Lab2 - AI Framework and Practices

Advisor：Tsai, Chia-Chi

TA：楊宗軒

Outline

1. Lab2 task
2. Framework introduction
3. Layer introduction
4. Sample code introduction

Outline

1. Lab2 task
2. Framework introduction
3. Layers introduction
4. Sample code introduction

1. Lab2 task

Goal:

- AI framework and deep learning layer overview.
- Learn to design, train, evaluate and implement your own CNN/NN model on MNIST dataset.

1. Lab2 task

- Recommend platform: colab with python

- Upload platform: NCKU moodle
- Upload compressed StudentID_lab2.zip file including:
  - 學號_lab2.ipynb
    - IPython notenook 須包含程式碼跟結果
  - 學號_lab2.pdf
    - 各項print以及plot輸出
    - 實作所運到的困難及解決方法

- Provided file: lab2 sample code
- Q&A: course.aislab@gmail.com

1. Lab2 task

1) Use pytorch or Tensorflow to implement an specific classification DNN model by sequential model, dataset MNIST
   - 3 layer of CNN network (10%)
   - 3 layer of NN network (10%)

2) Comparison w/ and w/o Batch Normalization Layer (10%)

3) Comparison w/ arbitrary layer of abovementioned CNN network. (10%)
   - The relationship between layers, model capacity(parameters) and accuracy.

4) Print model summary(including parameters) and plot model (10%)

5) Print test accuracy, plot epoch-train accuracy, epoch-val accuracy, epoch-train loss, epoch-val loss. (10%)

6) Plot certain image from dataset and successively predict (10%)

7) Report(30%)

1. Lab2 task

- Additional information:

Slide 6 term(1) means **1 model contains 3 convolution layers and 3 fully connected layers**, as a baseline for comparison. So the .ipynb should include the final model, report should include the following content:

1. Baseline model (3 conv2D + 3 dense), print model summary and plot model. (model summary, model plot and result)

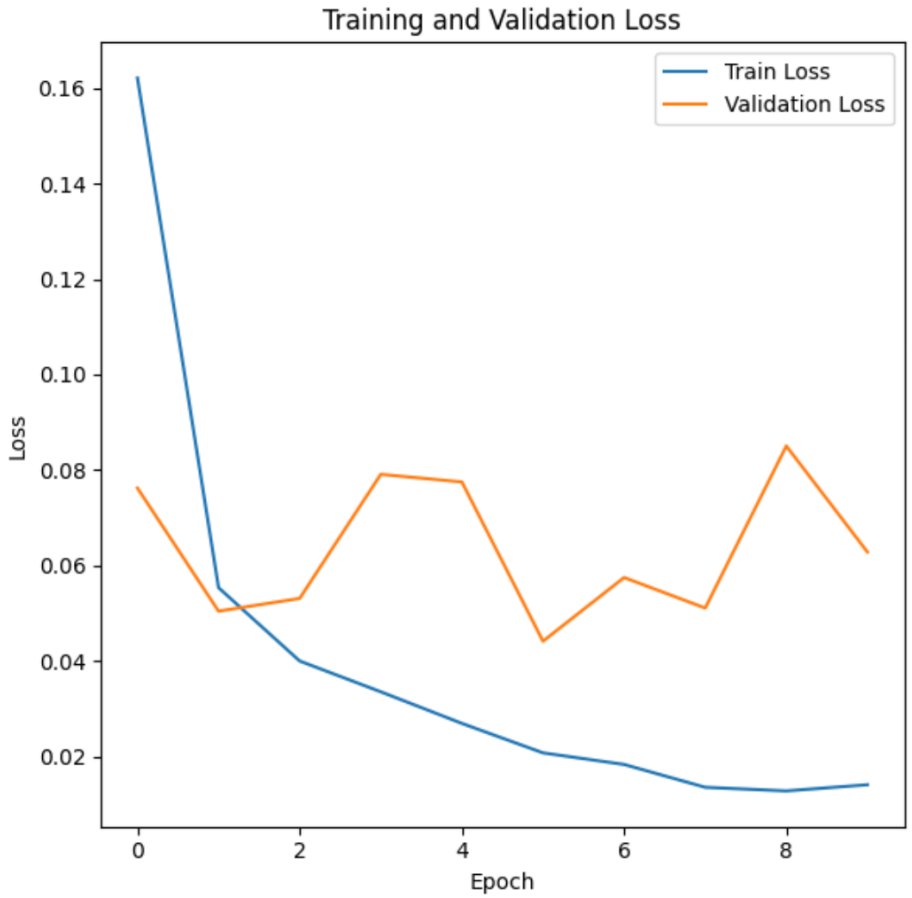
2. Baseline model vs baseline model+BN comparison. (baseline model+BN result)

3. Baseline model vs arbitrary layer of conv2D model comparison. (arbitrary layer of conv2D model result)

4. Final adjusted and tuned model, print model summary and plot model, predict the image from test set. (model summary, model plot and result)

Result: Means including printed train accuracy, validation accuracy, train loss, validation loss for each epoch

1. Result Example



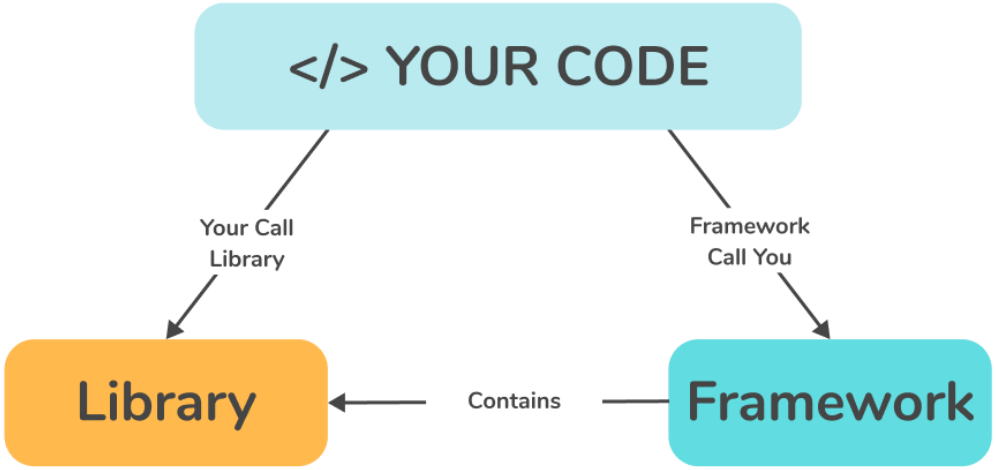

```
test accuracy: 0.9805
```

Outline

1. Lab2 task
2. Framework introduction
3. Layers introduction
4. Sample code introduction

2. Framework introduction

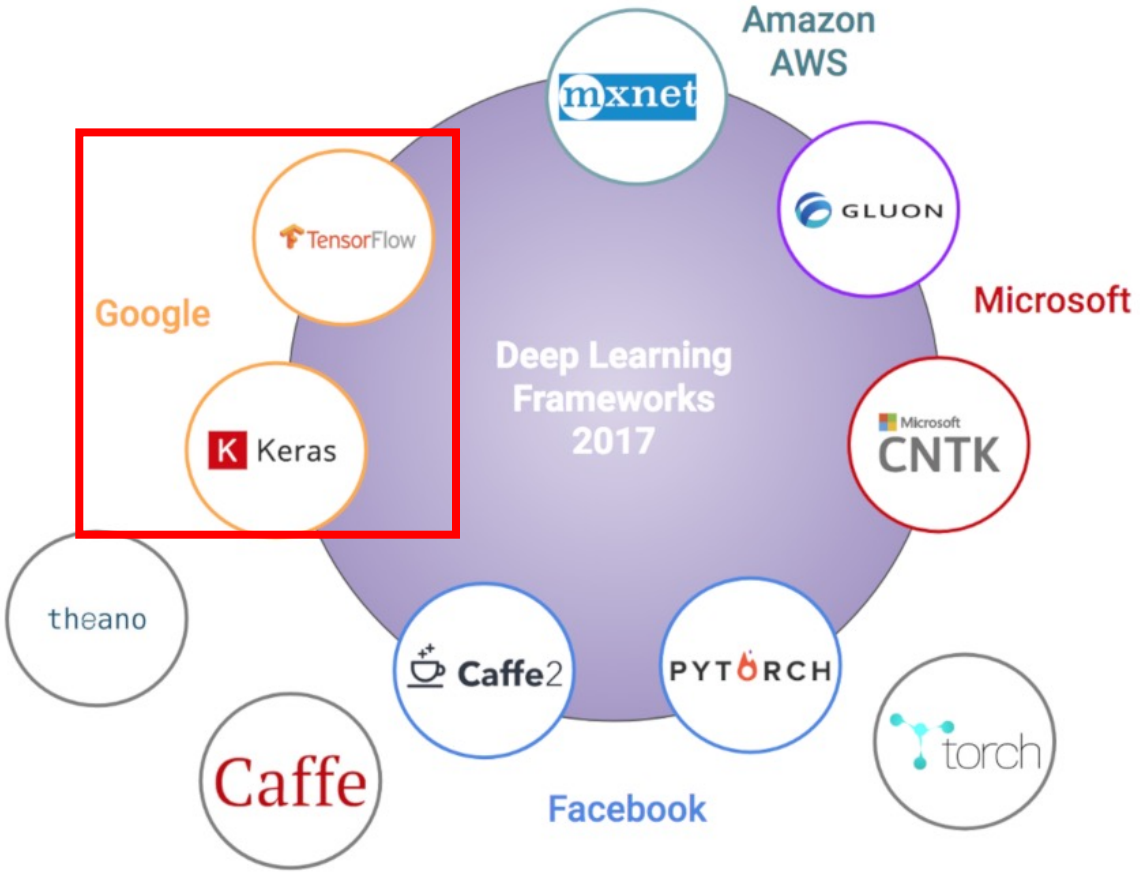
- What is framework?
- Why do we need framework?
- Library vs framework, e.g. Numpy vs Tensorflow

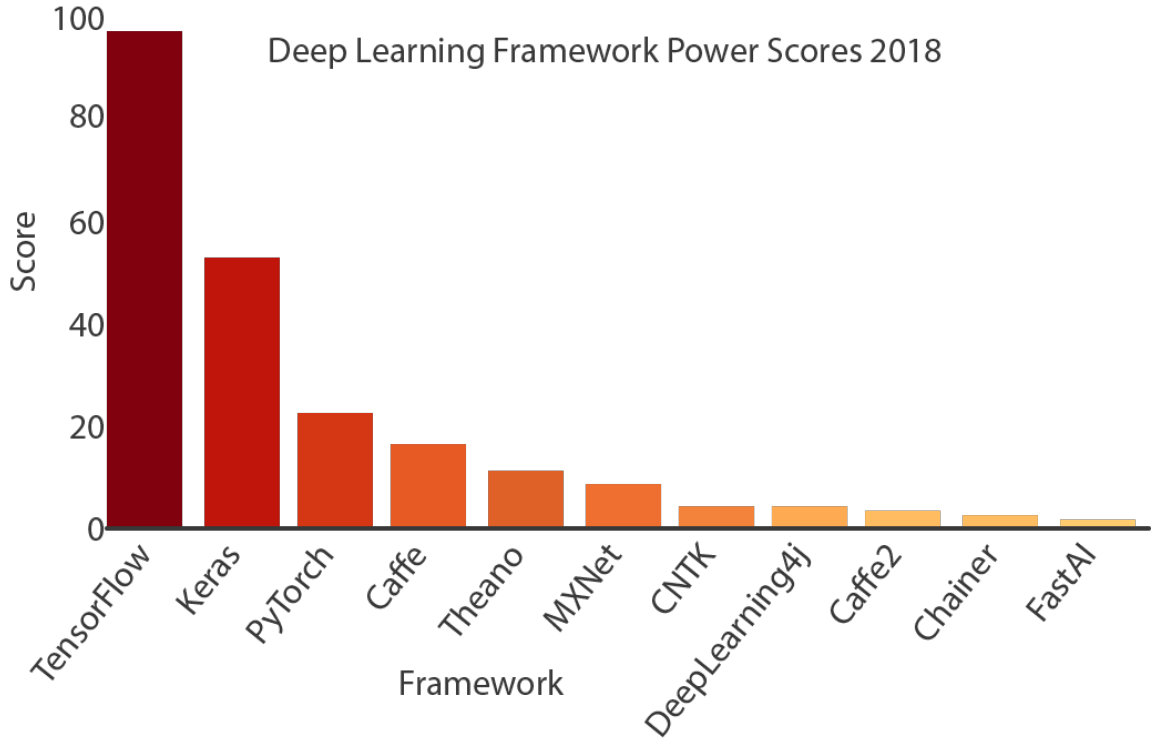


Parameters	Library	Framework
Definition	Libraries provide developers with predefined functions and classes to make their work easier and boost the development process.	Framework, on the other hand, is like the foundation upon which developers build applications for specific platforms.
Inversion of Control	By using a library, you can control the flow of the application and call the library.	In contrast, when you use a framework, the control is inverted, i.e., the framework controls the flow and calls your code.

https://www.interviewbit.com/blog/framework-vs-library/

2. Framework introduction

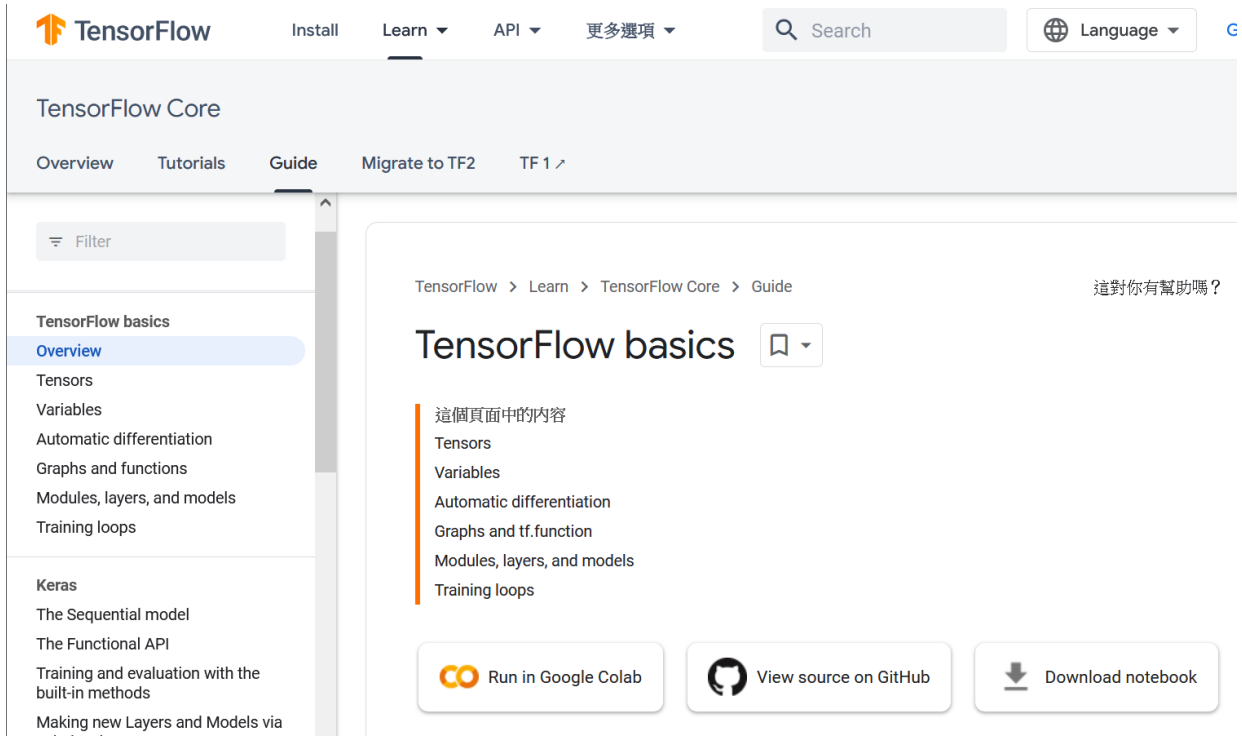




https://www.google.com/url?sa=i&url=https%3A%2F%2Fdevopedia.org%2Fdeep-learning-frameworks&psig=AOvVaw3D2GjOCj0j0ZzY7Rw17wD2&ust=1664229095612000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCIDZ64_3sPoCFQAAAAAdAAAAABAD

2. Framework introduction

- Enormous amount of resources for tf and keras.
- Sample code: Tensorflow 2.X(framework) and Keras(Libraries)



Load a dataset

Load and prepare the MNIST dataset. Convert the sample data from integers to floating-point numbers:

```
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
```

Build a machine learning model

Build a `tf.keras.Sequential` model by stacking layers.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

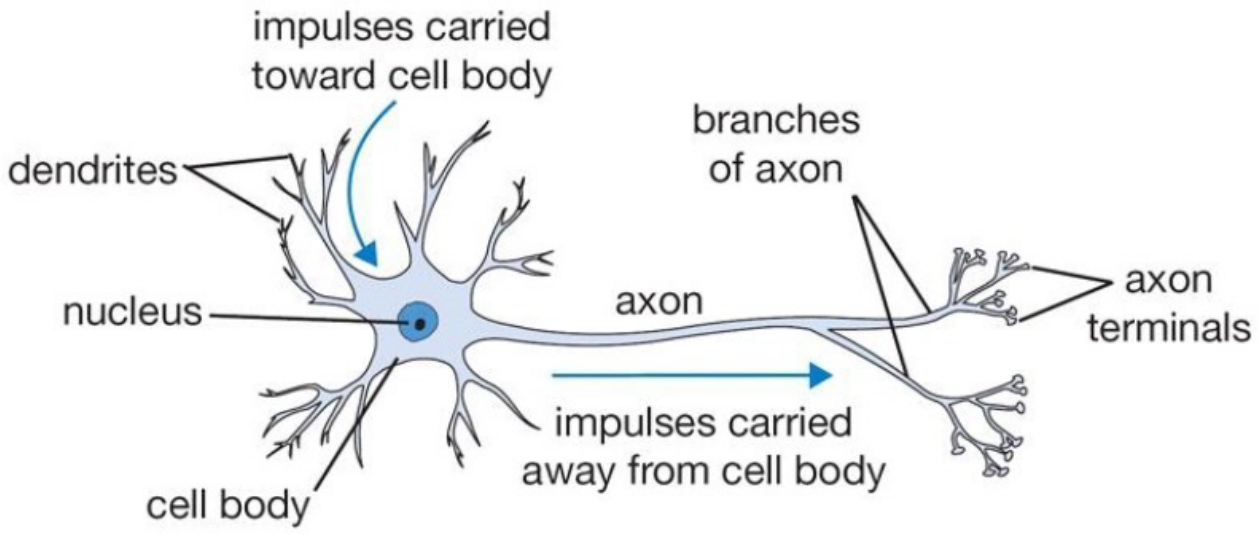
tensorflow.org / keras.io

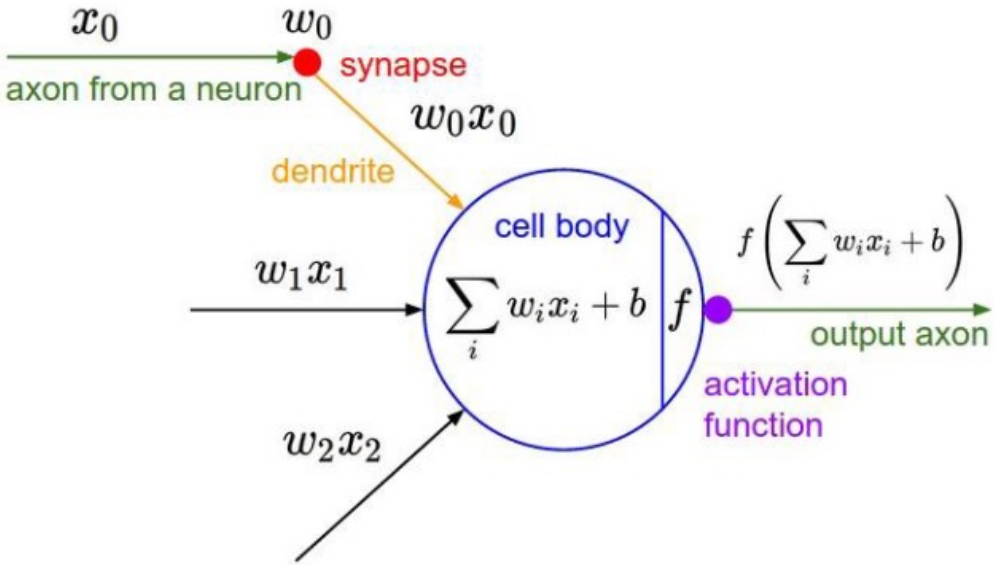
Outline

1. Lab2 task
2. Framework introduction
3. Layer introduction
4. Sample code introduction

3. Layers introduction

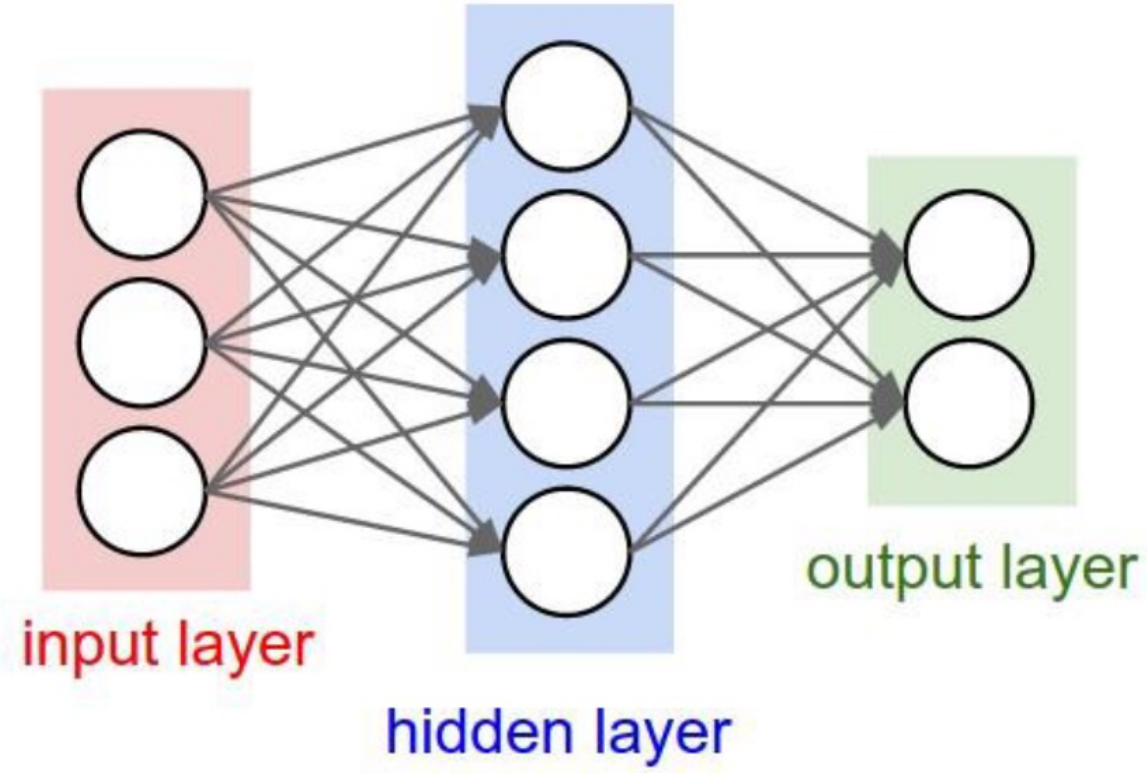
- Single neuron.





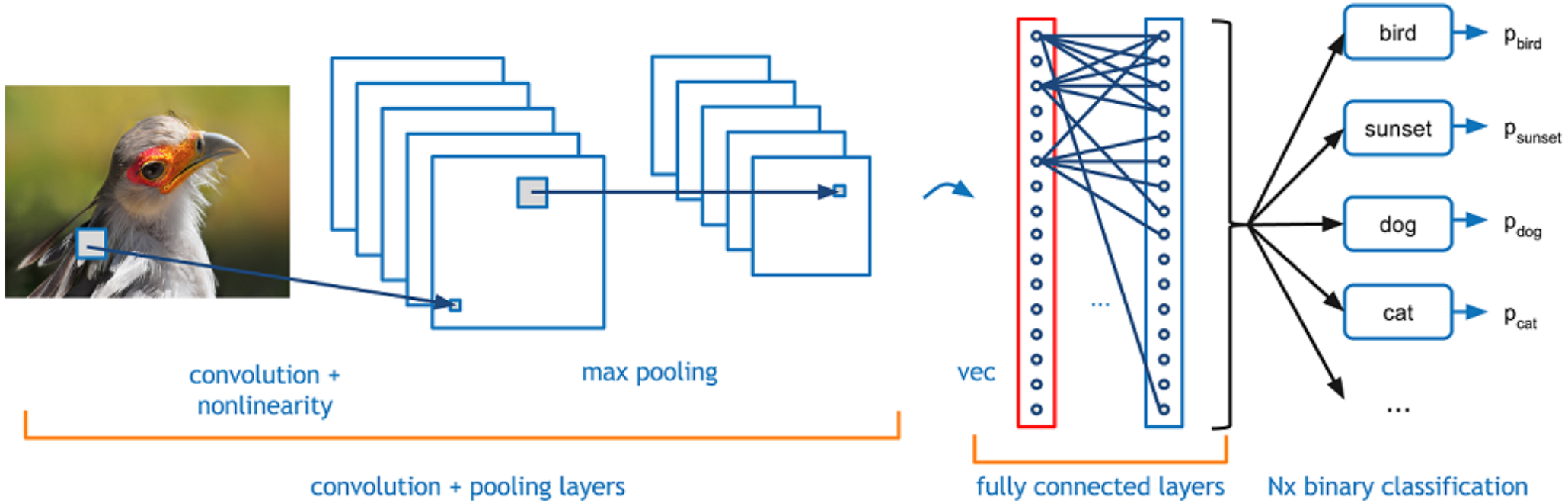
3. Layers introduction

- Multiple neurons.



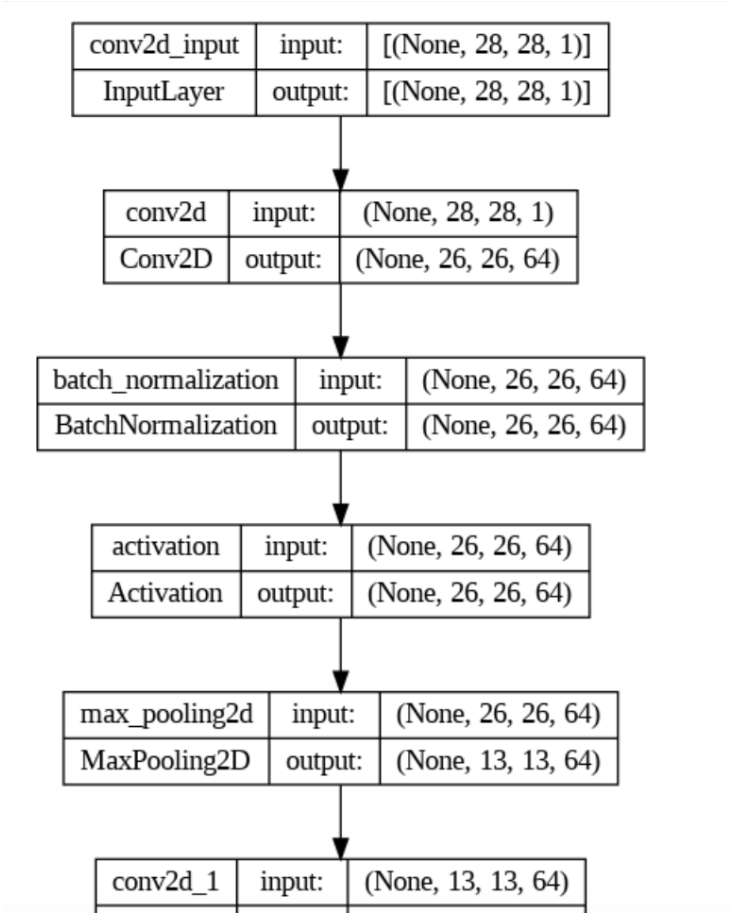
3. Layers introduction

- Mimicking multiple neurons by layers.



3. Layers introduction

- 3 convolutional neural layer model.



Conv1

Conv2

Conv3

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640

 batch_normalization (Batch  (None, 26, 26, 64)        256
 Normalization)

 activation (Activation)     (None, 26, 26, 64)        0

 max_pooling2d (MaxPooling2  (None, 13, 13, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        36928

 batch_normalization_1 (Bat  (None, 11, 11, 64)        256
 chNormalization)

 activation_1 (Activation)   (None, 11, 11, 64)        0

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 batch_normalization_2 (Bat  (None, 3, 3, 64)          256
 chNormalization)

 activation_2 (Activation)   (None, 3, 3, 64)          0

 max_pooling2d_2 (MaxPoolin  (None, 1, 1, 64)          0
 g2D)

 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 80074 (312.79 KB)
Trainable params: 79690 (311.29 KB)
Non-trainable params: 384 (1.50 KB)
```

3. Layers introduction – Activation function

```
model.add(Activation("relu"))
```

Activation Functions

1. step: $f(net) = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$

2. sigmoid: $t = \sum_{i=1}^{n} w_i x_i \qquad s(t) = 1/(1+e^{-t})$

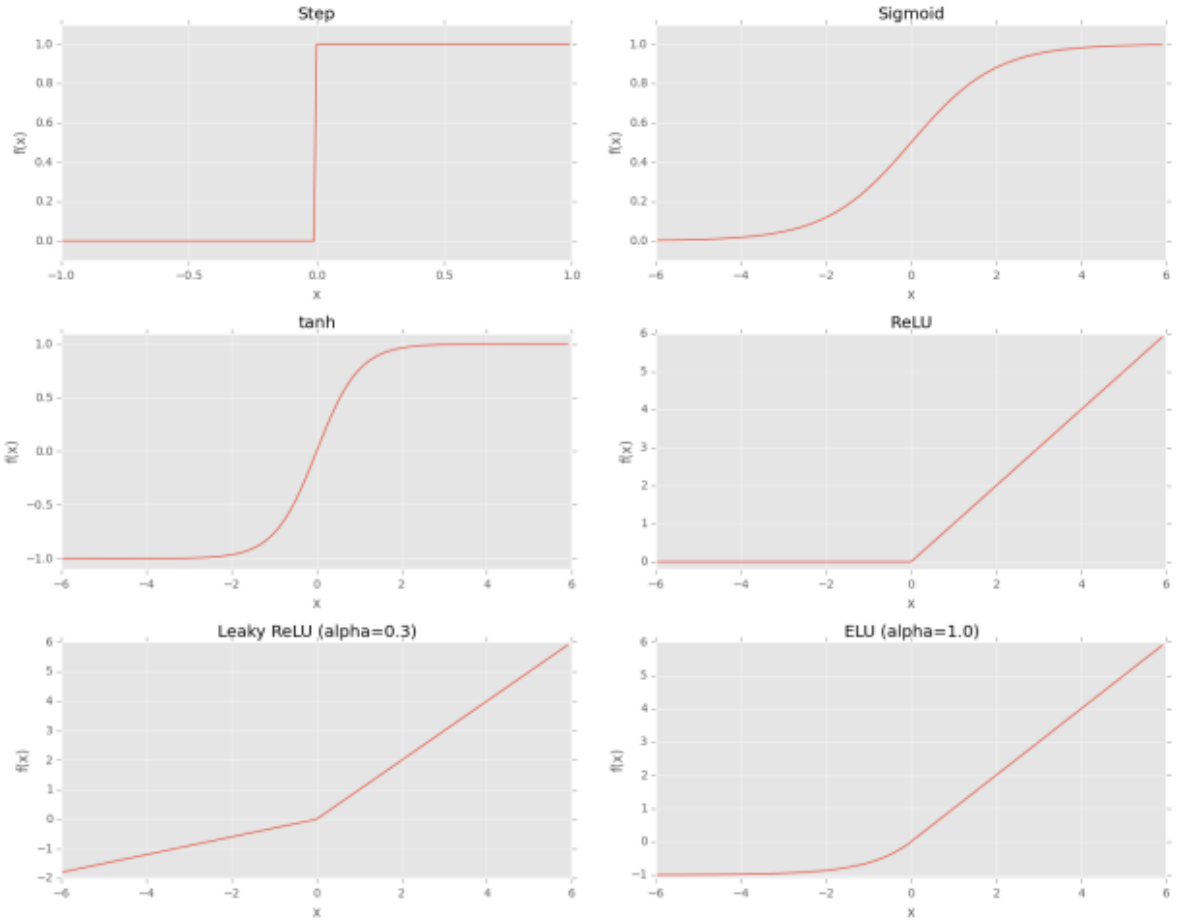
3. tanh: $f(z) = tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$

4. Rectified Linear Unit (ReLU) $f(x) = max(0, x)$

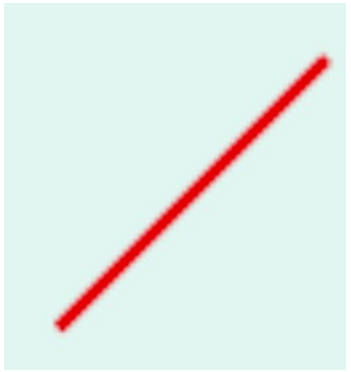
5. Leaky ReLUs $f(net) = \begin{cases} net & \text{if } net >= 0 \\ \alpha \times net & \text{otherwise} \end{cases}$

6. Exponential Linear Units (ELUs) $f(net) = \begin{cases} net & \text{if } net >= 0 \\ \alpha \times (exp(net) - 1) & \text{otherwise} \end{cases}$

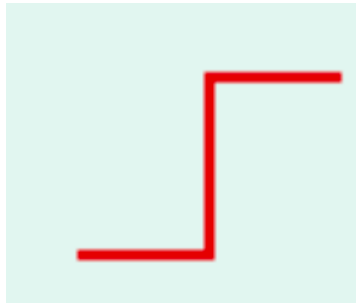
7. Softmax: $Softmax(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$



3. Layers introduction – Activation function

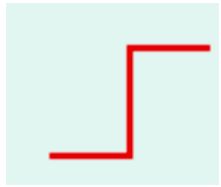


whole network linear and thus less useful



discrete and thus hard to optimize for weight

Easier to optimize, approximation of

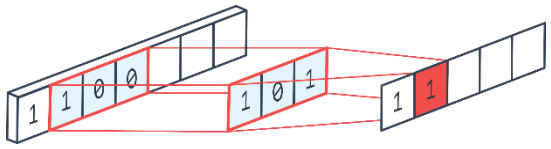


3. Layers introduction – Convolution layer

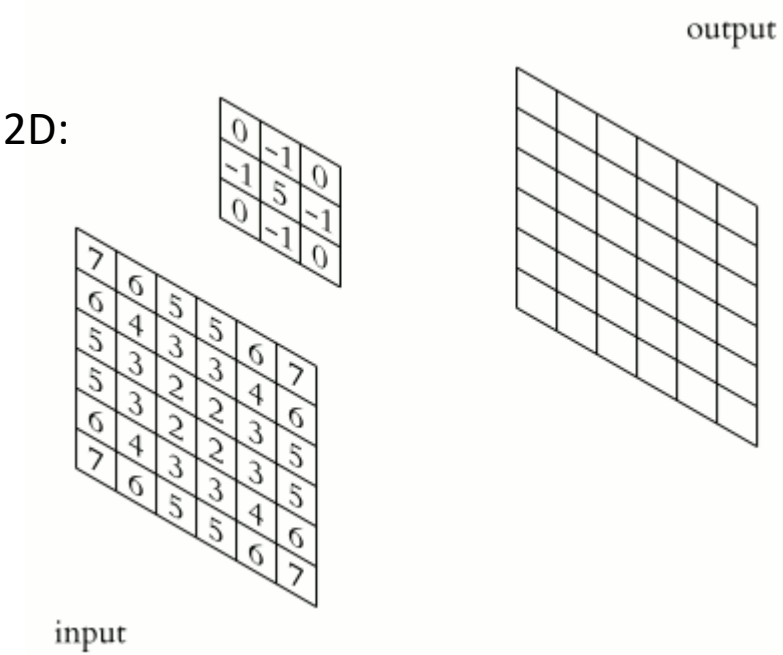
```
model.add(Conv2D(64, (3,3), strides=(1,1), padding="valid"))
```

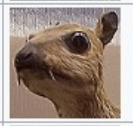
- 2D Convolution is using a 'kernel' to extract certain 'features' from an input image.

1D:



2D:



Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

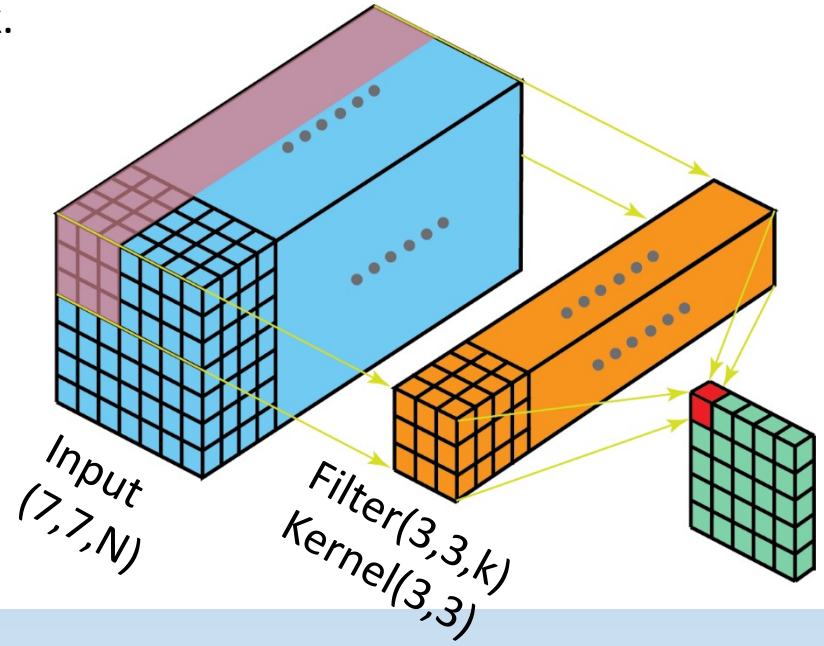
https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

3. Layers introduction – Convolution layer

- Kernel vs Filter

A kernel is, as described earlier, a matrix of weights which are multiplied with the input to extract relevant features. The dimensions of the kernel matrix is *how the convolution gets it's name*. For example, in 2D convolutions, the kernel matrix is a 2D matrix.

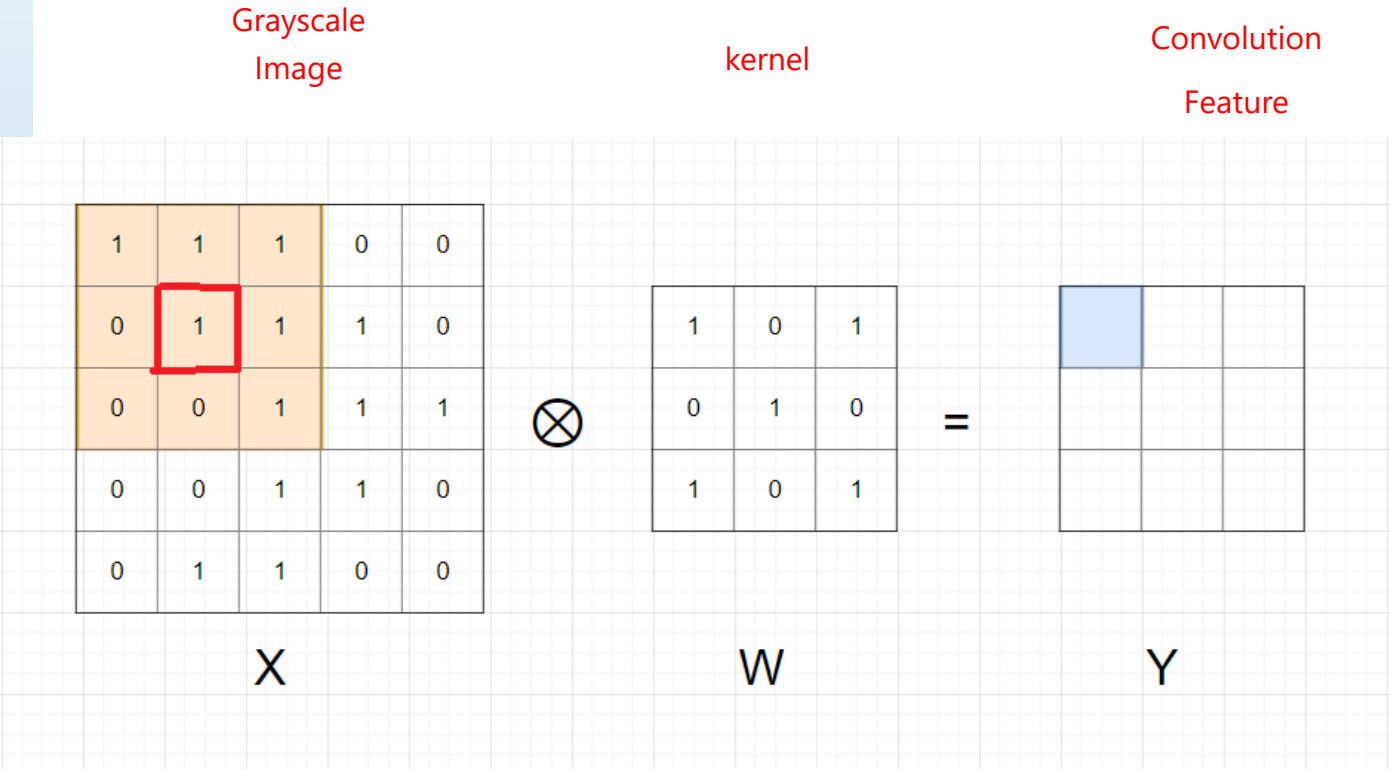
A filter however is a concatenation of multiple kernels, each kernel is assigned to a particular channel of the input. Filters are always one dimension more than the kernels. For example, in 2D convolutions, filters are 3D matrices (which is essentially a concatenation of 2D matrices i.e. the kernels). So for a CNN layer with kernel dimensions h*w and input channels k, the filter dimensions are h*w*k.



https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

3. Layers introduction – Convolution layer

- E.g. Convolution on grayscale image



X: Image sized m x n

W: kernel sized k x k

➔ Results Y: Image sized (m-k+1) x (n-k+1) ➔ reduce the X size

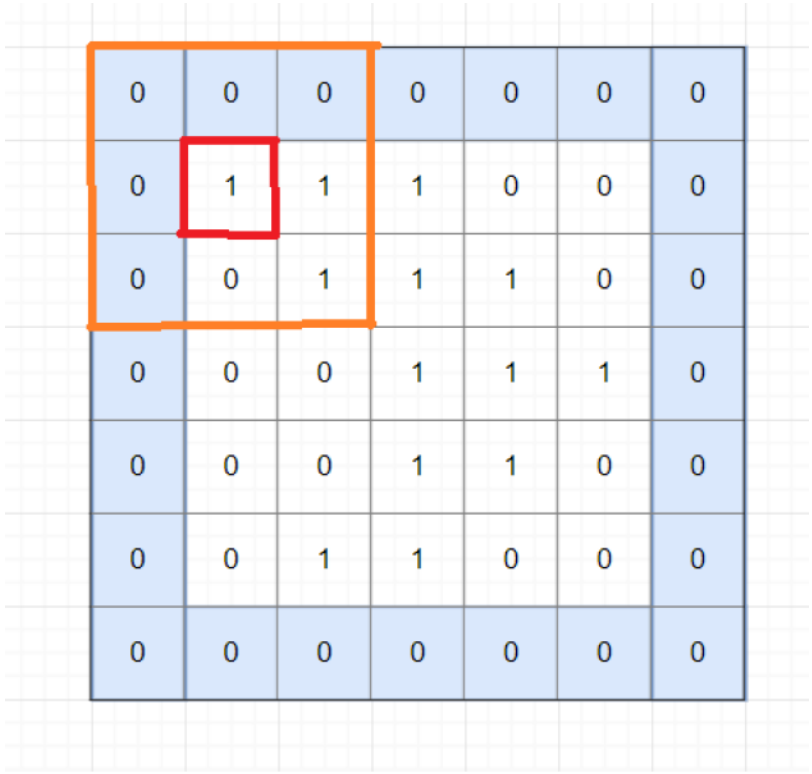
$$Y = X \otimes W$$

tf.keras.layers.Conv2D padding introduction:
https://keras.io/api/layers/convolution_layers/convolution2d/

3. Layers introduction – Convolution layer

- E.g. Convolution on grayscale image

Convolution/Padding



X: m x n

W: k x k

➔ Y: same size with X

Padding = p = 1

https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7

tf.keras.layers.Conv2D padding definition:

- **padding**: one of `"valid"` or `"same"` (case-insensitive). `"valid"` means no padding. `"same"` results in padding with zeros evenly to the left/right or up/down of the input. When `padding="same"` and `strides=1`, the output has the same size as the input.

3. Layers introduction – Convolution layer

- E.g. Convolution on grayscale image

Convolution/Stride = s, only compute convolution on pixels $x_{1+i*s,\ 1+j*s}$

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Padding = p = 1, Stride = s =1

X: m x n

W: k x k

➔ Y: same size with X

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Padding = p = 1, Stride = s =2

X: m x n

W: k x k

➔Y: $\left(\frac{m-k+2p}{s}+1\right)\times\left(\frac{n-k+2p}{s}+1\right)$

e.g.

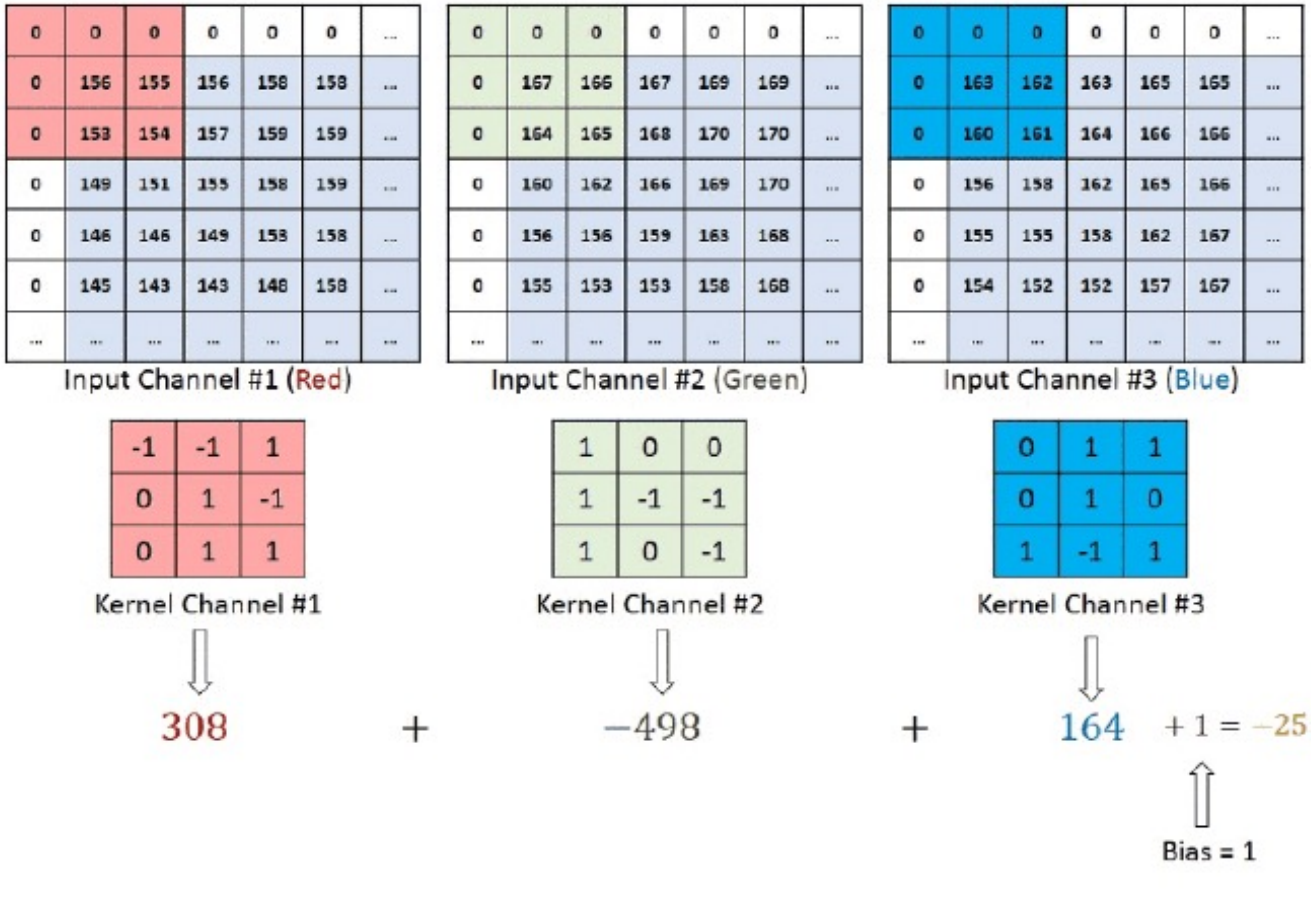
X: m x n, m = n = 5

W: k x k, k = 3

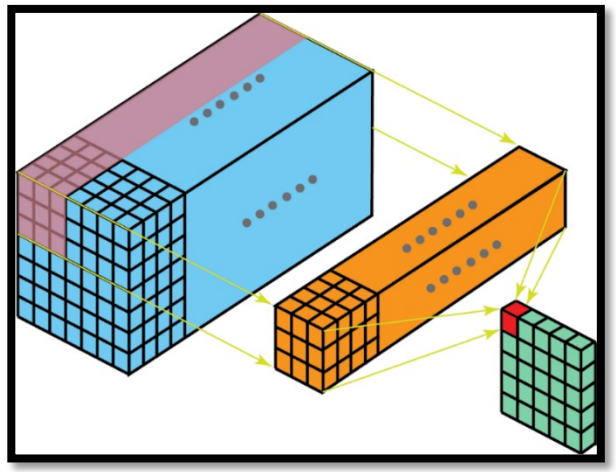
$\left(\frac{5-3+2*1}{2}+1\right)\times\left(\frac{5-3+2*1}{2}+1\right)=(3,3)$

3. Layers introduction – Convolution layer

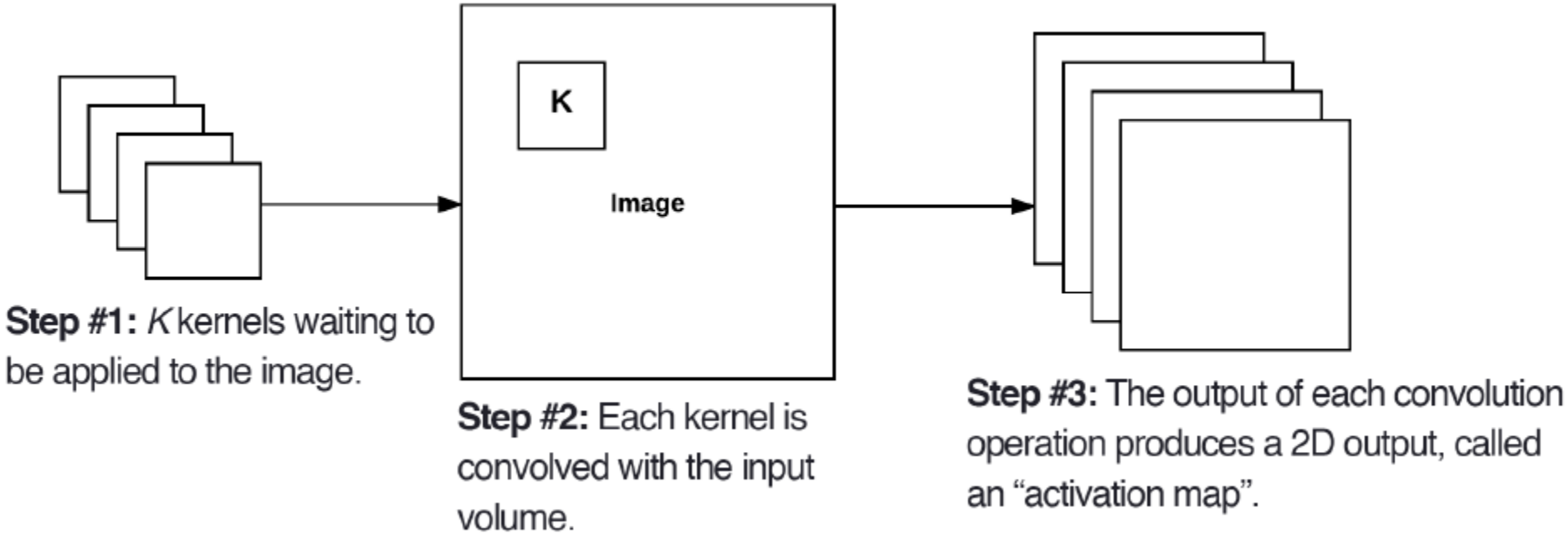
- E.g. Convolution on RGB image



e.g.
Input X: NxN
Kernel W: 3x3
Filters(channels =3) :3 x 3 x 3



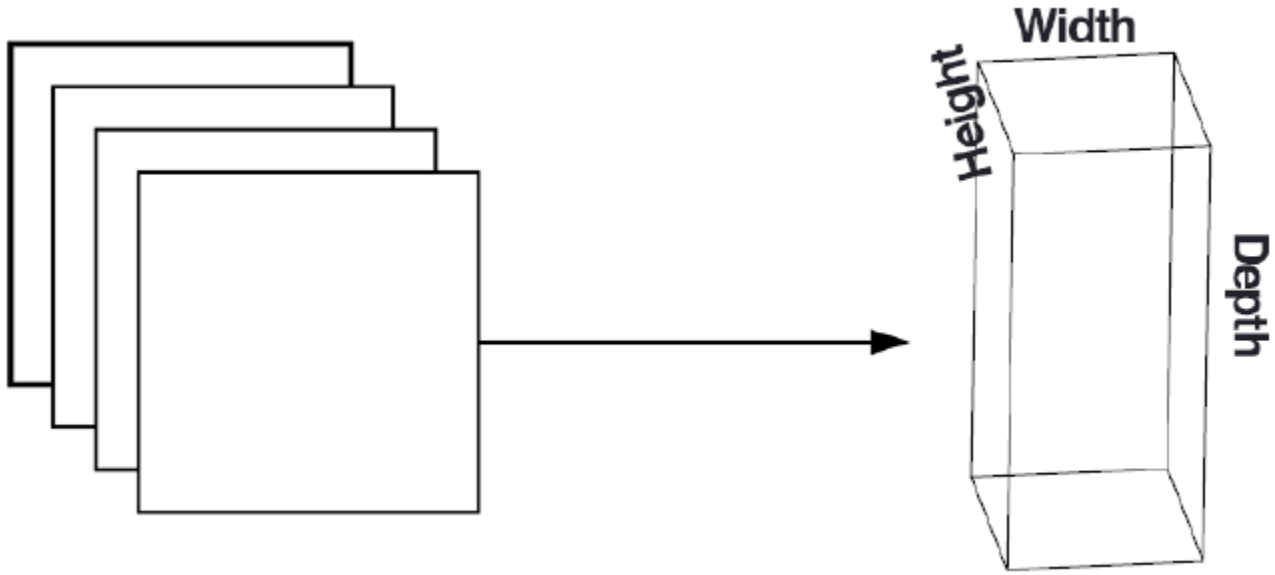
3. Layers introduction – Convolution layer



Left: At each convolutional layer in a CNN, there are K kernels applied to the input volume.
Middle: Each of the K kernels is convolved with the input volume.
Right: Each kernel produces an 2D output, called an activation map(feature map).

https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

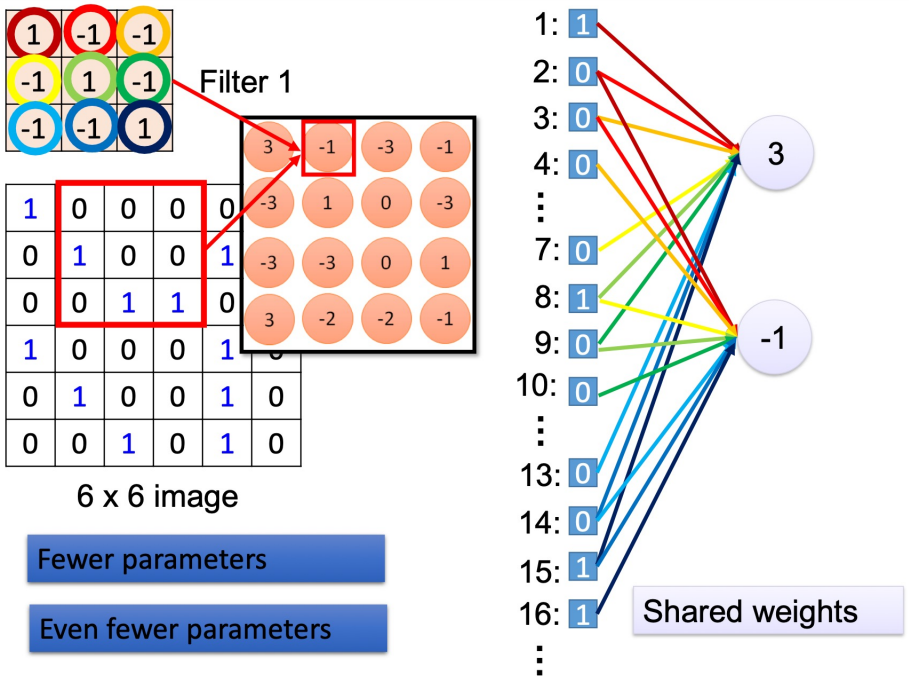
3. Layers introduction – Convolution layer

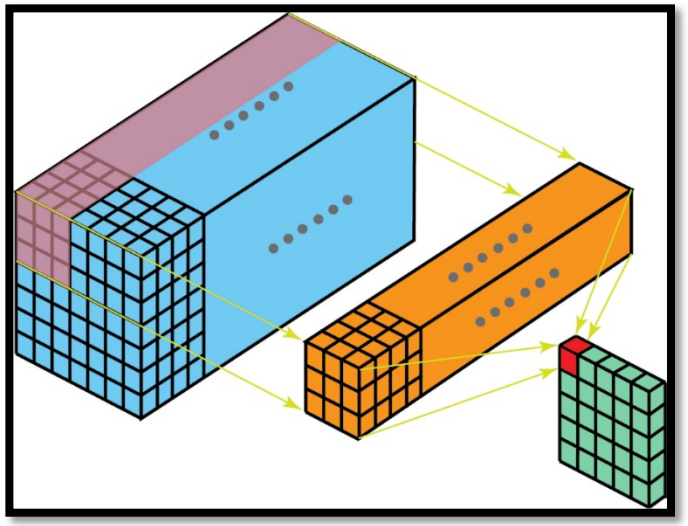


After obtaining the K activation maps(feature maps), they are stacked together to form the input volume to the next layer in the network.

3. Layers introduction – Convolution layer

- Why do we use CNN instead of NN?
  1. reduce the number of parameters(shared weights)
  2. accelerate execution time
  3. combine image features
  4. No longer need experts to design convolutional kernels





3. Layers introduction – Convolution layer

- Calculation of parameters, take LeNet for example:

Name	LeNet (first conv layer)
Input Image Size	28x28x1
Number of Filters (channels)	6
Filter size (kernel)	5x5x1
Stride	1
Padding	0
Number of parameters for a unit (without parameter sharing)	89,856
Number of parameters for a unit (with parameter sharing)	156
Output Volume	24x24x6

1. Output width of conv layer:

   = *((28–5+2*0) / 1) +1*

   = *24 (conv layer output width)*

2. Number of neurons/units within the conv layer

   = *output height * output width * number of feature maps(channels)*

   = *24x24x6 (conv output volume)*

   = *3,456 units*

3. Number of training parameters or weights within the conv layer (without weight sharing)

   = *3456* ((5 * 5 * 1) + 1 bias)*

   = *89,856 weights*

4. Number of training parameters or weights with weight sharing (with weight sharing)

   = *6*((5*5*1) + 1 bias)*

   = 156 weights

https://towardsdatascience.com/understanding-parameter-sharing-or-weights-replication-within-convolutional-neural-networks-cc26db7b645a

3. Layers introduction – Convolution layer

- Calculation of parameters, take AlexNet for example:

Name	AlexNet (first conv layer)
Input Image Size	227x227x3
Number of Filters channels	96
Filter size kernel	11x11x3
Stride	4
Padding	0
Number of parameters for a unit (without parameter sharing)	105,705,600
Number of parameters for a unit (with parameter sharing)	34,944
Output Volume	55x55x96

1. Output width of conv layer:

   = *((227–11) / 4) +1*

   = *55 (conv layer output width)*

2. Number of neurons/units within the conv layer

   = *output height * output width * number of feature maps*

   = *55x55x96 (conv output volume)*

   = *290,400 units*

3. Number of training parameters or weights within the conv layer (without weight sharing)

   = *290400 * ((11 * 11 * 3) + 1 bias)*

   = *105,415,600 weights*

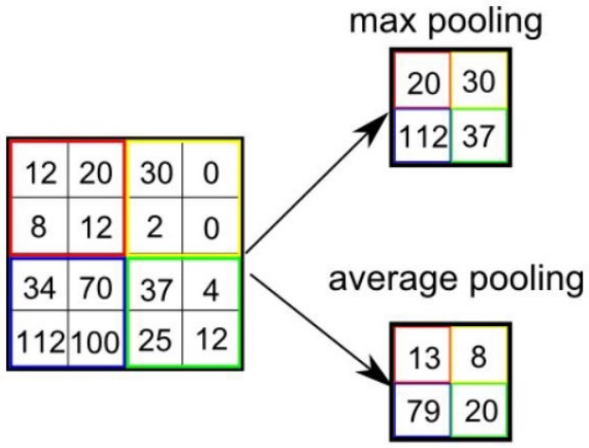
4. Number of training parameters or weights with weight sharing (with weight sharing)

   = *96*((11*11*3) + 1 bias)*

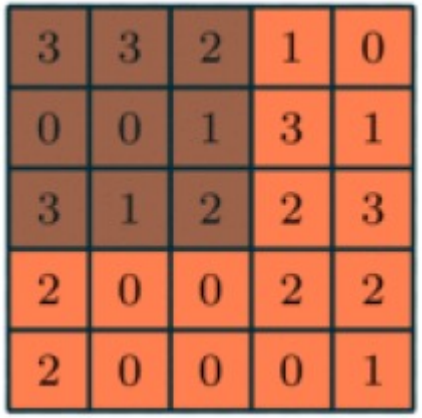
   = 34,944 weights

3. Layers introduction – Pooling layer

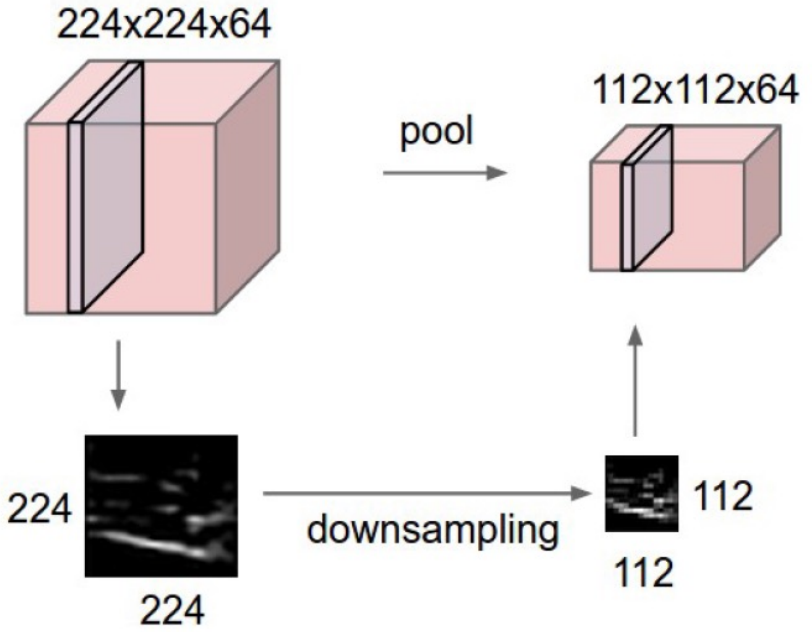
- To reduce model dimensionality by sliding window approach.
  - -Max pooling: Discard unnecessary or noisy features.
  - -Average pooling



Pooling layer size (2,2), stride = 2



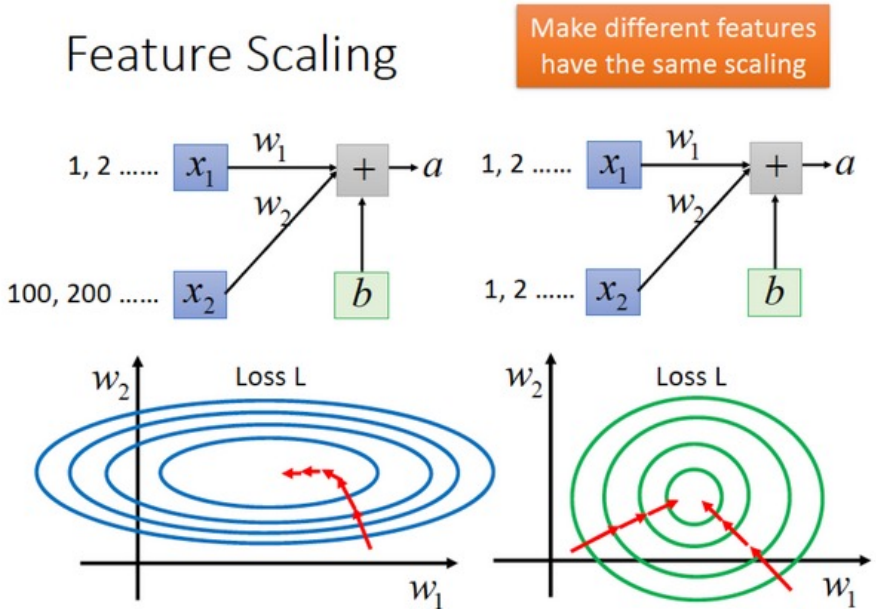
max pooling layer with size=(3,3), stride=1, padding=0

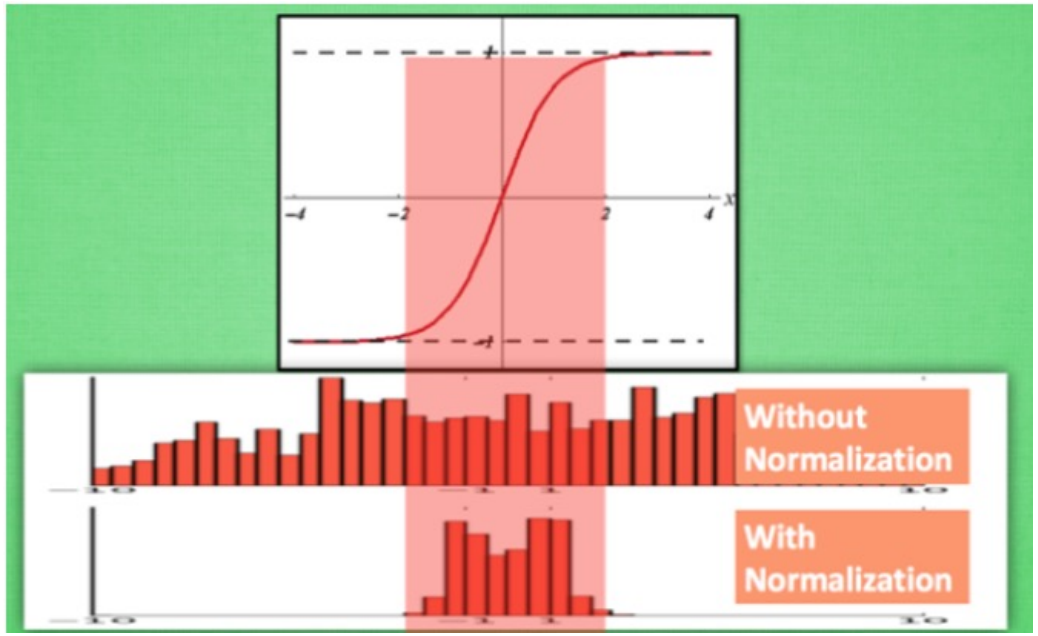


after pooling layer with size=(2,2), stride=2, padding=0

3. Layers introduction – Batch normalization layer

- Efficiently normalize based on loaded batch data.
- Train faster with bigger learning rate by avoiding covariant problem.
- Avoiding vanishing gradients especially effective for sigmoid, tanh.

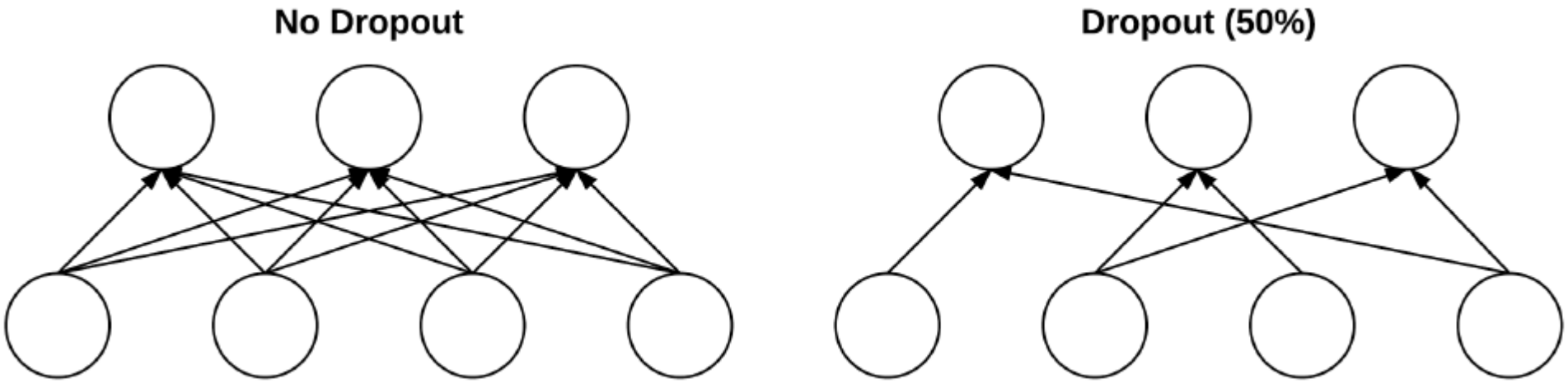




NTU Professor Lee' s ML: https://www.youtube.com/watch?v=BZh1ltr5Rkg

3. Layers introduction – Dropout layer

- To generalize the model by randomly dropping multiple redundant nodes.
- Usually placed between fully-connected layers.

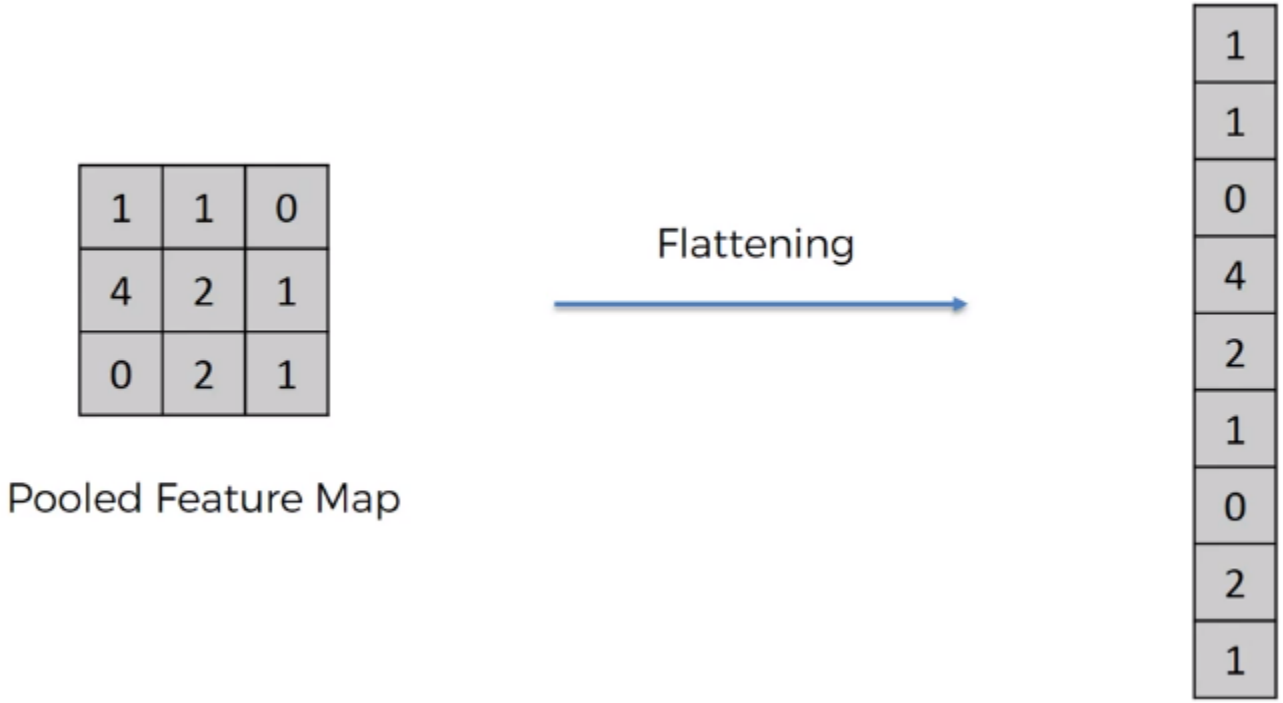


Left: Two layers of a neural network that are fully-connected with no dropout. Right: The same two layers after dropping 50% of the connections.

Connections: 12 p = 0.5 Connections: 6

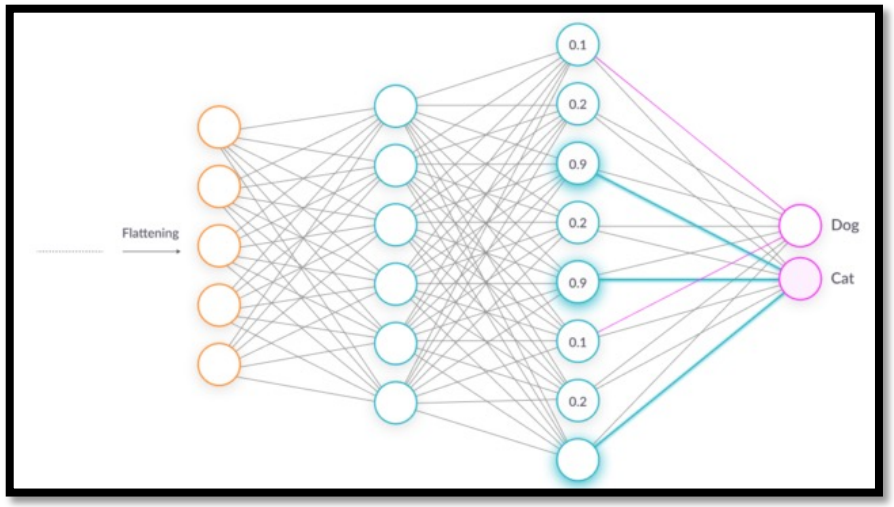
3. Layers introduction – Flatten layer

- Flatten layer is used to make the multidimensional input one-dimensional, commonly used in the transition from the convolution layer to the full connected layer.



3. Layers introduction – Dense layer

- Fully connected layer.
- Each neuron in the dense layer receives input from all neurons of its previous layer, where neurons of the dense layer perform matrix-vector multiplication



https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

`Dense` implements the operation: `output = activation(dot(input, kernel) + bias)`

Outline

1. Lab2 task
2. Framework introduction
3. Layers introduction
4. Sample code introduction

4. Sample code introduction

Lab2 task

Use pytorch/Tensorflow to implement an specific classification DNN model, dataset Fashion MNIST

1. Design model by keras.sequential model, 3 layer of CNN network (10%), 3 layer of NN network (10%)
2. Comparison w/ and w/o Batch Normalizatoin Layer (10%)
3. Comparison w/ arbitrary layer of abovementioned CNN network. (10%)
4. Print model summary and plot model (10%)
5. Print test accuracy, plot train-epoch, val-epoch, train-loss, val-loss (20%)
6. Plot certain image from dataset and successively predict (10%)
7. Report(30%)

Q&A: course.aislab@gmail.com

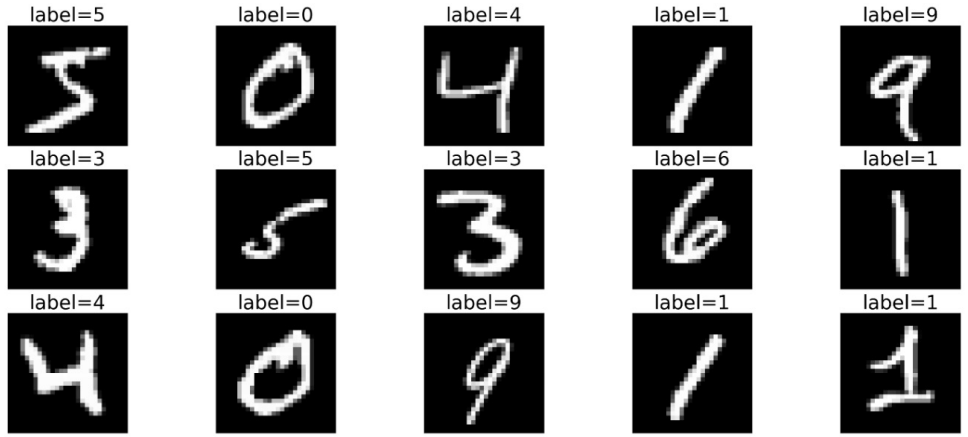
Import from framework

```
[1]  !pip install utils
     import tensorflow as tf
     import pickle
     import matplotlib
     import matplotlib.pyplot as plt
     import numpy as np

     from tensorflow.keras.datasets.mnist import load_data
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization
     from tensorflow.keras.utils import plot_model
     from sklearn.preprocessing import LabelBinarizer
     from sklearn.metrics import classification_report
     from sklearn import metrics
```

4. Sample code introduction – MNIST dataset

```
# Load MNIST dataset
# Keras images are 28x28 array rather than a 1D array of size 784
# Pixels intensities are integers (0 to 255) rather than floats (0.0 to 255.0)

(train_X, train_y), (test_X, test_y) = load_data()

train_X = train_X.reshape(-1, 28, 28, 1) # "-1"menas auto calculate
test_X = test_X.reshape(-1, 28, 28, 1)

train_X = train_X.astype('float32') / 255.0 # Normalization
test_X = test_X.astype('float32') / 255.0

train_y = train_y.reshape(-1,1)
test_y  = test_y.reshape(-1,1)

# Dataset already split in Training and Testing
# Train_X for images and train_y for labels
#shape of train_X should be (6000, 28, 28, 1), which is (class, height, width, channel)
#shape of train_y should be (6000, 1), which is (class, channel(only needs one to store class info))
```

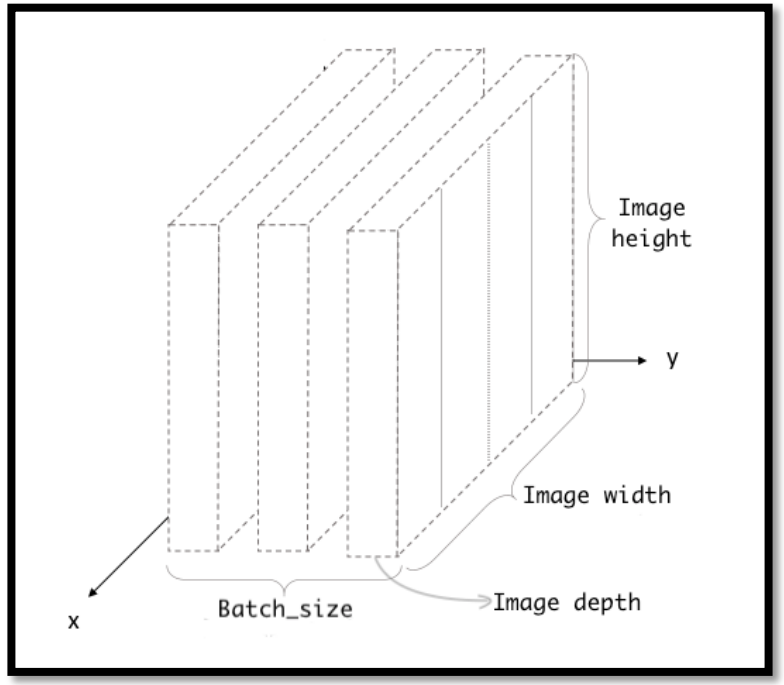
```
#show some image from MNIST dataset
imageMNIST = plt.figure(figsize=(75,15))
for i in range (30):
  ax = imageMNIST.add_subplot(3,10, i+1, xticks=[], yticks=[]) # remove xticks and yticks
  ax.imshow(np.squeeze(train_X[i]), cmap='gray')
  ax.set_title('label=' + str(train_y[i][0]), fontsize=40)
```

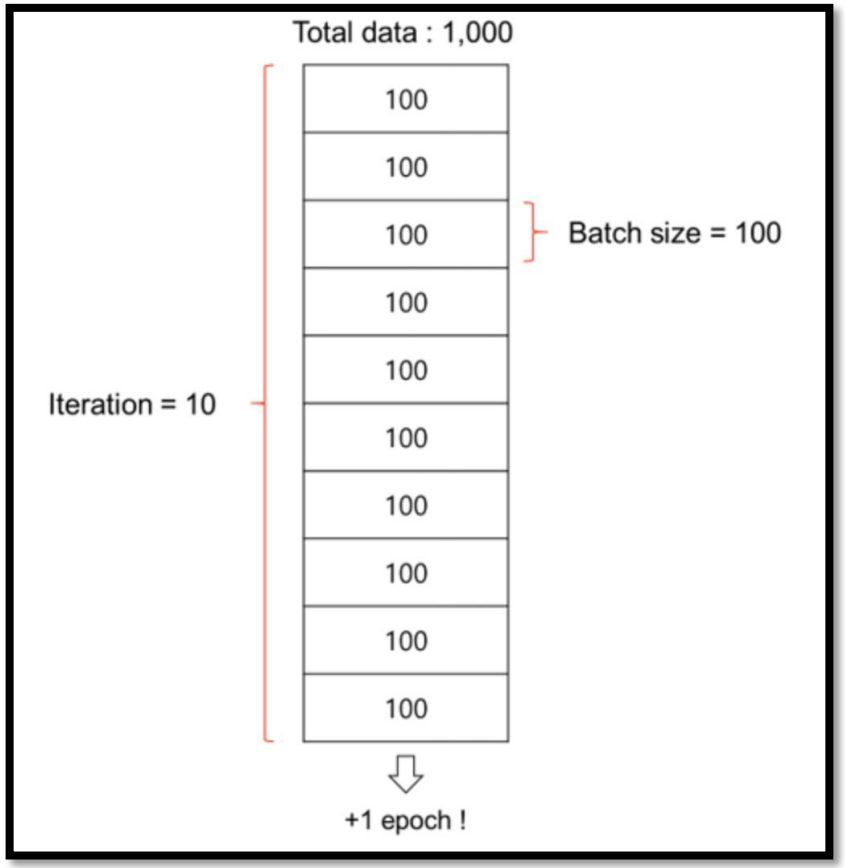


- Train: 60000 images
- Test: 10000 images

4. Sample code introduction

```
#Settings:
#inputShape = (height,width,depth)
InputShape =
classes =
num_epochs =
num_batch_size =
num_validation_split =

#show input shape
print(InputShape)
```





tensorflow.org / keras.io

4. Sample code introduction

Create your own model

```
#model.add to add layer
#model.add(layer_name(number_of_filters, (window_size,window_size), input_shape))

#2 way of adding activation function
#(1)model.add(Activation("relu"))
#(2)model.add(Dense(1), activation="relu")

model = Sequential()
model.add...
```

```
#print model
```

```
[ ] #plot model
```

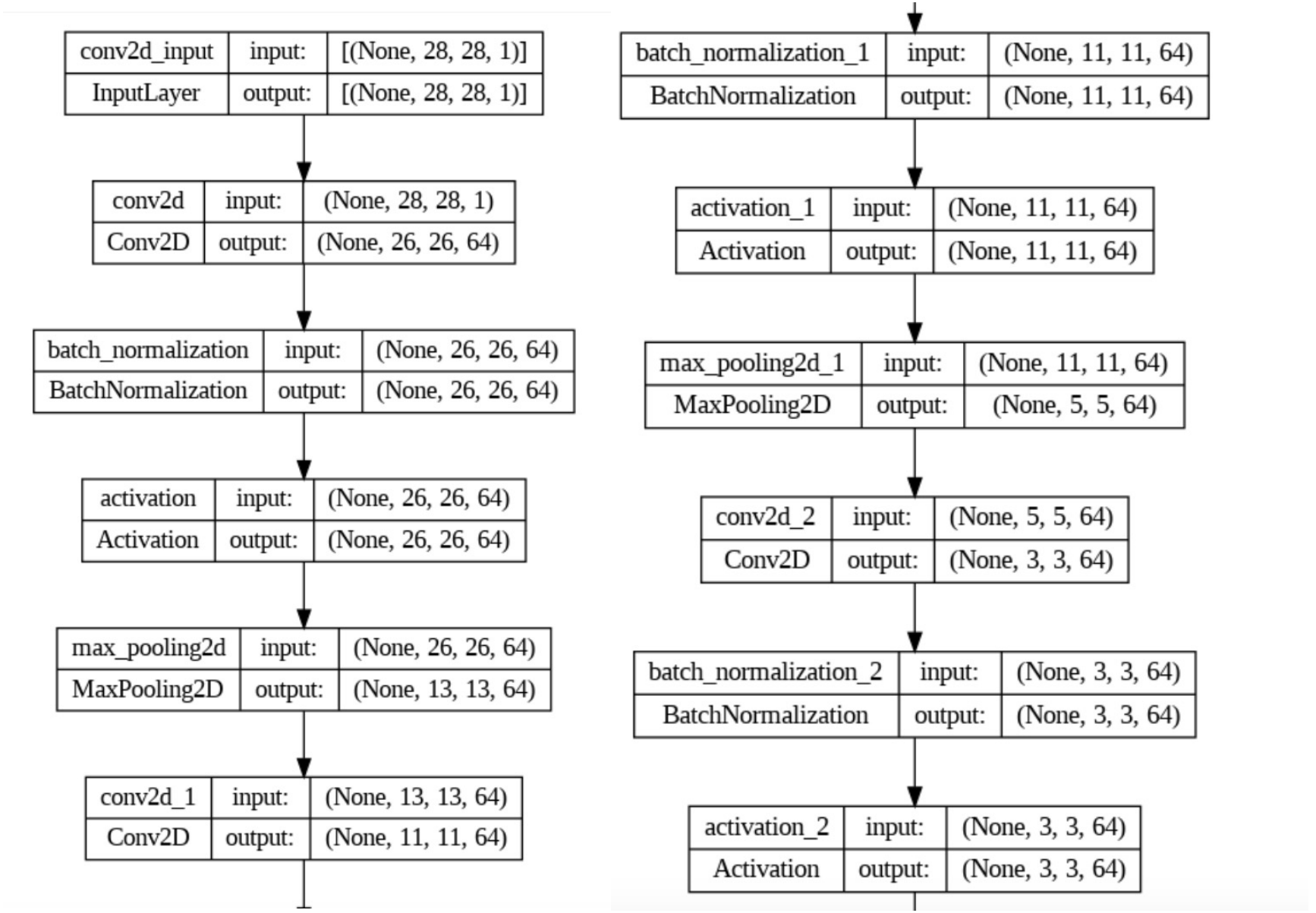
```
[ ] #model fit to train model
```

```
[ ] #evalutate the model
predictions = model.predict(test_X, batch_size=num_batch_size)

#plot and print
```

```
max_pooling2d (MaxPooling2  (None, 13, 13, 64)        0
D)

conv2d_1 (Conv2D)           (None, 11, 11, 64)        36928

batch_normalization_1 (Bat  (None, 11, 11, 64)        256
chNormalization)

activation_1 (Activation)   (None, 11, 11, 64)        0

max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
g2D)

conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

batch_normalization_2 (Bat  (None, 3, 3, 64)          256
chNormalization)

activation_2 (Activation)   (None, 3, 3, 64)          0

max_pooling2d_2 (MaxPoolin  (None, 1, 1, 64)          0
g2D)

flatten (Flatten)           (None, 64)                0

dense (Dense)               (None, 64)                4160

dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 80074 (312.79 KB)
Trainable params: 79690 (311.29 KB)
Non-trainable params: 384 (1.50 KB)
```

4. Sample code introduction

- Plot model



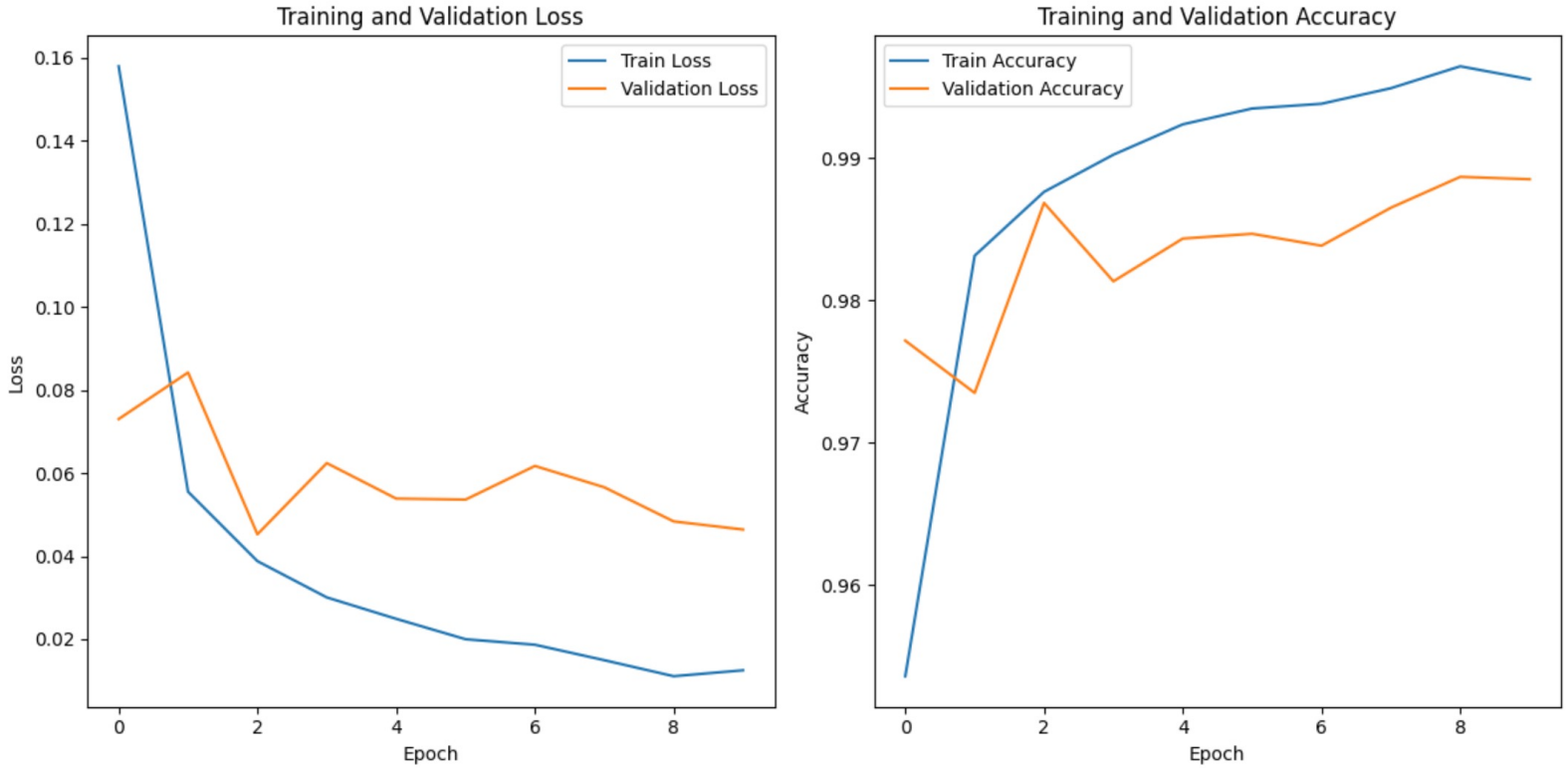
4. Sample code introduction

- Print training process.

```
Epoch 1/10
844/844 [==============================] - 22s 7ms/step - loss: 0.1580 - accuracy: 0.9536 - val_loss: 0.0730 - val_accuracy: 0.9772
Epoch 2/10
844/844 [==============================] - 5s 6ms/step - loss: 0.0555 - accuracy: 0.9831 - val_loss: 0.0842 - val_accuracy: 0.9735
Epoch 3/10
844/844 [==============================] - 6s 7ms/step - loss: 0.0388 - accuracy: 0.9876 - val_loss: 0.0453 - val_accuracy: 0.9868
Epoch 4/10
844/844 [==============================] - 5s 6ms/step - loss: 0.0301 - accuracy: 0.9902 - val_loss: 0.0624 - val_accuracy: 0.9813
Epoch 5/10
844/844 [==============================] - 5s 6ms/step - loss: 0.0249 - accuracy: 0.9924 - val_loss: 0.0539 - val_accuracy: 0.9843
Epoch 6/10
844/844 [==============================] - 6s 7ms/step - loss: 0.0200 - accuracy: 0.9935 - val_loss: 0.0536 - val_accuracy: 0.9847
Epoch 7/10
844/844 [==============================] - 5s 6ms/step - loss: 0.0187 - accuracy: 0.9938 - val_loss: 0.0617 - val_accuracy: 0.9838
Epoch 8/10
844/844 [==============================] - 6s 7ms/step - loss: 0.0149 - accuracy: 0.9949 - val_loss: 0.0566 - val_accuracy: 0.9865
Epoch 9/10
844/844 [==============================] - 5s 6ms/step - loss: 0.0111 - accuracy: 0.9964 - val_loss: 0.0484 - val_accuracy: 0.9887
Epoch 10/10
844/844 [==============================] - 6s 7ms/step - loss: 0.0125 - accuracy: 0.9955 - val_loss: 0.0464 - val_accuracy: 0.9885
```

4. Sample code introduction

- Print and plot result.

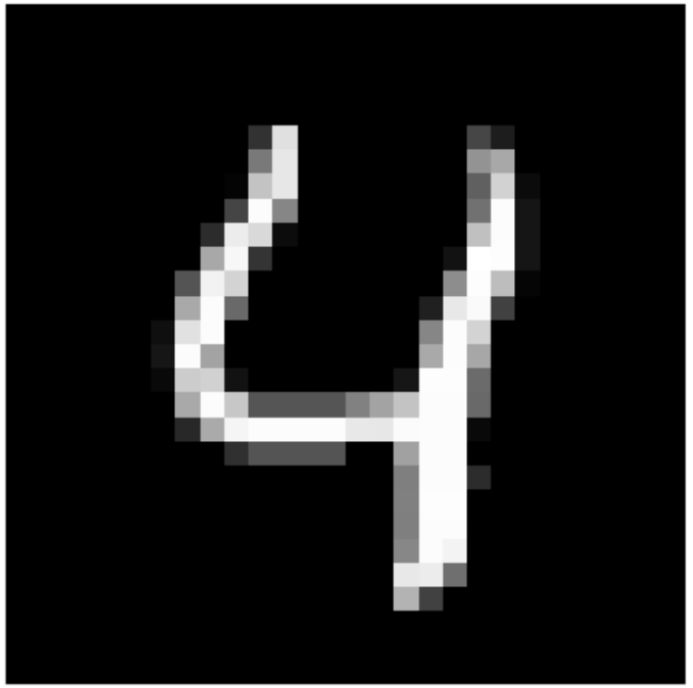



```
test accuracy: 0.9866
```

4. Sample code introduction

- Predict certain image from test dataset

```
#predict certian image from test dataset
#show image
image1 = test_X[4] # shape = (28, 28, 1)
plt.imshow(np.squeeze(image1), cmap='gray')
plt.axis('off')
plt.show()
```



4. Sample code intro - Lab2 task Recall

- Recommend platform: colab with python

- Upload platform: NCKU moodle
- Upload compressed .zip file including:
  - 學號_lab2.ipynb
    - IPython notenook 須包含程式碼跟結果
  - 學號_lab2.pdf
    - 各項print以及plot輸出
    - 實作所運到的困難及解決方法
- Provided file: lab2 sample code
- Q&A: course.aislab@gmail.com

Thank you for listening