SC310005 Artificial Intelligence

Lecture 10: Deep Learning (Part 2)

teerapong.pa@chula.ac.th

Reference:

1. 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.

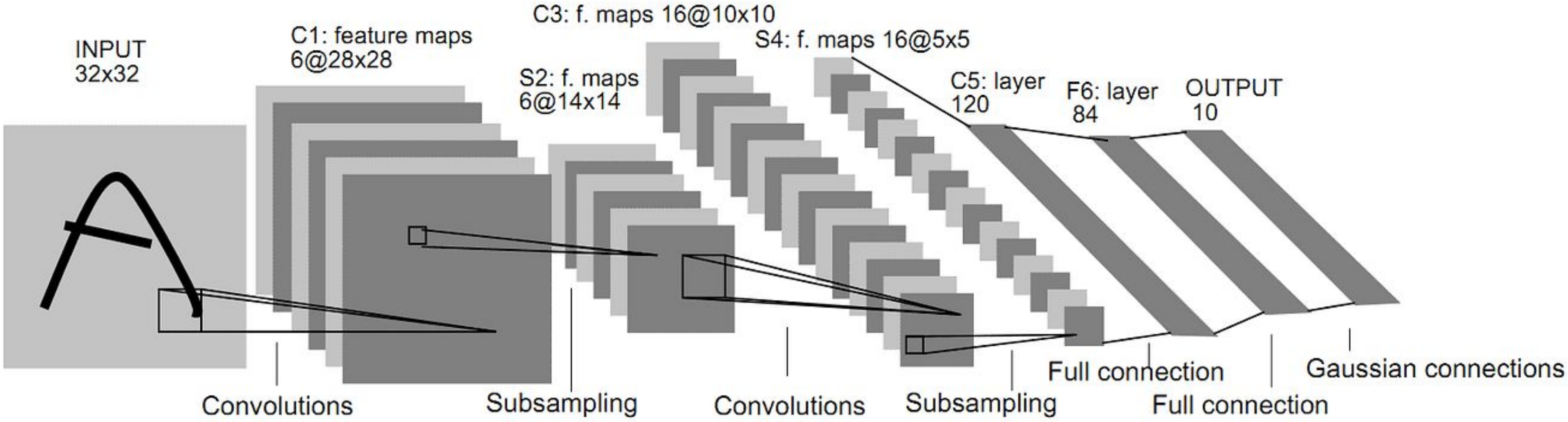
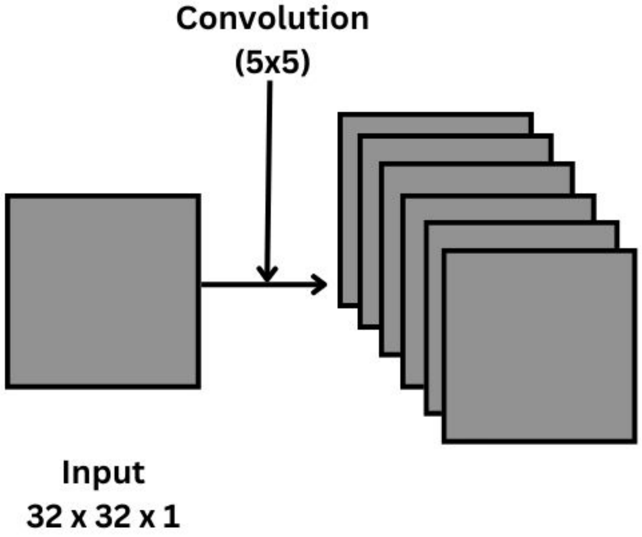


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet5: First layer

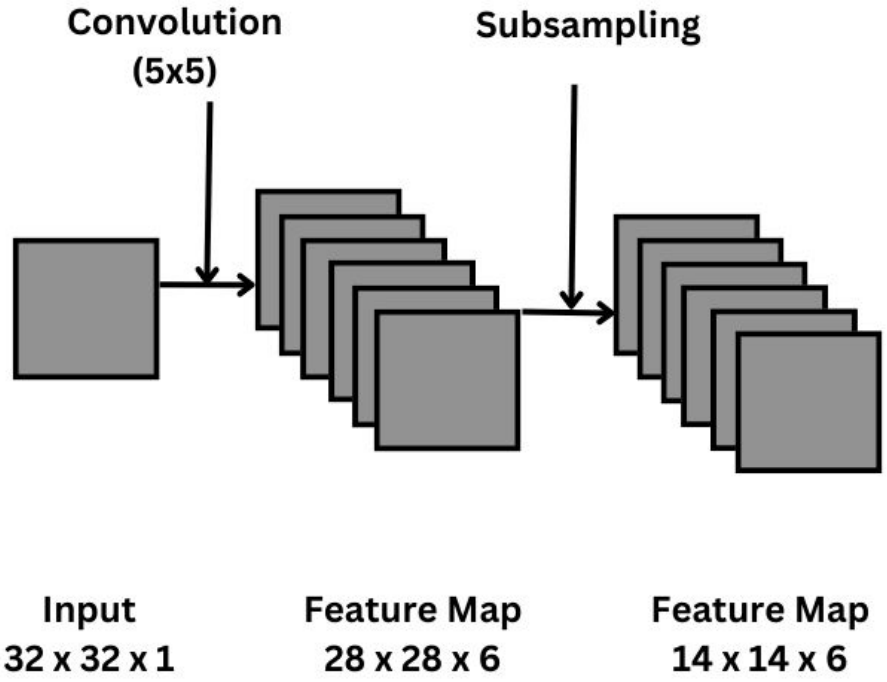
After this, the first convolution step takes place, and the input image is convoluted to the size of 28x28.



Output Shape = ((32 - 5 + 1) x (32 - 5 + 1) x 6)
= (28 x 28 x 6)

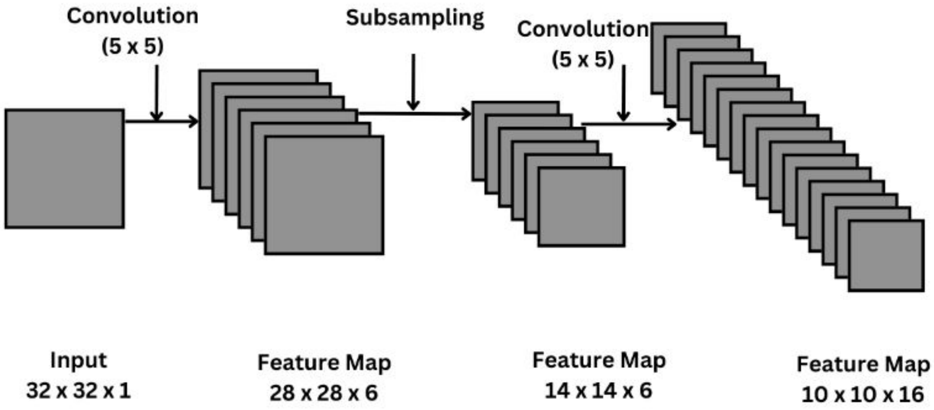
LeNet5: Second layer

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



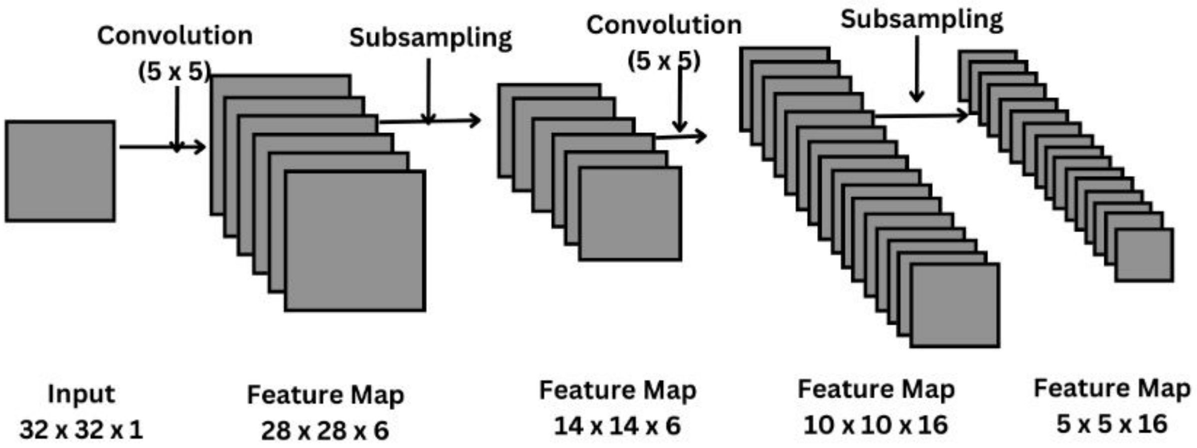
LeNet5: Third layer

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



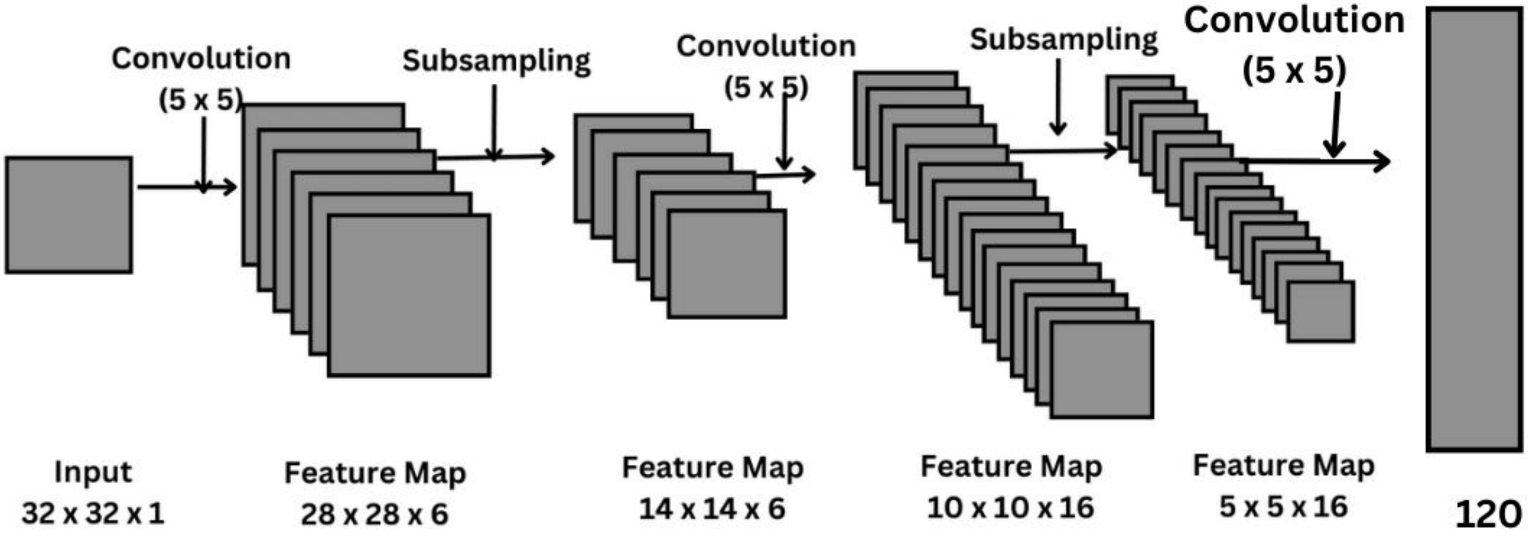
LeNet5: Fourth layer

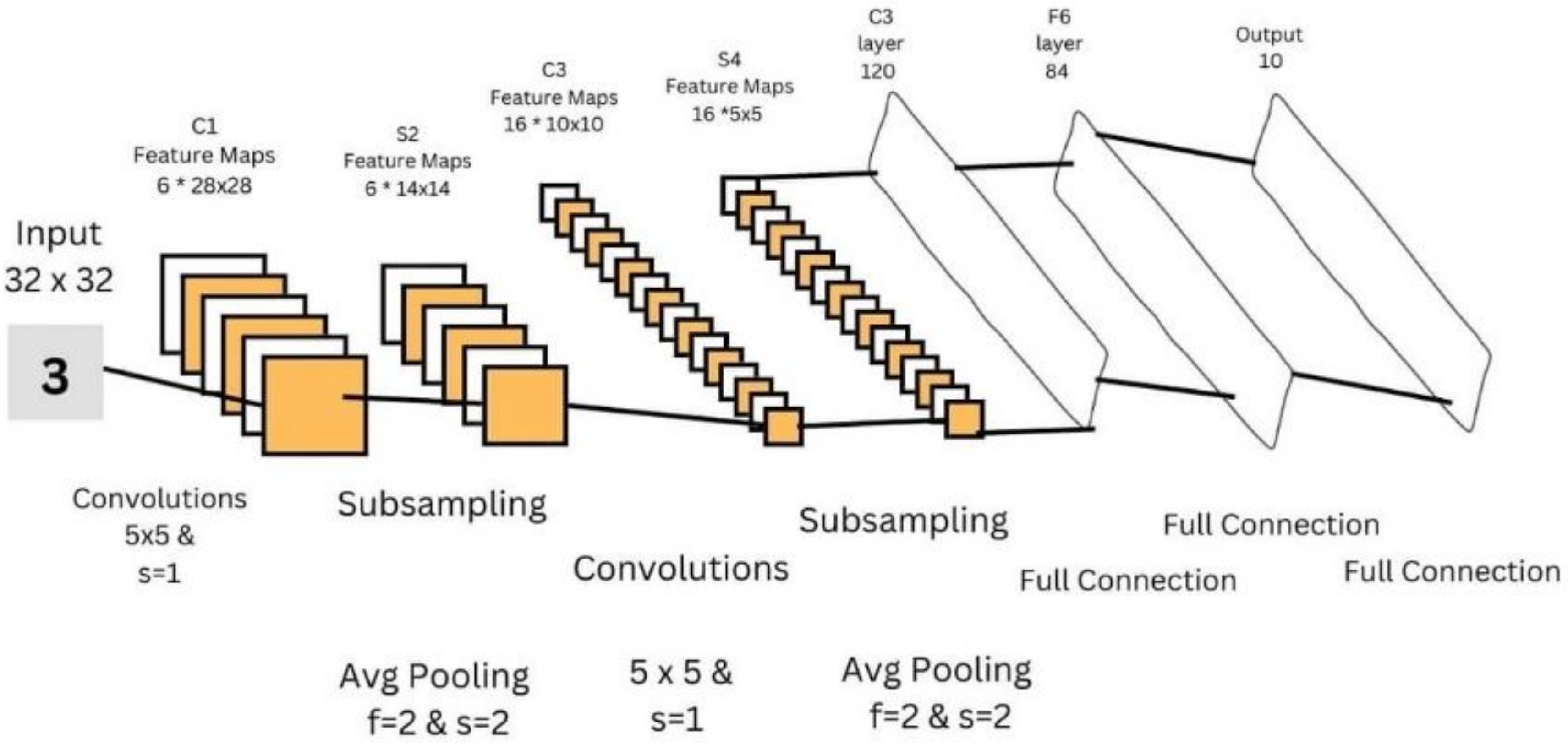
The subsampling takes place, and the image size in this step is reduced to 5x5x16. In this layer, the input for the very last function diagram comes from all the remaining function diagrams.



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 5x5. After this layer, the feature map is 1x1x120, and the flattened result is 120 values.

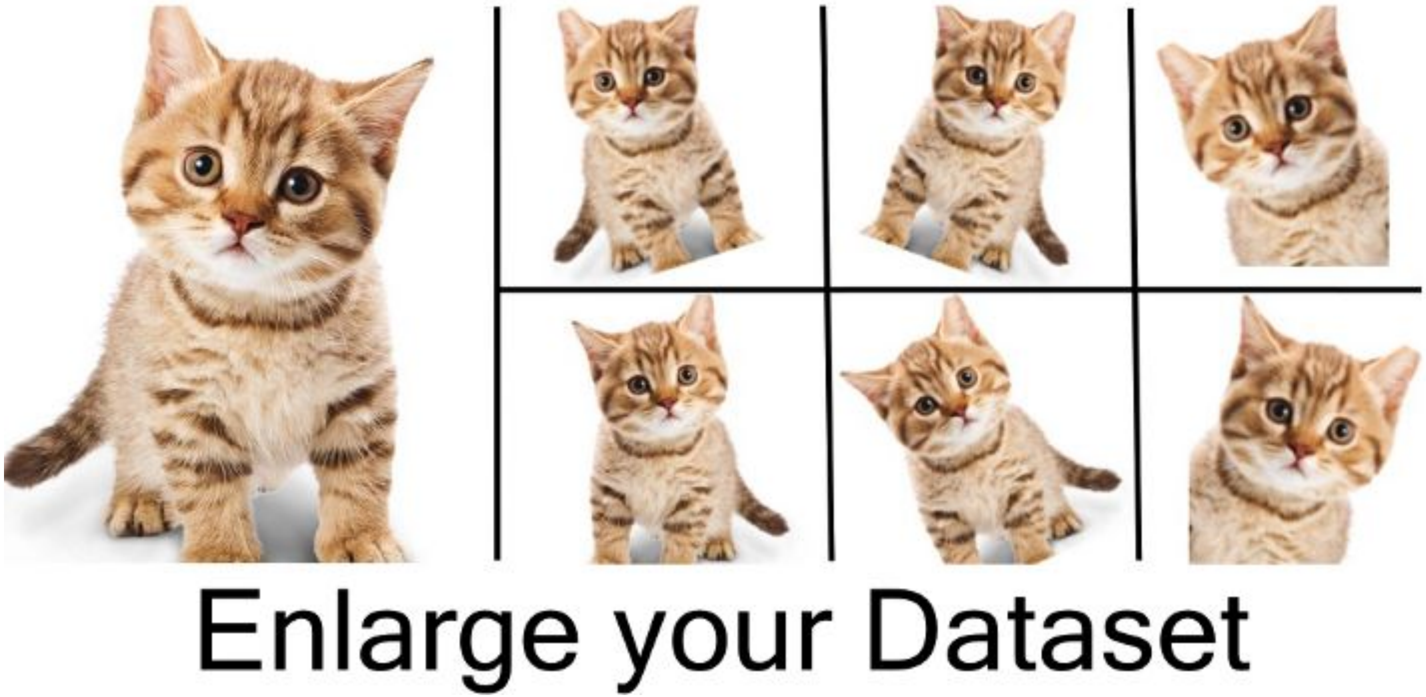


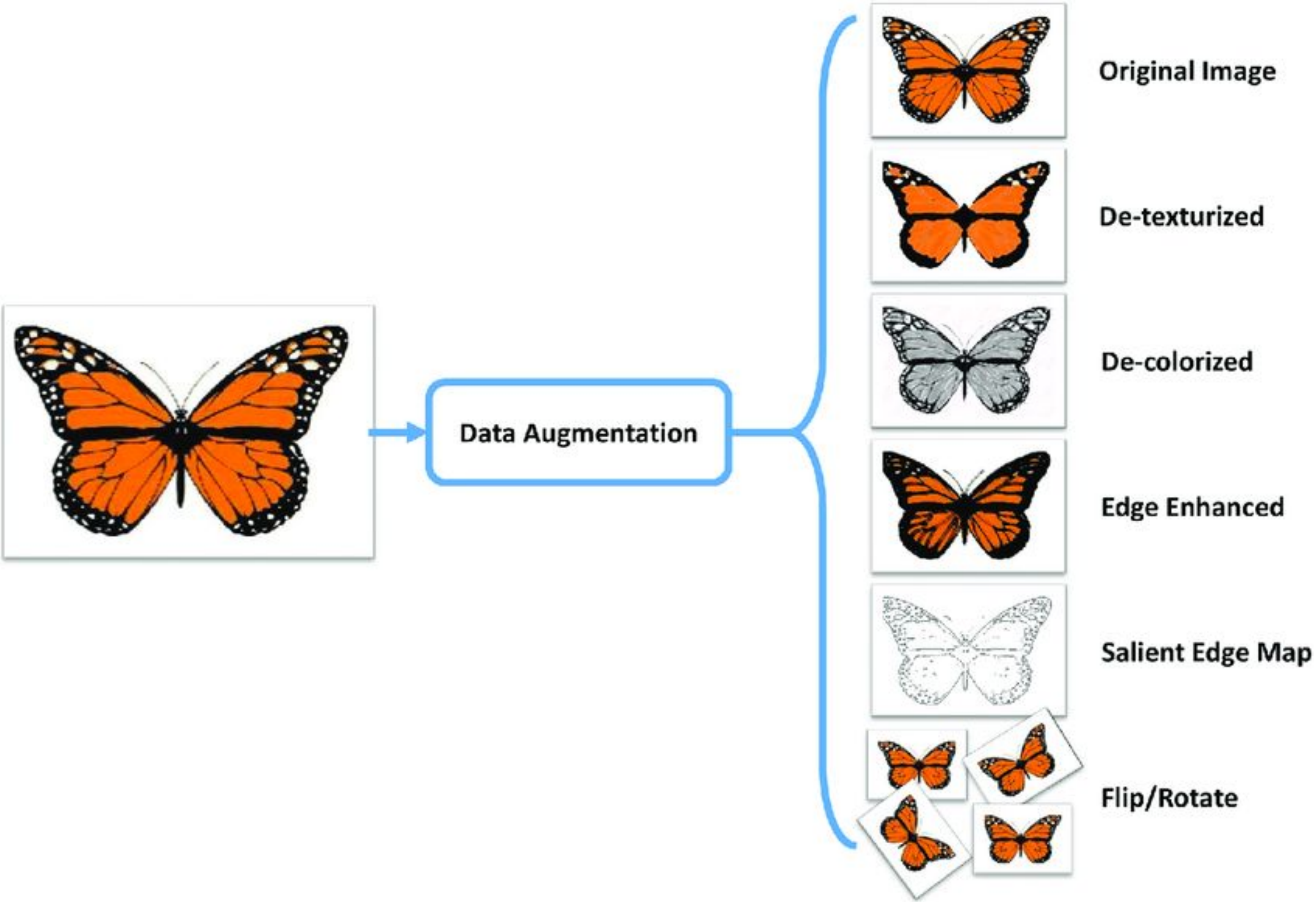

Input
32 x 32
3
C1
Feature Maps
6 * 28x28
S2
Feature Maps
6 * 14x14
C3
Feature Maps
16 * 10x10
S4
Feature Maps
16 *5x5
C3
layer
120
F6
layer
84
Output
10
Convolutions
5x5 &
s=1
Subsampling
Convolutions
Subsampling
Full Connection
Full Connection
Full Connection
Avg Pooling
f=2 & s=2
5 x 5 &
s=1
Avg Pooling
f=2 & s=2

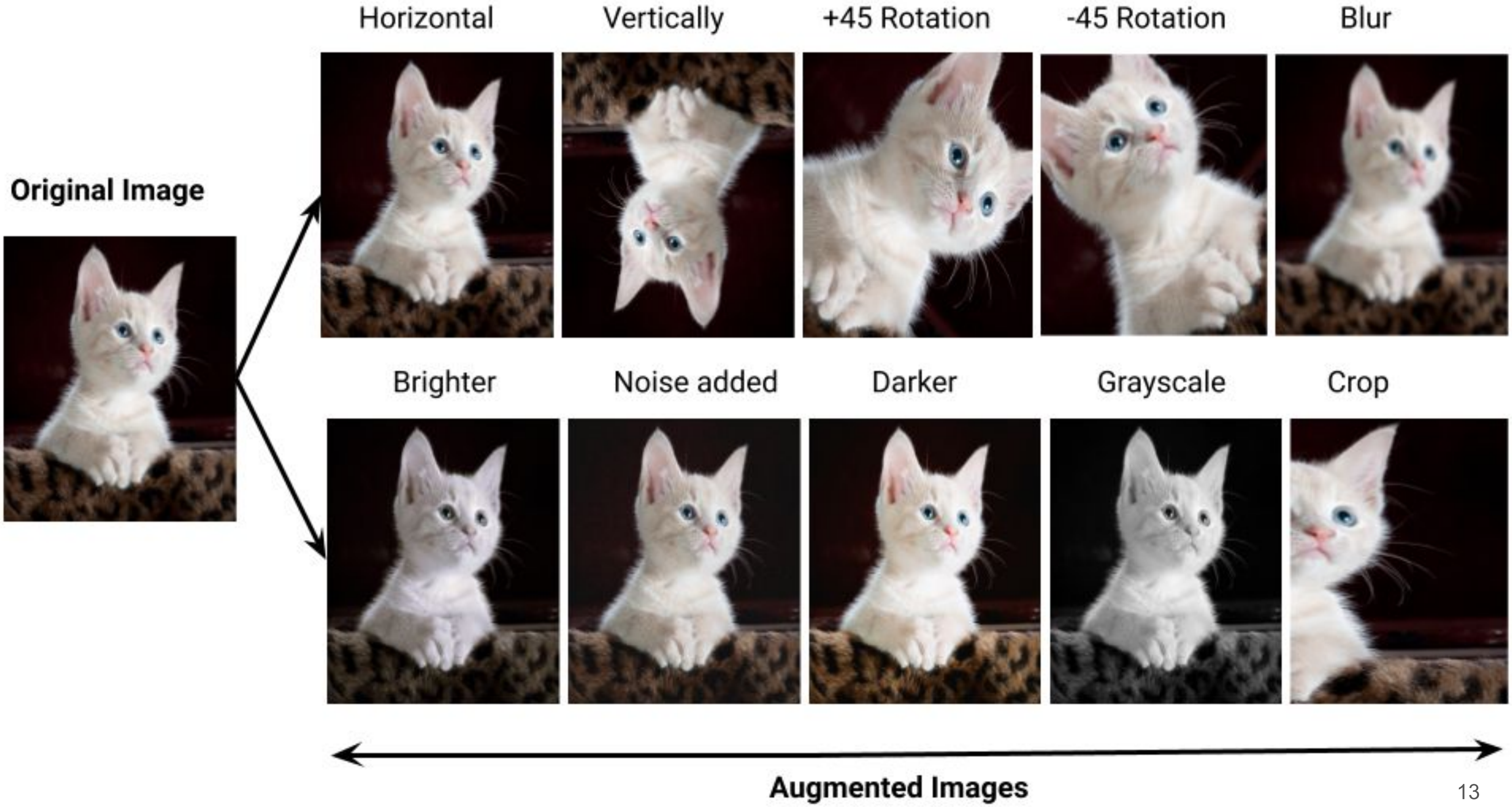
Summary of the architecture of LeNet5

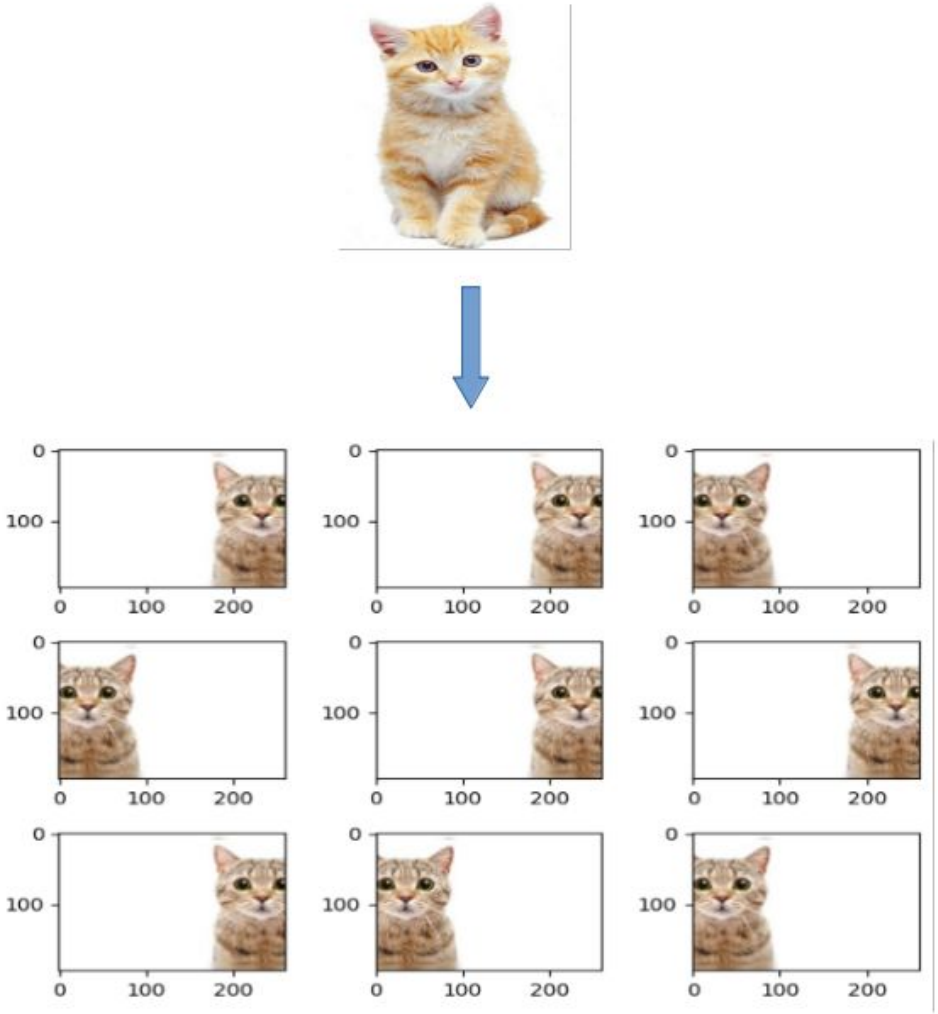
Layer	Filter	Filter Size	Stride	Size of feature map	Activation Function
Input	-	-	-	32 x 32 x 1	-
Conv 1	6	5 x 5	1	28 x 28 x 6	tanh
Pooling 1	-	2 x 2	2	14 x 14 x 6	-
Conv 2	16	5 x 5	1	10 x 10 x 16	tanh
Pooling 2	-	2 x 2	2	5 x 5 x 16	-
Conv 3	120	5 x 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully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

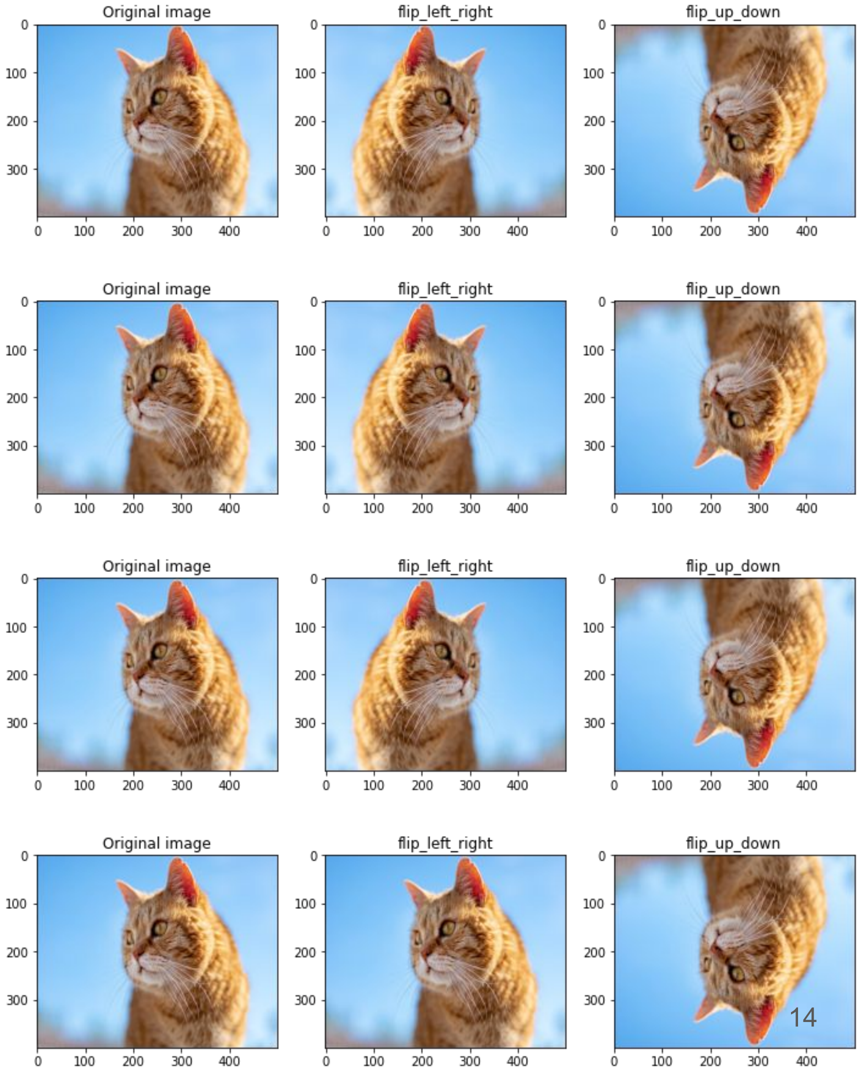
Data Augmentation




Data Augmentation
Original Image
De-texturized
De-colorized
Edge Enhanced
Salient Edge Map
Flip/Rotate


Original Image
Horizontal
Vertically
+45 Rotation
-45 Rotation
Blur
Brighter
Noise added
Darker
Grayscale
Crop
Augmented Images

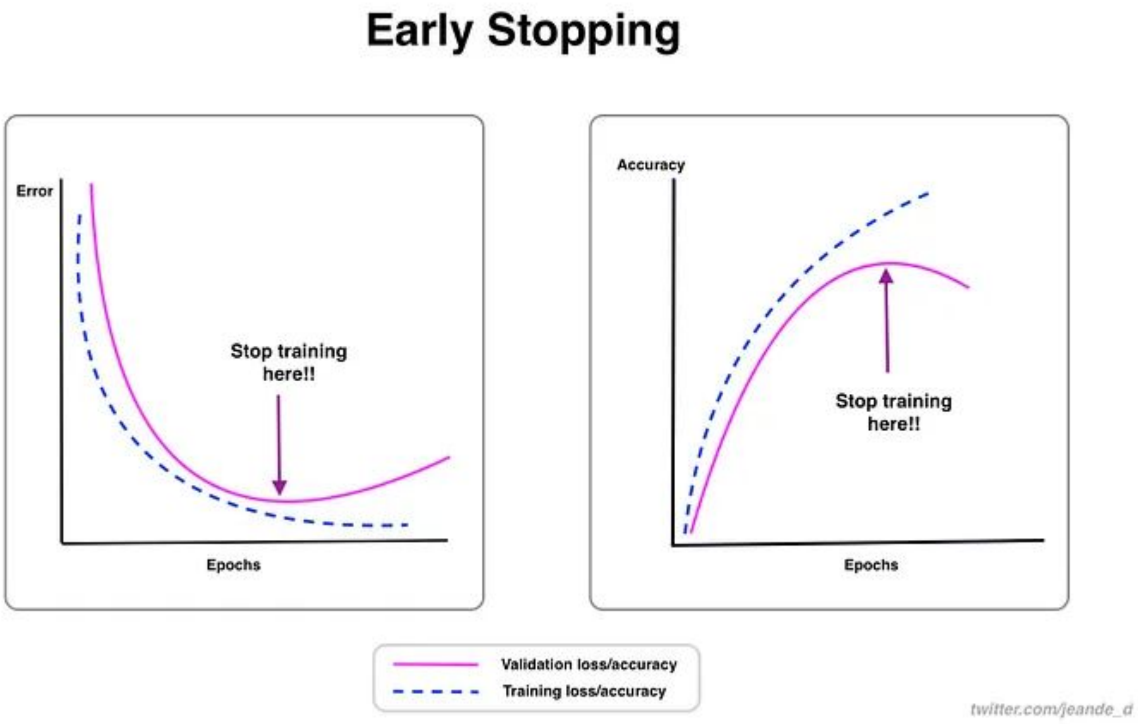


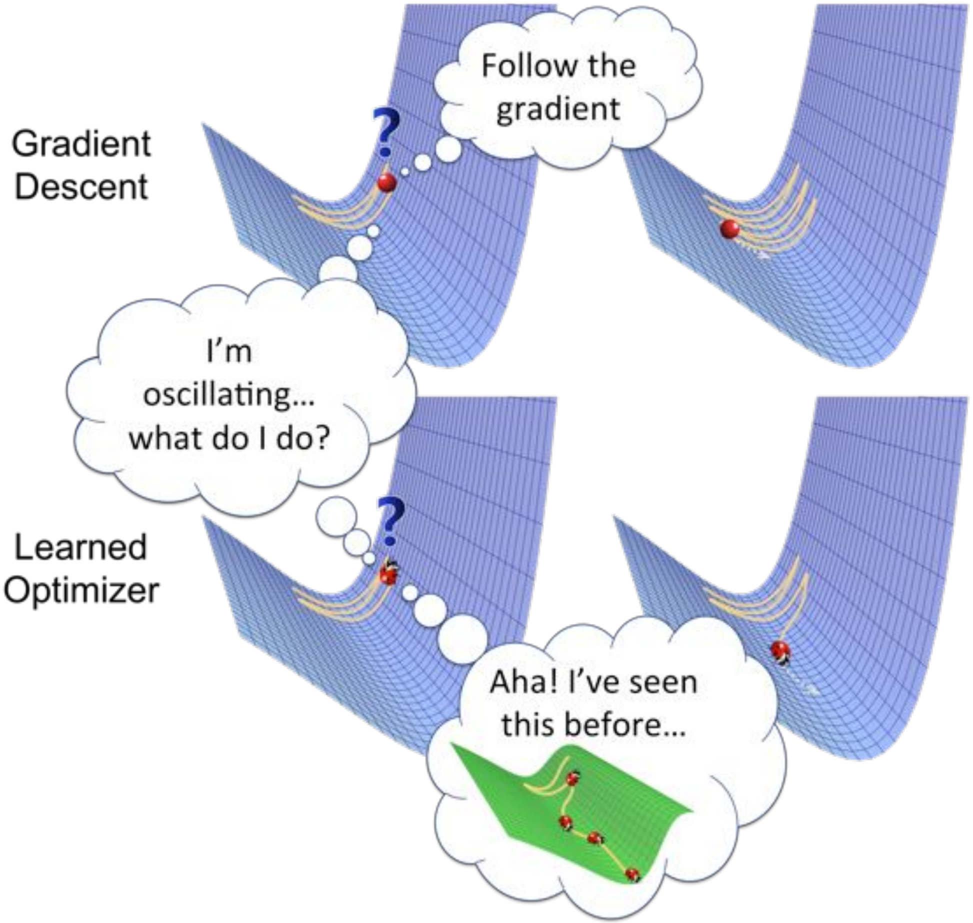

Original image
flip_left_right
flip_up_down
Original image
flip_left_right
flip_up_down
Original image
flip_left_right
flip_up_down
Original image
flip_left_right
flip_up_down

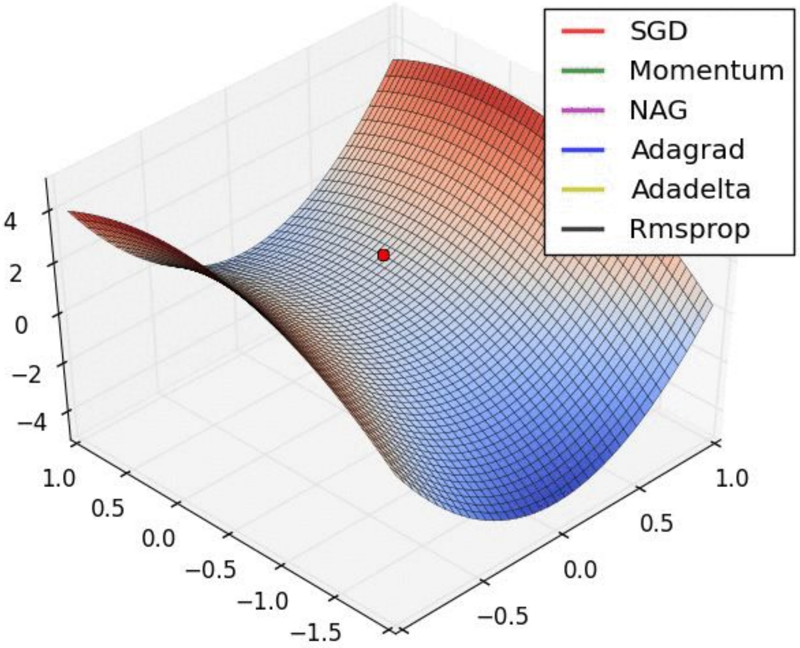
Early Stopping

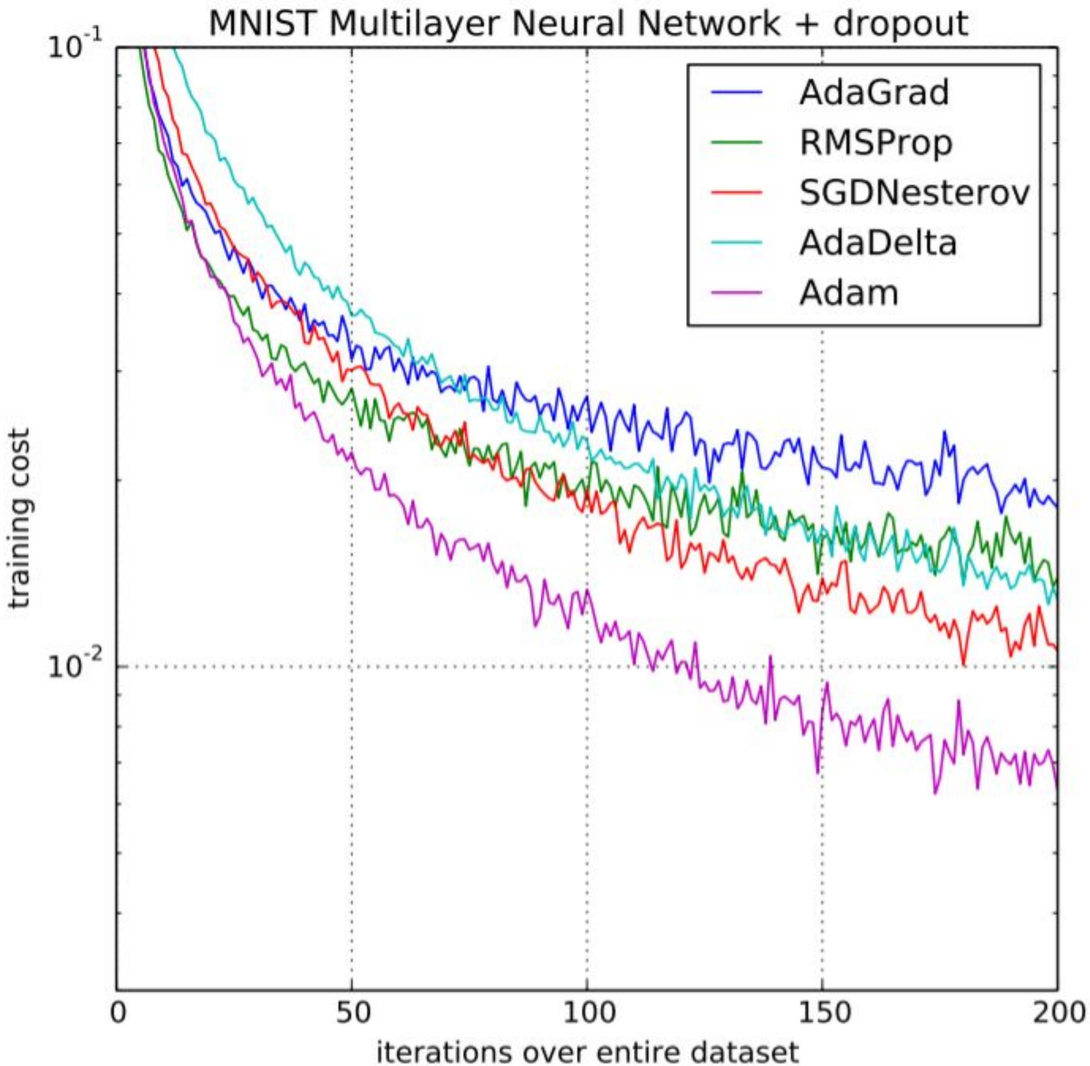
Early stopping is an optimization technique used to reduce overfitting without compromising on model accuracy.

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

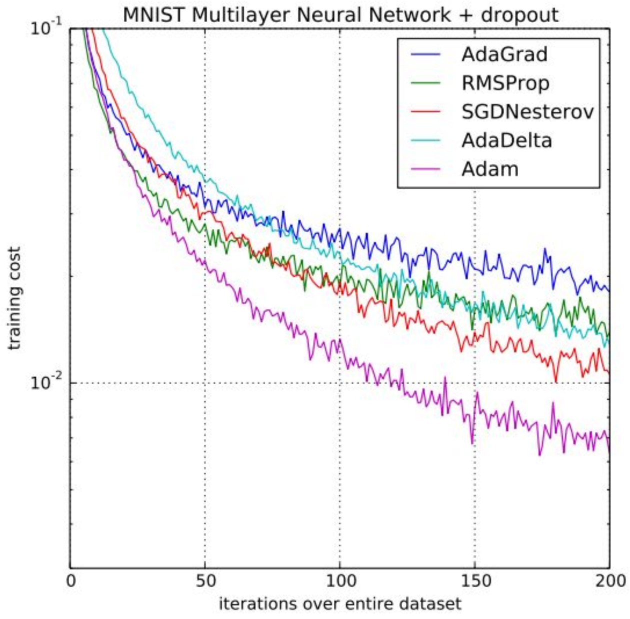



Gradient Descent
Follow the gradient
I'm oscillating... what do I do?
Learned Optimizer
Aha! I've seen this before...


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5


MNIST Multilayer Neural Network + dropout
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam
training cost
iterations over entire dataset
10^{-1}
10^{-2}
0
50
100
150
200

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.

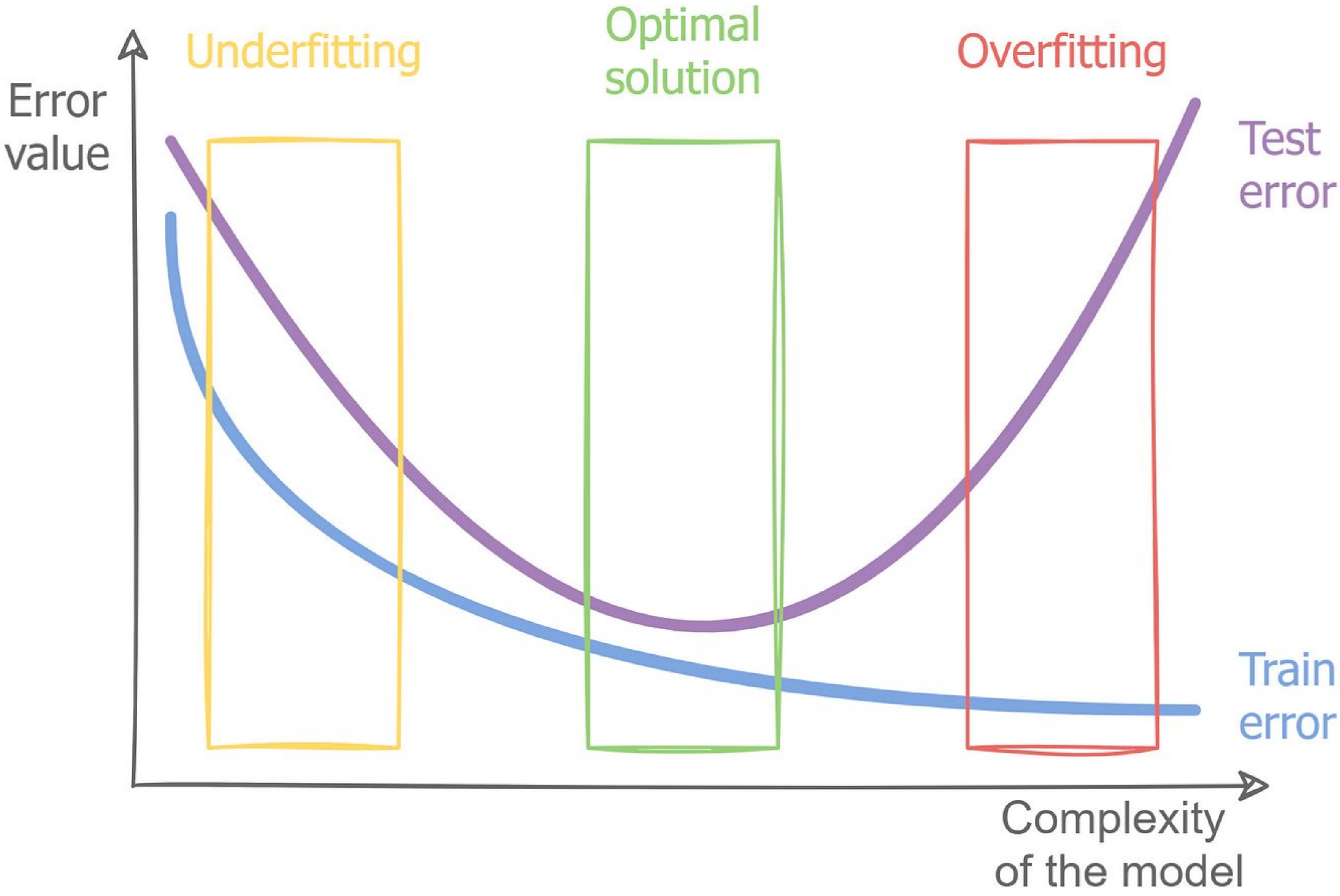
`class Nadam` : Optimizer that implements the Nadam algorithm.

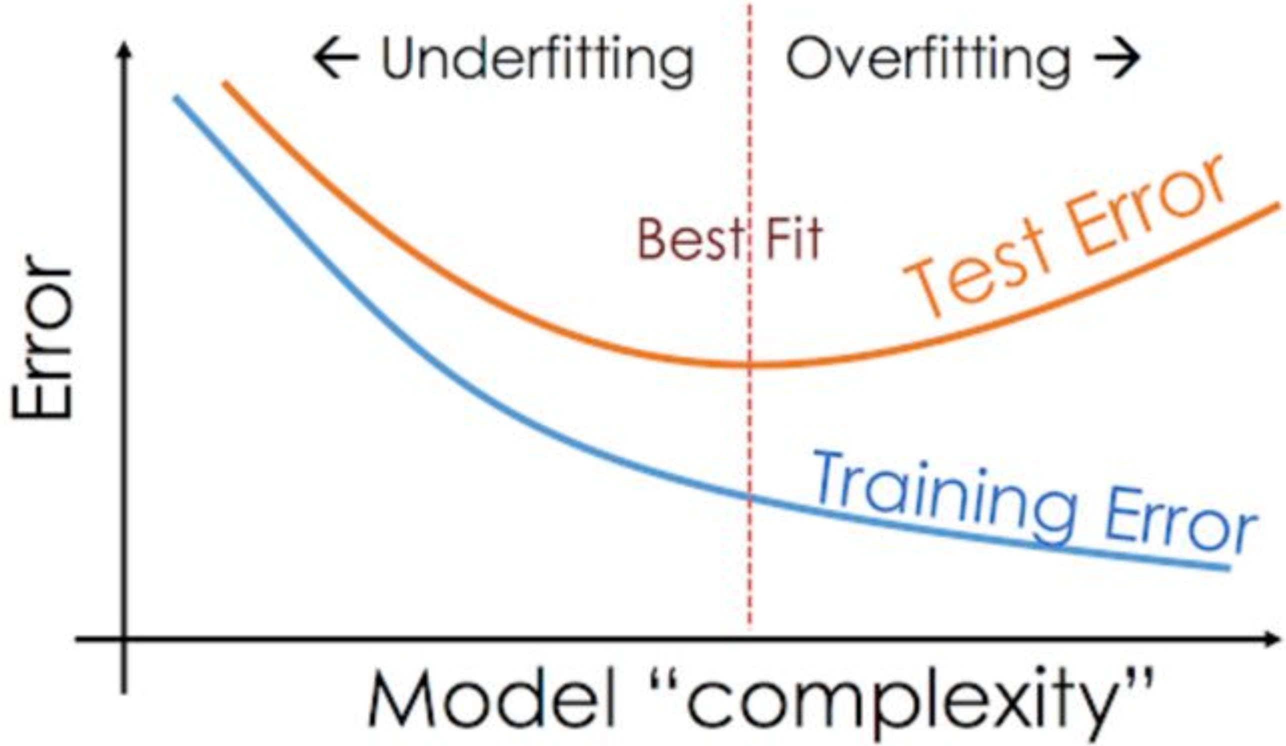
`class Optimizer` : Abstract optimizer base class.

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

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


I WENT FOR A LUNCH BREAK AND WHEN I CAME BACK MY MODEL HAD OVERFIT!
OH NO! WHAT ARE YOU GOING TO DO?
I'M NEVER GOING FOR LUNCH AGAIN.
GENIUS...


Underfitting
Optimal solution
Overfitting
Error value
Test error
Train error
Complexity of the model


← Underfitting
Overfitting →
Best Fit
Test Error
Training Error
Error
Model "complexity"

mango_leaf_train_and_save_weight_2024v3_part2.ipynb

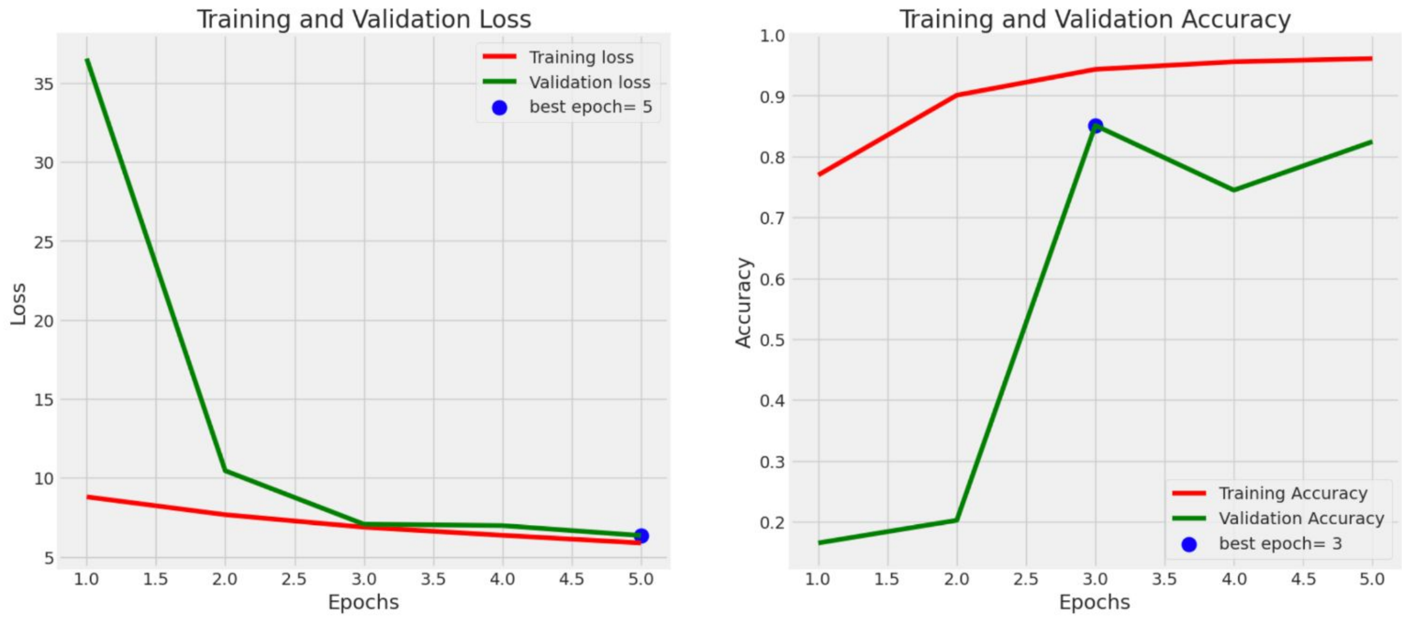
File Edit View Insert Runtime Tools Help All changes saved

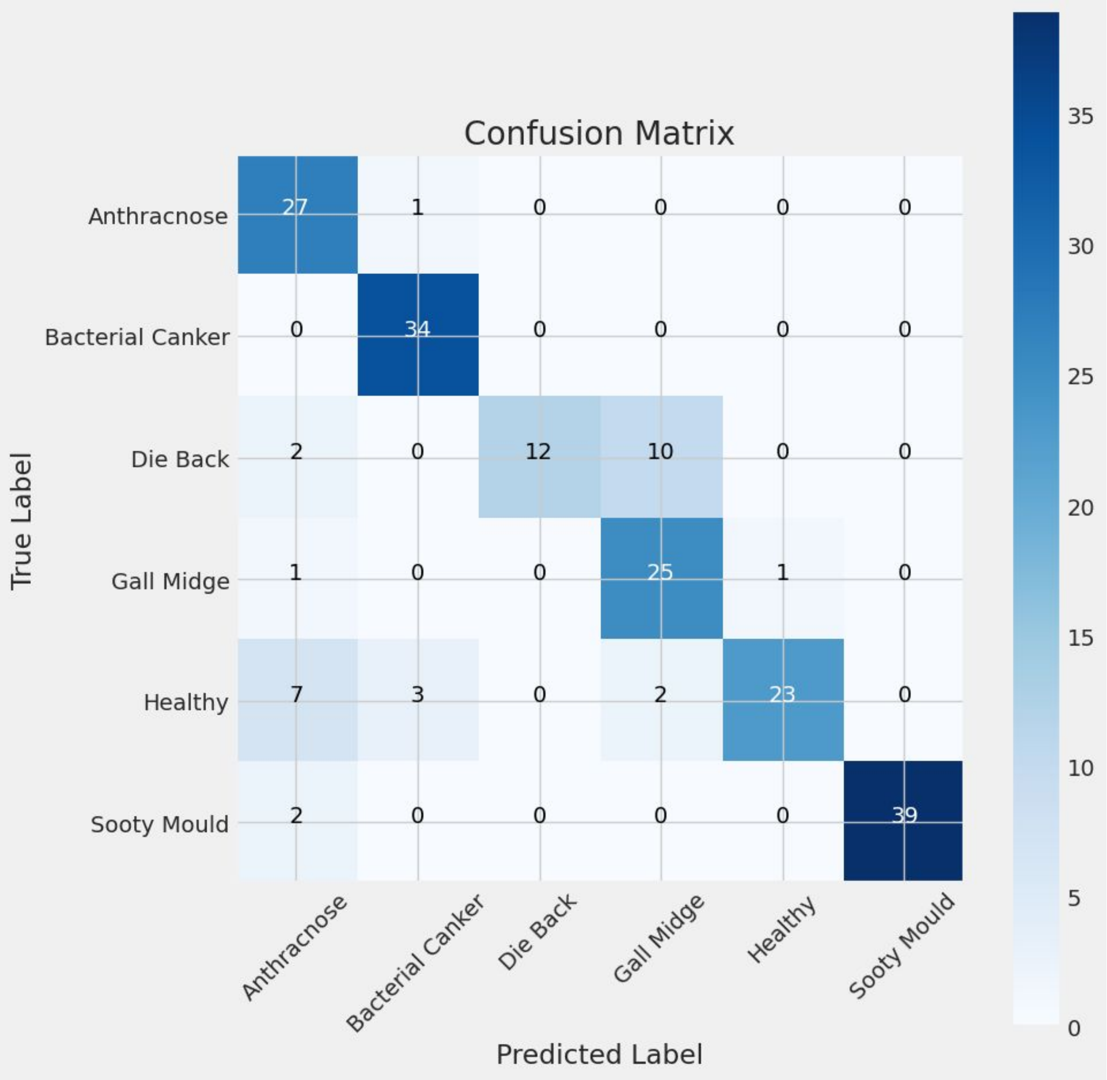
Comment Share

Table of contents

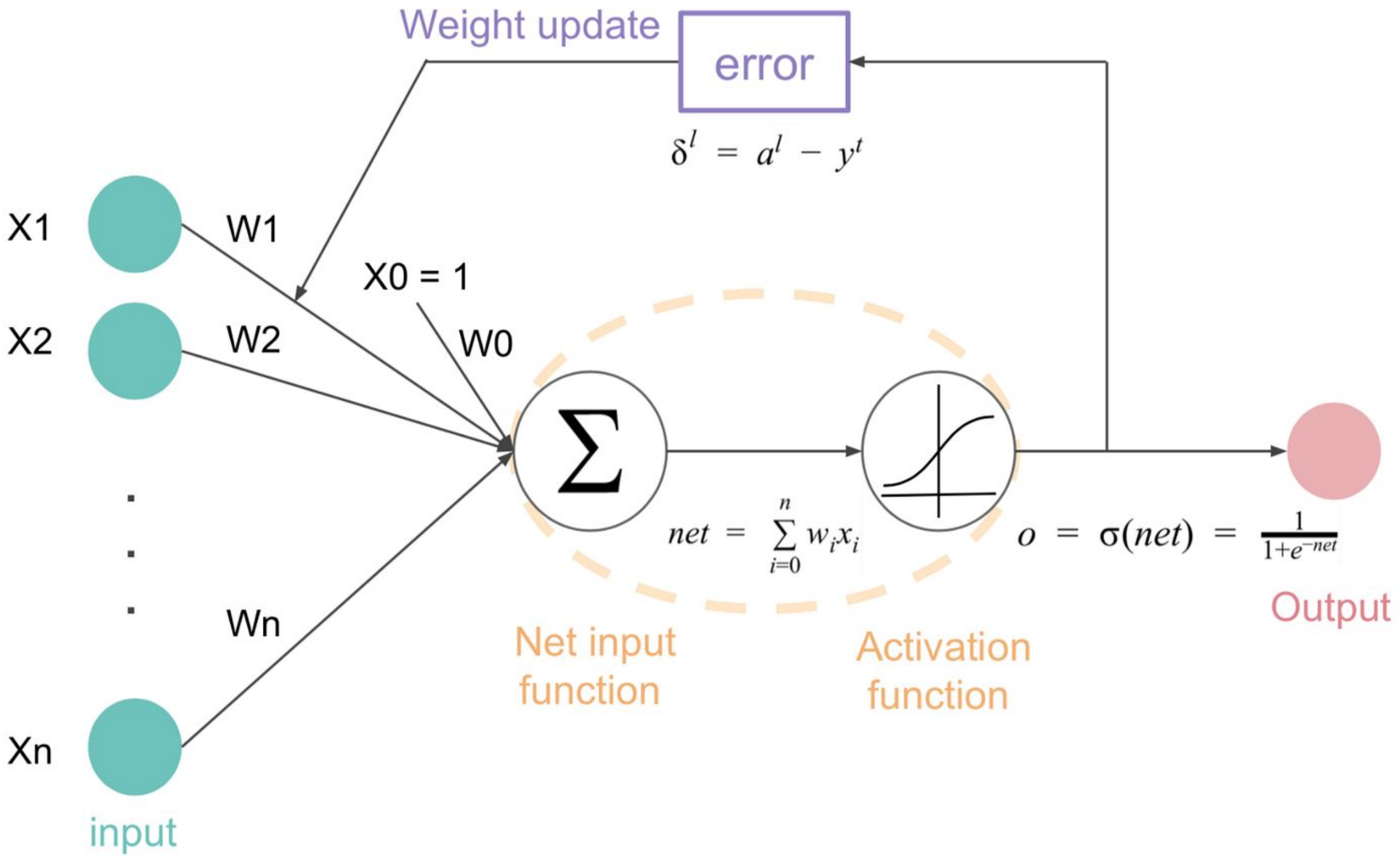
+ Code + Text

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






```
[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
```

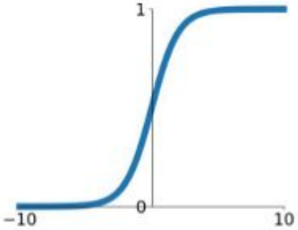


Weight update
error
$\delta^l = a^l - y^t$
X1
W1
X0 = 1
X2
W2
W0
Wn
Xn
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$
Net input function
Activation function
Output
input

Activation Functions

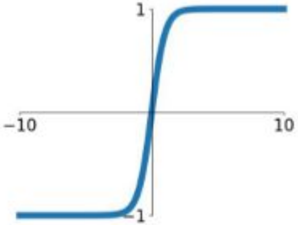
Sigmoid

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



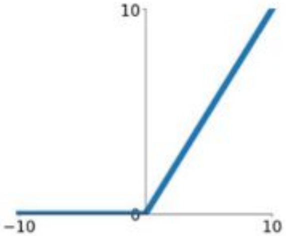
tanh

$\tanh(x)$



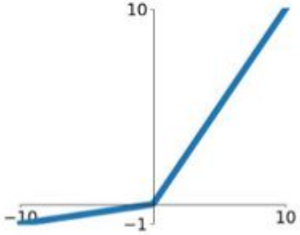
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

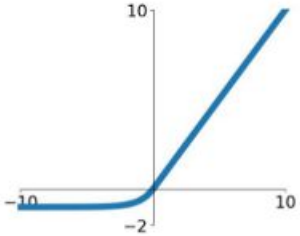


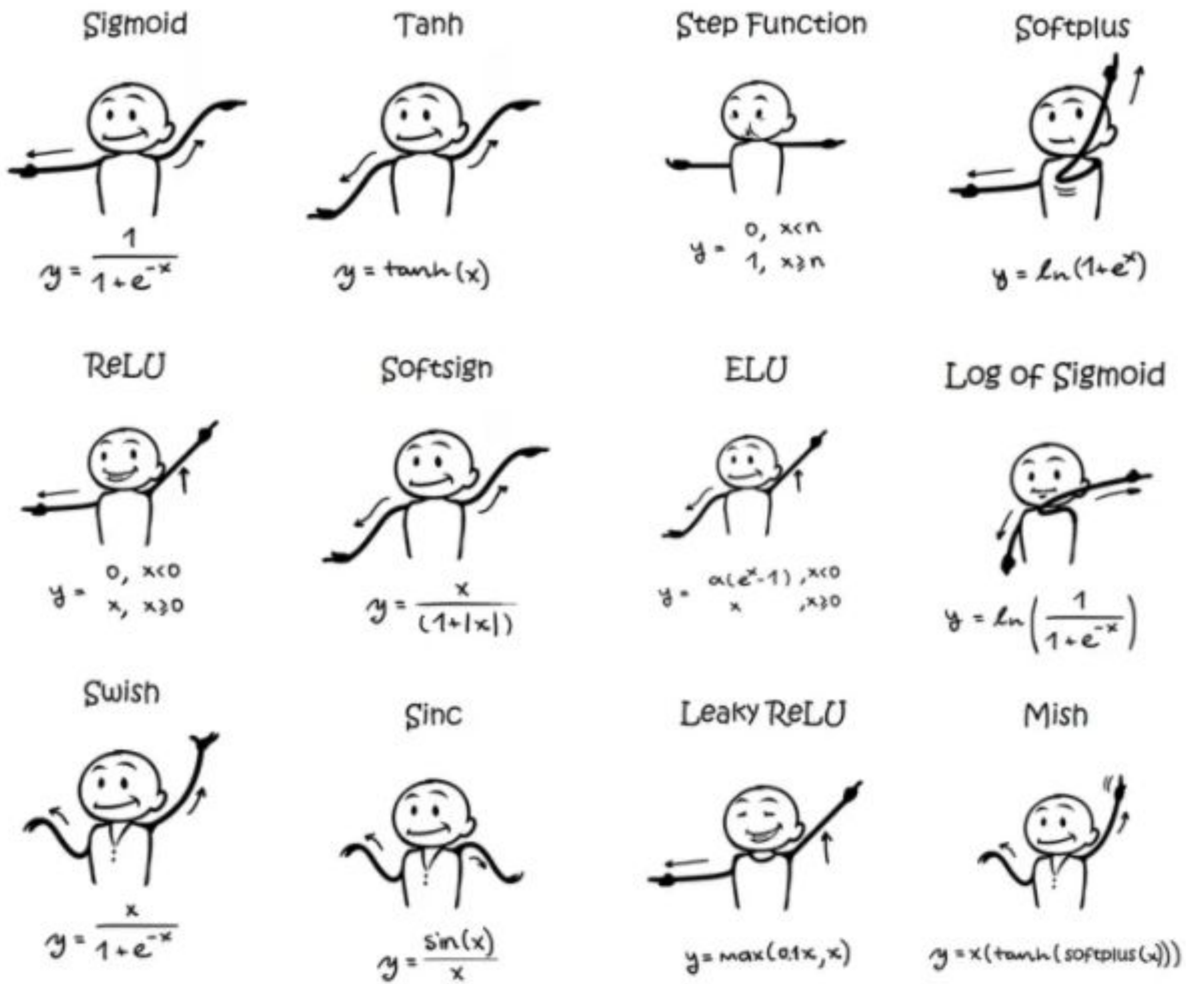
Maxout

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

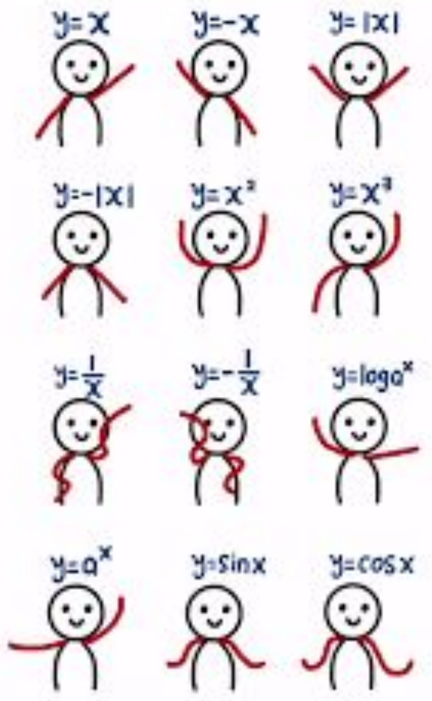
ELU

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

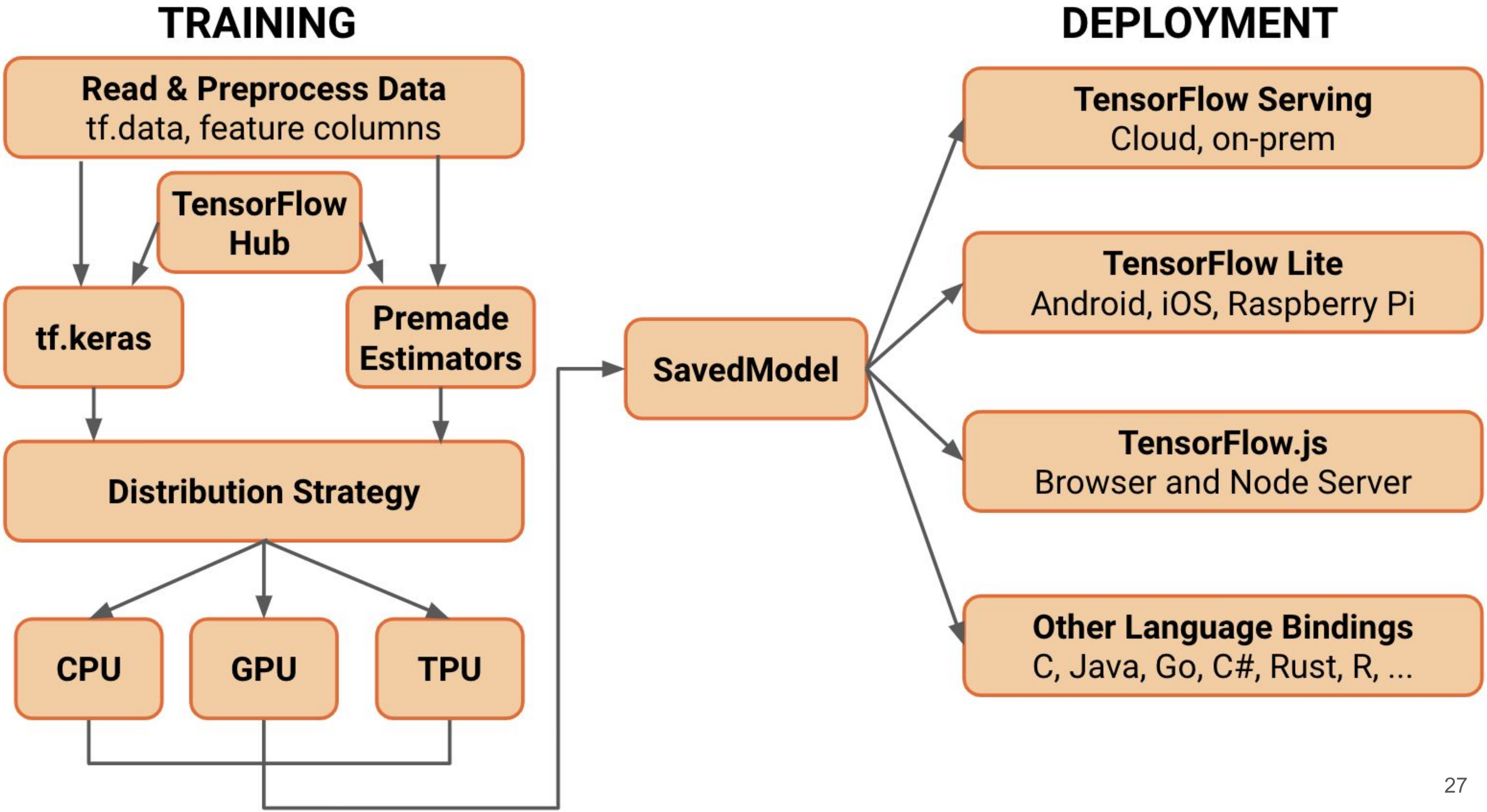




Sigmoid
Tanh
Step Function
Softplus
y = 1/(1+e^-x)
y = tanh(x)
y = 0, x<n 1, x≥n
y = ln(1+e^x)
ReLU
Softsign
ELU
Log of Sigmoid
y = 0, x<0 x, x≥0
y = x/(1+|x|)
y = α(e^x-1), x<0 x, x≥0
y = ln(1/(1+e^-x))
Swish
Sinc
Leaky ReLU
Mish
y = x/(1+e^-x)
y = sin(x)/x
y = max(0.1x, x)
y = x(tanh(softplus(x)))



y=x
y=-x
y=|x|
y=-|x|
y=x²
y=x³
y=1/x
y=-1/x
y=loga^x
y=a^x
y=sinx
y=cosx


TRAINING
DEPLOYMENT
Read & Preprocess Data
tf.data, feature columns
TensorFlow Hub
tf.keras
Premade Estimators
Distribution Strategy
CPU
GPU
TPU
SavedModel
TensorFlow Serving
Cloud, on-prem
TensorFlow Lite
Android, iOS, Raspberry Pi
TensorFlow.js
Browser and Node Server
Other Language Bindings
C, Java, Go, C#, Rust, R, ...

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

```
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 -
```

mango_leaf_train_and_save_weight_2024v3_part2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

..
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]
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
```

```
[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
```

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

[56] 0s

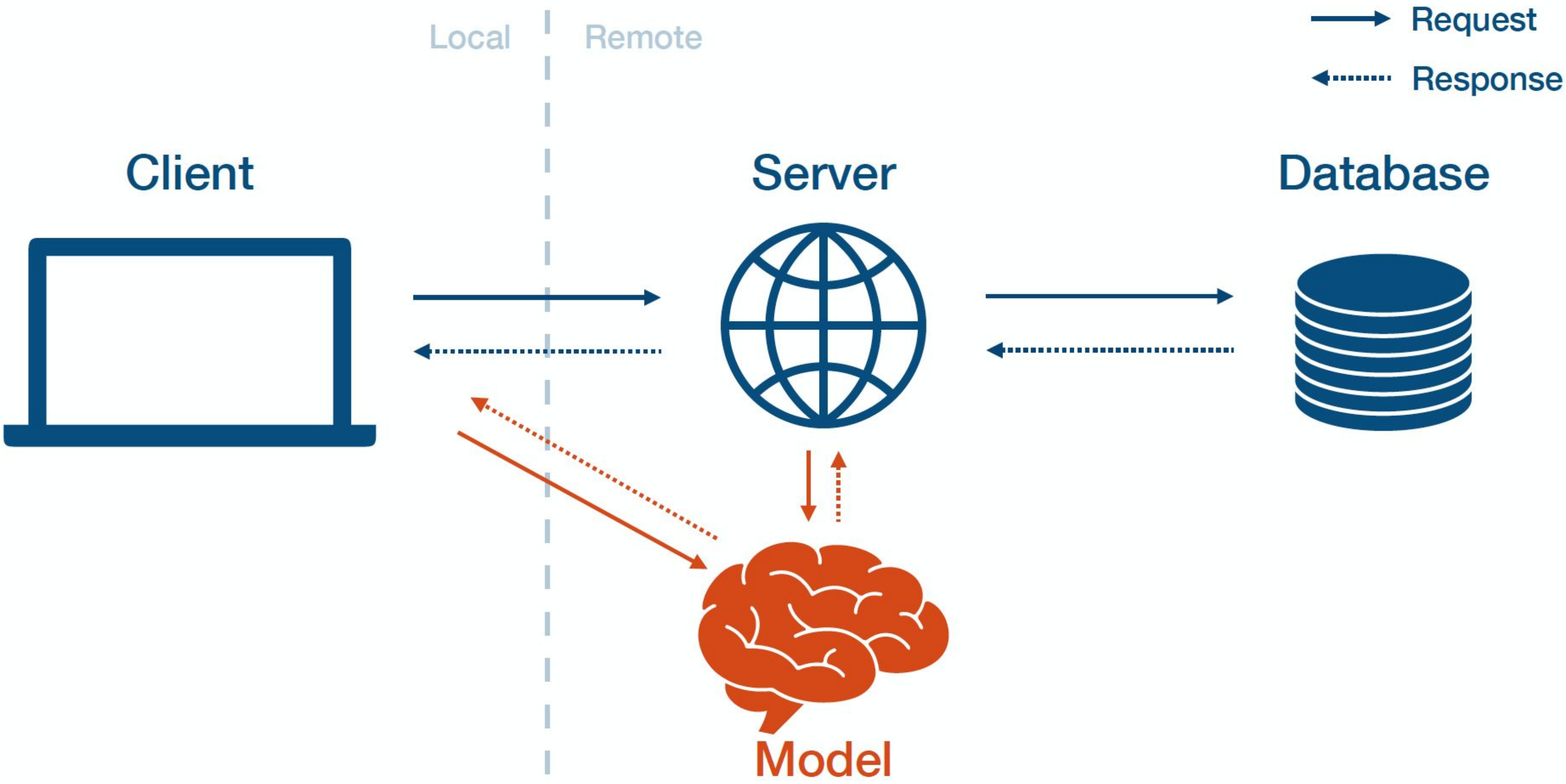
```python
# Get value counts
value_counts = df['labels'].value_counts()

# 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)

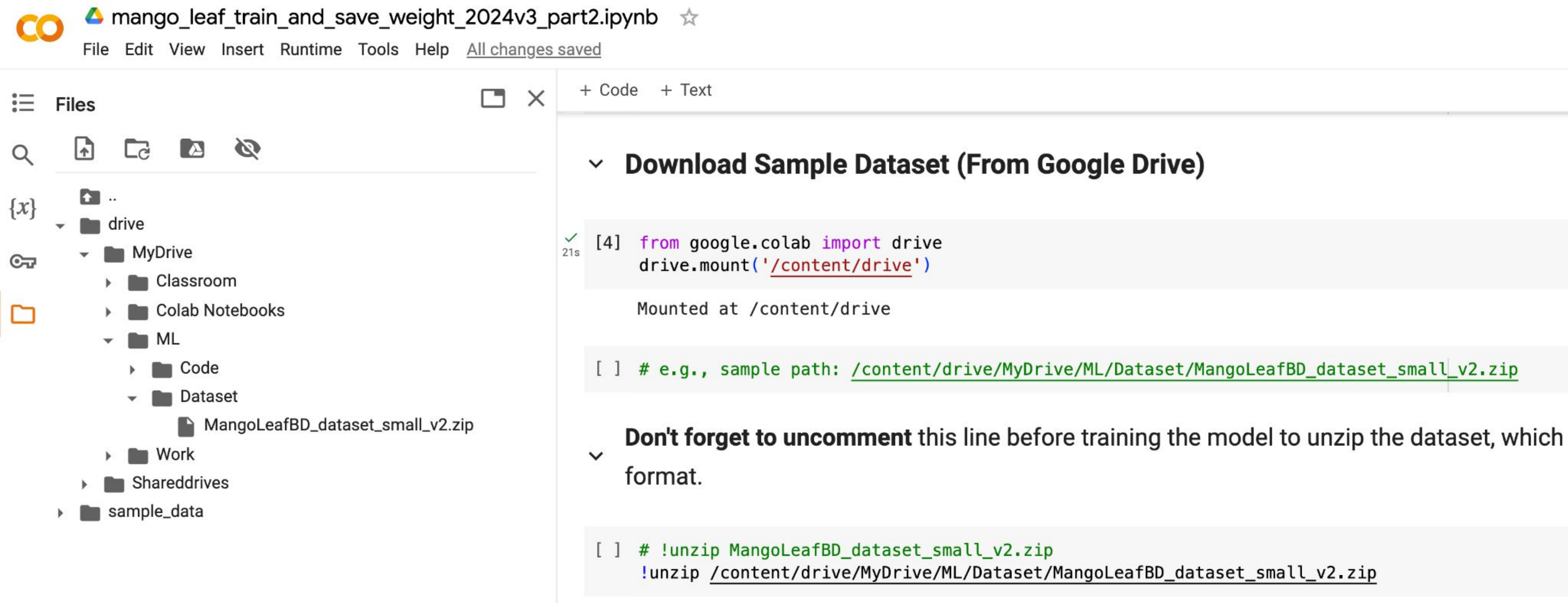
```
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 22ms/step
```

Prediction: Sooty Mould, Confidence: 0.3687029778957367

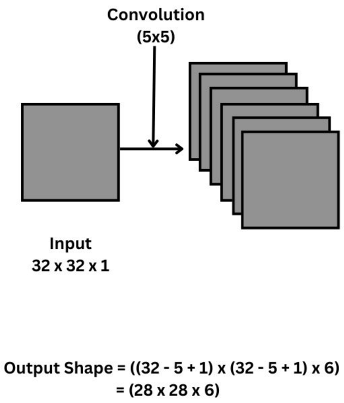


Local
Remote
Request
Response
Client
Server
Database
Model

How to Import Files from Google Drive to Colab



Week 10: Assignment



Build Your Own Deep Learning Architecture

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