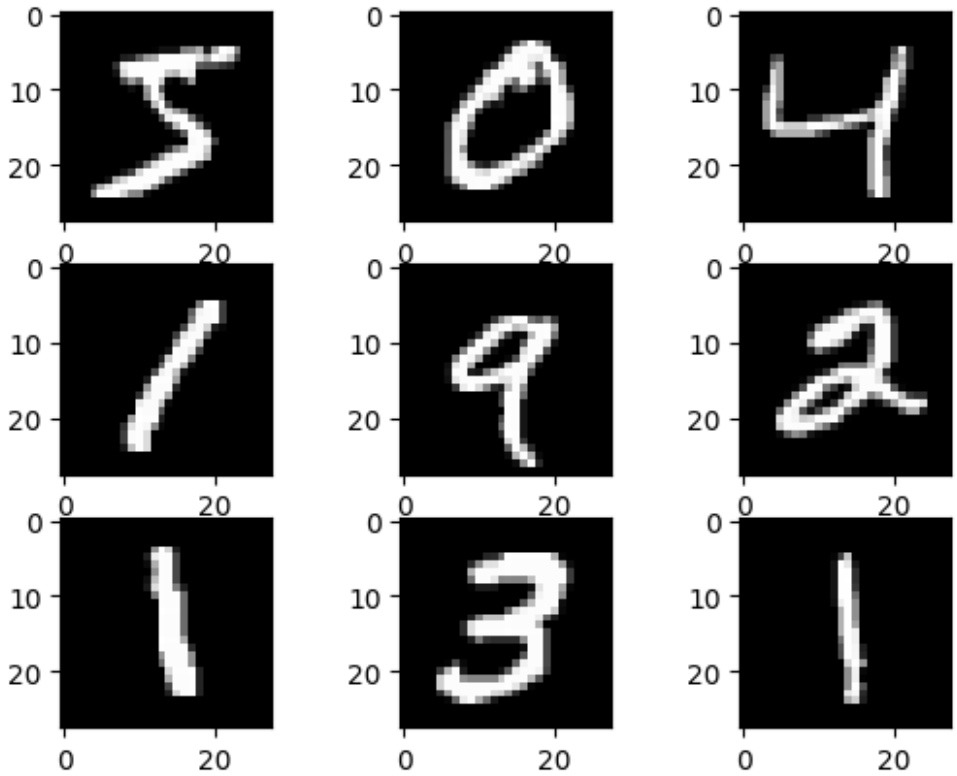
```
#@title Data loading, structuring
from keras.datasets import mnist

#loading the dataset
(train_X, train_Y), (test_X, test_Y) = mnist.load_data()

#printing the shapes of the vectors
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_Y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_Y.shape))

#------------------------------------------------------------

from matplotlib import pyplot

for i in range(9):
  pyplot.subplot(330 + 1 + i)
  pyplot.imshow(train_X[i], cmap=pyplot.get_cmap('gray'))
pyplot.show()

#------------------------------------------------------------

import numpy as np

def Concatenate(imageSeq):
  result = []
  iter = (len(imageSeq)//4)
  for i in range(0, iter):
    top_concatenated = np.concatenate(imageSeq[4*i:2+4*i], axis=1)
    bottom_concatenated = np.concatenate(imageSeq[2+4*i:4+4*i],
axis=1)
    concatenated_image = np.concatenate([top_concatenated,
bottom_concatenated], axis=0)
    result.append(concatenated_image)
  return result

def OneHotEncoding(numSeq):
  finalRes = []
  result = [0,0,0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0,0,0]
  for i in range(0, len(numSeq)):
    if((i+1)%4!=0):
      result[10 * (i%4) + numSeq[i]] = 1
    else:
      result[30 + numSeq[i]] = 1
      finalRes.append(result)
      result = [0,0,0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,   0,0,0,0,0,0,0,0,0,0]
  return finalRes
```

```
train_i=Concatenate(train_X)
train_o=OneHotEncoding(train_Y)
test_i=Concatenate(test_X)
test_o=OneHotEncoding(test_Y)

pyplot.imshow(train_i[0], cmap='gray')
pyplot.axis('off')
pyplot.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```



```
pyplot.imshow(test_i[0], cmap='gray')
pyplot.axis('off')
pyplot.show()
```

```
#@title Data normalization

print(train_i[3][14])

for i in range(0, len(train_i)):
  train_i[i] = train_i[i] / 255
for i in range(0, len(test_i)):
  test_i[i] = test_i[i] / 255

train_i = np.array(train_i)
train_o = np.array(train_o)
test_i = np.array(test_i)
test_o = np.array(test_o)

print(train_i[3][14])

print(train_i.shape)
print(train_o.shape)
print(test_i.shape)
print(test_o.shape)
```

```
[  0   0   0   0   0   0   0   0   0   0  58 181 234 254 254 254 254
254
 254 252 140  22   0   0   0   0   0   0   0   0   0   0   0   0   0
13
 109 252 228 130   0  38 165 253 233 164  49  63 253 214  31   0   0
0
   0   0]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.22745098 0.70980392
 0.91764706 0.99607843 0.99607843 0.99607843 0.99607843 0.99607843
 0.99607843 0.98823529 0.54901961 0.08627451 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.05098039
 0.42745098 0.98823529 0.89411765 0.50980392 0.         0.14901961
 0.64705882 0.99215686 0.91372549 0.64313725 0.19215686 0.24705882
 0.99215686 0.83921569 0.12156863 0.         0.         0.
 0.         0.        ]
(15000, 56, 56)
(15000, 40)
(2500, 56, 56)
(2500, 40)
```

```
#@title Model construction
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import utils
from tensorflow.keras import layers
from tensorflow.keras import datasets
from tensorflow.keras.callbacks import EarlyStopping
```

```
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D,
MaxPooling2D
import numpy as np
import matplotlib.pyplot as plt

model = keras.Sequential([
    Conv2D(8, kernel_size=(5, 5), padding='same', input_shape=(56, 56,
1), activation=tf.nn.relu),
    MaxPooling2D(pool_size=(2, 2)),
    #Dropout(0.25),

    Conv2D(16, kernel_size=(5, 5), padding='same',
activation=tf.nn.relu),
    MaxPooling2D(pool_size=(2, 2)),
    #Dropout(0.25),

    Flatten(),
    Dense(32, activation=tf.nn.sigmoid),
    Dense(32, activation=tf.nn.sigmoid),
    #Dropout(0.25),
    Dense(40, activation=tf.nn.sigmoid)
])

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)            (None, 56, 56, 8)         208

 max_pooling2d_8 (MaxPooling  (None, 28, 28, 8)         0
 2D)

 conv2d_9 (Conv2D)            (None, 28, 28, 16)        3216

 max_pooling2d_9 (MaxPooling  (None, 14, 14, 16)        0
 2D)

 flatten_4 (Flatten)          (None, 3136)              0

 dense_12 (Dense)             (None, 32)                100384

 dense_13 (Dense)             (None, 32)                1056

 dense_14 (Dense)             (None, 40)                1320

=================================================================
Total params: 106,184
Trainable params: 106,184
```

```
Non-trainable params: 0
_________________________________________________________________
```

```
model = keras.models.load_model('final.model')
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-6-a4739d448b5a> in <cell line: 1>()
----> 1 model = keras.models.load_model('final.model')

/usr/local/lib/python3.10/dist-packages/keras/saving/saving_api.py in load_model(filepath, custom_objects, compile, safe_mode, **kwargs)
    210
    211     # Legacy case.
--> 212     return legacy_sm_saving_lib.load_model(
    213         filepath, custom_objects=custom_objects, compile=compile, **kwargs
    214     )

/usr/local/lib/python3.10/dist-packages/keras/utils/traceback_utils.py in error_handler(*args, **kwargs)
     68             # To get the full stack trace, call:
     69             # `tf.debugging.disable_traceback_filtering()`
---> 70             raise e.with_traceback(filtered_tb) from None
     71         finally:
     72             del filtered_tb

/usr/local/lib/python3.10/dist-packages/keras/saving/legacy/save.py in load_model(filepath, custom_objects, compile, options)
    228                     if isinstance(filepath_str, str):
    229                         if not tf.io.gfile.exists(filepath_str):
--> 230                             raise IOError(
    231                                 f"No file or directory found at {filepath_str}"
    232                             )

OSError: No file or directory found at final.model
```

```
#@title Model training - first half of testset will be used for validation dataset
model.compile(loss=keras.losses.mse, optimizer='adam', metrics=['accuracy'])
history = model.fit(train_i, train_o, epochs=50, validation_data=(test_i[0:1250], test_o[0:1250]))
```

```
Epoch 1/50
469/469 [==============================] - 32s 66ms/step - loss: 0.1050 - accuracy: 0.0529 - val_loss: 0.0900 - val_accuracy:
```

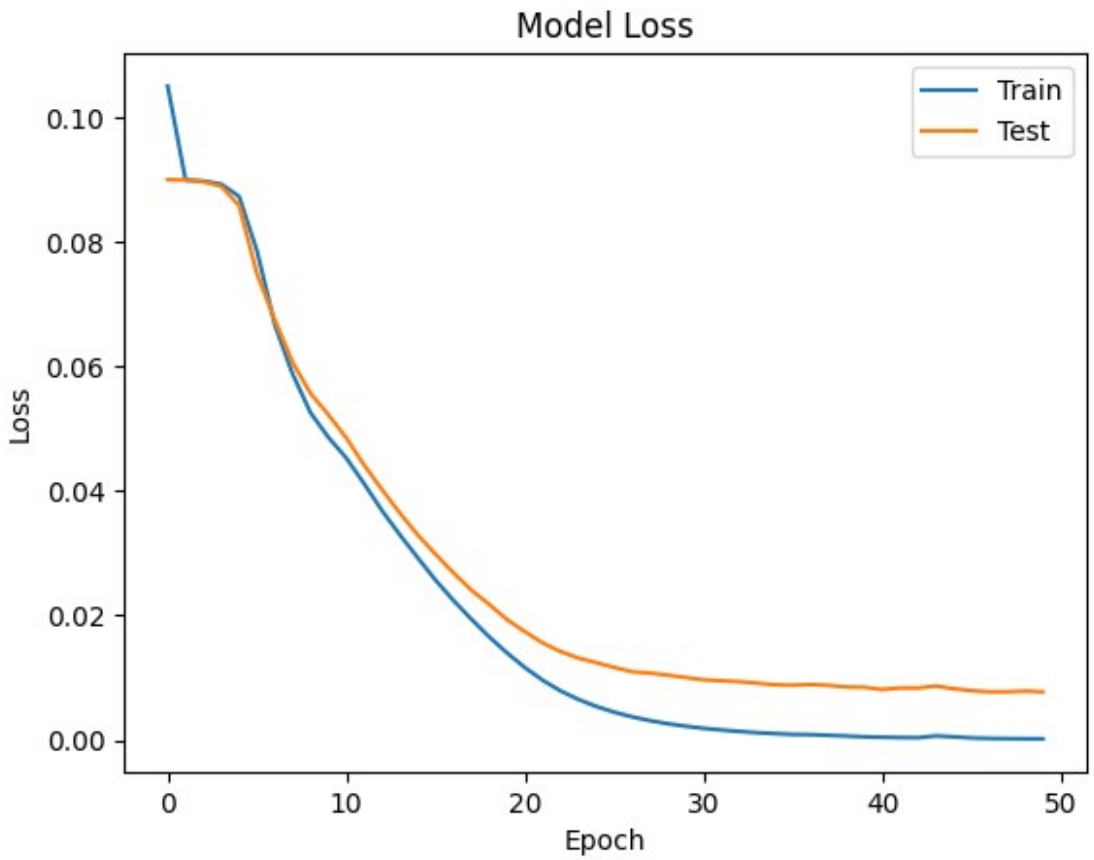
```
0.0000e+00
Epoch 2/50
469/469 [==============================] - 32s 67ms/step - loss:
0.0899 - accuracy: 0.0388 - val_loss: 0.0899 - val_accuracy:
0.0000e+00
Epoch 3/50
469/469 [==============================] - 33s 69ms/step - loss:
0.0897 - accuracy: 0.0307 - val_loss: 0.0896 - val_accuracy:
0.0000e+00
Epoch 4/50
469/469 [==============================] - 30s 65ms/step - loss:
0.0892 - accuracy: 9.3333e-04 - val_loss: 0.0889 - val_accuracy:
8.0000e-04
Epoch 5/50
469/469 [==============================] - 32s 67ms/step - loss:
0.0873 - accuracy: 0.0019 - val_loss: 0.0858 - val_accuracy:
0.0000e+00
Epoch 6/50
469/469 [==============================] - 31s 66ms/step - loss:
0.0785 - accuracy: 6.6667e-05 - val_loss: 0.0748 - val_accuracy:
0.0000e+00
Epoch 7/50
469/469 [==============================] - 31s 65ms/step - loss:
0.0666 - accuracy: 0.0000e+00 - val_loss: 0.0674 - val_accuracy:
0.0000e+00
Epoch 8/50
469/469 [==============================] - 31s 66ms/step - loss:
0.0587 - accuracy: 0.0000e+00 - val_loss: 0.0606 - val_accuracy:
0.0000e+00
Epoch 9/50
469/469 [==============================] - 30s 63ms/step - loss:
0.0525 - accuracy: 0.0000e+00 - val_loss: 0.0556 - val_accuracy:
0.0000e+00
Epoch 10/50
469/469 [==============================] - 31s 66ms/step - loss:
0.0486 - accuracy: 0.0000e+00 - val_loss: 0.0522 - val_accuracy:
8.0000e-04
Epoch 11/50
469/469 [==============================] - 29s 61ms/step - loss:
0.0453 - accuracy: 2.0000e-04 - val_loss: 0.0485 - val_accuracy:
0.0024
Epoch 12/50
469/469 [==============================] - 30s 64ms/step - loss:
0.0411 - accuracy: 0.0019 - val_loss: 0.0441 - val_accuracy: 0.0112
Epoch 13/50
469/469 [==============================] - 29s 63ms/step - loss:
0.0368 - accuracy: 0.0109 - val_loss: 0.0402 - val_accuracy: 0.0376
Epoch 14/50
469/469 [==============================] - 29s 61ms/step - loss:
```

```
0.0329 - accuracy: 0.0230 - val_loss: 0.0364 - val_accuracy: 0.0560
Epoch 15/50
469/469 [==============================] - 29s 62ms/step - loss:
0.0293 - accuracy: 0.0338 - val_loss: 0.0329 - val_accuracy: 0.0672
Epoch 16/50
469/469 [==============================] - 29s 62ms/step - loss:
0.0257 - accuracy: 0.0454 - val_loss: 0.0298 - val_accuracy: 0.0744
Epoch 17/50
469/469 [==============================] - 29s 62ms/step - loss:
0.0224 - accuracy: 0.0553 - val_loss: 0.0268 - val_accuracy: 0.0912
Epoch 18/50
469/469 [==============================] - 30s 64ms/step - loss:
0.0194 - accuracy: 0.0644 - val_loss: 0.0241 - val_accuracy: 0.1080
Epoch 19/50
469/469 [==============================] - 29s 62ms/step - loss:
0.0166 - accuracy: 0.0707 - val_loss: 0.0218 - val_accuracy: 0.1168
Epoch 20/50
469/469 [==============================] - 29s 63ms/step - loss:
0.0140 - accuracy: 0.0795 - val_loss: 0.0193 - val_accuracy: 0.1264
Epoch 21/50
469/469 [==============================] - 30s 64ms/step - loss:
0.0116 - accuracy: 0.0847 - val_loss: 0.0174 - val_accuracy: 0.1280
Epoch 22/50
469/469 [==============================] - 30s 63ms/step - loss:
0.0096 - accuracy: 0.0945 - val_loss: 0.0156 - val_accuracy: 0.1472
Epoch 23/50
469/469 [==============================] - 29s 62ms/step - loss:
0.0079 - accuracy: 0.1044 - val_loss: 0.0142 - val_accuracy: 0.1640
Epoch 24/50
469/469 [==============================] - 29s 61ms/step - loss:
0.0065 - accuracy: 0.1093 - val_loss: 0.0132 - val_accuracy: 0.1504
Epoch 25/50
469/469 [==============================] - 29s 62ms/step - loss:
0.0054 - accuracy: 0.1169 - val_loss: 0.0124 - val_accuracy: 0.1728
Epoch 26/50
469/469 [==============================] - 30s 63ms/step - loss:
0.0045 - accuracy: 0.1251 - val_loss: 0.0116 - val_accuracy: 0.1760
Epoch 27/50
469/469 [==============================] - 29s 61ms/step - loss:
0.0037 - accuracy: 0.1300 - val_loss: 0.0110 - val_accuracy: 0.1984
Epoch 28/50
469/469 [==============================] - 29s 63ms/step - loss:
0.0031 - accuracy: 0.1343 - val_loss: 0.0107 - val_accuracy: 0.2024
Epoch 29/50
469/469 [==============================] - 30s 63ms/step - loss:
0.0026 - accuracy: 0.1415 - val_loss: 0.0104 - val_accuracy: 0.2216
Epoch 30/50
469/469 [==============================] - 29s 62ms/step - loss:
0.0022 - accuracy: 0.1435 - val_loss: 0.0100 - val_accuracy: 0.2152
```

```
Epoch 31/50
469/469 [==============================] - 30s 64ms/step - loss:
0.0019 - accuracy: 0.1518 - val_loss: 0.0097 - val_accuracy: 0.2168
Epoch 32/50
469/469 [==============================] - 30s 64ms/step - loss:
0.0016 - accuracy: 0.1521 - val_loss: 0.0095 - val_accuracy: 0.2160
Epoch 33/50
469/469 [==============================] - 31s 66ms/step - loss:
0.0014 - accuracy: 0.1576 - val_loss: 0.0094 - val_accuracy: 0.2176
Epoch 34/50
469/469 [==============================] - 30s 65ms/step - loss:
0.0012 - accuracy: 0.1631 - val_loss: 0.0091 - val_accuracy: 0.2240
Epoch 35/50
469/469 [==============================] - 30s 65ms/step - loss:
0.0010 - accuracy: 0.1653 - val_loss: 0.0089 - val_accuracy: 0.2344
Epoch 36/50
469/469 [==============================] - 30s 63ms/step - loss:
8.9747e-04 - accuracy: 0.1680 - val_loss: 0.0088 - val_accuracy:
0.2328
Epoch 37/50
469/469 [==============================] - 31s 65ms/step - loss:
8.6454e-04 - accuracy: 0.1717 - val_loss: 0.0089 - val_accuracy:
0.2440
Epoch 38/50
469/469 [==============================] - 31s 66ms/step - loss:
7.5713e-04 - accuracy: 0.1779 - val_loss: 0.0088 - val_accuracy:
0.2448
Epoch 39/50
469/469 [==============================] - 30s 64ms/step - loss:
6.4243e-04 - accuracy: 0.1780 - val_loss: 0.0085 - val_accuracy:
0.2272
Epoch 40/50
469/469 [==============================] - 30s 65ms/step - loss:
5.1520e-04 - accuracy: 0.1822 - val_loss: 0.0085 - val_accuracy:
0.2320
Epoch 41/50
469/469 [==============================] - 30s 65ms/step - loss:
4.4493e-04 - accuracy: 0.1821 - val_loss: 0.0081 - val_accuracy:
0.2424
Epoch 42/50
469/469 [==============================] - 29s 62ms/step - loss:
3.9847e-04 - accuracy: 0.1842 - val_loss: 0.0084 - val_accuracy:
0.2344
Epoch 43/50
469/469 [==============================] - 29s 63ms/step - loss:
3.6632e-04 - accuracy: 0.1857 - val_loss: 0.0083 - val_accuracy:
0.2544
Epoch 44/50
469/469 [==============================] - 29s 63ms/step - loss:
```

```
6.6382e-04 - accuracy: 0.1990 - val_loss: 0.0087 - val_accuracy:
0.2568
Epoch 45/50
469/469 [==============================] - 29s 62ms/step - loss:
5.1478e-04 - accuracy: 0.2012 - val_loss: 0.0082 - val_accuracy:
0.2640
Epoch 46/50
469/469 [==============================] - 29s 62ms/step - loss:
3.3466e-04 - accuracy: 0.1985 - val_loss: 0.0079 - val_accuracy:
0.2352
Epoch 47/50
469/469 [==============================] - 29s 61ms/step - loss:
2.5965e-04 - accuracy: 0.1913 - val_loss: 0.0077 - val_accuracy:
0.2416
Epoch 48/50
469/469 [==============================] - 30s 63ms/step - loss:
2.2488e-04 - accuracy: 0.1849 - val_loss: 0.0077 - val_accuracy:
0.2440
Epoch 49/50
469/469 [==============================] - 30s 65ms/step - loss:
2.0370e-04 - accuracy: 0.1864 - val_loss: 0.0079 - val_accuracy:
0.2360
Epoch 50/50
469/469 [==============================] - 30s 64ms/step - loss:
1.8913e-04 - accuracy: 0.1872 - val_loss: 0.0077 - val_accuracy:
0.2368

#@title Model evaluation - since we're not using internal function
from keras, accuracy is calculated in manual way, second half of
testset will be used for evaluating model performance
def plt_loss(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc=0)

plt_loss(history)
plt.show()
data = model.evaluate(test_i, test_o)
print("Final Total loss : ", str(data[0]))


#Calculating accuracy of trainset
pred = model.predict(train_i)
result=[]
for z in range(0, len(pred)):
  res2=[]
  for i in range(0, 4):
```

```
    res = pred[z][0+i*10:10+i*10]
    highest=0
    for k in range(0, 10):
      if(res[highest] <= res[k]):
        highest = k
    res2.append(highest)
  result += res2

rc=0
for i in range(0, len(result)):
  if(result[i] == train_Y[i]):
    rc += 1
print("Final Total train accuracy : ", rc/len(result)*100, "%")


#Calculating accuracy of testset
pred = model.predict(test_i[1250:2500])
result=[]
for z in range(0, len(pred)):
  res2=[]
  for i in range(0, 4):
    res = pred[z][0+i*10:10+i*10]
    highest=0
    for k in range(0, 10):
      if(res[highest] <= res[k]):
        highest = k
    res2.append(highest)
  result += res2

rc=0
for i in range(0, len(result)):
  if(result[i] == test_Y[i+5000]):
    rc += 1
print("Final Total test accuracy : ", rc/len(result)*100, "%")
```




```
79/79 [==============================] - 2s 21ms/step - loss: 0.0057 -
accuracy: 0.2208
Final Total loss :  0.005654845852404833
469/469 [==============================] - 17s 35ms/step
Final Total train accuracy :  99.945 %
40/40 [==============================] - 1s 17ms/step
Final Total test accuracy :  98.02 %
```

```
#@title To save model in local environment
model.save("final.model")
```

```
WARNING:absl:Found untraced functions such as
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_update_step_xla while saving (showing 3 of 3). These functions will
not be directly callable after loading.
```

```
!zip -r /content/final.model.zip /content/final.model
```

```
  adding: content/final.model/ (stored 0%)
  adding: content/final.model/assets/ (stored 0%)
  adding: content/final.model/fingerprint.pb (stored 0%)
  adding: content/final.model/keras_metadata.pb (deflated 91%)
  adding: content/final.model/variables/ (stored 0%)
  adding: content/final.model/variables/variables.index (deflated 63%)
```

```
  adding: content/final.model/variables/variables.data-00000-of-00001
(deflated 7%)
  adding: content/final.model/saved_model.pb (deflated 88%)
```

```
#@title [External]-Model Load and test, final.model.zip file must be
located in /content directory
!unzip -qq /content/final.model.zip
```

```
#@title [External]-Model Load and test, final.model folder should be
in /content directory
from tensorflow import keras
model = keras.models.load_model('final.model')
```

```
#@title [External]-Model test - Data must be loaded and normalized by
running codes from previous code block (block 1 and block 3)
model = keras.models.load_model('final.model')

def Predict(test):
  pred = model.predict(test)
  r=[]
  for z in range(0, len(pred)):
    res2=[]
    for i in range(0, 4):
      res = pred[z][0+i*10:10+i*10]
      highest=0
      for k in range(0, 10):
        if(res[highest] <= res[k]):
          highest = k
      res2.append(highest)
    r.append(res2)
  return r

x=int(input("Put Index of testset image to predict : "))
pyplot.imshow(test_i[x], cmap='gray')
pyplot.axis('off')
pyplot.show()
result = Predict(test_i[x:x+1])
print("Predicted result : ", result)
```

```
Put Index of testset image to predict : 1
```

```
1/1 [==============================] - 0s 237ms/step
Predicted result :  [[4, 1, 4, 6]]
```

```
#@title [External]-Model Evaluation
data = model.evaluate(test_i, test_o)
print("Final Total loss : ", str(data[0]))


#Calculating accuracy of trainset
pred = model.predict(train_i)
result=[]
for z in range(0, len(pred)):
  res2=[]
  for i in range(0, 4):
    res = pred[z][0+i*10:10+i*10]
    highest=0
    for k in range(0, 10):
      if(res[highest] <= res[k]):
        highest = k
    res2.append(highest)
  result += res2

rc=0
for i in range(0, len(result)):
  if(result[i] == train_Y[i]):
    rc += 1
print("Final Total train accuracy : ", rc/len(result)*100, "%")
```

```
#Calculating accuracy of testset
pred = model.predict(test_i[1250:2500])
result=[]
for z in range(0, len(pred)):
  res2=[]
  for i in range(0, 4):
    res = pred[z][0+i*10:10+i*10]
    highest=0
    for k in range(0, 10):
      if(res[highest] <= res[k]):
        highest = k
    res2.append(highest)
  result += res2

rc=0
for i in range(0, len(result)):
  if(result[i] == test_Y[i+5000]):
    rc += 1
print("Final Total test accuracy : ", rc/len(result)*100, "%")
```

```
79/79 [==============================] - 2s 20ms/step - loss: 0.0057 -
accuracy: 0.2208
Final Total loss :  0.005654845852404833
469/469 [==============================] - 9s 19ms/step
Final Total train accuracy :  99.945 %
40/40 [==============================] - 1s 17ms/step
Final Total test accuracy :  98.02 %
```