#TEAM ID : PNT2022TMID04039

```
#importing libraries
import keras
from keras.datasets import mnist
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Flatten, Dropout
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras import regularizers
from keras import metrics
from keras.utils.np_utils import to_categorical
from keras import optimizers
from scipy import misc
import numpy as np
```

```
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

```
print(X_train.shape, Y_train.shape)
```

```
(60000, 28, 28) (60000,)
```

```
# reshape the data so as to fit the format of (samples, height, width,
channels)
X_train = X_train.reshape(60000, 28, 28, 1).astype('float32')
X_test = X_test.reshape(10000, 28, 28, 1).astype('float32')

Y_train = Y_train.reshape(60000)
Y_test = Y_test.reshape(10000)

Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)


# MODEL DEFINITION
model = Sequential()

model.add(Conv2D(filters=20, kernel_size=(6,6),
kernel_regularizer=regularizers.l2(0.04), strides=(1,1),
padding='valid', activation='relu', data_format='channels_last',
input_shape=(28,28,1)))
model.add(Conv2D(filters=20, kernel_size=(3,3),
kernel_regularizer=regularizers.l2(0.04), strides=(1,1),
padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(4,4), strides=(1,1)))
model.add(Dropout(rate=0.05,seed=3))
model.add(Conv2D(filters=10, kernel_size=(6,6),
kernel_regularizer=regularizers.l2(0.04), strides=(1,1),
padding='valid', activation='relu'))
model.add(Conv2D(filters=10, kernel_size=(3,3),
kernel_regularizer=regularizers.l2(0.04), strides=(1,1),
padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(4,4), strides=(1,1)))
model.add(Dropout(rate=0.05,seed=8))
model.add(Flatten())

model.add(Dense(units=30, activation='tanh',
kernel_regularizer=regularizers.l2(0.04)))
model.add(Dense(units=10, activation='softmax',
kernel_regularizer=regularizers.l2(0.04)))


# MODEL COMPILATION
# reduce the learning rate if training accuracy suddenly drops and
keeps decreasing
sgd = optimizers.SGD(lr=0.003) # lr by default is 0.01 for SGD

model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=[metrics.categorical_accuracy])
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/
gradient_descent.py:108: UserWarning: The `lr` argument is deprecated,
use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

```
# MODEL FIT
model.fit(X_train, Y_train, epochs=5, batch_size=50)
model.save('mnist-classifier-model.h5')
model.save_weights('mnist-classifier-weights.h5')
```

```
Epoch 1/5
1200/1200 [==============================] - 16s 6ms/step - loss:
5.0627 - categorical_accuracy: 0.5675
Epoch 2/5
1200/1200 [==============================] - 5s 4ms/step - loss:
2.6873 - categorical_accuracy: 0.9143
Epoch 3/5
1200/1200 [==============================] - 5s 4ms/step - loss:
1.7160 - categorical_accuracy: 0.9505
Epoch 4/5
1200/1200 [==============================] - 5s 4ms/step - loss:
1.2230 - categorical_accuracy: 0.9613
Epoch 5/5
```

```
1200/1200 [==============================] - 5s 4ms/step - loss:
0.9486 - categorical_accuracy: 0.9666
```

```
# MODEL EVALUATION
print("\nEvaluating the model on test data. This won't take long.
Relax!")
test_loss, test_accuracy = model.evaluate(X_test, Y_test,
batch_size=10)
print("\nAccuracy on test data : ", test_accuracy*100)
print("\nLoss on test data : ", test_loss)
```

```

Evaluating the model on test data. This won't take long. Relax!
1000/1000 [==============================] - 3s 3ms/step - loss:
0.8243 - categorical_accuracy: 0.9770

Accuracy on test data :  97.69999980926514

Loss on test data :  0.8242666125297546
```

```
#TEST THE MODEL
prediction = model.predict(X_test[:4])
print(prediction)
```

```
1/1 [==============================] - 0s 146ms/step
[[0.00552364 0.01395816 0.01411558 0.03165388 0.00503101 0.01049465
  0.00091657 0.8815433  0.009309   0.02745431]
 [0.04030754 0.03318598 0.82771784 0.01982366 0.00735839 0.00482082
  0.02058108 0.01209613 0.02671191 0.00739667]
 [0.00538969 0.9012829  0.0158494  0.00913932 0.01340986 0.00374549
  0.01759027 0.01883631 0.0076736  0.00708309]
 [0.8935922  0.00533478 0.01120698 0.00872513 0.00371715 0.01620317
  0.03634179 0.00775124 0.00643945 0.0106882 ]]
```

```
print(np.argmax(prediction, axis=1))
print(Y_test[:4])
```

```
[7 2 1 0]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
#SAVE THE MODEL
model.save('models/CNNmnist.h5')
```