ARTIFICIAL NEURAL NETWORKS - WEEK 5

Convolution Neural Networks (CNNs)

Dr. Aamir Akbar

Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

CONTENTS

CONVOLUTION NEURAL NETWORK

What are CNNs?

Convolutional Neural Networks (CNNs) are a class of deep neural networks, primarily used for analyzing visual imagery. They have proven highly effective in tasks such as image classification, object detection, and image segmentation.

Mathematical Formulation of Convolution:

$$(f \otimes g)(x, y) = \sum_{m} \sum_{n} f(m, n) \cdot g(x - m, y - n) \tag{1}$$

where f represents the input image and g represents the convolutional kernel (also called filter or mask).

The convolution operation is performed by sliding the kernel over the input image, computing element-wise multiplication, and summing the results. This process helps in extracting features from the input image.

CONVOLUTIONS ON IMAGE PROCESSING

1. Blurring, sharpening, embossing, and edge detection are typical functions of image processing.
2. They are accomplished by means of convolution between a kernel and an image. For example, the `Laplacian kernel` of an image highlights regions of rapid intensity change. The Laplacian is therefore often used for edge detection

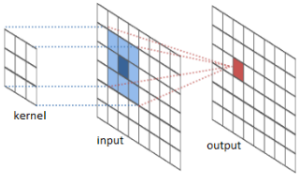
Laplacian of Gaussian (LoG) kernel.

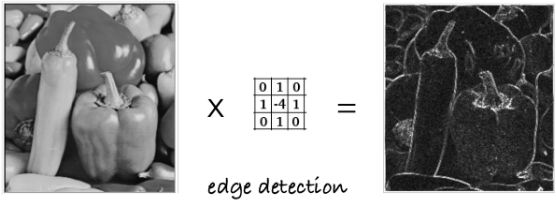
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Laplacian kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

CONVOLUTIONS ON IMAGE PROCESSING





A KERNEL OR FILTER

In the context of convolutional neural networks (CNNs), a filter or kernel refers to a set of `learnable weights` that are applied to input data using the convolution operation.

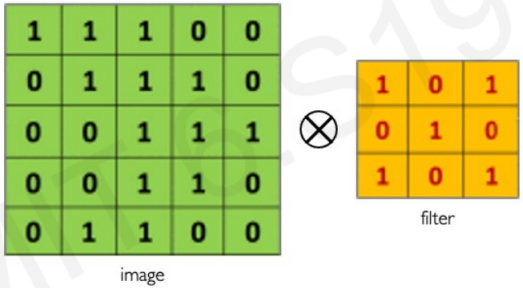
Each filter/kernel is typically a small matrix (e.g., 3×3 or 5×5) containing numeric values. During the training process, these values are adjusted (learned) through backpropagation in order to extract useful features from the input data.

In a convolutional layer, multiple filters/kernels are applied to the input data simultaneously, each producing a `feature map`. These feature maps capture different aspects or patterns present in the input data, allowing the network to learn hierarchical representations of features at different levels of abstraction.

Example: *Laplacian Kernel*

CONVOLUTIONS OPERATION

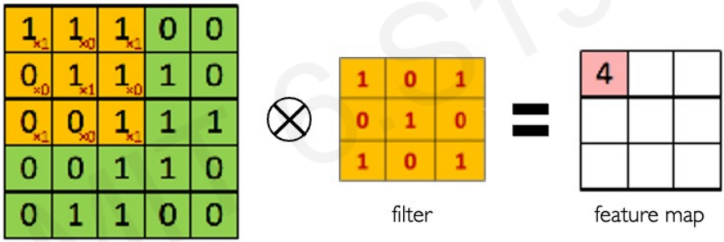
Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:



We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

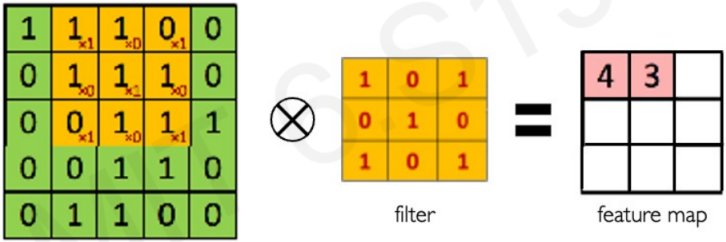
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



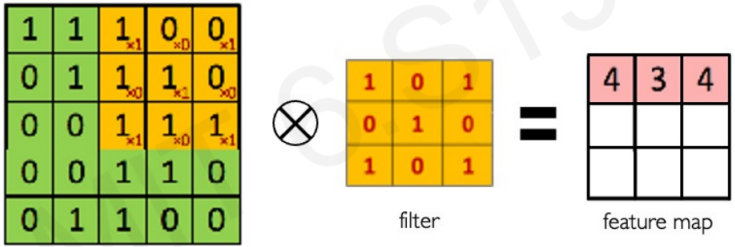
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



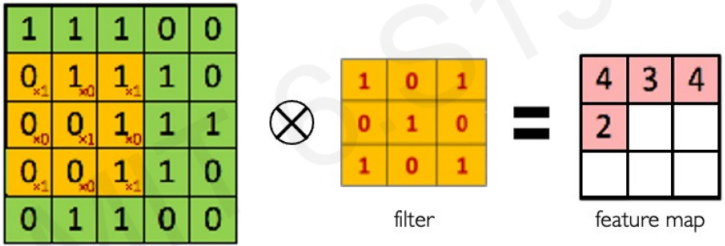
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



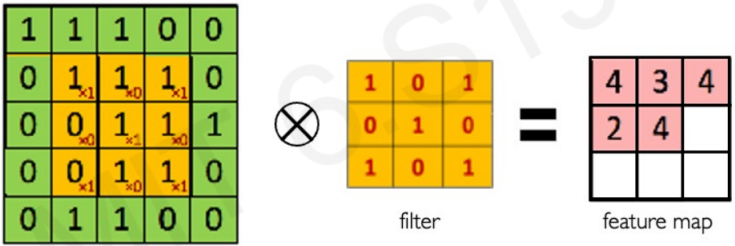
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



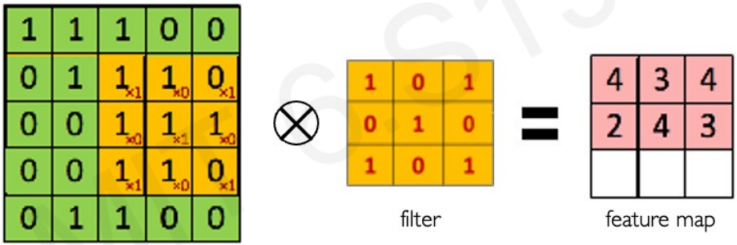
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



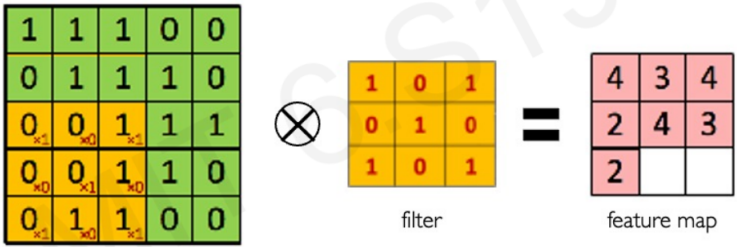
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



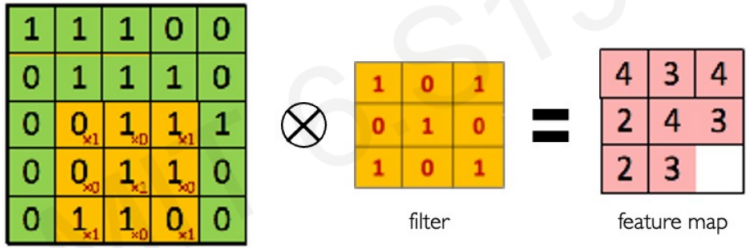
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



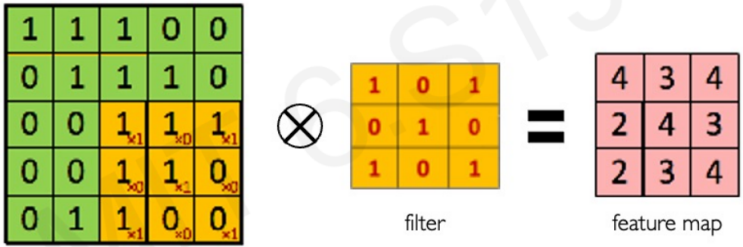
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



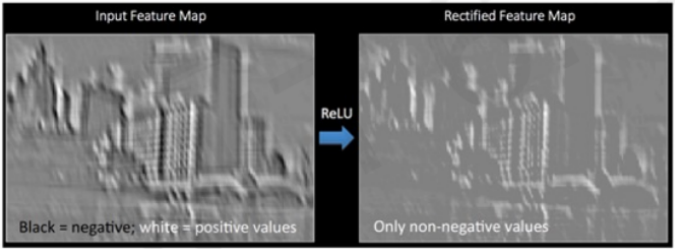
CONVOLUTIONS OPERATION

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

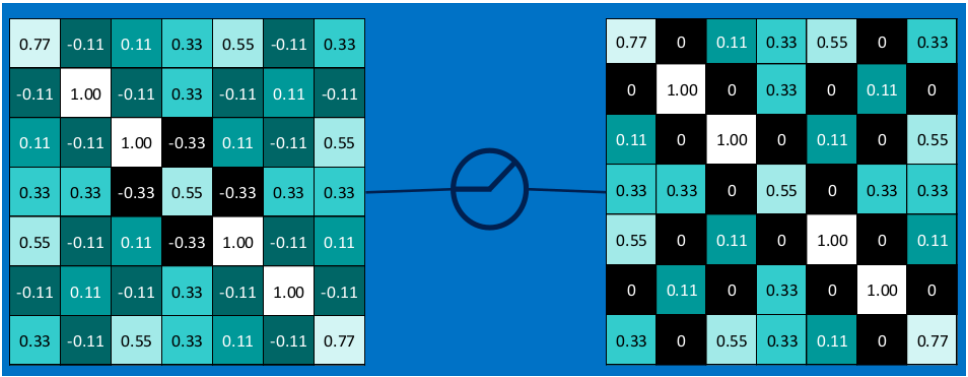


APPLYING RELU - NON LINEAR OPERATION

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



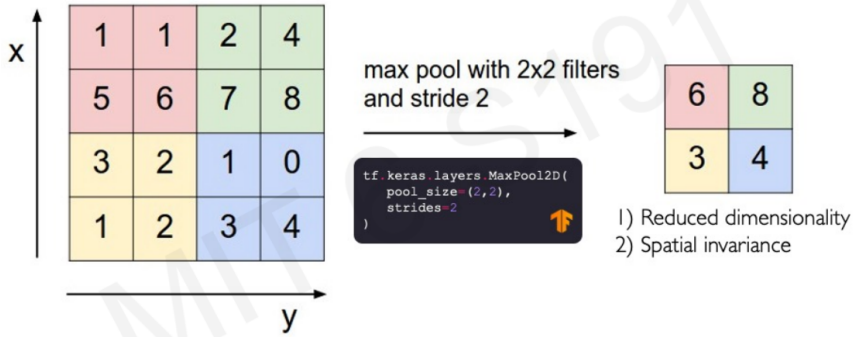
APPLYING RELU - NON LINEAR OPERATION



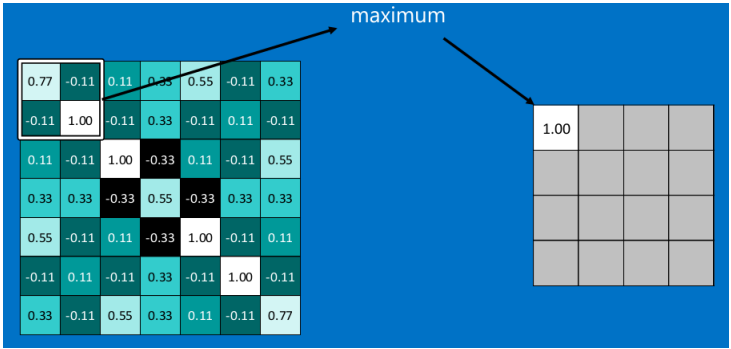
POOLING

Pooling operation shrinks the stack of filtered images. How to perform pooling?

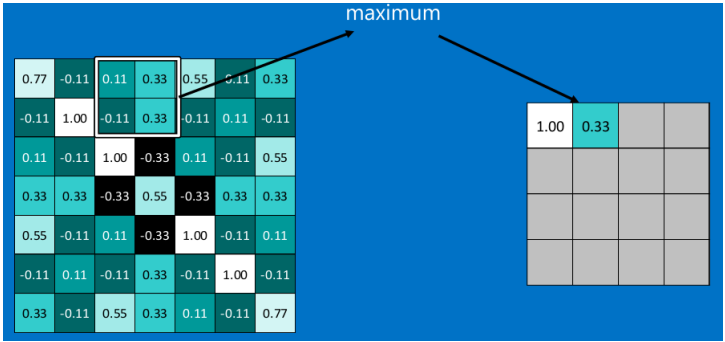
1. Pick a window size (usually 2 or 3)
2. Pick a stride (usually 2).
3. Move the window across the filtered images.



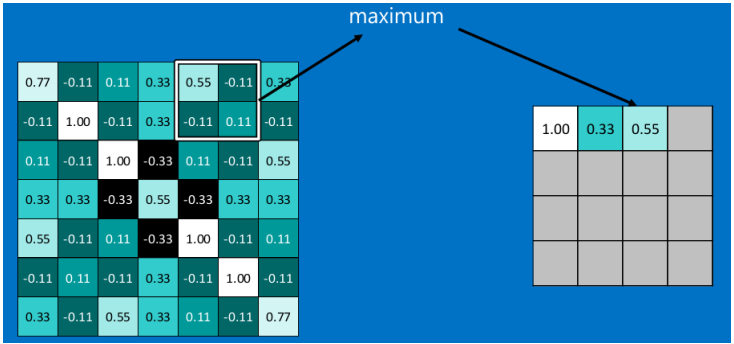
POOLING OPERATION



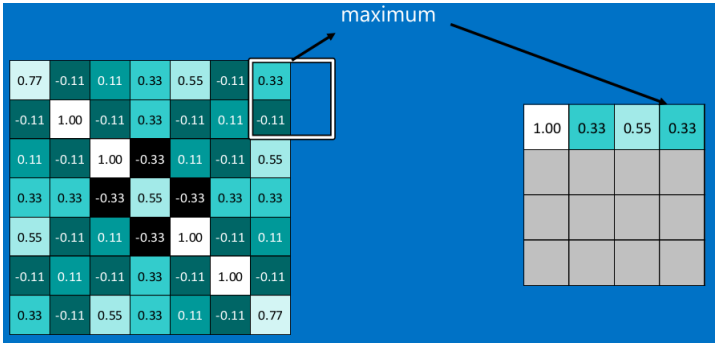
POOLING OPERATION



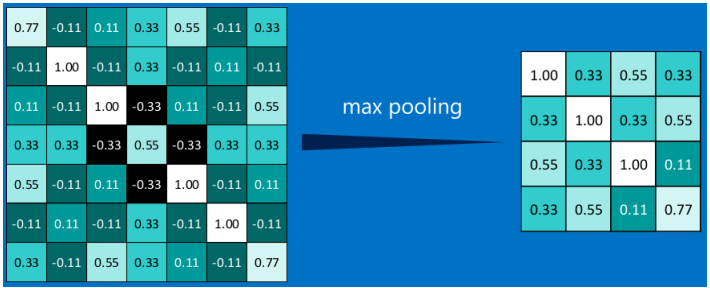
POOLING OPERATION



POOLING OPERATION

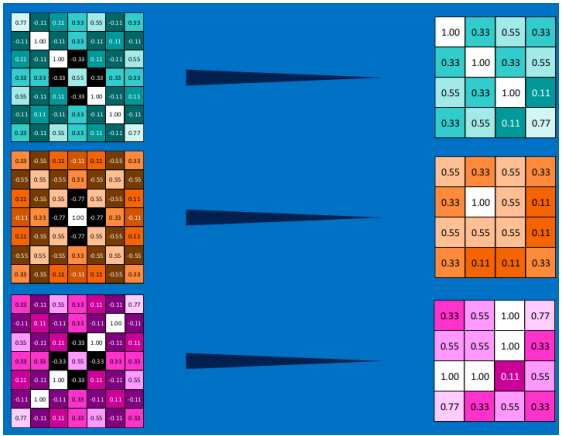


POOLING OPERATION



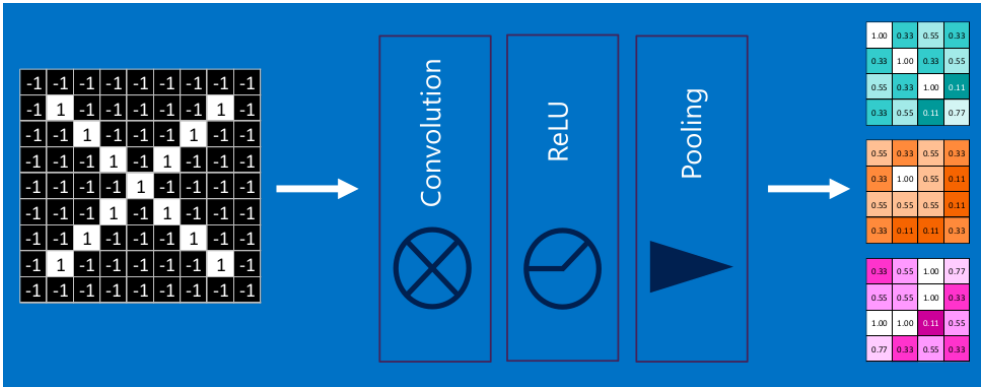
POOLING

A stack of filtered images become a stack of smaller images



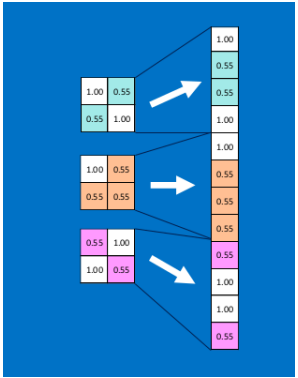
CNNS STACKED LAYERS

The output of one layer becomes the input of the next layer.

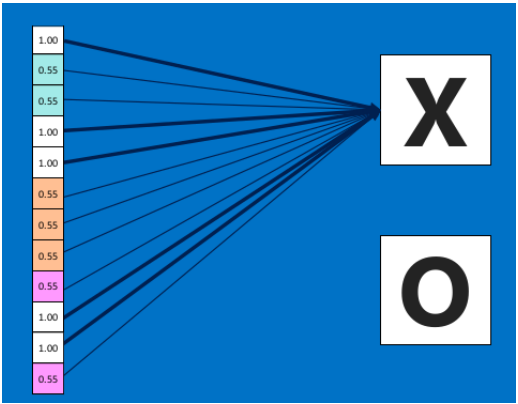


FLATTEN LAYER

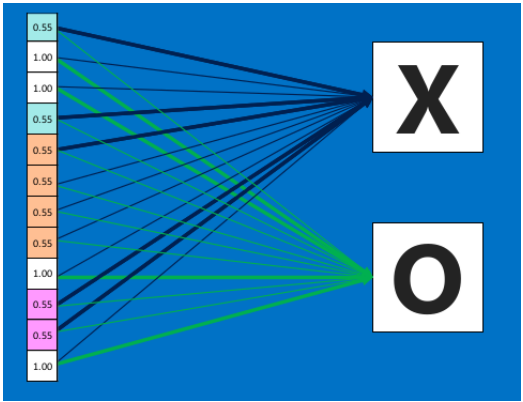
In CNN, the flatten layer reshape the output of the preceding convolutional and pooling layers into a one-dimensional (1D) vector. Before connecting the output of the convolutional and pooling layers to the fully connected (dense) layers, the data must be flattened to a 1D tensor.



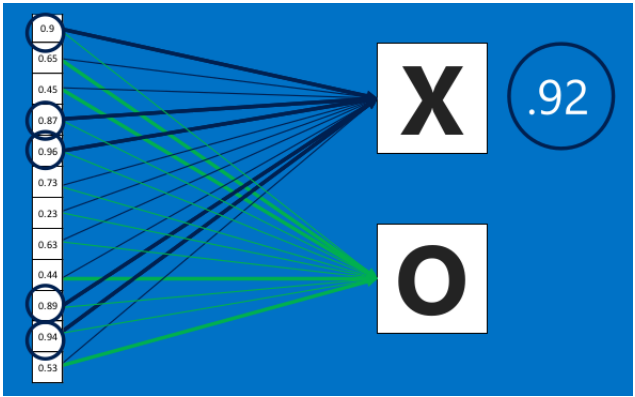
FULLY CONNECTED NEURAL NETWORK



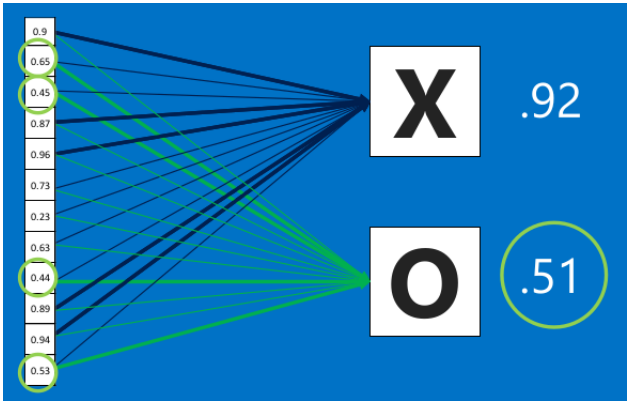
FULLY CONNECTED NEURAL NETWORK



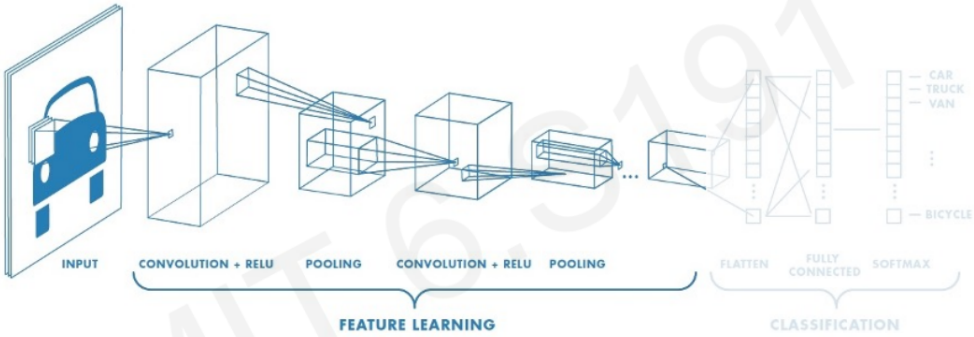
FULLY CONNECTED NEURAL NETWORK



FULLY CONNECTED NEURAL NETWORK

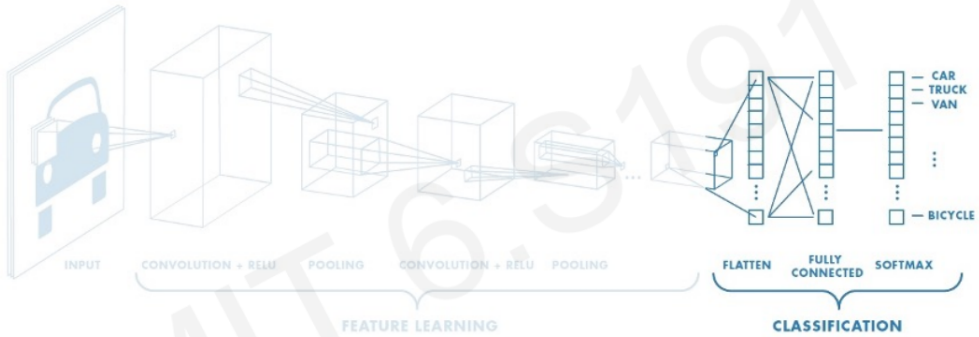


CNN FOR CLASSIFICATION



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

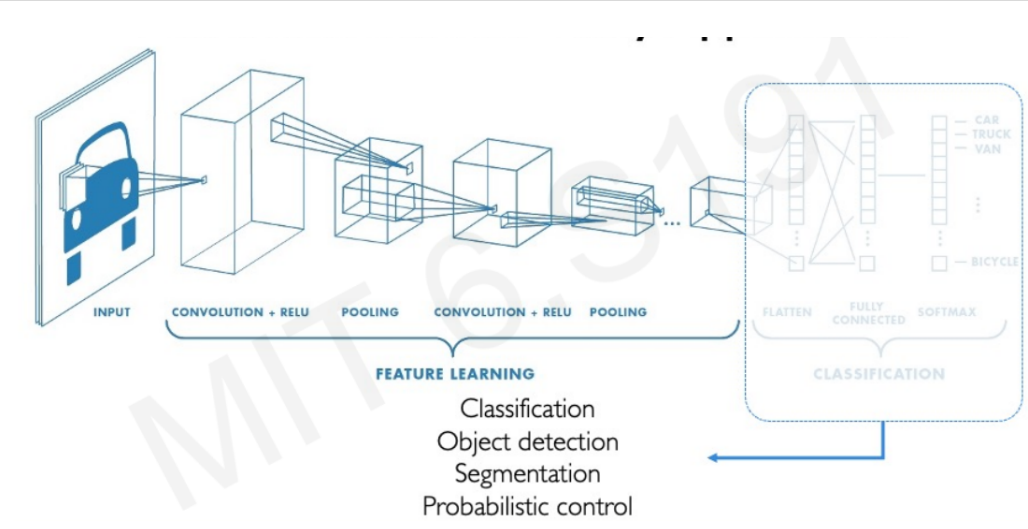
CNN FOR CLASSIFICATION



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNN FOR CLASSIFICATION

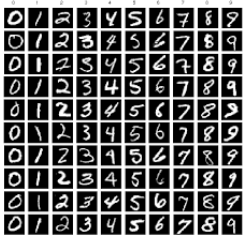


CASE STUDY: MNIST DIGITS CLASSIFIER USING CNN

MNIST Dataset Overview

The MNIST dataset is a classic benchmark dataset widely used in machine learning research, containing 70,000 grayscale images of handwritten digits (0-9) with a resolution of 28x28 pixels. It is a subset of a larger dataset collected by the National Institute of Standards and Technology (NIST). MNIST is commonly used for training and evaluating machine learning models, particularly for image classification tasks, due to its simplicity and availability.

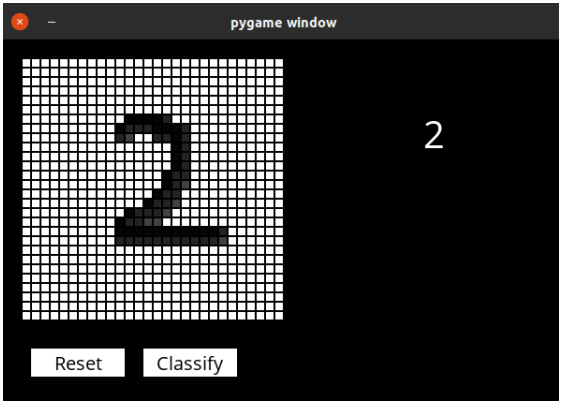
Training set: 60,000 images, Testing set: 10,000 images



CNN MODEL ARCHITECTURE

```
Model: "digit_classifier"
_________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             multiple                  320
 max_pooling2d               multiple                  0
 flatten (Flatten)           multiple                  0
 dense (Dense)               multiple                  692352
 dropout (Dropout)           multiple                  0
 dense_1 (Dense)             multiple                  1290
=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
_________________________________________________________________
```

CNN: DIGITS CLASSIFICATION



RESOURCES

To download the source codes used in the previous slides, follow the link:

▸ Download Source Codes

Import the codes into your preferred development environment, such as Visual Studio Code (VS Code), to practice and explore further.

To learn programming in Python, follow my comprehensive 15-week Programming in Python course at:

▸ Programming in Python