TensorFlow & Keras for ML

TensorFlow & Keras

TensorFlow

- an open-source platform for machine learning and deep learning
- developed by Google
- flexible architecture for building, training, and deploying AI-powered applications across various platforms
  - from servers to mobile devices (e.g., TensorFlow Lite)

Keras

Keras

- high-level Python library for building neural networks on TensorFlow (and PyTorch)
  - provides pre-built components (layers, optimizers, loss functions)
- allow user to focus on problem and rapid prototyping
  - enabling users to quickly go from idea to working models

Structure of TF-Keras based code

TF-Keras based sample code structure

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

x_train, y_train),(x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
  ])
```

Structure of TF-Keras based code (cont.)

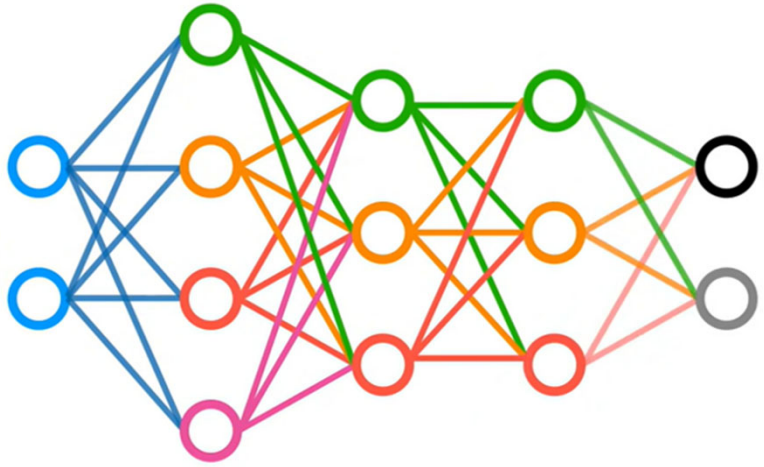
```
model.compile(
  optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Artificial Neural Network (ANN)

Artificial Neural Network

- conventional and intuitive approach
- each neuron is connected to every other neurons in the next layer
  - known as Fully connected neural network (FCNN)
  - ‘Dense’ and ‘sequential’ in Keras



FCNN for Image Classification

Consider the following handwritten image

- grayscale (0 to 255)
- assume 7 pixels x 7 pixels size (actual is 28x28)

0	0	0	0	0	0	0
0	87	240	210	24	0	0
0	13	0	101	195	0	0
0	35	167	99	210	0	0
0	145	230	240	201	189	140
0	0	102	67	17	13	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	87	240	210	24	0	0
0	13	0	101	195	0	0
0	35	167	99	210	0	0
0	145	230	240	201	189	140
0	0	102	67	17	13	0
0	0	0	0	0	0	0

FCNN for Image Classification

Flatten the 7x7 2D array to 1D array (i.e. a feature vector)

- size = 1x49

0	0	0	0	0	0	0
0	87	240	210	24	0	0
0	13	0	101	195	0	0
0	35	167	99	210	0	0
0	145	230	240	201	189	140
0	0	102	67	17	13	0
0	0	0	0	0	0	0

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 87 \\ 240 \\ 210 \\ 24 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

(Source: Youtube – codebasics)

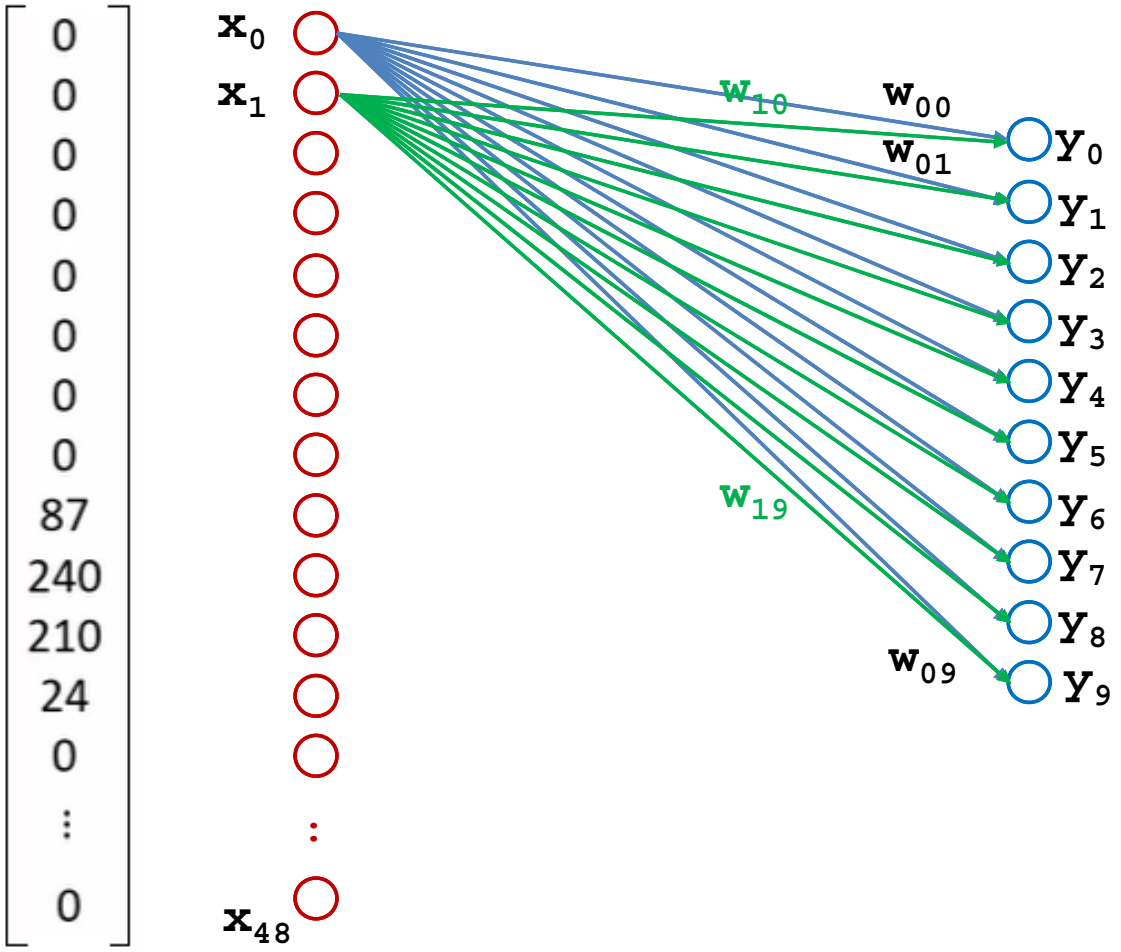
FCNN for Image Classification

Applied to a simple FCNN

- no hidden layer
- number of input node = 49
- number of output node = 10

Fully connected

⇒ each input neuron is connected to every output neurons

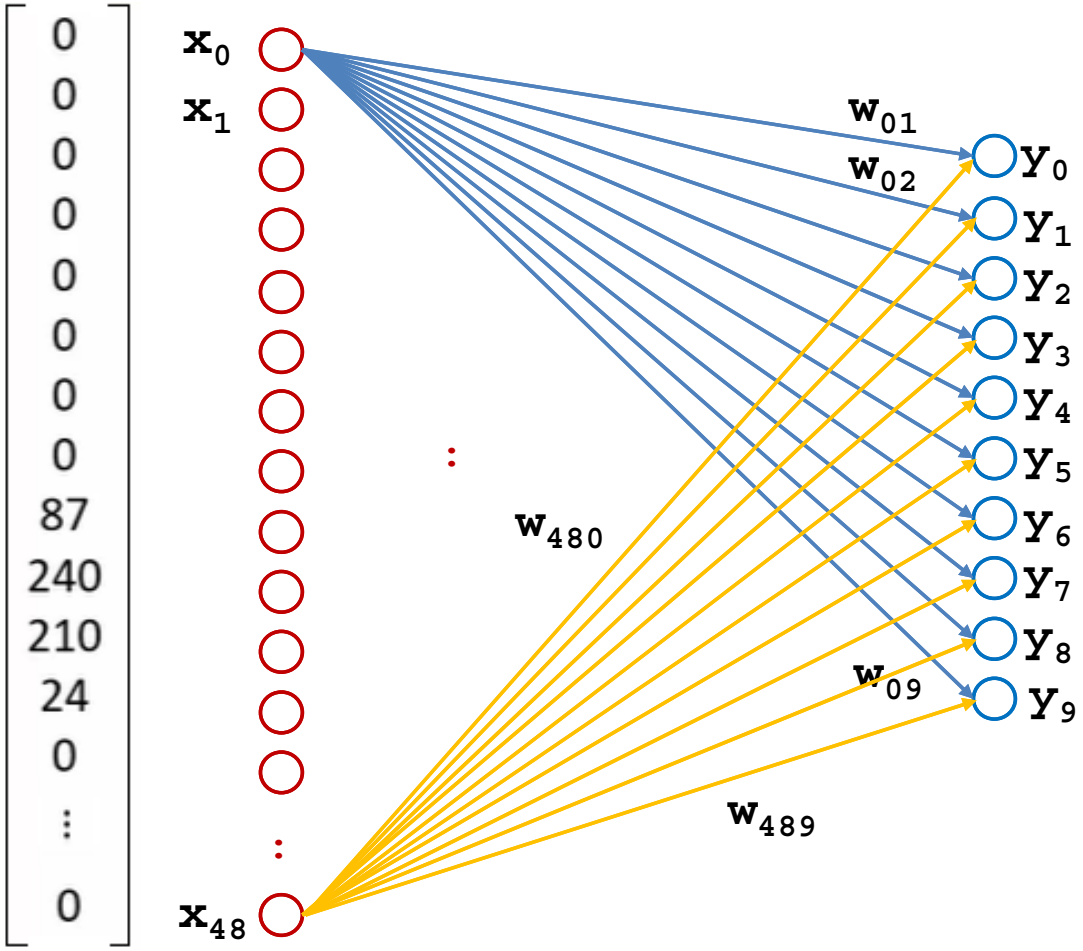


FCNN for Image Classification

- Number of connections
  = 49 x 10
  = 490
- Number of weights to adjust
  = 490

Each output node

- Softmax activation computation for multi-class classification



ANN for MNIST Image Recognition

We will use the TensorFlow (Keras) framework to code an ANN to recognize MNIST digits

- Import all necessary library

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
```

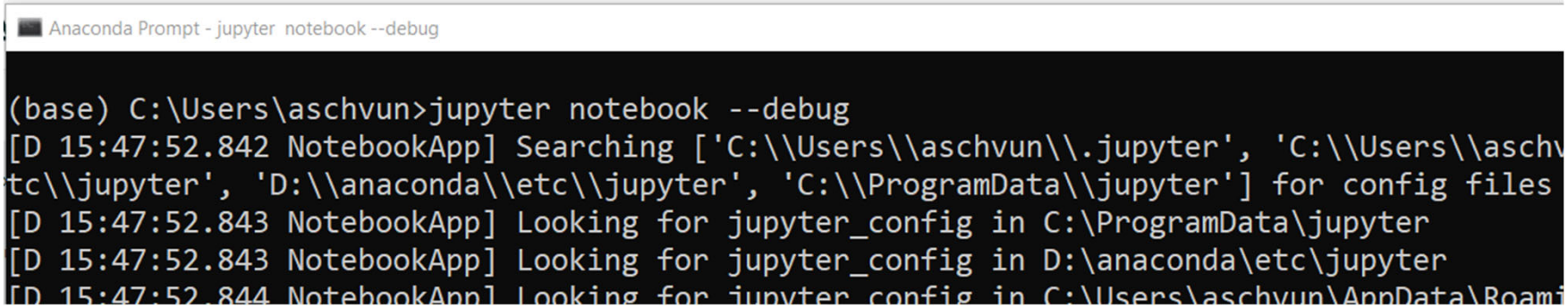
- Load the built-in MNIST dataset through Keras

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

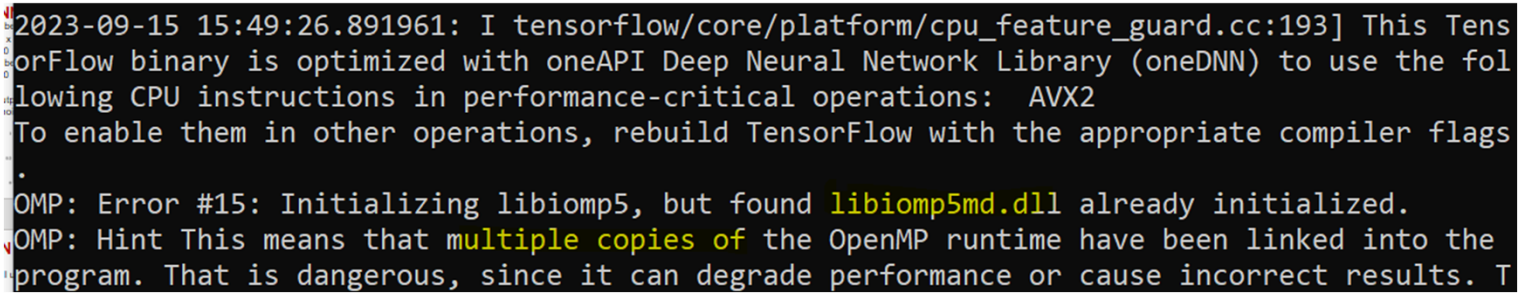
Aside: Debugging Kernel Terminate

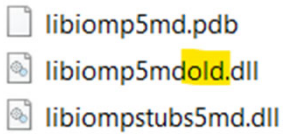
If you encounter problem (e.g., Kernel crash) during the running of your program

- launch the Jupyter Notebook through Anaconda Prompt with debug option



- Execute the script until you see the point it crashes.
- Look at the error message and google for possible remedy (e.g., rename the file)





MNIST Data

- Check the size of the dataset

```
len(X_train)
```

```
len(X_test)
```

- Check the data dimension

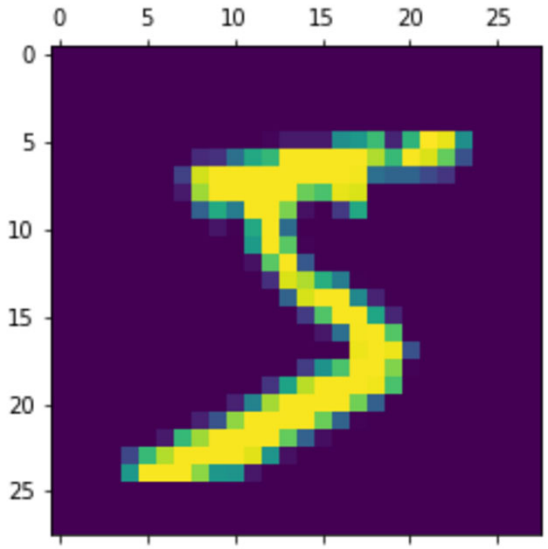
```
X_train[0].shape
```

```
X_train.shape
```

MNIST Data

- View an image

```
plt.matshow(X_train[0])
```



- Its label (for supervised training)

```
y_train[0]
```

- Check first few entries

```
y_train[:5]
```

Preprocessing the data

- Convert the data to have range with (0:1)

```
X_train[0]
```

```
X_train = X_train / 255
X_test = X_test / 255
```

```
X_train[0]
```

- Reshape the Image from 2D to 1D

```
X_train_flattened = X_train.reshape(len(X_train), 28*28)
X_test_flattened = X_test.reshape(len(X_test), 28*28)
```

```
X_train_flattened.shape
```

Setting up the Model

- A simple 1-layer FCNN model
  - Input = 784 nodes
  - Output = 10 nodes with Softmax activation function

```
model = keras.Sequential([
    keras.layers.Input(shape=(784,)), #input = 784 nodes
    keras.layers.Dense(10, activation='softmax')  # output = 10
    ])
```

Training Setup

- Optimizing algorithm for training = Adam
  - Loss parameter = cross entropy
  - Measurement metric = accuracy

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- Run the training

```
history = model.fit(X_train_flattened, y_train, epochs=10, verbose = 1)
```

Inference Performance

- Evaluate the model using the test data

```
model.evaluate(X_test_flattened, y_test)
```

- Use the model to do prediction

```
y_predicted = model.predict(X_test_flattened)
y_predicted[0] #predicted ouput based on 1st image
```

```
#check which is the biggest predicted probability
np.argmax(y_predicted[0])
```

```
#Verify against the actual input image
plt.matshow(X_test[0])
```

Inference Performance

- Process all the predicted results

```
#get all the predicted output
y_predicted_labels = [np.argmax(i) for i in y_predicted]
```

- View the first 10 predicted values

```
#see the first 10 predicted value
y_predicted_labels[:10]
```

- Compare against the actual values

```
#check against the actual first 10 value
y_test[:10]
```

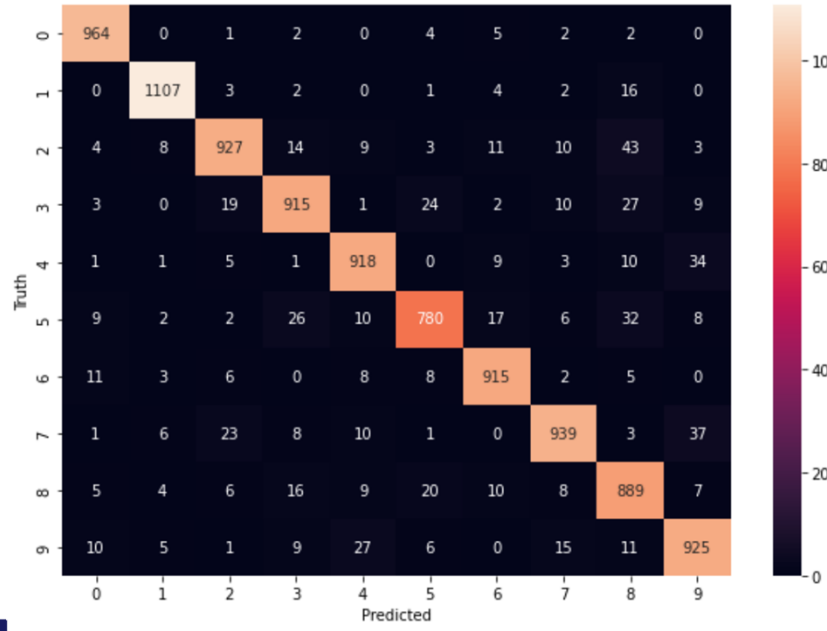
Confusion Matrix and Seaborn Heatmap

We use the confusion matrix function to provide a summary of the number of correct/incorrect predictions for each number

```
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
```

Seaborn heatmap function can then be used to visually display the accuracy of the predictions.

```
import seaborn as sns
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```



Add (more) hidden layer(s)

We can try to improve the performance by adding hidden layer

Example: A 2-layer network

- Input = 784 nodes
- Hidden layer = 128 nodes with ReLU activation function
- Output = 10 nodes with Softmax activation function

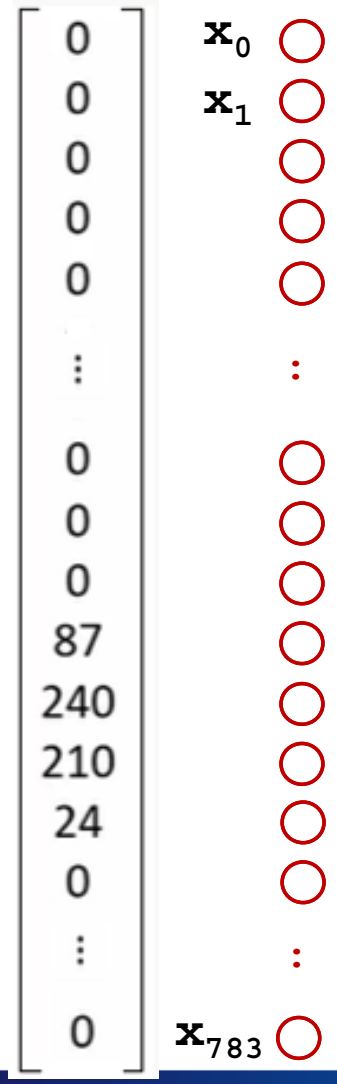
```
model = keras.Sequential([
    keras.layers.Input(shape=(784,)), #input = 784 nodes
    keras.layers.Dense(128, activation='relu'),  # hidden layer = 128 nodes
    keras.layers.Dense(10, activation='softmax')  # output = 10
    ])
```

However .....

MNIST handwritten images: 28x28 size

- number of inputs = 784

0	0	0	0	0	..	0
0	87	240	210	24	...	0
0	13	0	101	195	...	0
0	35	167	99	210	...	0
0	145	230	240	201	...	140
...	...	...	...	...	...	...
0	0	0	0	0	...	0



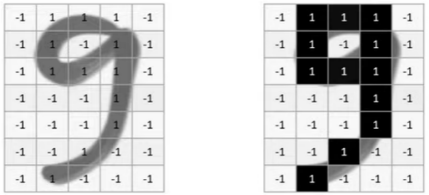
(Source: Youtube – codebasics)

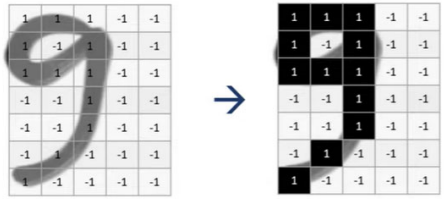
Limitations of FCNN

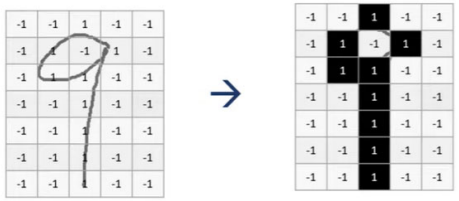
Not scalable – particularly important for memory constraint device

- E.g., high resolution colour image: 1920 x 1080 x 3 channels

Sensitivities to shift and variations





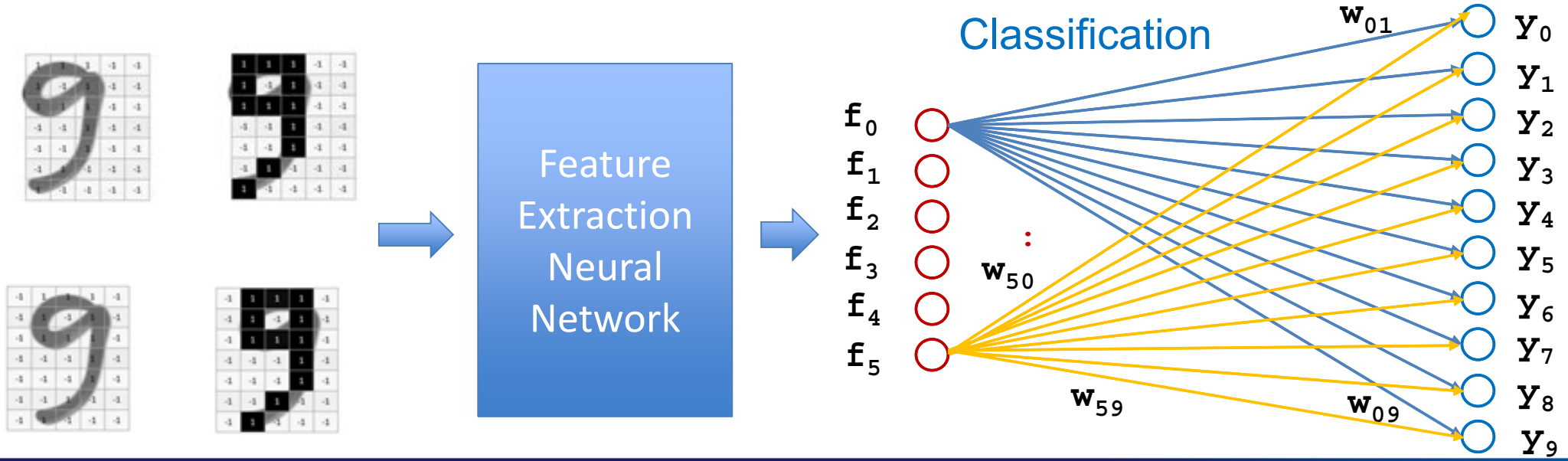


(Source: Youtube – codebasics)

‘Smarter’ Approach

Instead of doing the classification directly from (every pixel of) the raw image

- extract distinct features from (from regions of) the raw image
- then classified based on distinct features

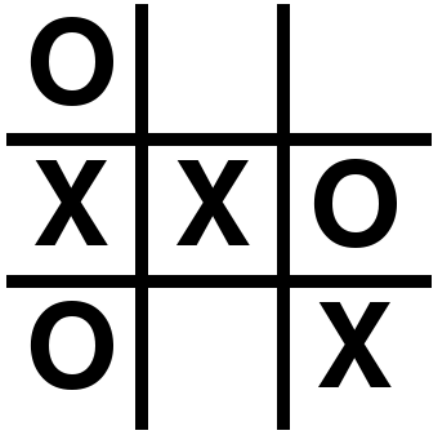


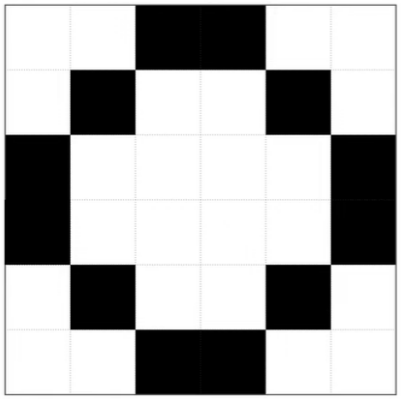
CNN for Image Classification

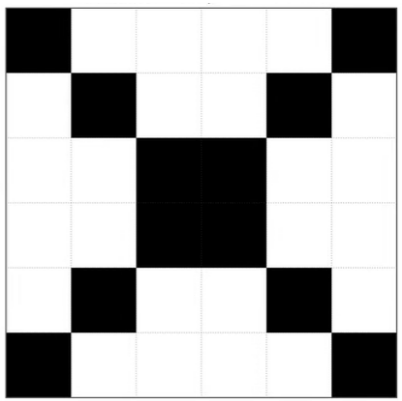
Convolutional Neural Network

- first extract distinct features from the Images
- then classify based on distinct features detected
- also reduce the number of inputs needed

Simple example: Classify the **X** and **O** in Tic-Tac-Toe game

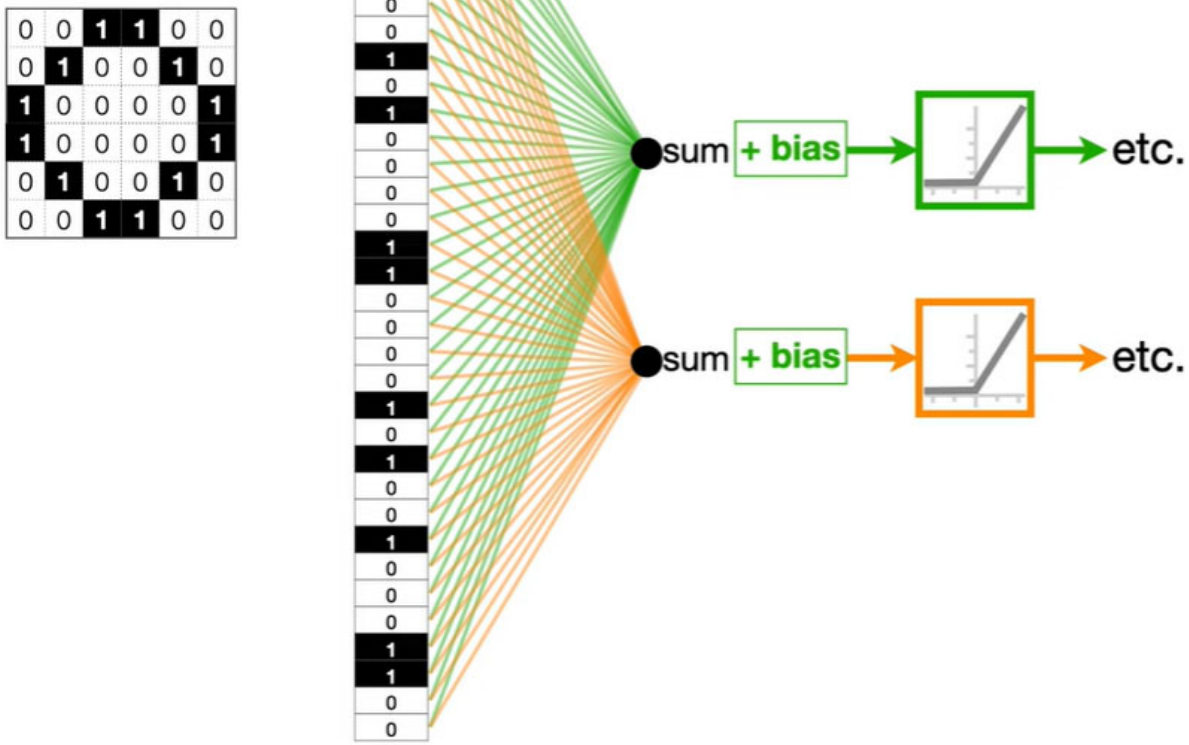




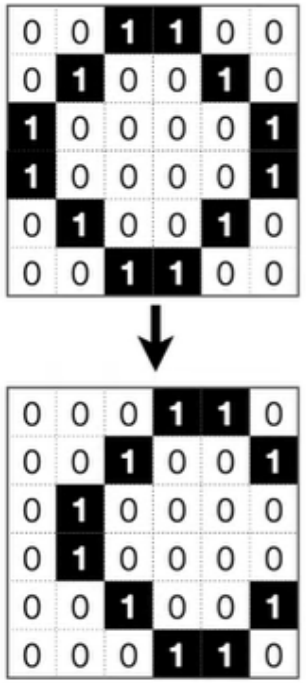


(Source: Youtube – StatQuest)

If using FCNN Approach

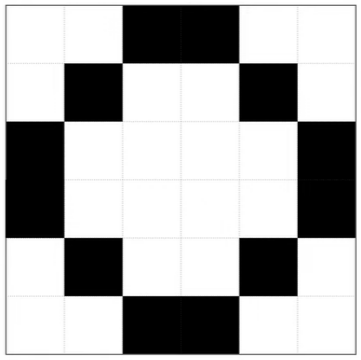
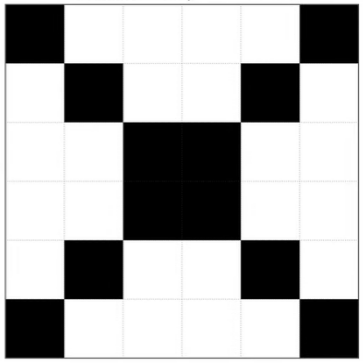
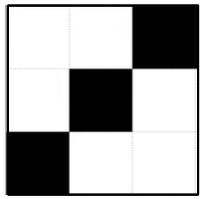
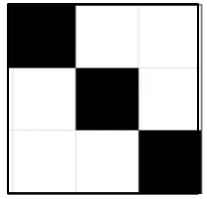


What if ?



(Source: Youtube – StatQuest)

Features Detection Approach

Distinct feature

- shape is definitely different

Basic features

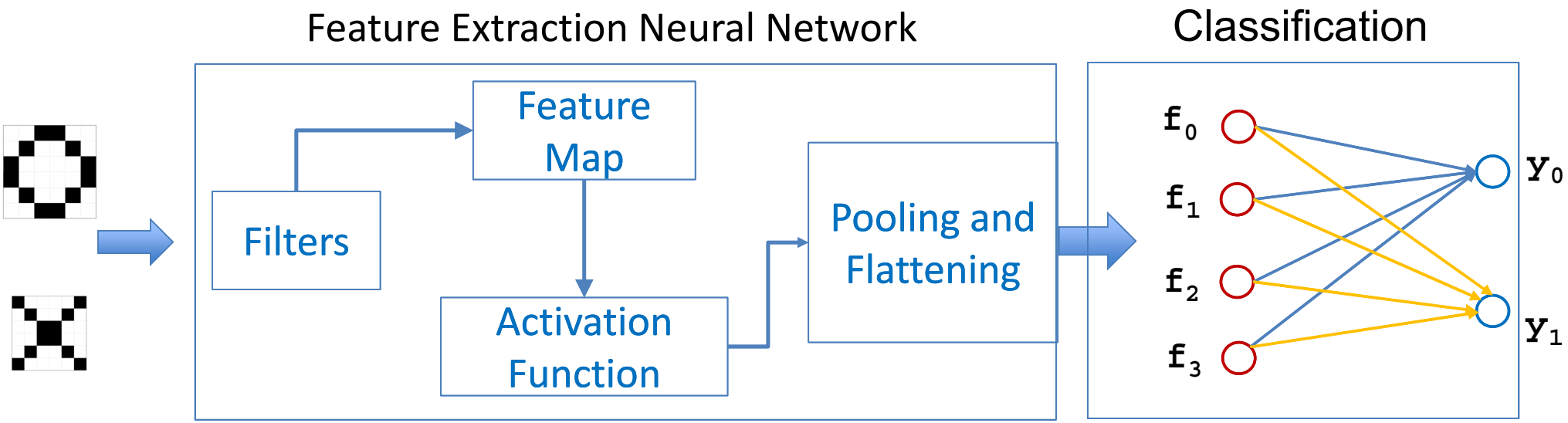
- both shapes contain (only) two basic (and common) features that are arranged in different way.

A properly trained Model would have detected these two basic features and use them to run inference.

CNN Approach

Instead of doing the classification directly from the raw image

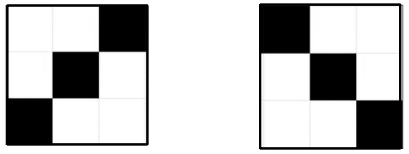
- extract distinct features from the raw image using filters (kernels)
- classified based on distinct features

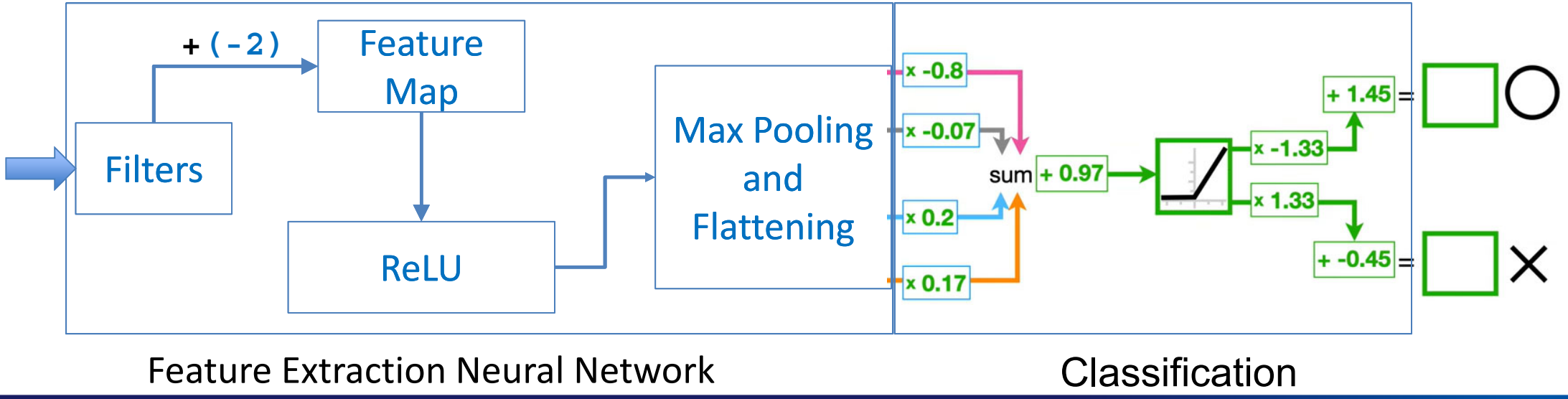


CNN Image Classification Model

Assuming that we have a trained model to classify the ‘X’ and ‘O’ as shown below

- Filters (kernel) to be used to run inference



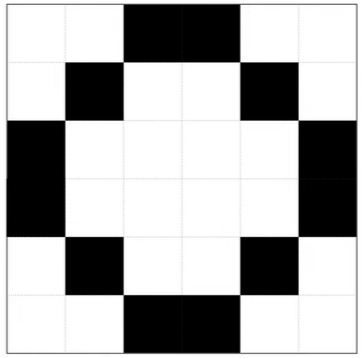


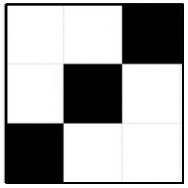
Feature Extraction Neural Network

Classification

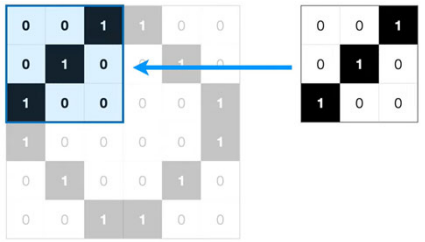
Filter Operation (2D Conv)

Convoluted the filter with the input image





- Executing the dot-product (aka correlation)



(Source: Youtube – StatQuest)

$(0 \times 0) + (0 \times 0) + (1 \times 1)$

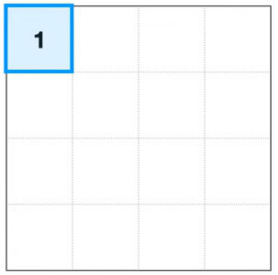
$+ (0 \times 0) + (1 \times 1) + (0 \times 0)$

$+ (1 \times 1) + (0 \times 0) + (0 \times 0)$

$= 3$

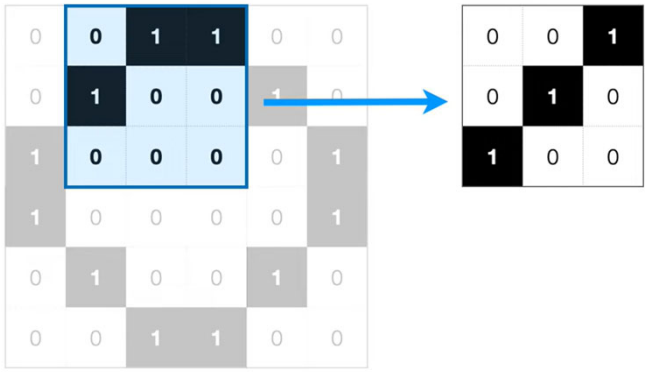
$+ (-2)$

Feature Map



Filter Operation (cont.)

Shift to the next pixel

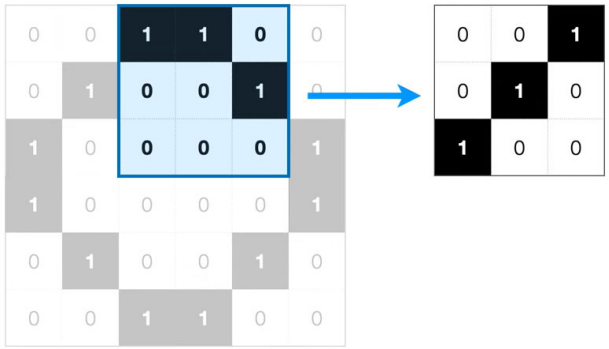


$$(0 \times 0) + (1 \times 0) + (1 \times 1)$$
$$+ (1 \times 0) + (0 \times 1) + (0 \times 0)$$
$$+ (0 \times 1) + (0 \times 0) + (0 \times 0)$$
$$= 1$$

+ (-2)

Feature Map

1	-1		



$$(1 \times 0) + (1 \times 0) + (0 \times 1)$$
$$+ (0 \times 0) + (0 \times 1) + (1 \times 0)$$
$$+ (0 \times 1) + (0 \times 0) + (0 \times 0)$$
$$= 0$$

+ (-2)

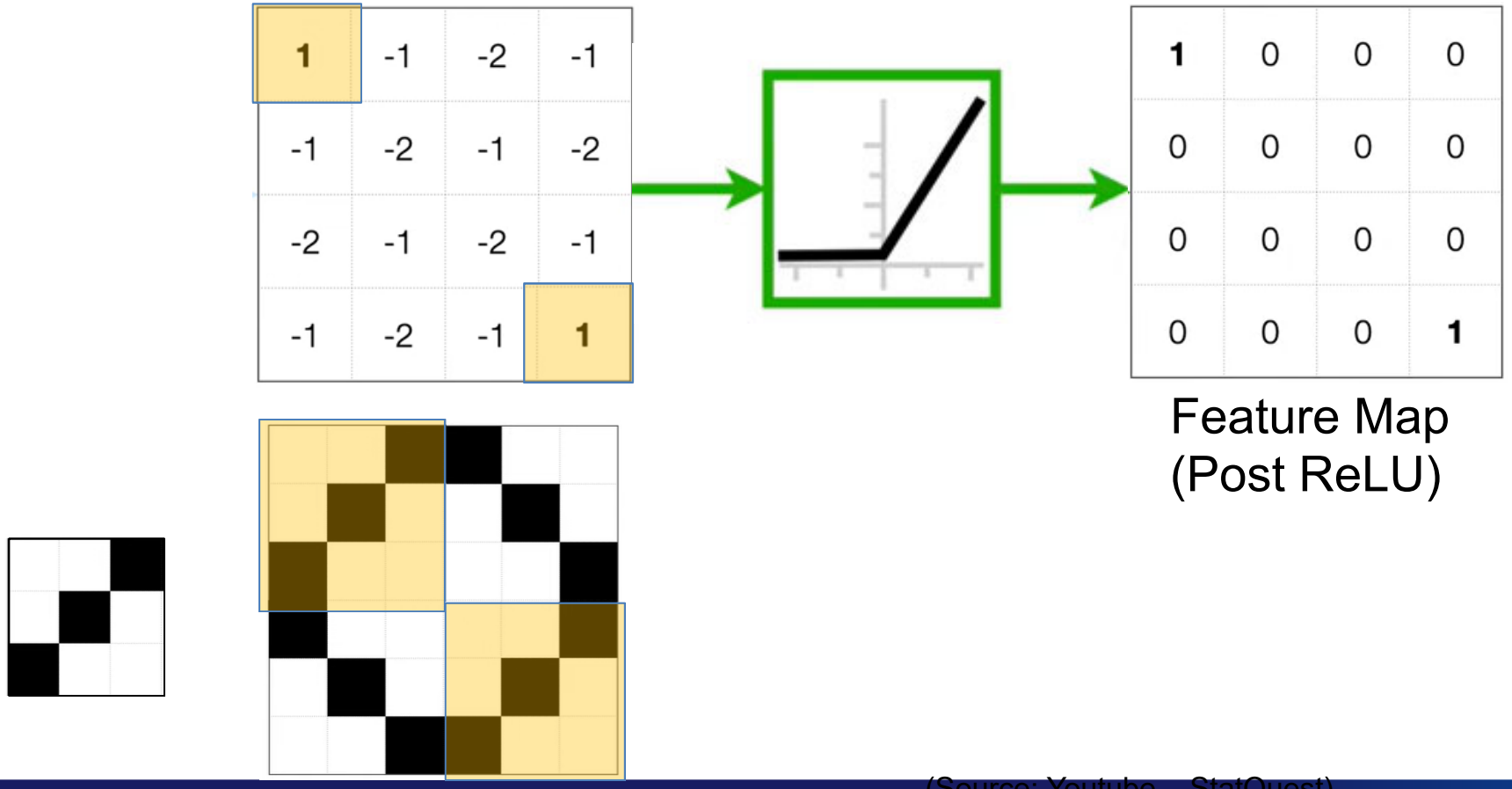
Feature Map

1	-1	-2	

(Source: Youtube – StatQuest)

Feature Map

Applied to ReLU

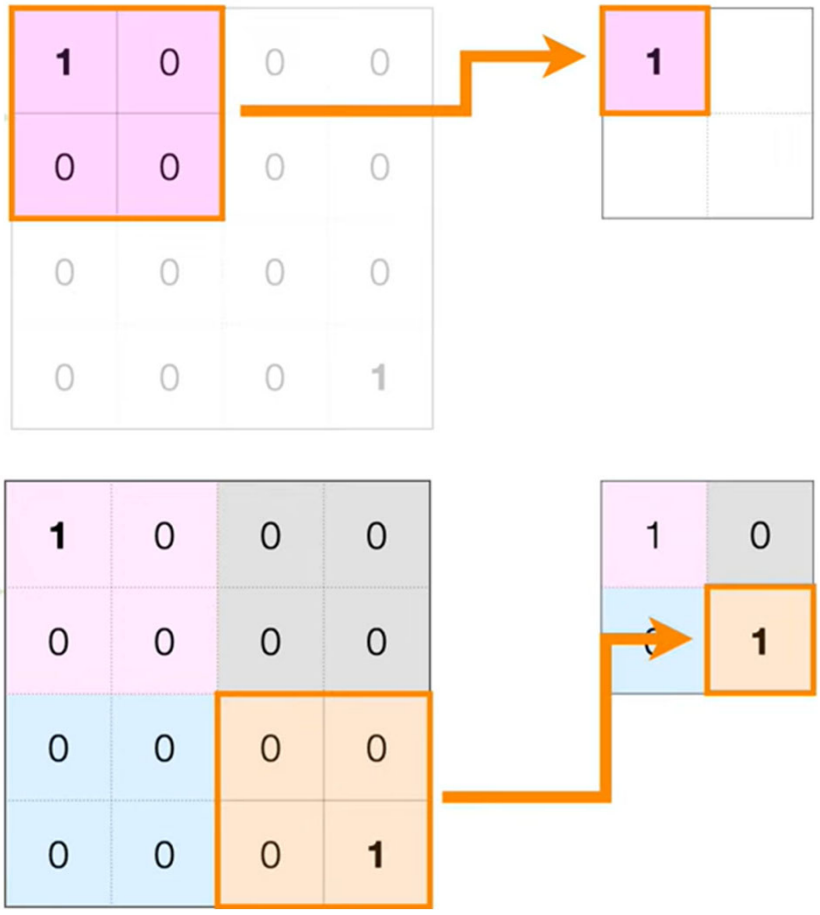


(Source: Youtube – StatQuest)

Max Pooling Feature Map

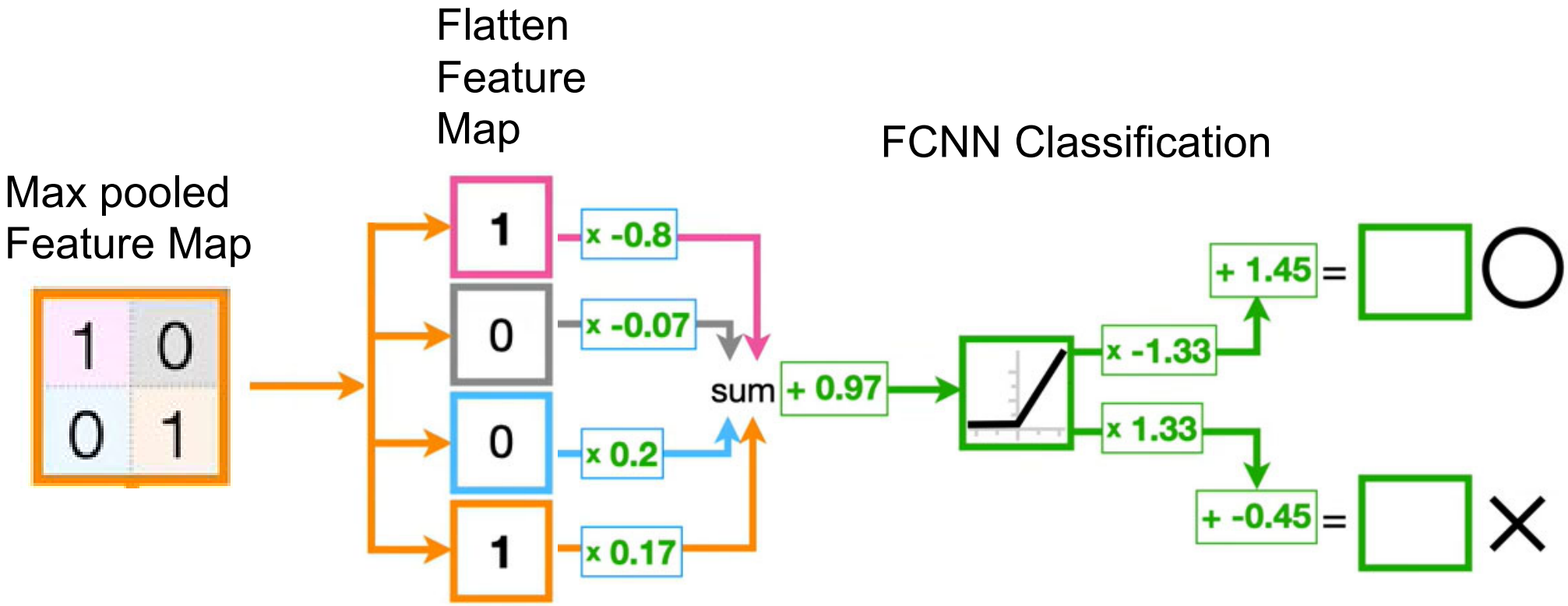
1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	**1**

Feature Map
(Post ReLU)



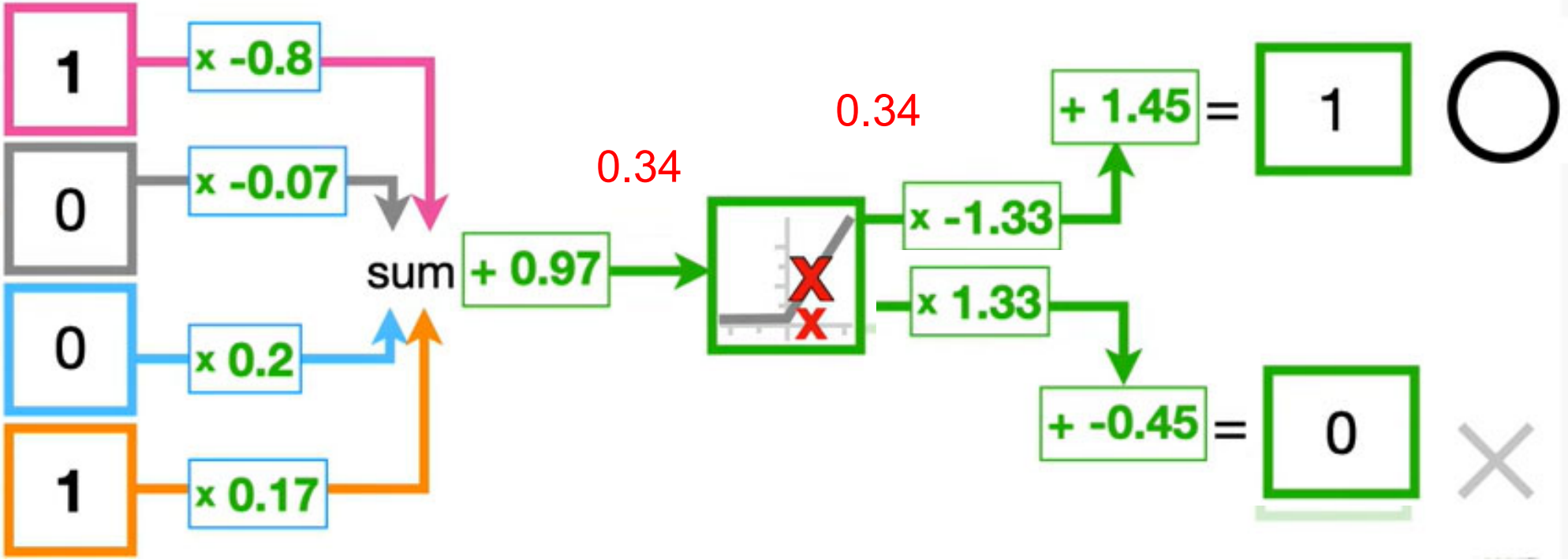
(Source: Youtube – StatQuest)

Flatten for ANN Classification



(Source: Youtube – StatQuest)

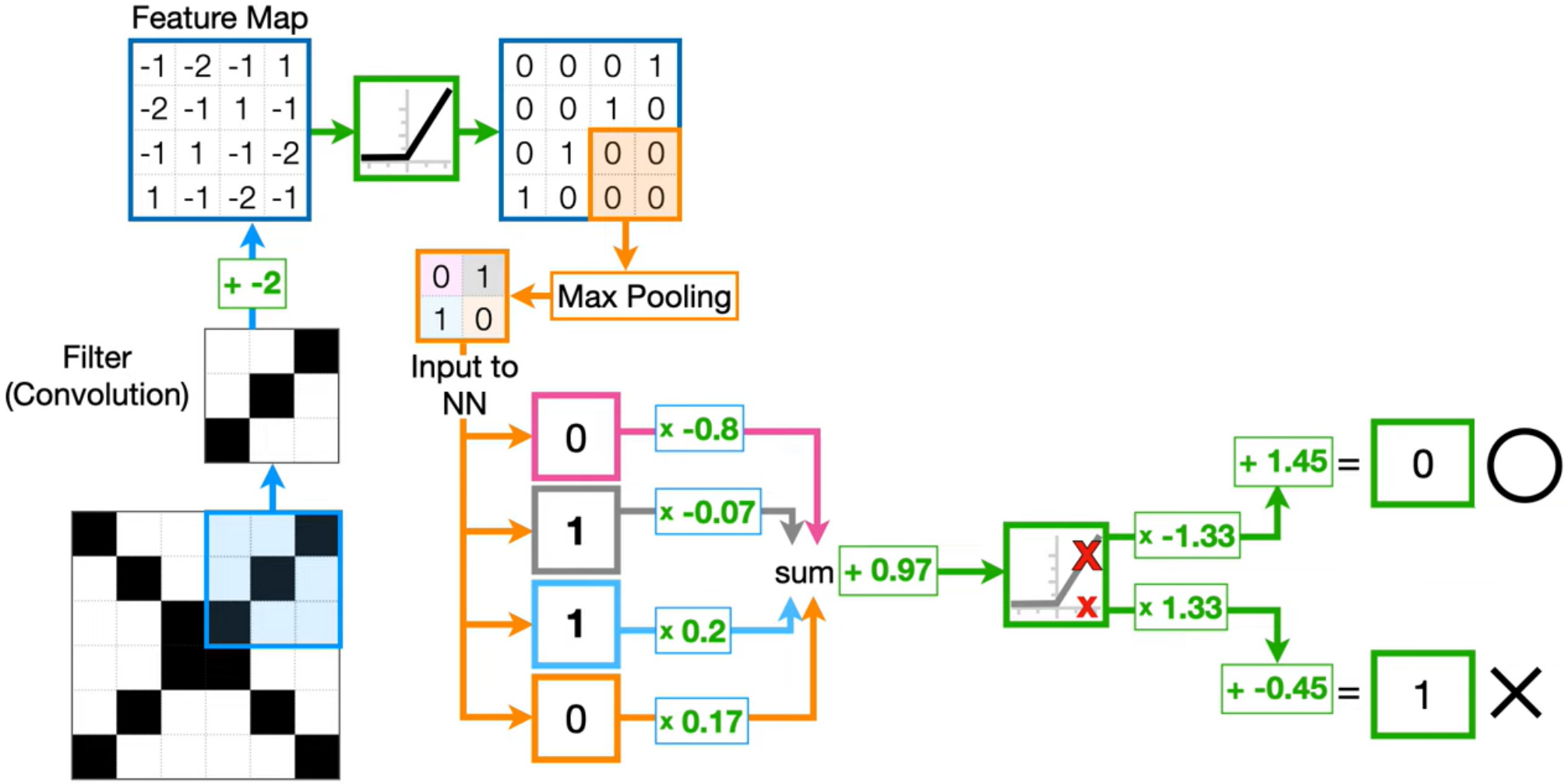
Classification for ‘O’



$$(1 \times -0.8) + (0 \times -0.07) + (0 \times 0.2) + (1 \times 0.17) + 0.97 = 0.34$$

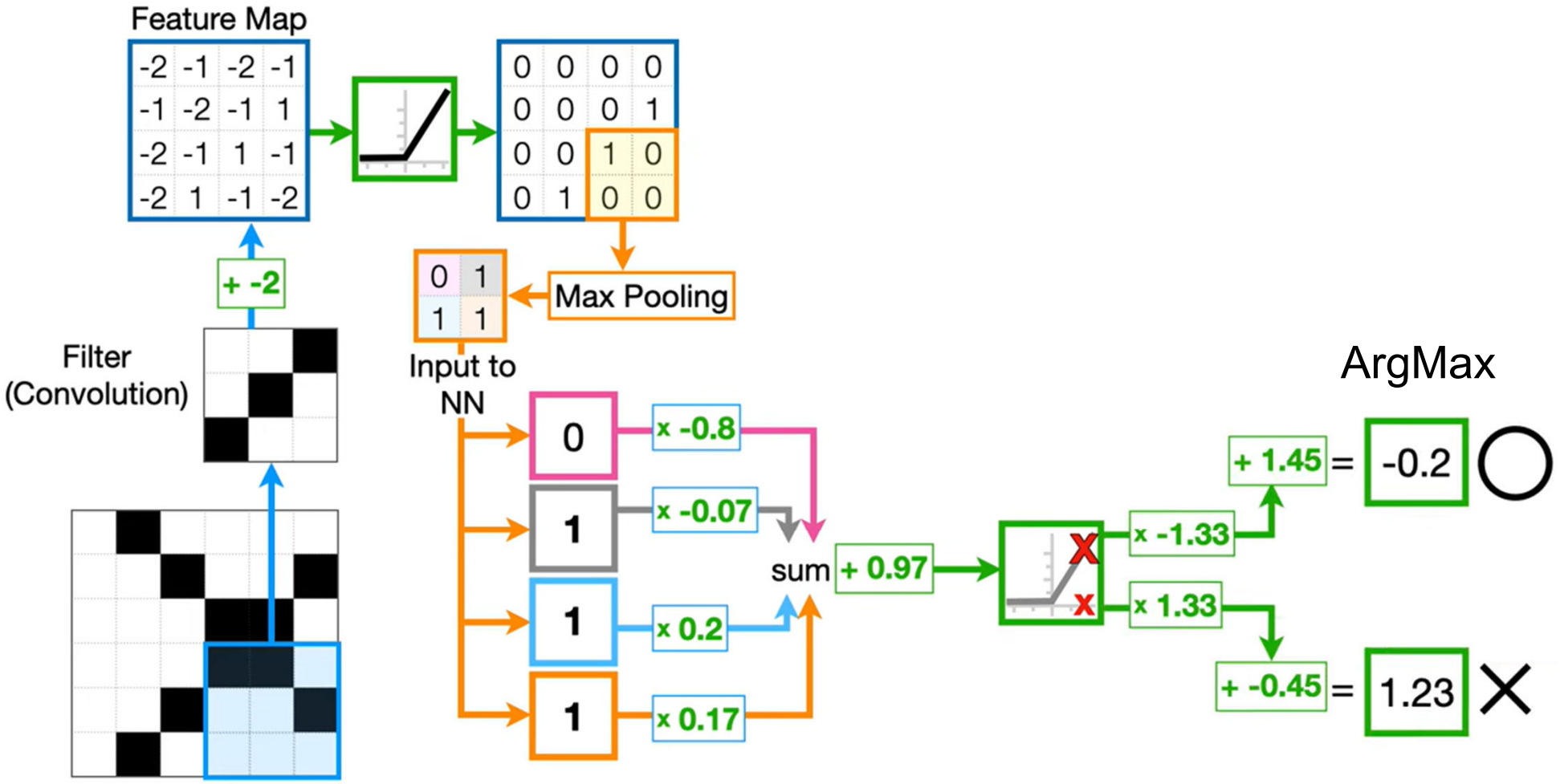
(Source: Youtube – StatQuest)

Classification for ‘X’



(Source: Youtube – StatQuest)

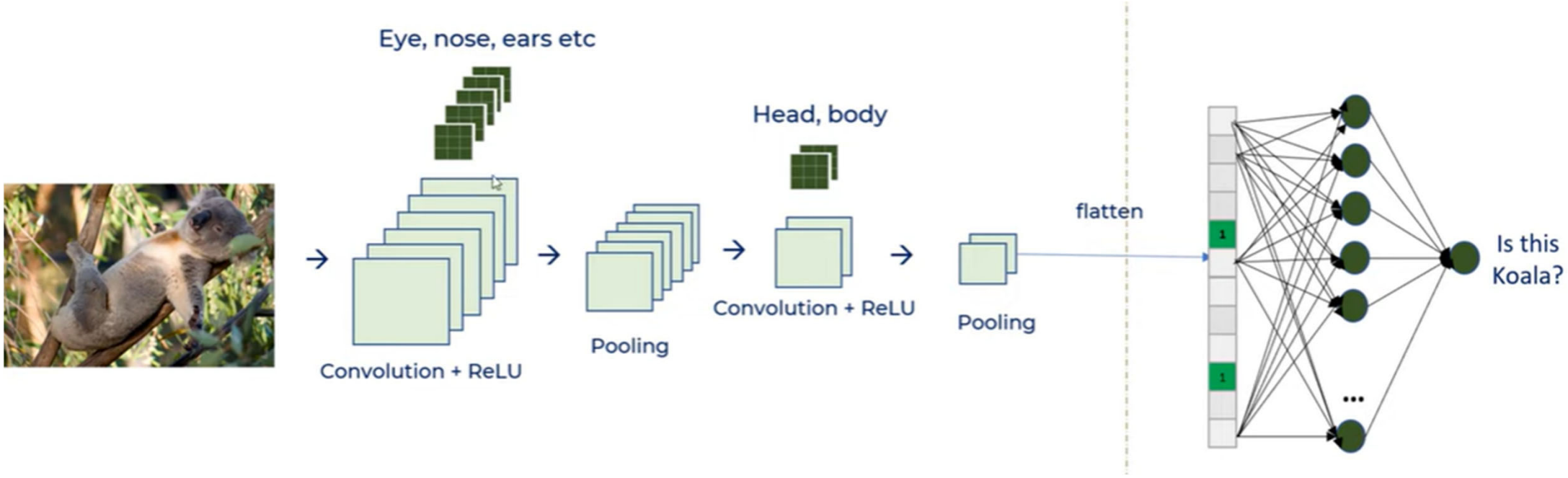
Classification for shifted ‘X’



Practical CNN structure

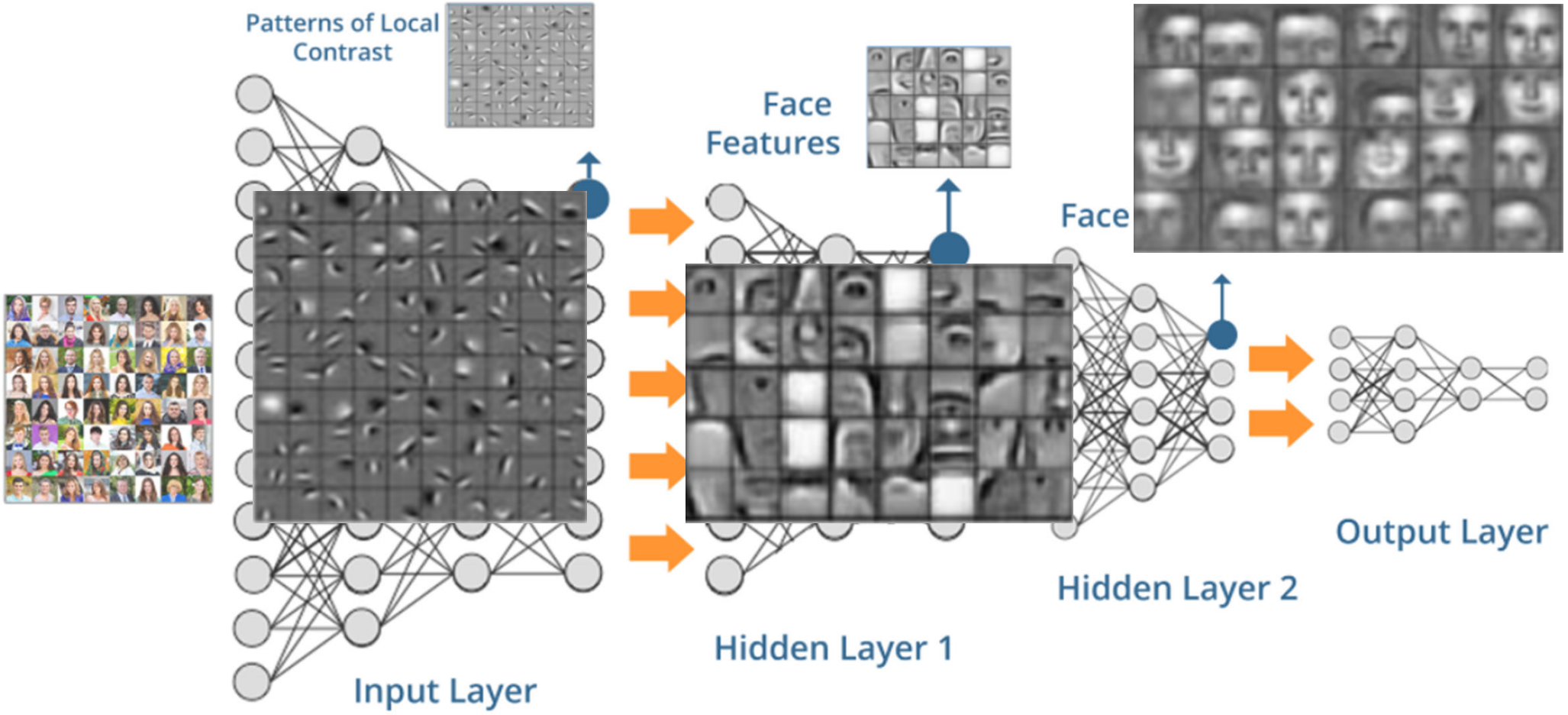
In practice

- multiple levels of convolution and pooling layers are used
- each level extract basic features, which are analysed by the subsequent layer(s)



(Source: Youtube – codebasics)

Face Detection Training



CNN for MNIST Image Recognition

We will use Keras and TF to define a CNN model to recognize MNIST digit

- Import all necessary library

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
```

- Load the built-in MNIST dataset through Keras

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

```
X_train.shape[0]
```

```
]: 60000
```

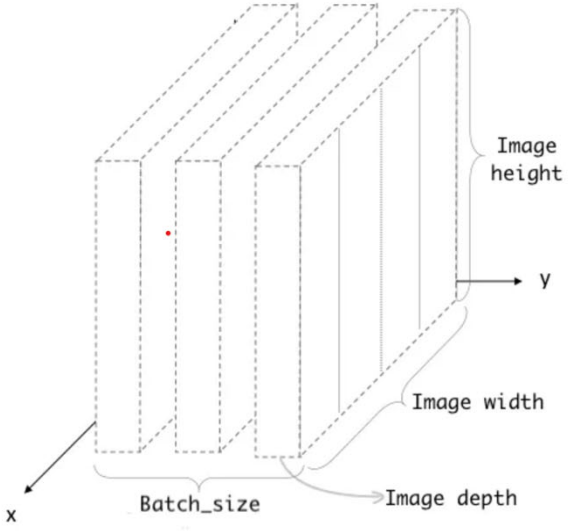
```
X_test.shape[0]
```

```
]: 10000
```

CNN for MNIST Image Recognition

Reshaping the data to 4-dimension needed by Keras API

```
#Keras API, we need 4-dims NumPy arrays
# Reshaping the array to 4-dims so that it can work with the Keras API
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
```



CNN input: 4D array with a shape of (batch_size, height, width, depth)

Height and width = image size

depth = channel: color = 3, greyscale = 1

CNN for MNIST Image Recognition

Reshaping the data to 4-dimension needed by Keras KPI

```
print('X_train shape:', X_train.shape)
print('Number of images in X_train', X_train.shape[0])
print('Number of images in X_test', X_test.shape[0])
```

```
X_train shape: (60000, 28, 28, 1)
Number of images in X_train 60000
Number of images in X_test 10000
```

```
# Normalizing the RGB codes
X_train /= 255
X_test /= 255
```

CNN for MNIST Image Recognition

Import the required Keras modules

```
# Importing the required Keras modules containing model and layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Conv2D, Dropout, Flatten, MaxPooling2D
```

Define the CNN model

```
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Input(shape=input_shape))
model.add(Conv2D(28, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu)) #add another hidden layer
model.add(Dropout(0.2)) #dropout layers to reduce overfitting
model.add(Dense(10,activation=tf.nn.softmax))
```

CNN for MNIST Image Recognition

Check the CNN model: model.summary

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 28)	280
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 28)	0
flatten_1 (Flatten)	(None, 4732)	0
dense_2 (Dense)	(None, 128)	605,824
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

Total params: 607,394 (2.32 MB)

Trainable params: 607,394 (2.32 MB)

Non-trainable params: 0 (0.00 B)

CNN for MNIST Image Recognition

Specify the optimizer algorithm and measurement parameters

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Train the model (with 10% of the data reserved for validation)

```
model.fit(x=X_train,y=y_train, epochs=5, validation_split=0.1)
#Model is being trained on 1875 batches of 32 images each (default batch size)
#1875*32 = 60000 images
#Validation = 10%
#Training = 1875 * 0.9 = 1688
```

Graph the Training History

To visualise the loss progression during the training

```
train_history=model.fit(x=X_train,y=y_train, epochs=50, validation_split=0.2)
#Graph the training history
loss = train_history.history['loss']
val_loss = train_history.history['val_loss']
epochs = range(1, len(loss)+1)
plt.plot(epochs, loss, 'g', label = 'Training loss')
plt.plot(epochs, val_loss, 'b', label = 'Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show
```

Training Loss Progression

CNN for MNIST Image Recognition

Run model against Test data to evaluate accuracy of model

```
#evaulate performance based on test data
baseline_model_accuracy = model.evaluate(X_test, y_test)
print('Baseline test accuracy:', baseline_model_accuracy)
```

Get the predicted output values

```
y_predicted = model.predict(X_test)
```

CNN for MNIST Image Recognition

Generate the confusion matrix and heatmap

```
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
import seaborn as sns
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

CNN for MNIST Image Recognition

Compare against the FCNN heatmap