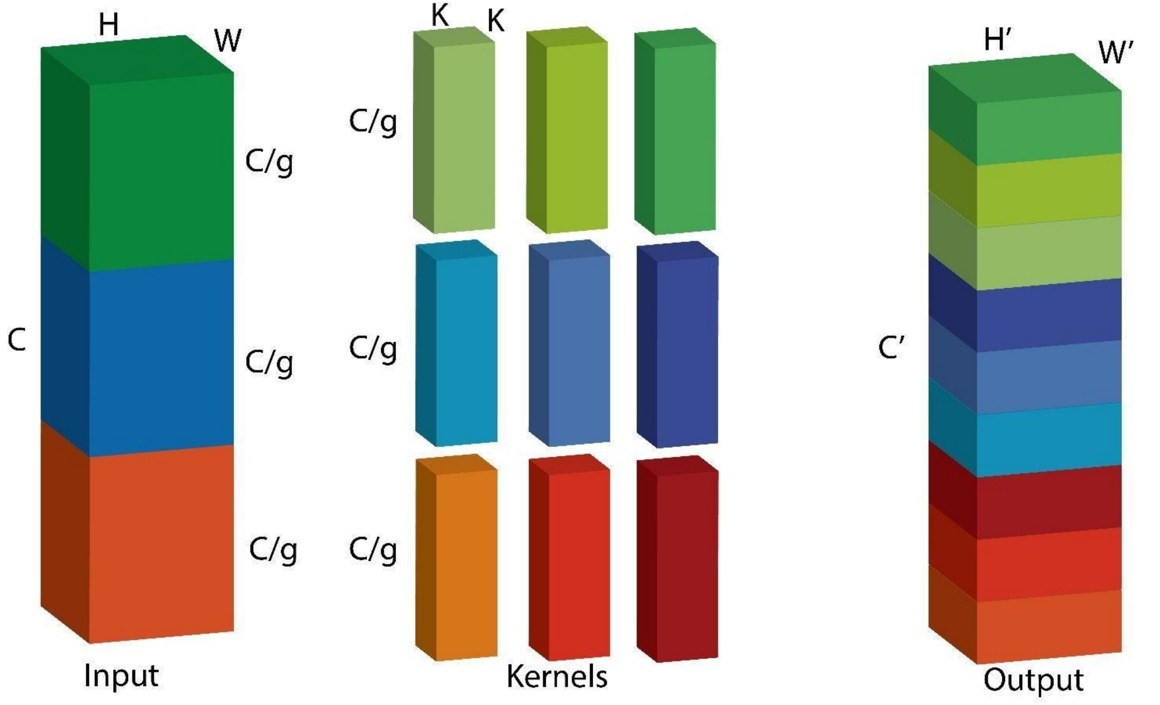
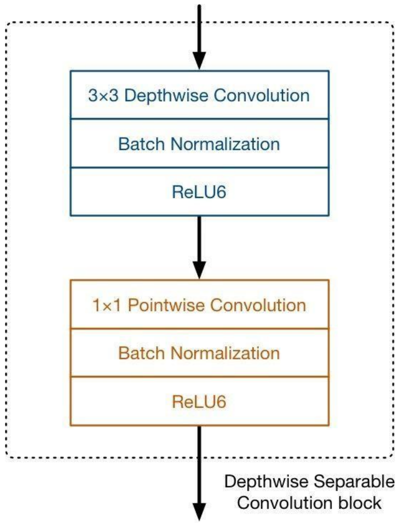


*Figure 0: Grouped Convolutions with $N_i$ = C input Channels, g groups, and $N_o$ = C' output Channels*

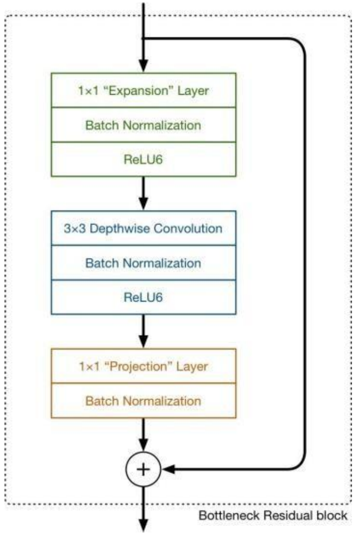
# MobileNet

- Introduced in 2017, intended to run neural networks efficiently on mobile
  devices

  - V1 introduces Depth wise
    separable convolution
  - V2 introduces
    Inverted Residual
    block



Depthwise Separable Convolution block



Bottleneck Residual block

https://arxiv.org/pdf/1704.04861.pdf

# Residual and Inverted Residual block

Bottleneck Architecture

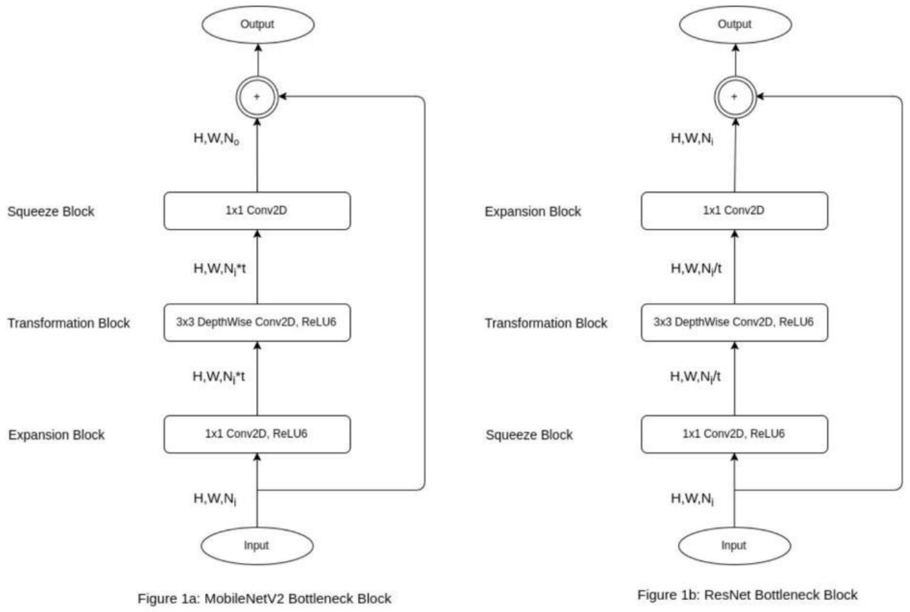
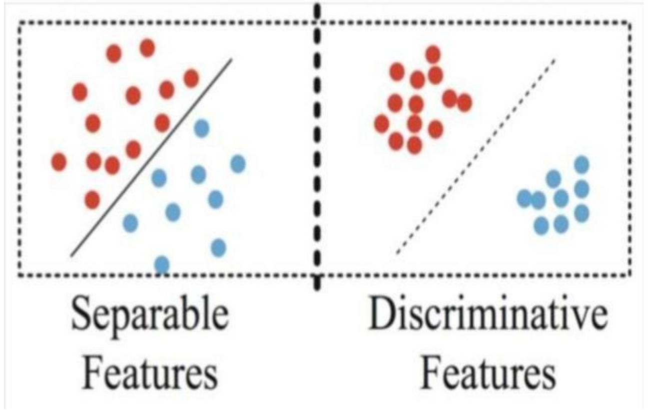


*Figure 1: MobileNetV2 Architecture, BN is used after every conv and is omitted for brevity*

## Other Interesting Papers

- ResNeXt (2016)
  - https://arxiv.org/pdf/1611.05431.pdf
  - Generally a strict improvement to ResNet, but slower. It's like 3 lines of code changed.
- SENet (2017)
  - https://arxiv.org/pdf/1709.01507.pdf
  - Channel-wise attention in CNNs. It's like 20 lines of code.
- EfficientNet (2019)
  - https://arxiv.org/pdf/1905.11946.pdf
  - Optimized model scaling. Probably can hard code this with some effort.
- RegNet (2020)
  - https://arxiv.org/pdf/2003.13678.pdf
  - ResNet with optimized layer sizes. It's probably… 10 lines changed?
- ResNeSt (2020)
  - https://arxiv.org/pdf/2004.08955.pdf
  - ResNeXt on steroids + attention. I (we?) will be really impressed ☺
- NFNet (2021, SOTA)
  - https://arxiv.org/pdf/2102.06171v1.pdf
  - Quite doable actually

## Discriminative Features

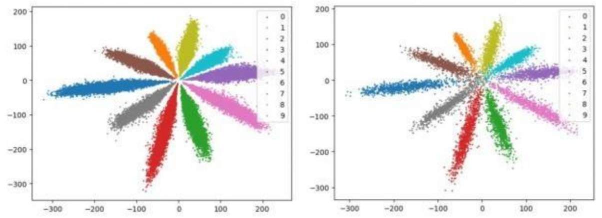
- Classification optimizes learning separable features
- Optimally we wish to learn discriminative features
    - Maximum inter class distance
    - 

# Center Loss

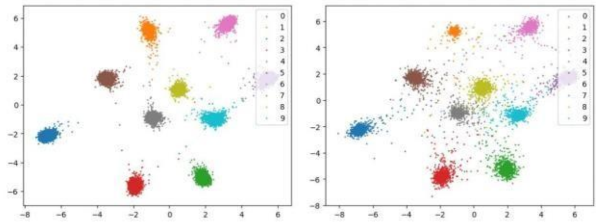
- Tries to minimize the intra class distance by adding a euclidean distance loss term
- If you use this, **YOU MUST USE CENTER LOSS FROM THE BEGINNING OF TRAINING CLASSIFICATION!**
  - Every semester we get around 50 questions about this.

$$
\begin{aligned}
\mathcal{L} &= \mathcal{L}_S + \lambda \mathcal{L}_C \\
&= -\sum_{i=1}^{m} \log \frac{e^{W_{y_i}^T \boldsymbol{x}_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T \boldsymbol{x}_i + b_j}} + \frac{\lambda}{2} \sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{c}_{y_i}\|_2^2
\end{aligned}
$$

Softmax only. Left: training set. Right: test set.



Softmax + center loss. Left: training set. Right: test set.



# Triplet Loss

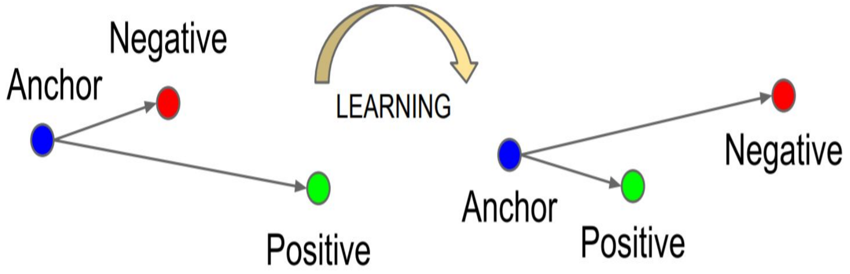
Positive

Anchor

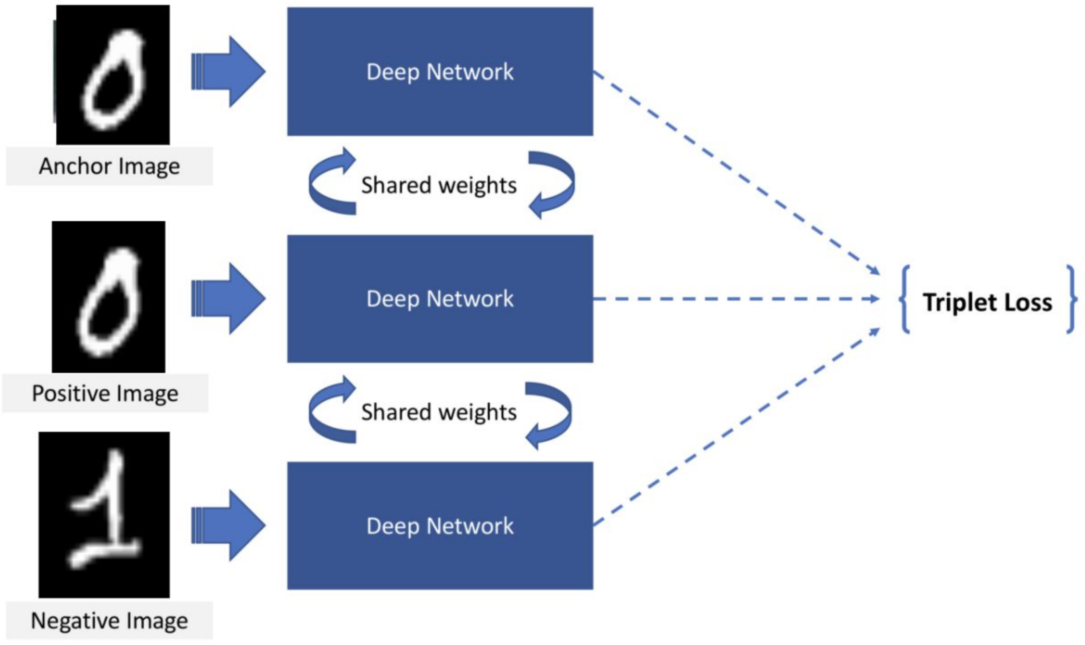
Negative

$$\sum_{i}^{N}\left[\|f(x_i^a)-f(x_i^p)\|_2^2-\|f(x_i^a)-f(x_i^n)\|_2^2+\alpha\right]$$

- Minimizing first term → distance between Anchor and Positive image.
- Maximizing second term → distance between Anchor and Negative image.



# Training with Triplet Loss



## Siamese Network

This network does not classify the images into certain categories or labels, rather it only finds out the distance between any two given images

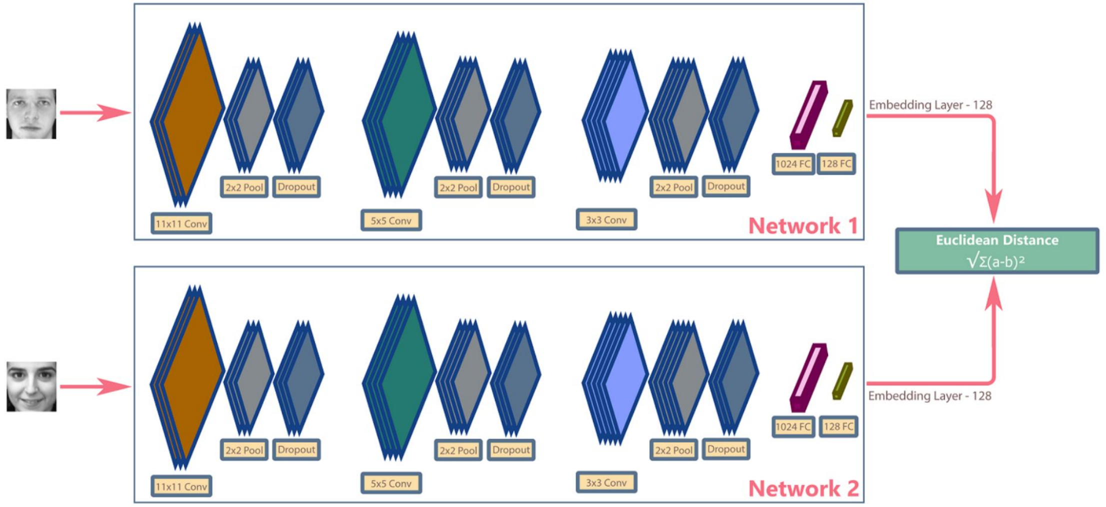


Fig. Architecture of a Siamese Network.

# Contrastive Loss

- Contrastive loss is a **metric learning** loss, which operates on the data points produced by network and their positions relative to each other.
- The model can learn any features regardless of whether similar data points would be located closely to each other or not after the transformation.
- **Y** term here specifies, whether the two given data points ($X_1$ and $X_2$) are similar (**Y**=0) or dissimilar (**Y**=1)
- So **Ls** (loss for similar data points) is just **Dw**, distance between them, if two data points are labeled as similar, we will minimize the euclidean distance between them.
- **Ld**, (loss for dissimilar data points) on the other hand, needs some explanation. One may think that for two dissimilar data points we just need to **maximize distance between them but with a margin**

$$L(W, Y, \vec{X_1}, \vec{X_2}) = (1 - Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m - D_W)\}^2$$

$$D_W(\vec{X_1}, \vec{X_2}) = \|G_W(\vec{X_1}) - G_W(\vec{X_2})\|_2$$

# Understanding the Contrastive Loss Function

- In the first figure, we would naturally like to pull black dots closer to the blue dots and push white dots farther away from it.
  - Specifically, we would like to minimize the intra-class distances **(blue arrows)** and maximize the inter-class distances (**red arrows)**
- In the second figure, what we would like to achieve is to make sure that for each class/group of similar points (in case of Face Recognition task it would be all the photos of the same person) the maximum intra-class distance is smaller than the minimum
  - This means is that if we define some radius/margin m, all the black dots should fall inside of this margin, and all the white dots — outside
  - This way we would be able to use a nearest neighbour algorithm for new data — if a new data point lies within **m distance** from other, they are similar/belong to same group/class. inter-class distance.
  - If ***Dw*** is ≥ ***m***, the {***m - Dw***} expression is negative and the whole right part of the loss function is thus 0 due to ***max()*** operation — and the gradient is also 0, i.e. we don’t force the dissimilar points farther away than necessary.

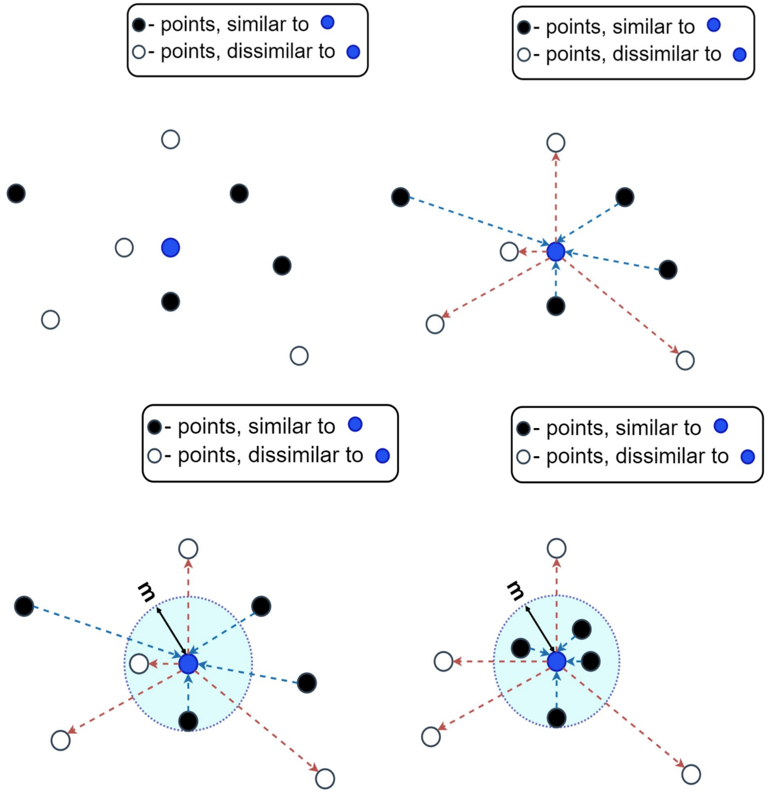
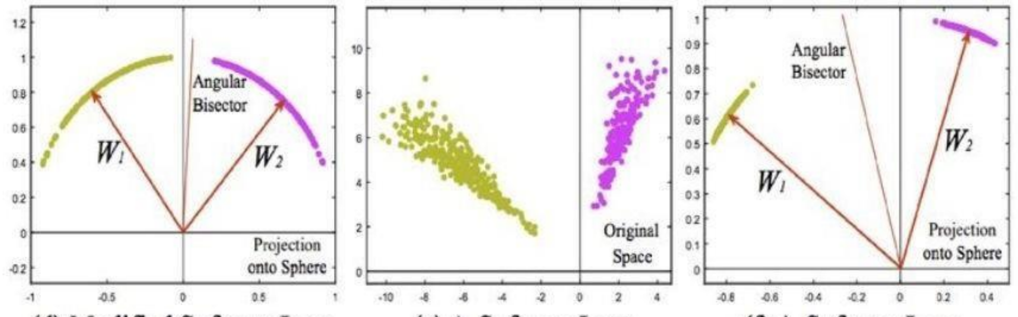


Figure 5 — What we would like the algorithm to do. Notice how the white dots that were outside weren't moved farther away from the margin.

# Other types of Losses

- Contrastive Loss (maintain margin between classes)
- Pair Wise Loss (separate distributions of similarity scores)
- Angular Softmax Loss



(d) Modified Softmax Loss (e) A-Softmax Loss (f) A-Softmax Loss

## References

- https://arxiv.org/pdf/1512.03385.pdf
- https://arxiv.org/pdf/1608.06993v3.pdf
- https://arxiv.org/pdf/1409.1556.pdf
- https://arxiv.org/pdf/1704.08063.pdf
- https://arxiv.org/pdf/1503.03832v3.pdf
- http://ydwen.github.io/papers/WenECCV16.pdf
- https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf
- https://towardsdatascience.com/densenet-2810936aeebb
- http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf
- https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convoluti
  onal-neural-networks.pdf