```
**********MLP(MULTI LAYER PERCEPTION)**********
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# normalize image pixel values by dividing
# by 255
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale
print("Feature matrix:", x_train.shape)
print("Target matrix:", x_train.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_train.shape)
fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
    for j in range(10):
        ax[i] [j].imshow(x_train[k].reshape(28, 28),aspect='auto')
        k +=1
plt.show()
model = Sequential([
    # reshape data to 28*28 rows
    Flatten(input_shape=(28, 28)),
      # dense layer 1
    Dense(256, activation='sigmoid'),
    # dense layer 2
    Dense(128, activation='sigmoid'),
      # output layer
    Dense(10, activation='sigmoid'),])
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=2000, validation_split=0.2)
results = model.evaluate(x_test, y_test, verbose=0)
print('test loss, test acc:', results)

**********CNN(CONVOLUTIONAL NEURAL NETWORK)**********
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
#normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
```

```
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
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.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
 metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=2, validation_data=(test_images, test_labels))
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(test_acc)

*****************************RNN(RECURRENT NEURAL NETWORK)************************
import numpy as np
import tensorflow_datasets as tfds
import tensorflow as tf

dataset, info = tfds.load('imdb_reviews',with_info =True,as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']

train_dataset.element_spec
for example, label in train_dataset.take(1):
  print('text: ',example.numpy())
  print('label: ',label.numpy())
  BUFFER_SIZE =10000
  BATCH_SIZE = 64
  train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

  test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
  VOCAB_SIZE = 1000
  encoder = tf.keras.layers.TextVectorization(max_tokens=VOCAB_SIZE)
  encoder.adapt(train_dataset.map(lambda text, label: text))
  model = tf.keras.Sequential([encoder,
                               tf.keras.layers.Embedding(
                                   input_dim=len(encoder.get_vocabulary()),
                                   output_dim=64,

                                   mask_zero=True),
                               tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
                               tf.keras.layers.Dense(64, activation='relu'),
                               tf.keras.layers.Dense(1)
                               ])

  sample_text = ('The movie was cool. The animation and the graphics were out of the world. I would rcom
mend this movie')
  prediction = model.predict(np.array([sample_text]))
  print(prediction[0])
```

```
  print("$$$$$$$$$$$")
  model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
  history = model.fit(train_dataset, epochs=2,
                    validation_data=test_dataset,
                    validation_steps=30)
  test_loss, test_acc = model.evaluate(test_dataset)
  print('Test Loss:',test_loss)
  print('Test Accuaracy:', test_acc)

*************************VISUALIZING A DEEP LEARNING MODEL********************
import tensorflow as tf
import visualkeras

from tensorflow import keras

from keras.models import Sequential

from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.models import Model

model = Sequential()

model.add(Conv2D(64, (4,4),input_shape=(32,32,3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(128,(4,4), input_shape=(32,32,3),activation='relu',padding='same' ))
model.add(MaxPooling2D(pool_size=(2,2)))

model.summary()
visualkeras.layered_view(model)

*************************LOAD AND SAVE A MODEL************************************
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize image pixel values by dividing by 255
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale
```

```
# Define the Sequential model
model = Sequential([
  # Reshape data to 28*28 rows
  Flatten(input_shape=(28, 28)),
  # Dense layer 1
  Dense(256, activation='sigmoid'),
  # Dense layer 2
  Dense(128, activation='sigmoid'),
  # Output layer
  Dense(10, activation='sigmoid'),
])

# Print the model summary
model.summary()

# Saving and loading the .h5 model

# Save the model
model.save_weights('MLPWeights.h5')
print('Model Saved!')

# Load the saved model
savedModel = model.load_weights('MLPWeights.h5')
print('Model Loaded!')

**************************PLOT CURVE FIGURE************************
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from keras import models
from keras import layers
from keras import optimizers


bc = datasets.load_breast_cancer()
x = bc.data
y = bc.target


network = models.Sequential()
network.add(layers.Dense(32,activation='relu', input_shape=(30,)))
network.add(layers.Dense(32,activation='relu'))
network.add(layers.Dense(1,activation='sigmoid'))


network.compile(optimizer=optimizers.RMSprop(lr=0.01),
              loss='binary_crossentropy',
              metrics=['accuracy'])

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3 ,stratify=y, random_state=42)

history = network.fit(x_train,y_train,
                    validation_data=(x_test, y_test),
                    epochs=10,
```

```
                batch_size=20)


history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
accuracy = history_dict['accuracy']

val_accuracy = history_dict['val_accuracy']

epochs = range(1, len(loss_values) +1)
fig, ax = plt.subplots(1,2, figsize=(14,6))



ax[0].plot(epochs, accuracy, 'bo', label='Training accuracy')
ax[0].plot(epochs, val_accuracy, 'b', label='validation accuracy')
ax[0].set_title('Training & validation Accuracy', fontsize=16)
ax[0].set_xlabel('Epochs', fontsize=16)
ax[0].set_ylabel('Accuarcy', fontsize=16)
ax[0].legend()

ax[1].plot(epochs, loss_values, 'bo', label='Training loss')
ax[1].plot(epochs, val_loss_values, 'b', label='validation accuracy')
ax[1].set_title('Training & validation Loss', fontsize=16)
ax[1].set_xlabel('Epochs', fontsize=16)
ax[1].set_ylabel('Loss', fontsize=16)
ax[1].legend()

**********Dropout Early FIgure**********
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop
from keras.callbacks import EarlyStopping
from keras.datasets import mnist
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Load and preprocess the data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000, 784).astype('float32') / 255
X_test = X_test.reshape(10000, 784).astype('float32') / 255

y_train = keras.utils.to_categorical(y_train, num_classes=10)
y_test = keras.utils.to_categorical(y_test, num_classes=10)

# Define the model architecture
model = Sequential([
    Dense(512, activation='relu', input_shape=(784,)),
    Dropout(0.2),
    Dense(512, activation='relu'),
```

```
    Dropout(0.2),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Define early stopping monitor
early_stopping_monitor = EarlyStopping(patience=3)

# Train the model
history = model.fit(X_train, y_train, batch_size=128, epochs=20,
                    callbacks=[early_stopping_monitor], verbose=1, validation_data=(X_test, y_test))

# Evaluate the model
score = model.evaluate(X_test, y_test, verbose=0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])

**************************IMAGE CLASSIFICATION********************
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow. keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras. applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input, decode_predictions
import numpy as np

#ResNet
model_resnet = ResNet50(weights='imagenet')
img_path ='orange.JPG'
img = image.load_img(img_path, target_size= (224, 224))
x = image.img_to_array(img)
x = np.expand_dims (x, axis=0)
x = preprocess_input(x)
#model. summary ()
preds_resnet = model_resnet.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('***ResNe†****')
print('Predicted:', decode_predictions (preds_resnet, top=3)[0])

model_vgg = VGG16 (weights='imagenet')

img_path = 'orange.JPG'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array (img)
x = np.expand_dims (x, axis=0)
x = preprocess_input (x)
#model.summary ()
preds_vgg = model_vgg.predict(x)
print('****VGG****')
print('Predicted:', decode_predictions (preds_vgg, top=3)[0])
```

```
model_inception = InceptionV3 (weights='imagenet')

img_path = 'orange.JPG'
img = image. load_img(img_path, target_size=(299, 299))
x = image.img_to_array(img)
x = np.expand_dims (x, axis=0)
x = preprocess_input (x)

preds_inception = model_inception.predict(x)
print('***INCEPTION****')
print('Predicted:', decode_predictions (preds_inception, top=3)[0])

**************************BATCH NORMALIZATION************************
import tensorflow as tf
from tensorflow import keras
(train_images, train_labels), (test_images, test_labels) = keras.datasets.fashion_mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
validation_images = train_images[:5000]
validation_labels = train_labels[:5000]
# Placing batch normailization layer before the activation layers
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(300, use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation(keras.activations.relu),
    keras.layers.Dense(200, use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation(keras.activations.relu),
    keras.layers.Dense(100, use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation(keras.activations.relu),
    keras.layers.Dense(10, activation=keras.activations.softmax)
])
model.layers[2].variables
for variable in model.layers[2].variables:
    print(variable.name)
sgd = tf.keras.optimizers.legacy.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss="sparse_categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])
model.fit(train_images, train_labels, epochs=10, validation_data=(validation_images, validation_labels))
model.evaluate(test_images, test_labels)
```