

# SC310005 Artificial Intelligence

## Lecture 11: Deep Learning (Part 2)

[teerapong.pa@chula.ac.th](mailto:teerapong.pa@chula.ac.th)

# Reference:

1. [https://pytorch.org/tutorials/beginner/basics/quickstart\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html)
2. <https://www.codingninjas.com/studio/library/lenet-5>
3. <https://livebook.manning.com/book/grokking-machine-learning/chapter-4/>
4. <https://www.labmedico.com/?m=deep-learning-activation-functions-using-dance-moves-r-learnmachinelearning-gg-D35gqNHR>

# Architecture of LeNet5

LeNet5 is a network made up of 7 layers. It consists of 3 convolution layers, two subsampling layers, and two fully connected layers.

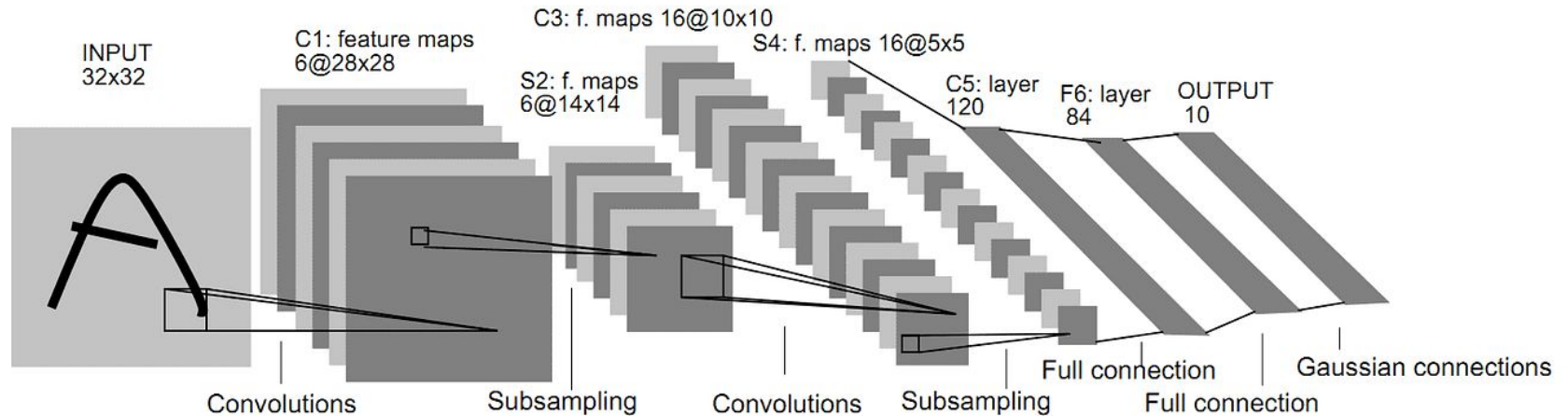
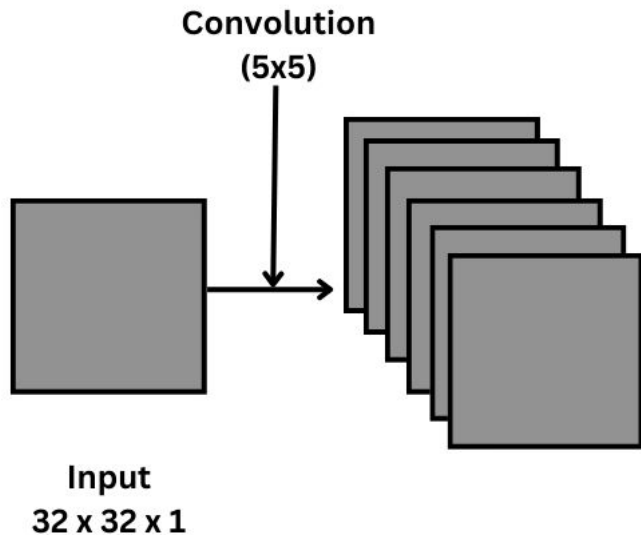


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

# LeNet5: First layer

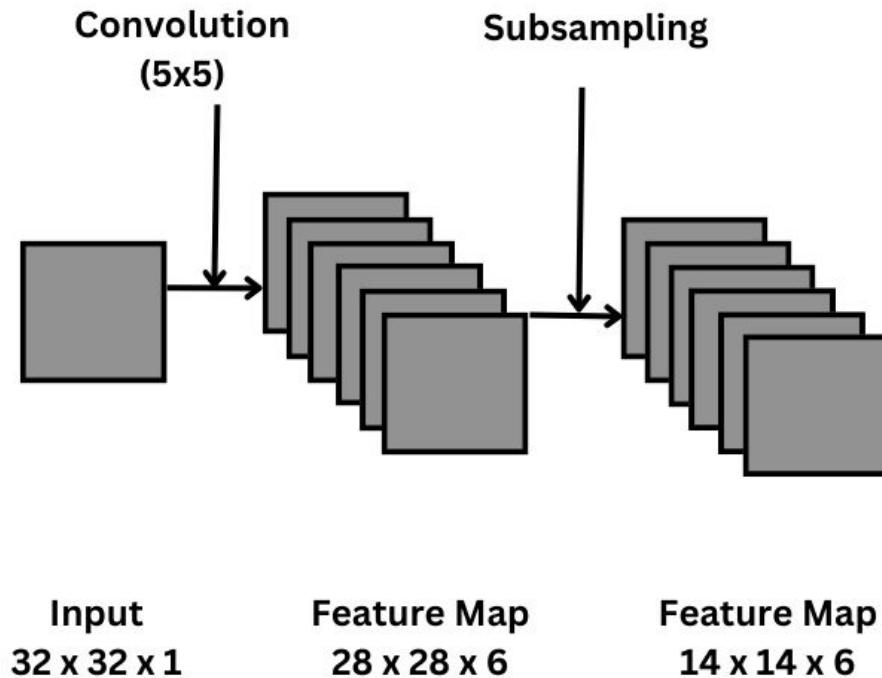
After this, the first convolution step takes place, and the input image is convoluted to the size of 28x28.



$$\begin{aligned}\text{Output Shape} &= ((32 - 5 + 1) \times (32 - 5 + 1) \times 6) \\ &= (28 \times 28 \times 6)\end{aligned}$$

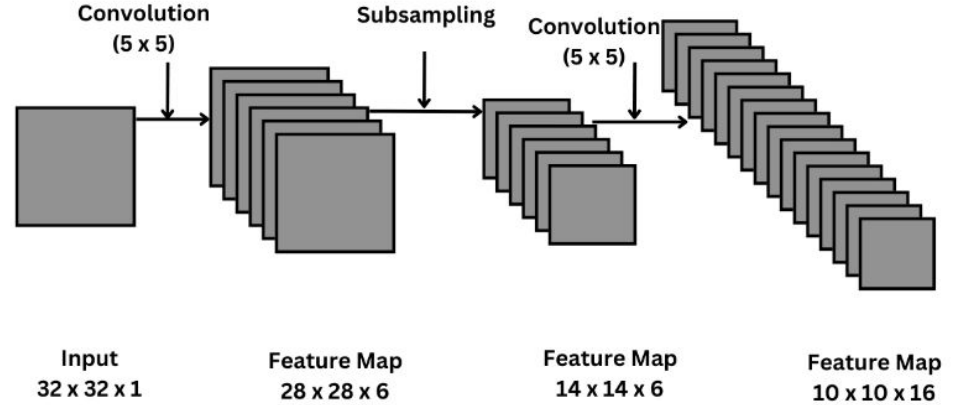
# LeNet5: Second layer

Next is the subsampling layer, in which the size is reduced to half, i.e., 14x14.



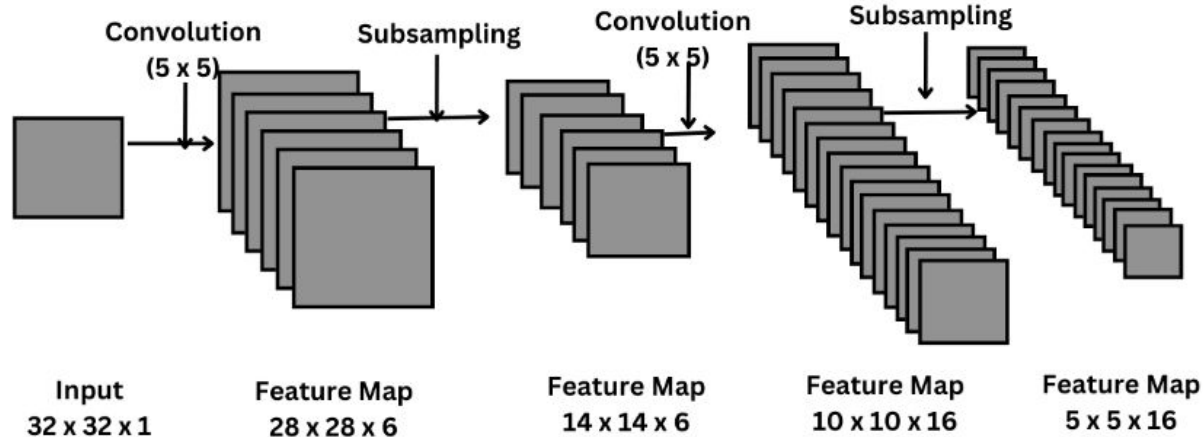
# LeNet5: Third layer

In the third layer, convolution occurs again, but this time with 16 filters of 5x5 size. After this layer, the size of the input image is reduced to 10x10x16.



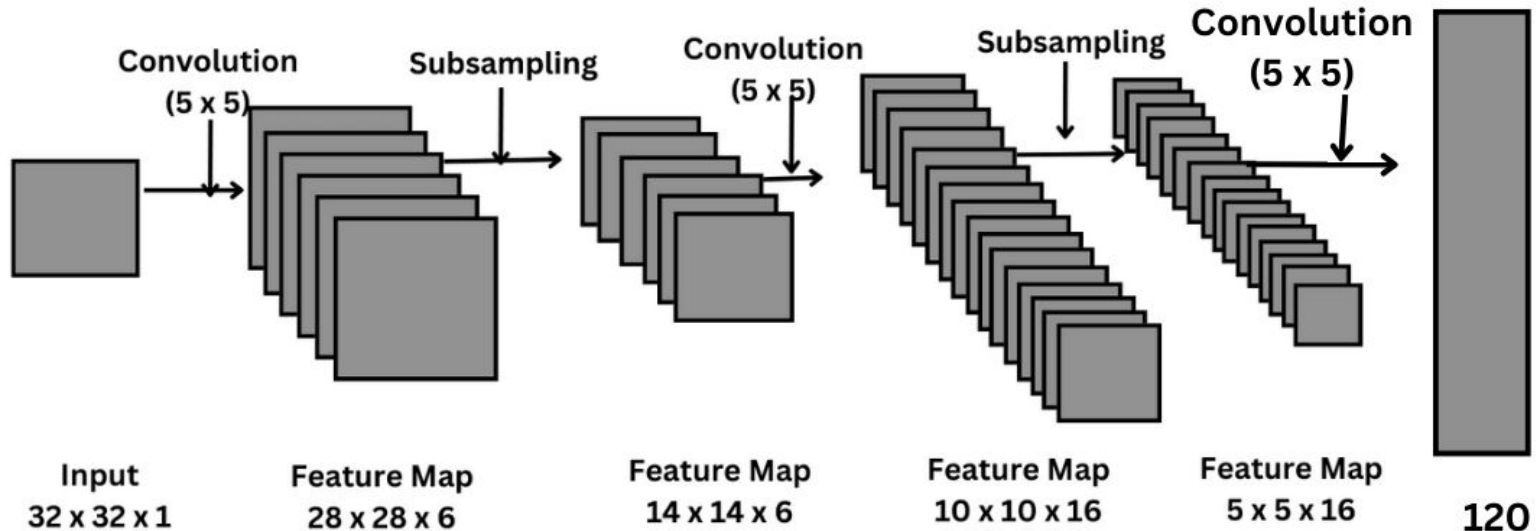
# LeNet5: Fourth layer

The subsampling takes place, and the image size in this step is reduced to  $5 \times 5 \times 16$ . In this layer, the input for the very last function diagram comes from all the remaining function diagrams.

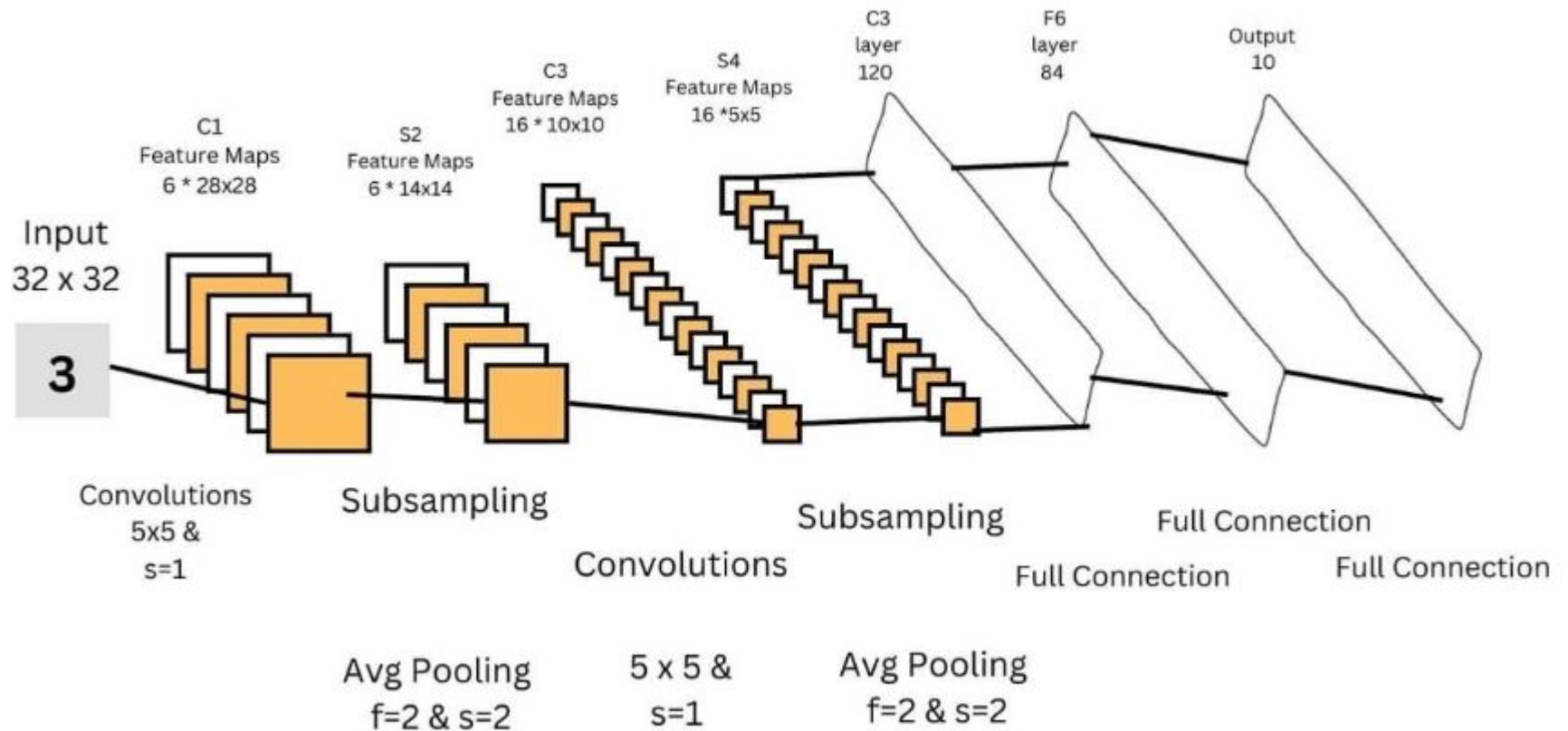


# LeNet5: Fifth layer

This is the final layer of the convolution-subsampling pair. There are 120 filters in this layer with a convolution size of  $5 \times 5$ . After this layer, the feature map is  $1 \times 1 \times 120$ , and the flattened result is 120 values.



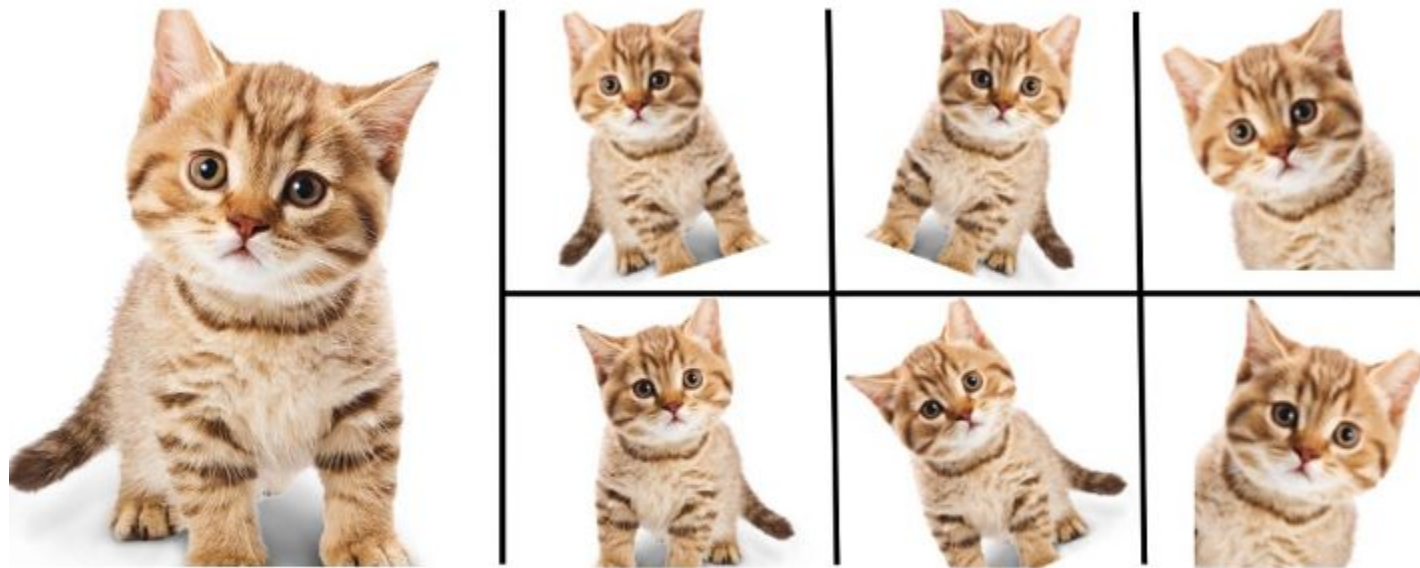




# Summary of the architecture of LeNet5

Layer	Filter	Filter Size	Stride	Size of feature map	Activation Function
Input	-	-	-	32 x 32 x 1	-
Conv 1	6	5 x 5	1	28 x 28 x 6	tanh
Pooling 1	-	2 x 2	2	14 x 14 x 6	-
Conv 2	16	5 x 5	1	10 x 10 x 16	tanh
Pooling 2	-	2 x 2	2	5 x 5 x 16	-
Conv 3	120	5 x 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

# Data Augmentation



## Enlarge your Dataset



Data Augmentation



Original Image



De-texturized



De-colored



Edge Enhanced



Salient Edge Map



Flip/Rotate

Horizontal

Vertically

+45 Rotation

-45 Rotation

Blur



Brighter

Noise added

Darker

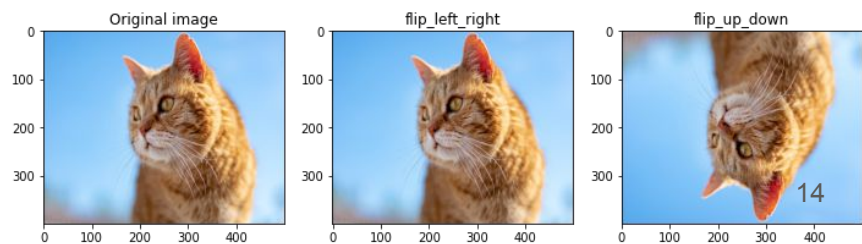
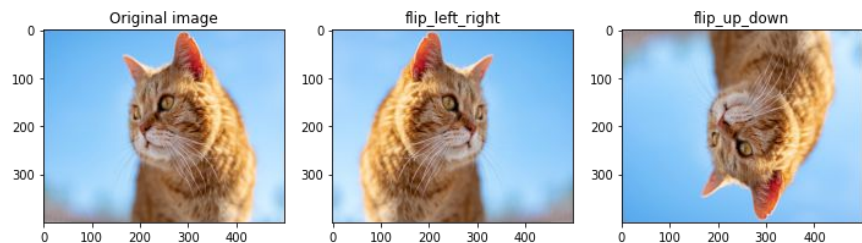
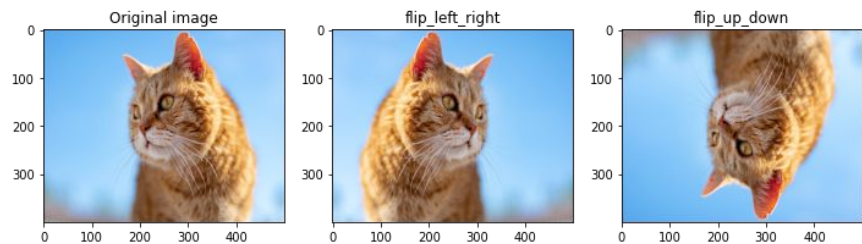
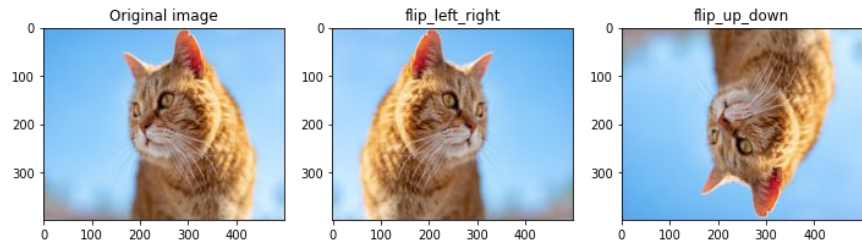
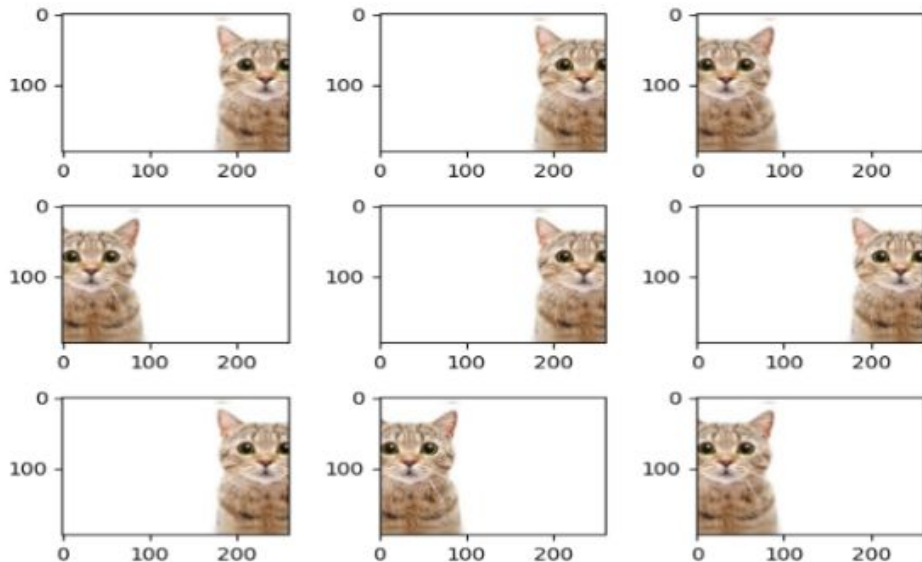
Grayscale

Crop



Augmented Images



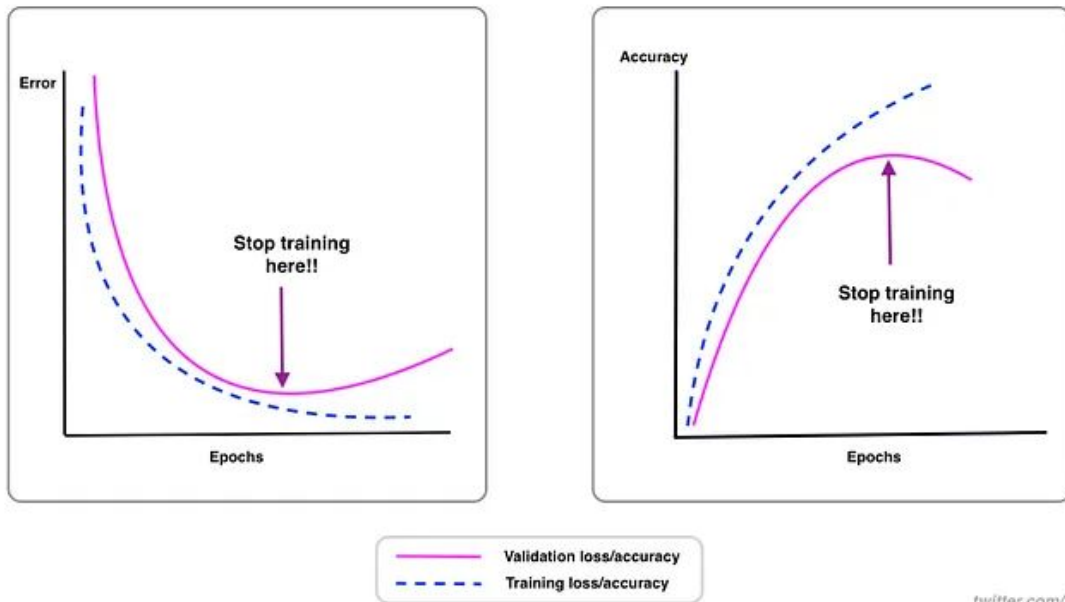


# Early Stopping

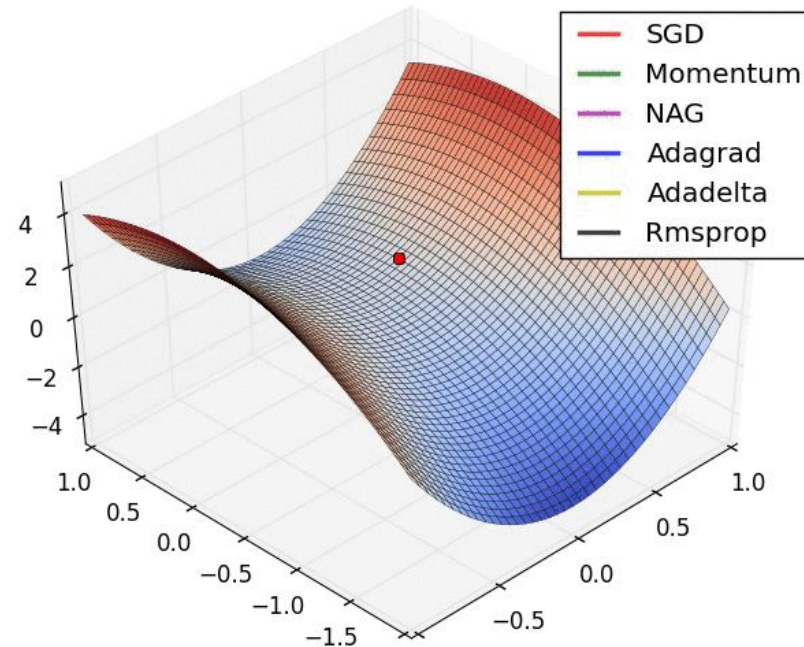
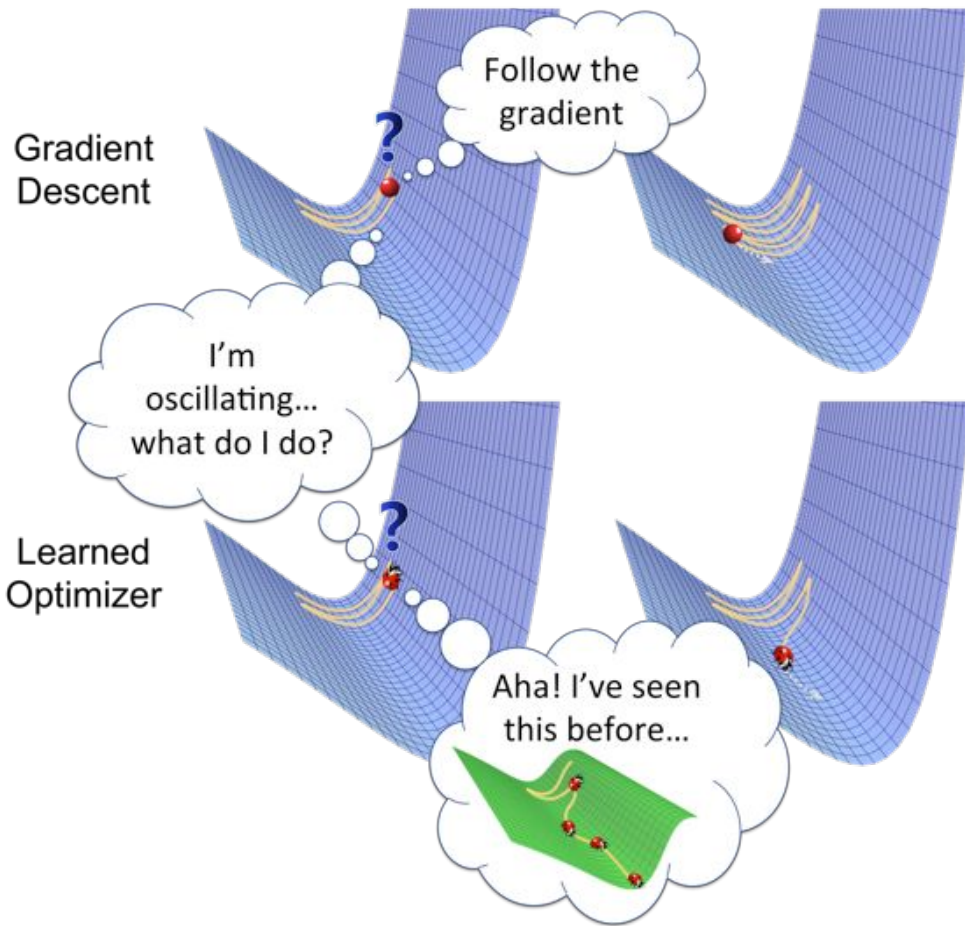
Early stopping is an optimization technique used to reduce overfitting without compromising on model accuracy.

The main idea behind early stopping is to stop training before a model starts to overfit.

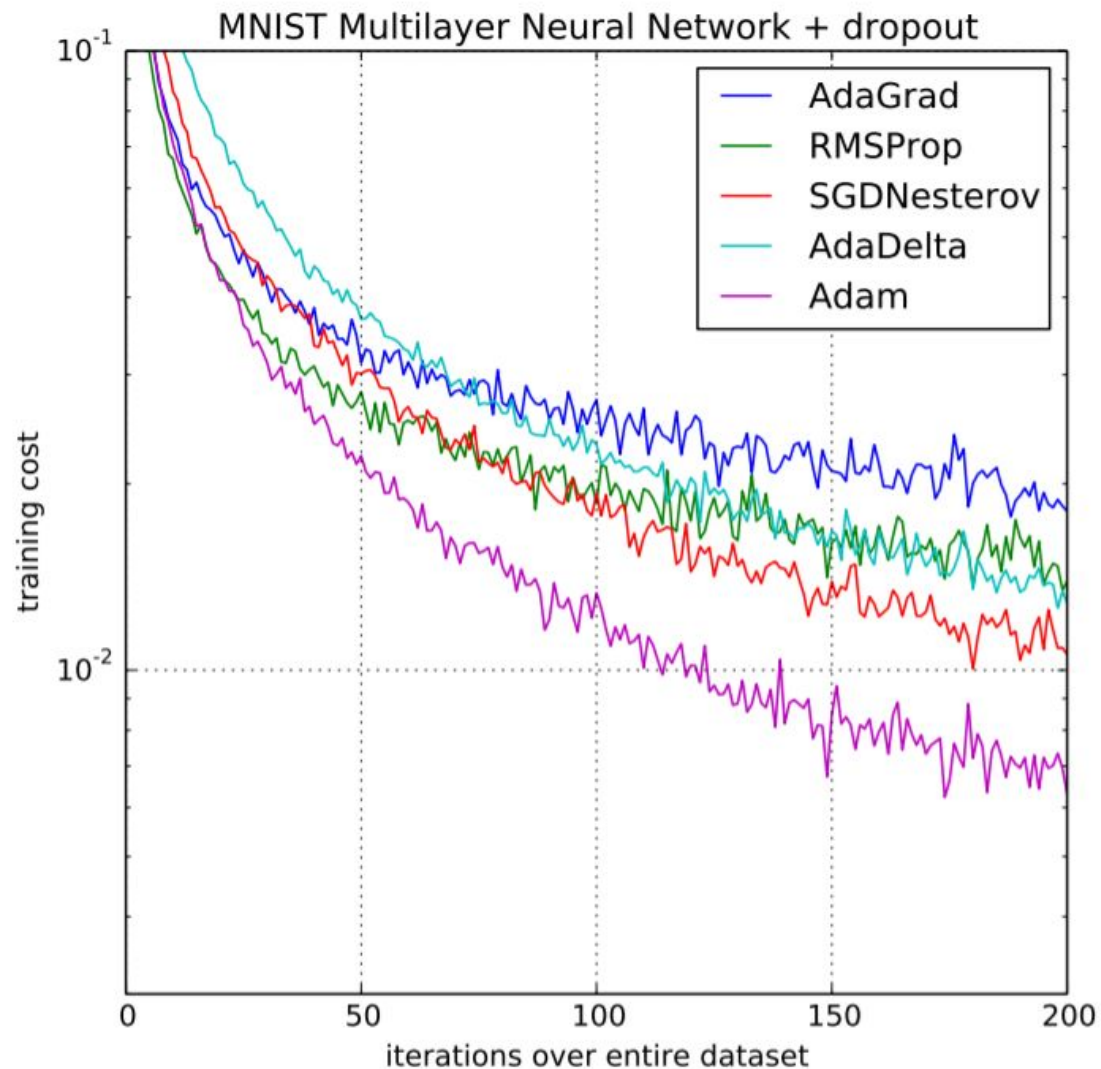
## Early Stopping



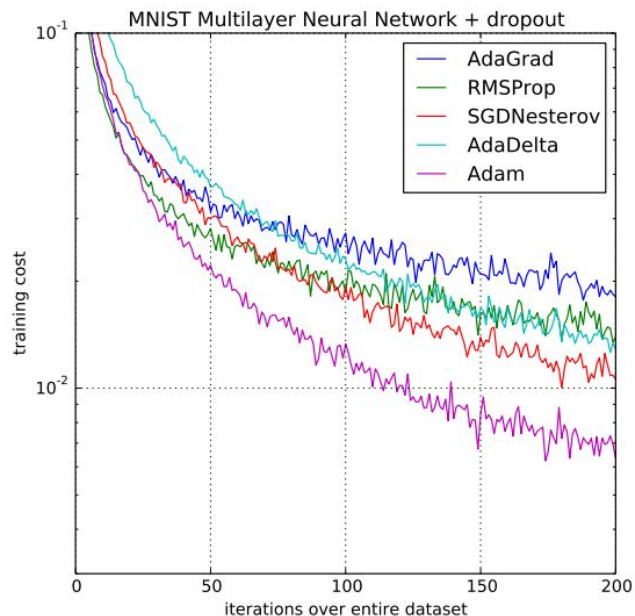
[twitter.com/jeande\\_d](https://twitter.com/jeande_d)







[https://www.tensorflow.org/api\\_docs/python/tf/keras/optimizers](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers)



## Classes

`class Adadelta` : Optimizer that implements the Adadelta algorithm.

`class Adafactor` : Optimizer that implements the Adafactor algorithm.

`class Adagrad` : Optimizer that implements the Adagrad algorithm.

`class Adam` : Optimizer that implements the Adam algorithm.

`class AdamW` : Optimizer that implements the AdamW algorithm.

`class Adamax` : Optimizer that implements the Adamax algorithm.

`class Ftrl` : Optimizer that implements the FTRL algorithm.

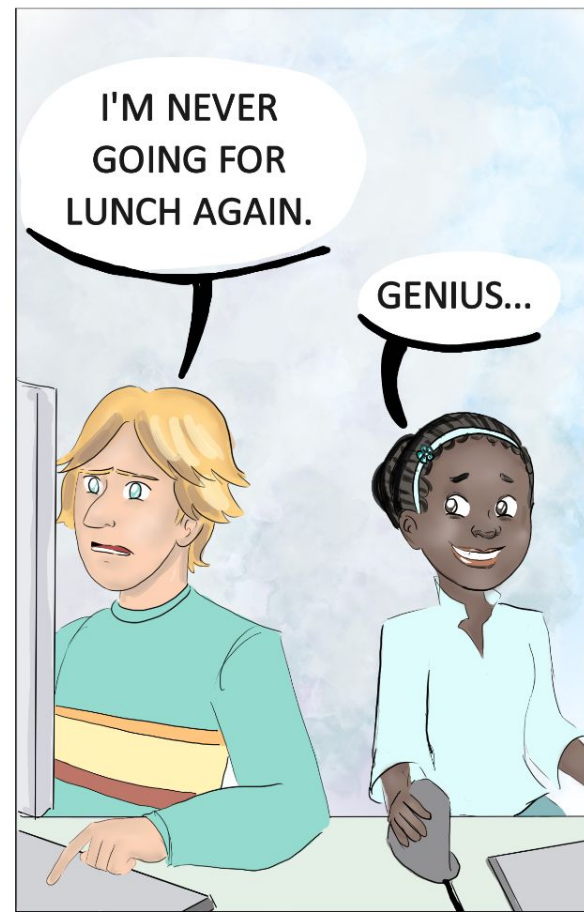
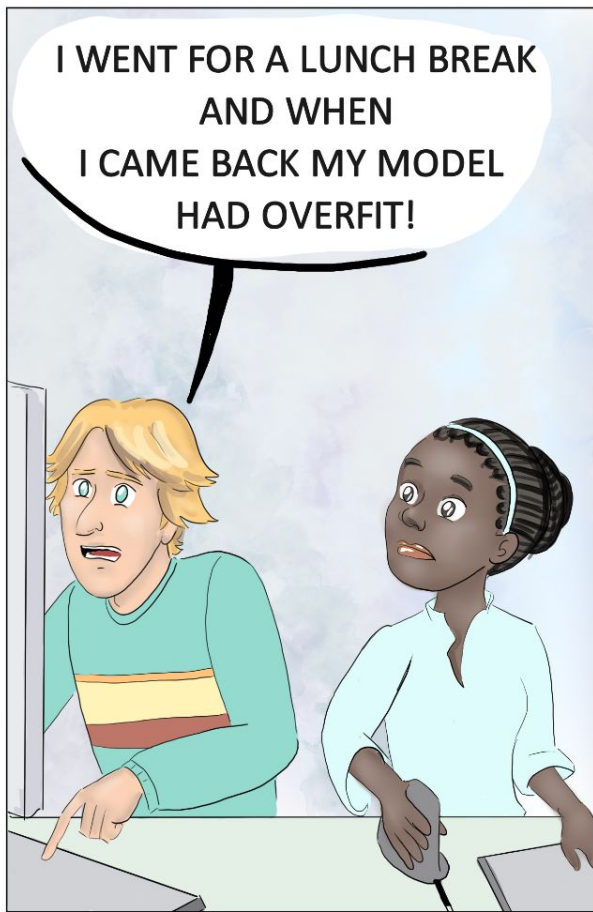
`class Lion` : Optimizer that implements the Lion algorithm.

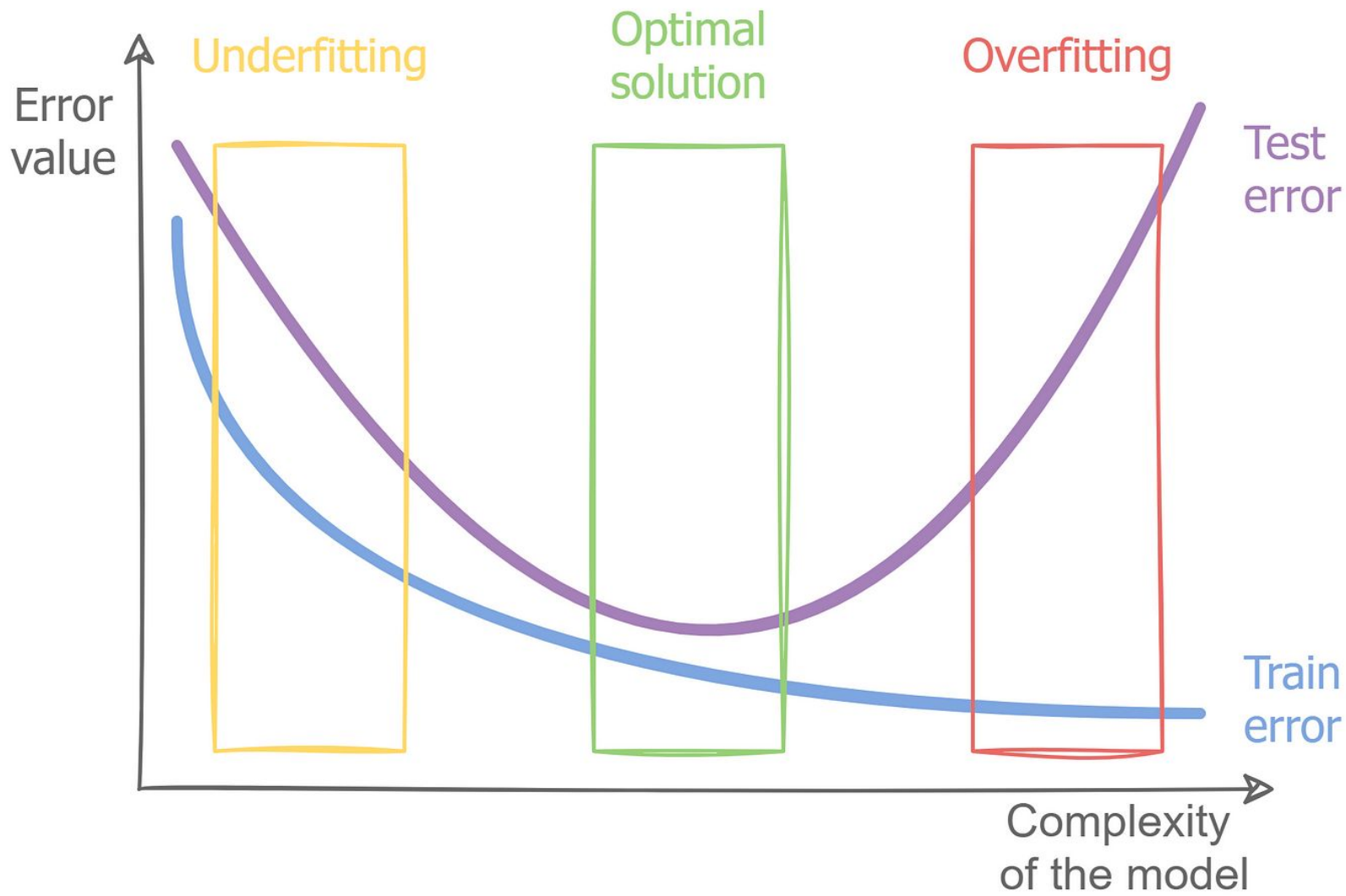
`class Nadam` : Optimizer that implements the Nadam algorithm.

`class Optimizer` : Abstract optimizer base class.

`class RMSprop` : Optimizer that implements the RMSprop algorithm.

`class SGD` : Gradient descent (with momentum) optimizer.





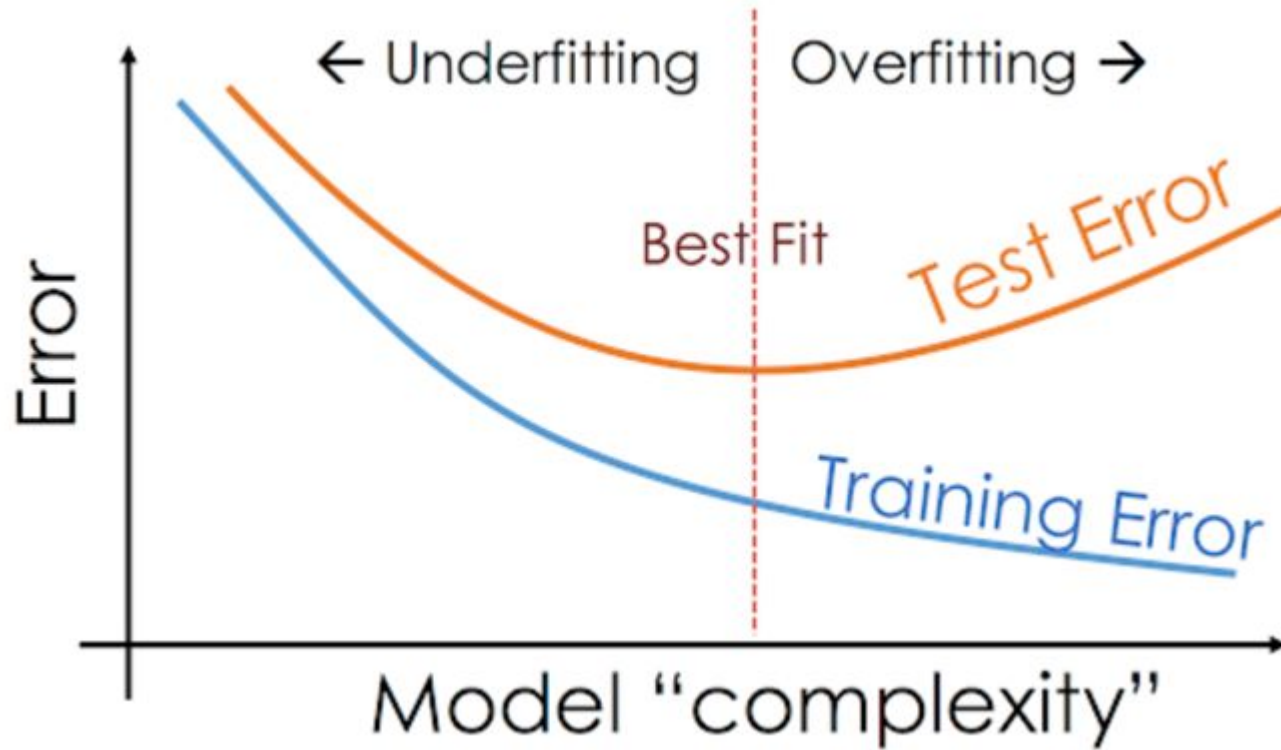
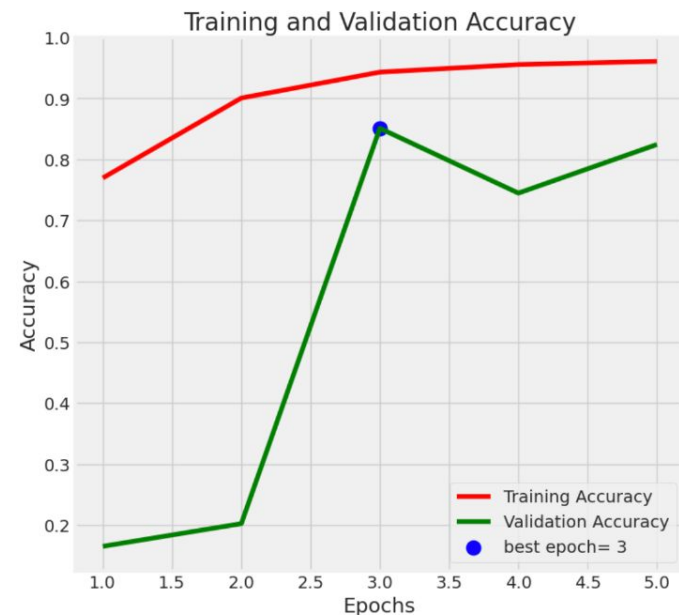
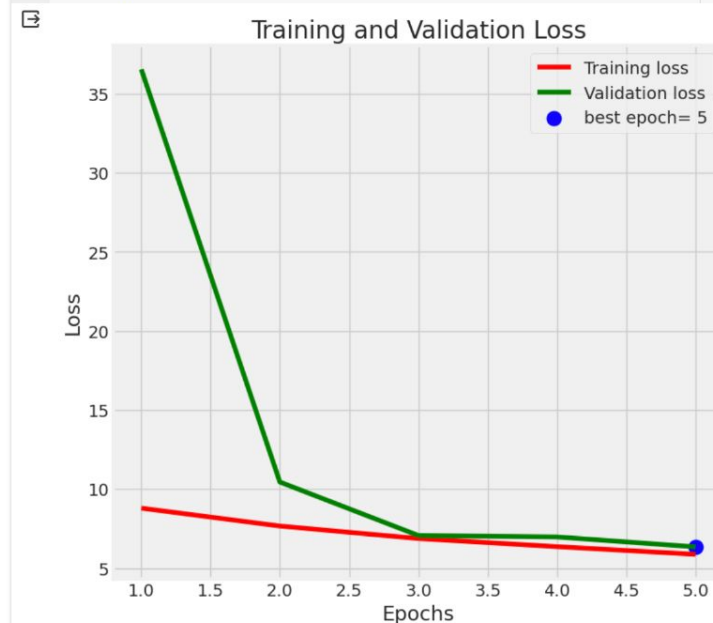


Table of contents

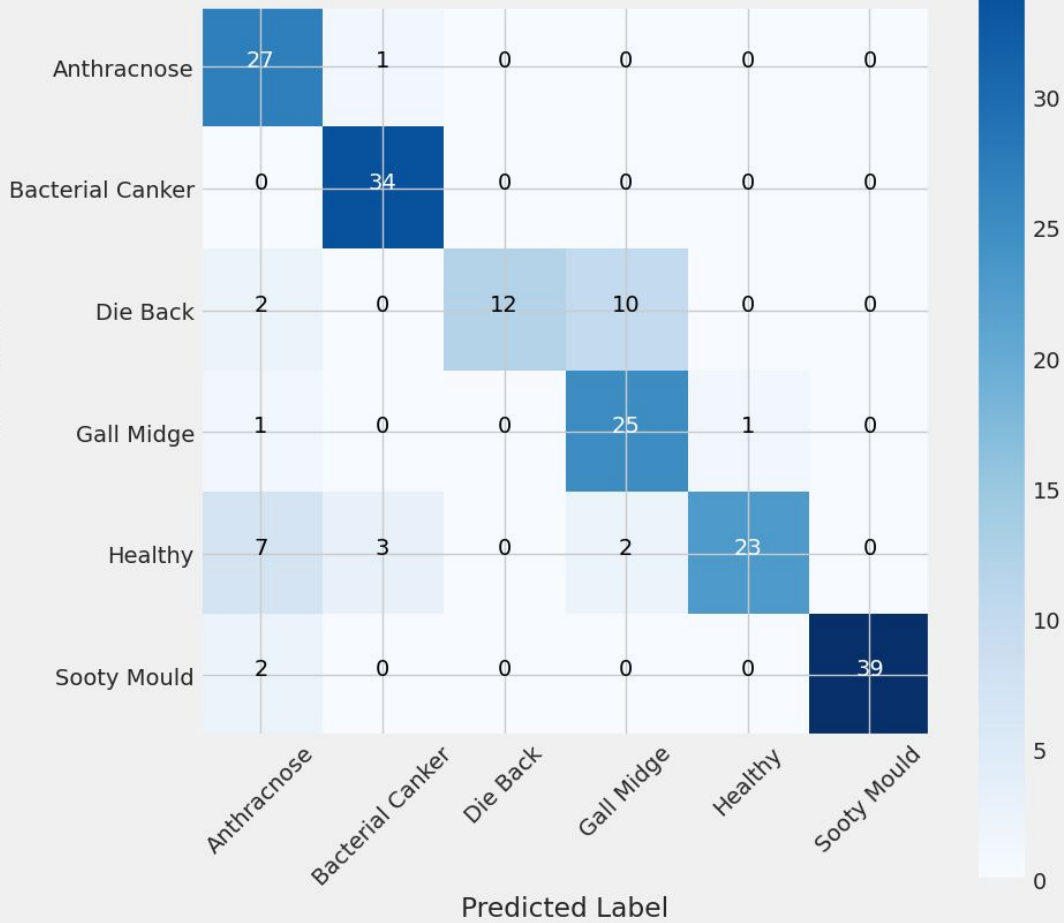
- Split dataframe into train, valid, and test
- Create image data generator
- Define data generators with augmentation
  - Show sample from train data
- Model Structure
  - Generic Model Creation
  - CNN Model Type1: EfficientNetB0
  - CNN Model Type2: ResNet50
  - CNN Model Type3: (Custom Deep Learning Network) LeNet-5 with Average Pooling
  - CNN Model Type4: (Custom Deep Learning Network) LeNet-5 with Max Pooling
  - CNN Model Type5: (Custom Deep Learning Network) LeNet-5 with ReLU Activation Function
  - CNN Model Type6: (Custom Deep Learning Network) VGG16 (Oxford U. Model)
- Homework
  - Train the model while implementing EarlyStopping.
- Display model performance**
- Evaluate model
- Get Predictions
  - Confusion Matrices and Classification Report

```
plt.tight_layout()
plt.show()
```



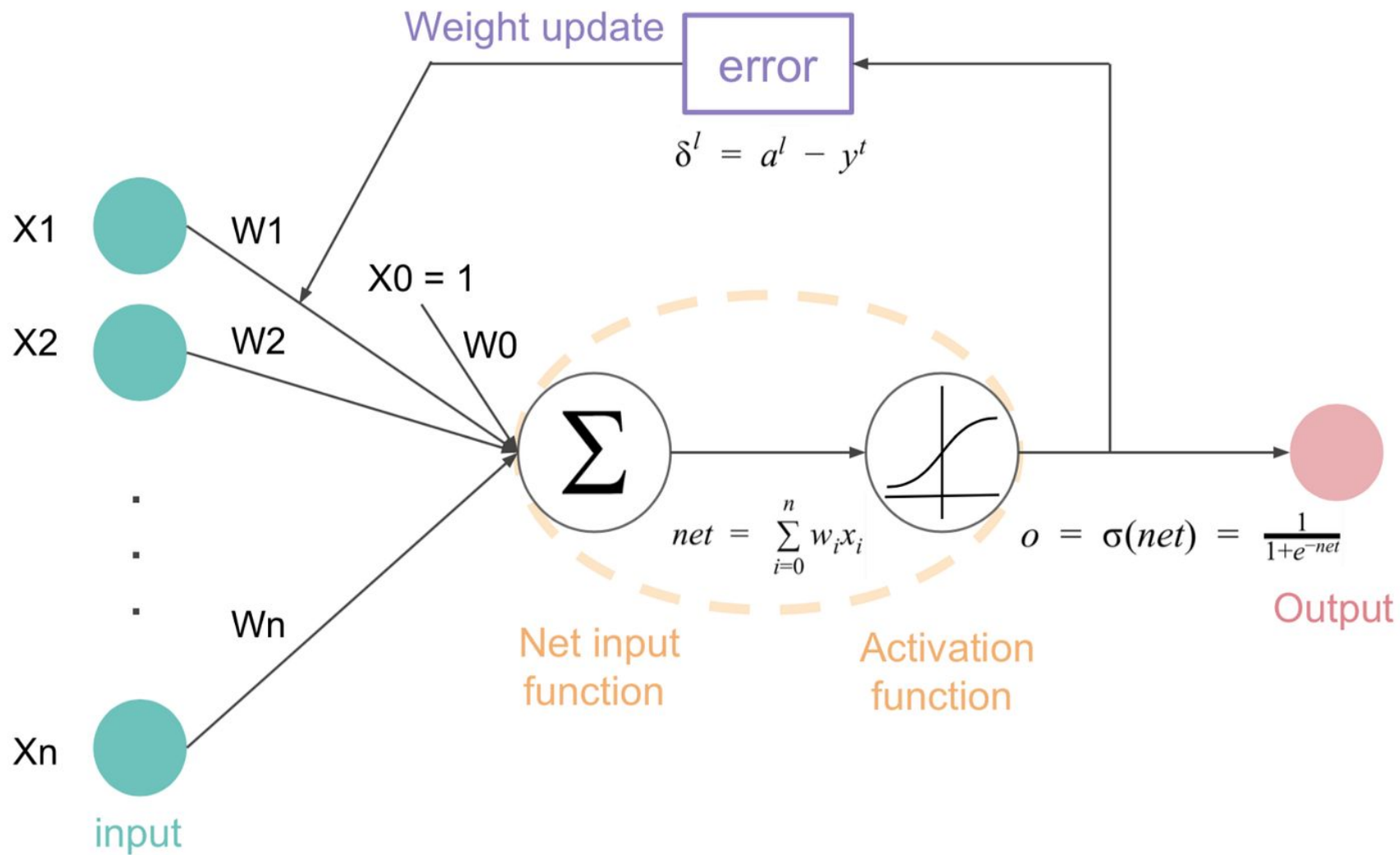


Confusion Matrix



```
[40] # Classification report
print(classification_report(test_gen.classes, y_pred, target_names= classes))
```

	precision	recall	f1-score	support
Anthracnose	0.69	0.96	0.81	28
Bacterial Canker	0.89	1.00	0.94	34
Die Back	1.00	0.50	0.67	24
Gall Midge	0.68	0.93	0.78	27
Healthy	0.96	0.66	0.78	35
Sooty Mould	1.00	0.95	0.97	41
accuracy			0.85	189
macro avg	0.87	0.83	0.83	189
weighted avg	0.88	0.85	0.84	189

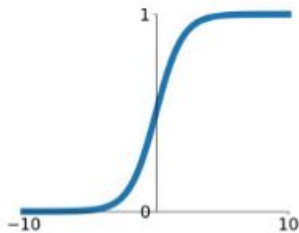




# Activation Functions

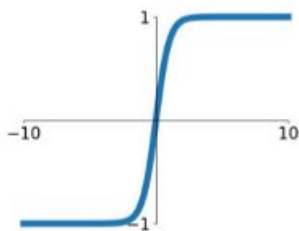
## Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



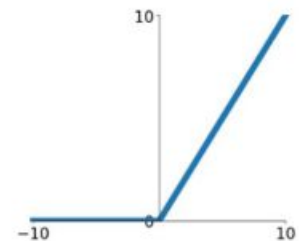
## tanh

$$\tanh(x)$$



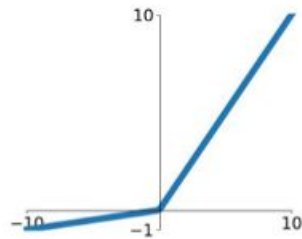
## ReLU

$$\max(0, x)$$



## Leaky ReLU

$$\max(0.1x, x)$$

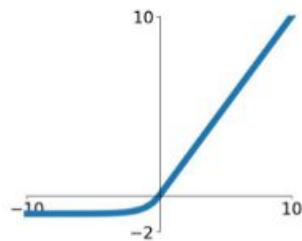


## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

## ELU

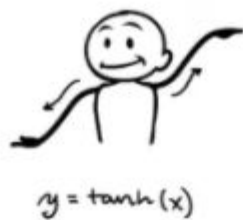
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



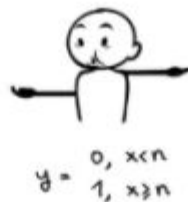
Sigmoid



Tanh



Step Function



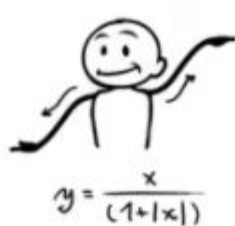
Softplus



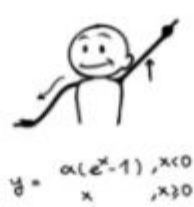
ReLU



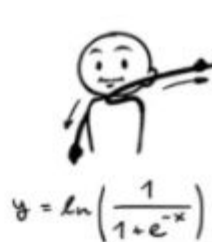
Softsign



ELU



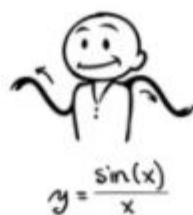
Log of Sigmoid



Swish



Sinc



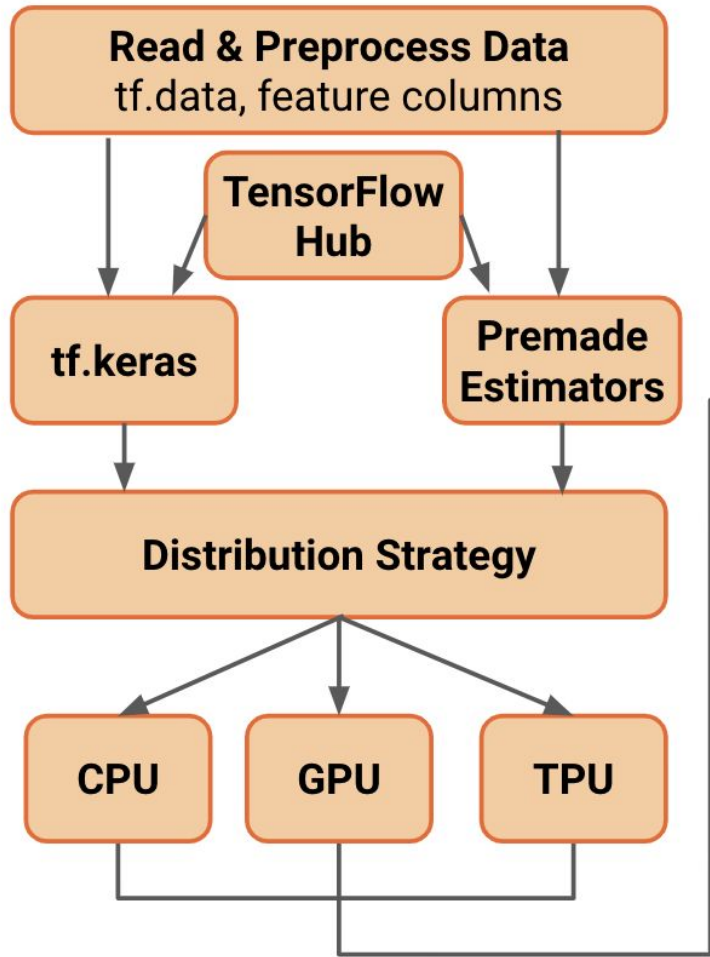
Leaky ReLU



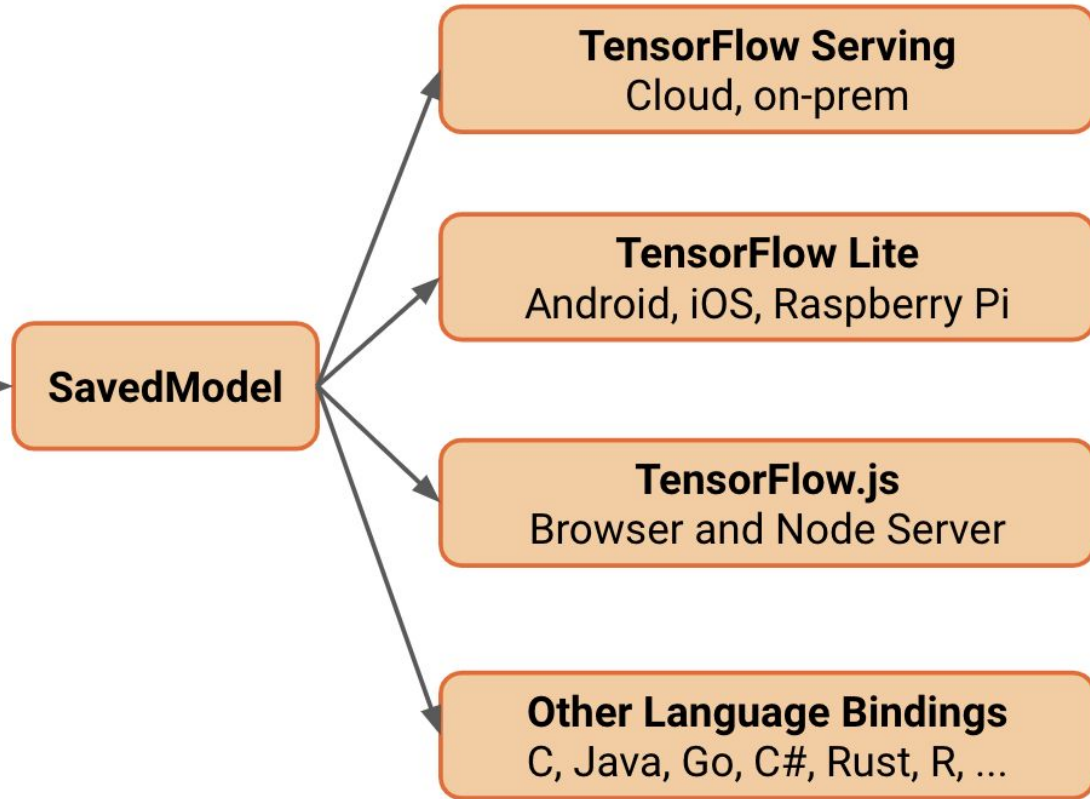
Mish



## TRAINING



## DEPLOYMENT



✓  
0s

```
[32] from tensorflow.keras.callbacks import EarlyStopping
```

✓  
2m

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
batch_size = 16 # set batch size for training
```

```
epochs = 5 # number of all epochs in training
```

```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

```
# Add ModelCheckpoint callback to save the model weights
```

```
model_checkpoint = ModelCheckpoint('cnn_mango_leaf_best_model.h5', save_best_only=True)
```

```
history = model.fit(x=train_gen, epochs=epochs, verbose=1, validation_data=valid_gen,  
                    validation_steps=None, shuffle=False,  
                    callbacks=[early_stopping, model_checkpoint])
```



```
Epoch 1/5
```

```
36/36 [=====] - 53s 497ms/step - loss: 8.7925 - accuracy: 0.7695 - val_loss: 36.5454
```

```
Epoch 2/5
```

```
36/36 [=====] - 11s 309ms/step - loss: 7.6616 - accuracy: 0.9007 - val_loss: 10.4449
```

```
Epoch 3/5
```

```
36/36 [=====] - 11s 310ms/step - loss: 6.8699 - accuracy: 0.9433 - val_loss: 7.0636
```

```
Epoch 4/5
```

```
36/36 [=====] - 12s 335ms/step - loss: 6.3583 - accuracy: 0.9557 - val_loss: 6.9716
```

```
Epoch 5/5
```

```
36/36 [=====] - 12s 324ms/step - loss: 5.8728 - accuracy: 0.9610 - val_loss: 6.3440
```

## Files



- {x} ..
- ▶ MangoLeafBD\_dataset\_small\_v2
- ▶ \_\_MACOSX
- ▶ drive
- ▶ sample\_data
  - Mango Diseases-class\_dict.csv
  - cnn\_mango\_leaf\_best\_model.h5
  - resnet50-Mango Diseases-84.66.h5
  - resnet50-Mango Diseases-weights...

+ Code + Text

## ▼ Save model

```

✓ [41] model_name = model.input_names[0][: -6]
2s      subject = 'Mango Diseases'
      acc = test_score[1] * 100
      save_path = ''

      # Save model
      save_id = str(f'{model_name}-{subject}-{ "%.2f" %round(acc, 2)}.h5')
      model_save_loc = os.path.join(save_path, save_id)
      model.save(model_save_loc)
      print(f'model was saved as {model_save_loc}')

      # Save weights
      weight_save_id = str(f'{model_name}-{subject}-weights.h5')
      weights_save_loc = os.path.join(save_path, weight_save_id)
      model.save_weights(weights_save_loc)
      print(f'weights were saved as {weights_save_loc}')

```

model was saved as resnet50-Mango Diseases-84.66.h5  
 weights were saved as resnet50-Mango Diseases-weights.h5

✓  
0s

```
[43] from tensorflow.keras.models import load_model  
      from tensorflow.keras.preprocessing import image  
      import numpy as np  
      import matplotlib.pyplot as plt  
      import ipywidgets as widgets  
      from IPython.display import display  
      import os, io
```

✓  
3s

```
[44] # Load your pre-trained model  
      model = load_model('/content/cnn_mango_leaf_best_model.h5')
```



✓  
0s

```
[56] # Get value counts
value_counts = df['labels'].value_counts()

# Create class indices dictionary
class_indices = {idx: label for idx, label
                  in enumerate(value_counts.index)}
class_indices

{0: 'Die Back',
 1: 'Bacterial Canker',
 2: 'Sooty Mould',
 3: 'Healthy',
 4: 'Gall Midge',
 5: 'Anthracnose'}
```

Upload (3)

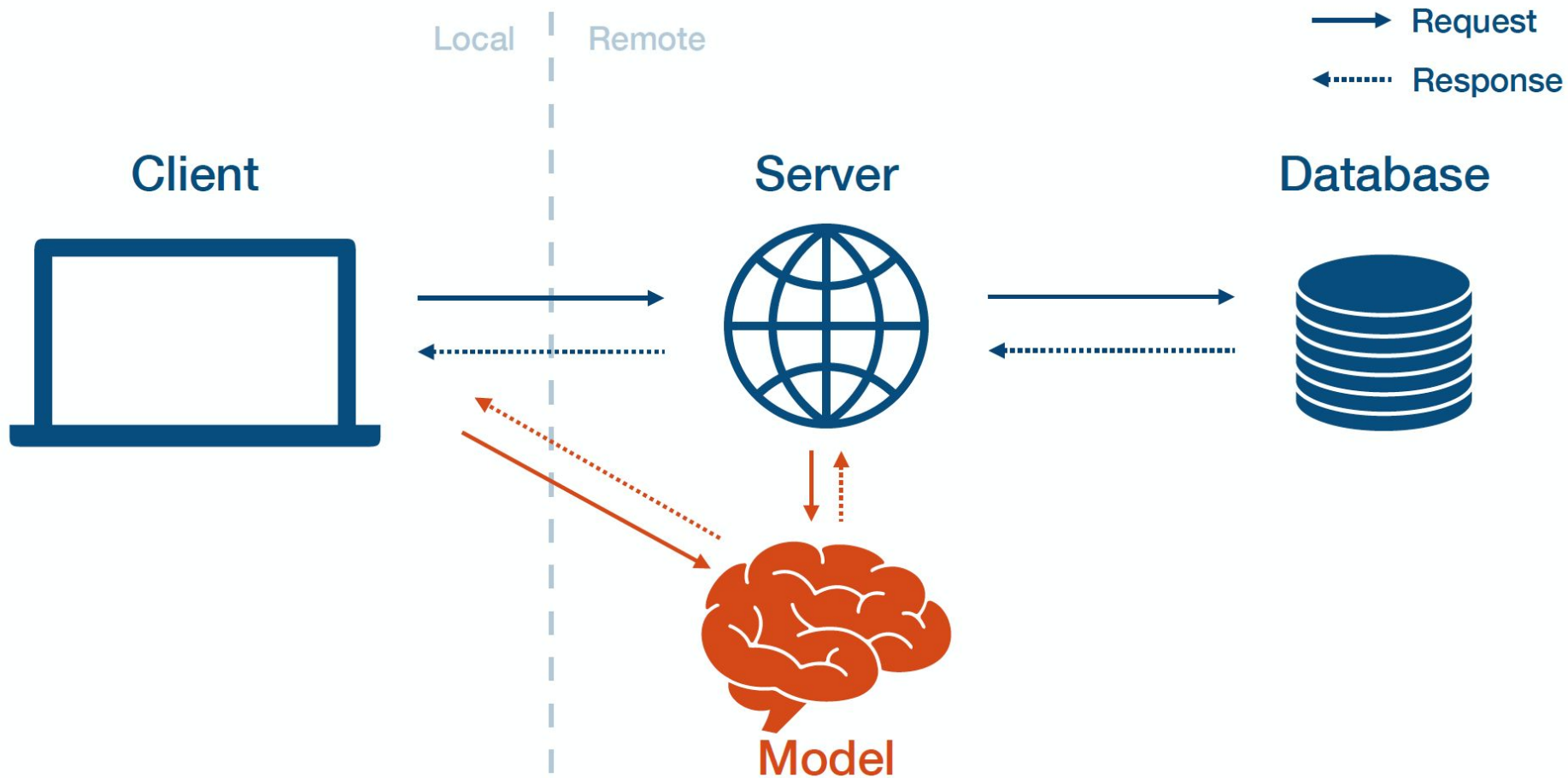
1/1 [=====] - ETA: 0s

1/1 [=====] - 0s 22ms/step

Prediction: Sooty Mould, Confidence: 0.3687029778957367





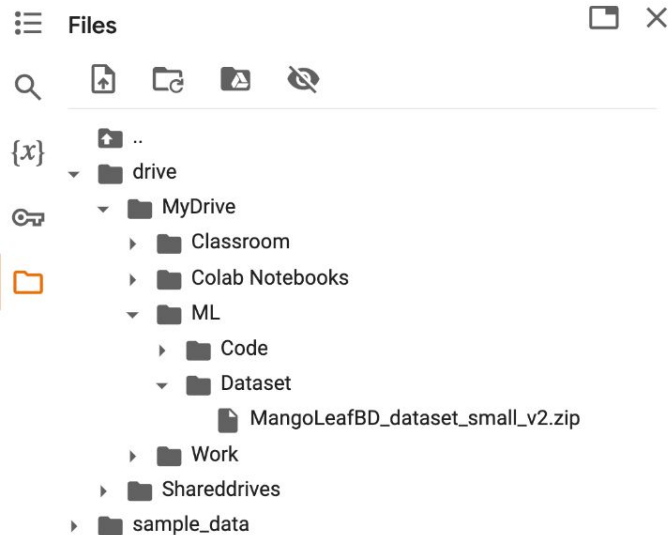


# How to Import Files from Google Drive to Colab



Colab logo mango\_leaf\_train\_and\_save\_weight\_2024v3\_part2.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)



+ Code + Text

## Download Sample Dataset (From Google Drive)

```
✓ [4] from google.colab import drive  
21s drive.mount('/content/drive')
```

Mounted at /content/drive

```
[ ] # e.g., sample path: /content/drive/MyDrive/ML/Dataset/MangoLeafBD_dataset_small_v2.zip
```

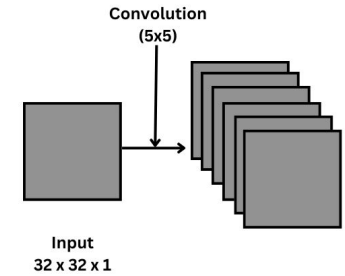
Don't forget to uncomment this line before training the model to unzip the dataset, which format.

```
[ ] # !unzip MangoLeafBD_dataset_small_v2.zip  
!unzip /content/drive/MyDrive/ML/Dataset/MangoLeafBD_dataset_small_v2.zip
```

# Week 11: Assignment

## Build Your Own Deep Learning Architecture

Design and implement your own deep-learning architecture for a given task or dataset.



$$\begin{aligned}\text{Output Shape} &= ((32 - 5 + 1) \times (32 - 5 + 1) \times 6) \\ &= (28 \times 28 \times 6)\end{aligned}$$

