In [8]:

```
import tensorflow as tf # tensorflow 2.0
from keras.datasets import mnist
import numpy as np
seed=0
np.random.seed(seed) # fix random seed
tf.random.set_seed(seed)
# input image dimensions
num_classes = 10 # 10 digits

img_rows, img_cols = 28, 28 # number of pixels

# the data, shuffled and split between train and test sets
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

In [9]:

```
X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)
# cast floats to single precision
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# rescale data in interval [0,1]
X_train /= 255
X_test /= 255
```

In [14]:

```
Y_train = keras.utils.np_utils.to_categorical(Y_train, num_classes)
Y_test = keras.utils.np_utils.to_categorical(Y_test, num_classes)
```

In [15]:

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
from keras.layers import MaxPooling2D, Dropout
model = Sequential()#add model layers
model.add(Conv2D(32, kernel_size=(5, 5),
                     activation='relu',
                     input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
# add second convolutional layer with 20 filters
model.add(Conv2D(64, (5, 5), activation='relu'))

# add 2D pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# flatten data
model.add(Flatten())

# add a dense all-to-all relu layer
model.add(Dense(1024, activation='relu'))

# apply dropout with rate 0.5
model.add(Dropout(0.5))

# soft-max layer
model.add(Dense(num_classes, activation='softmax'))
```

In [16]:

```
#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
```

In [17]:

```
#train the model
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 95s 50ms/step - loss: 0.1196 - accura
cy: 0.9635 - val_loss: 0.0320 - val_accuracy: 0.9891
Epoch 2/3
1875/1875 [==============================] - 93s 50ms/step - loss: 0.0430 - accura
cy: 0.9863 - val_loss: 0.0240 - val_accuracy: 0.9919
Epoch 3/3
1875/1875 [==============================] - 93s 50ms/step - loss: 0.0311 - accura
cy: 0.9901 - val_loss: 0.0332 - val_accuracy: 0.9901
```

Out[17]:

```
<keras.callbacks.History at 0x7f76dc7d1850>
```

In [18]:

```
# evaluate the model
score = model.evaluate(X_test, Y_test, verbose=1)

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

```
313/313 [==============================] - 5s 16ms/step - loss: 0.0332 - accuracy:
0.9901

Test loss: 0.03317980840802193
Test accuracy: 0.9901000261306763
```

In [21]:

```
predict_x=model.predict(X_test[:4])
classes_x=np.argmax(predict_x,axis=1)
```

In [22]:

```
#predict first 4 images in the test set
model.predict(X_test[:4])
```

Out[22]:

```
array([[4.44570977e-11, 7.88449361e-10, 1.57920912e-08, 1.73734200e-08,
        1.06866394e-09, 4.25641689e-10, 1.14703603e-13, 9.99999881e-01,
        2.63262634e-10, 1.12674442e-07],
       [6.04240370e-07, 1.02885814e-07, 9.99999285e-01, 6.81972256e-10,
        2.71400263e-10, 1.80242770e-13, 1.30808320e-09, 1.46937351e-10,
        1.58902793e-08, 1.04200828e-12],
       [2.68074842e-08, 9.99998927e-01, 1.80928353e-07, 1.01474285e-09,
        5.21466355e-08, 8.92409702e-09, 4.40140866e-08, 5.91038884e-07,
        1.31291557e-07, 3.73260489e-09],
       [9.99999166e-01, 5.14480236e-10, 6.72057467e-07, 1.56106472e-09,
        3.76253917e-10, 4.15453255e-10, 2.55185695e-09, 5.21882138e-09,
        1.13521956e-08, 9.95265168e-08]], dtype=float32)
```

In [24]:

```
#actual results for first 4 images in test set
Y_test[:4]
```

Out[24]:

```
array([[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.]], dtype=float32)
```

In [ ]: