UCS2612 Machine Learning Laboratory

Assignment 2 : Handwritten Character Recognition using Neural Networks

Name : **Mega V**

Roll No : **3122 21 5001 051**

Aim

To Develop a python program to recognize handwritten characters using Neural Network (NN) Model.

Code

Importing the Necessary Libraries For Single layer perceptron model , Multiple Layer Perceptron Model and CNN Using Keras

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
from PIL import Image
import cv2

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

from sklearn.linear_model import Perceptron

from sklearn.neural_network import MLPClassifier

import tensorflow as tf
```

Loading the dataset and Pre-Processing the data (Image Enhancement techniques)

1) **Image resize**
2) **Normalization of image frequencies**

```
data_dir = "/content/drive/MyDrive/Image_dataset/Img"
x=[]
```

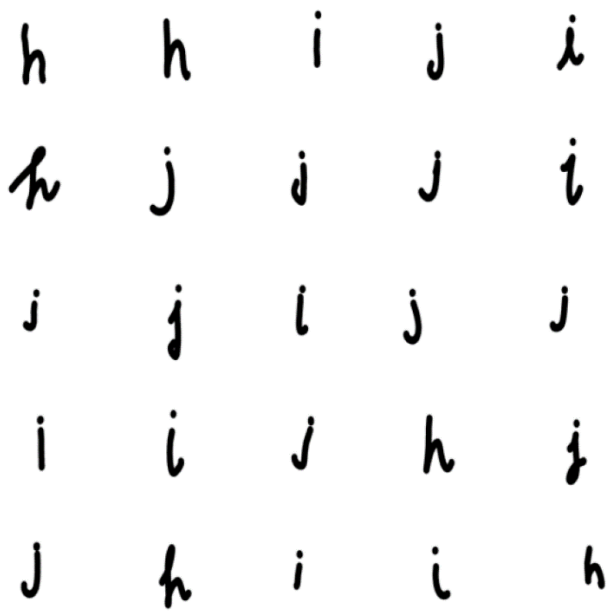
```
x1=[]
x2=[]
for image in os.listdir(data_dir):
    image_path = os.path.join(data_dir, image)
    img = Image.open(image_path).resize((100, 100))
    img_array = np.array(img)
    img_array1 = np.array(img).flatten() / 255.0
    img_array2 = np.array(img).flatten() / 255.0
    x.append(img_array)
    x1.append(img_array1)
    x2.append(img_array2)
x = np.array(x)
x1 = np.array(x1)
x2 = np.array(x1)
```

Printing the Sample Images

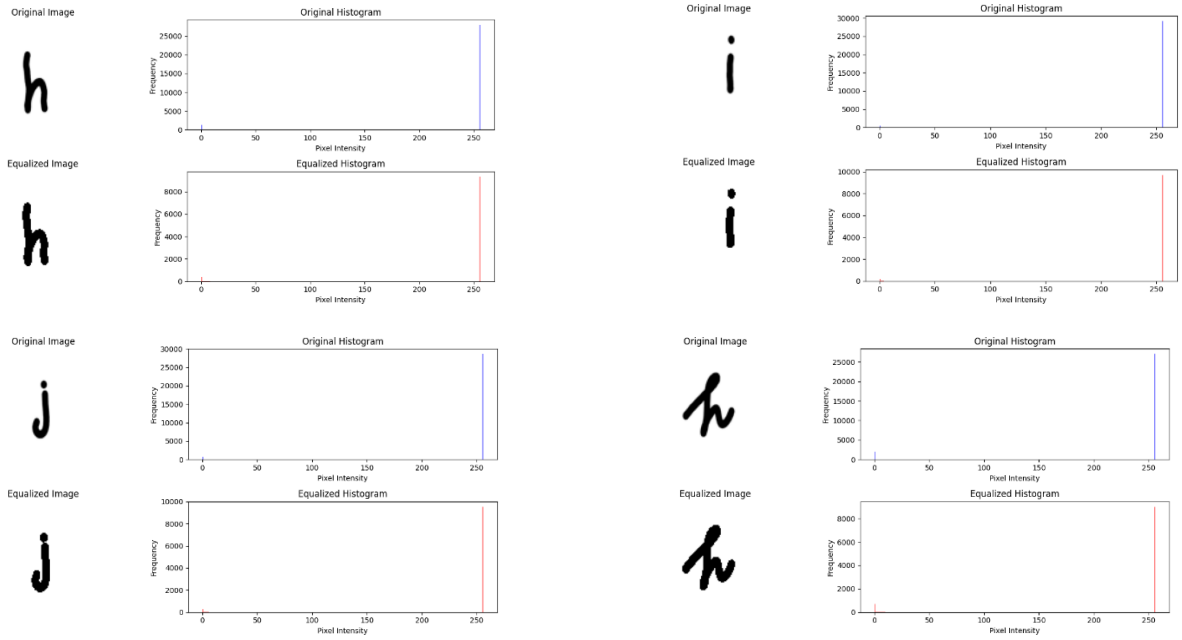
```
print("the no of Images in dataset : ",len(x1))
```

```
the no of Images in dataset :  3410
```

```
plt.figure(figsize=(10, 10))
for i in range(min(25, len(x))):
    plt.subplot(5, 5, i + 1)
    plt.imshow(x[i], cmap='gray')
    plt.axis('off')
plt.show()
```



Exploratory Data Analysis. Histogram For The Images



Split the data into training, testing and validation sets

```
y=pd.read_csv("/content/drive/MyDrive/english.csv")
len(y)
```

```
y
```

	image	label
0	Img/img001-001.png	0
1	Img/img001-002.png	0
2	Img/img001-003.png	0
3	Img/img001-004.png	0
4	Img/img001-005.png	0
...	...	...
3405	Img/img062-051.png	z
3406	Img/img062-052.png	z
3407	Img/img062-053.png	z
3408	Img/img062-054.png	z
3409	Img/img062-055.png	z

3410 rows × 2 columns

CNN Model

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

print("x_train Dataset shape:", x_train.shape)
print("y_train shape:", len(y_train))
print("x_test shape:", x_test.shape)
print("y_test shape:", len(y_test))
```

```
x_train Dataset shape: (2387, 100, 100, 3)
y_train shape: 2387
x_test shape: (1023, 100, 100, 3)
y_test shape: 1023
```

```
y_train_array = y_train.to_numpy()
y_test_array = y_test.to_numpy()

y_train_indices = np.argmax(y_train_array, axis=1)
y_test_indices = np.argmax(y_test_array, axis=1)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_indices)
y_test_encoded = label_encoder.transform(y_test_indices)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(100, 100, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(62, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train_encoded, epochs=20,
validation_data=(x_test, y_test_encoded))
```

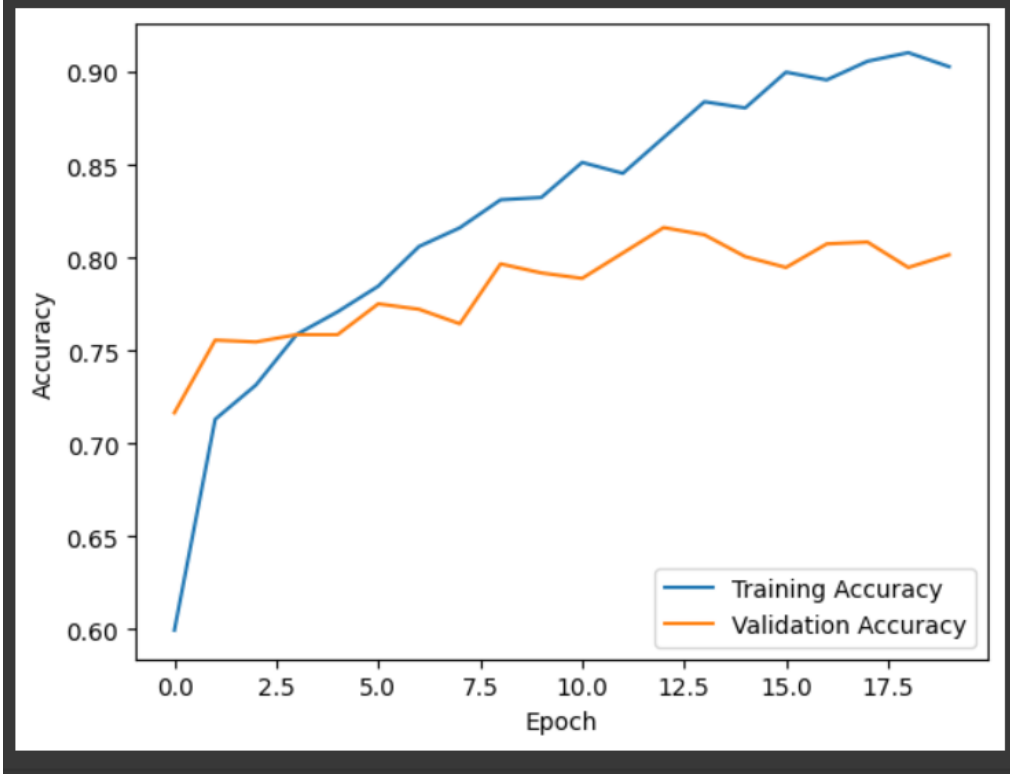
```
Epoch 1/20
75/75 [==============================] - 57s 694ms/step - loss: 8.5602 - accuracy: 0.5995 - val_loss: 0.6032 - val_accuracy: 0.7165
Epoch 2/20
75/75 [==============================] - 51s 685ms/step - loss: 0.6668 - accuracy: 0.7130 - val_loss: 0.5317 - val_accuracy: 0.7556
Epoch 3/20
75/75 [==============================] - 56s 751ms/step - loss: 0.5911 - accuracy: 0.7315 - val_loss: 0.5056 - val_accuracy: 0.7546
Epoch 4/20
75/75 [==============================] - 55s 736ms/step - loss: 0.5474 - accuracy: 0.7587 - val_loss: 0.4952 - val_accuracy: 0.7586
Epoch 5/20
75/75 [==============================] - 55s 732ms/step - loss: 0.4936 - accuracy: 0.7708 - val_loss: 0.4866 - val_accuracy: 0.7586
Epoch 6/20
75/75 [==============================] - 49s 655ms/step - loss: 0.4637 - accuracy: 0.7847 - val_loss: 0.4831 - val_accuracy: 0.7752
Epoch 7/20
75/75 [==============================] - 53s 704ms/step - loss: 0.4327 - accuracy: 0.8060 - val_loss: 0.4847 - val_accuracy: 0.7722
Epoch 8/20
75/75 [==============================] - 50s 666ms/step - loss: 0.4214 - accuracy: 0.8161 - val_loss: 0.4927 - val_accuracy: 0.7644
Epoch 9/20
75/75 [==============================] - 55s 738ms/step - loss: 0.3982 - accuracy: 0.8312 - val_loss: 0.4561 - val_accuracy: 0.7967
Epoch 10/20
75/75 [==============================] - 51s 682ms/step - loss: 0.3730 - accuracy: 0.8324 - val_loss: 0.4688 - val_accuracy: 0.7918
Epoch 11/20
75/75 [==============================] - 51s 688ms/step - loss: 0.3446 - accuracy: 0.8513 - val_loss: 0.4841 - val_accuracy: 0.7889
Epoch 12/20
75/75 [==============================] - 58s 767ms/step - loss: 0.3526 - accuracy: 0.8454 - val_loss: 0.4950 - val_accuracy: 0.8025
```

```
Epoch 13/20
75/75 [==============================] - 55s 733ms/step - loss: 0.3123 - accuracy: 0.8647 - val_loss: 0.4905 - val_accuracy: 0.8162
Epoch 14/20
75/75 [==============================] - 50s 669ms/step - loss: 0.2884 - accuracy: 0.8840 - val_loss: 0.5008 - val_accuracy: 0.8123
Epoch 15/20
75/75 [==============================] - 54s 711ms/step - loss: 0.2808 - accuracy: 0.8806 - val_loss: 0.5250 - val_accuracy: 0.8006
Epoch 16/20
75/75 [==============================] - 58s 772ms/step - loss: 0.2619 - accuracy: 0.8999 - val_loss: 0.5377 - val_accuracy: 0.7947
Epoch 17/20
75/75 [==============================] - 51s 682ms/step - loss: 0.2682 - accuracy: 0.8957 - val_loss: 0.5187 - val_accuracy: 0.8074
Epoch 18/20
75/75 [==============================] - 51s 676ms/step - loss: 0.2222 - accuracy: 0.9057 - val_loss: 0.6579 - val_accuracy: 0.8084
Epoch 19/20
75/75 [==============================] - 65s 873ms/step - loss: 0.2188 - accuracy: 0.9103 - val_loss: 0.5547 - val_accuracy: 0.7947
Epoch 20/20
75/75 [==============================] - 51s 674ms/step - loss: 0.2390 - accuracy: 0.9028 - val_loss: 0.5875 - val_accuracy: 0.8016
```

```
test_loss, test_accuracy = model.evaluate(x_test, y_test_encoded)
print(f"testing Accuracy of Neural Network model using keras :
{test_accuracy}\n\n\n\n")

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```
32/32 [==============================] - 11s 329ms/step - loss: 0.5875 - accuracy: 0.8016
testing Accuracy of Neural Network model using keras : 0.8015640377998352
```



Neural network model Using Single Layer perceptron model

```
x_train, x_test, y_train, y_test = train_test_split(x1, y,
test_size=0.3, random_state=42)

y_train_array = y_train.to_numpy()
y_test_array = y_test.to_numpy()
y_train_indices = np.argmax(y_train_array, axis=1)
y_test_indices = np.argmax(y_test_array, axis=1)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_indices)
y_test_encoded = label_encoder.transform(y_test_indices)
```

```
perceptron_model = Perceptron(max_iter=20, eta0=0.01)

train_accuracies = []
val_accuracies = []

for epoch in range(20):
```

```
    perceptron_model.partial_fit(x_train, y_train_encoded,
classes=np.unique(y_train_encoded))

    y_train_pred = perceptron_model.predict(x_train)

    train_accuracy = accuracy_score(y_train_encoded, y_train_pred)
    train_accuracies.append(train_accuracy)

    y_val_pred = perceptron_model.predict(x_test)

    val_accuracy = accuracy_score(y_test_encoded, y_val_pred)
    val_accuracies.append(val_accuracy)

    print(f"Epoch {epoch+1}/100 - Training Accuracy:
{train_accuracy:.4f}, Validation Accuracy: {val_accuracy:.4f}")

y_test_pred = perceptron_model.predict(x_test)
```

```
Epoch 1/100 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 2/100 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 3/100 - Training Accuracy: 0.7570, Validation Accuracy: 0.7214
Epoch 4/100 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 5/100 - Training Accuracy: 0.7713, Validation Accuracy: 0.7243
Epoch 6/100 - Training Accuracy: 0.7838, Validation Accuracy: 0.7341
Epoch 7/100 - Training Accuracy: 0.7059, Validation Accuracy: 0.6716
Epoch 8/100 - Training Accuracy: 0.7910, Validation Accuracy: 0.7390
Epoch 9/100 - Training Accuracy: 0.7968, Validation Accuracy: 0.7283
Epoch 10/100 - Training Accuracy: 0.7507, Validation Accuracy: 0.7097
Epoch 11/100 - Training Accuracy: 0.7922, Validation Accuracy: 0.7263
Epoch 12/100 - Training Accuracy: 0.7105, Validation Accuracy: 0.6755
Epoch 13/100 - Training Accuracy: 0.7926, Validation Accuracy: 0.7341
Epoch 14/100 - Training Accuracy: 0.7868, Validation Accuracy: 0.7234
Epoch 15/100 - Training Accuracy: 0.7872, Validation Accuracy: 0.7263
Epoch 16/100 - Training Accuracy: 0.8027, Validation Accuracy: 0.7283
Epoch 17/100 - Training Accuracy: 0.7863, Validation Accuracy: 0.7204
Epoch 18/100 - Training Accuracy: 0.7905, Validation Accuracy: 0.7155
Epoch 19/100 - Training Accuracy: 0.7947, Validation Accuracy: 0.7224
Epoch 20/100 - Training Accuracy: 0.7914, Validation Accuracy: 0.7214
```

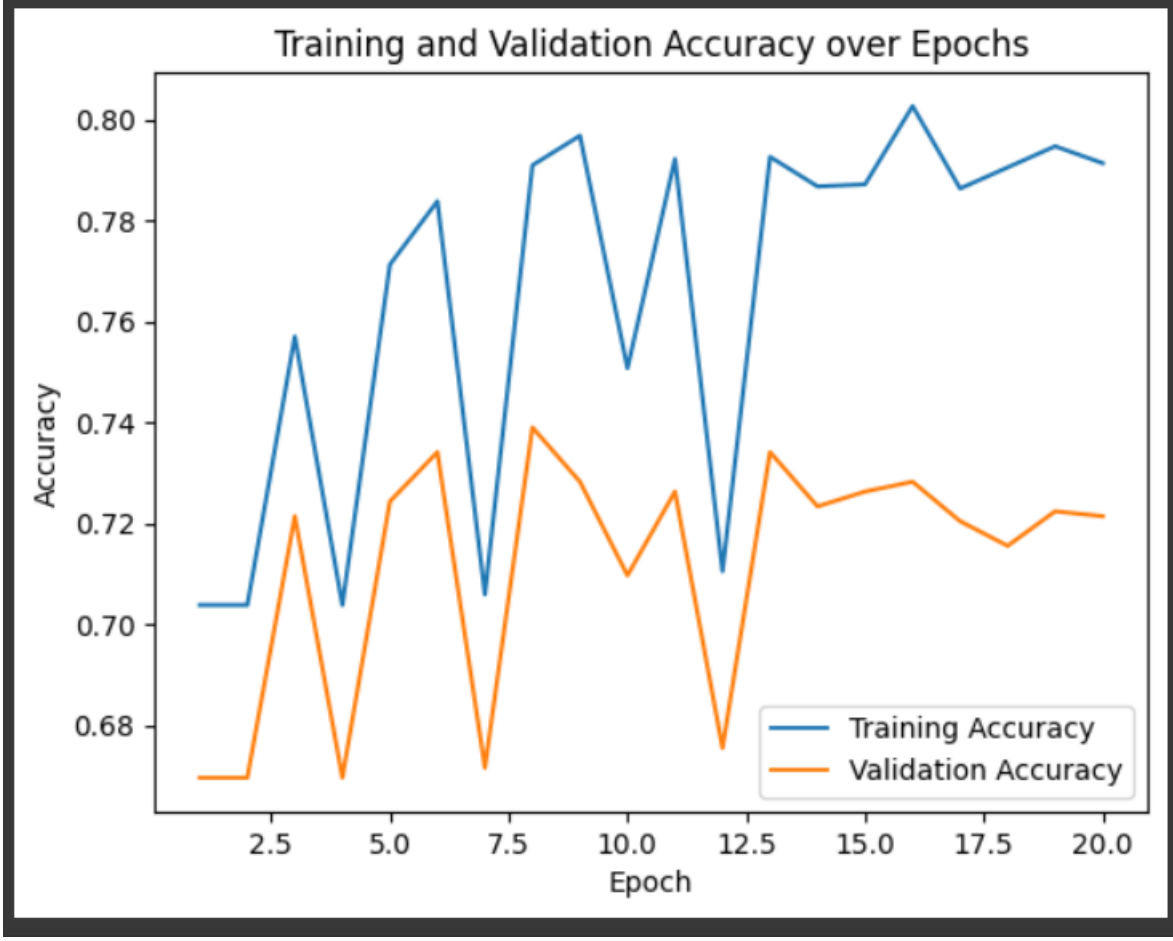
```
test_accuracy = accuracy_score(y_test_encoded, y_test_pred)

print(f"\n\n\nTesting Accuracy fo Single Layer perceptron model :
{test_accuracy}\n\n\n")

plt.plot(range(1, 21), train_accuracies, label='Training Accuracy')
plt.plot(range(1, 21), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
plt.title('Training and Validation Accuracy over Epochs')
plt.show()
```

```
Testing Accuracy fo Single Layer perceptron model : 0.7214076246334311
```



Neural network model Using multiple Layer perceptron model

```
x_train, x_test, y_train, y_test = train_test_split(x2, y,
test_size=0.3, random_state=42)

y_train_array = y_train.to_numpy()
y_test_array = y_test.to_numpy()
y_train_indices = np.argmax(y_train_array, axis=1)
y_test_indices = np.argmax(y_test_array, axis=1)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_indices)
y_test_encoded = label_encoder.transform(y_test_indices)
```

```
mlp_model = MLPClassifier(hidden_layer_sizes=(100,), max_iter=20,
learning_rate_init=0.01)

train_accuracies = []
val_accuracies = []

for epoch in range(20):
    mlp_model.partial_fit(x_train, y_train_encoded,
classes=np.unique(y_train_encoded))

    y_train_pred = mlp_model.predict(x_train)

    train_accuracy = accuracy_score(y_train_encoded, y_train_pred)
    train_accuracies.append(train_accuracy)

    y_val_pred = mlp_model.predict(x_test)

    val_accuracy = accuracy_score(y_test_encoded, y_val_pred)
    val_accuracies.append(val_accuracy)

    print(f"Epoch {epoch+1}/20 - Training Accuracy:
{train_accuracy:.4f}, Validation Accuracy: {val_accuracy:.4f}")

y_test_pred = mlp_model.predict(x_test)
```

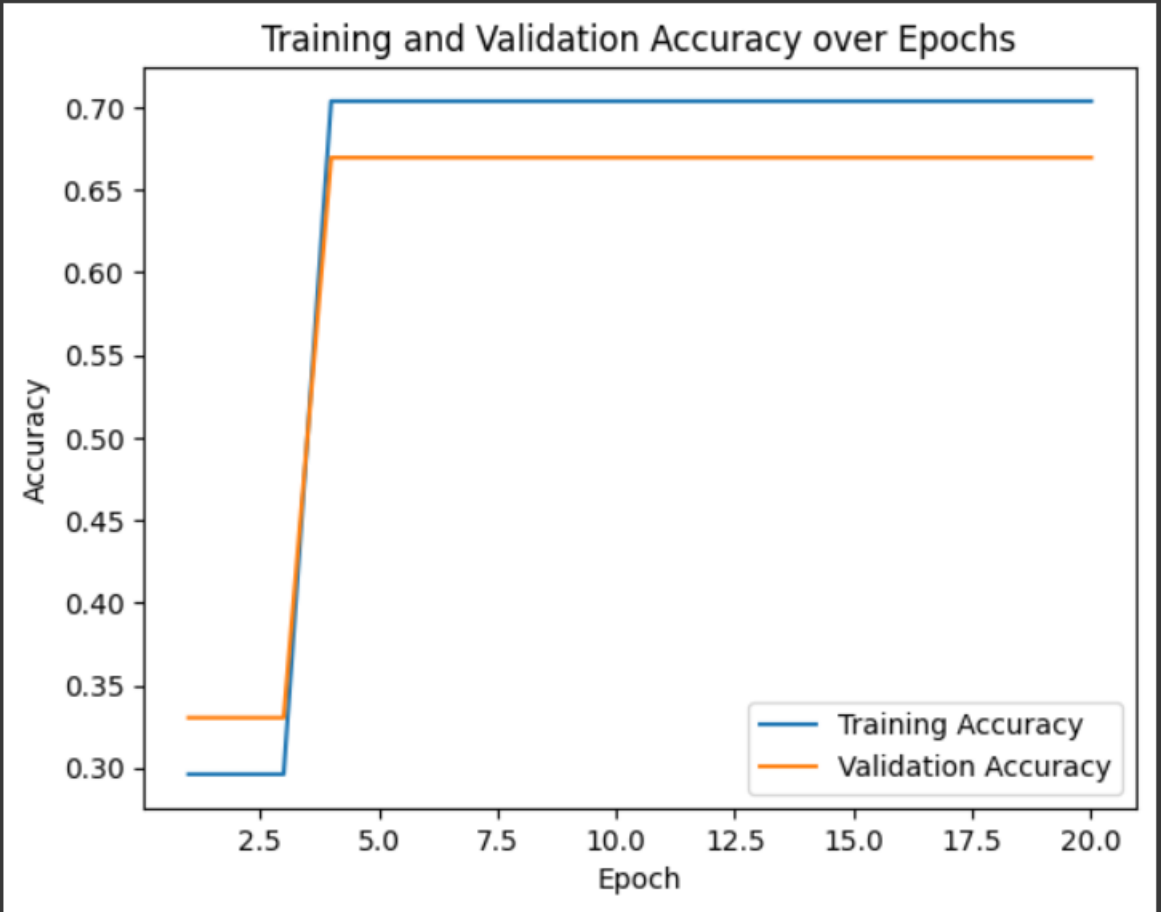
```
Epoch 1/20 - Training Accuracy: 0.2962, Validation Accuracy: 0.3304
Epoch 2/20 - Training Accuracy: 0.2962, Validation Accuracy: 0.3304
Epoch 3/20 - Training Accuracy: 0.2962, Validation Accuracy: 0.3304
Epoch 4/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 5/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 6/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 7/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 8/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 9/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 10/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 11/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 12/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 13/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 14/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 15/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 16/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 17/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 18/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 19/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
Epoch 20/20 - Training Accuracy: 0.7038, Validation Accuracy: 0.6696
```

```
test_accuracy = accuracy_score(y_test_encoded, y_test_pred)
```

```
print(f"Testing Accuracy fo Multiple Layer perceptron model :
{test_accuracy}")

plt.plot(range(1, 21), train_accuracies, label='Training Accuracy')
plt.plot(range(1, 21), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy over Epochs')
plt.show()
```

```
Testing Accuracy fo Multiple Layer perceptron model : 0.6695992179863147
```



Inferences

- All the Images in the dataset are **normalized**
- All the Images in the dataset are **resized** for th**e image Enhancement**
- The same Input Images are divided for **training and the testing** with the **ratio** of **70 : 30** for the Three Models
- The Accuracy For the Three model are shown below

1) **Convolutional neural network 80.15640377998352**
2) **Single Layer perceptron model 72.14076246334311**
3) **Multiple Layer perceptron model 66.95992179863147**

Learning Outcome

- The Size of the Image dataset will affect the ML model
- While reading the Images we need to consider the image size because it will affect the quality of the Images. It will affect the Accuracy of the Model also
- The Maximum number of Epochs for the Neural network Model plays the major role in the accuracy
- We need to consider the Learning rate for the models.