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()
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()
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))
```

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

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())
print(dataset.describe())
dataset.isna().sum()
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()
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)
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)
```

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())
bnotes.shape
bnotes.describe(include='all')
X=bnotes.drop('Class',axis=1)
y=bnotes['Class']
print(X.head(2))
print(y.head(2))
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)
mlp=MLPClassifier(hidden_layer_sizes=(3,2), max_iter=500, activation='relu')
mlp.fit(X_train,y_train)
pred=mlp.predict(X_test)
pred
confusion_matrix(y_test,pred)
print(classification_report(y_test,pred))
```

Week-6

```
!pip install ultralytics -q

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

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

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)))
```

```python
denoised_images = autoencoder.predict(x_test_noisy.reshape(-1, 28, 28, 1))
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}")
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])
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")
```