Carnegie Mellon University

11-785 - Introduction to Deep Learning - Fall 22 -

Recitation 6: CNNs (Classification & Verification)

By:

Aditya Singh

Cedric Manouan

Agenda

1. Problem statement:
   a. Classification
   b. Verification

2. Data description

4. Verification optimized approaches

3. Different types of convolution layers

   a. Depthwise convolution
   b. Pointwise convolution
   c. Depthwise Separable convolution

5. Run through the starter notebook

Carnegie Mellon University

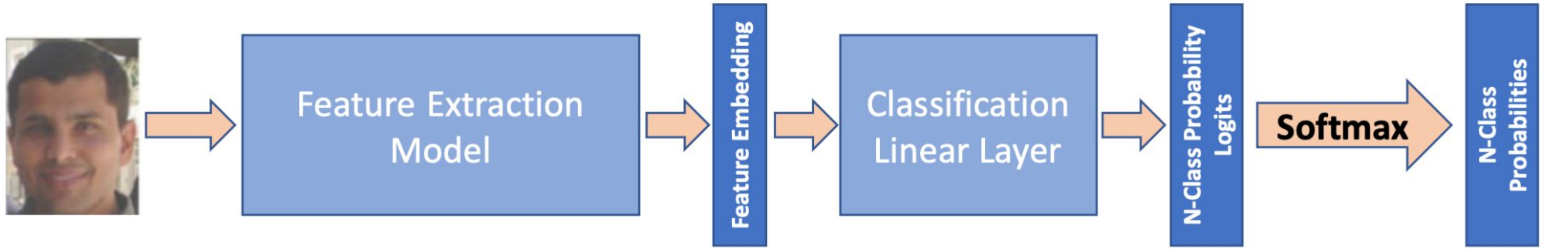
1. Problem statement

Face Classification

Given an image, figure out which person it is.

Face Verification

Determine if the person in a given "query" picture is also present in a given gallery of images or not, with no reference to their identity?



Carnegie Mellon University

In General

Classification

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

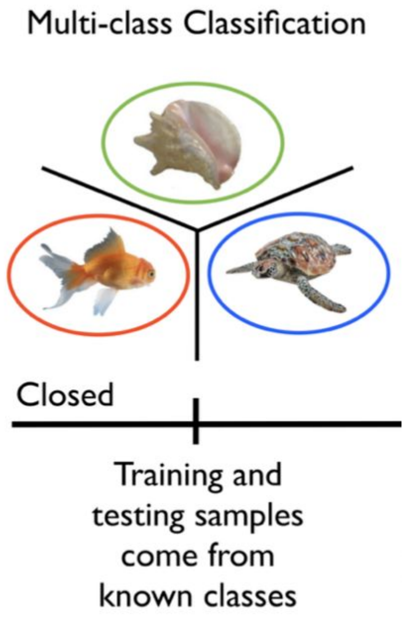
Verification

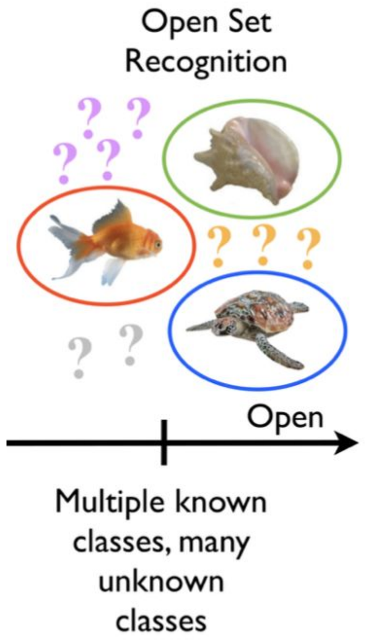
It is a **matching operation**, where you match the given sample to the closest sample from a reference of N other samples

Can also be a 1 to 1 task, where we want to verify if the two embeddings are similar (belong to the same class)

Carnegie Mellon University

Closed Set vs. Open Set


Multi-class Classification
Closed
Training and testing samples come from known classes



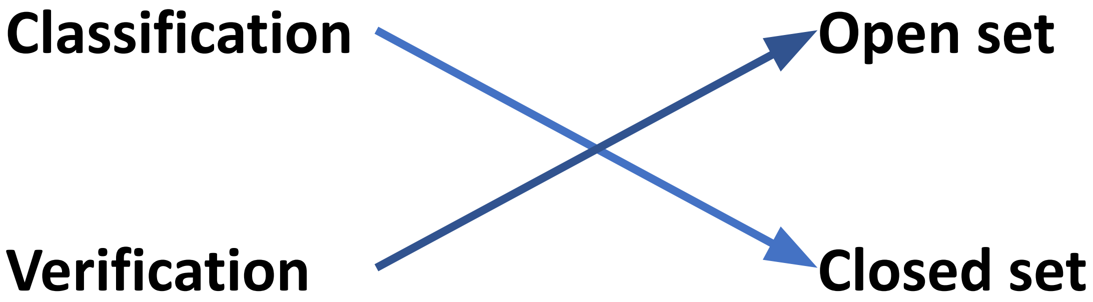
Carnegie Mellon University

Closed Set or Open Set?

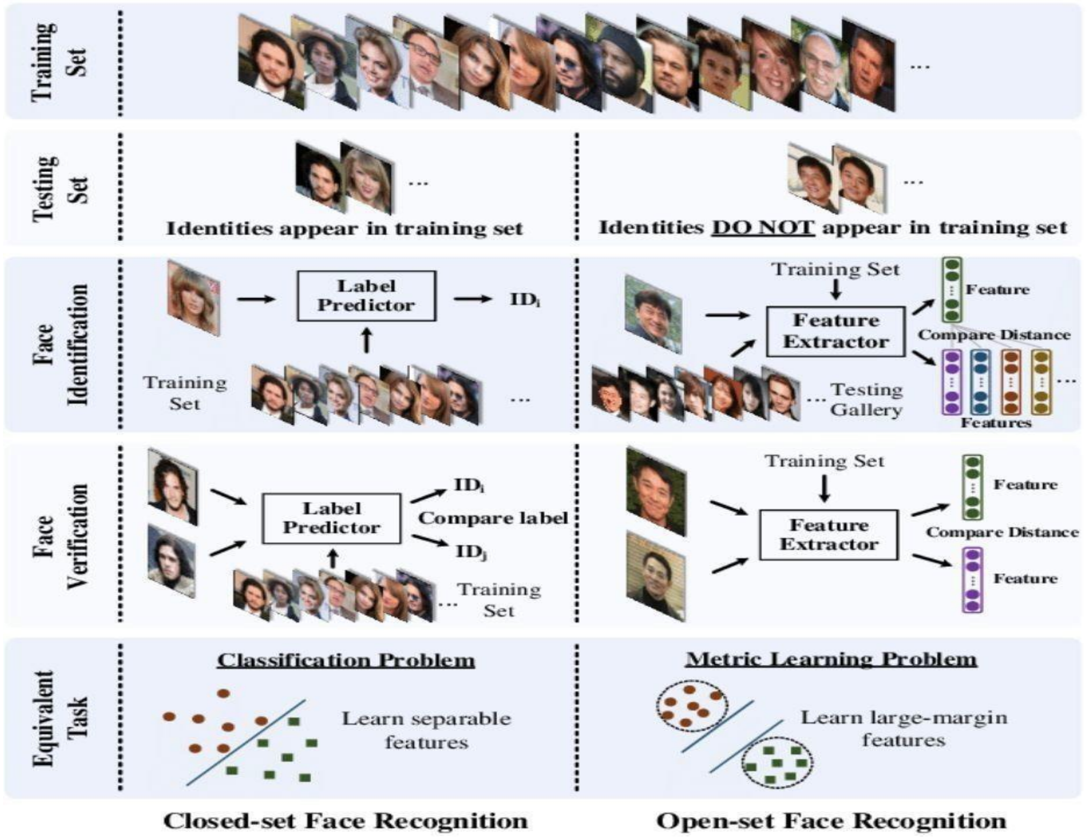
Classification	Open set
Verification	Closed set

Carnegie Mellon University

Closed Set or Open Set?



Open Set vs Closed Set



Source: https://www.researchgate.net/figure/Comparison-of-open-set-and-closed-set-face-recognition_fig1_316505674

1. Problem statement

Classification

This is a closed set problem, where the subjects in the test set have also been seen in the training set, although the precise pictures in the test set will not be in the training set.

Verification

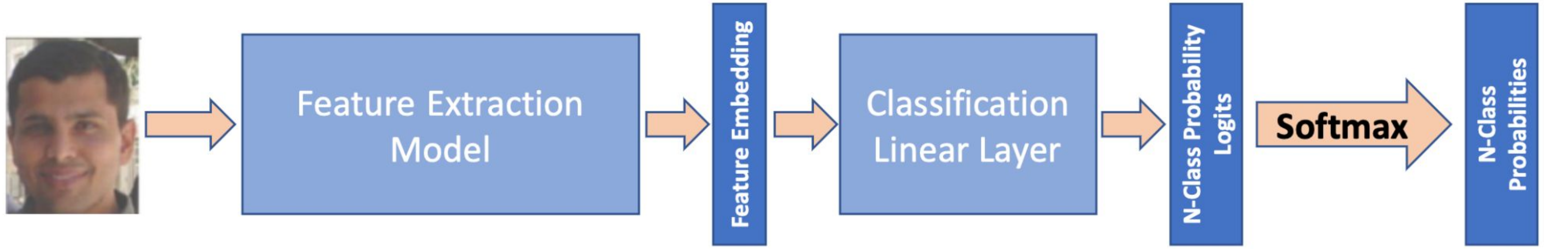
This is an open set problem, where the subjects in the test data may not have been seen during training at all.

Task: determine if the person in a given "query" picture is also present in a given gallery of images or not, with no reference to their identity

Training for classification

Task: Identify the person in a picture
aka Multi-class classification

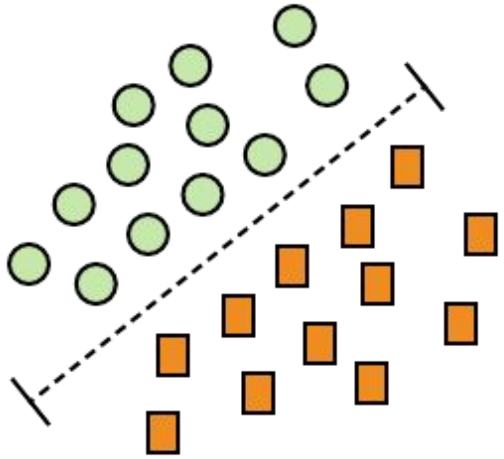
$$Div(Y,d) = \sum_i d_i \log \frac{d_i}{y_i}$$



Training for classification

Task: Identify the person in a picture
aka Multi-class classification

$$Div(Y, d) = \sum_i d_i \log \frac{d_i}{y_i}$$

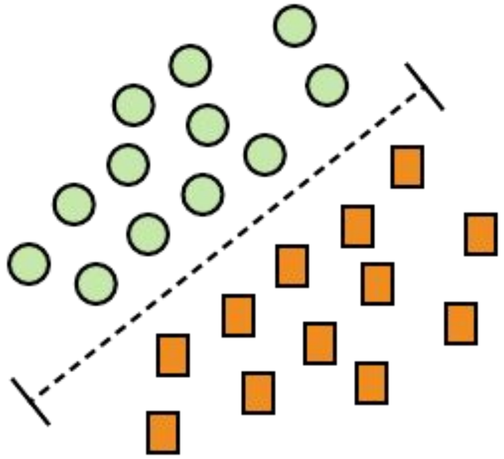


Good for classification, Gives us separable features !

Training for classification

Task: Identify the person in a picture
aka Multi-class classification

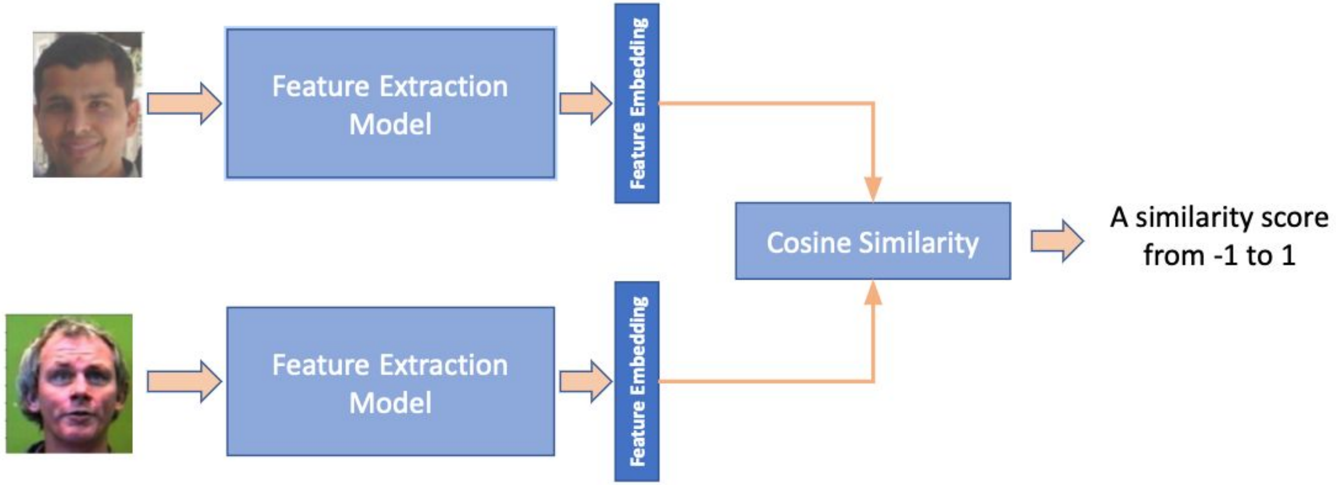
$$Div(Y,d) = \sum_i d_i \log \frac{d_i}{y_i}$$



Good for classification, Gives us separable features !

How can we use this network for verification?

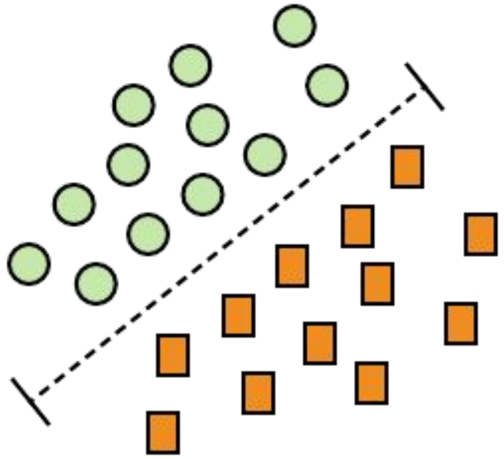
Face Verification



Training for classification

Task: Identify the person in a picture
aka Multi-class classification

$$Div(Y, d) = \sum_i d_i \log \frac{d_i}{y_i}$$

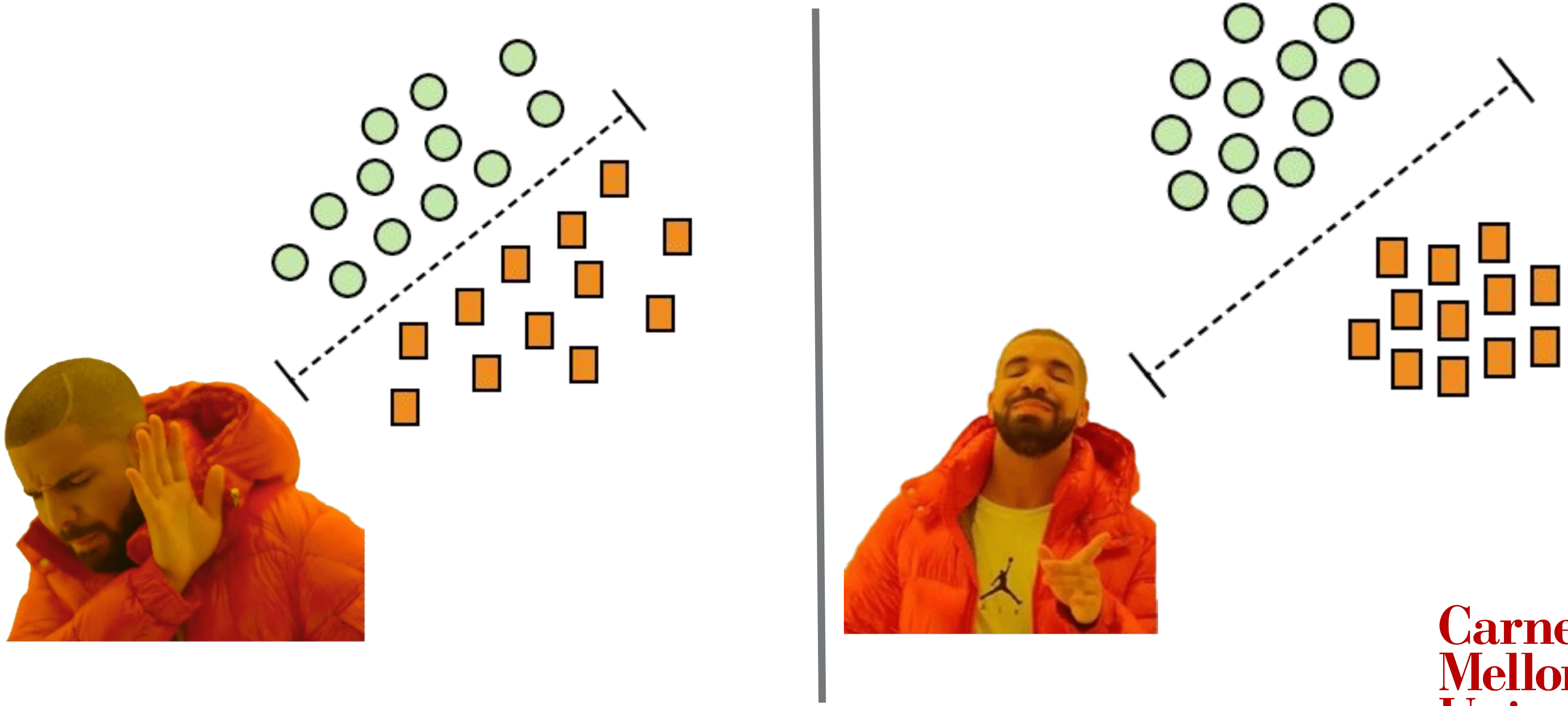


Good for classification, Gives us separable features !

How can we use this network for verification?

Is separable good enough for verification?

What we need - Discriminative Features



Carnegie Mellon University

How can we get discriminative features?

How can we get discriminative features?

Use a loss function which correlates to your evaluation criteria !

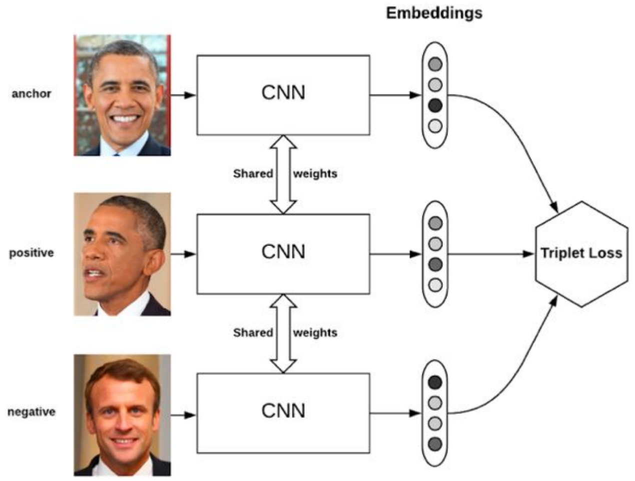
Train your network to generate
discriminative embeddings

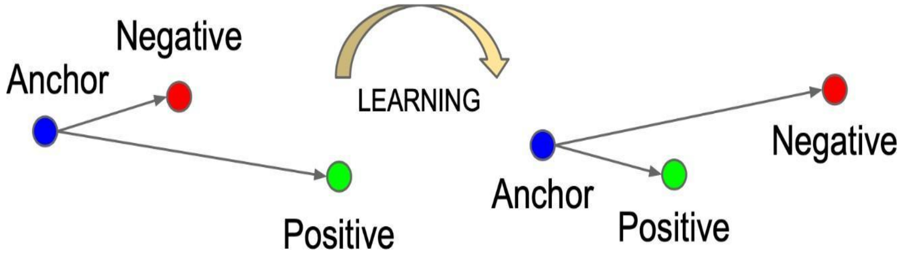
Enter Contrastive Losses

Carnegie Mellon University

Training for verification - Metric Learning

Task: determine if the person in a given “query” picture is also present in a given gallery of images or not, with no reference to their identity





$$\mathcal{L}_{\text{triplet}}(\mathbf{x}, \mathbf{x}^{+}, \mathbf{x}^{-}) = \sum_{\mathbf{x} \in \mathcal{X}} \max\left(0, \|f(\mathbf{x}) - f(\mathbf{x}^{+})\|_2^2 - \|f(\mathbf{x}) - f(\mathbf{x}^{-})\|_2^2 + \epsilon\right)$$

Triplet Loss setup

Contrastive Losses

1. Centre Loss
2. Triplet Loss
3. Sphere Face (Angular Softmax)
4. CosFace Loss
5. ArcFace Loss

Contrastive Losses

1. Centre Loss
2. Triplet Loss
3. Sphere Face (Angular Softmax)
4. CosFace Loss
5. ArcFace Loss

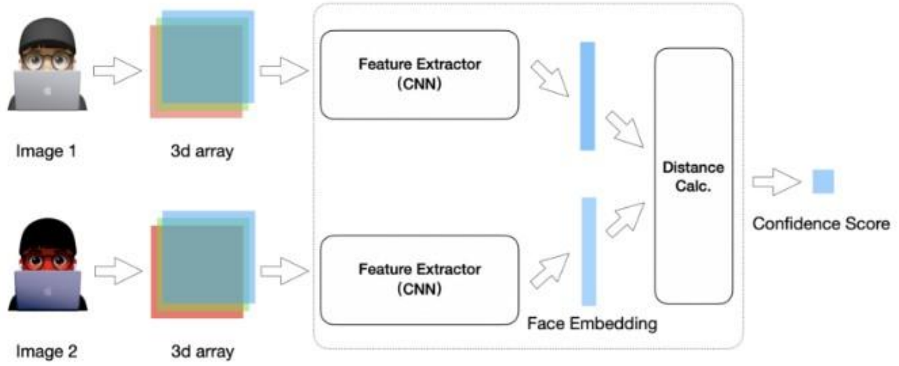
See discussion in HW2 bootcamp

1. Problem statement

Verification

This is an open set problem, where the subjects in the test data may not have been seen during training at all.

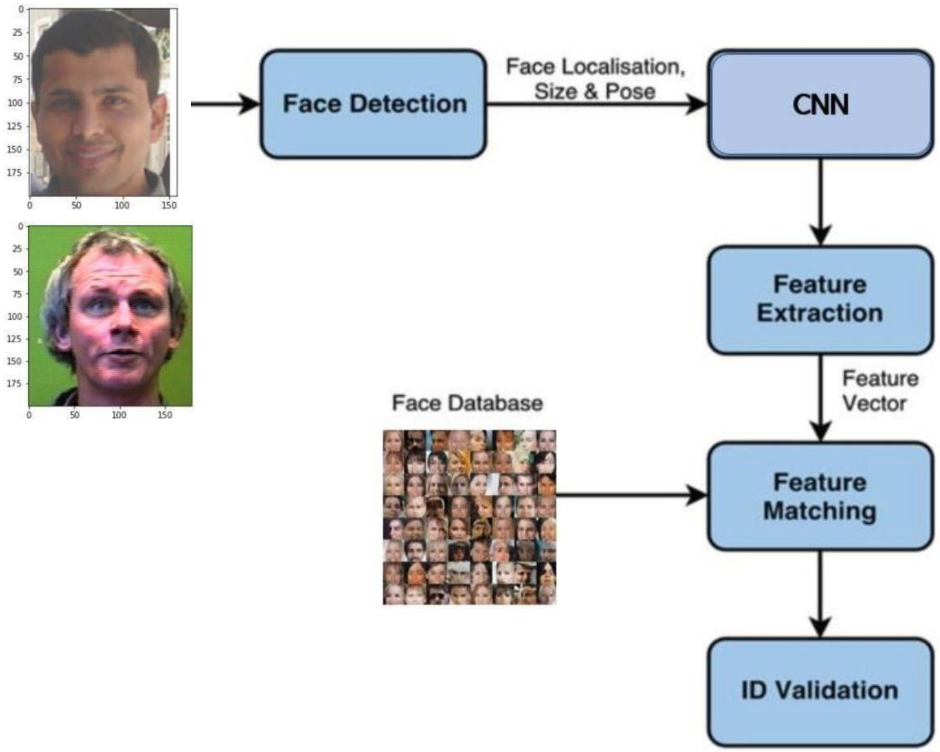
Task: determine if the person in a given “query” picture is also present in a given gallery of images or not, with no reference to their identity



For this problem you will have to find the similarity between each unknown identity and all the known identities, and then predict the one with the highest similarity

Carnegie Mellon University

Face Verification as image retrieval

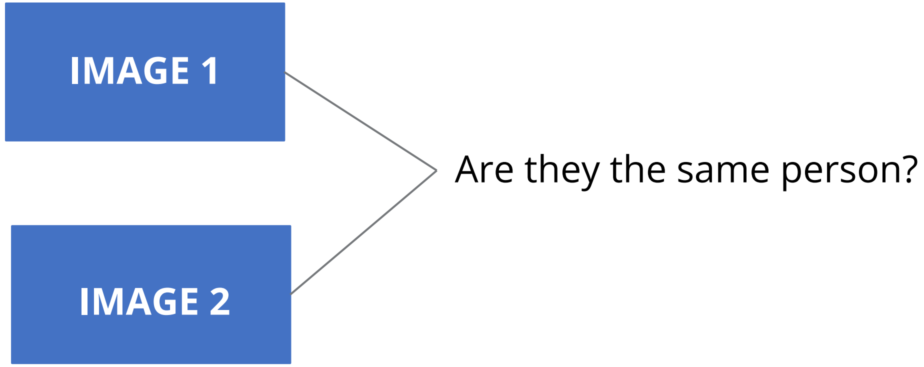


2. Data description- VGG Face2

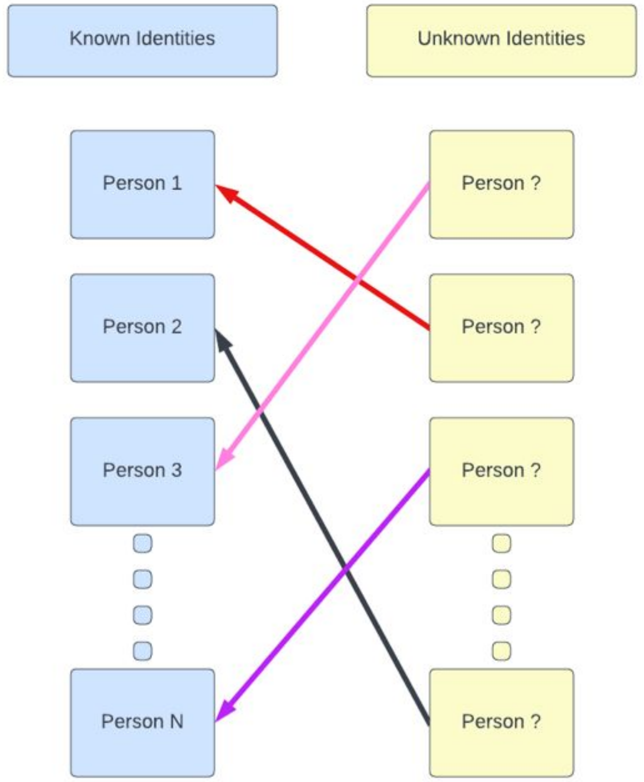
- State of the art VGG Face2 Dataset
- Largest in the world as of 2020
- 3.3 Million Face Images, ~362 samples per subject
- ~9130 identities, large variations in pose, age, demographics
- **7000** identities for classification, **1000** for verification
- Some demographics data also known (age, ethnicity)

Verification Task - Previous semesters

- **Test set**- pairs of images with labels about whether they belong to the same person
- **Task**: calculate AUC using similarity scores



Verification Task - now

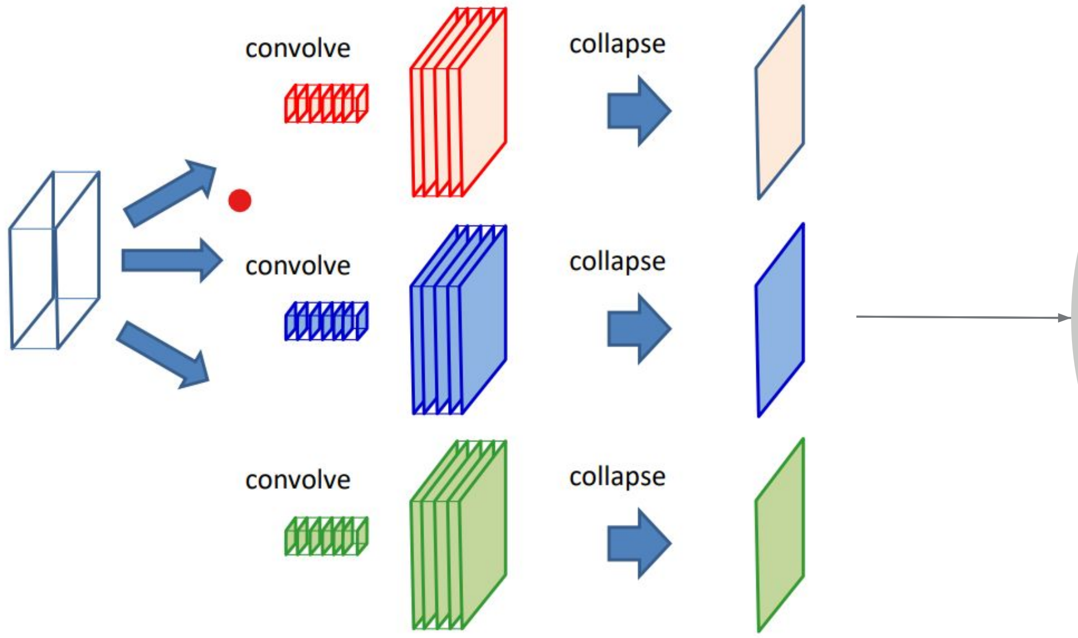


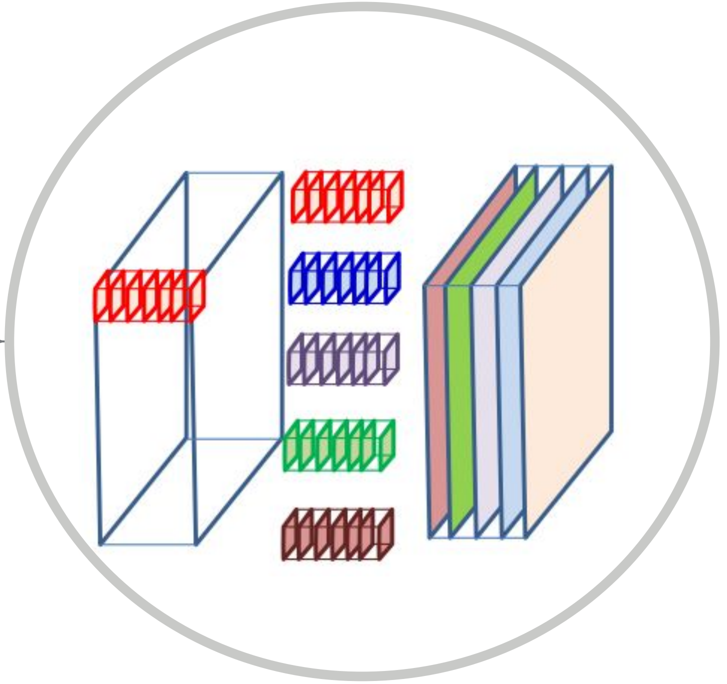
- **Test set- Balanced set of 1000 known identities and 1000 unknown identities**

- **Task:** find the similarity between each unknown identity and all the known identities, and then predict the one with the highest similarity

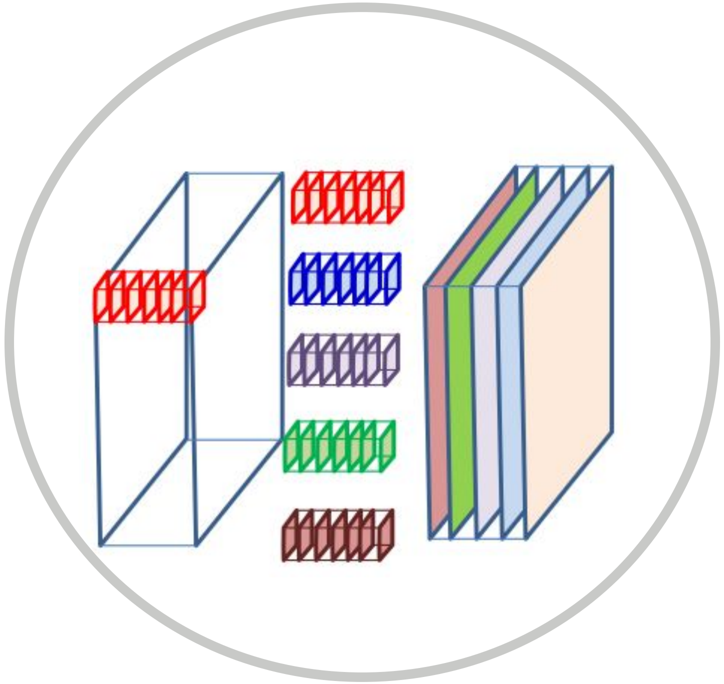
3. Different types of convolutions

Conventional convolution





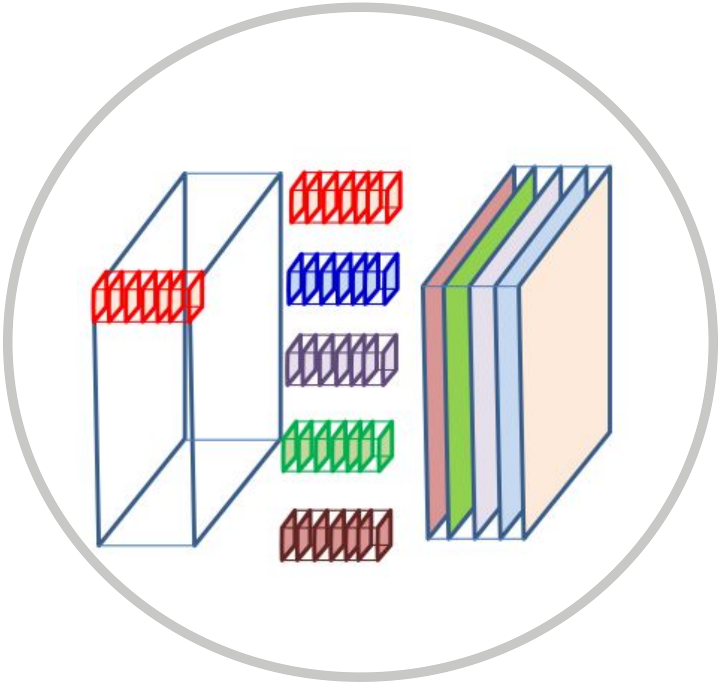
Conventional convolution



- Each layer of each filter scans its corresponding map to produce a convolved map
- N input channels **will require a filter with N layers**
- The independent convolutions of each layer of the filter result in N convolved maps
- The N convolved maps are added together to produce the final output map (or channel) for that filter

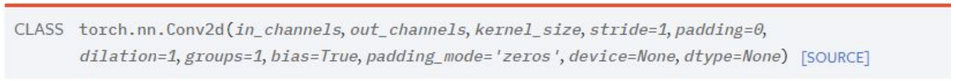
Carnegie Mellon University

Conventional convolution



- **M** input channels, **N** output channels
- **N** independent **MxKxK** 3D filters which span all M input channels
- Each filter produces one output channel

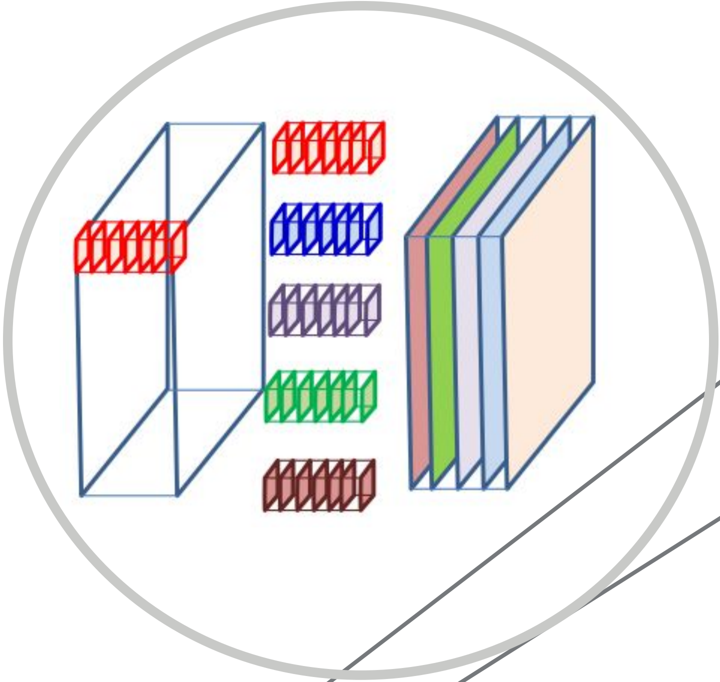
Total : **$N*M*K^2$** parameters



```
CLASS  torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
       dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)  [SOURCE]
```

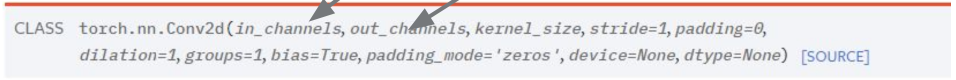
Carnegie Mellon University

Conventional convolution



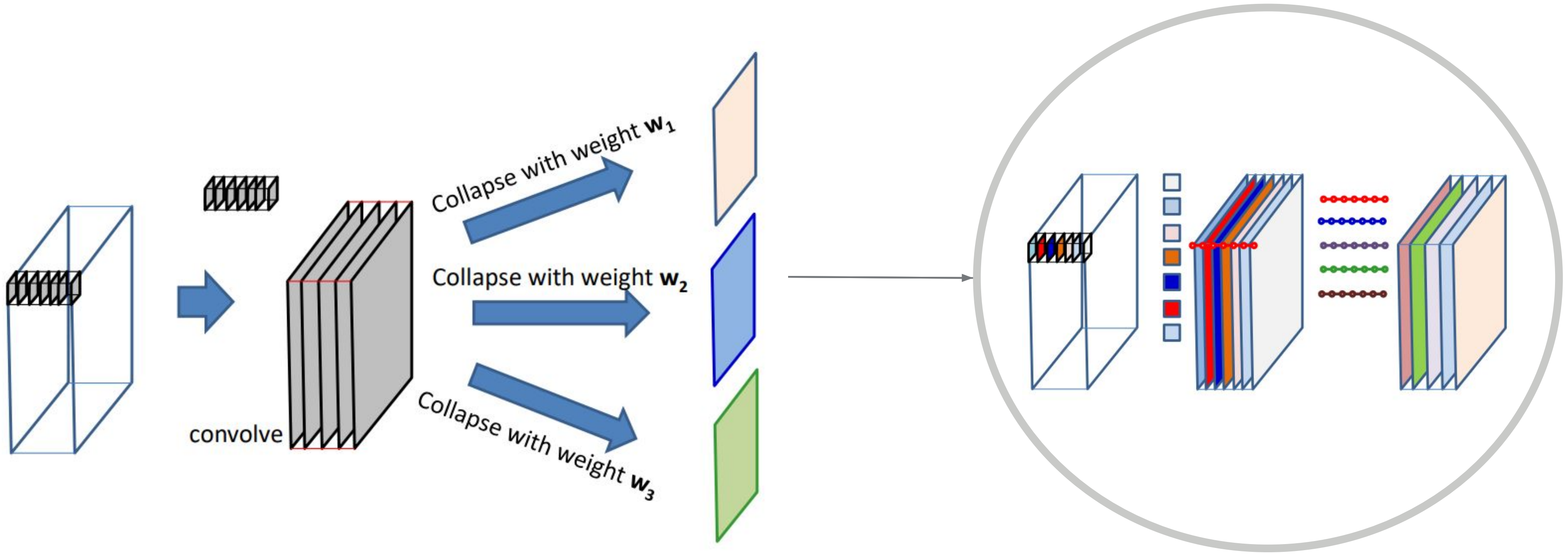
- ❑ M input channels, N output channels
- ❑ N independent MxKxK 3D filters which span all M input channels
- ❑ Each filter produces one output channel

Total : **N*M*K^2** parameters

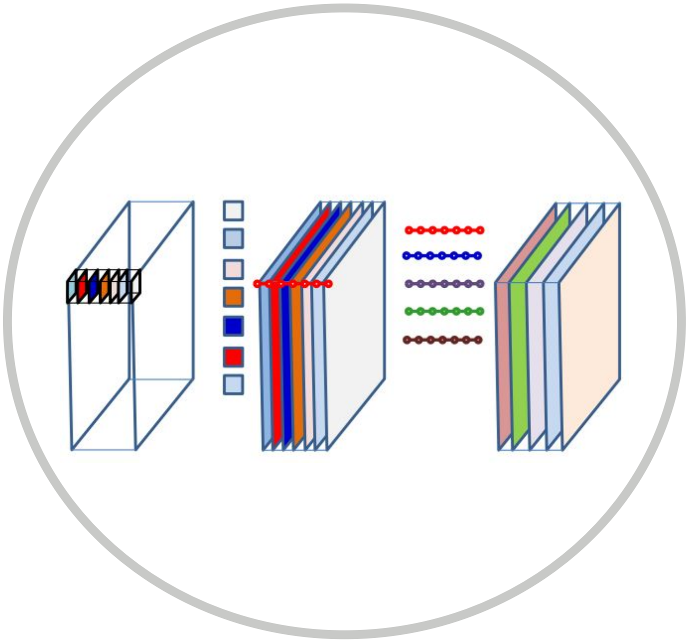


```
CLASS  torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
       dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)  [SOURCE]
```

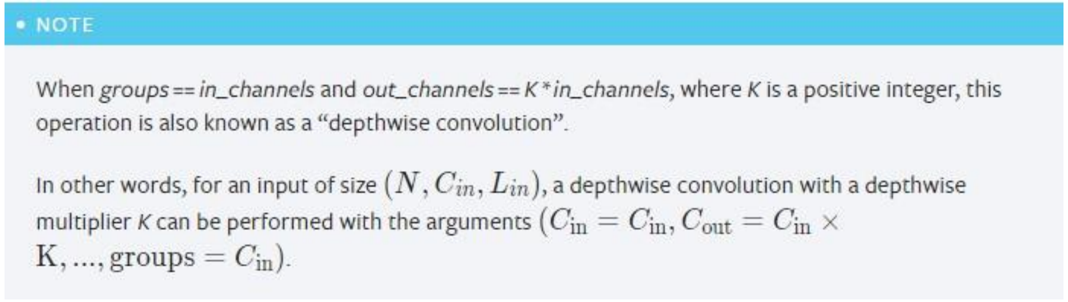
Depthwise separable convolution


Collapse with weight $\mathbf{w}_1$
Collapse with weight $\mathbf{w}_2$
Collapse with weight $\mathbf{w}_3$
convolve

Depthwise separable convolution



- ❑ Apply convolution step once
- ❑ Replace summation by a weighted sum across channels



• NOTE

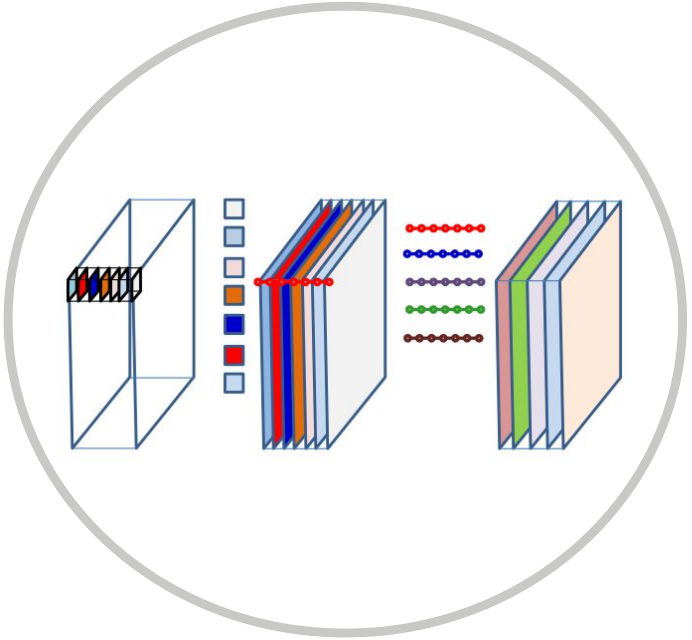
When *groups* == *in_channels* and *out_channels* == *K* * *in_channels*, where *K* is a positive integer, this operation is also known as a "depthwise convolution".

In other words, for an input of size (N, C_{in}, L_{in}), a depthwise convolution with a depthwise multiplier *K* can be performed with the arguments $(C_{\text{in}} = C_{\text{in}}, C_{\text{out}} = C_{\text{in}} \times \text{K}, ..., \text{groups} = C_{\text{in}})$.

Pytorch Docs: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Carnegie Mellon University

Depthwise separable convolution

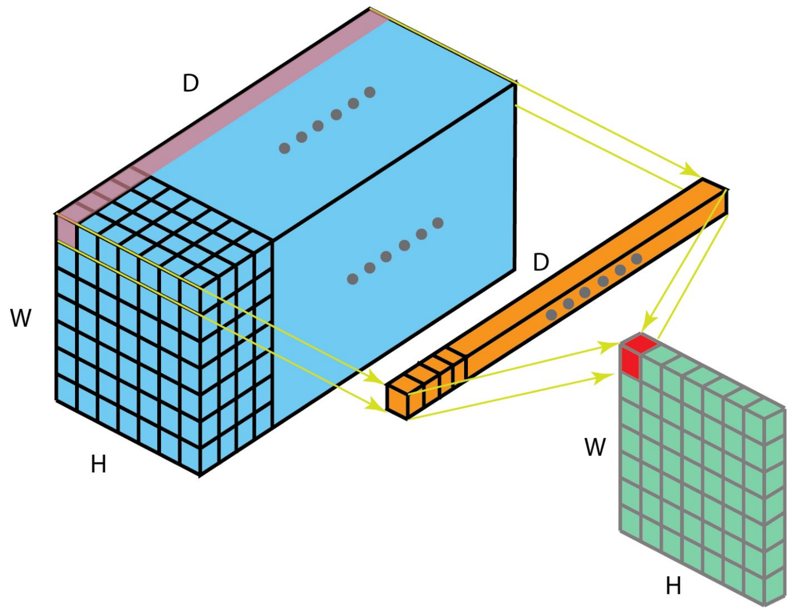


- M input channels, N output channels in 2 stages:
- **Stage 1**: aka Filtering
  - **M** independent **KxK** 2D filters, one per input channel
  - Each filter applies to only one input channel
  - **# of output channels = # input channels**
- **Stage 2**: aka Combining - Point-wise convolution
  - N **Mx1x1** 1D filters
  - Each applies to one 2D location across all M input channels

Total NM + MK^2 parameters

Carnegie Mellon University

Depthwise separable convolution: Combining stage



Stage 2

- M input channels, N output channels in 2 stages:
- **Stage 1**: aka Filtering
  - **M** independent **KxK** 2D filters, one per input channel
  - Each filter applies to only one input channel
  - **# of output channels = # input channels**
- **Stage 2**: aka Combining - Point-wise convolution
  - N **Mx1x1** 1D filters
  - Each applies to one 2D location across all M input channels

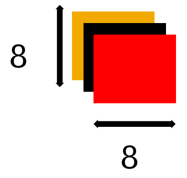
Total NM + MK^2 parameters

Source: https://blog.csdn.net/shiner_chen/article/details/124319246

Carnegie Mellon University

Depthwise separable convolution

8x8x3 input image (in_channels = 3)



Regular conv

5x5x3 regular conv kernel

Depthwise conv

Filtering

🟩	**5x5x1 kernel**
🟪	**5x5x1 kernel**
🟦	**5x5x1 kernel**

Combining

1x1x3 regular conv kernel

- Let **out_channels = 256** (# of desired filters)
  - With **kernel_size = 5**
- Using conventional convolution **(5x5x3 kernel)**:
  - Requires:

    (8x8)*(5x5x3)*(256) = **1, 228, 800** parameters
- Using Depthwise convolution
  - **Filtering stage :** 3* (5x5x1) kernel

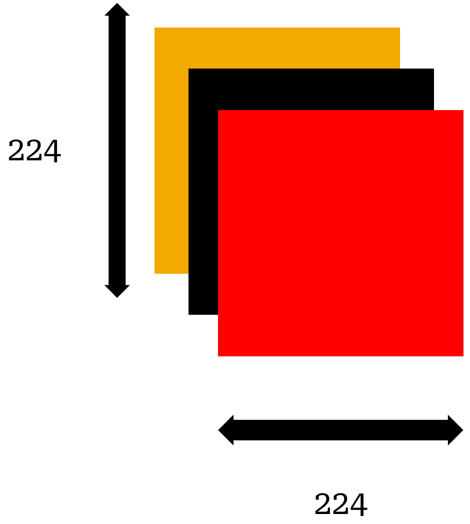
    Requires: (8x8)*(5x5x1)*3 = **4, 800** parameters
  - **Combining stage: 256*(1x1x3)**

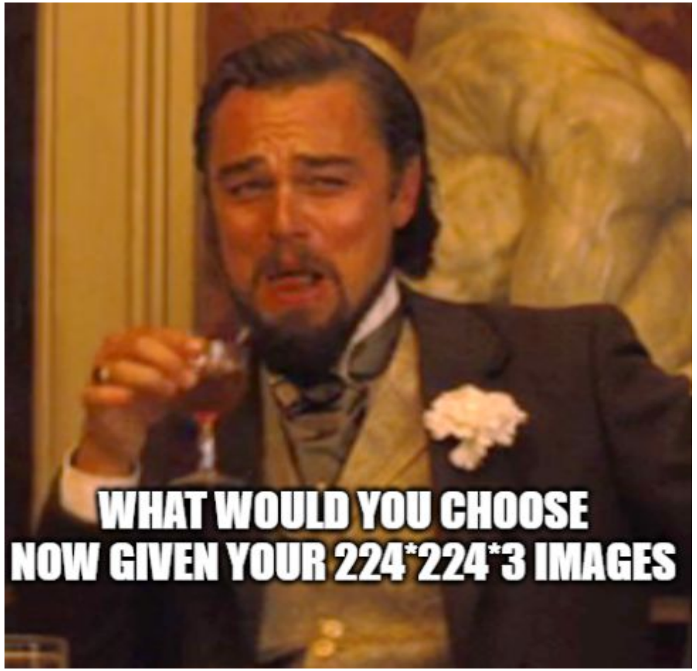
    Requires: (8x8)*(1x1x3)*(256) = **49, 152** parameters

    In total: **53, 952** parameters

Carnegie Mellon University

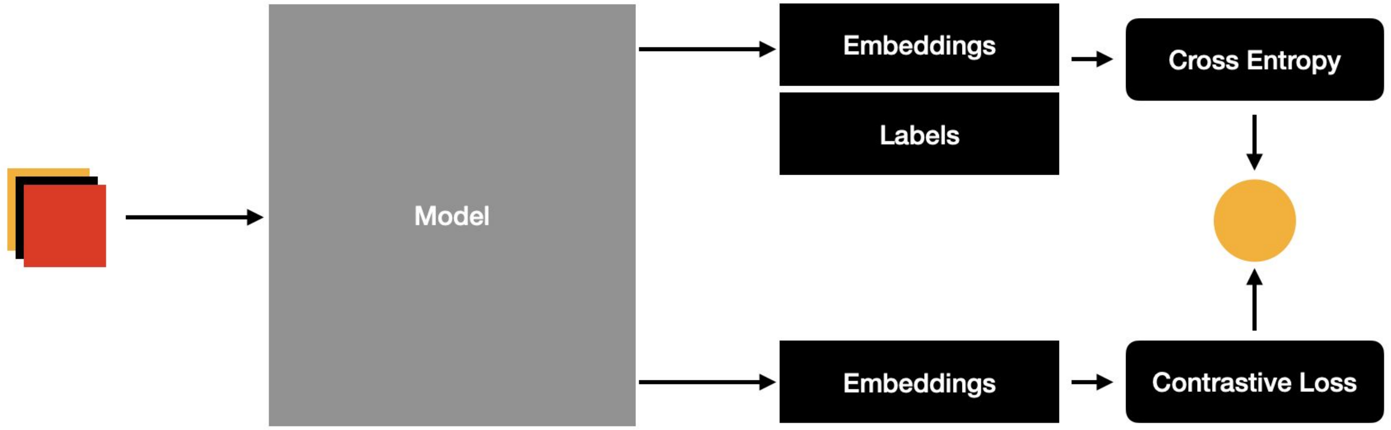
Depthwise separable convolution





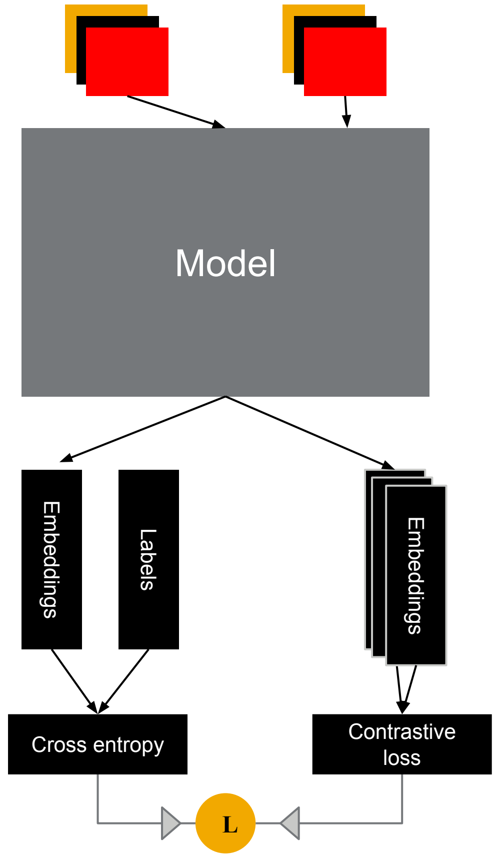
4. Verification optimized approaches

Approach 1 - Joint Loss Optimization



Carnegie Mellon University

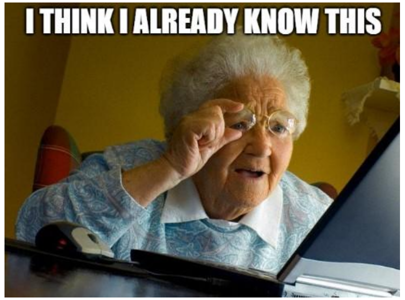
Approach 1: Joint loss optimization

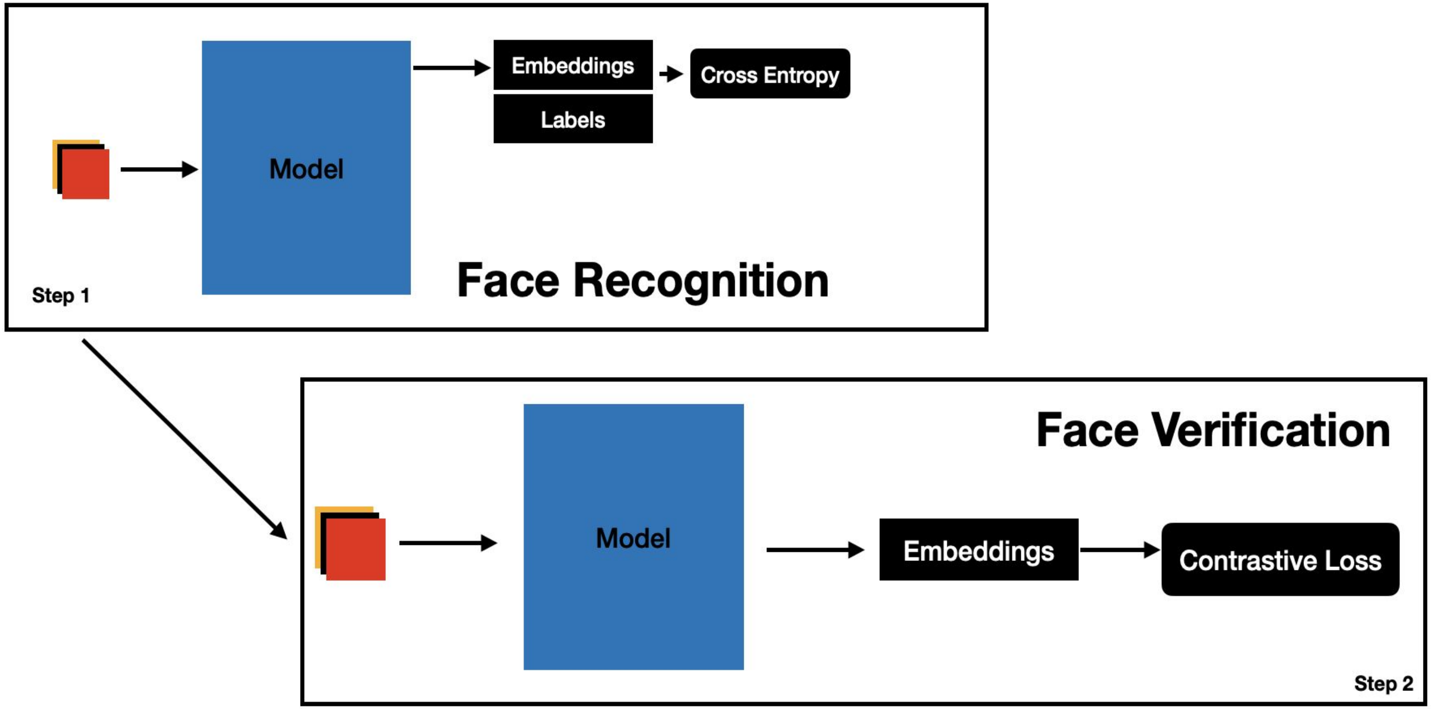


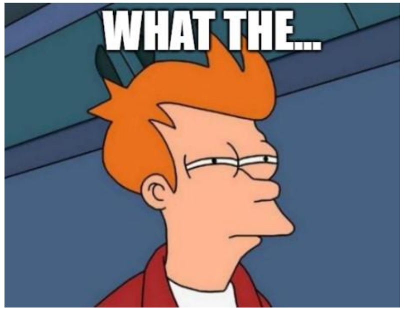
```
logits= model(x, return_feats=False)  # → 7_000 features

# get anchor, positive and negative from TripletDataset
anchor_emb = model(anchor, return_feats=True)
positive_emb = model(positive, return_feats=True)
negative_emb = model(negative, return_feats=True)

loss_1 = CrossEntropyLoss(logits, targets)
loss_2 = TripletLoss(anchor_emb, positive_emb, negative_emb)

L = w1*loss_1 + w2*loss_2 # e.g w1 = 0.7 and w2 = 0.3
```

Carnegie Mellon University

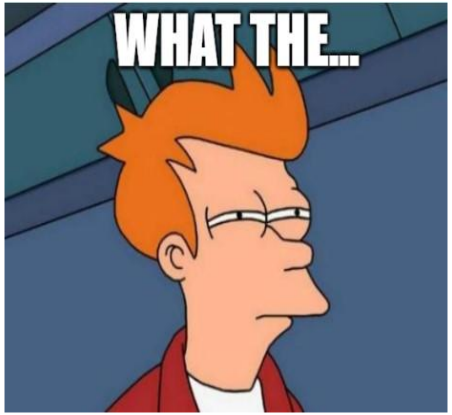
Approach 2: Sequential

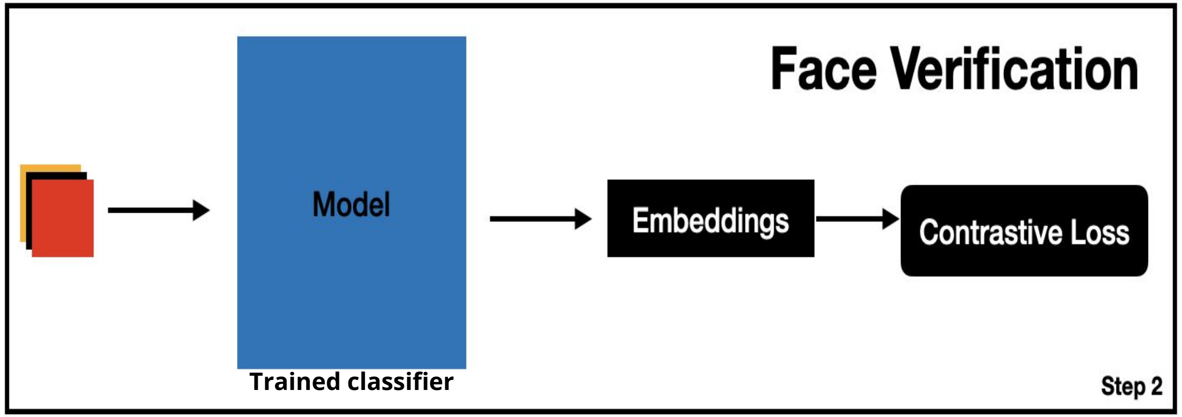




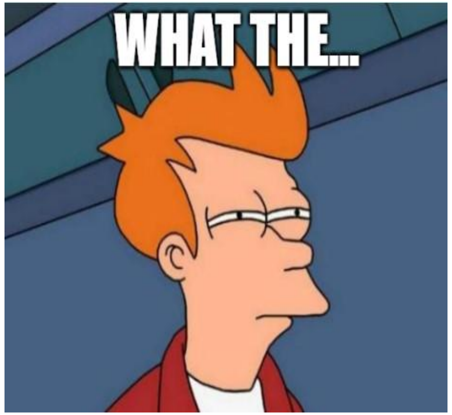


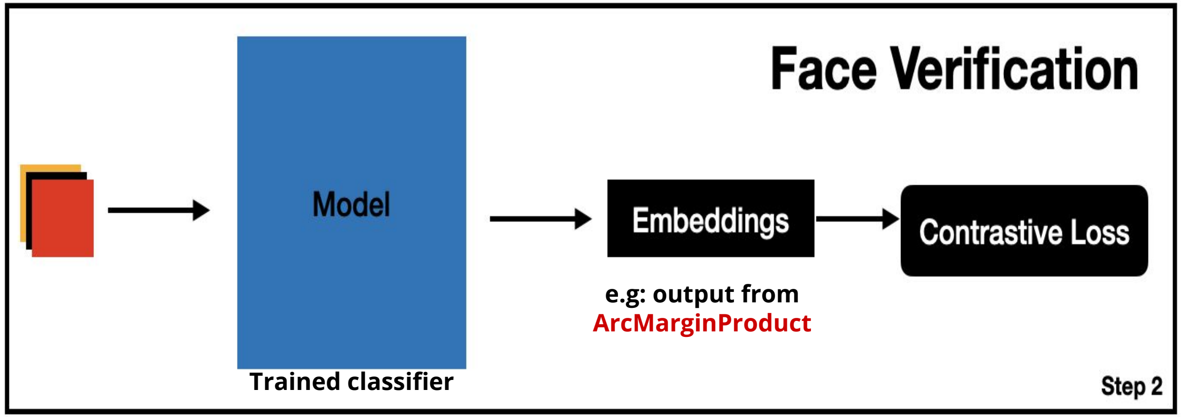
Approach 2: Sequential





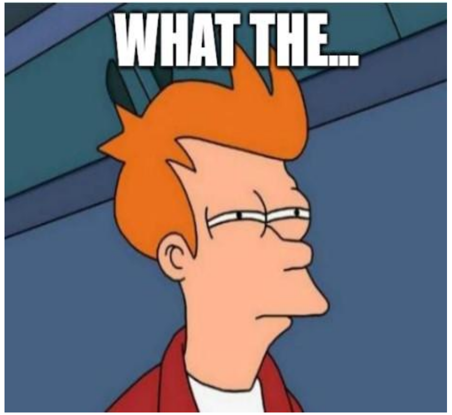
Approach 2: Sequential

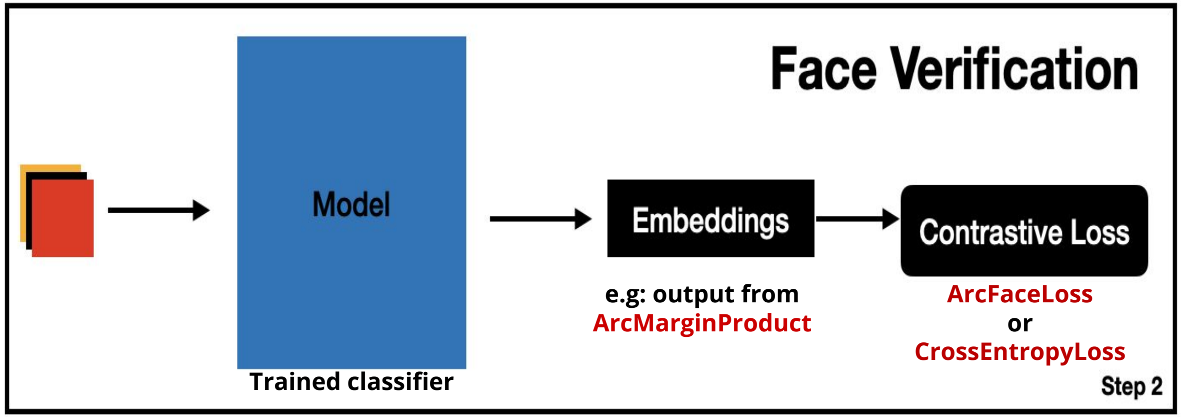




Some contrastive losses and more: https://kevinmusgrave.github.io/pytorch-metric-learning/losses/

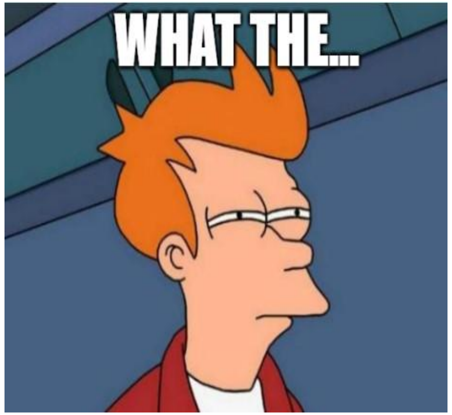
Approach 2: Sequential

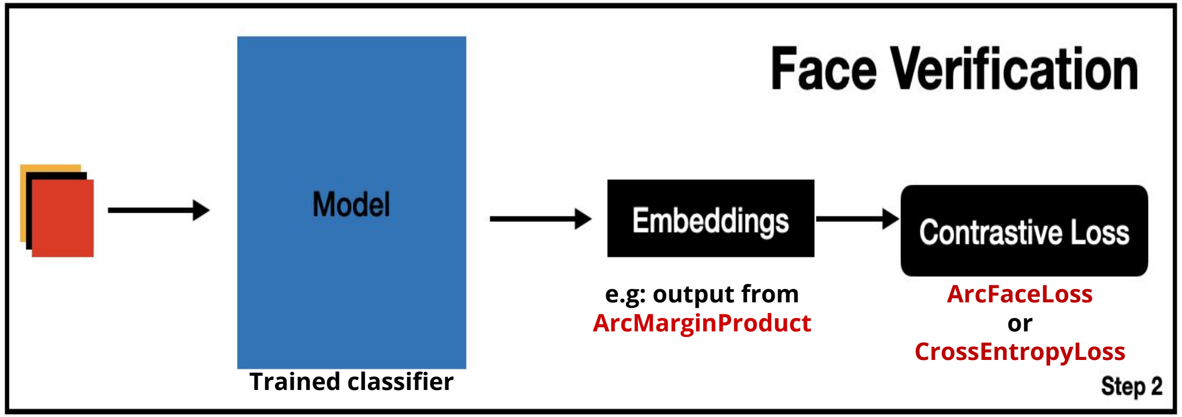




Some contrastive losses and more: https://kevinmusgrave.github.io/pytorch-metric-learning/losses/

Approach 2: Sequential





Ideally, ArcMarginProduct would be an additional layer to your classification network.

Some contrastive losses and more: https://kevinmusgrave.github.io/pytorch-metric-learning/losses/

Approach 2: Sequential

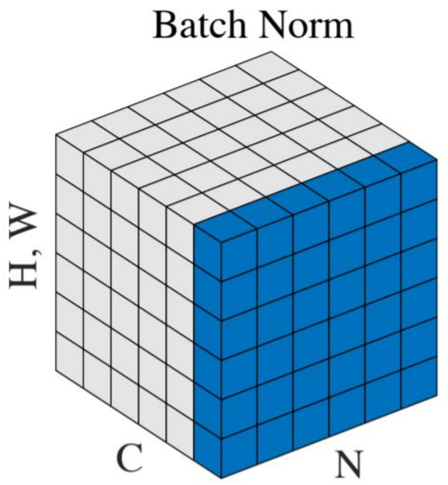
Ideally, ArcMarginProduct would be an additional layer to your classification network.

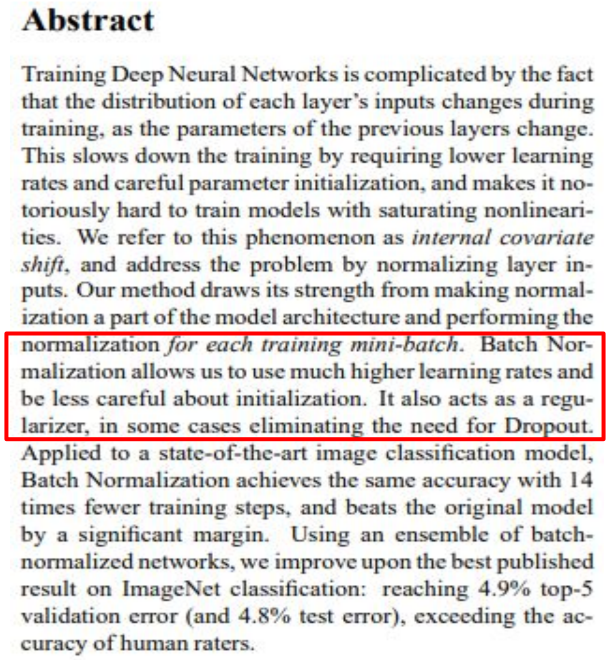
```
class VerificationNetwork(torch.nn.Module):
    def __init__(self, num_classes=7000):
        super().__init__()
        self.backbone = model.load_state_dict(torch.load(# TODO))
        self.arcFaceLayer = ArcMarginProduct(
                                embedding_size=# TODO,
                                n_classes=num_classes
        )

    def forward(self, x):
        feats = self.backbone(x, return_feats=True)
        out = self.arcFaceLayer(feats)
        return out
```

Carnegie Mellon University

Student after HW2P2

Some tips: Normalization





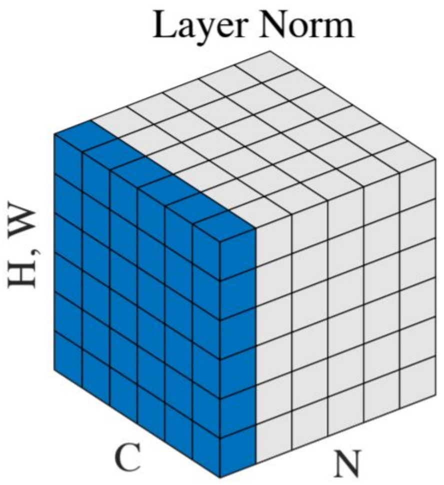
Abstract

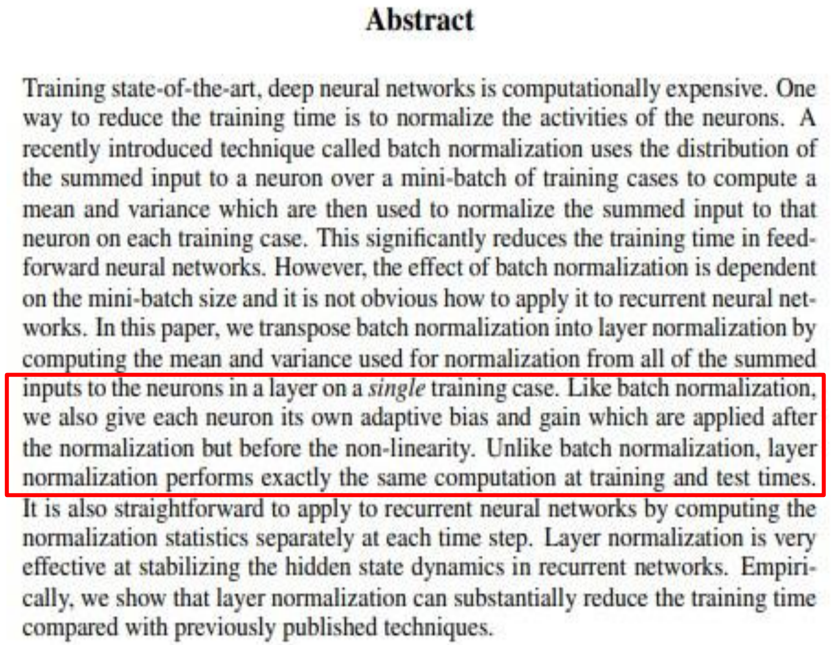
Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

BatchNorm paper: https://arxiv.org/pdf/1502.03167.pdf

Carnegie Mellon University

Some tips: Normalization





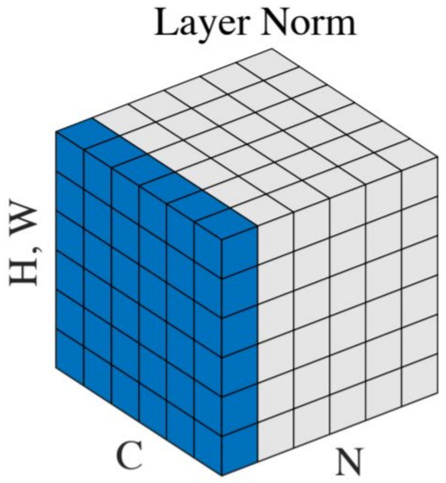
Abstract

Training state-of-the-art, deep neural networks is computationally expensive. One way to reduce the training time is to normalize the activities of the neurons. A recently introduced technique called batch normalization uses the distribution of the summed input to a neuron over a mini-batch of training cases to compute a mean and variance which are then used to normalize the summed input to that neuron on each training case. This significantly reduces the training time in feed-forward neural networks. However, the effect of batch normalization is dependent on the mini-batch size and it is not obvious how to apply it to recurrent neural networks. In this paper, we transpose batch normalization into layer normalization by computing the mean and variance used for normalization from all of the summed inputs to the neurons in a layer on a *single* training case. Like batch normalization, we also give each neuron its own adaptive bias and gain which are applied after the normalization but before the non-linearity. Unlike batch normalization, layer normalization performs exactly the same computation at training and test times. It is also straightforward to apply to recurrent neural networks by computing the normalization statistics separately at each time step. Layer normalization is very effective at stabilizing the hidden state dynamics in recurrent networks. Empirically, we show that layer normalization can substantially reduce the training time compared with previously published techniques.

LayerNorm paper: https://arxiv.org/pdf/1607.06450.pdf

Carnegie Mellon University

Some tips: Normalization



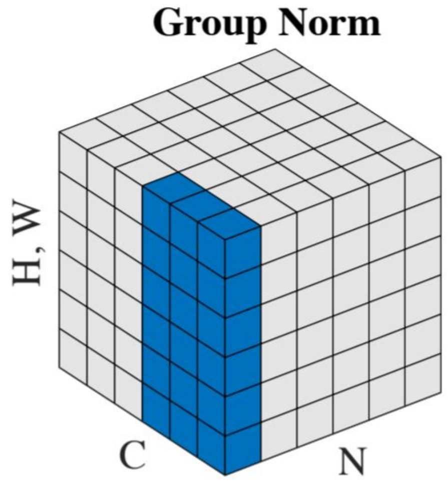
$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

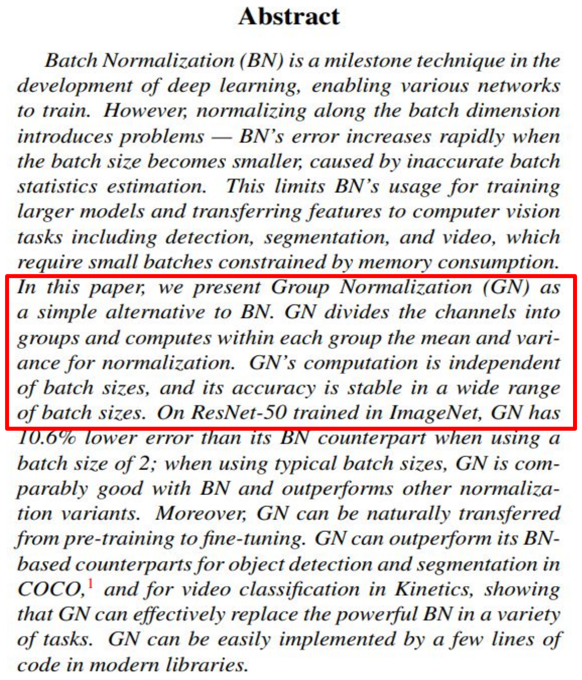
- ❑ $\boldsymbol{H}$ denotes the number of hidden units in a layer
- ❑ No constraints on the size of the mini-batches
- ❑ All the hidden units in a layer share the same normalization terms $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$

LayerNorm paper: https://arxiv.org/pdf/1607.06450.pdf

Carnegie Mellon University

Some tips: Normalization



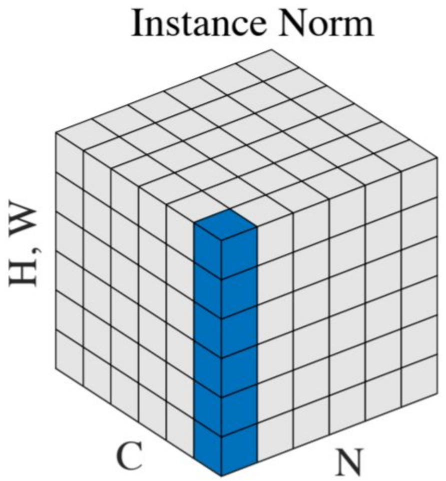


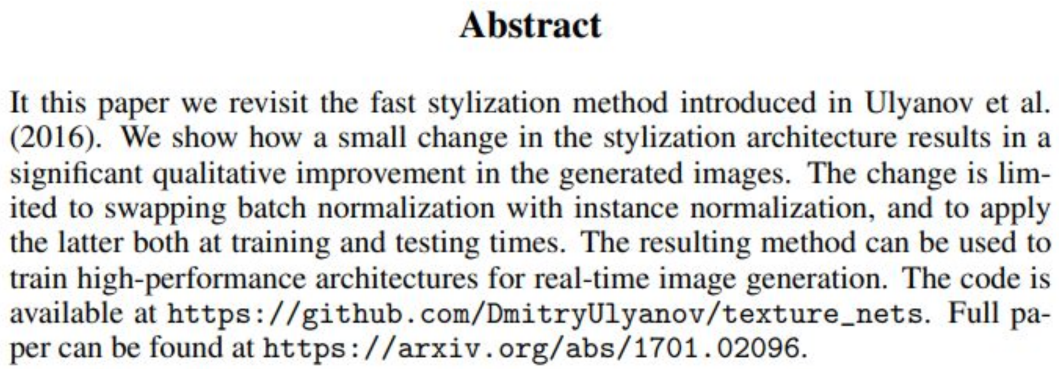
Abstract

Batch Normalization (BN) is a milestone technique in the development of deep learning, enabling various networks to train. However, normalizing along the batch dimension introduces problems — BN's error increases rapidly when the batch size becomes smaller, caused by inaccurate batch statistics estimation. This limits BN's usage for training larger models and transferring features to computer vision tasks including detection, segmentation, and video, which require small batches constrained by memory consumption. In this paper, we present Group Normalization (GN) as a simple alternative to BN. GN divides the channels into groups and computes within each group the mean and variance for normalization. GN's computation is independent of batch sizes, and its accuracy is stable in a wide range of batch sizes. On ResNet-50 trained in ImageNet, GN has 10.6% lower error than its BN counterpart when using a batch size of 2; when using typical batch sizes, GN is comparably good with BN and outperforms other normalization variants. Moreover, GN can be naturally transferred from pre-training to fine-tuning. GN can outperform its BN-based counterparts for object detection and segmentation in COCO,[1] and for video classification in Kinetics, showing that GN can effectively replace the powerful BN in a variety of tasks. GN can be easily implemented by a few lines of code in modern libraries.

GroupNorm paper: https://arxiv.org/pdf/1803.08494.pdf

Some tips: Normalization




Abstract

It this paper we revisit the fast stylization method introduced in Ulyanov et al. (2016). We show how a small change in the stylization architecture results in a significant qualitative improvement in the generated images. The change is limited to swapping batch normalization with instance normalization, and to apply the latter both at training and testing times. The resulting method can be used to train high-performance architectures for real-time image generation. The code is available at `https://github.com/DmitryUlyanov/texture_nets`. Full paper can be found at `https://arxiv.org/abs/1701.02096`.

- InstanceNorm removes the effect of contrast in images
- Much useful in stylization (image generation)
  - One can argue that the result of stylization should not, in general, depend on the contrast of the content image

InstanceNorm paper: https://arxiv.org/pdf/1607.08022.pdf

Carnegie Mellon University

Label Smoothing

- Deep Learning Models undergo a problem of Overfitting and Overconfidence.
- Label Smoothing is a technique that can help us solve the problem of Overconfidence.

What is Overconfidence?
For each sample, the model predicts outcomes with higher probabilities than the accuracy over the entire dataset.
This is a poorly calibrated model.
For example, it may predict 0.9 for inputs where the accuracy is only 0.6.

Carnegie Mellon University

Example

Without Label Smoothing

Suppose we have K = 3 classes, and our label belongs to the 1st class. Logit Vector z = [a, b, c]

Label vector y = [1, 0, 0] (one-hot encoded)

Gradient of Loss = softmax(z) – y

Our model will make $a \gg b$ and $a \gg c$

z = [10, 0, 0]

softmax(z) = [0.9999, 0, 0]

Label smoothing

```
y_ls = (1 - α) * y_hot + (α / K)
```

Example: With Label Smoothing (*α = 0.1)*

```
y_ls = [0.9333, 0.0333, 0.0333]
```

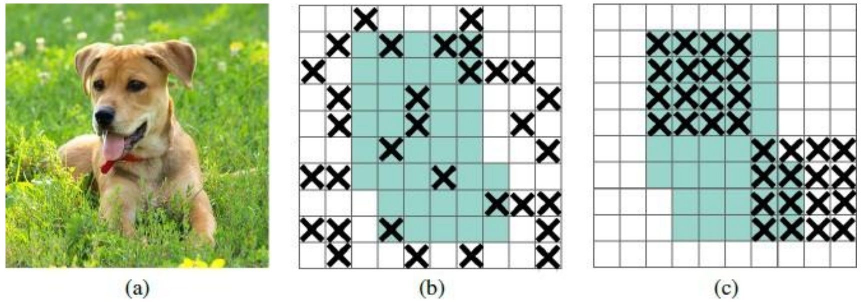
This would result into the logits z = [3.3332, 0, 0]

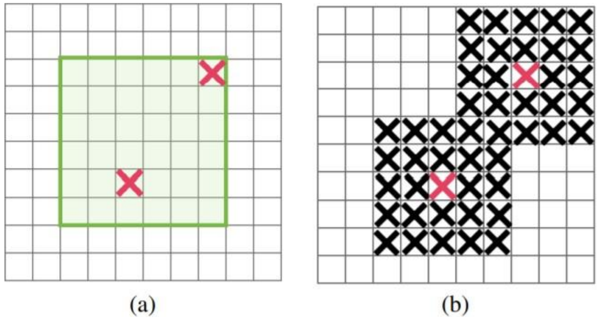
```
softmax(z) = [0.9333, 0.0333, 0.0333]
```

https://arxiv.org/pdf/1812.01187.pdf
https://arxiv.org/abs/1812.01187
https://github.com/ankandrew/online-label-smoothing-pt

Some tips: DropBlock

- Dropout usually works better with Fully Connected Networks
- Dropout has not proven to be useful in CNNs because of the spatial correlation between the activation outputs
- DropBlock is a regularization technique that has proven to be useful for CNNs
- It is a structured form of dropout that drops contiguous regions and not just random pixels





Paper: https://arxiv.org/pdf/1810.12890.pdf

Pytorch docs: https://pytorch.org/vision/main/generated/torchvision.ops.drop_block2d.html

Some tips: And…

5. Run through the starter notebook