EE-5353

Neural Networks and Deep Learning

Programming Assignment-6

(Convolutional Neural Networks using Keras using Google Collab)

Submitted by:
Chinmayee Makaraju(1002091569)

Task 1:

For the task 1, we need to implement the python code and run it on google collab for 10 epochs.

Python Code:

```
#reference https://towardsdatascience.com/building-a-convolutional-neural-
network-cnn-in-keras-329fbbadc5f5
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-
python
from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np
def gen_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt
from google.colab import drive
drive.mount('/content/drive')
#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#check image shape
print(X_train[0].shape)
#normalization values between 0 and 1
#X_train = X_train / 255
#X_test = X_test / 255
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]
#create model
```

```
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,
1)))
       # 64 are the number of filters, kernel size is the size of the fil
ters example 3*3 here. activation used is relu.finally shape of the image
#model.add(Conv2D(32, kernel_size=3, activation='relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# 8. Compile model
model.compile(loss='categorical_crossentropy',
             optimizer='adam',
             metrics=['accuracy'])
# 9. Fit model on training data
model.fit(X_train, y_train,
         batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit
)
# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)
print('Testing accuracy - > ',score[1] * 100)
ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image belo
w
  print("The Predicted Testing image is =%s" % (ytested[i]))
```

Task 1 Outputs:

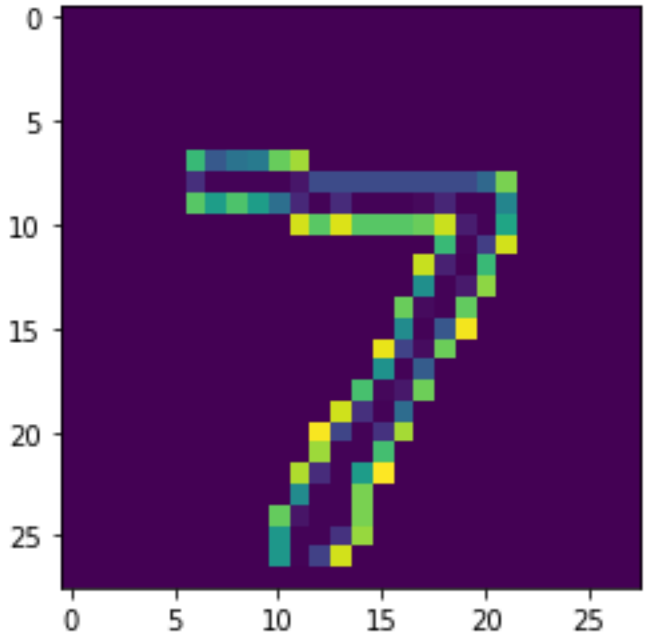
60000/60000 [==============================] - 293s 5ms/step - loss: 11.8708 - acc: 0.2630
Epoch 2/10

60000/60000 [==============================] - 301s 5ms/step - loss: 11.2132 - acc: 0.3042
Epoch 3/10

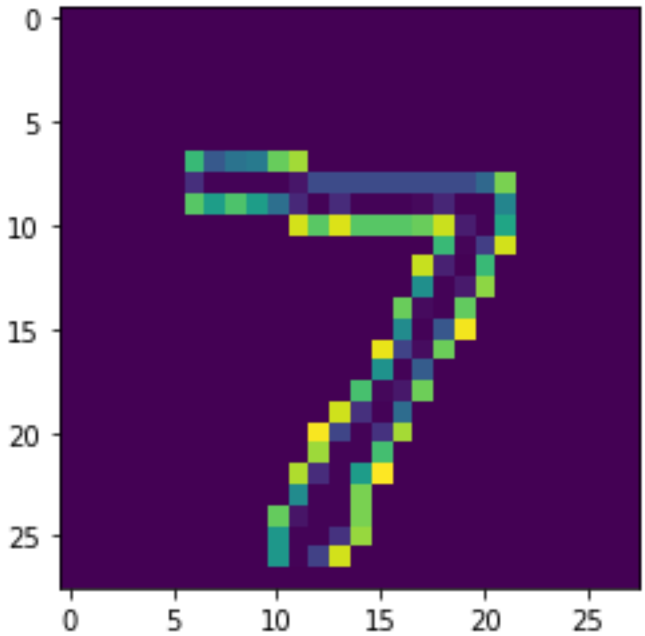
60000/60000 [==============================] - 279s 5ms/step - loss:

```
10.8851 - acc: 0.3246
Epoch 4/10

60000/60000 [================================] - 285s 5ms/step - loss:
10.7022 - acc: 0.3359
Epoch 5/10

60000/60000 [================================] - 289s 5ms/step - loss:
10.3866 - acc: 0.3555
Epoch 6/10

60000/60000 [================================] - 288s 5ms/step - loss:
10.3527 - acc: 0.3576
Epoch 7/10

60000/60000 [================================] - 280s 5ms/step - loss:
10.2740 - acc: 0.3625
Epoch 8/10

60000/60000 [================================] - 299s 5ms/step - loss:
10.0885 - acc: 0.3740
Epoch 9/10

60000/60000 [================================] - 293s 5ms/step - loss:
9.4303 - acc: 0.4148
Epoch 10/10

60000/60000 [================================] - 289s 5ms/step - loss:
9.6021 - acc: 0.4042
10000/10000 [================================] - 5s 478us/step
Testing accuracy - >  44.42
```

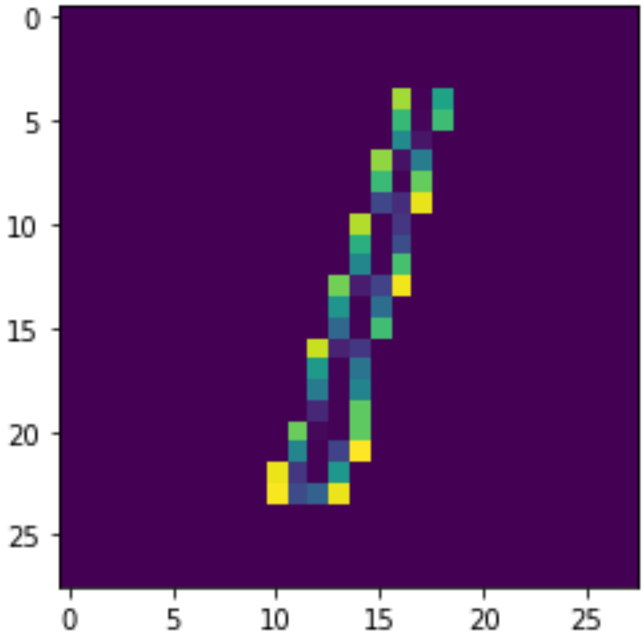
Predicted Testing Image:



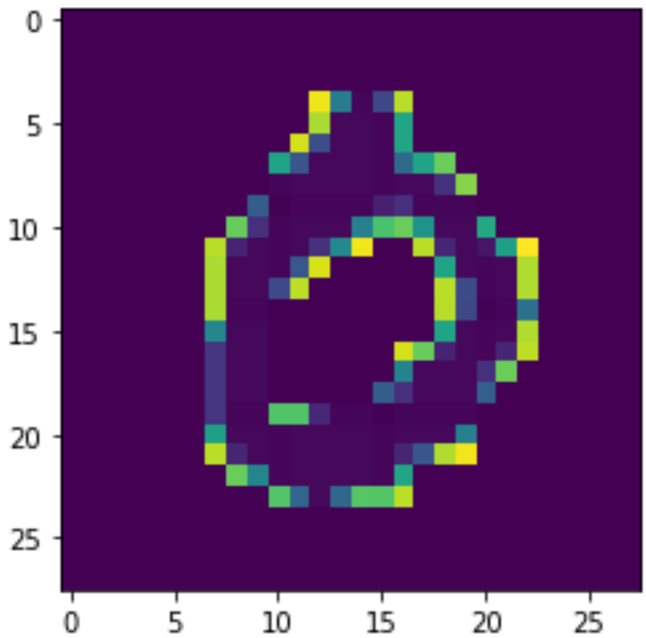
The Predicted Testing image is =7



The Predicted Testing image is =0



The Predicted Testing image is =1



The Predicted Testing image is =0

Task 2 : In this, we need to include the max pooling layer in the network and for that we need to uncomment the line in the code.

Python Code:

```
#reference https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-python
from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np
def gen_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt
from google.colab import drive
drive.mount('/content/drive')
#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#check image shape
print(X_train[0].shape)
#normalization values between 0 and 1
#X_train = X_train / 255
#X_test = X_test / 255
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
        # 64 are the number of filters, kernel size is the size of the fil
```

```
ters example 3*3 here. activation used is relu.finally shape of the image
#model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# 8. Compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# 9. Fit model on training data
model.fit(X_train, y_train,
          batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit)
# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)
print('Testing accuracy - > ',score[1] * 100)
ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image below
  print("The Predicted Testing image is =%s" % (ytested[i]))
```

Task 2 Outputs:

Epoch 1/10

60000/60000 [==============================] - 98s 2ms/step - loss: 7.3954
- acc: 0.5343

Epoch 2/10

60000/60000 [==============================] - 96s 2ms/step - loss: 2.4038
- acc: 0.8311

Epoch 3/10

60000/60000 [==============================] - 96s 2ms/step - loss: 0.3169
- acc: 0.9219

Epoch 4/10

60000/60000 [==============================] - 95s 2ms/step - loss: 0.1928
- acc: 0.9475

Epoch 5/10

60000/60000 [==============================] - 91s 2ms/step - loss: 0.1629
- acc: 0.9548

Epoch 6/10

60000/60000 [==============================] - 91s 2ms/step - loss: 0.1435
- acc: 0.9596

Epoch 7/10

60000/60000 [==============================] - 88s 1ms/step - loss: 0.1306
- acc: 0.9634

Epoch 8/10

60000/60000 [==============================] - 86s 1ms/step - loss: 0.1225
- acc: 0.9654

Epoch 9/10

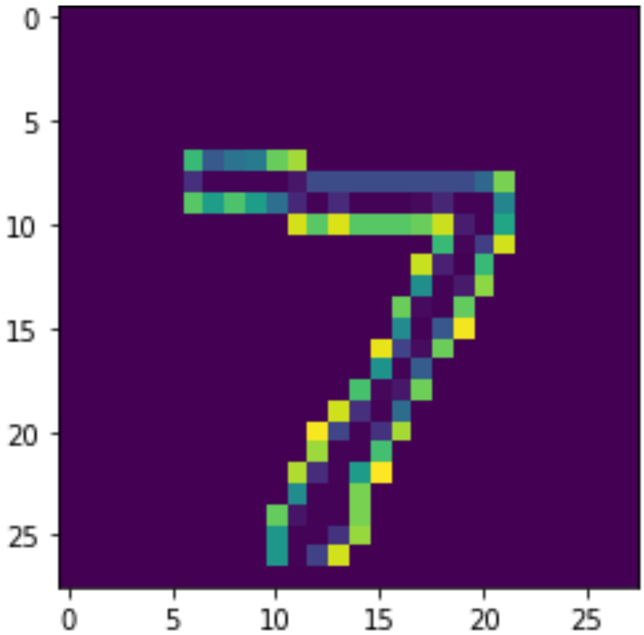
60000/60000 [==============================] - 91s 2ms/step - loss: 0.1182
- acc: 0.9663

Epoch 10/10

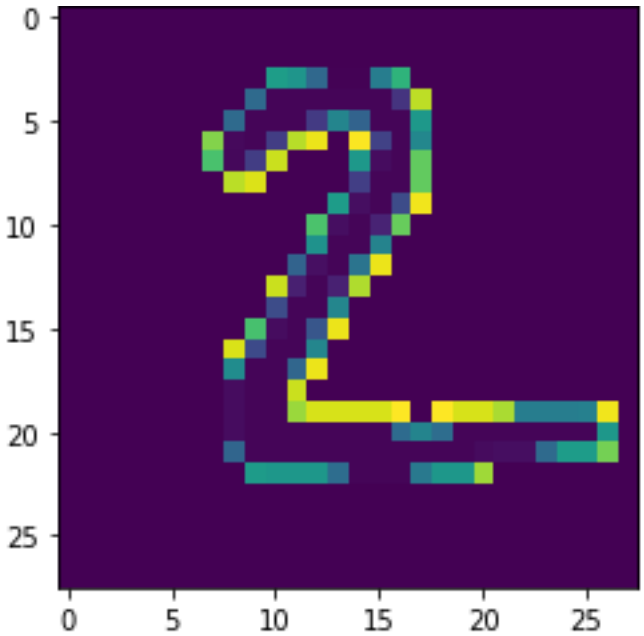
60000/60000 [==============================] - 87s 1ms/step - loss: 0.1103
- acc: 0.9695

10000/10000 [==============================] - 2s 243us/step
Testing accuracy - > 98.11999999999999

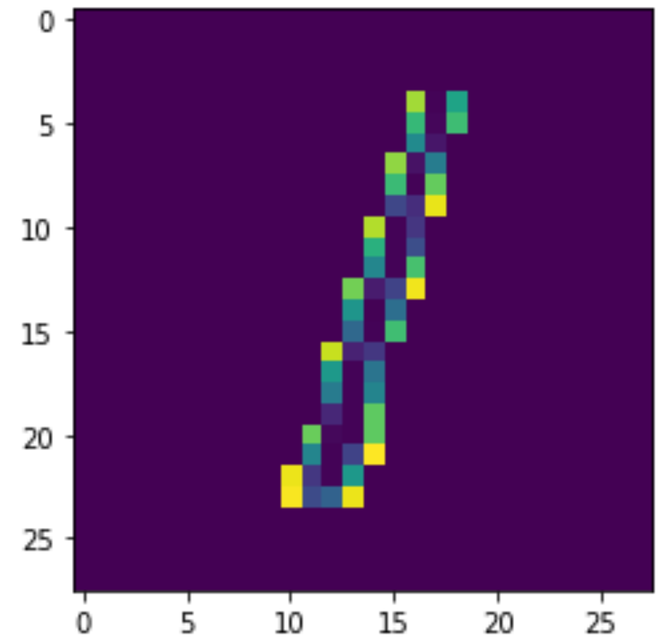
Predicted Testing Image:



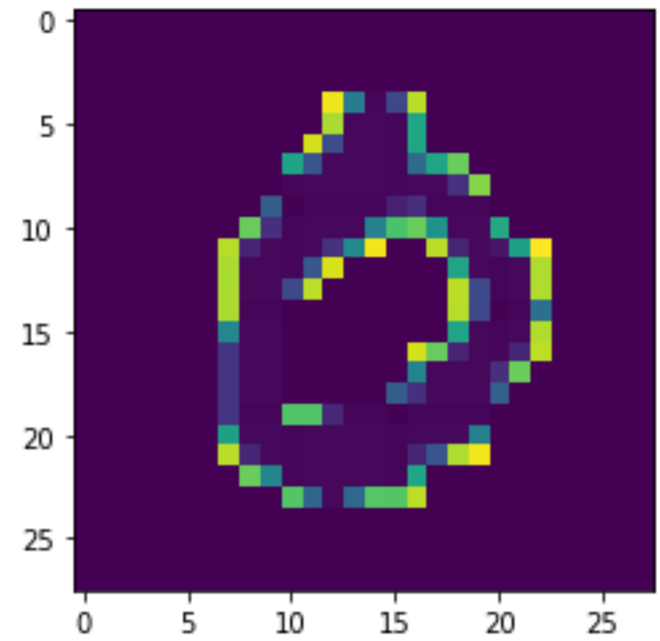
The Predicted Testing image is =7



The Predicted Testing image is =2




```
The Predicted Testing image is =1
```




```
The Predicted Testing image is =0
```

Task 3: In this, we need to add the convolution layer 2 in the network and uncomment the line in the code.

Python Code:

```
#reference https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
```

```
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-
python
from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np
def gen_image(arr):
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt
from google.colab import drive
drive.mount('/content/drive')
#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#check image shape
print(X_train[0].shape)
#normalization values between 0 and 1
#X_train = X_train / 255
#X_test = X_test / 255
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,
1)))
        # 64 are the number of filters, kernel size is the size of the fil
ters example 3*3 here. activation used is relu.finally shape of the image
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# 8. Compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# 9. Fit model on training data
model.fit(X_train, y_train,
          batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit
)

# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)
print('Testing accuracy - > ',score[1] * 100)
ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image belo
w
  print("The Predicted Testing image is =%s" % (ytested[i]))
```

Task 3 Outputs:

Epoch 1/10

60000/60000 [==============================] - 200s 3ms/step - loss: 1.0409 - acc: 0.8780
Epoch 2/10

60000/60000 [==============================] - 203s 3ms/step - loss: 0.1340 - acc: 0.9605
Epoch 3/10

60000/60000 [==============================] - 206s 3ms/step - loss: 0.1064 - acc: 0.9688
Epoch 4/10

60000/60000 [==============================] - 206s 3ms/step - loss: 0.0942 - acc: 0.9725
Epoch 5/10

60000/60000 [==============================] - 206s 3ms/step - loss: 0.0847 - acc: 0.9751
Epoch 6/10

60000/60000 [==============================] - 206s 3ms/step - loss: 0.0746 - acc: 0.9775
Epoch 7/10

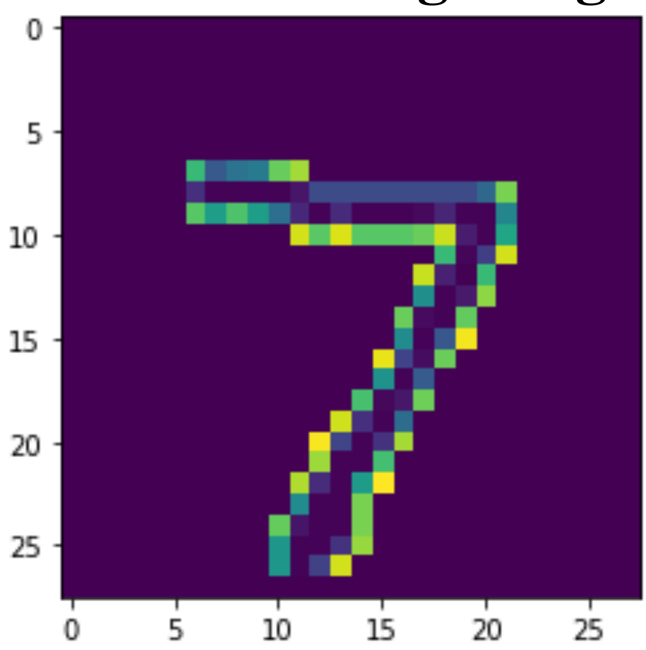
60000/60000 [==============================] - 206s 3ms/step - loss: 0.0698 - acc: 0.9787
Epoch 8/10

60000/60000 [==============================] - 206s 3ms/step - loss: 0.0608 - acc: 0.9816
Epoch 9/10

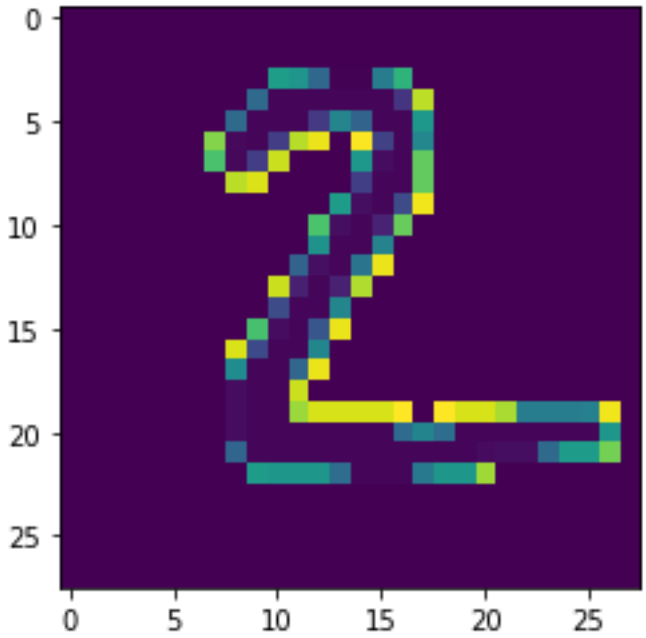
60000/60000 [==============================] - 205s 3ms/step - loss: 0.0600 - acc: 0.9816
Epoch 10/10

60000/60000 [==============================] - 205s 3ms/step - loss: 0.0579 - acc: 0.9829
10000/10000 [==============================] - 7s 736us/step
Testing accuracy - > 98.74000000000001

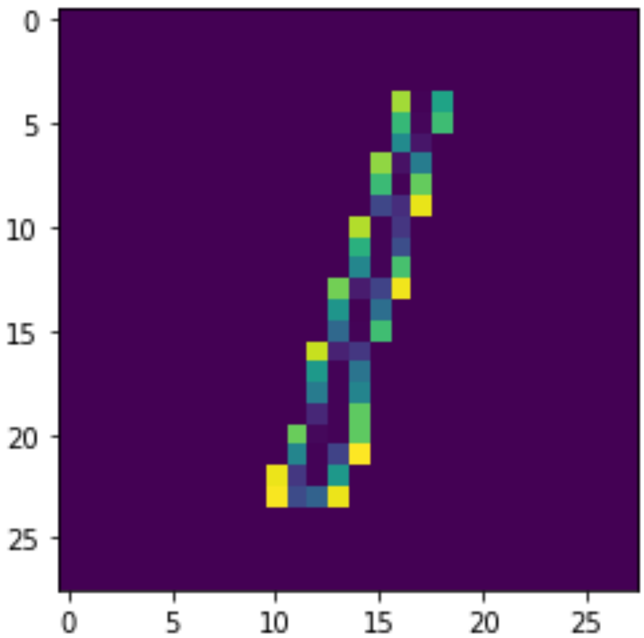
Predicted Testing Image:



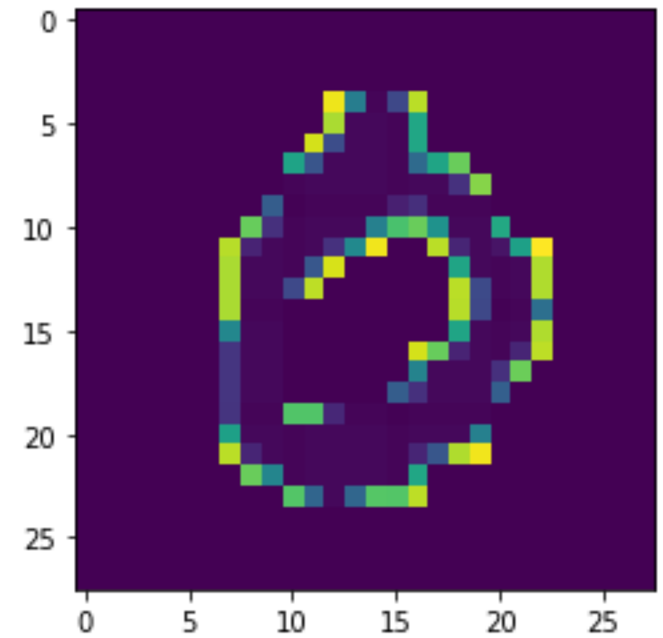
The Predicted Testing image is =7



The Predicted Testing image is =2



The Predicted Testing image is =1



```
The Predicted Testing image is =0
```

Task 4: In this, we need to normalize the values of training and testing data set. For normalization, we divide the training set by the number of values. So, X_train = X_train/255 and X_test = X_test/255. Two lines from the code are uncommented.

Python Code:

```
#reference https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
# reference https://elitedatascience.com/keras-tutorial-deep-learning-in-python
from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
import numpy as np
def gen_image(arr):
```

```
    two_d = (np.reshape(arr, (28, 28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation='nearest')
    return plt
from google.colab import drive
drive.mount('/content/drive')
#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#check image shape
print(X_train[0].shape)
#normalization values between 0 and 1
X_train = X_train / 255
X_test = X_test / 255
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
#one-hot encode target column which is equal to generate_t in program 5
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,
1)))
        # 64 are the number of filters, kernel size is the size of the fil
ters example 3*3 here. activation used is relu.finally shape of the image
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# 8. Compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# 9. Fit model on training data
model.fit(X_train, y_train,
          batch_size=32, nb_epoch=10, verbose=1) #epochs  = iterations(Nit)
```

```
# 10. Evaluate model on test data
score = model.evaluate(X_test, y_test, verbose=1)
print('Testing accuracy - > ',score[1] * 100)
ytested = model.predict_classes(X_test)
for i in range(4):
  gen_image(X_test[i]).show() # printing image vs the predicted image below
  print("The Predicted Testing image is =%s" % (ytested[I]))
```

Task 4 Outputs:

Epoch 1/10

60000/60000 [==============================] - 203s 3ms/step - loss: 0.1929 - acc: 0.9409
Epoch 2/10

60000/60000 [==============================] - 203s 3ms/step - loss: 0.0836 - acc: 0.9752
Epoch 3/10

60000/60000 [==============================] - 204s 3ms/step - loss: 0.0651 - acc: 0.9804
Epoch 4/10

60000/60000 [==============================] - 205s 3ms/step - loss: 0.0529 - acc: 0.9837
Epoch 5/10

60000/60000 [==============================] - 207s 3ms/step - loss: 0.0475 - acc: 0.9856
Epoch 6/10

60000/60000 [==============================] - 207s 3ms/step - loss: 0.0408 - acc: 0.9876
Epoch 7/10

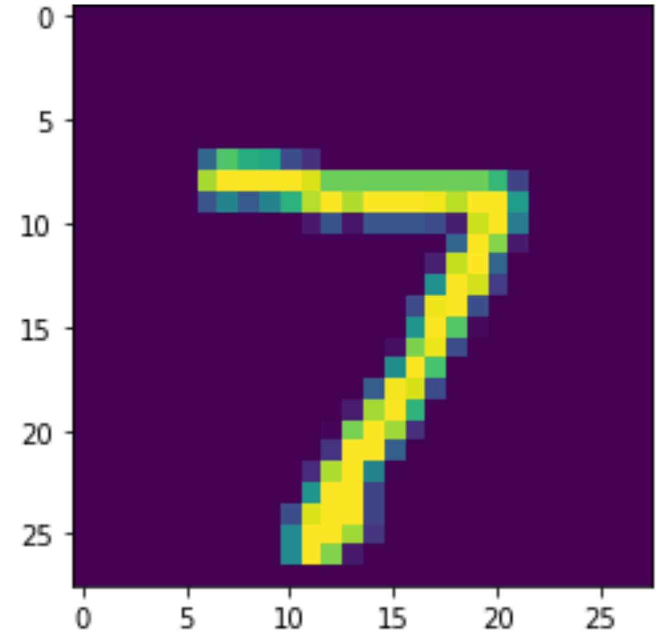
60000/60000 [==============================] - 206s 3ms/step - loss: 0.0382 - acc: 0.9883
Epoch 8/10

60000/60000 [==============================] - 205s 3ms/step - loss: 0.0337 - acc: 0.9893
Epoch 9/10

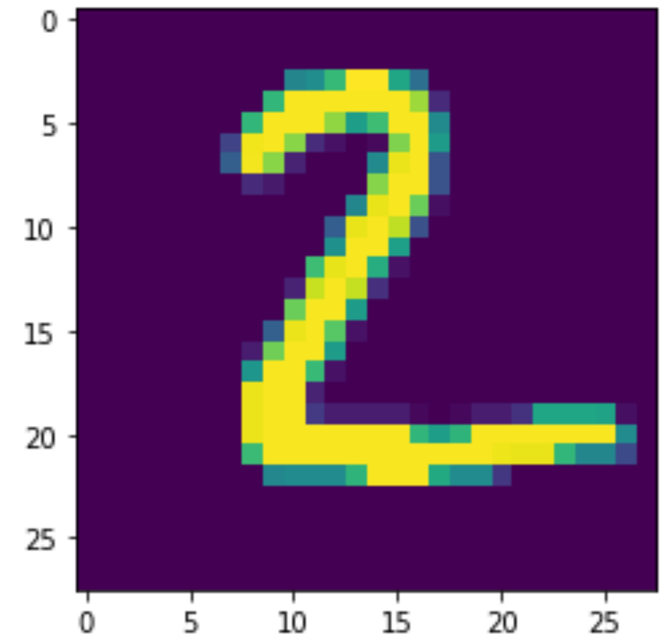
60000/60000 [==============================] - 205s 3ms/step - loss: 0.0322 - acc: 0.9898
Epoch 10/10

60000/60000 [==============================] - 203s 3ms/step - loss: 0.0310 - acc: 0.9899
10000/10000 [==============================] - 7s 744us/step
Testing accuracy - > 99.18

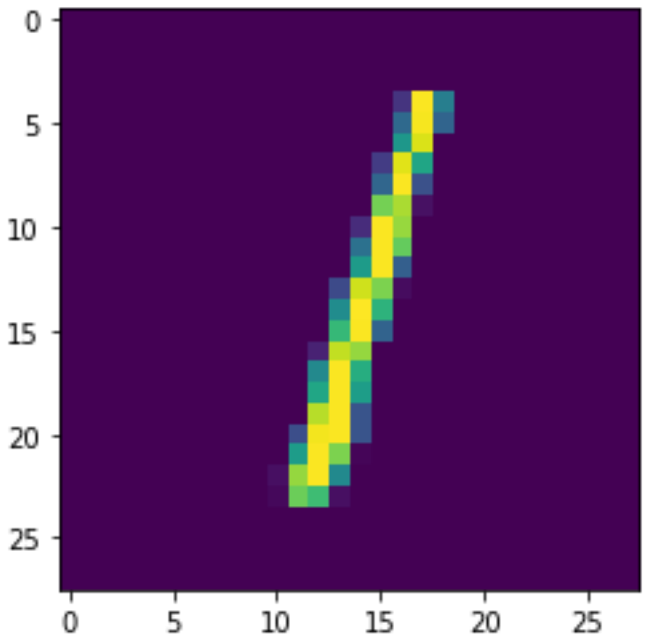
Predicted Testing Image:



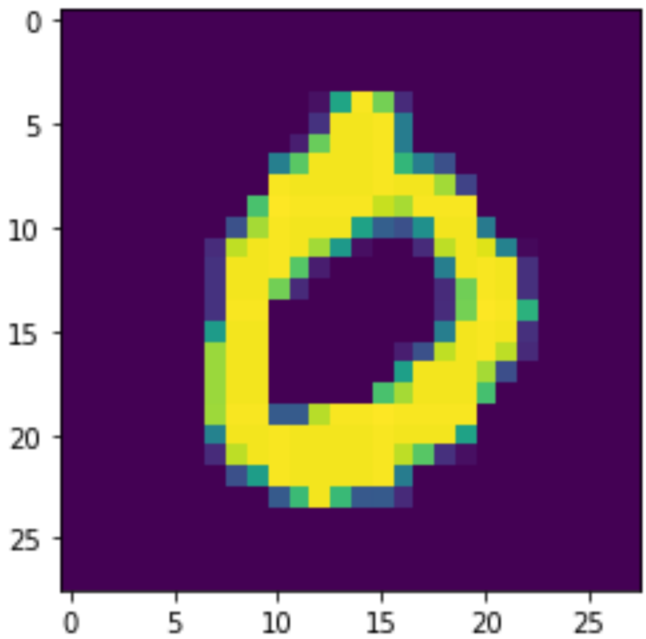
The Predicted Testing image is =7



The Predicted Testing image is =2

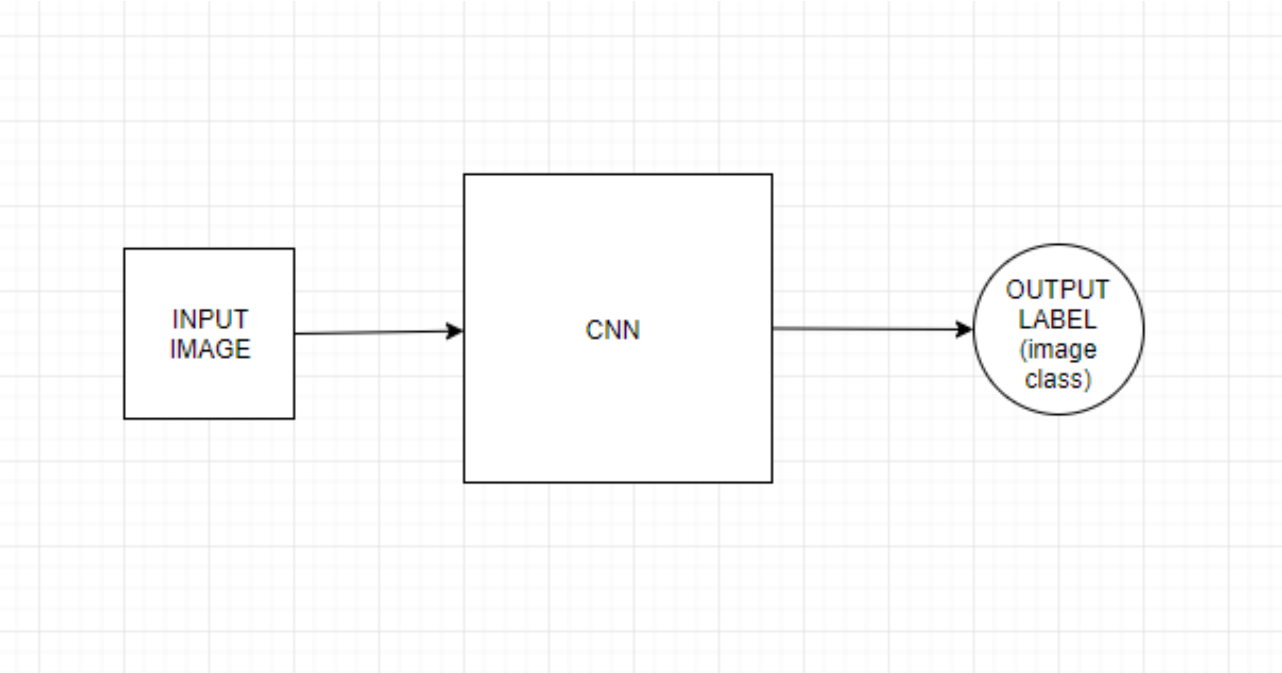



```
The Predicted Testing image is =1
```

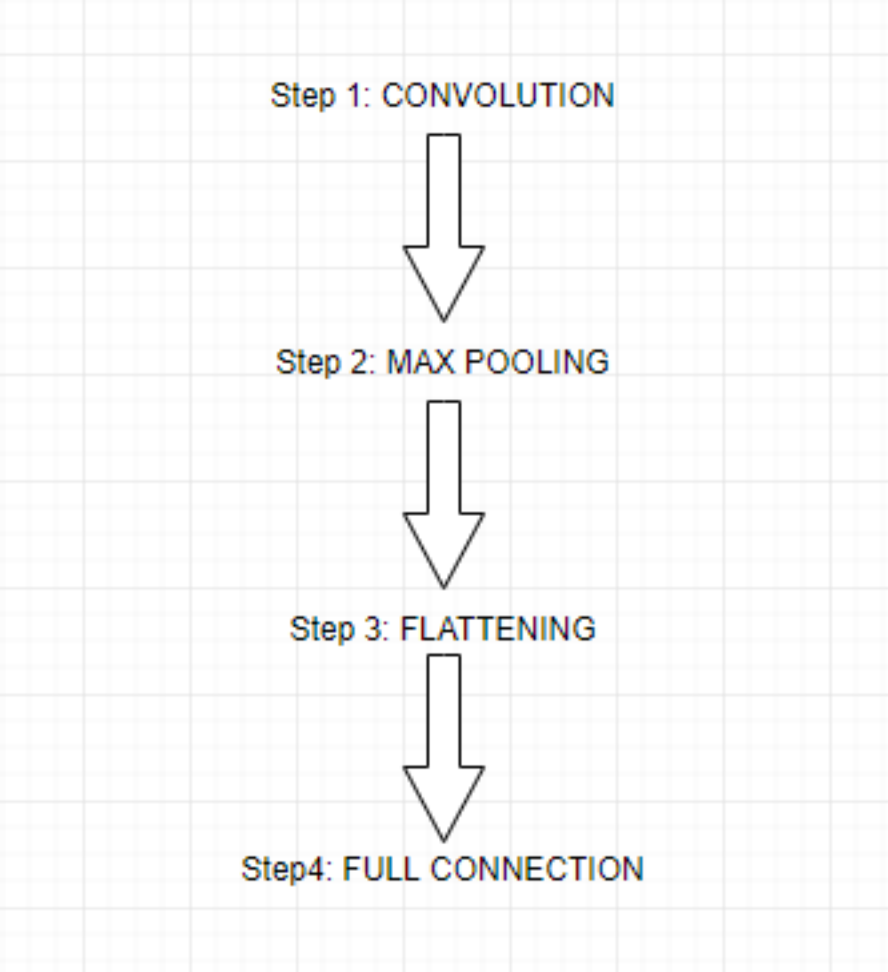



```
The Predicted Testing image is =0
```

Conclusion:



Convolutional Neural Networks are great deep learning models for computer vision to classify some images, photographs, and even some videos. They classify the image and tell the class of the image.



In the CNN code, we import all the keras packages

Sequential: This is used to in initialize a neural network and initialize it either in sequence of layers or graphs.

Convo 2D: Convolution is basically a combined integration of 2 functions. We convert the image into the pixel values. The convolution step consists of applying several feature detectors onto the input image and then we get a feature map as a result. The feature map contains some numbers and the highest number in the feature map is where the feature detector could detect a specific feature into the input image and that is the convolution operation. The convolution layer consists of feature maps. Feature map can also be called as a convoluted feature or activation map. The main property of this feature extraction is to make the image smaller, because a smaller image can be analyzed more efficiently and fast.

ReLU Layer: ReLU stands for rectifier linear unit. It’s an activation function we use to activate. The reason we’re using rectifier is to increase the non-linearity because images are highly non-linear. The rectifier linear function removes all black from the images (only non-negative values).

Pooling: The pooling layer is responsible for reducing the spatial size of the convolved feature. We’ve to be sure that our NN has a property called as the spatial variance, meaning that it does not care about each and every part of the image. If a feature is a bit distorted, the NN has the ability to detect that feature and that’s what pooling does.There are two types of pooling: Average pooling and max pooling. Max pooling results the maximum value from the portion of the image covered by the kernel whereas, average pooling, returns the average of all the values from the portion of the image covered by the kernel. Pooling is also called as down sampling.

Flattening: Used for flattening the converted pooled featured maps into large feature vector which will then become the input to the fully connected layer. It consists of all the different cells of different feature maps. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

Dense: Used to add the fully connected layers to the artificial neural networks. Each package corresponds to one step of construction of CNN. Tensor flow backend is used to make the computation faster.In the given code, For task 1, we need to run the code for 10 epochs and find out the testing accuracy. The testing accuracy for this task comes out to be 44.42. The CNN is unable to detect some images. For task 2, we need to uncomment the line which includes the max pooling layer, after running the code, we get the testing accuracy as 98.11. The CNN is able to detect the images. For task 3, we now need to uncomment the convo2D line which includes the convolution layer in the network. The testing accuracy comes out to be 98.74 and the CNN is able to detect the images well. For task 4, we need to normalize the training and testing data. Now, all the layers are included in the network and we get the maximum accuracy, which is 99.18. In this case, the CNN is able to detect all the images perfectly. We can easily see differences in the testing accuracy for each task. So, after including all the layers in CNN in task 4, we get the maximum accuracy.