Carnegie Mellon University

11-785 Introduction to Deep Learning

- Spring 2023 -

Recitation 6: CNN Classification

Prakruthi Pradeep
Ewuzie Paul

Objectives

1. What is Classification?
2. Description of our Dataset
3. Types of Convolutions
4. Tips and run-through of HW2P2

The Classification Problem

- How can I determine which of my emails are spam or not spam?
- How can I identify positive and negative feedback from tons of customers review
- **This picture contains an image of what or who?**
- How did Face unlock know I am the owner!!!? (more on verification)
- What digit have I written on this paper?

What is Classification?

- A supervised learning method where the model tries to predict the label/class for a given input data

In Context...

- Given an image, figure out which class the image belongs to or rather the person in the image

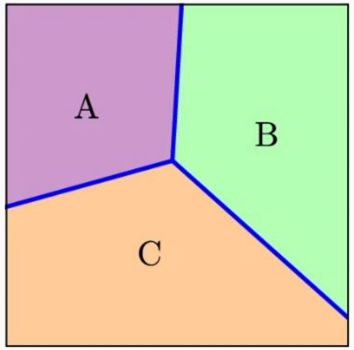
Categories

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

Closed Set & Open Set

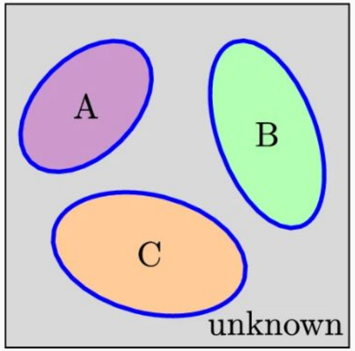
Closed Set

- K known classes are present during training and testing
- Learns decision boundaries that divide the feature space into K parts

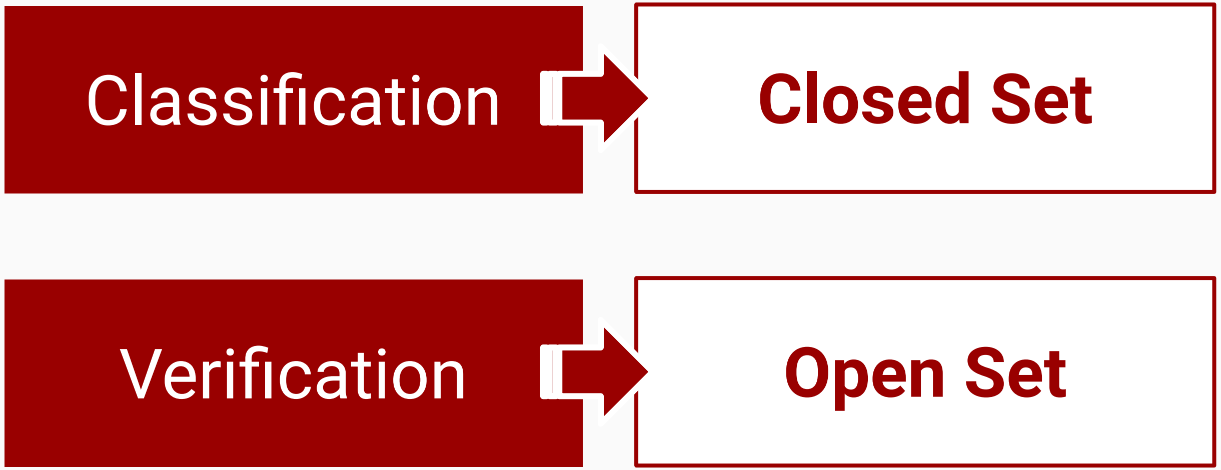


Open set

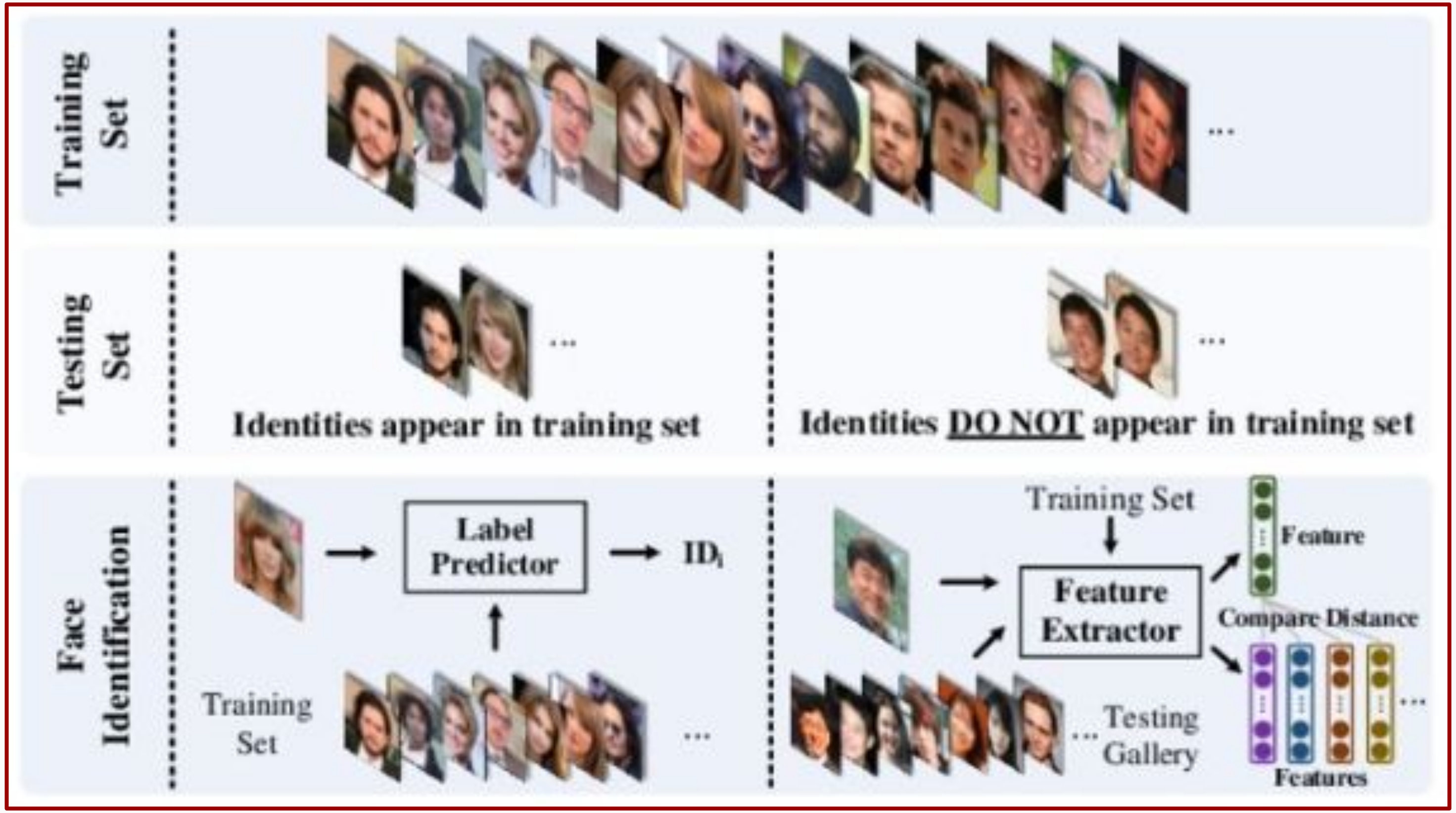
- K known classes during training but K known and U unknown classes present during inference
- Tight decision boundary around the K classes are learned



Closed Set or Open Set

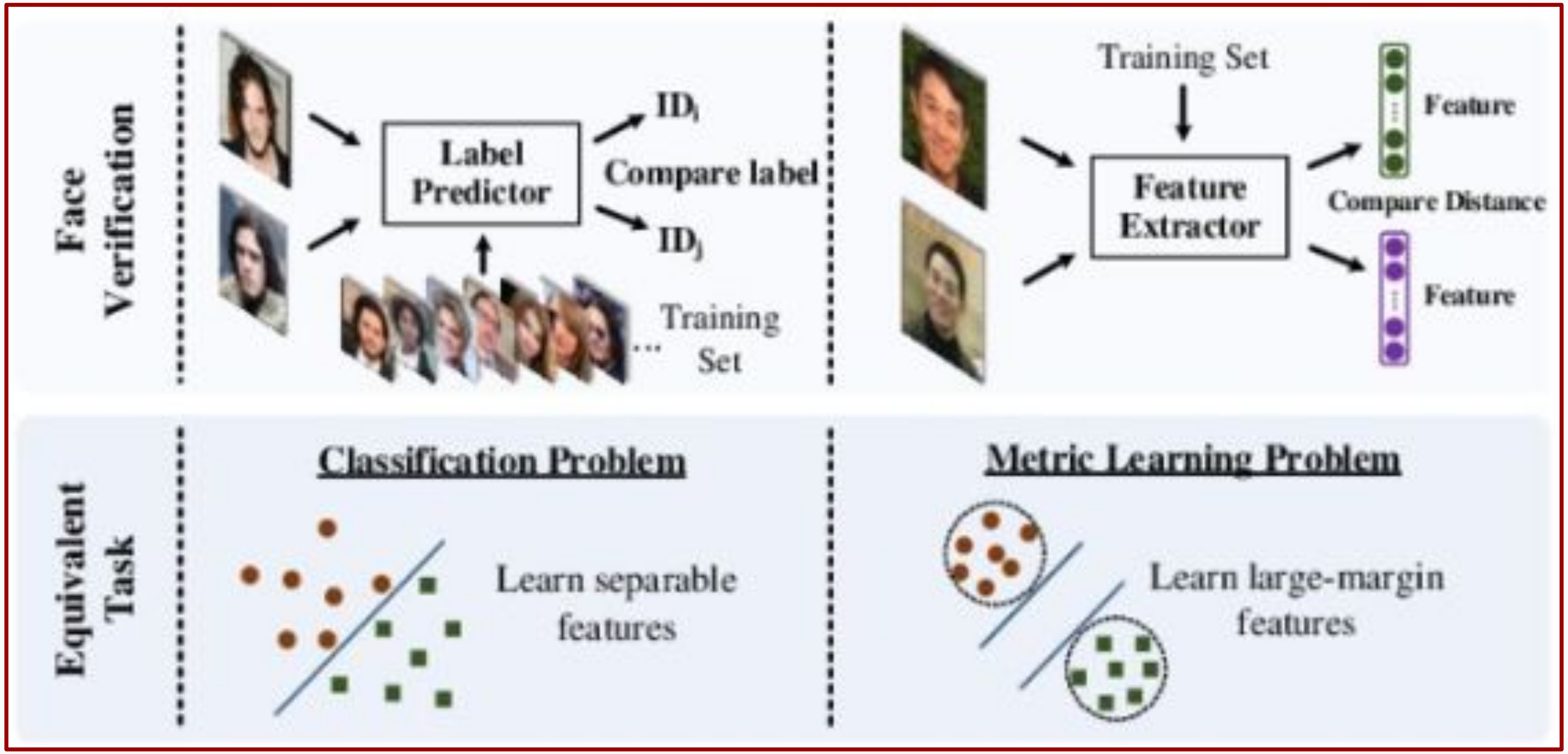


Closed Set vs. Open Set (Classification)



Closed Set **Open Set**

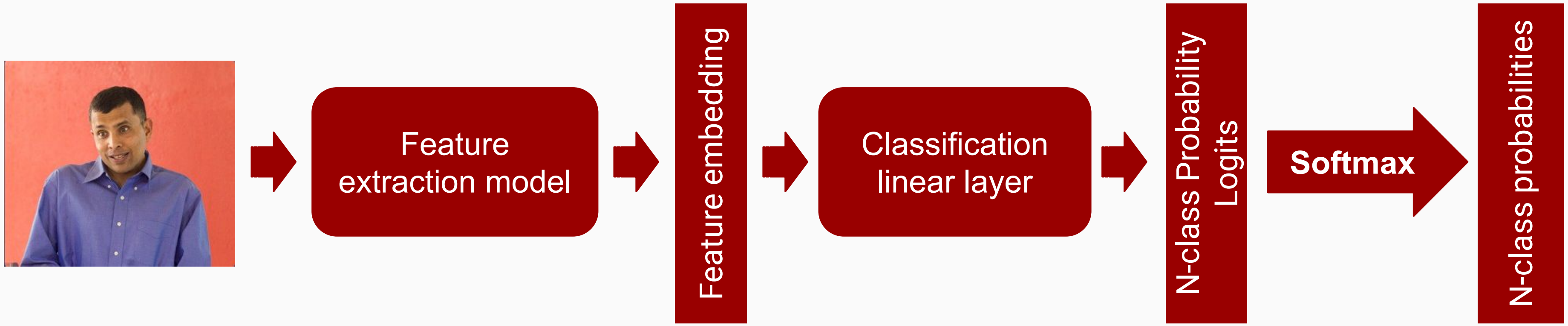
Closed Set vs. Open Set (Verification)



Closed Set **Open Set**

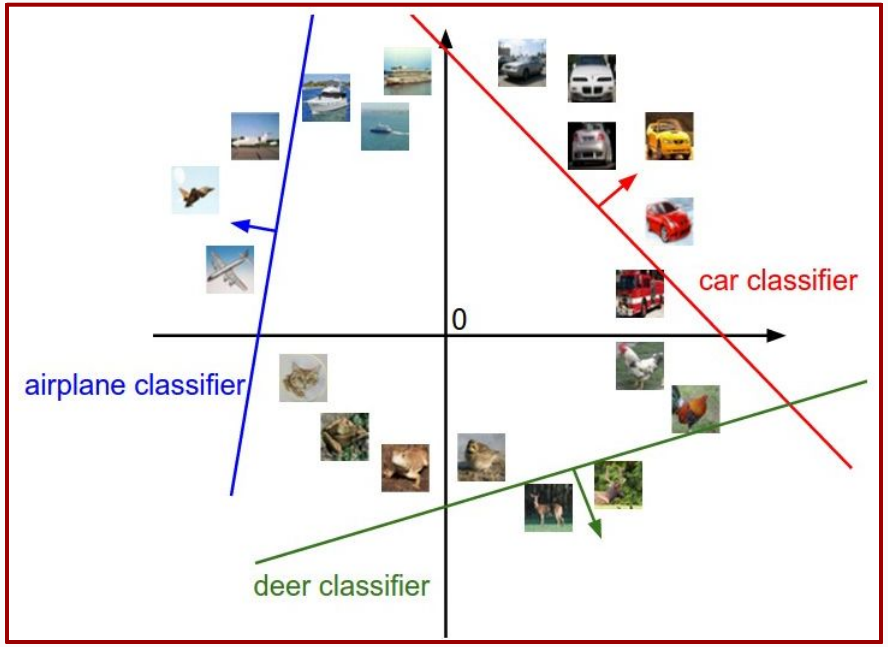
Problem Statement

Definition	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.
Task	Identify the person in a picture; about 7000 unique images therefore, Multi-class classification



Training for Classification

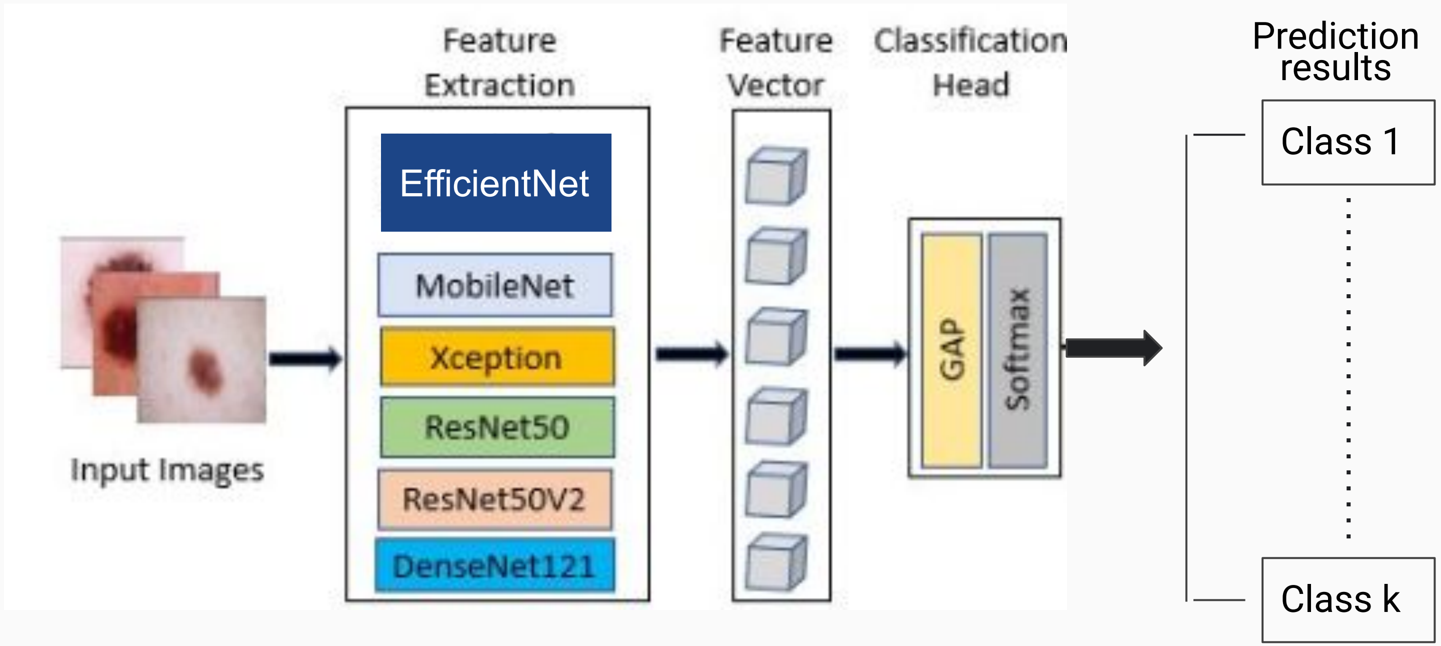
Task	Identify the person in a picture; about 7000 unique images therefore, Multi-class classification



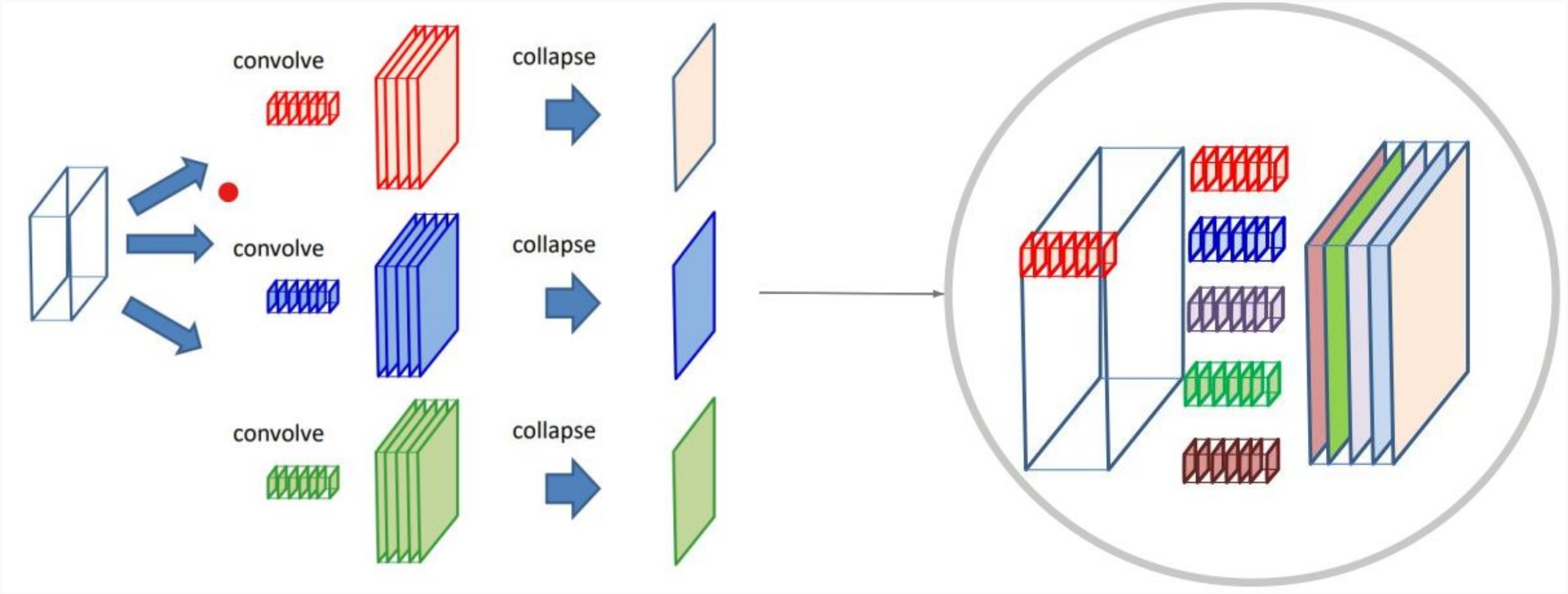
- Feature extraction (CNNs)
- Feature embeddings (creates important features which are separable)
- Generate Logits (unnormalized probabilities)
- Softmax (normalizes probabilities between 0 and 1)

Training for Classification

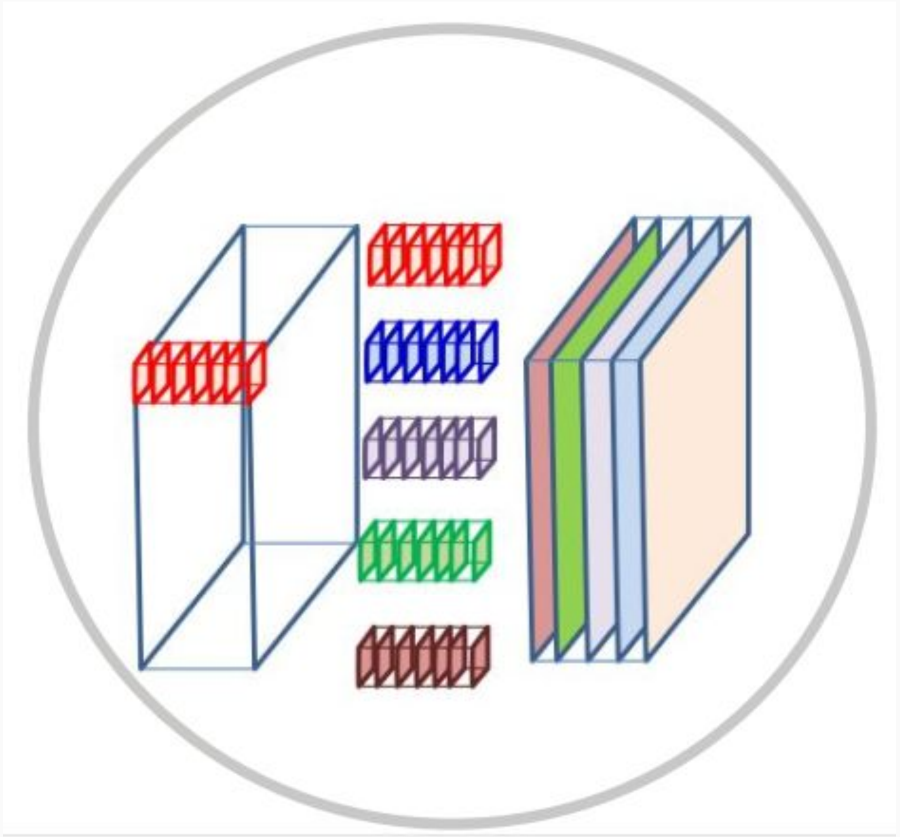
Task	Identify the person in a picture; about 7000 unique images therefore, Multi-class classification



Conventional Convolution

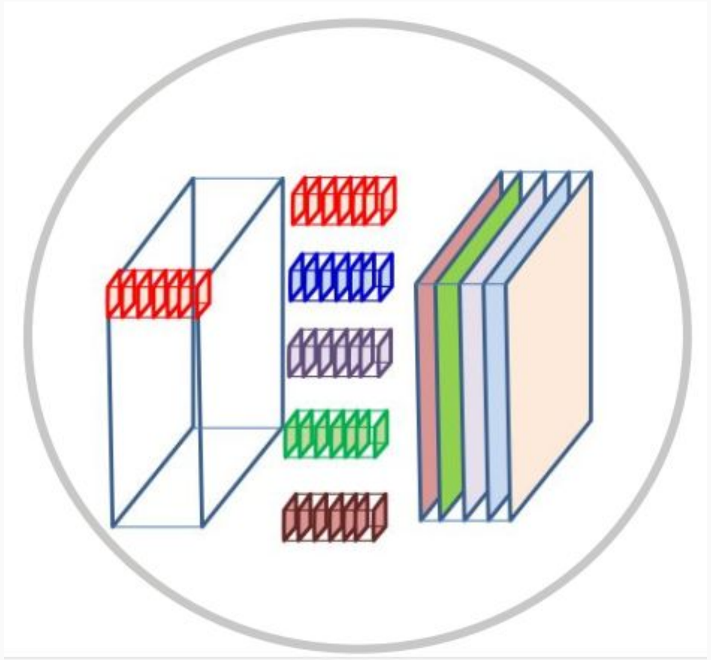


Conventional Convolution

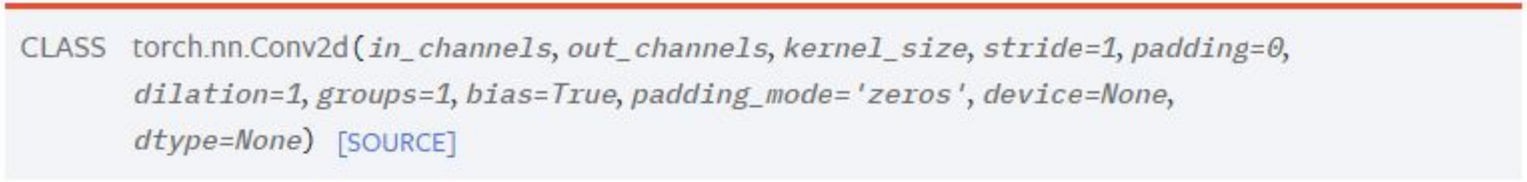


- ❑ Each filter (red, blue, ..) has N layers which correspond to the N number of input channels
- ❑ Each layer of each filter scans a corresponding input channel to produce a convolved map
- ❑ A filter with N layer will produce N convolved maps
- ❑ The N convolved maps produced by a filter is added together to produce an output channel
- ❑ Consequently, number of output channels determine the number of filters for convolving

Conventional Convolution

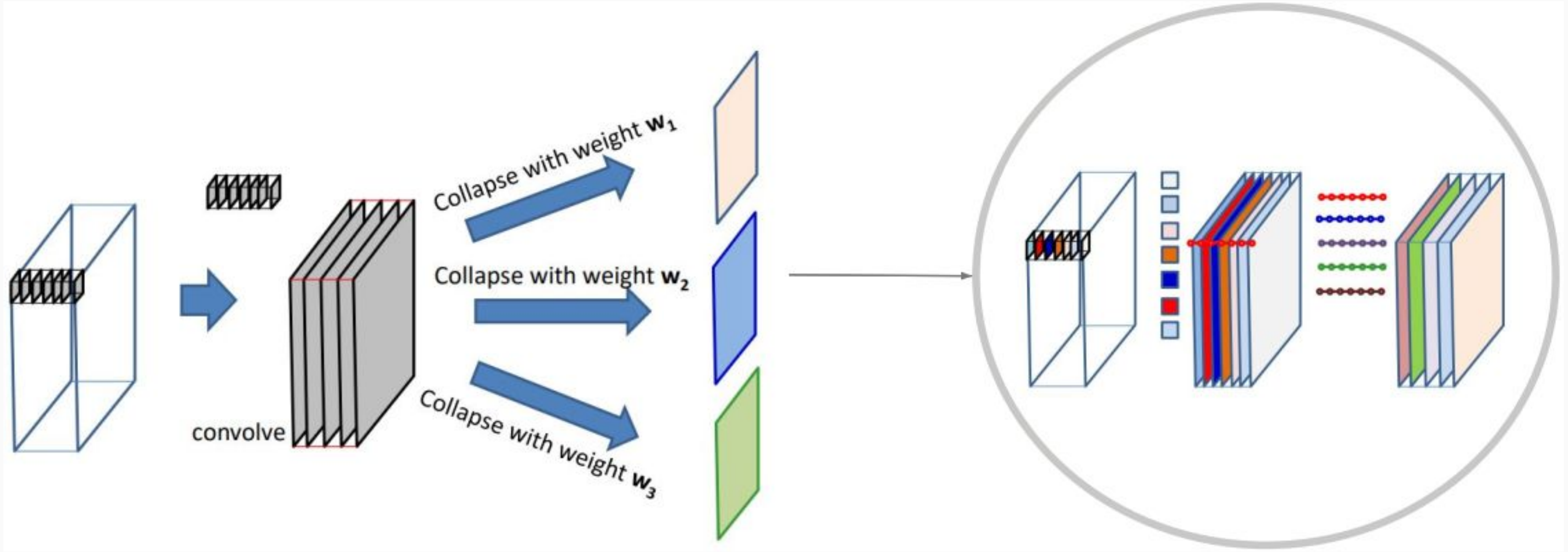


- ❑ **N** input channels, **M** output channels
- ❑ **M** independent filters with **N** layers each. Assuming each layer has a size **K * K**
- ❑ Each filter will produce an output channel, therefore **M** output channels
- ❑ **Total Parameters: N * M * K^2**

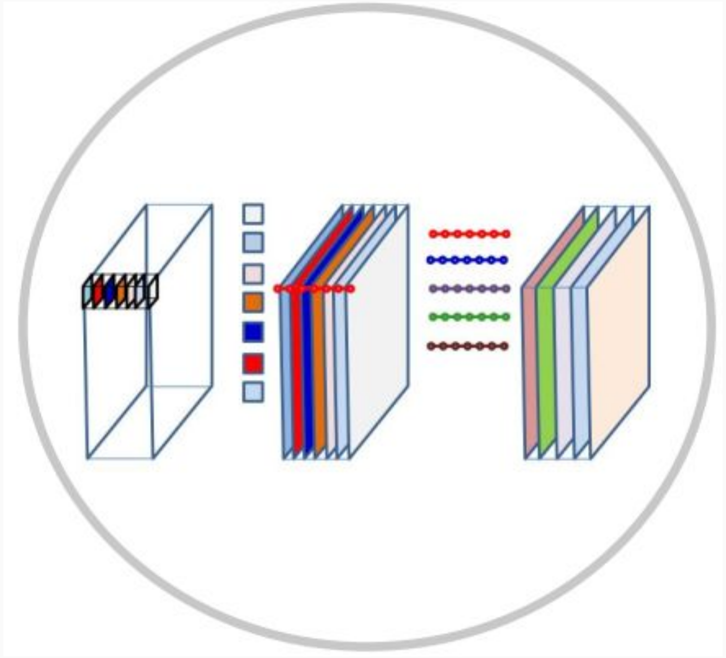


```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
      dilation=1, groups=1, bias=True, padding_mode='zeros', device=None,
      dtype=None) [SOURCE]
```

Depthwise Separable Convolution

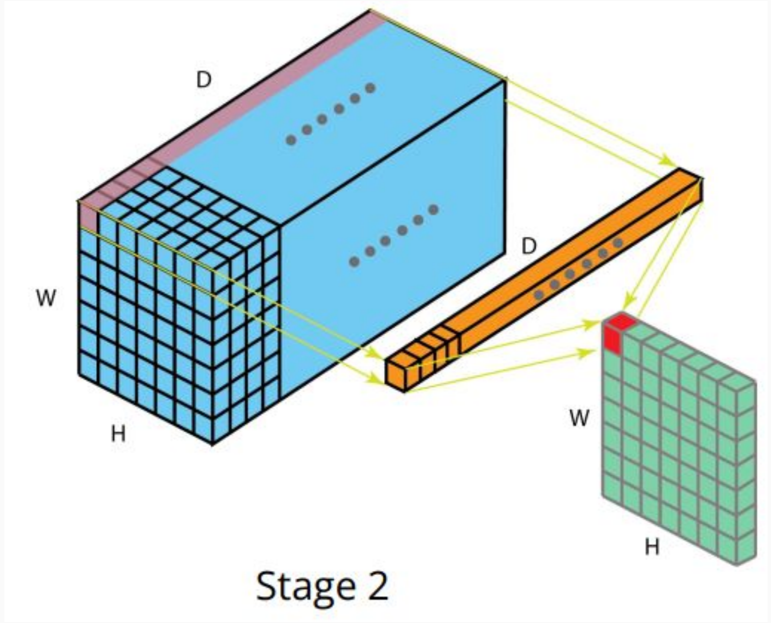

convolve
Collapse with weight $\mathbf{w}_1$
Collapse with weight $\mathbf{w}_2$
Collapse with weight $\mathbf{w}_3$

Depthwise Separable Convolution: Filtering

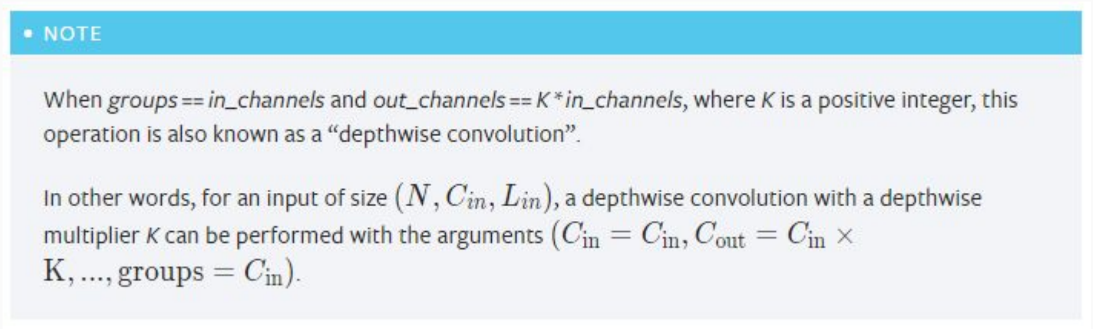


- **M input channels** and **N output channels in 2 stages:**
- **Stage 1: Filtering**
  - We need M independent KxK 2D filters, one per input channel
  - Each filter convolves with an input channel to produce intermediary output channels
  - # of input channels == # of output channels

Depthwise Separable Convolution: Combining



- **Stage 2: Combining - Point wise convolution**
  - N Mx1x1 filters
  - Each filter (Mx1x1) will be applied on the intermediary output channel a final output channel (just like conventional convolution)
  - We would have a total of N output channels after all N filters have been applied
- **Total parameter: NM + MK^2**



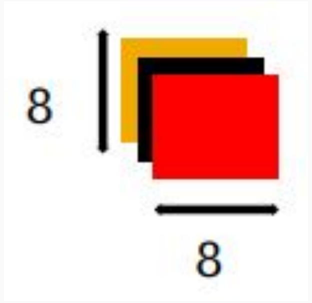
• NOTE

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 $(C_{\text{in}} = C_{\text{in}}, C_{\text{out}} = C_{\text{in}} \times \text{K}, ..., \text{groups} = C_{\text{in}})$.

Depthwise separable vs. Regular (Parameter comparison)

3x8x8 (in_channels = 3)



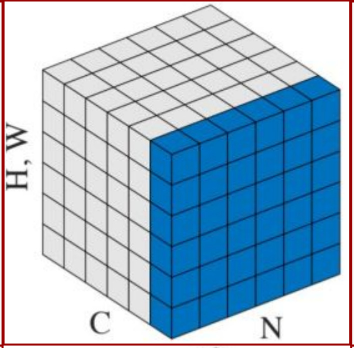
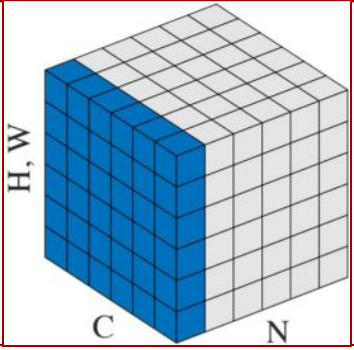
Regular Convolution	**Depthwise convolution**	
🟦 3x5x5 kernel	**Filtering:** 🟦 1x5x5 kernel 🟪 1x5x5 kernel 🟩 1x5x5 kernel	**Combining:** 🟥 3x1x1 kernel

Let:

out_channels = 256
Kernel size = 5

Regular Convolution	**Depthwise convolution**	
Kernel: (3x5x5) **Total Parameters:** (8x8)*(3x5x5)*(256) = **1,228,800**	**Filtering:** **Kernel:** 3*(1x5x5) **Parameters:** (8x8) * 3 * (1x5x5) = **4,800**	**Combining:** **Kernel:** 256*(3x1x1) **Parameters:** (8x8) * 256 *(3x1x1) = **49,152**
	Total Parameters : 4,800 + 49,152 = 53,952	

Basic Tips: Normalization

Normalization			
Batch Normalization		● 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 drop out	BatchNorm Paper
Layer Normalization		● Each neuron has its own adaptive bias and gain ● Layer normalization performs same computation at training and testing	LayerNorm Paper $\mu_l = \frac{1}{d}\sum_{i=1}^{d} x_i$ $\sigma_l^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu_l)^2$

Good to explore: Group Normalization & Instance Normalization

Basic Tips: Label Smoothing

Problem	● Deep neural network models often find themselves falling prey to Overfitting and Over-confidence ● Overconfidence is when the model predicts a very high probability for input making it rigid (Hard label) and less generalizing; A good model should give room for uncertainty for other classes ● For example, if k = 3 classes, with a label belonging to the first class, we will get an output probability result of [0.9999, 0, 0]; this is a poorly calibrated model

Label Smoothing	● Solves the problem of over-confidence by introducing output distribution regularization to the network ● Applies weighted average across the output labels to “smoothen” or soften them ● Increases robustness of model and improves model performance

Basic Tips: Label Smoothing

Label Smoothing:

$$y\_ls = (1 - \alpha) * y\_hot + (\alpha / K)$$

Where:

y_ls = smoothed labels; **y_hot** = Original labels; **α** = smoothing factor; **K** = classes

Example: 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)

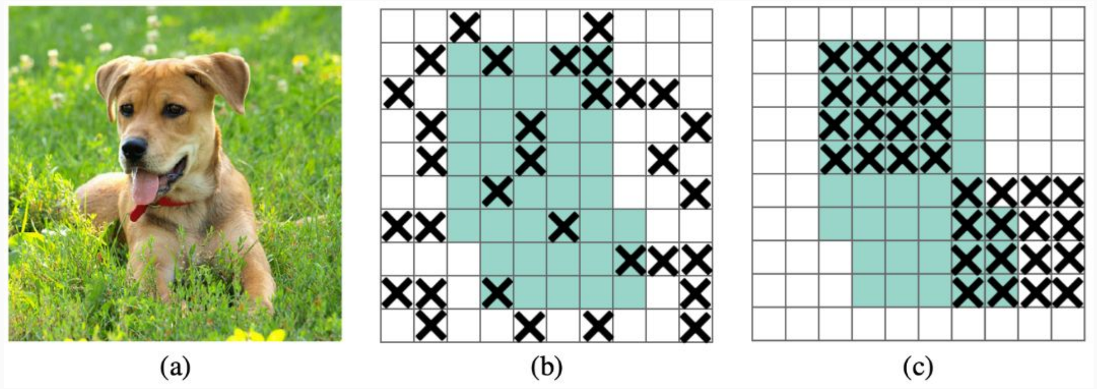
Without Label Smoothing	With Label Smoothing (α = 0.1)
Gradient of Loss = softmax(z) – y	y_ls = [0.9333, 0.0333, 0.0333]
Our model will make $a \gg b$ and $a \gg c$	Gradient of Loss = softmax(z) – y_ls
z = [10, 0, 0]	z = [3.3332, 0, 0]
softmax(z) = [0.9999, 0, 0]	softmax(z) = [0.9333, 0.0333, 0.0333]

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

Basic Tips: DropBlock

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

Homework 2 Part 2 Overview

- **Objective:** To solve an image-based face classification problem using a CNN
- **Scenario at hand:** Recognizing and verifying faces in images.
- **Motivation:** Pictures of the faces have *indeterminacy of position* and CNN's are *position invariant*
- **Problem Type:** 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.
- **Requirement:** The embeddings for the subjects in our vocabulary be linearly separable from each other

What to implement

- **Goal**: To implement a face classifier that can extract feature vectors from face images.
- **Two main parts**: Feature Extractor and Classification Layer
- Learning facial features (e.g., skin tone, hair color, nose size, etc.) from an image of a person’s face and represent them as a fixed-length feature vector called face embedding.

Steps:

- Implement architectures consisting of multiple convolutional layers outputting a feature vector.
- The vector is passed through a linear layer followed by Softmax to classify it among ’N’ categories.
- Use cross-entropy loss for optimization.

Dataset

- Subset of the VGGFace2 dataset.
- Images are downloaded from Google Image Search
- Large variations in pose, age, illumination, ethnicity, and profession
- 7,000 identities.
- Class-balanced, so each class has the equal number of training images, and all the images are resized to 224 x 224 pixels.
- **Aim**: Learn to classify images with the correct face identity from 7000 identities.

Dataset and Dataloader class

- Use the ImageFolder class from the torchvision library and passing it the path to the training and validation dataset
- ImageFolder class will automatically infer the labels and make a dataset object, which we can then pass on to the dataloader.
- Make sure to pass the image transforms to the dataset class for doing data augmentation.
- The images in subfolders of classification data are arranged in a way that is compatible with this dataset class

How do we train CNN’s?

- Conducting face classification = multi-class classification
- Input to system = face’s image, model’s output = predicted ID of the face
- **Goal**: Train your model on data to produce “good” face embeddings.
- **Optimize** these embeddings to predict the face IDs from the images. The resulting embeddings will encode a lot of discriminative facial features, just as desired. This suggests an N-class classification task.
- A typical multi-class classifier conforms to the following architecture: **Classic multi-class classifier = feature extractor(CNN) + classifier(FC)**

How do we train CNN's?

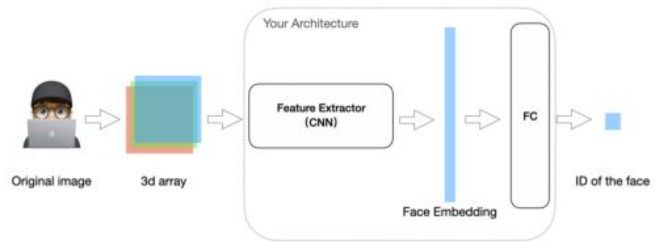


Figure 3: A typical face classification architecture

- The input will be several images
- Network consists of several (convolutional) layers for feature extraction.
- The output of the last feature extraction layers is the face embedding.
- Then pass this face embedding through a linear layer, followed by a Softmax, to classify the image among the N people.
- Use cross-entropy loss to optimize your network to predict the correct person for every training image (the ground truth will be provided in the training data)
- Use validation set for fine-tuning your model.

Residual networks

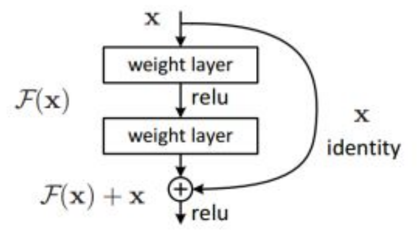

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4: A Residual Block

- Deep neural networks suffer from vanishing and exploding gradients
- Residual networks (resnets) enable us to train very deep neural networks
- Resnets consist of residual blocks- set of layers that are connected to each other, and the input of the first layer is added to the output of the last layer in the block.
- This is a residual connection, allowing deeper networks to be built and trained efficiently.
- Popular architectures that make use of residual blocks: MobilNet, ResNet and ConvNext etc. Please read their respective research papers!!

Starter Notebook Overview

- **Download Data** from Kaggle (Add your Kaggle key and username)
- Set your **Configurations**
  - Epochs - 10 for early submission, but potentially 100 for final submission. Start early!!!
  - Batch_size
  - Learning rate
- **Classification Dataset**: Transforms/augmentation methods using torchvision
- **Data visualization :** Sanity check
- **Very Simple Network (for Mandatory Early Submission):** 4-layer CNN
- **Resnets :** MobilNet, ResNet and ConvNext, etc
- **Setup everything for training:** Criterion, Optimizer, Scheduler

Questions?