Stock Price Prediction using CNNs

Applied Masters Project

M.Sc. in Financial Engineering
Session 3

ACADEMIC YEAR 2024-2025

Dr. Dominic O'Kane

This Version: March 2025

Introduction

- In this project you will test whether technical indicators may be used to build a successful equity trading strategy.
- You have been using simple ANNs and last time we discussed using RNNs
- Today we finish with one further extension
- We are going to use **Convolutional Neural Networks**

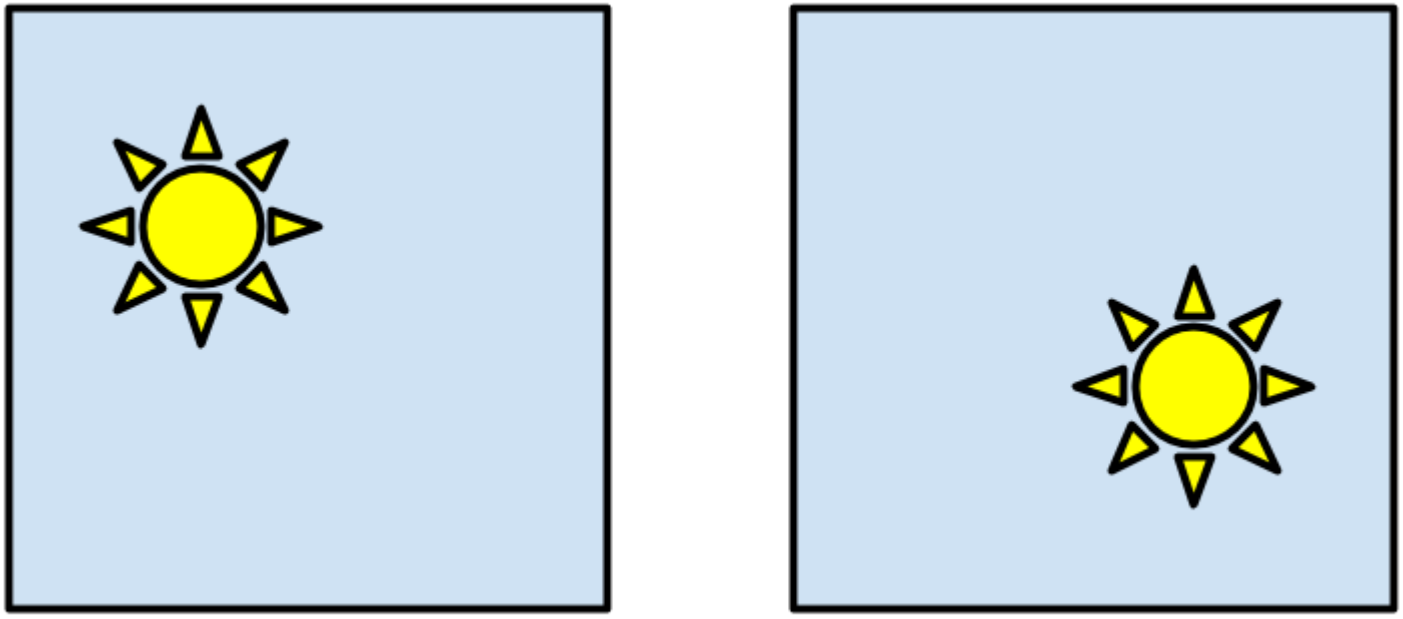
Motivation

- You have already learnt the definition and main concepts of an Artificial Neural Network (ANN in short).
- You will learn the main concepts of convolutional neural networks (CNN in short).
- These are well-designed for computer vision classification problems.
- They are also widely used for time-series forecasting (stock market prediction) and for NLP (text classification).
- They are essentially clever pattern detection algorithms.

Convolutional Neural Networks

The Challenge of Image Detection: Position and Scale

- How can a neural network detect the “sun” in the two images
- It needs to learn the shape of a sun
- It needs to find it even if it is not in the same position
- It needs to find it even if it has different sizes



Where else can we have detectable patterns ?

- Stock market prices as sequences
- Stock market prices as images !
- Text – sentence may have a pattern of words
- Economic data

Convolutions

Visual Cortex

- Experiments on animal brains in the 1950s showed that neurons in the brain have a small local receptive field
- They react only to visual stimuli located in a limited region of the visual field
- These local receptive fields overlap and may cover the entire visual field
- Some neurons react to images of horizontal lines while others react to vertical lines
- This led AI researchers like Yann LeCun to introduce the LeNet-5 architecture in 1998 used to recognize handwritten digits
- It introduced **convolutional layers** and **pooling layers**

What is a Convolution ?

- To understand a CNN, we must define convolution
- Convolution is a simple mathematical operation on a matrix
- An example in most photoshop applications is a Gaussian blur
- We take the input image and **convolve** a Gaussian **filter/kernel**

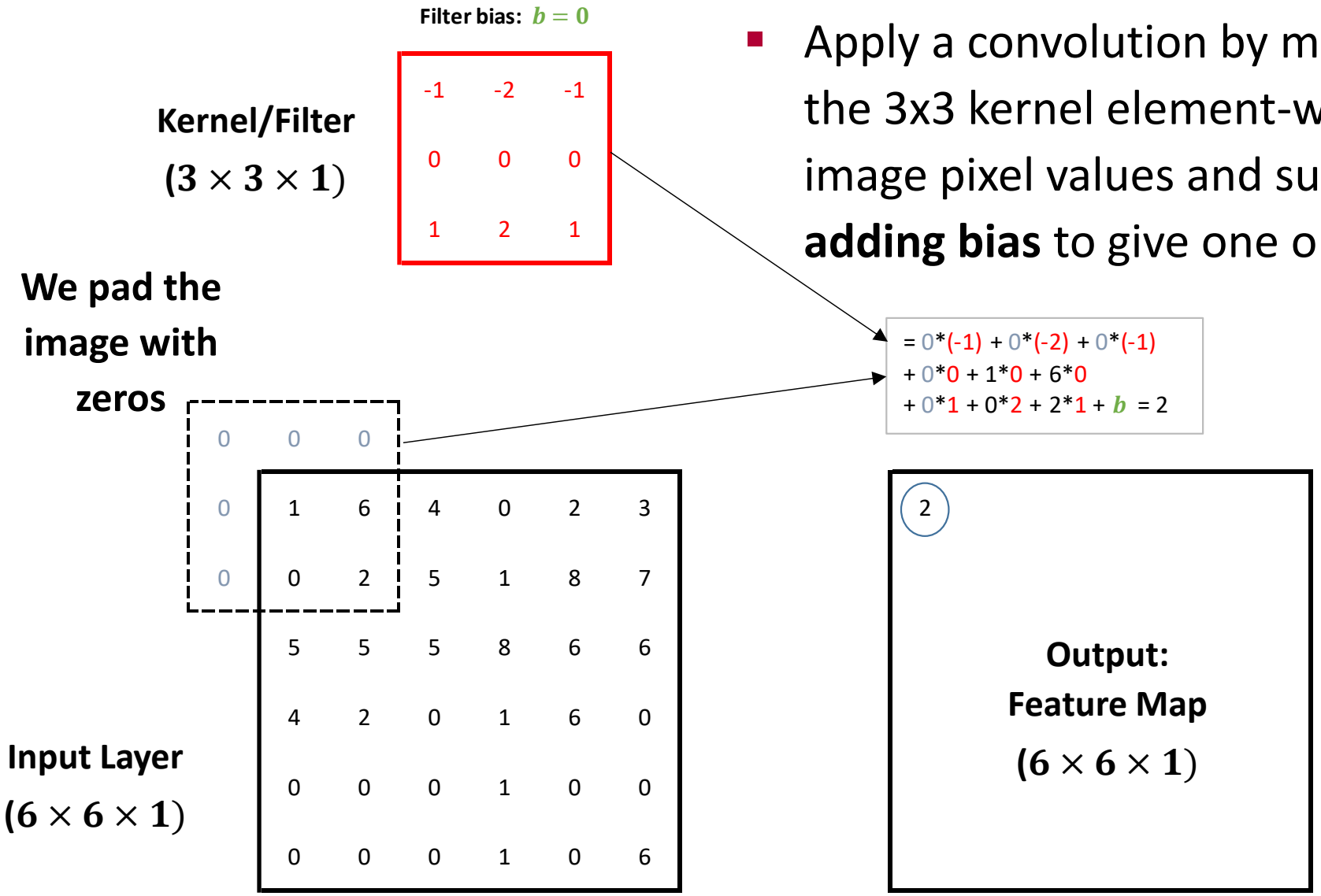
Gaussian Blur Kernel/Filter

- I will use the words **kernel** and **filter** for the same thing
- A kernel or filter is simply a matrix, e.g., here is a 5 x 5 gaussian blur filter

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

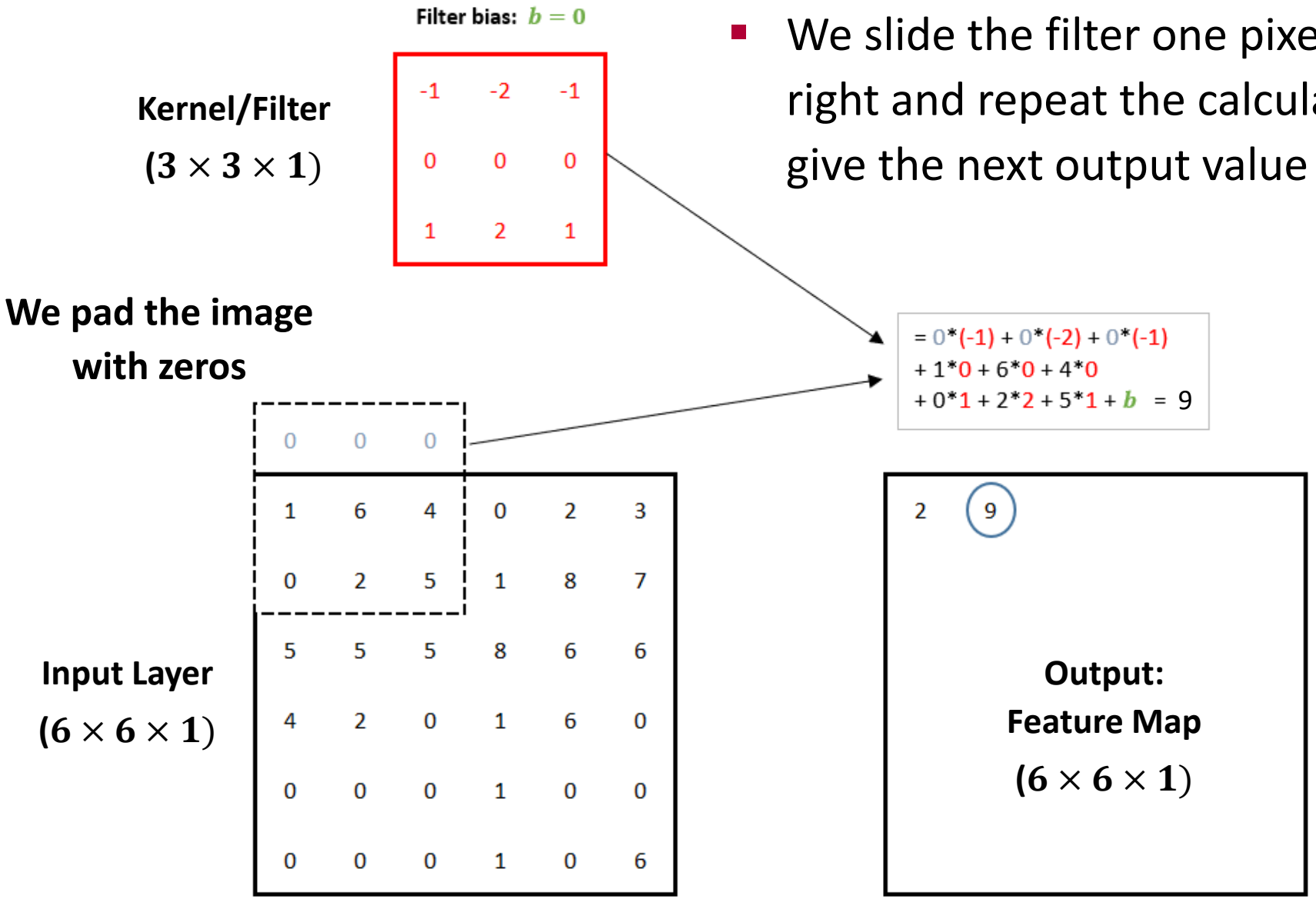
- The pre-factor ensures that it sums up to 1
- Highest values in centre and smallest values away from centre
- Convolution has something to do with applying this filter (kernel) to the pixels of the image
- The output after applying the kernel is called a **feature map**
- Let us see the specifics

Convolution Operation: First output pixel



- Apply a convolution by multiplying the 3x3 kernel element-wise by the image pixel values and summing and **adding bias** to give one output value

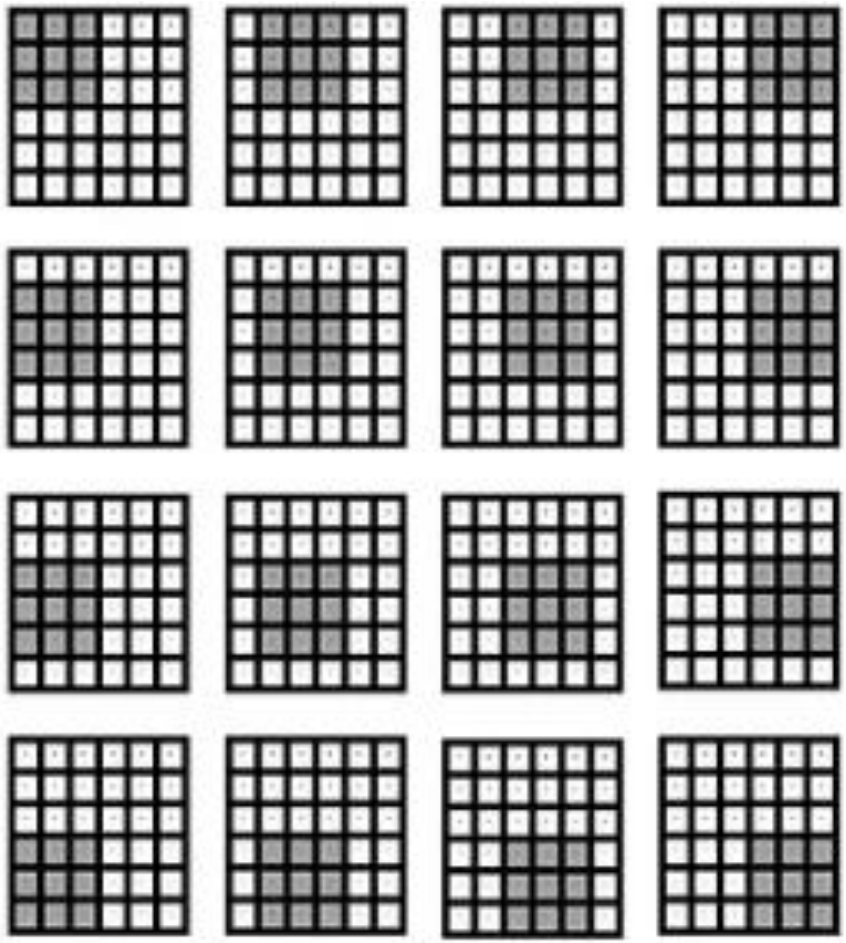
Convolution Operation: At next pixel



- We slide the filter one pixel to the right and repeat the calculation to give the next output value

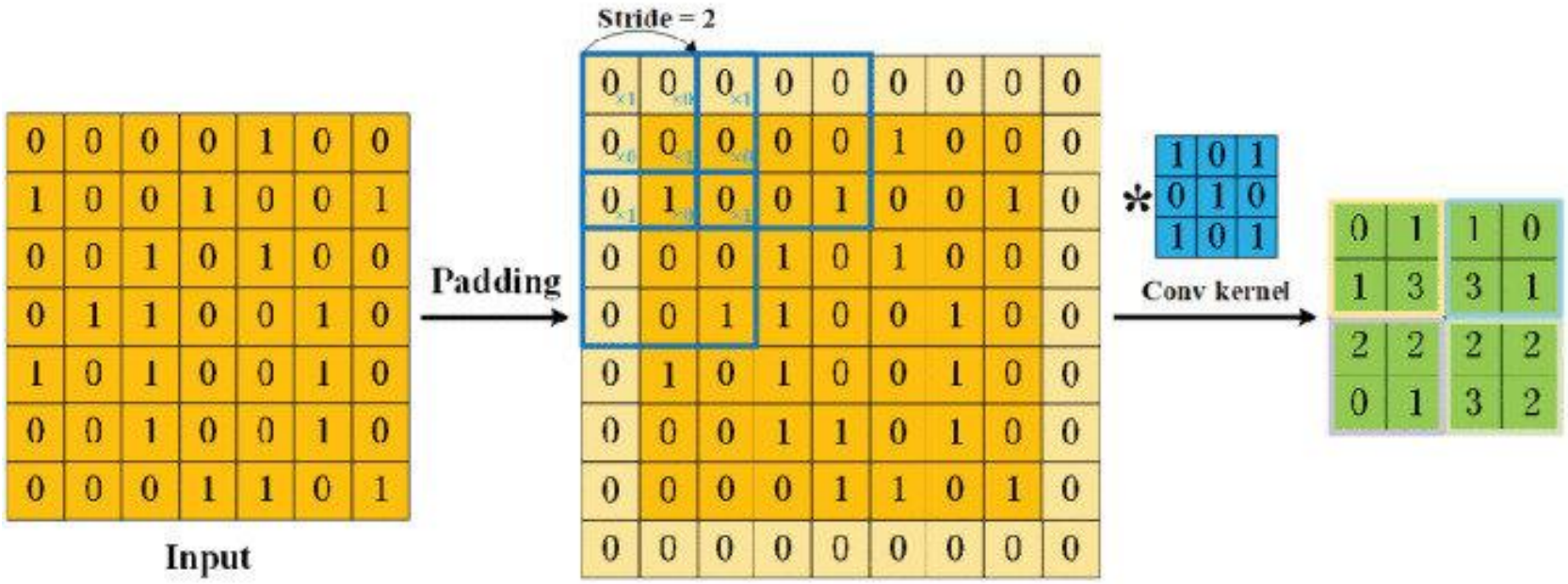
The Full Convolution: Stride

- To finish the convolution, we move the kernel across the N x N image ensuring that we cover all the pixels
- We move the K x K kernel one pixel at a time – but this can be more – known as the **stride**
- With the example here we get an image of size (N-K+1) x (N-K+1)
- We can add padding to get a larger image back



Example with a Stride of 2 and Padding 1

- Here we have an N x N image where N = 7
- The kernel filter is K x K where K = 3



- The output image is of size (N+2) – 3 + 1 = 7 for a stride of 1
- It is of size ((N+2) – 3)/2 + 1 = (9 – 3)/2 + 1 = 4

What is the point of the Kernel ?

- Depending on the choice of Kernel, we can highlight certain features of an image
- Well-known examples of kernels are the Sobel filters

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Vertical Edges **Horizontal edges**

- The one on the left highlights vertical edges
- The one on the right highlights horizontal edges
- The vertical edge filter takes a horizontal derivative, vice-versa
- The vertical edge filter does Gaussian smoothing in Y direction

Why does this kernel detect vertical edges ?

- Suppose the image is gray with all pixels at value 100

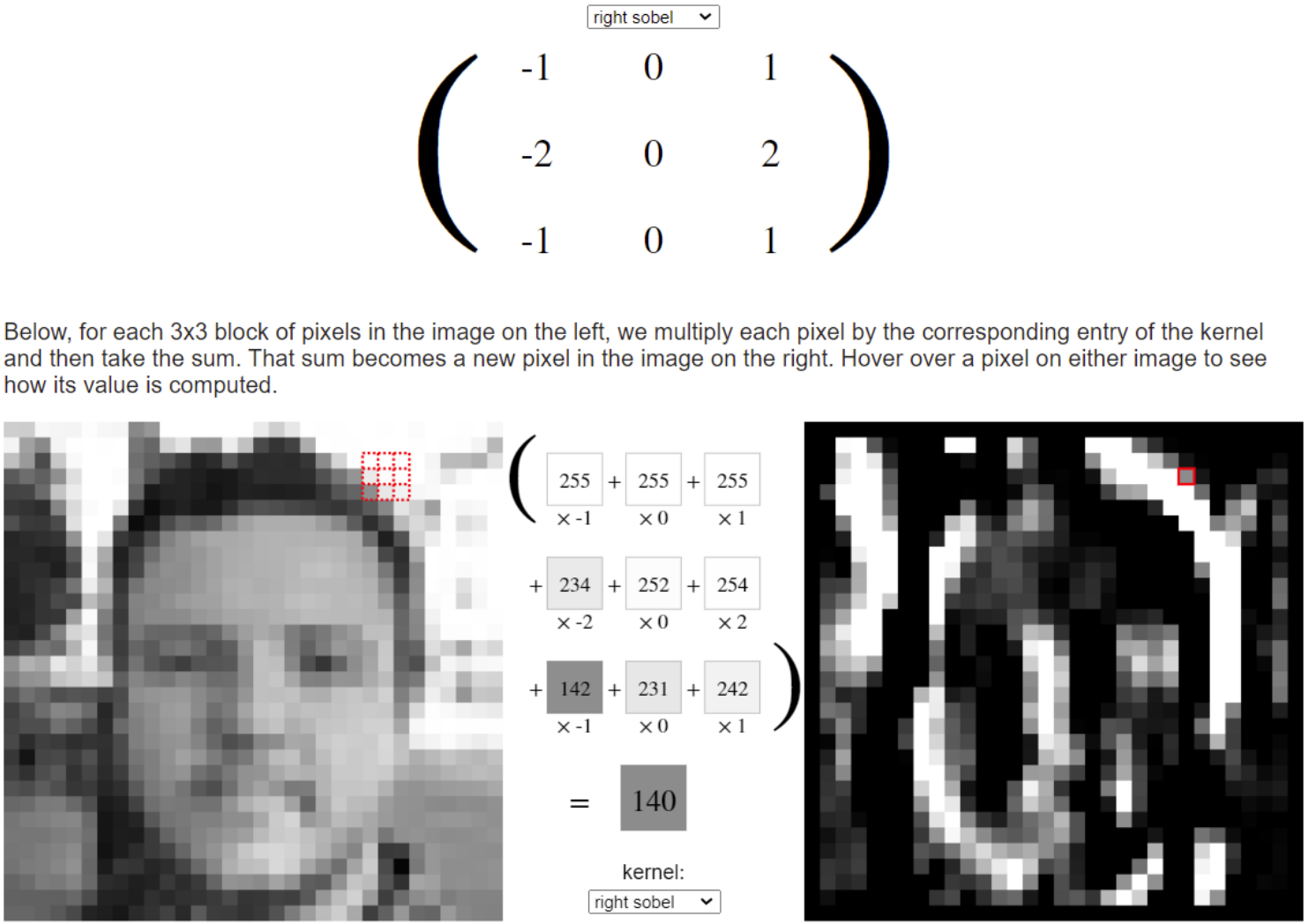
$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

- If I apply this filter to the matrix of 100's I will get zero in all the output feature pixels because positive and negative terms cancel
- If the pixels on the left half of image are at 50, those on the right half are at 150, I will get zeros everywhere that is not an edge
- Along the edge I will get -1 x 50 – 2 x 50 – 1 x 50 + 1 x 150 + 2 x 150 + 1 x 150 = -200 + 600 = +400 !
- I get a bright vertical line at the edge

Filters: Example of Recognising Vertical Edges

Source:

https://setosa.io/ev/image-kernels/



Filters: Example of Recognising Horizontal Edges

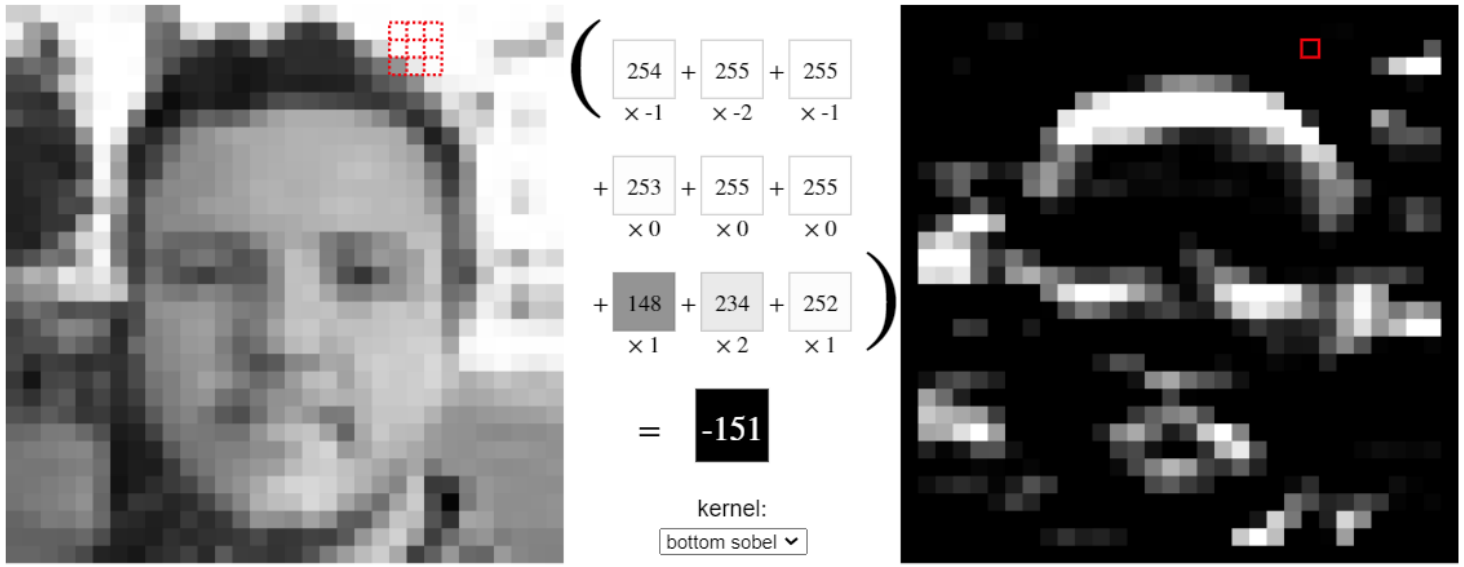
Source:

https://setosa.io/ev/image-kernels/

bottom sobel

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



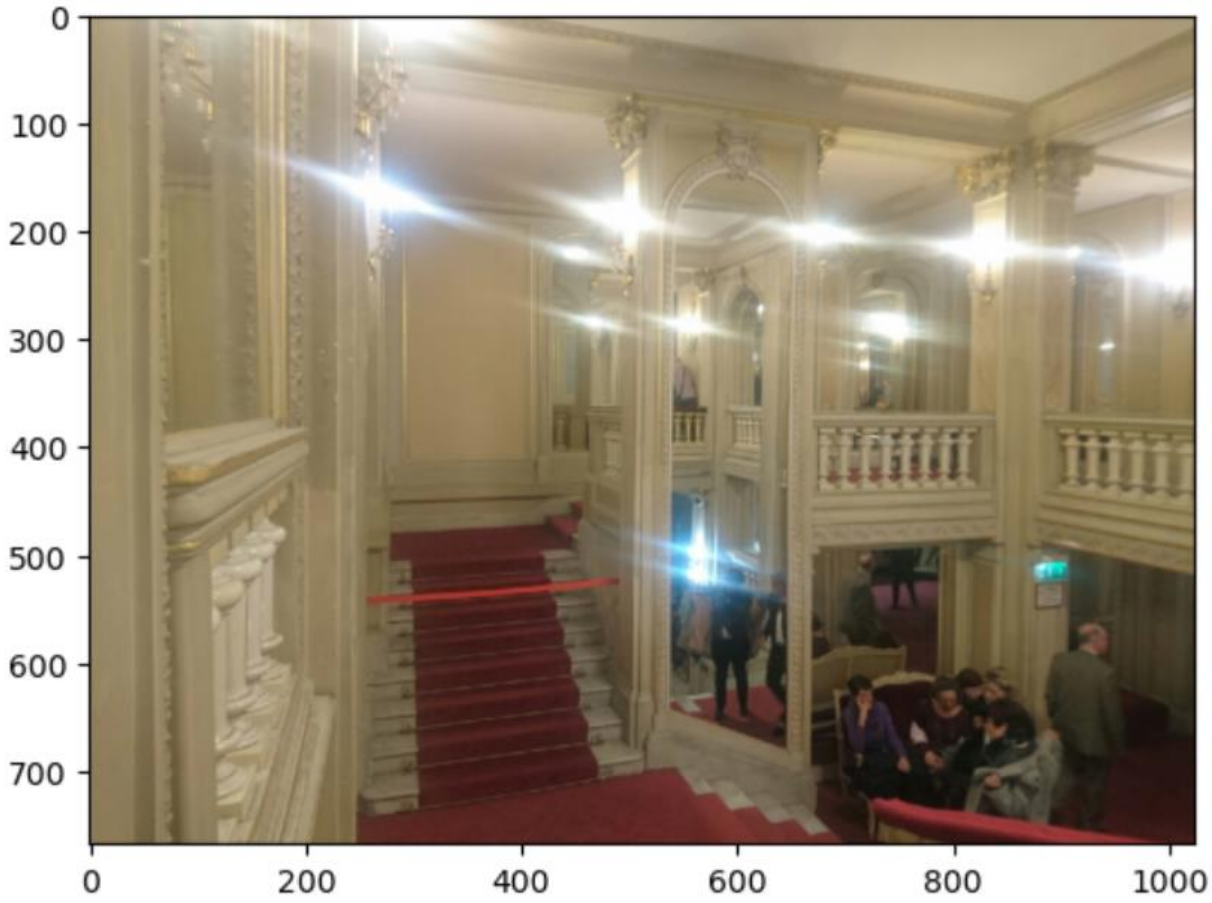
input image

output image

Example of Applying Filters

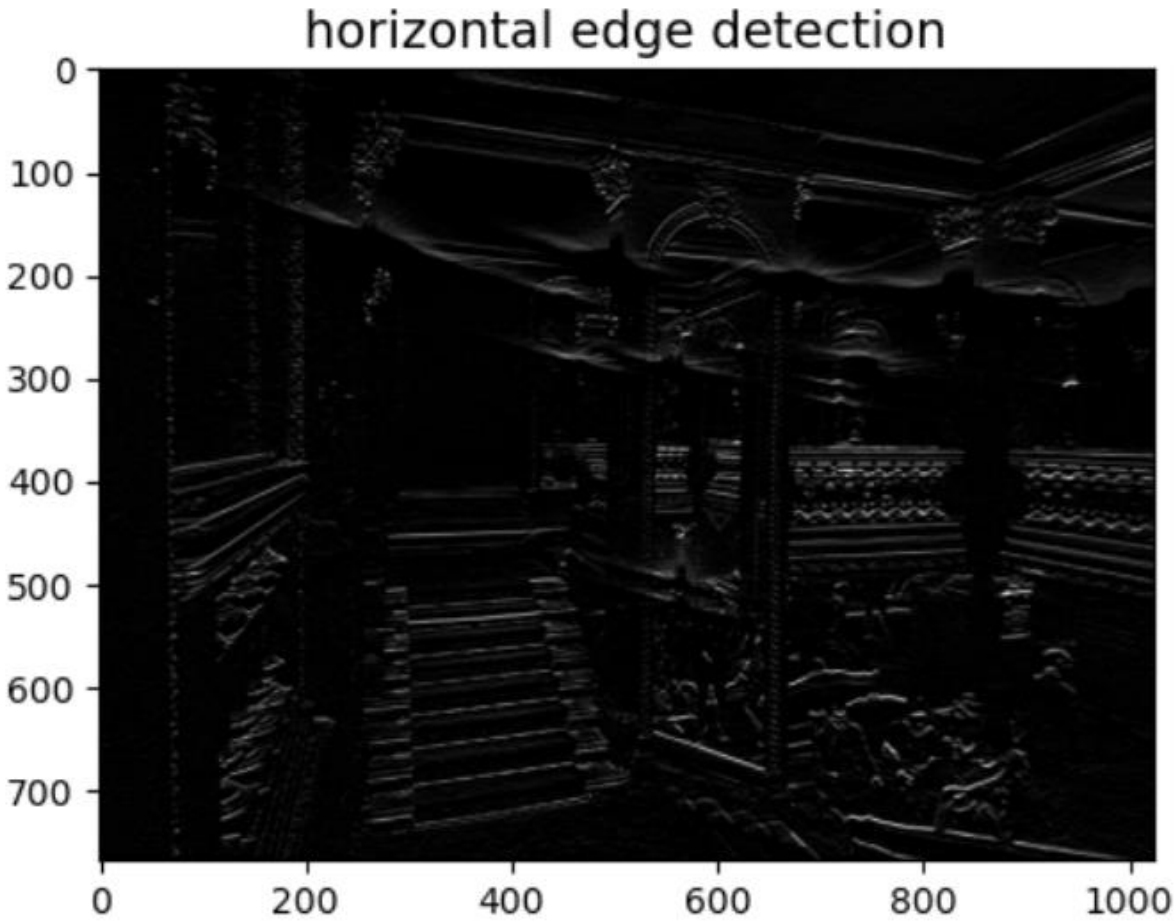
- For this example, we use OpenCV – a computer vision library
- You need to install it - **pip install opencv-python**
- Here is an example image with lots of complex edges

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# Write pip install opencv-python
import cv2
# Read in the image
image = mpimg.imread('Opera.jpg')
plt.imshow(image);
```



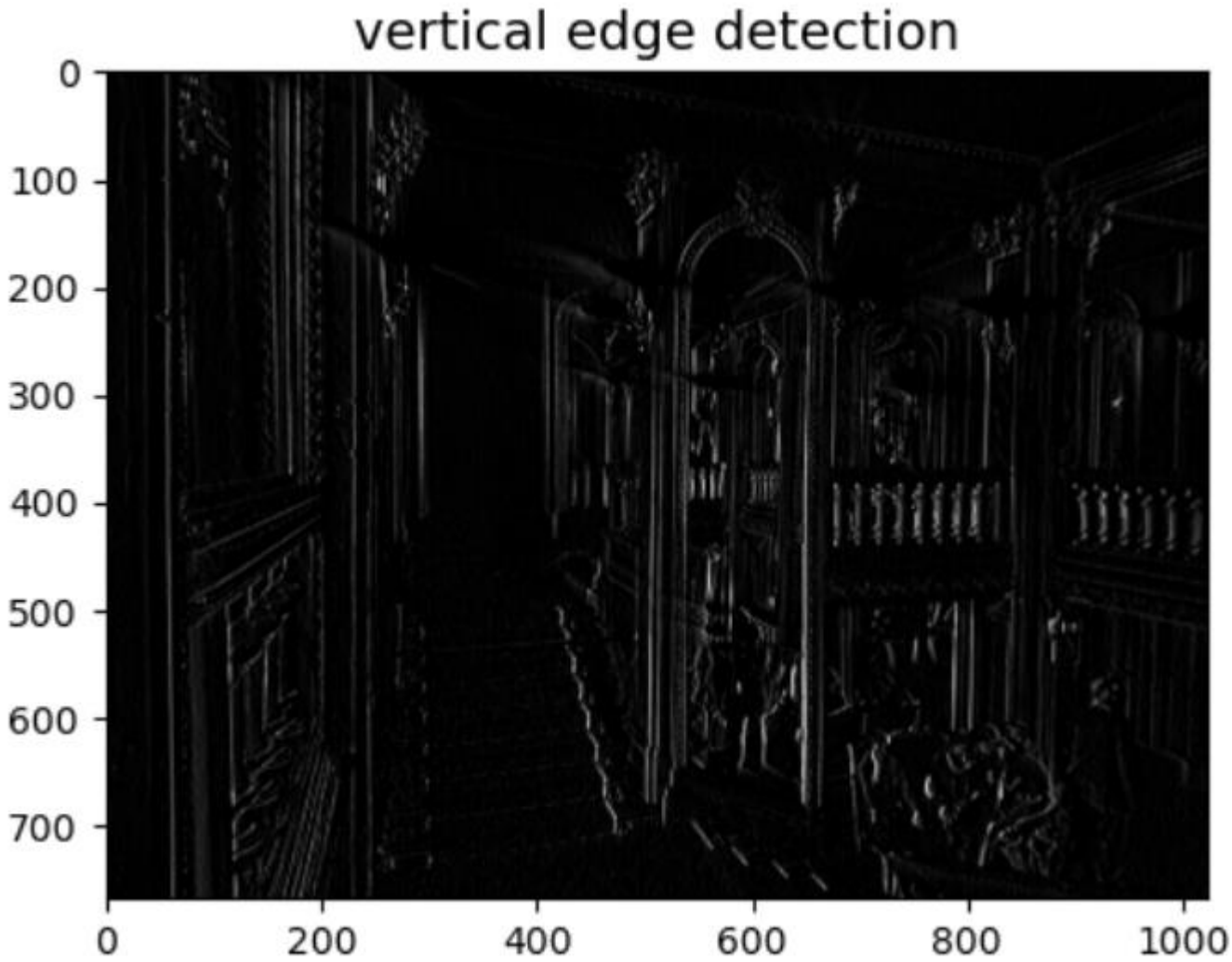
Horizontal Edge Detection

```
# 3x3 sobel filter for horizontal edge detection
sobel_x = np.array([[ -1, -2, -1],
                    [ 0, 0, 0],
                    [ 1, 2, 1]])
```



Vertical Edge Detection

```
sobel_y = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])
```



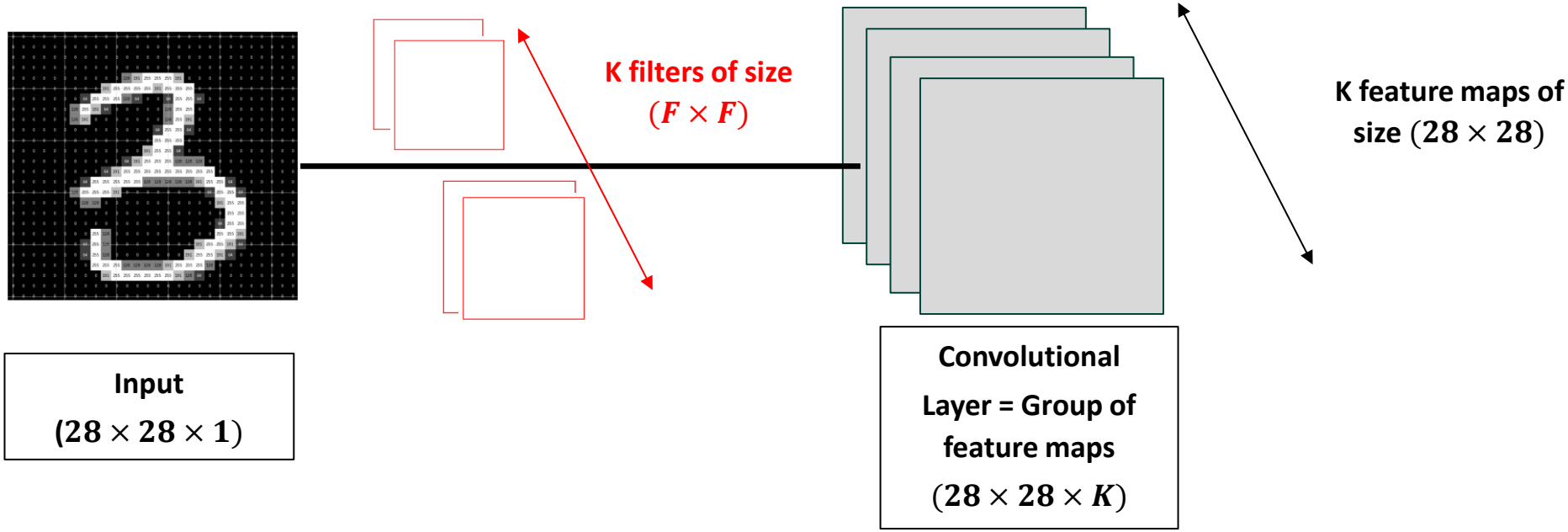
So how do we link Kernels to Neural Networks

- The idea is that we learn the Kernels
- We input an image and a label, e.g. some classification
- We set up the network in such a way that the Kernels take the place of weights
- We allow the network to have multiple layers of Kernels
- **We train these kernel "weights" by back propagation to be able to learn the classification**
- So, the network finds kernels that extract the important features that allow the network to solve the classification task
- But we can't do this with the layers we have used so far in TensorFlow

Convolutional Layer

Convolutional Layer

- A convolutional layer applies K filters to the original image.
- The outputs are K feature maps. K is a hyperparameter.
- The filter values and bias are parameters of the layer
- **They are equivalent to weights and biases !**
- **They are learned through the backprop training process !**



Convolutional Layer Hyperparameters

- The convolution layer hyperparameters are:
  1. The number of **feature maps** K
  2. The **activation function**
  3. The **size of the filter** F
  4. The **stride** S is the number of pixels by which the filter matrix steps over the image matrix after each convolution operation (S=1 in our example)
  5. The **padding** P, which denotes the process of adding (or not) zeros to each side of the input (in our example where we added zeros, P="SAME")

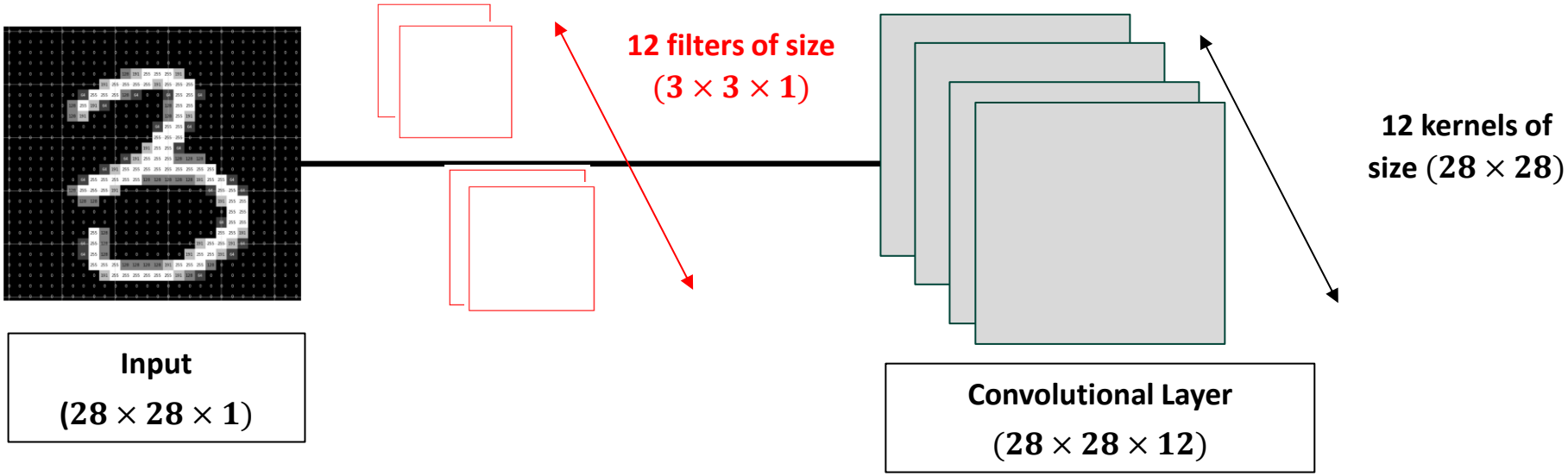
We need to set the output image size

- There are three choices shown below

Mode	Output Size	Usage
Valid	N – K + 1	Typical
Same	N	Typical
Full	N + K – 1	Unusual

- If we use Valid then the output image is slightly smaller
- If we use Same then it is the same size
- If we use Full then it is bigger
- All of this is achieved by an appropriate padding

Convolutional Layer with TensorFlow




```
import tensorflow
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Conv2D, Input

#Define and build a neural network made of only one convolutional layer
model = Sequential()
model.add(Input(shape=(28,28,1)))
model.add(Conv2D(12, kernel_size=(3,3), activation='relu',strides=1, padding='same'))
model.summary()
```

Size of Convolution Layer

- The CNN has the following structure

```
Model: "sequential_1"
```

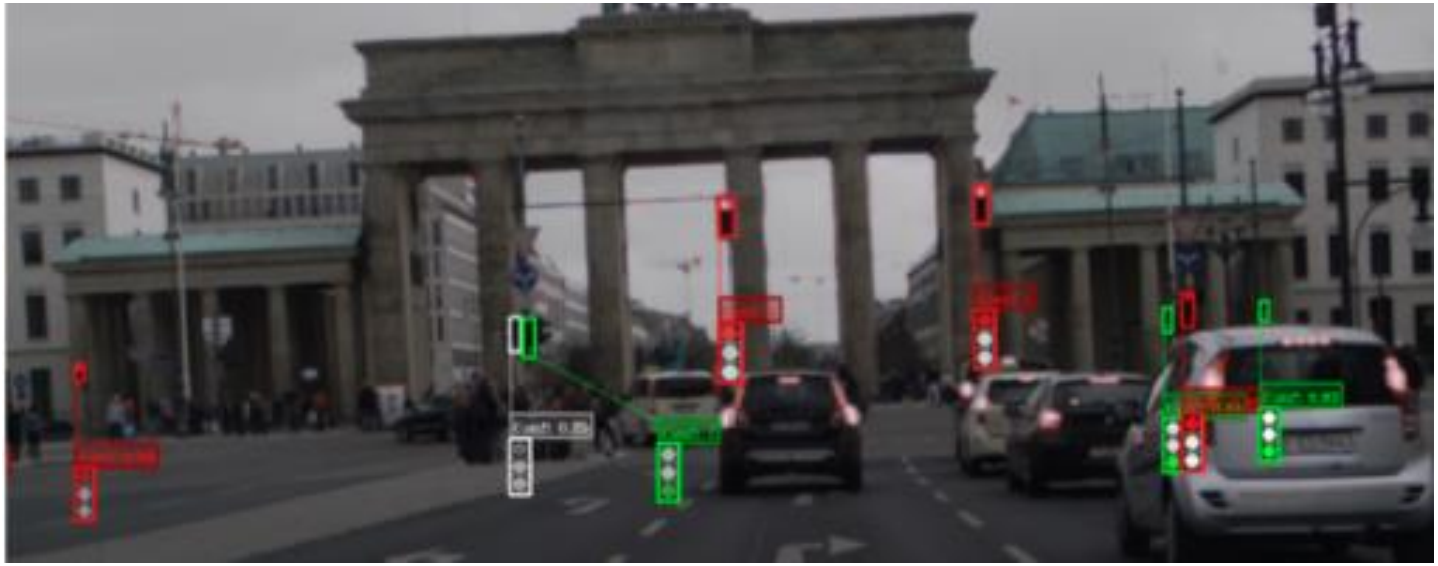
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 12)	120

```
Total params: 120 (480.00 B)
Trainable params: 120 (480.00 B)
Non-trainable params: 0 (0.00 B)
```

- 120 parameters ?
  - Kernel size = 3 x 3 = 9 + 1 bias
  - 12 kernels
  - So 12 x (9 + 1) = 120
- 120 parameters is not a lot for a 28 x 28 image !!

Convolution Layer needs Multiple Filters

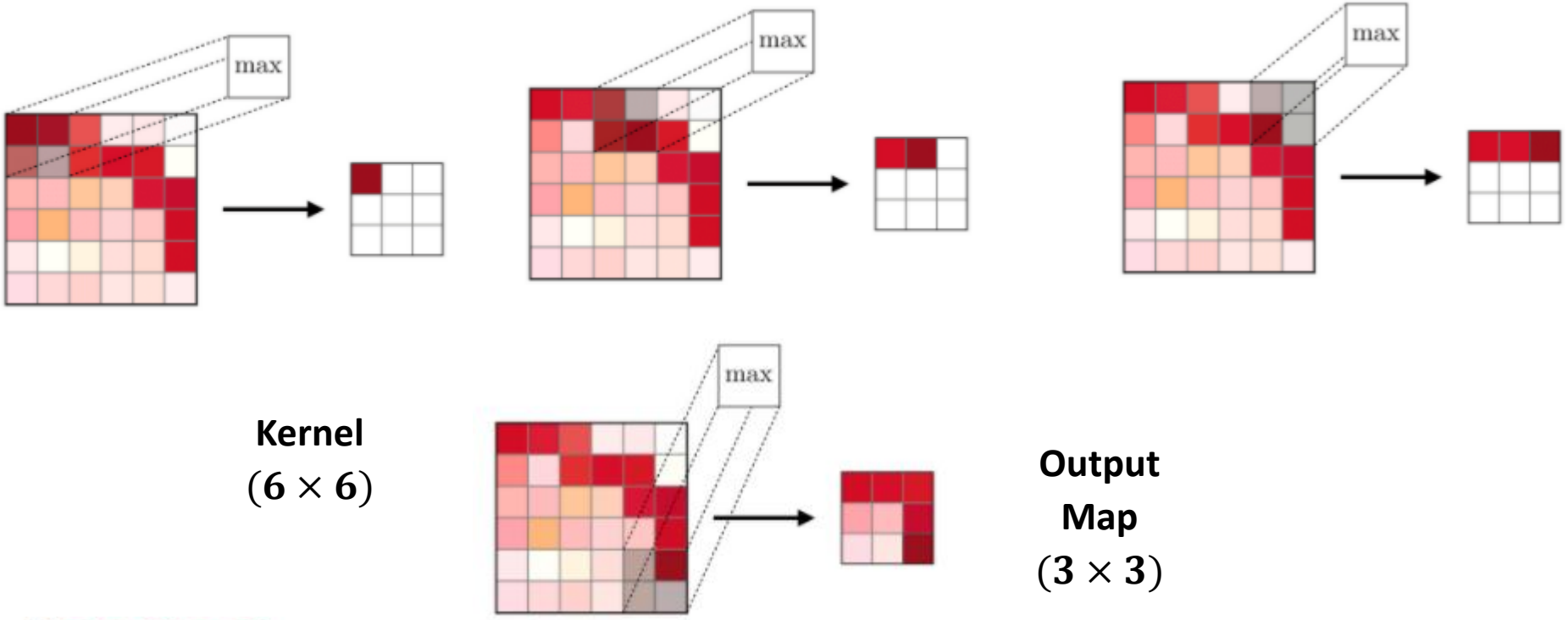
- Convolution layer is a feature finder
- It sees the whole image and can identify the same feature wherever it occurs – **translational invariance**
- But an image may have multiple features
- We wish to find all of them so need multiple filters
- For example, we want to detect cars, lights, road signs



Pooling Layer

Pooling Layer

- The pooling layer **down-samples** the feature maps.
- It reduces the output size, reduces the number of parameters to learn, and the amount of computation performed in the network.
- The pooling layer summarises the features present in a region of the feature map generated by a convolution layer.

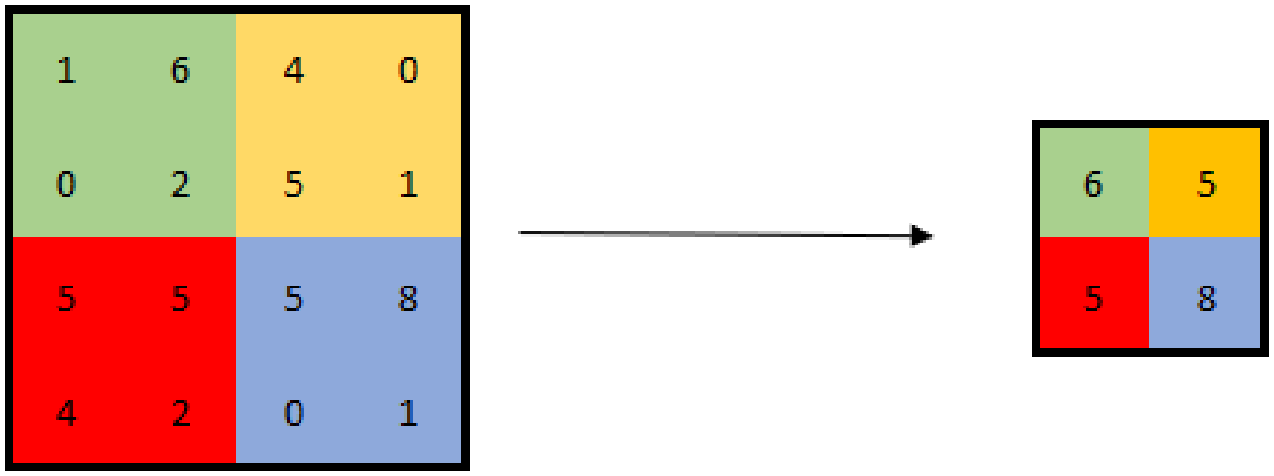


Pooling Layer

- A pooling layer can take in the output of a convolution layer
- A pooling layer **does not contain any weights** to be learnt during the CNN training process
- Pooling reduce the size of the output by aggregating inputs
- The main hyperparameters of a pooling layer are:
  - Filter size F
  - Stride S (typically S=2)
  - Padding P (typically P="valid")
  - Aggregation rule (typically max or mean)
- It's a bit like using a Kernel that never changes
- There are two types of Pooling
  - **Max Pooling and Average Pooling**

Max Pooling: Take the Maximum Value

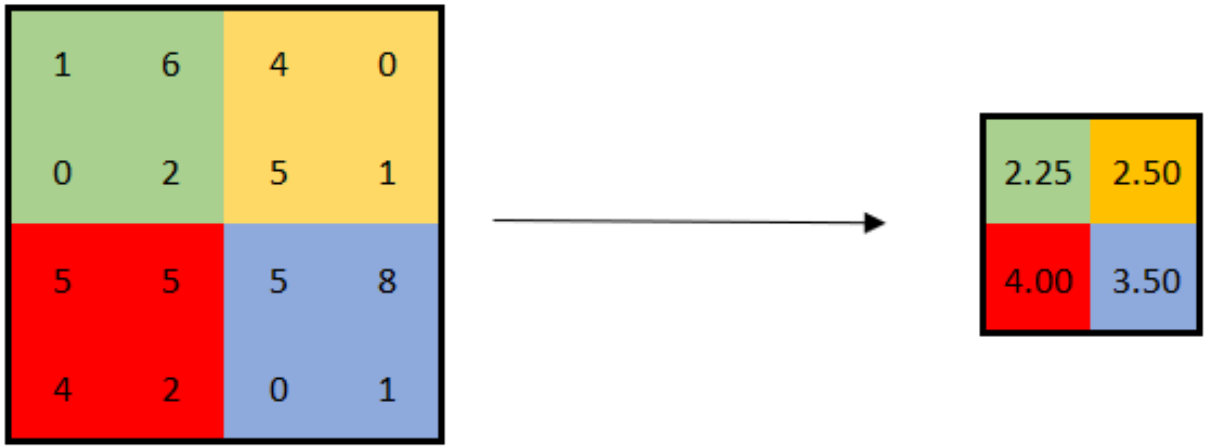
- Example with a (4×4) feature map with filter size F=2, stride S=2 and aggregation rule=max



- Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark, and we are interested in only the lighter pixels of the image.
- It can extract more pronounced features like edges.

Average Pooling

- Example with a (4×4) feature map with filter size F=2, stride S=2 and aggregation function=mean



- Average pooling method smooths out the image and hence the sharp features may not be identified when this pooling method is used.

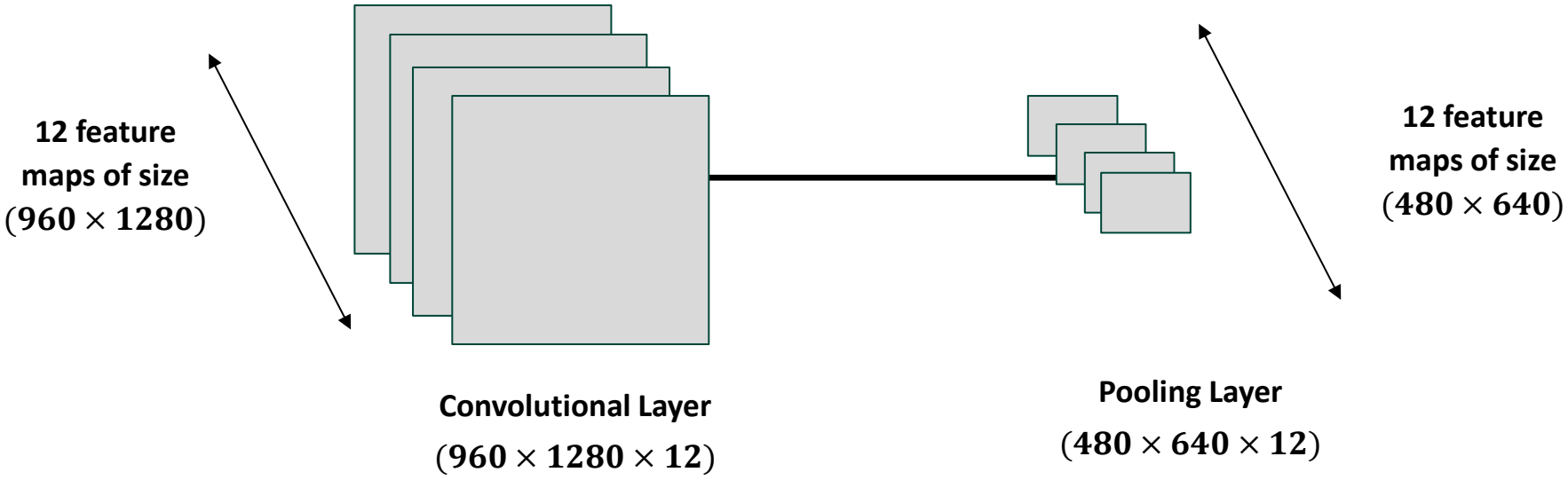
What Aggregation Function is the Best?

- In practice Max Pooling works better than Average Pooling for computer vision tasks like image classification
- One of the leaders in the field said …

"In a nutshell, the reason is that features tend to encode the spatial presence of some pattern or concept over the different tiles of the feature map (hence, the term feature map), and it's more informative to look at the maximal presence of different features than at their average presence".

Deep Learning with Python (2017), François Chollet

Pooling Layer with TensorFlow




```
model = Sequential()
model.add(Input(shape=(960,1280,3)))
model.add(Conv2D(12, kernel_size=(3,3), activation='relu',strides=1, padding='same'))
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='valid'))
model.summary()
```

```
Model: "sequential_2"
```

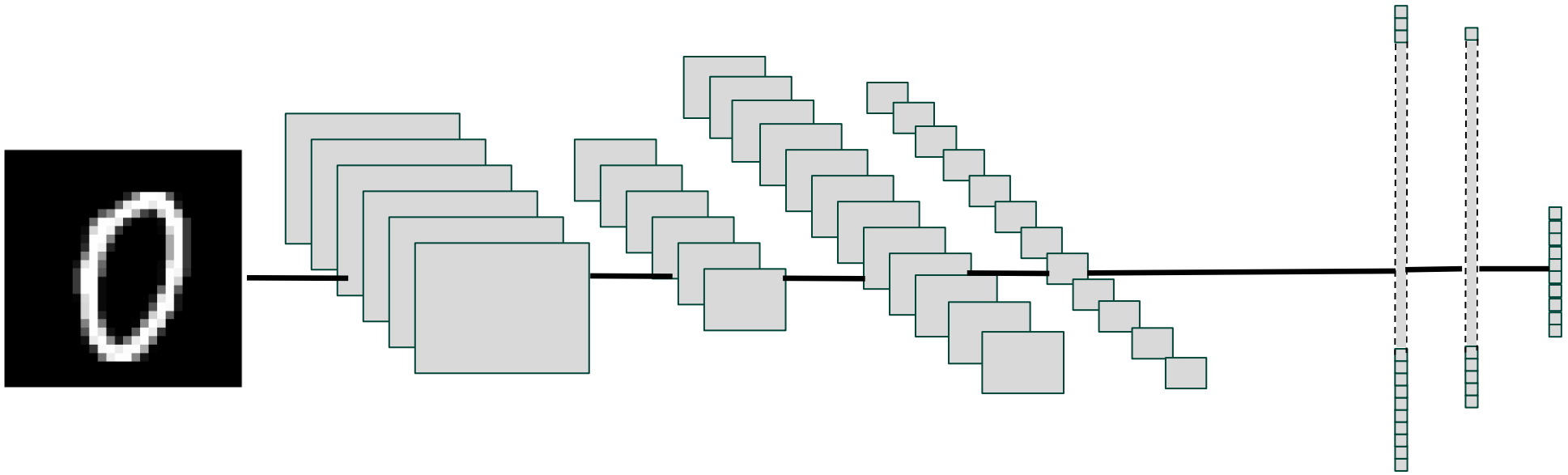
Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 960, 1280, 12)	336
max_pooling2d (MaxPooling2D)	(None, 480, 640, 12)	0

Convolutional Neural Network (CNN)

The Architecture of a CNN

- The building blocks of a CNN are:
  - **Convolutional layers**
  - **Pooling layers**
  - **Fully connected layers**
- Fully connected layers are the last few layers of the CNN.
- The input to the fully connected layer is the output from the final pooling or convolutional layer, which is flattened.
- It follows the same principles as the ANN studied in the first session of the course.

Image Classification CNN (Grayscale Image)



Input	Convolutional Layer (Group of Feature Maps)	Pooling Layer	Convolutional Layer (Group of Feature Maps)	Pooling Layer	Fully Connected Layers
	Step 1 →	Step 2 →	Step 3 →	Step 4 →	Step 5 →
$(960 \times 1280 \times 1)$	$6 \times (960 \times 1280)$ which is equivalent to $(960 \times 1280 \times 6)$	$6 \times (480 \times 640)$ which is equivalent to $(480 \times 640 \times 6)$	$12 \times (480 \times 640)$ which is equivalent to $(480 \times 640 \times 12)$	$12 \times (240 \times 320)$ which is equivalent to $(240 \times 320 \times 12)$	---------- (10×1)

Main Architecture Insights

- We start of with a large kernel that covers a lot of the image
- This extracts larger features
- Then we reduce the filter size to say 3 x 3
- However, using pooling the image size is halved at each layer
- However, the kernel stays the same size
- The kernel is seeing more of the down-sampled image
- The number of feature maps increases as we go forward
- More features can be revealed and captured

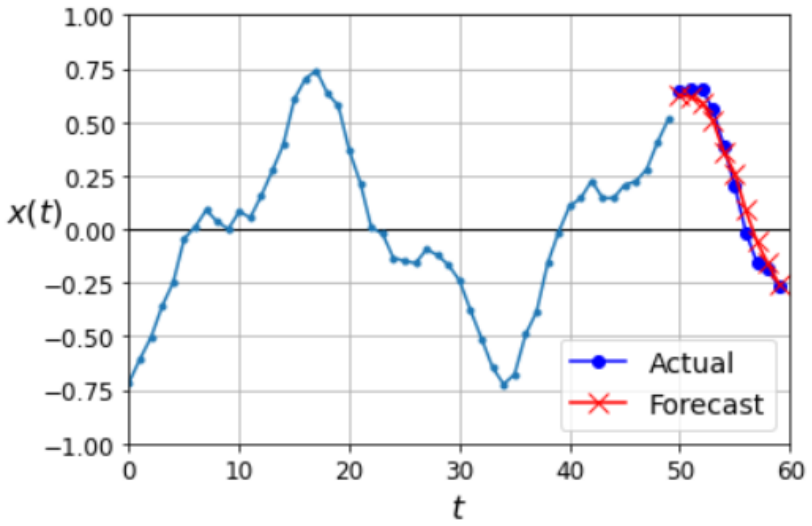
CNN for a Time Series

Remember last time – we try a CNN Layer in an RNN

- We can use a CNN Layer in time series analysis
- Here it detects features in the time series that it passes to an RN

```
model = keras.models.Sequential([
    keras.layers.Input(shape=[None, 1]),
    keras.layers.Conv1D(filters=20, kernel_size=4, strides=2, padding="valid"),
    keras.layers.GRU(20, return_sequences=True),
    keras.layers.GRU(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

- It worked well !



How to use CNNs

- You can construct a CNN that takes in a time series
- You try to use the CNN to identify features that help it to predict
- Or you can combine the pattern recognition properties of a CNN with the time series properties of an RNN

References

- Some references – some have code you can try
- https://arxiv.org/pdf/1809.04356
- https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00599-y
- https://thejaskiran99.medium.com/unlocking-the-potential-of-convolutional-neural-networks-cnns-in-time-series-forecasting-b2fac329e184
- https://royalsocietypublishing.org/doi/full/10.1098/rsta.2020.0209

Conclusions

- Convolutional networks are inspired by biology of human vision
- They use kernels to learn about the features in an image
- Kernels are translationally invariant
- Pooling layers are scale invariant
- Very powerful models which excel at image classification
- They can also be used to extract features from other data sets such as time series and text

Detailed Guidelines

- **Chapter 1**: Discussion of time series forecasting, issues, challenges, literature review, intro to Artificial Neural Nets (ANN)
- **Chapter 2**: Calculation of technical indicators. Focus on S&P 500 stocks. Use technicals in a simple NN to do single-period prediction. Examine price prediction versus return prediction. **Chapter 3**: Using TensorFlow construct a neural network with recursive neurons. Describe the simple RNN but focus on the more powerful LSTM. Consider multi-period forecasting. Analyze same questions as Chapter 2.
- **Chapter 4**: repeat Chapter 3 but using a CNN approach – do you use CNN alone or combine it with an RNN
- **Chapter 5**: Conclusions

Final Comments

- Perform cross-validation and provide results
- Describe these clearly in your report
- If you have time, consider transaction costs
- Provide a table in the conclusions comparing the different forecasting models
- It helps if you can provide the code separately in a Notebook – do not put it into your report
- Write clearly