In [1]:
```
#Convolutional Neural Network on MNIST handwritten digit dataset
```

In [2]:
```
#importing libraries
import tensorflow.keras
from tensorflow.keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [3]:
```
from tensorflow.python.framework import ops
ops.reset_default_graph()
```

In [4]:
```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
```

In [5]:
```
data = pd.read_csv('mnist.csv')
```

In [6]:
```
data.head()
```

Out[6]:

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [7]:
```
#reshaping into 28X28 array
data.iloc[3,1:].values.reshape(28,28).astype('uint8')
```

Out[7]:
```
array([[  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,   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,   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,   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, 124, 253, 255,  63,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,  96, 244, 251, 253,  62,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 127, 251, 251, 253,  62,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  68, 236, 251, 211,  31,   8,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  60, 228, 251, 251,  94,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, 155, 253, 253, 189,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  20, 253, 251, 235,  66,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         32, 205, 253, 251, 126,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        104, 251, 253, 184,  15,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  80,
        240, 251, 193,  23,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  32, 253,
        253, 253, 159,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 151, 251,
        251, 251,  39,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  48, 221, 251,
        251, 172,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 234, 251, 251,
        196,  12,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 253, 251, 251,
         89,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0, 159, 255, 253, 253,
         31,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  48, 228, 253, 247, 140,
          8,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  64, 251, 253, 220,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  64, 251, 253, 220,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  24, 193, 253, 220,   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,   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,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

In [8]:
```
#preprocessing data
```

In [9]:
```
#Storing Pixel array in form length width and channel in df_x
df_x = data.iloc[:,1:].values.reshape(len(data),28,28,1)

#Storing the labels in y
y = data.iloc[:,0].values
```

In [10]:
```
#Converting labels to categorical features

df_y = tensorflow.keras.utils.to_categorical(y,num_classes=10)
```

In [11]:
```
df_x = np.array(df_x)
df_y = np.array(df_y)
```

In [12]:
```
#lables
y
```

Out[12]:
```
array([5, 0, 4, ..., 5, 6, 8])
```

In [13]:
```
#categorical labels
df_y
```

Out[13]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 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., 1., 0.]], dtype=float32)
```

In [14]:
```
df_x.shape
```

Out[14]:
```
(60000, 28, 28, 1)
```

In [15]:
```
#test train split

x_train, x_test, y_train, y_test = train_test_split(df_x,df_y,test_size=0.2,random_state=0)
```

In [16]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(100))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10))
model.add(layers.Activation('softmax'))
```

In [17]:
```
#CNN model
...
#model = Sequential()
#model.add(Convolution2D(32,3,data_format='channels_last',activation='relu',input_shape=(28,28,1)))
#model.add(MaxPooling2D(pool_size=(2,2)))
#model.add(Flatten())
#model.add(Dense(100))
#model.add(Dropout(0.5))
#model.add(Dense(10))
#model.add(Activation('softmax'))
#model.compile(loss='categorical_crossentropy', optimizer = 'adadelta', metrics = ['accuracy'])


```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0         
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0         
_________________________________________________________________
dense (Dense)                (None, 100)               540900    
_________________________________________________________________
dropout (Dropout)            (None, 100)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010      
_________________________________________________________________
activation (Activation)      (None, 10)                0         
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
model.compile(optimizer='adadelta',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [20]:
```
#model.compile(optimizer='adam',
#              loss='sparse_categorical_crossentropy',
#              metrics=['accuracy'])
```

In [21]:
```
#fitting it with just 100 images for testing

#model.fit(x_train,y_train,validation_data=(x_test,y_test))

history = model.fit(x_train,y_train, epochs=10,
                    validation_data=(x_test, y_test))
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 27s 564us/sample - loss: 48.7012 - accuracy: 0.1858 - val_loss: 11.7726 - val_accuracy: 0.4852
Epoch 2/10
48000/48000 [==============================] - 28s 574us/sample - loss: 22.0350 - accuracy: 0.4022 - val_loss: 6.1711 - val_accuracy: 0.6975
Epoch 3/10
48000/48000 [==============================] - 27s 566us/sample - loss: 14.1892 - accuracy: 0.5434 - val_loss: 4.4611 - val_accuracy: 0.7763
Epoch 4/10
48000/48000 [==============================] - 27s 566us/sample - loss: 10.8774 - accuracy: 0.6282 - val_loss: 3.6647 - val_accuracy: 0.8165
Epoch 5/10
48000/48000 [==============================] - 27s 560us/sample - loss: 9.1629 - accuracy: 0.6742 - val_loss: 3.1938 - val_accuracy: 0.8391
Epoch 6/10
48000/48000 [==============================] - 27s 564us/sample - loss: 8.0801 - accuracy: 0.7058 - val_loss: 2.8875 - val_accuracy: 0.8556
Epoch 7/10
48000/48000 [==============================] - 28s 581us/sample - loss: 7.1791 - accuracy: 0.7316 - val_loss: 2.6724 - val_accuracy: 0.8658
Epoch 8/10
48000/48000 [==============================] - 28s 582us/sample - loss: 6.6847 - accuracy: 0.7463 - val_loss: 2.4803 - val_accuracy: 0.8759
Epoch 9/10
48000/48000 [==============================] - 28s 581us/sample - loss: 6.1414 - accuracy: 0.7623 - val_loss: 2.3396 - val_accuracy: 0.8831
Epoch 10/10
48000/48000 [==============================] - 28s 586us/sample - loss: 5.8192 - accuracy: 0.7741 - val_loss: 2.2259 - val_accuracy: 0.8891
```

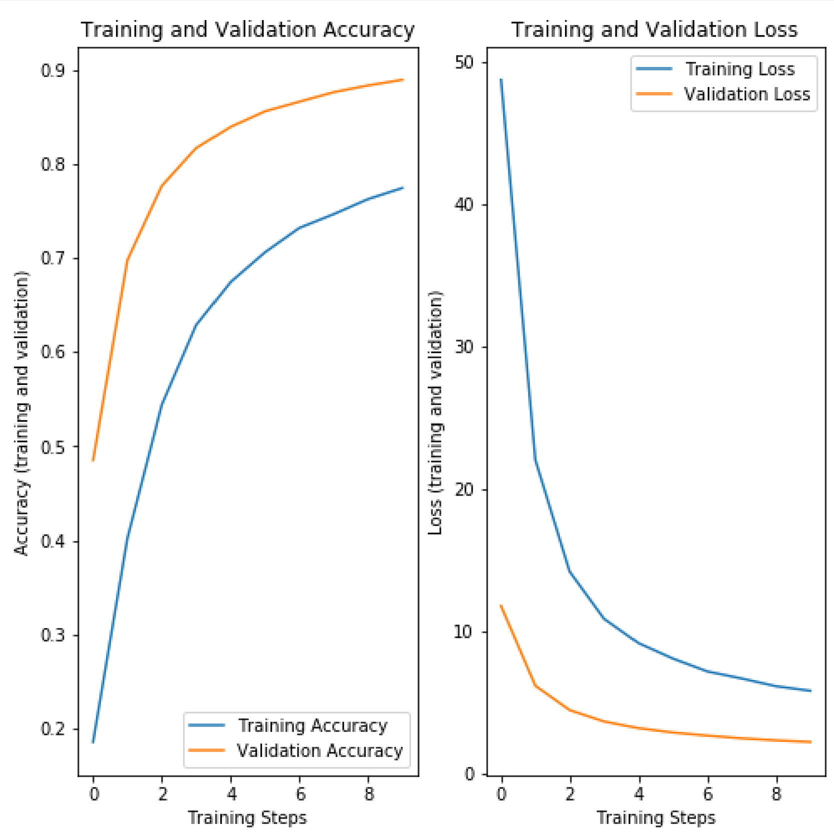
In [22]:
```
test_loss, test_acc = model.evaluate(x_test,y_test, verbose=2)
```

```
12000/1 - 2s - loss: 1.9400 - accuracy: 0.8891
```

In [23]:
```
#imporove accuracy by more epocs till the loss is almost same
print(test_acc)
```

```
0.8890833
```

In [25]:
```
EPOCHS = 10

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.show()
```



In [ ]:
```

```