Importing required libraries

In [1]:
```
import numpy as np # for matrix calculations
import os # for file hiearchy handling
import matplotlib.pyplot as plt # plotting the images and data
import matplotlib.image as mpimg # reading the images
from  sklearn.model_selection import train_test_split # for preprocessing
from PIL import Image # image processing
import glob # file handing and folder scanner
import cv2 # for converting images to matrix
```

Getting folder paths of each images in list

In [2]:
```
listOfSubNumberDict = os.listdir("/kaggle/input/practice2/imgs")
path = '/kaggle/input/practice2/imgs/'
listOfSubNumberDict.sort()
print(listOfSubNumberDict)
```

```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

getting data from images and display images

In [3]:
```
# getting views of images in dataset
filepath = "/kaggle/input/practice2/imgs/0/1.jpg"
img = Image.open(filepath)

# get width and height
width = img.width
height = img.height

# display width and height
print("The height of the image is: ", height)
print("The width of the image is: ", width)

Image.open("/kaggle/input/practice2/imgs/0/1.jpg")
```

```
The height of the image is:  70
The width of the image is:  50
```

Out[3]:

finding number of images in each folder

In [4]:
```
for i in listOfSubNumberDict:
    print(f'there is {len(os.listdir(f"/kaggle/input/practice2/imgs/{i}"))} of images of {i}')
```

```
there is 130 of images of 0
there is 130 of images of 1
there is 130 of images of 2
there is 130 of images of 3
there is 130 of images of 4
there is 130 of images of 5
there is 130 of images of 6
there is 130 of images of 7
there is 130 of images of 8
there is 130 of images of 9
```

converting all images of 0-9 to matrix in form of numpy array

In [5]:
```
img_extension = ['jpg']
files = []

for i in listOfSubNumberDict:
    allImgPath = f'/kaggle/input/practice2/imgs/{i}/'
    [files.extend(glob.glob(allImgPath + '*.' + e)) for e in img_extension]

allImg = np.asarray([cv2.imread(file) for file in files])
```

getting information of matrix converted from images

In [6]:
```
print(allImg.shape)
print(allImg[0].shape)
print(allImg.dtype)
print(type(allImg))
print(len(allImg))
```

```
(1300, 70, 50, 3)
(70, 50, 3)
uint8
<class 'numpy.ndarray'>
1300
```

labelling the list of images for training

In [7]:
```
counter = 0
dig = 0
labels = []
for i in range(0,130):
    labels.append(0)

while(dig<=9):
    if(counter%130==0):
        dig +=1
    labels.append(dig)
    counter+=1
labels.remove(10)
labels = np.array(labels)
```

verifing the labels

In [8]:
```
print(labels)
c=0
for i in range(0,10):
    for j in labels:
        if i==j:
            c +=1
    print(f"number of {i} is {c}")
    c = 0
```

```
[0 0 0 ... 9 9 9]
number of 0 is 130
number of 1 is 130
number of 2 is 130
number of 3 is 130
number of 4 is 130
number of 5 is 130
number of 6 is 130
number of 7 is 130
number of 8 is 130
number of 9 is 130
```

scaling the matrix for getting values of each elements 0-1 that will optimise the training

In [9]:
```
for i in range(0, len(allImg)):
    allImg[i] = allImg[i]/255
```

Train - Test Split for finding accuracy of unidentified images

In [10]:
```
X = allImg
Y = np.asarray(labels)
x_train , x_test , y_train , y_test = train_test_split(X,Y,test_size=0.2,random_state=1)
```

getting number of images splitted by train test split

In [11]:
```
print (f'The X images are splitted into train set : {len(x_train)} test set : {len(x_test)}')
print (f'The Y labels are splitted into train set : {len(y_train)} test set : {len(y_test)}')
print(x_train[0])

The X images are splitted into train set : 1040 test set : 260
The Y labels are splitted into train set : 1040 test set : 260
[[[1 1 1]
  [1 1 1]
  [1 1 1]
  ...
  [1 1 1]
  [1 1 1]
  [1 1 1]]

 [[1 1 1]
  [1 1 1]
  [1 1 1]
  ...
  [1 1 1]
  [1 1 1]
  [1 1 1]]

 [[1 1 1]
  [1 1 1]
  [1 1 1]
  ...
  [1 1 1]
  [1 1 1]
  [1 1 1]]

 ...

 [[1 1 1]
  [1 1 1]
  [1 1 1]
  ...
  [1 1 1]
  [1 1 1]
  [1 1 1]]

 [[1 1 1]
  [1 1 1]
  [1 1 1]
  ...
  [1 1 1]
  [1 1 1]
  [1 1 1]]

 [[1 1 1]
  [1 1 1]
  [1 1 1]
  ...
  [1 1 1]
  [1 1 1]
  [1 1 1]]]
```

importing tensorflow for building the model

In [12]:
```
import tensorflow as tf
```

implementing neural network

In [16]:
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

feed the model with our training matrix and testing labels

In [17]:
```
model.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
33/33 [==============================] - 1s 14ms/step - loss: 2.8251 - accuracy: 0.2529
Epoch 2/10
33/33 [==============================] - 0s 15ms/step - loss: 1.3050 - accuracy: 0.5760
Epoch 3/10
33/33 [==============================] - 0s 15ms/step - loss: 0.8494 - accuracy: 0.7240
Epoch 4/10
33/33 [==============================] - 0s 14ms/step - loss: 0.8794 - accuracy: 0.7260
Epoch 5/10
33/33 [==============================] - 0s 15ms/step - loss: 0.7080 - accuracy: 0.7731
Epoch 6/10
33/33 [==============================] - 0s 15ms/step - loss: 0.6567 - accuracy: 0.7769
Epoch 7/10
33/33 [==============================] - 0s 15ms/step - loss: 0.5724 - accuracy: 0.8202
Epoch 8/10
33/33 [==============================] - 0s 15ms/step - loss: 0.4198 - accuracy: 0.8663
Epoch 9/10
33/33 [==============================] - 0s 15ms/step - loss: 0.4765 - accuracy: 0.8288
Epoch 10/10
33/33 [==============================] - 1s 15ms/step - loss: 0.3843 - accuracy: 0.8731
```

Out[17]:
```
<keras.callbacks.History at 0x792f0e034110>
```

finding loss and accuracy of unidentified images

In [18]:
```
val_loss, val_acc = model.evaluate(x_test, y_test)
print(val_loss)
print(val_acc*100)
```

```
9/9 [==============================] - 0s 5ms/step - loss: 0.9190 - accuracy: 0.7077
0.9189915657043457
70.76923251152039
```

implementing model with CNN

In [23]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten,Conv2D,MaxPooling2D
```

In [28]:
```
cnnmodel = Sequential()

cnnmodel.add(Conv2D(64,(3,3),input_shape = x_train[0].shape))
cnnmodel.add(Activation("relu"))
cnnmodel.add(MaxPooling2D(pool_size=(2,2)))


cnnmodel.add(Conv2D(64,(3,3)))
cnnmodel.add(Activation("relu"))
cnnmodel.add(MaxPooling2D(pool_size=(2,2)))


cnnmodel.add(Conv2D(64,(3,3)))
cnnmodel.add(Activation("relu"))
cnnmodel.add(MaxPooling2D(pool_size=(2,2)))


cnnmodel.add(Flatten())
cnnmodel.add(Dense(64))
cnnmodel.add(Activation("relu"))

cnnmodel.add(Dense(32))
cnnmodel.add(Activation("relu"))


cnnmodel.add(Dense(10))
cnnmodel.add(Activation("softmax"))
```

In [29]:
```
cnnmodel.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 68, 48, 64)        1792

activation_12 (Activation)   (None, 68, 48, 64)        0

max_pooling2d_6 (MaxPooling  (None, 34, 24, 64)        0
2D)

conv2d_7 (Conv2D)            (None, 32, 22, 64)        36928

activation_13 (Activation)   (None, 32, 22, 64)        0

max_pooling2d_7 (MaxPooling  (None, 16, 11, 64)        0
2D)

conv2d_8 (Conv2D)            (None, 14, 9, 64)         36928

activation_14 (Activation)   (None, 14, 9, 64)         0

max_pooling2d_8 (MaxPooling  (None, 7, 4, 64)          0
2D)

flatten_4 (Flatten)          (None, 1792)              0

dense_12 (Dense)             (None, 64)                114752

activation_15 (Activation)   (None, 64)                0

dense_13 (Dense)             (None, 32)                2080

activation_16 (Activation)   (None, 32)                0

dense_14 (Dense)             (None, 10)                330

activation_17 (Activation)   (None, 10)                0

=================================================================
Total params: 192,810
Trainable params: 192,810
Non-trainable params: 0
_________________________________________________________________
```

In [30]:
```
cnnmodel.compile(optimizer='adam',
                 loss='sparse_categorical_crossentropy',
                 metrics=['accuracy'])
```

In [33]:
```
cnnmodel.fit(x_train,y_train,epochs=7)
```

```
Epoch 1/7
33/33 [==============================] - 5s 154ms/step - loss: 0.3221 - accuracy: 0.8971
Epoch 2/7
33/33 [==============================] - 5s 150ms/step - loss: 0.2835 - accuracy: 0.9010
Epoch 3/7
33/33 [==============================] - 5s 148ms/step - loss: 0.1863 - accuracy: 0.9394
Epoch 4/7
33/33 [==============================] - 5s 163ms/step - loss: 0.1331 - accuracy: 0.9567
Epoch 5/7
33/33 [==============================] - 5s 149ms/step - loss: 0.0977 - accuracy: 0.9702
Epoch 6/7
33/33 [==============================] - 5s 151ms/step - loss: 0.0694 - accuracy: 0.9798
Epoch 7/7
33/33 [==============================] - 5s 149ms/step - loss: 0.0434 - accuracy: 0.9856
```

Out[33]:
```
<keras.callbacks.History at 0x792f9f1f6708>
```

In [34]:
```
val_loss, val_acc = cnnmodel.evaluate(x_test, y_test)
print(val_loss)
print(val_acc*100)
```

```
9/9 [==============================] - 1s 37ms/step - loss: 0.4586 - accuracy: 0.9000
0.4585790276828766
89.99999761581421
```

In [ ]: