Week-1

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets
(train_img, train_labels), (test_img, test_labels)=datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

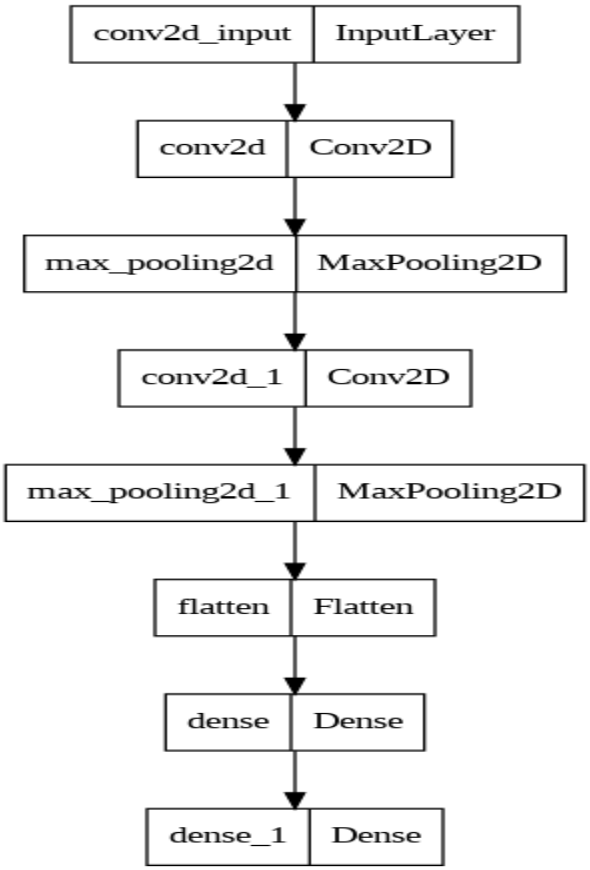
```
train_img, test_img=train_img/255.0, test_img/255.0
class_name=['airplane', 'automobile', 'bird','cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
plt.figure(figsize=(15,15))
for i in range(10):
  plt.subplot(5,5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(train_img[i])
  plt.xlabel(class_name[train_labels[i][0]])
  plt.show()
```


```
from keras.models import Sequential
classifier=Sequential()
from keras.layers import Conv2D
classifier.add(Conv2D (32, (3, 3), input_shape=(32,32,3), activation = 'relu'))
from keras.layers import MaxPooling2D
classifier.add(MaxPooling2D(pool_size = (2,2)))
classifier.add(Conv2D(32, (3,3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))
from keras.layers import Flatten
classifier.add(Flatten())
from keras.layers import Dense
classifier.add(Dense (units =64, activation = 'relu'))
classifier.add(Dense (units=10, activation='softmax'))
```

```
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 32)          0
 g2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 64)                73792

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 84586 (330.41 KB)
Trainable params: 84586 (330.41 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
from tensorflow.keras.utils import plot_model
plot_model(classifier, to_file='cnn_mode.png')
```



```
classifier.compile(optimizer = 'adam',loss=tf.keras.losses.
SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

history=classifier.fit(train_img, train_labels, epochs=10,
validation_data=(test_img, test_labels))
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5729: UserWarning:
  output, from_logits = _get_logits(
1563/1563 [==============================] - 63s 40ms/step - loss: 1.4759 - accu
Epoch 2/10
1563/1563 [==============================] - 54s 34ms/step - loss: 1.1471 - accu
Epoch 3/10
1563/1563 [==============================] - 53s 34ms/step - loss: 1.0245 - accu
Epoch 4/10
1563/1563 [==============================] - 55s 35ms/step - loss: 0.9511 - accu
Epoch 5/10
```

Week-2

```
from tensorflow.keras.models import load_model
from PIL import Image
import numpy as np
image_height = 128
image_width = 128
num_channels = 3
model = load_model('trained_model_NEW_2_2_Dataset.h5')
new_face_path = '/content/hjhhh.jpg'
new_face = Image.open(new_face_path)
display(new_face)
new_face = new_face.resize((image_width, image_height))
new_face = np.array(new_face)
new_face = np.expand_dims(new_face, axis=0)
predictions = model.predict(new_face)
predicted_age_group = np.argmax(predictions)
print("predictions are ",predictions)
print("Predicted Age Group:", predicted_age_group)
age_mapping = {0: 'YOUNG', 1: 'MIDDLE', 2: 'OLD'}
predicted_age_group_label = age_mapping[predicted_age_group]
print("Predicted Age Group:", predicted_age_group_label)
```

Week-3

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import LearningRateScheduler
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
train_images, val_images, train_labels, val_labels =
train_test_split(train_images, train_labels, test_size=0.1, random_state=42)
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
def lr_schedule(epoch):
  initial_lr = 0.001
  if epoch >= 40:
    return initial_lr * 0.1
  return initial_lr
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',metrics=['accuracy'])
datagen = ImageDataGenerator(rotation_range=15,
width_shift_range=0.1,
height_shift_range=0.1,
horizontal_flip=True,
fill_mode='nearest')
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

```
history = model.fit(datagen.flow(train_images, train_labels,
batch_size=64),epochs=50,steps_per_epoch=len(train_images) //
64,validation_data=(val_images, val_labels),
callbacks=[LearningRateScheduler(lr_schedule)])
```

```
Epoch 1/50
 85/703 [==>...........................] - ETA: 43s - loss: 2.2007 - accuracy: 0.0066
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 4s 12ms/step - loss: 0.9610 - accuracy: 0.1351
Test accuracy: 0.13510000705718994
```

Week-4

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from tensorflow.keras.optimizers import SGD
from tensorflow.random import set_seed
set_seed(455)
np.random.seed(455)
dataset = pd.read_csv("/content/Mastercard_stock_history rr.csv",
index_col="Date", parse_dates=["Date"]). drop(["Dividends", "Stock Splits"],
axis=1)
print(dataset.head())
```

Date	Open	High	Low	Close	Volume
2006-05-25	3.748967	4.283869	3.739664	4.279217	395343000
2006-05-26	4.307126	4.348058	4.103398	4.179680	103044000
2006-05-30	4.183400	4.184330	3.986184	4.093164	49898000
2006-05-31	4.125723	4.219679	4.125723	4.180608	30002000
2006-06-01	4.179678	4.474572	4.176887	4.419686	62344000

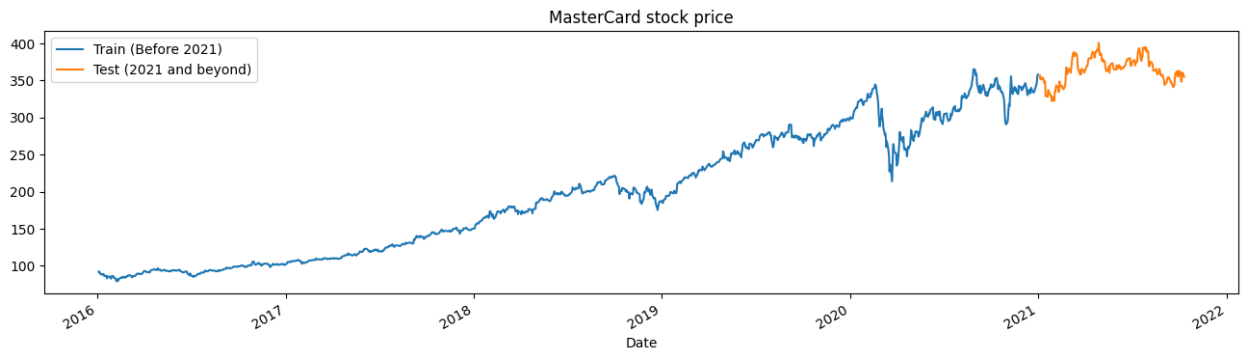
```
print(dataset.describe())
```

	Open	High	Low	Close	Volume
count	3872.000000	3872.000000	3872.000000	3872.000000	3.872000e+03
mean	104.896814	105.956054	103.769349	104.882714	1.232250e+07
std	106.245511	107.303589	105.050064	106.168693	1.759665e+07
min	3.748967	4.102467	3.739664	4.083861	6.411000e+05
25%	22.347203	22.637997	22.034458	22.300391	3.529475e+06
50%	70.810079	71.375896	70.224002	70.856083	5.891750e+06
75%	147.688448	148.645373	146.822013	147.688438	1.319775e+07
max	392.653890	400.521479	389.747812	394.685730	3.953430e+08

```
dataset.isna().sum()
```

```
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

```
tstart = 2016
tend = 2020
def train_test_plot(dataset, tstart, tend):
  dataset.loc[f"{tstart}":f"{tend}", "High"].plot(figsize=(16, 4), legend=True)
  dataset.loc[f"{tend+1}":, "High"].plot(figsize=(16, 4), legend=True)
  plt.legend([f"Train (Before {tend+1})", f"Test ({tend+1} and beyond)"])
  plt.title("MasterCard stock price")
  plt.show()
train_test_plot(dataset,tstart,tend)
```




```
def train_test_split(dataset, tstart, tend):
  train = dataset.loc[f"{tstart}":f"{tend}", "High"].values
  test = dataset.loc[f"{tend+1}":, "High"].values
  return train, test
training_set, test_set = train_test_split(dataset, tstart, tend)
sc = MinMaxScaler(feature_range=(0, 1))
training_set = training_set.reshape(-1, 1)
training_set_scaled = sc.fit_transform(training_set)
def split_sequence(sequence, n_steps):
  X, y = list(), list()
  for i in range(len(sequence)):
    end_ix = i + n_steps
    if end_ix > len(sequence) - 1:
      break
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    y.append(seq_y)
  return np.array(X), np.array(y)
n_steps = 60
```

```
features = 1
X_train, y_train = split_sequence(training_set_scaled, n_steps)
X_train = X_train.reshape(X_train.shape[0],X_train.shape[1],features)
model_lstm = Sequential()
model_lstm.add(LSTM(units=125, activation="tanh", input_shape=(n_steps, features)))
model_lstm.add(Dense(units=1))
model_lstm.compile(optimizer="RMSprop", loss="mse")
model_lstm.summary()
```

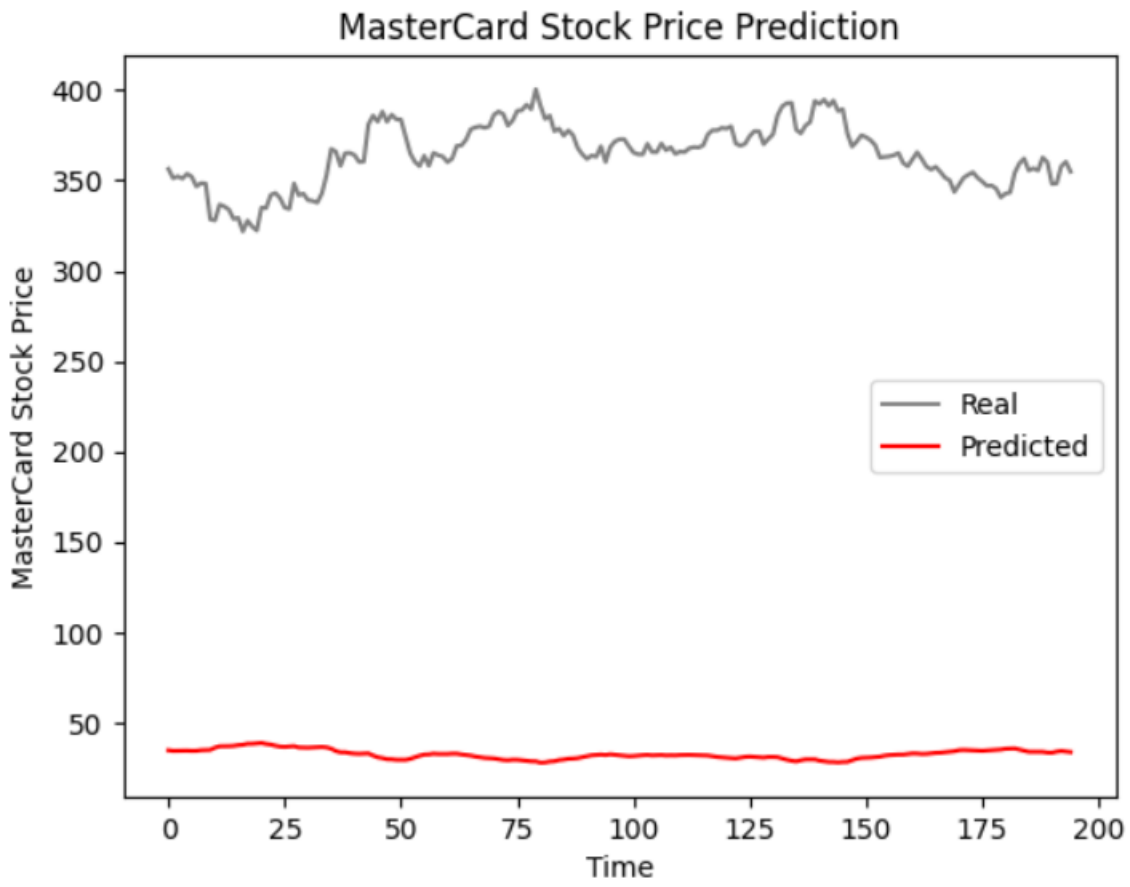
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 125)               63500

 dense (Dense)               (None, 1)                 126

=================================================================
Total params: 63626 (248.54 KB)
Trainable params: 63626 (248.54 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
dataset_total = dataset.loc[:,"High"]
inputs = dataset_total[len(dataset_total) - len(test_set) - n_steps :].values
inputs = inputs.reshape(-1, 1)
inputs = sc.transform(inputs)
X_test, y_test = split_sequence(inputs, n_steps)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], features)
predicted_stock_price = model_lstm.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
7/7 [==============================] - 1s 23ms/step
```

```
def plot_predictions(test, predicted):
  plt.plot(test, color="gray", label="Real")
  plt.plot(predicted, color="red", label="Predicted")
  plt.title("MasterCard Stock Price Prediction")
  plt.xlabel("Time")
  plt.ylabel("MasterCard Stock Price")
  plt.legend()
  plt.show()
```

```
def return_rmse(test, predicted):
  rmse = np.sqrt(mean_squared_error(test, predicted))
  print("The root mean squared error is {:.2f}.".format(rmse))
plot_predictions(test_set,predicted_stock_price)
return_rmse(test_set,predicted_stock_price)
```




```
The root mean squared error is 332.47.
```

Week-5

```
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
bnotes=pd.read_csv("/content/book.csv")
print(bnotes.head())
print(bnotes['Class'].unique())
```

```
   Image.Var  Image.Skew  Image.Curt  Entropy  Class
0    3.62160      8.6661     -2.8073 -0.44699      0
1    4.54590      8.1674     -2.4586 -1.46210      0
2    3.86600     -2.6383      1.9242  0.10645      0
3    3.45660      9.5228     -4.0112 -3.59440      0
4    0.32924     -4.4552      4.5718 -0.98880      0
[0 1]
```

```
bnotes.shape
```

```
(1372, 5)
```

```
bnotes.describe(include='all')
```

	Image.Var	Image.Skew	Image.Curt	Entropy	Class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

```
X=bnotes.drop('Class',axis=1)
y=bnotes['Class']
print(X.head(2))
print(y.head(2))
```

```
   Image.Var  Image.Skew  Image.Curt  Entropy
0     3.6216      8.6661     -2.8073 -0.44699
1     4.5459      8.1674     -2.4586 -1.46210
0    0
1    0
Name: Class, dtype: int64
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
print(X_train.shape)
print(X_test.shape)
```

```
(960, 4)
(412, 4)
```

```
mlp=MLPClassifier(hidden_layer_sizes=(3,2), max_iter=500, activation='relu')
mlp.fit(X_train,y_train)
```

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(3, 2), max_iter=500)
```

```
pred=mlp.predict(X_test)
pred
```

```
array([1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1,
```

```
confusion_matrix(y_test,pred)
```

```
array([[215,   2],
       [  0, 195]])
```

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       217
           1       0.99      1.00      0.99       195

    accuracy                           1.00       412
   macro avg       0.99      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412
```

Week-6

```
!pip install ultralytics -q
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 645.2/645.2 kB 7.6 MB/s eta 0:00:00
```

```
!yolo detect predict model=yolov8m.pt source="/content/INPUT VIDEO.mp4"
```

```
video 1/1 (5/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 947.2ms
video 1/1 (6/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 910.0ms
video 1/1 (7/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 893.1ms
video 1/1 (8/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 960.1ms
video 1/1 (9/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1491.1ms
video 1/1 (10/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1427.0ms
video 1/1 (11/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1384.0ms
video 1/1 (12/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1131.1ms
video 1/1 (13/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 898.8ms
video 1/1 (14/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 922.4ms
video 1/1 (15/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1 truck, 889.2ms
video 1/1 (16/489) /content/INPUT VIDEO.mp4: 384x640 1 person, 1 car, 1 truck, 890.3ms
video 1/1 (17/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1 truck, 892.7ms
video 1/1 (18/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 917.8ms
video 1/1 (19/489) /content/INPUT VIDEO.mp4: 384x640 1 person, 1 car, 1 truck, 916.3ms
video 1/1 (20/489) /content/INPUT VIDEO.mp4: 384x640 1 person, 1 car, 1 truck, 922.2ms
video 1/1 (21/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1 truck, 892.2ms
video 1/1 (22/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1 truck, 892.2ms
video 1/1 (23/489) /content/INPUT VIDEO.mp4: 384x640 1 car, 1396.7ms
video 1/1 (24/489) /content/INPUT VIDEO.mp4: 384x640 2 cars, 1456.6ms
```

```
!ffmpeg -i {"/content/runs/detect/predict/INPUT VIDEO.avi"} -vcodec libx264
{"final.avi"}
```

Week-7

```
import math
def sigmoid_func(x):
  return 1.0/(1+math.exp(-x))
sigmoid_func(100)
sigmoid_func(-100)
sigmoid_func(0)
```

```
0.5
```

```
import pandas as pd
import numpy as np
x = pd.Series(np.arange(-8, 8, 0.5))
y = x.map(sigmoid_func)
print(x)
```

```
0    -8.0
1    -7.5
2    -7.0
3    -6.5
4    -6.0
5    -5.5
6    -5.0
7    -4.5
8    -4.0
9    -3.5
```

```
22    3.0
23    3.5
24    4.0
25    4.5
26    5.0
27    5.5
28    6.0
29    6.5
30    7.0
31    7.5
dtype: float64
```

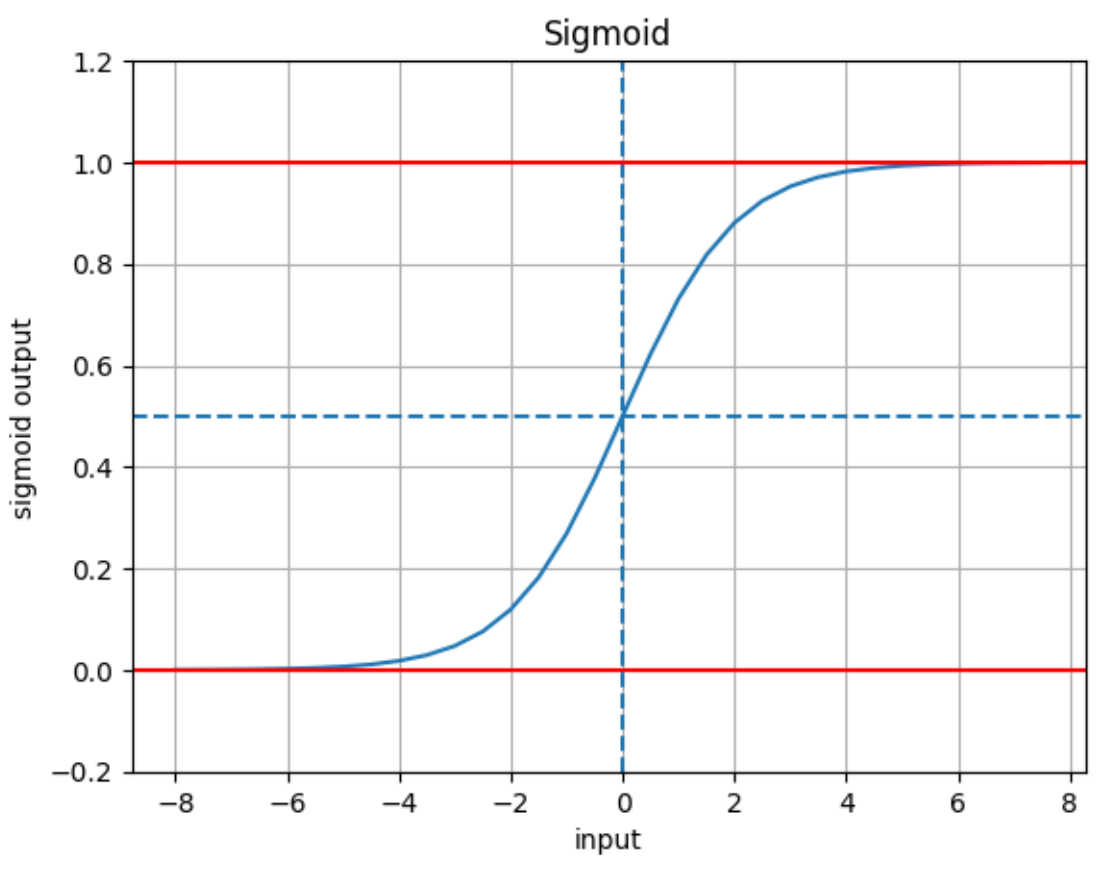
```
print(y)
```

```
0     0.000335
1     0.000553
2     0.000911
3     0.001501
4     0.002473
5     0.004070
6     0.006693
7     0.010987
8     0.017986
9     0.029312
```

```
28    0.997527
29    0.998499
30    0.999089
31    0.999447
dtype: float64
```

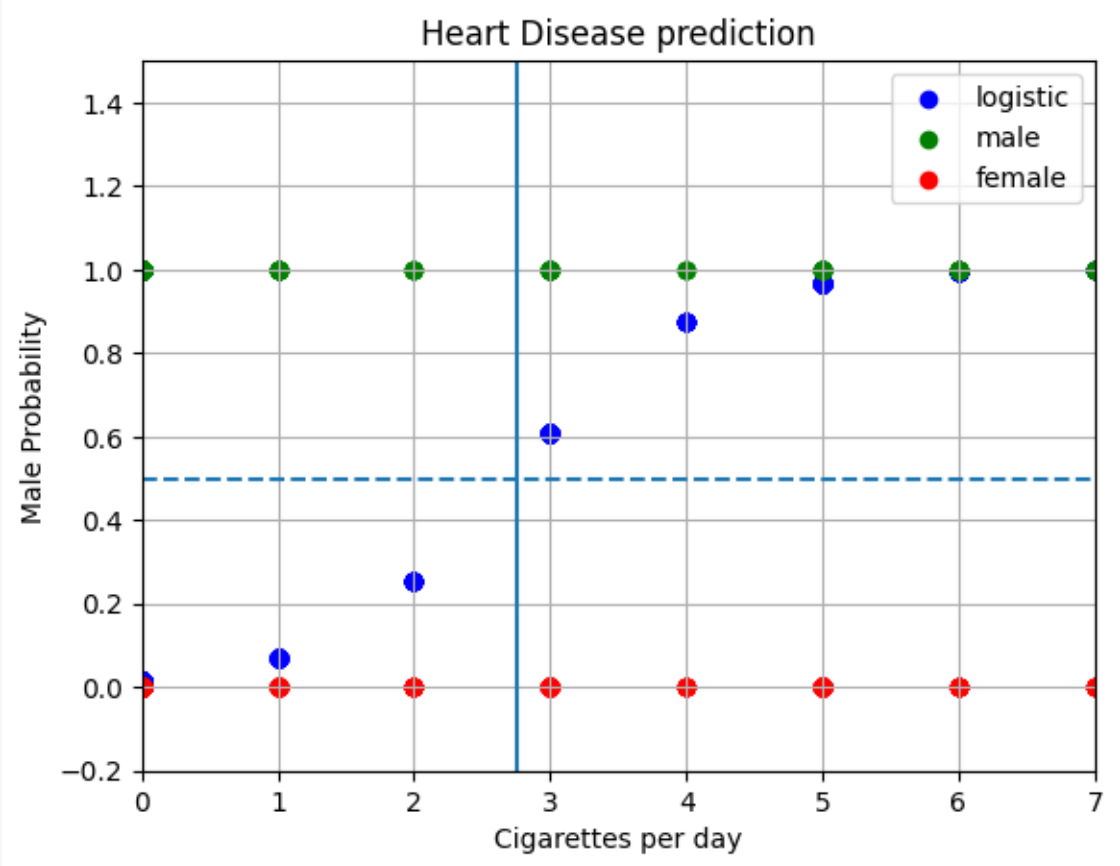
```
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.ylim(-0.2, 1.2)
plt.xlabel("input")
plt.ylabel("sigmoid output")
plt.grid(True)
plt.axvline(x=0, ymin=0, ymax=1, ls='dashed')
plt.axhline(y=0.5, xmin=0, xmax=10, ls='dashed')
plt.axhline(y=1.0, xmin=0, xmax=10, color='r')
plt.axhline(y=0.0, xmin=0, xmax=10, color='r')
plt.title("Sigmoid")
```

```
Text(0.5, 1.0, 'Sigmoid')
```




```
import pandas as pd
df = pd.read_csv('/content/framingham_heart_disease.csv')
def straight_line(x):
  return 1.5046*x - 4.0777
```

```
def straight_line_weight(weight, x):
  return weight*x - 4.0777
y_vals = df.cigsPerDay.map(straight_line).map(sigmoid_func)
import matplotlib.pyplot as plt
plt.scatter(x=df.cigsPerDay, y=y_vals, color='b', label='logistic')
plt.scatter(x=df[df.male==1].cigsPerDay, y=df[df.male==1].male, color='g',
label='male')
plt.scatter(x=df[df.male==0].cigsPerDay, y=df[df.male==0].male, color='r',
label='female')
plt.title("Heart Disease prediction")
plt.xlabel("Cigarettes per day")
plt.ylabel("Male Probability")
plt.grid(True)
plt.legend()
plt.xlim((0, 7))
plt.ylim((-0.2, 1.5))
plt.axvline(x=2.75, ymin=0, ymax=1)
plt.axhline(y=0.5, xmin=0, xmax=6, label="cutoff at 0.5", ls='dashed')
```



Week-8

```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.14.0'
```

```
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten,\
 Conv2D, MaxPooling2D, BatchNormalization
model = Sequential()
model.add(Conv2D (filters=96, input_shape=(227,227,3), kernel_size=(11,11), \
 strides=(4,4), padding= 'valid'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides = (2,2), padding='valid'))
model.add(BatchNormalization())
model.add(Conv2D (filters=256, kernel_size=(11,11), strides=(1,1),
padding='valid'))
model.add(Activation ('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D (filters=384, kernel_size=(3,3), strides=(1,1),
padding='valid'))
model.add(BatchNormalization())
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'))
model.add(Activation ('relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='valid'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense (4096, input_shape=(224*224*3,)))
model.add(Activation('relu'))
model.add(Dropout (0.4))
model.add(BatchNormalization())
```

```
model.add(Dense(4096))
model.add(Activation ('relu'))
model.add(Dropout (0.4))
model.add(Dense(1000))
model.add(Activation ('relu'))
model.add(Dropout (0.4))
model.add(BatchNormalization())
model.add(Dense(17))
model.add(Activation ('softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)            (None, 55, 55, 96)        34944

 activation_8 (Activation)    (None, 55, 55, 96)        0

 max_pooling2d_5 (MaxPoolin   (None, 27, 27, 96)        0
 g2D)

 batch_normalization_8 (Bat   (None, 27, 27, 96)        384
 chNormalization)

 conv2d_8 (Conv2D)            (None, 17, 17, 256)       2973952

 activation_9 (Activation)    (None, 17, 17, 256)       0

 max_pooling2d_6 (MaxPoolin   (None, 8, 8, 256)         0
 g2D)

 batch_normalization_9 (Bat   (None, 8, 8, 256)         1024
 chNormalization)

 activation_10 (Activation)   (None, 8, 8, 256)         0

 conv2d_9 (Conv2D)            (None, 6, 6, 384)         885120

 batch_normalization_10 (Ba   (None, 6, 6, 384)         1536
 tchNormalization)

 conv2d_10 (Conv2D)           (None, 4, 4, 384)         1327488

 activation_11 (Activation)   (None, 4, 4, 384)         0

 batch_normalization_11 (Ba   (None, 4, 4, 384)         1536
 tchNormalization)

 conv2d_11 (Conv2D)           (None, 2, 2, 256)         884992
```

activation_12 (Activation)	(None, 2, 2, 256)	0
max_pooling2d_7 (MaxPooling2D)	(None, 1, 1, 256)	0
batch_normalization_12 (BatchNormalization)	(None, 1, 1, 256)	1024
flatten_1 (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 4096)	1052672
activation_13 (Activation)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0
batch_normalization_13 (BatchNormalization)	(None, 4096)	16384
dense_2 (Dense)	(None, 4096)	16781312
activation_14 (Activation)	(None, 4096)	0
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 1000)	4097000
activation_15 (Activation)	(None, 1000)	0
dropout_3 (Dropout)	(None, 1000)	0
batch_normalization_14 (BatchNormalization)	(None, 1000)	4000
dense_4 (Dense)	(None, 17)	17017
activation_16 (Activation)	(None, 17)	0

```
=================================================================
Total params: 28080385 (107.12 MB)
Trainable params: 28067441 (107.07 MB)
Non-trainable params: 12944 (50.56 KB)
_________________________________________________________________
```

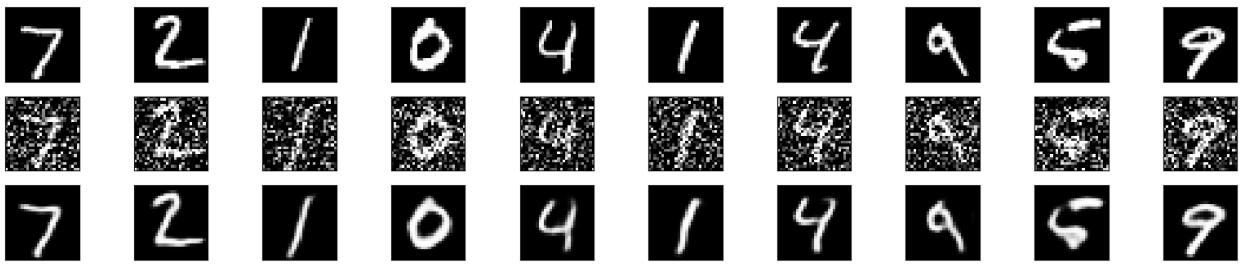
Week-9

```
pip install tensorflow

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0

noise_factor = 0.5

x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_train.shape)

x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)

x_test_noisy = np.clip(x_test_noisy, 0., 1.)

input_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)

x = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(encoded)

x = UpSampling2D((2, 2))(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train_noisy.reshape(-1, 28, 28, 1),

x_train.reshape(-1, 28, 28, 1),

epochs=10,

batch_size=128,

shuffle=True,

validation_data=(x_test_noisy.reshape(-1, 28, 28, 1), x_test.reshape(-1, 28, 28,
1)))
```

```
denoised_images = autoencoder.predict(x_test_noisy.reshape(-1, 28, 28, 1))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/10
469/469 [==============================] - 178s 377ms/step - loss: 0.1578 - val_loss: 0.1120
Epoch 2/10
469/469 [==============================] - 161s 344ms/step - loss: 0.1081 - val_loss: 0.1043
Epoch 3/10
469/469 [==============================] - 163s 347ms/step - loss: 0.1032 - val_loss: 0.1008
Epoch 4/10
469/469 [==============================] - 160s 341ms/step - loss: 0.1008 - val_loss: 0.0991
Epoch 5/10
469/469 [==============================] - 160s 341ms/step - loss: 0.0992 - val_loss: 0.0979
```

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  ax = plt.subplot(3, n, i + 1)
  plt.imshow(x_test[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  ax = plt.subplot(3, n, i + 1 + n)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  ax = plt.subplot(3, n, i + 1 + 2 * n)
  plt.imshow(denoised_images[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```



Week-10

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import torch
import torch.nn as nn
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
n = 1000
first_column = torch.rand(n, 1).to(device)
second_column = 2 * first_column
third_column = 2 * second_column
data = torch.cat([first_column, second_column, third_column], dim=1)
class Generator(nn.Module):
  def __init__(self):
    super(Generator, self).__init__()
    self.model = nn.Sequential( nn.Linear(3, 50),nn.ReLU(),nn.Linear(50, 3))
  def forward(self, x):
    return self.model(x)
class Discriminator(nn.Module):
  def __init__(self):
    super(Discriminator, self).__init__()
    self.model = nn.Sequential(nn.Linear(3, 50),nn.ReLU(),nn.Linear(50, 1),
nn.Sigmoid() )
  def forward(self, x):
    return self.model(x)
generator = Generator().to(device)
discriminator = Discriminator().to(device)
criterion = nn.BCELoss()
optimizer_g = torch.optim.Adam(generator.parameters(), lr=0.001)
optimizer_d = torch.optim.Adam(discriminator.parameters(), lr=0.001)
num_epochs = 5000
for epoch in range(num_epochs):
  optimizer_d.zero_grad()
  real_data = data
```

```
  real_labels = torch.ones(n, 1).to(device)
  outputs = discriminator(real_data)
  d_loss_real = criterion(outputs, real_labels)
  noise = torch.randn(n, 3).to(device)
  fake_data = generator(noise)
  fake_labels = torch.zeros(n, 1).to(device)
  outputs = discriminator(fake_data.detach())
  d_loss_fake = criterion(outputs, fake_labels)
  d_loss = d_loss_real + d_loss_fake
  d_loss.backward()
  optimizer_d.step()
  optimizer_g.zero_grad()
  outputs = discriminator(fake_data)
  g_loss = criterion(outputs, real_labels)
  g_loss.backward()
  optimizer_g.step()
  if (epoch+1) % 1000 == 0:
    print(f"Epoch [{epoch+1}/{num_epochs}], d_loss: {d_loss.item():.4f}, g_loss:
{g_loss.item():.4f}")
```

```
Epoch [1000/5000], d_loss: 1.3697, g_loss: 0.7050
Epoch [2000/5000], d_loss: 1.3790, g_loss: 0.6994
Epoch [3000/5000], d_loss: 1.3824, g_loss: 0.6903
Epoch [4000/5000], d_loss: 1.3830, g_loss: 0.6995
Epoch [5000/5000], d_loss: 1.3826, g_loss: 0.7056
```

```
with torch.no_grad():
  test_noise = torch.randn(n, 3).to(device)
  generated_data = generator(test_noise).cpu().numpy()
print("Generated Data (First 10 rows):")
for i in range(10):
  print(generated_data[i])
```

```
Generated Data (First 10 rows):
[0.8868288 1.834546  3.6621556]
[0.85063785 1.7277099  3.460173  ]
[0.25114644 0.53021014 1.0508499 ]
[0.45784512 0.939868   1.880346  ]
[0.8890573 1.8542204 3.6935523]
```

```python
print("\nValidation (For the first 10 rows):")
for i in range(10):
  print(f"First: {generated_data[i][0]:.4f}, Expected Second: {2*generated_data[i][0]:.4f}, Actual Second: {generated_data[i][1]:.4f}")
  print(f"Second: {generated_data[i][1]:.4f}, Expected Third: {2*generated_data[i][1]:.4f}, Actual Third: {generated_data[i][2]:.4f}\n")
```

```
Validation (For the first 10 rows):
First: 0.8868, Expected Second: 1.7737, Actual Second: 1.8345
Second: 1.8345, Expected Third: 3.6691, Actual Third: 3.6622

First: 0.8506, Expected Second: 1.7013, Actual Second: 1.7277
Second: 1.7277, Expected Third: 3.4554, Actual Third: 3.4602

First: 0.2511, Expected Second: 0.5023, Actual Second: 0.5302
Second: 0.5302, Expected Third: 1.0604, Actual Third: 1.0508

First: 0.4578, Expected Second: 0.9157, Actual Second: 0.9399
Second: 0.9399, Expected Third: 1.8797, Actual Third: 1.8803

First: 0.8891, Expected Second: 1.7781, Actual Second: 1.8542
Second: 1.8542, Expected Third: 3.7084, Actual Third: 3.6936

First: 0.8630, Expected Second: 1.7261, Actual Second: 1.7652
Second: 1.7652, Expected Third: 3.5303, Actual Third: 3.5208

First: 0.7149, Expected Second: 1.4298, Actual Second: 1.4700
```

LINK TO DOWNLOAD THE DATASETS OF WEEK-4,5,6,7

https://github.com/Prahashit4/deep-learning-lab/