11-485/685/785, Fall 2024

HW2P2: Bootcamp

TAs: Syed Abdul Hannan, Hao Chen

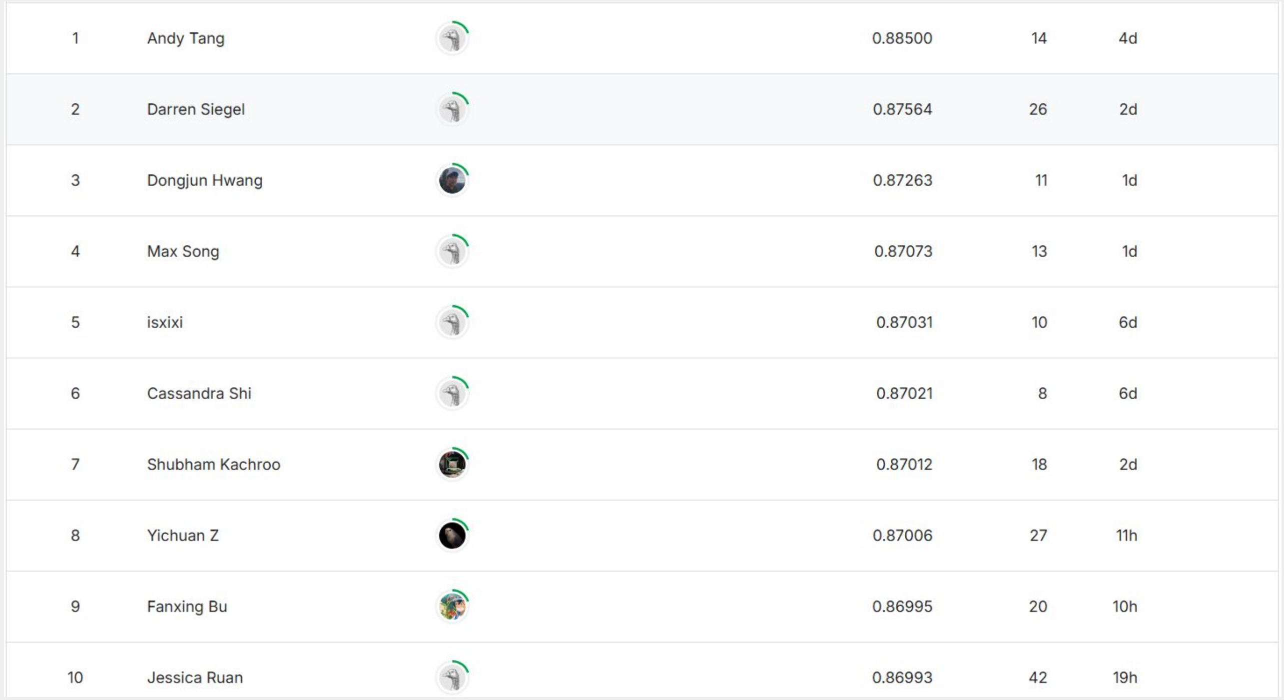
Reflection of HW1P2

1. **What was challenging but you figured that out?**
2. **Which strategies / resources helped you the most?**
3. **What would you do differently if you could start over?**

Reflection of HW1P2

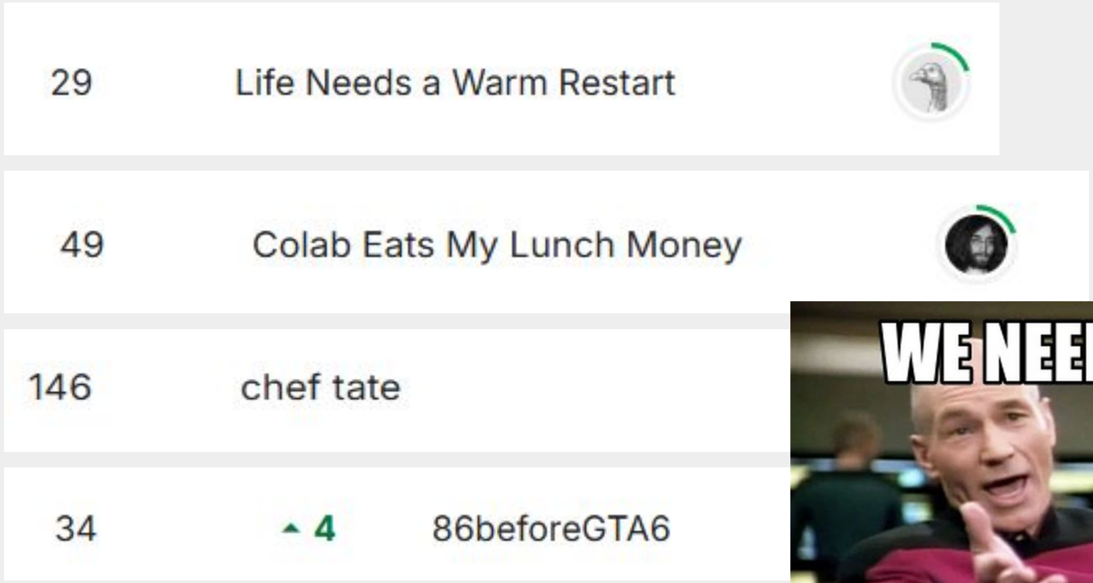
1. **What was challenging but you figured that out? → Great job**
2. **Which strategies / resources helped you the most? → Things to keep for HW2P2**
3. **What would you do differently if you could start over? → Things to change for HW2P2**

Kaggle shoutouts - top 10 submissions!



1	Andy Tang	0.88500	14	4d
2	Darren Siegel	0.87564	26	2d
3	Dongjun Hwang	0.87263	11	1d
4	Max Song	0.87073	13	1d
5	isxixi	0.87031	10	6d
6	Cassandra Shi	0.87021	8	6d
7	Shubham Kachroo	0.87012	18	2d
8	Yichuan Z	0.87006	27	11h
9	Fanxing Bu	0.86995	20	10h
10	Jessica Ruan	0.86993	42	19h

Kaggle shoutouts - most funny names!



29		Life Needs a Warm Restart
49		Colab Eats My Lunch Money
146		chef tate
34	▲ 4	86beforeGTA6

Overview of HW2P2

- HW2P2 is significantly more time consuming than HW1P2
- Models will be harder to develop, train, and converge
- Models must be written yourself and trained from scratch
- Use what you learned from HW1P2 💪
- Please start early!

Goal of this HW Bootcamp:

Help you to get started with HW2P2 🚀

Agenda

1. **Problem statement**
2. **Workflow**
    a. Data loading and preprocessing
    b. Building a model
    c. Training, monitoring, testing, and Kaggle submission
    d. Different loss functions for the task
3. **FAQ**

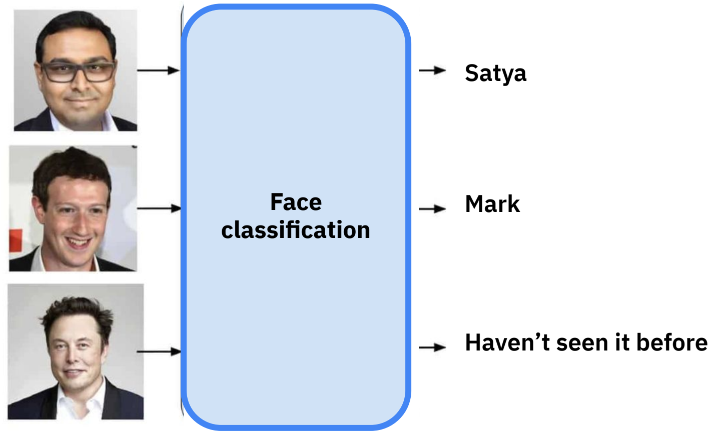
Problem statement

HW2P2 objective

Two tasks, one model, one submission

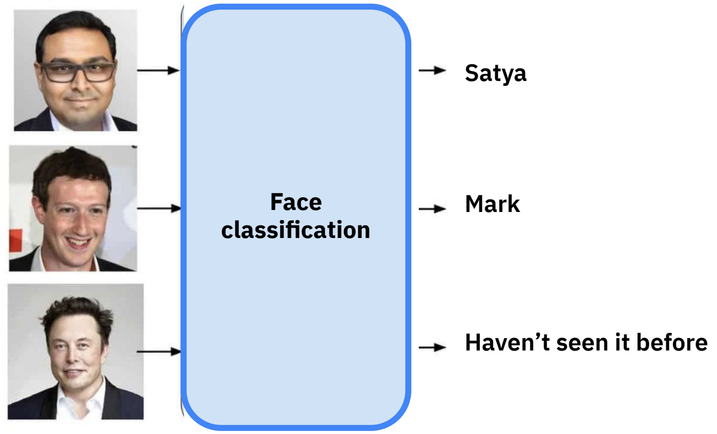
- Face verification
- Face classification

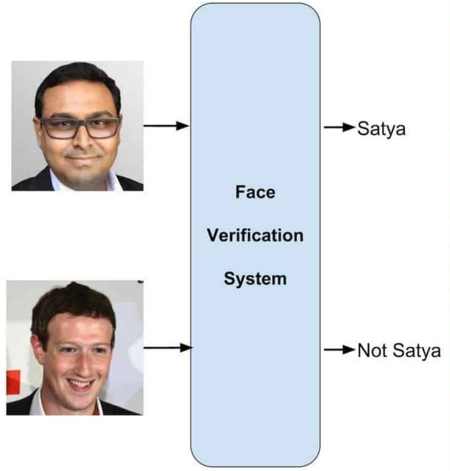
Face classification VS verification



Face classification (aka identification, recognition)
Given an image, who is this person? Person X, Y, or Z
Example: can be used in applications like attendance system

Face classification VS verification





Face classification (aka identification, recognition)	Face verification
Given an image, who is this person? Person X, Y, or Z	Is the pair of Images given to you of the same person?
Example: can be used in applications like attendance system	Example: you can unlock your smartphone using your face, but others can't

Workflow

HW Parts 2 ideal workflow

Step 0: Download the notebook.

Step 1: Complete all #TODOs and ensure your code runs and reaches very low cutoff.

Step 2: Divide the experiments among the study group members to achieve the high cutoff.

HW Parts 2 components

1: Data loading and preprocessing

2: Building a model

3: Training and monitoring

4: Testing and Kaggle submission

5: Different loss functions and model finetuning

1. Data loading and Preprocessing

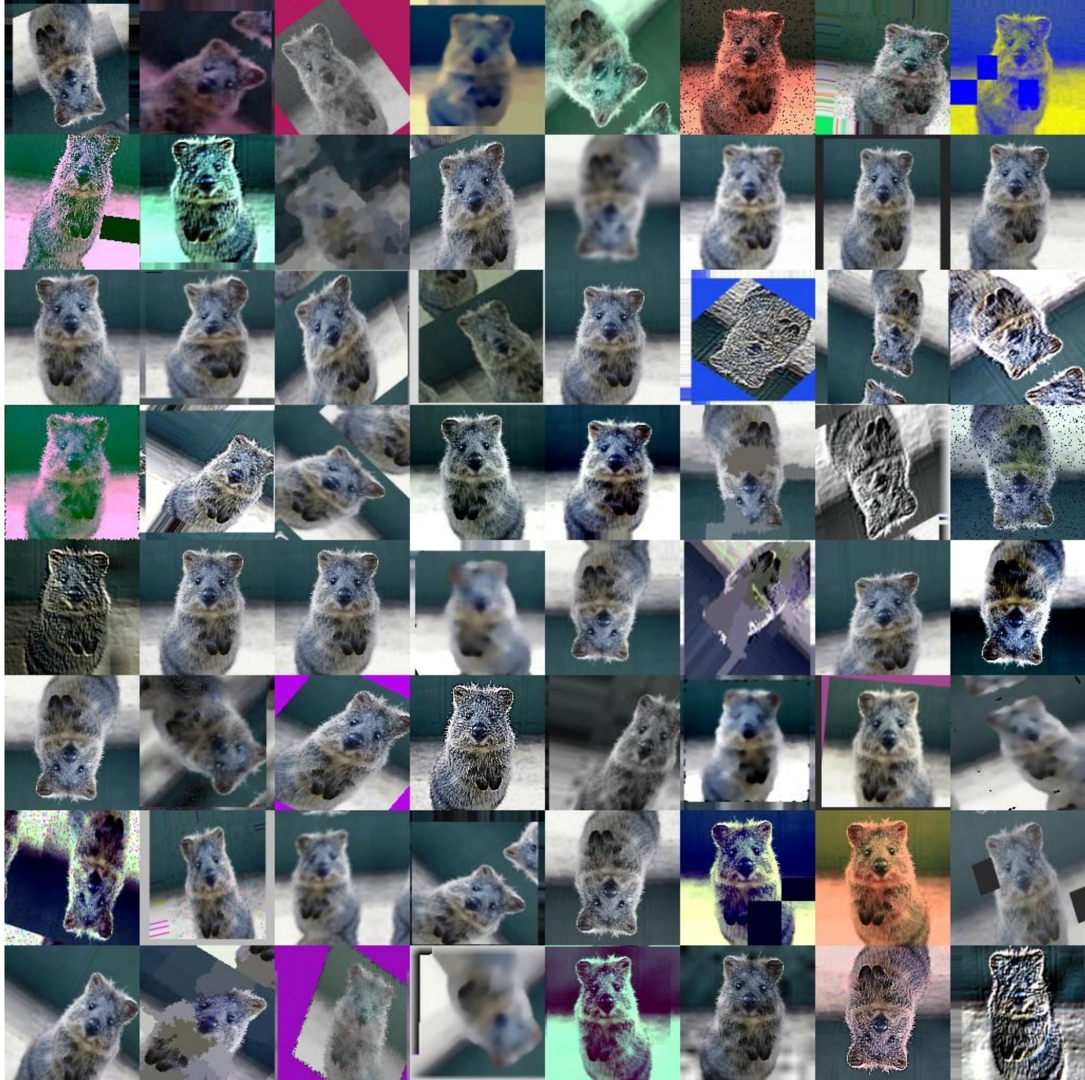
Dataset and DataLoader

1: We will be using the **ImageFolder** class from the torchvision library for classification. The images in subfolders of classification data are arranged in a way that is compatible with this dataset.

2: The ImageFolder class will automatically infer the labels and make a dataset object, which we can then pass on to the dataloader.

3: We will write our own dataset which returns a pair of images for verification.

Image Augmentations



Why Do We Need Augmentations

1. **Emulating More Data:** Transformations increase the perceived dataset size by applying various alterations to input images, leading to improved training and generalization.

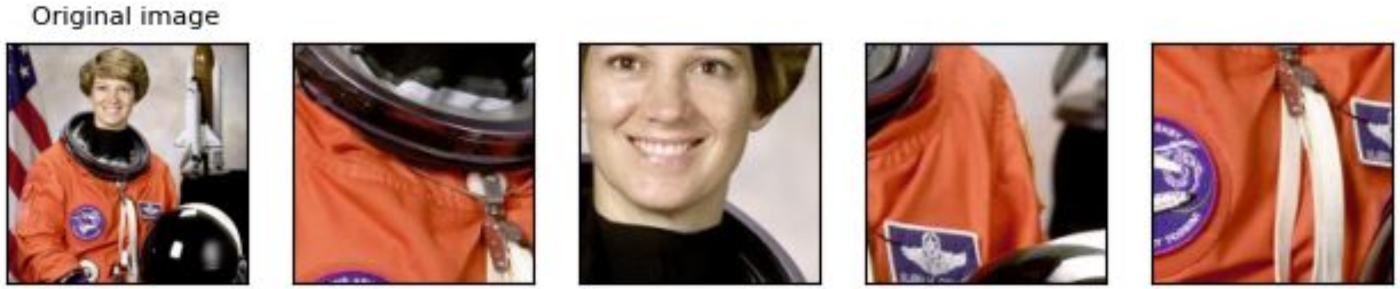
2. **Preventing Overfitting:** Transformations expose the model to ’new’ versions of images in every epoch, reducing the memorization of specific images and promoting the learning of robust features

3. **Invariance:** Training the model on transformed images teaches it to recognize objects independently of their orientation or position

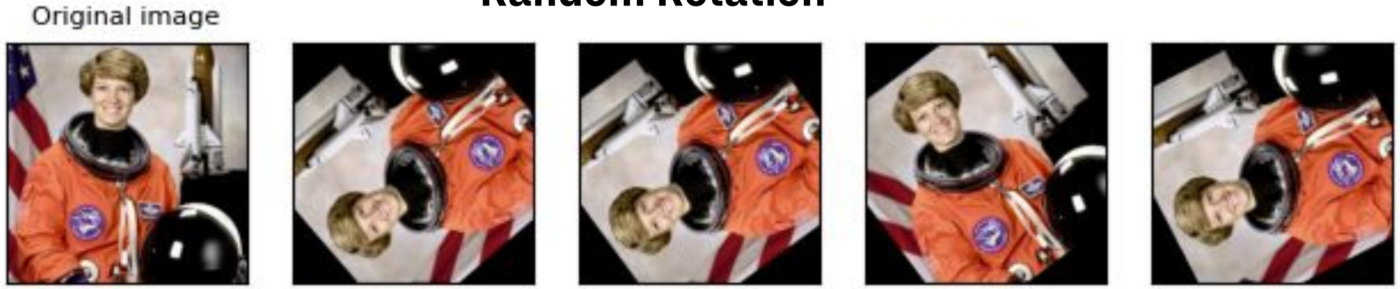
4. **Better Generalization:** Transformations diversify the training set, exposing the model to a wide variety of examples, which can enhance performance on unseen data

Sample Augmentations

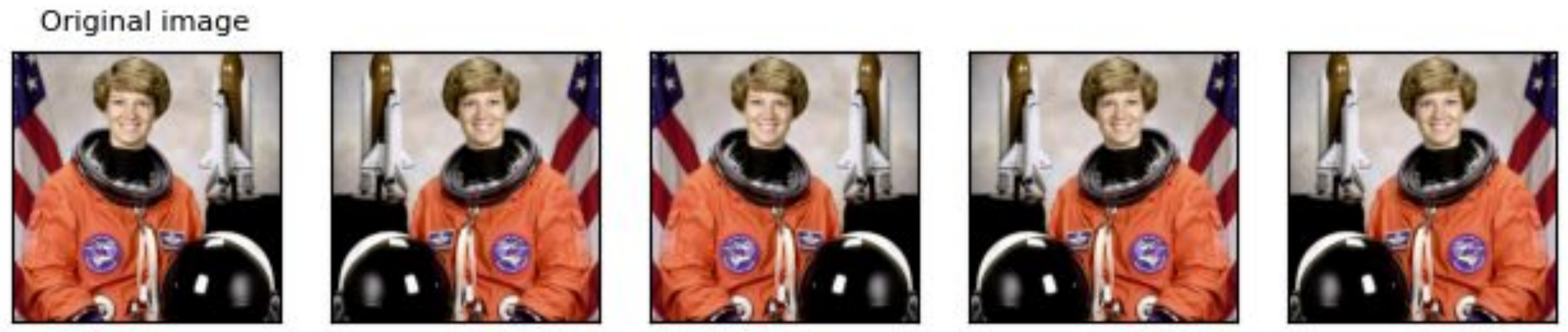
Random Crop



Random Rotation

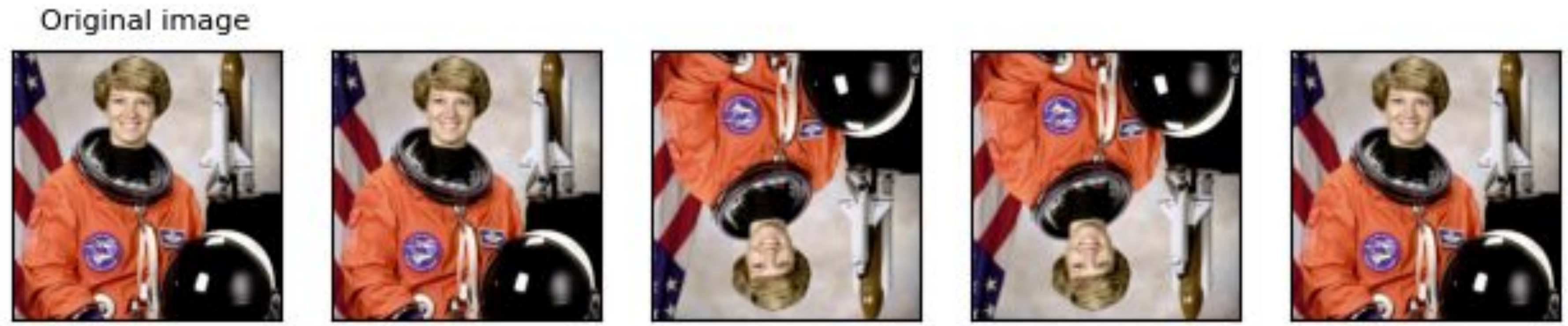


Random Horizontal Flip



Good Transform

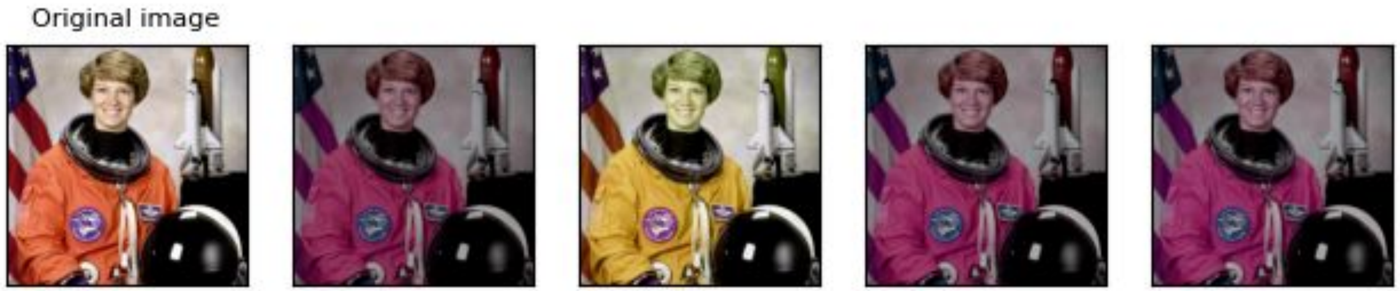
Random Vertical Flip



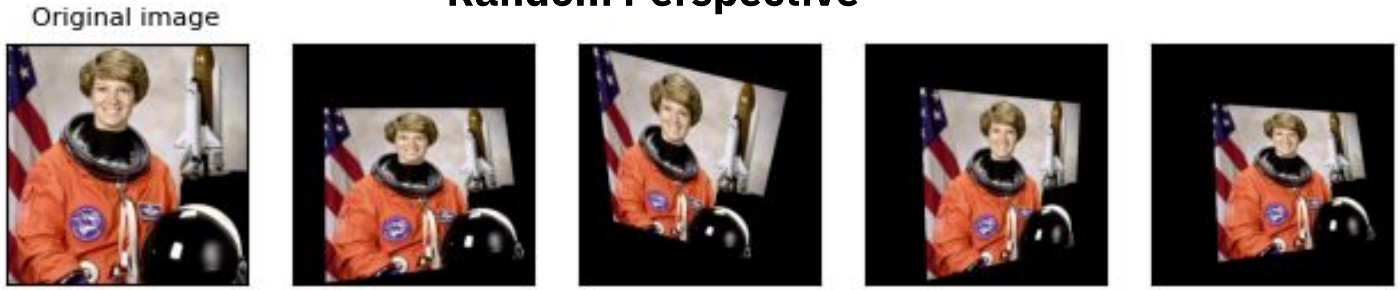
Not So Good Transform

Sample Augmentations

Color Jitter



Random Perspective



Transformation Tips

Consider Normalising the Data:

- Use torchvision.transforms.Normalize() after calculating the mean and standard deviation of each pixel of each image of the dataset over all 3 channels(RGB)

Common Issue:

```
TypeError: Input tensor should be a torch tensor. Got <class 'PIL.Image.Image'>.
```

- Please check the sequencing of your transforms. Read the documentation and verify the kind of input required.

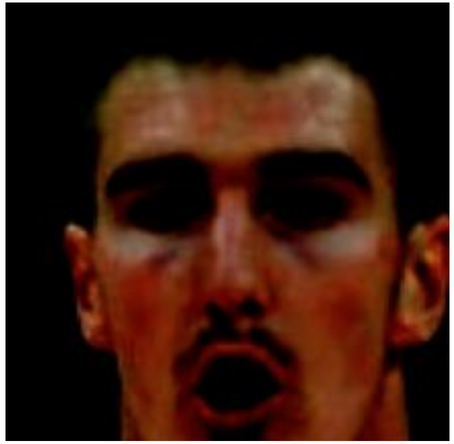
This Is What A Typical Transform Pipeline Could Look Like:

```
transforms = transforms.Compose([
        transforms.Random_____(112),
        transforms.Random_____(),
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
```

Torchvision Transforms Illustrations URL:

https://pytorch.org/vision/0.11/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

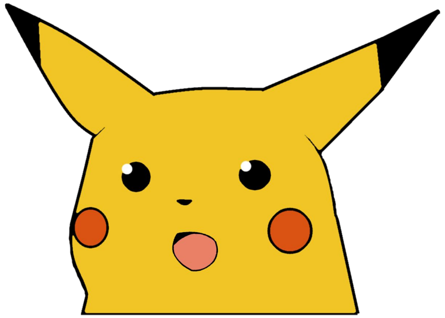
Final Images After Transformation (After applying Augmentations)

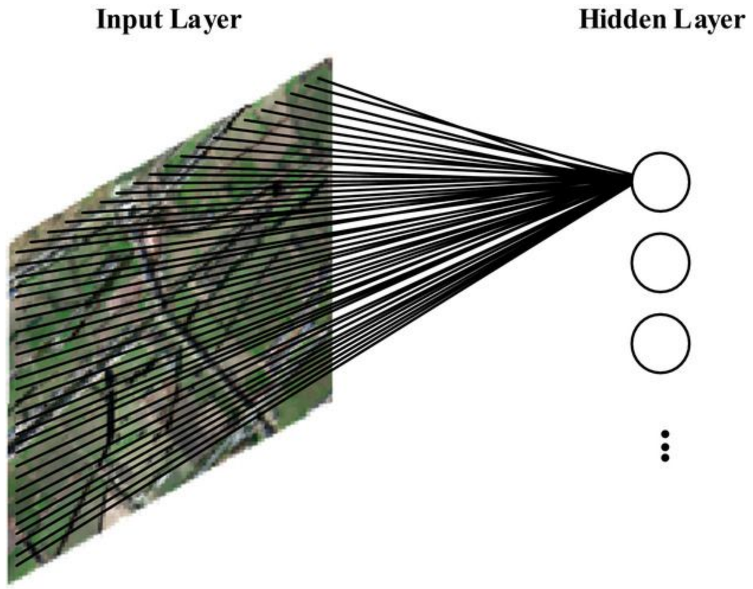


WE MIGHT HAVE A PROBLEM IF IT LOOKS LIKE THIS!!

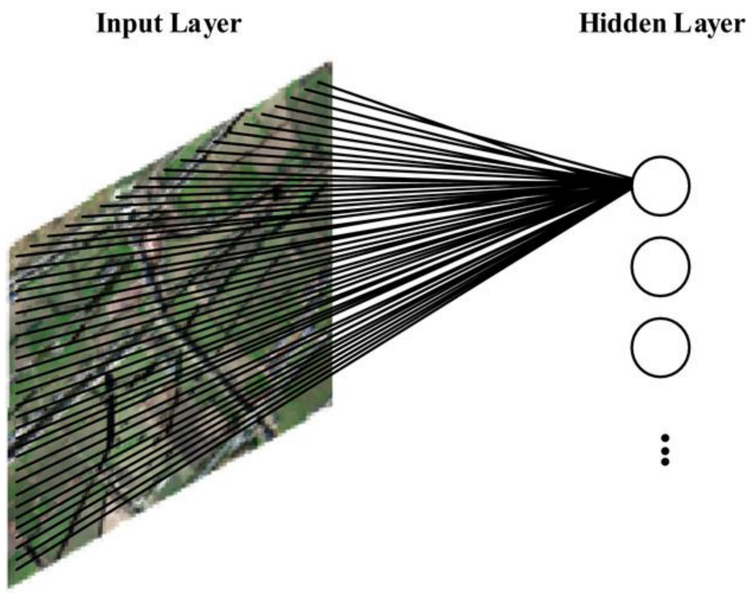
2. Building a model

WHY NOT USE LINEAR LAYERS???





HINT: For a 1000 x 1000 image
With 1 M Hidden Neurons.



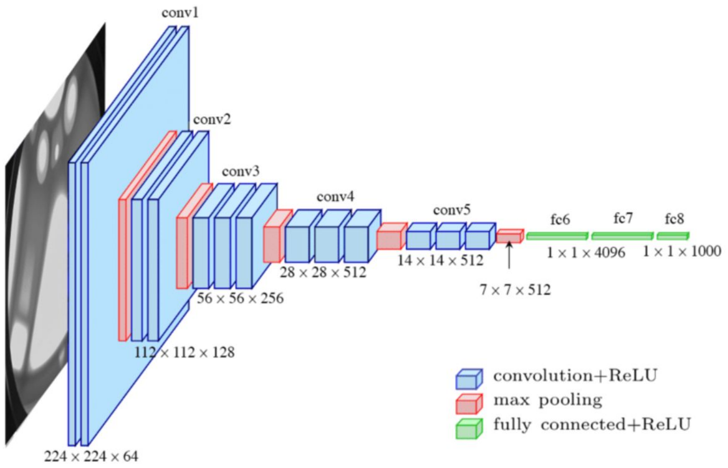
HINT: For a 1000 x 1000 image
With 1 M Hidden Neurons.

That is 10^12 M Trainable
Parameters

Rather use something else...

We Also Lose Spatial Features

OUR HERO: CNNs

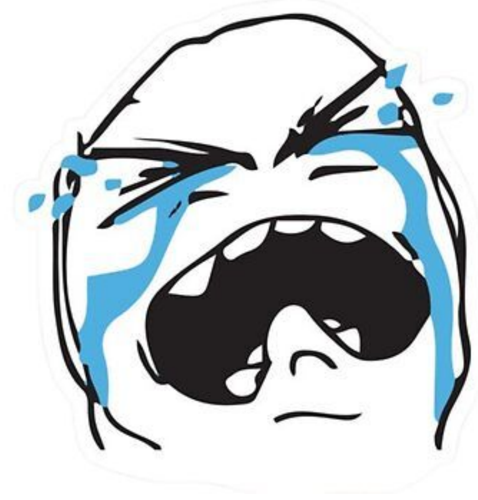


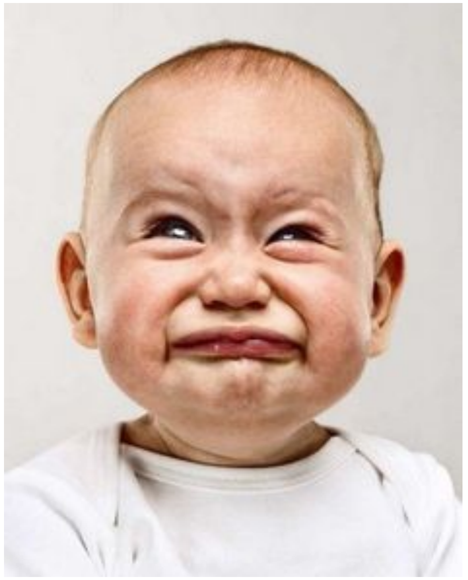
BUT WHY?????

CNN

1. They are good in capturing spatial patterns (can handle 2D images)
2. Good at feature detection.
3. They share parameters across local region, reducing number of parameters
4. Translational Invariance

BUT ONLY CNN'S ARE NOT ENOUGH



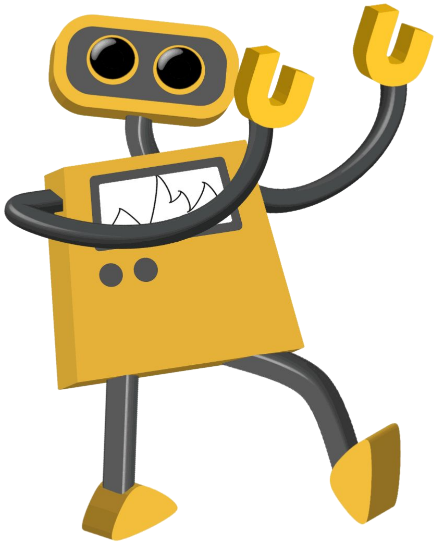


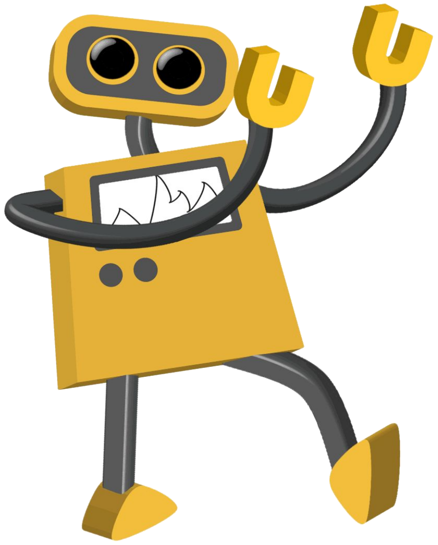
NEED MORE COMPLEX ARCHITECTURES WITH DIFFERENT WAYS TO CONVERGE THE MODEL AND MITIGATE OVERFITTING

GOOD MODELS (*for this homework)

1. RESNET
2. SE-RESNET
3. (ANY MAGICAL ARCHITECTURE)

LET'S TALK ABOUT RESNETS





ResNet

- To Convert Paper -> Code [understand the blocks + main aspects]
- ResNet Blocks: [Basic Blocks and Bottleneck Blocks]
- ResNet Main Aspect: Residual Connections

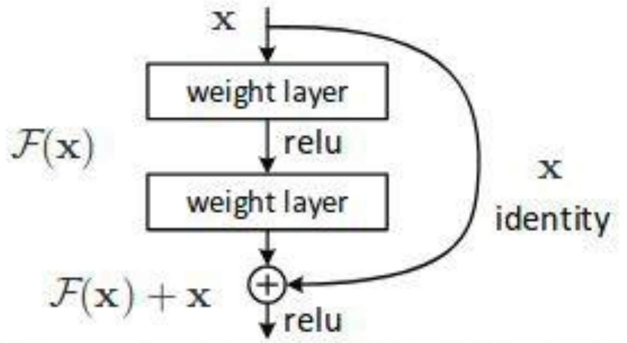
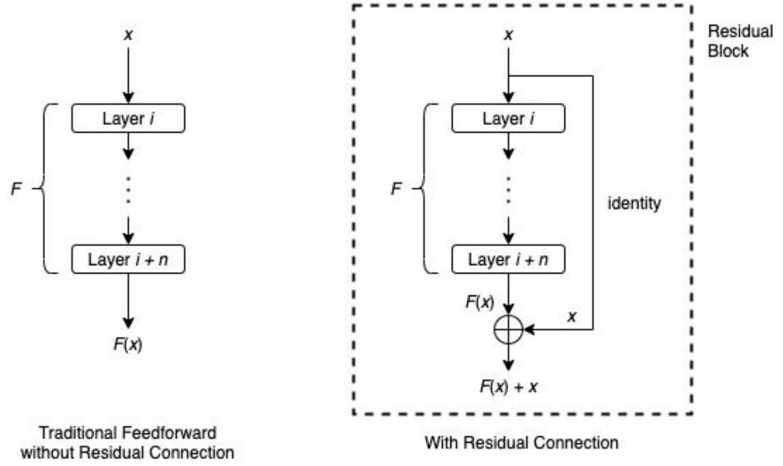


Figure 2. Residual learning: a building block.

Residual Connections

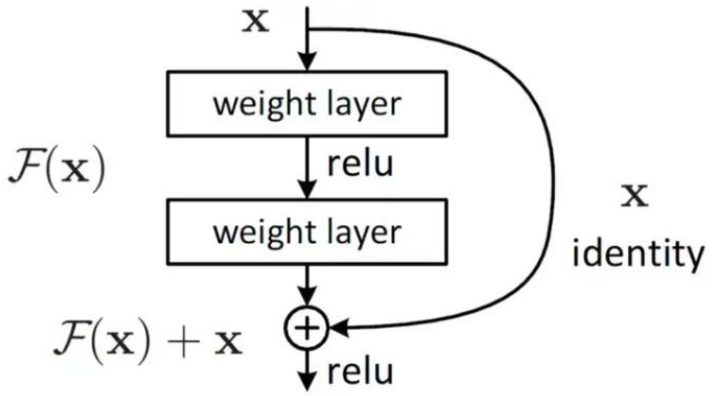
In traditional feedforward neural networks, data flows through each layer **sequentially**: The output of a layer is the input for the next layer.

Residual connection provides another path for data to reach latter parts of the neural network by **skipping** some layers.



How do they help??

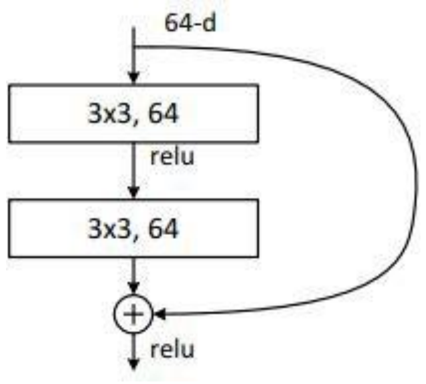
- For feedforward neural networks, training a deep network is usually very difficult, due to problems such as **exploding gradients and vanishing gradients**.
- On the other hand, the training process of a neural network with residual connections is empirically shown to converge much more easily, even if the network has several hundreds layers.
- Without Residual Connections, the gradients (signal for learning) get weaker as they flow backward to these layers.



Residual Connections

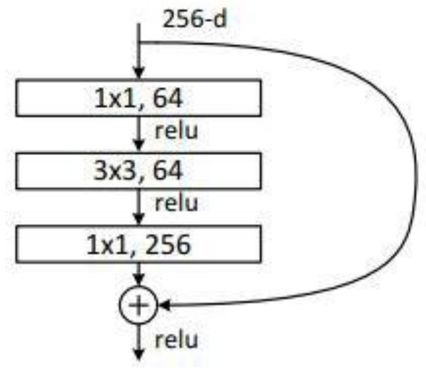
- The residual connection first applies identity mapping to *x*
- Then it performs element-wise addition $F(x) + x$.
- The whole architecture that takes an input *x* and produces output $F(x) + x$ is usually called a residual block or a building block.
- Quite often, a residual block will also include an activation function such as ReLU applied to $F(x) + x$.

ResNet: BasicBlock



- It’s just a regular 3x3 convolution (then BN, ReLU), another 3x3 convolution (then BN).
- Then, a skip connection adding input and output, then ReLU.

ResNet: Bottleneck Block



- A bit more involved.
- A 256-channel input goes through a point-wise convolution, reducing channels to 64.
- Then, a 3x3 regular convolution maintains channels at 64.
- Then, a point-wise convolution expands channels back to 256.
- Finally, the residual connection.

Residual Connection - Basic Block

```
class BasicBlock(torch.nn.Module):

  def __init__(self, n_h):

    self.linear0 = torch.nn.Linear(n_h, n_h)
    self.linear1 = torch.nn.Linear(n_h, n_h)

    self.bn0 = torch.nn.BatchNorm1d(n_h)
    self.bn1 = torch.nn.BatchNorm1d(n_h)

    self.relu = torch.nn.ReLU(inplace=True)
```

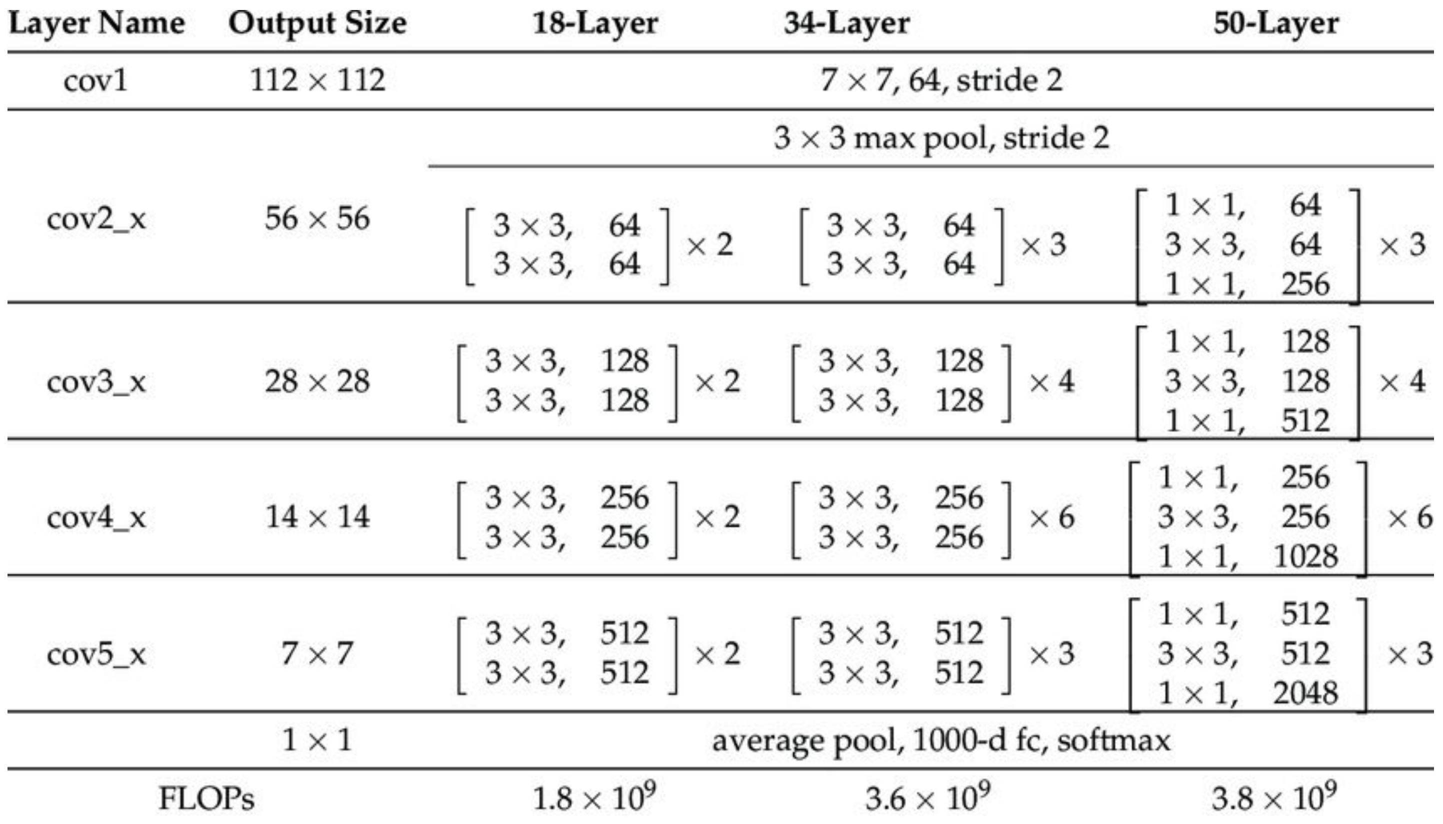
```
def forward(self, A0):

  R0  = A0

  Z0  = self.linear0(A0)
  BZ0 = self.bn0(Z0)
  A1  = self.relu(BZ0)

  Z1  = self.linear1(A1)
  BZ1 = self.bn1(Z1)
  A2  = self.relu(BZ1 + R0)

  return A2
```

Residual Connection - Bottleneck Block

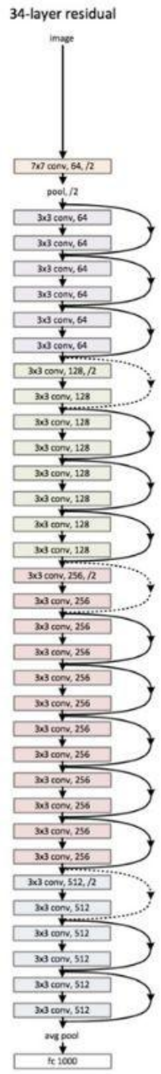
```
class Bottleneck(torch.nn.Module):

  def __init__(self, n_h):

    self.residual = torch.nn.Linear(n_h, n_h*4)

    self.linear0 = torch.nn.Linear(n_h, n_h  )
    self.linear1 = torch.nn.Linear(n_h, n_h  )
    self.linear2 = torch.nn.Linear(n_h, n_h*4)

    self.bn0 = torch.nn.BatchNorm1d(n_h  )
    self.bn1 = torch.nn.BatchNorm1d(n_h  )
    self.bn2 = torch.nn.BatchNorm1d(n_h*4)

    self.relu = torch.nn.ReLU(inplace=True)
```

```
def forward(self, A0):
  R0  = self.residual(A0)

  Z0  = self.linear0(A0)
  BZ0 = self.bn0(Z0)
  A1  = self.relu(BZ0)

  Z1  = self.linear1(A1)
  BZ1 = self.bn1(Z1)
  A2  = self.relu(BZ1)

  Z2  = self.linear2(A2)
  BZ2 = self.bn2(Z2)
  A3  = self.relu(BZ2 + R0)
  return A3
```

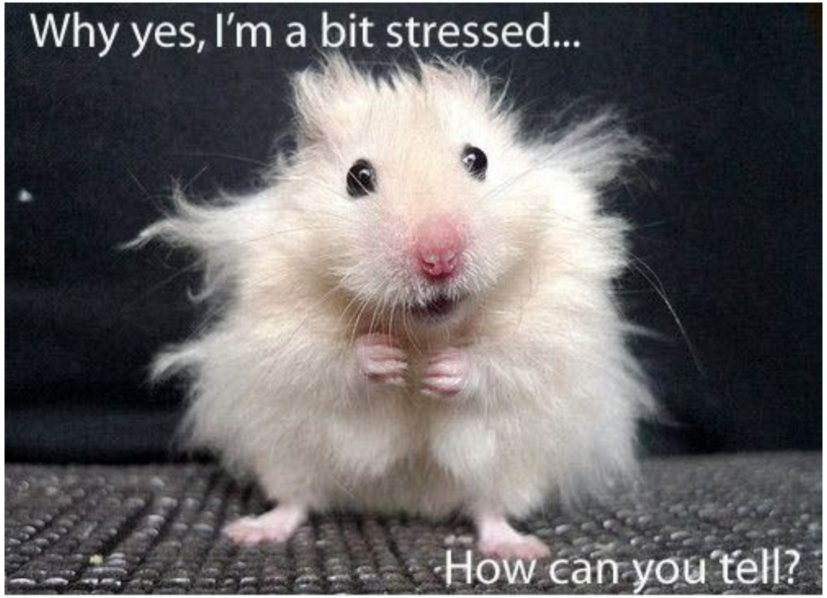
ResNet: Overall Architecture



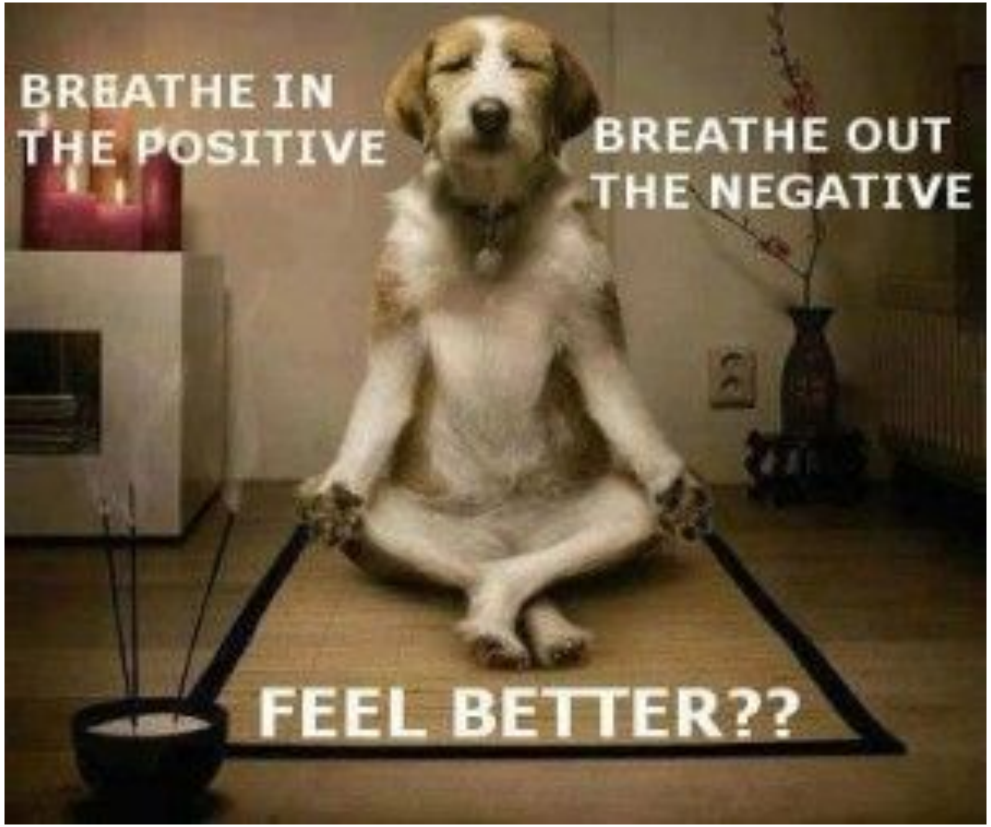
Layer Name	Output Size	18-Layer	34-Layer	50-Layer
cov1	112×112	7×7, 64, stride 2		
		3×3 max pool, stride 2		
cov2_x	56×56	$\begin{bmatrix} 3\times3, & 64 \\ 3\times3, & 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, & 64 \\ 3\times3, & 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, & 64 \\ 3\times3, & 64 \\ 1\times1, & 256 \end{bmatrix} \times 3$
cov3_x	28×28	$\begin{bmatrix} 3\times3, & 128 \\ 3\times3, & 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, & 128 \\ 3\times3, & 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, & 128 \\ 3\times3, & 128 \\ 1\times1, & 512 \end{bmatrix} \times 4$
cov4_x	14×14	$\begin{bmatrix} 3\times3, & 256 \\ 3\times3, & 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, & 256 \\ 3\times3, & 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, & 256 \\ 3\times3, & 256 \\ 1\times1, & 1028 \end{bmatrix} \times 6$
cov5_x	7×7	$\begin{bmatrix} 3\times3, & 512 \\ 3\times3, & 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, & 512 \\ 3\times3, & 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, & 512 \\ 3\times3, & 512 \\ 1\times1, & 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9



AHHHHHHHHHHH



BREATHE!!!!


BREATHE IN THE POSITIVE
BREATHE OUT THE NEGATIVE
FEEL BETTER??

General Architecture Flow

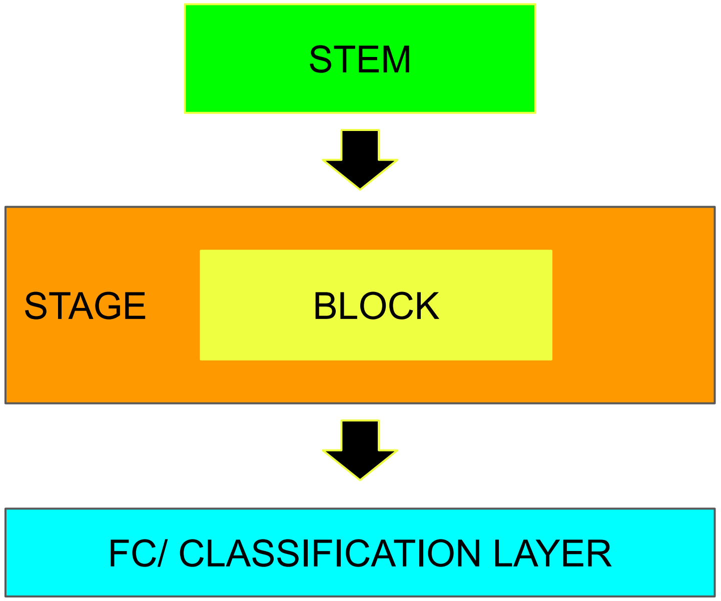
- CNN architectures are divided into stages, which are divided into blocks.
    - Each “stage” consists of (almost) equivalent “blocks”
    - Each “block” consists of a few CNN layers, BN, and ReLUs.
- To understand an architecture, we mostly need to understand its blocks.
- All that changes for blocks in different stages is the base # of channels

ConvNeXt

- This is a very new paper, a state-of-the-art architecture.
- However, its **intuitions are very similar to MobileNetV2**.
- Again, remember that to understand a paper, we just really need to understand its **blocks**.
- Just a single block type for ConvNeXt
- **Read the paper** for details on stages/channel sizes, etc.
    - We recommend ConvNeXt-T size which has less than 35M parameters.

General Architecture Flow

- However, you do need to piece these blocks together into a final model.
- The general flow is like this:
  - Stem
  - Stage 1
  - Stage 2
  - …
  - Stage n
  - Classification Layer



Summary

- A normal convolution mixes information from **both different channels and different spatial locations (pixels)**
- A depth-wise convolution only mixes information **over spatial locations**
  - Different channels do not interact.
- A point-wise convolution only mixes information **over different channels**
  - Different spatial locations do not interact

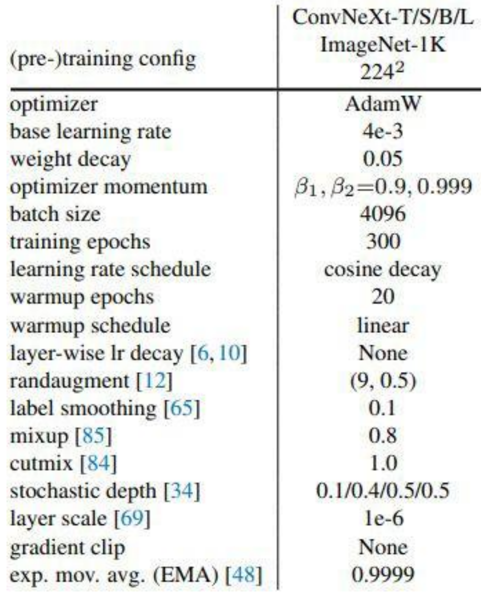
3. Training, monitoring, testing, and Kaggle submission

Monitoring Training vs Validation Acc

- The standard intuition of “overfitting” is – if the training & validation gap is too large, you should stop training as it’s overfitting.
- However, in modern DL, this intuition is not as relevant.
- XELoss != Accuracy
  - Model can keep improving after training accuracy hits 100%.
  - There is recent research that finds that on some problems, training accuracy hits 100% at epoch 10 while validation accuracy is <50%. Then, on epoch 1000,
    validation hits 100%.
- Of course, we can’t train for that long, but train until validation stops improving.
  - Or just set a standard LR schedule/setup like “CosineAnnealingLR for 50 epochs” and just let it run. □ what I prefer to do.

How to tackle overfitting?

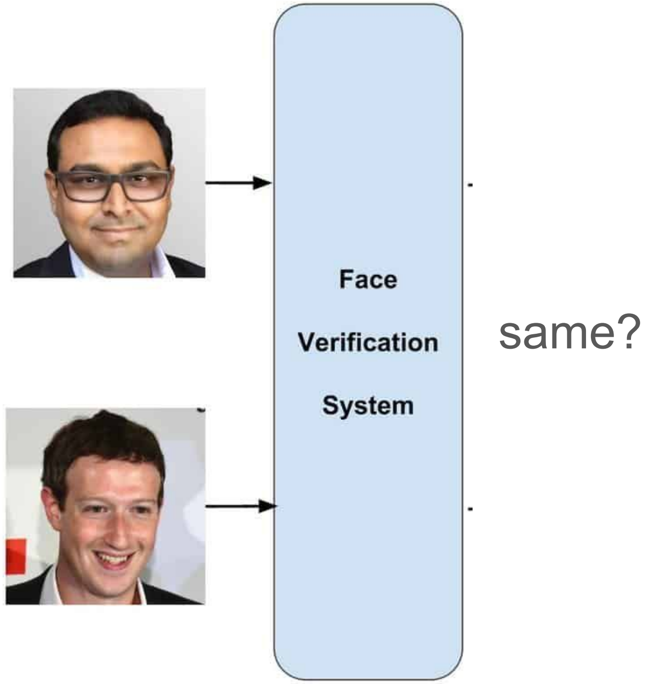
- There are *a lot* of different trick to improving your CNN model.
- From the recent ConvNeXt paper
- What we recommend trying first:
  - Label Smoothing (huge boost)
  - Stochastic Depth
  - DropBlock (paper)
  - Dropout before final classification layer
- Then you can try the others
- Check out "Bag of Tricks for Image Classification with Convolutional Neural Networks"
  - https://arxiv.org/abs/1812.01187



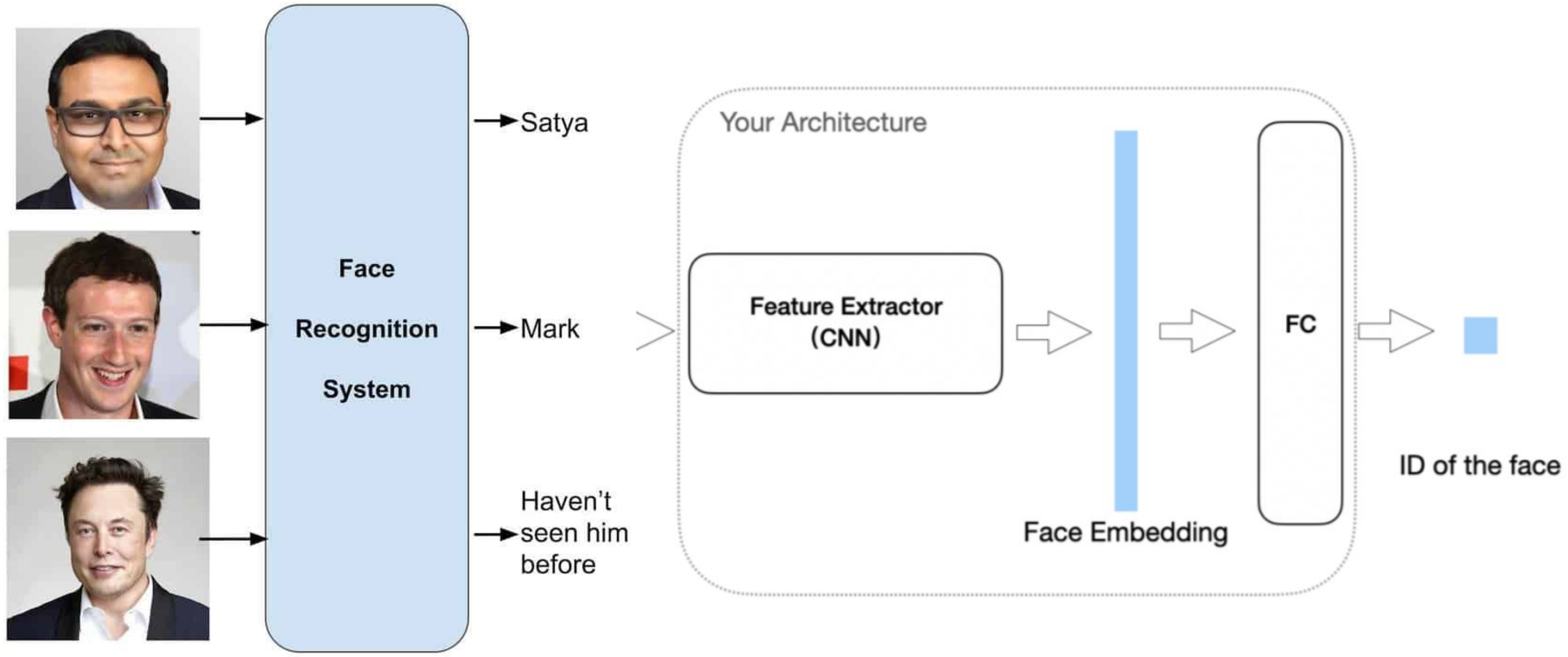
(pre-)training config	ConvNeXt-T/S/B/L ImageNet-1K 224^2
optimizer	AdamW
base learning rate	4e-3
weight decay	0.05
optimizer momentum	β_1, β_2=0.9, 0.999
batch size	4096
training epochs	300
learning rate schedule	cosine decay
warmup epochs	20
warmup schedule	linear
layer-wise lr decay [6, 10]	None
randaugment [12]	(9, 0.5)
label smoothing [65]	0.1
mixup [85]	0.8
cutmix [84]	1.0
stochastic depth [34]	0.1/0.4/0.5/0.5
layer scale [69]	1e-6
gradient clip	None
exp. mov. avg. (EMA) [48]	0.9999

4. Different loss functions and model fine-tuning

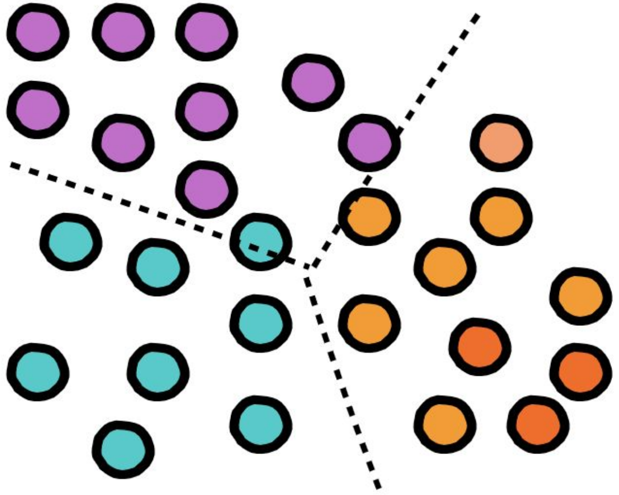
How is this different from **Face Classification**?



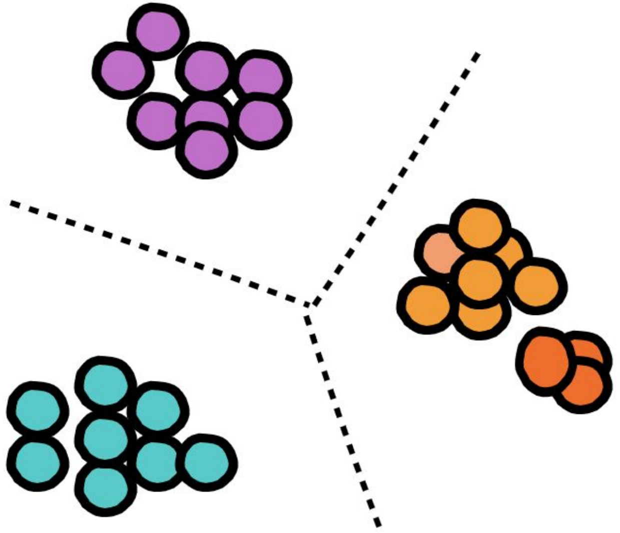
How is this different from **Face Classification**?



How is this different from **Face Classification**?



Separable Features (e.g. classification)



Discriminative Features (e.g. metric learning)

Why classification cannot do verification?

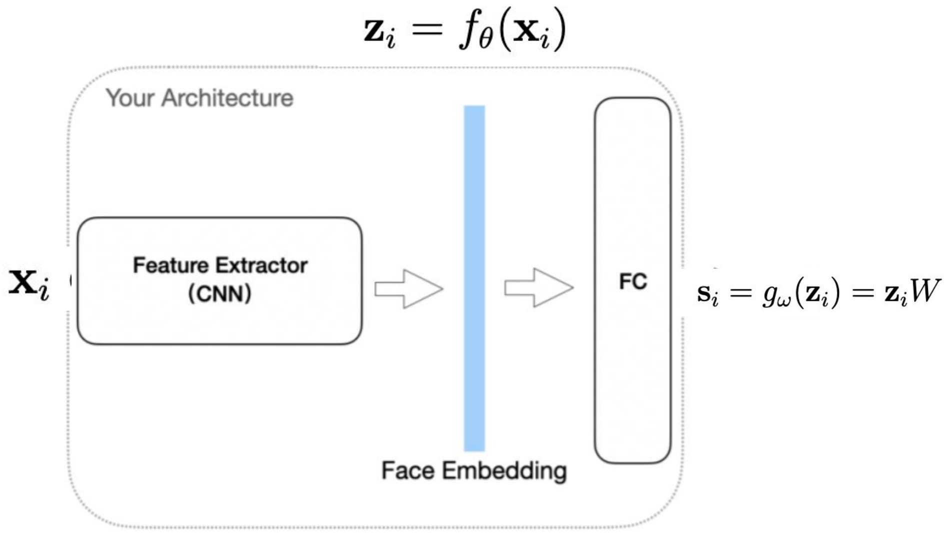
Classifier can only perform closed-set recognition

- Need to re-train with open-set new identities everytime

But…we can just use the backbone to extract features and compare features, which does not require classifier

Features learned by the classifier with Cross-Entropy is **not discriminative** enough

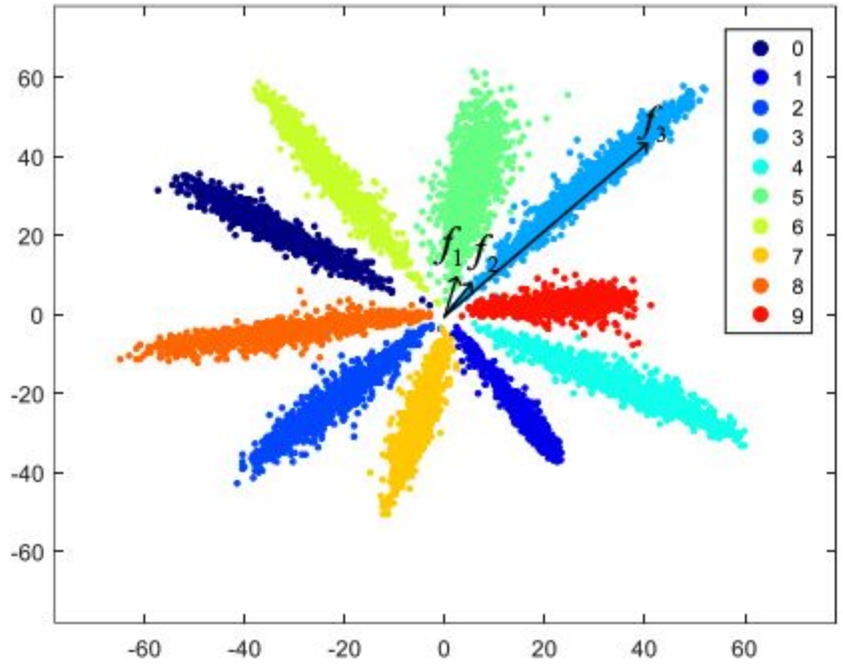
A Closer Look into Cross-Entropy

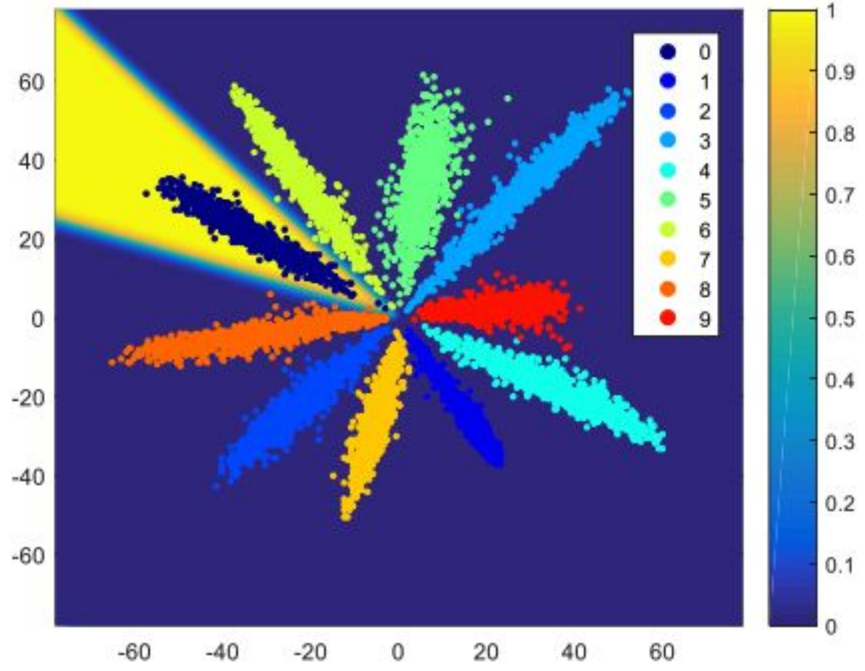

$\mathbf{z}_i = f_\theta(\mathbf{x}_i)$
Your Architecture
$\mathbf{x}_i$
Feature Extractor (CNN)
Face Embedding
FC
$\mathbf{s}_i = g_\omega(\mathbf{z}_i) = \mathbf{z}_i W$

$$\mathbf{p} = \mathrm{Softmax}(\mathbf{s}) = [\frac{\exp(s^1)}{\sum_{k=1}^{C}\exp(s^k)}, ..., \frac{\exp(s^C)}{\sum_{k=1}^{C}\exp(s^k)}] \in \mathbb{R}^C$$

$$\mathcal{L}_{\mathrm{CE}} = \frac{1}{N}\sum_{i}^{N}\sum_{j}^{C} -y_i^j \log p_i^j.$$

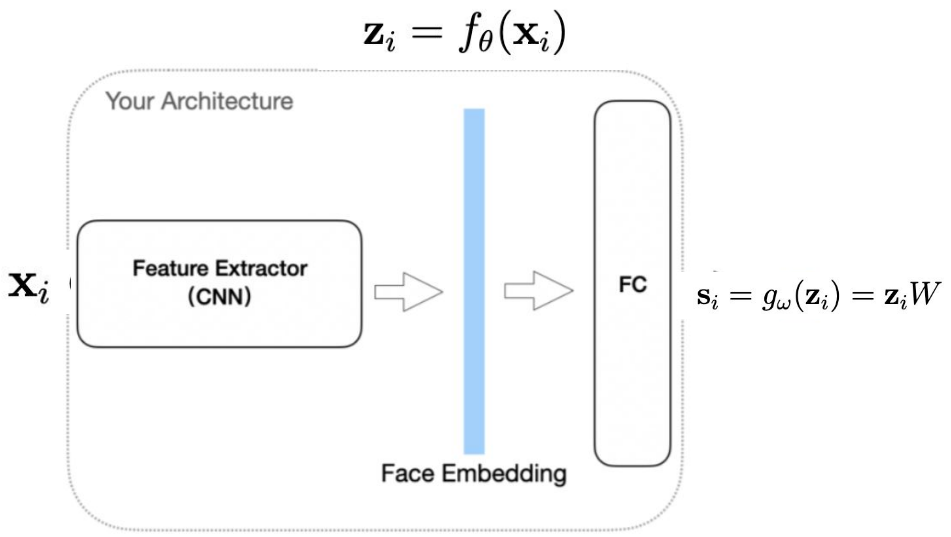
Feature Visualization of Cross-Entropy





NormFace: L2 Hypersphere Embedding for Face Verification. Feng Wang et al.

A Closer Look into Cross-Entropy

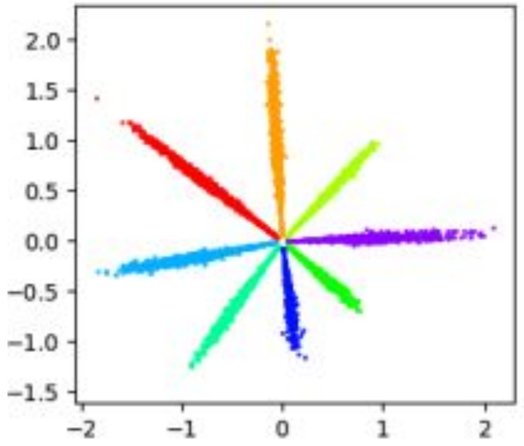

$\mathbf{z}_i = f_\theta(\mathbf{x}_i)$
Your Architecture
$\mathbf{x}_i$
Feature Extractor (CNN)
Face Embedding
FC
$\mathbf{s}_i = g_\omega(\mathbf{z}_i) = \mathbf{z}_i W$

$$\mathbf{p} = \mathrm{Softmax}(\mathbf{s}) = [\frac{\exp(s^1)}{\sum_{k=1}^{C}\exp(s^k)}, ..., \frac{\exp(s^C)}{\sum_{k=1}^{C}\exp(s^k)}] \in \mathbb{R}^C$$

$$\mathcal{L}_{\mathrm{CE}} = \frac{1}{N}\sum_i^N \sum_j^C -y_i^j \log p_i^j.$$

$$\begin{aligned} l_{\mathrm{CE}} &= \sum_j^C -y^j \log p^j \\ &= \sum_j^C -y^j \log \frac{\exp(s^j)}{\sum_{k=1}^{C}\exp(s^k)} \\ &= -\log \frac{\sum_{k=1,k\neq y}^{C}\exp(s^k)}{\exp(s^y)} \\ &= s^y - \log \sum_{k=1,k\neq y}^{C}\exp(s^k) \\ &\approx s^y - \max_{k\in[C],k\neq y}(s^k) \end{aligned}$$

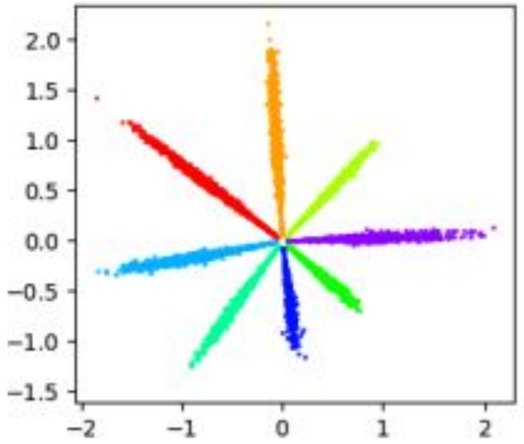
What are better features?



Larger intra-class similarity
(Smaller intra-class distance)

Smaller inter-class similarity
(Larger inter-class distance)

What are better features?



Larger intra-class similarity
(Smaller intra-class distance)

Smaller inter-class similarity
(Larger inter-class distance)

Human language?
Features of the same class have larger similarity (smaller distance) than other classes

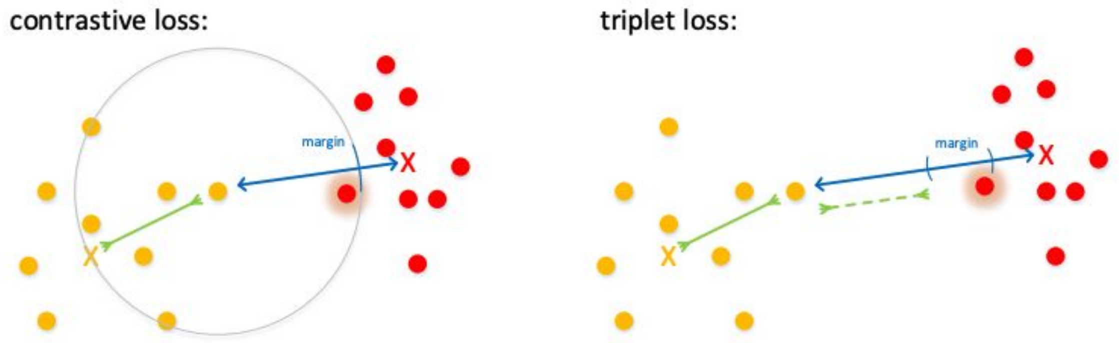
How do we enforce the model to learn these **discriminative features**?

How do we maximize the similarity between **“positive” pairs** and minimize the similarity between **“negative” pairs**?

Two Paradigms

- Metric Learning/Pairwise Learning
- Margin-based Softmax

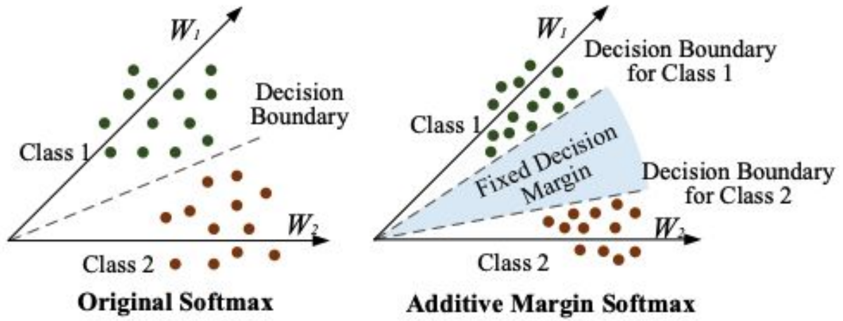
Metric Learning/Pairwise Learning



Centre Loss
Triplet Loss
N-Pair Loss
Contrastive Loss…

$$\mathcal{L} = \max(s^n - s^p + m, 0)$$

Margin-based Softmax



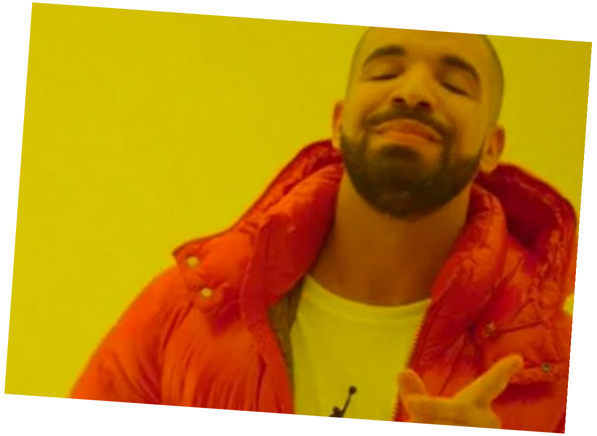
AM-Softmax
SphereFace
CosFace
ArcFace
CombinedMarginFace
…

$$L_1 = -\log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{N} e^{W_j^T x_i + b_j}} \quad L_3 = -\log \frac{e^{s \cos(\theta_{y_i} + m)}}{e^{s \cos(\theta_{y_i} + m)} + \sum_{j=1, j \neq y_i}^{N} e^{s \cos \theta_j}}$$

How to Get Started

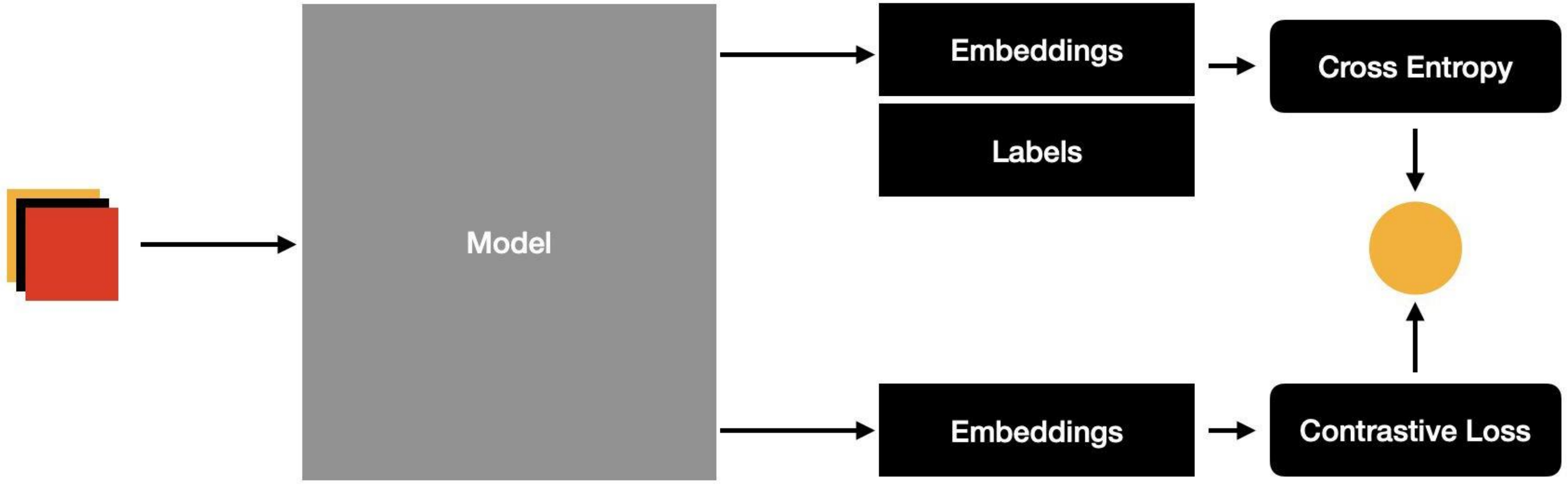
- Use a simple network and cross-entropy loss for early deadline
- Try better architectures with cross-entropy loss
- Try data augmentation
- Try other loss functions
- Fine-tune your cross-entropy loss with other loss functions

Contrastive Losses

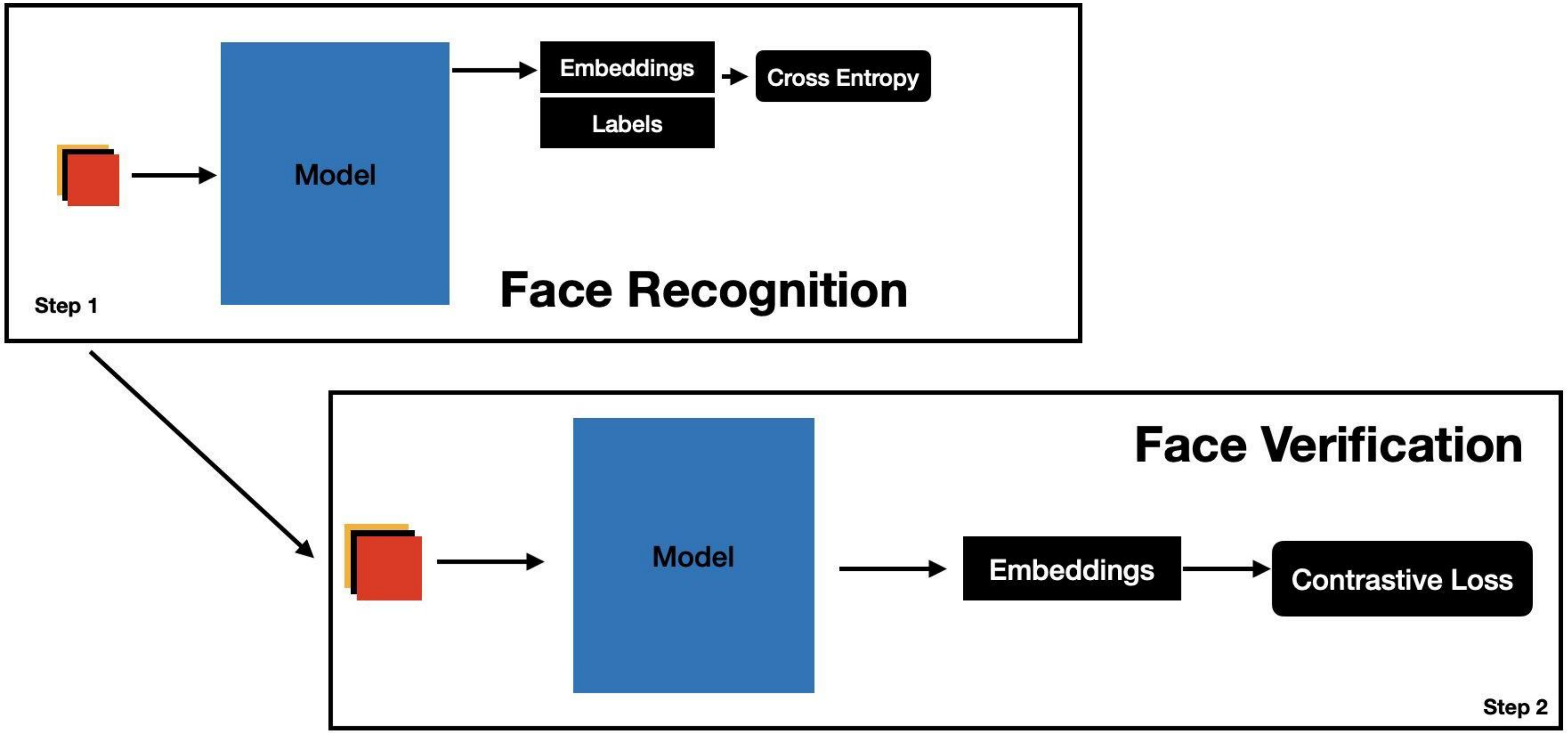


Let’s compute a similarity metric between model embeddings to learn more discriminative features between the input data.

Approach 1 - Joint Loss Optimization



Approach 2 - Sequential (Fine-tuning)



Types of Contrastive Losses

Centre Loss

ArcFace Loss

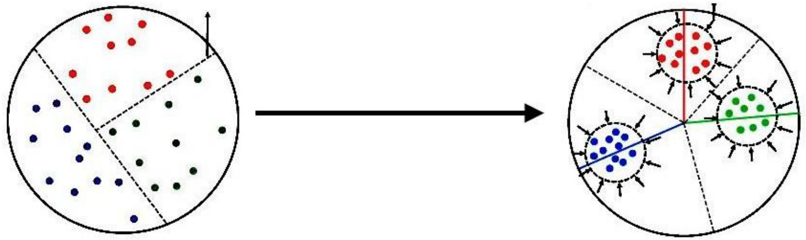
SphereFace Loss

Centre Loss

$$\mathcal{L}_C = \frac{1}{2}\sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{c}_{y_i}\|_2^2$$

- Increases the disparity between classes using softmax
- Increases inter-class distance by reducing intra-class Euclidean distance by assigning centers to each class.

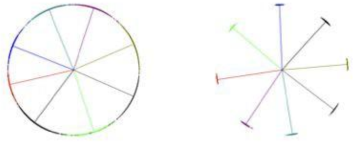
- Calculating the centre for each class, is difficult

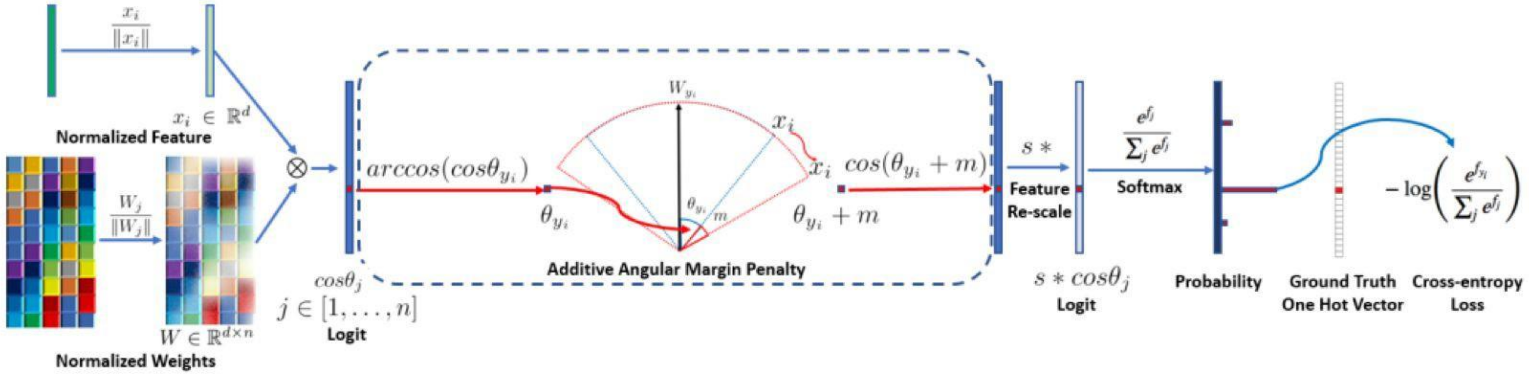


ArcFace

$$L = -\frac{1}{N}\sum_{i=1}^{N}\ln\frac{\exp\left\{s\cdot\cos(\theta_{y_i,i}+m)\right\}}{\exp\left\{s\cdot\cos\left(\theta_{y_i,i}+m\right)\right\}+\sum_{j\neq y_i}\exp\left\{s\cdot(\cos(\theta_{j,i})\right\}}$$

- Builds on the concepts of the sphere face and cos face.
- Replaced the multiplicative angular margin in CosFace, with an additive margin ‘m’


(a) Norm-Softmax (b) ArcFace



ArcFace

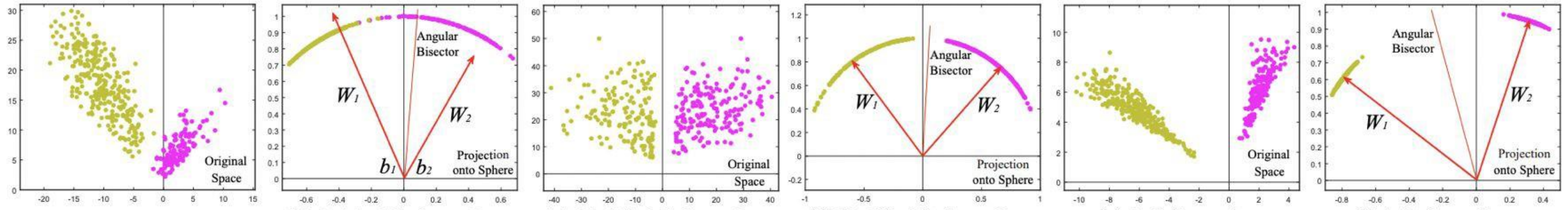
- The additive factor of ‘m’ has found to lead to better convergence as compared to its multiplicative counterpart in Sphere Face.

Sphere Face

$$L_{\text{ang}} = -\sum_i \ln \frac{\exp\left\{\|\mathbf{x_i}\| \cos(m \cdot \theta_{y_i,i})\right\}}{\exp\left\{\|\mathbf{x_i}\| \cos(m \cdot \theta_{y_i,i})\right\} + \sum_{j \neq y_i} \exp\left\{\|\mathbf{x_i}\| \cos\left(\theta_{j,i}\right)\right\}}$$

- **Makes use of an angular margin, imposed by θ**
- **The learned features construct a discriminative angular distance equivalent to the geodesic distance on a hypersphere manifold**

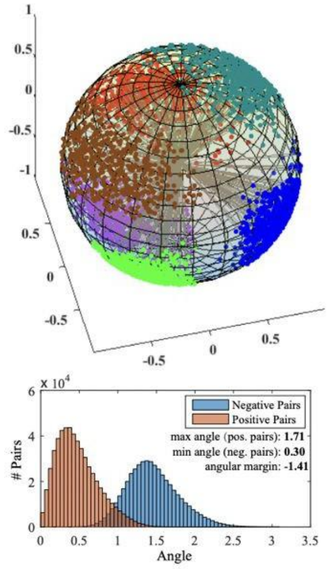
- **θ :- denotes the type of decision boundary learned, which leads to different margins for different classes**



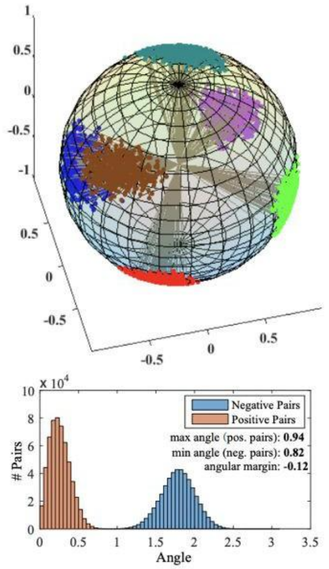
(a) Original Softmax Loss (b) Original Softmax Loss (c) Modified Softmax Loss (d) Modified Softmax Loss (e) A-Softmax Loss (f) A-Softmax Loss

Sphere Face

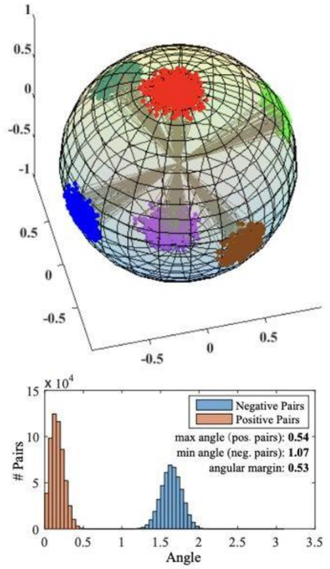
M ∝ Angular Margin



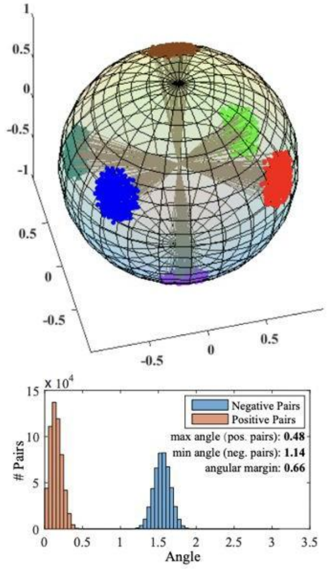
A-Softmax (m=1)



A-Softmax (m=2)



A-Softmax (m=3)



A-Softmax (m=4)

FAQ

Thanks!

See you on Piazza and in OHs!