Surakshith Shetty-53026240013

4a . Implementation of convolutional neural network(CNN) to predict handwritten digits using MNIST Dataset

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from random import randint
import matplotlib.pyplot as plt
import seaborn as sns

# Importing Keras layers
import keras
from keras.models import Sequential # the model we use to hold our CNN layers
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

Load the MNIST Training and Test sets

Import the training and test csv datasets, and format them to be fed into the Keras Sequential Model

```
#import the MNIST data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

Displaying a random sample of the training dataset

```
train.sample(5)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pi
19902	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3712	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3086	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
28248	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
31863	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 785 columns

Displaying a random sample of the test dataset

```
test.sample(5)
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	p
24704	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
11066	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
24514	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
17314	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2855	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 784 columns

Retrieve the shape of each set for any potential future uses

```
# /get the number of features (n), and the number of examples (m) from the test and train data
(m_train, n_train) = train.shape
(m_test, n_test) = test.shape
```

```
m_train,n_train
```

```
(42000, 785)
```

```
m_test,n_test
```

```
(28000, 784)
```

```
# Split the training set between the labels and the training data
train_labels = train['label']
train = train.drop(['label'], axis=1)
```

```
train_labels
```

```
0        1
1        0
2        1
3        4
4        0
        ..
41995    0
41996    1
41997    7
41998    6
41999    9
Name: label, Length: 42000, dtype: int64
```

```
train.shape
```

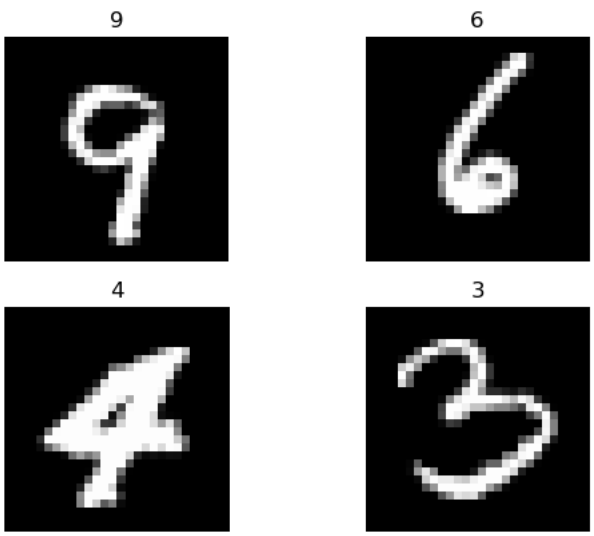
```
(42000, 784)
```

```
# Convert the training data to a 3D array for visualization purposes
train_images = train.to_numpy().reshape(42000, 28, 28)
test_images = test.to_numpy().reshape(28000, 28, 28)
```

```
#reshape(28000, 28, 28): Similarly reshapes the test data to a 3D array with shape (28000, 28, 28).
```

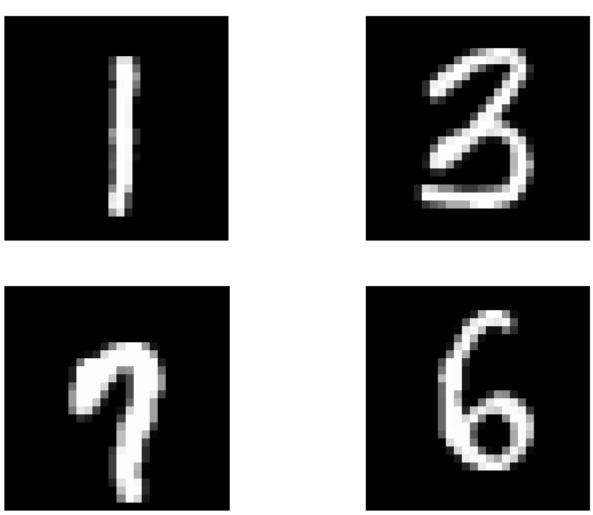
View 4 random images of the training set, with

```
# Plot 4 random images from the training data set, with the corresponding labels
for i in range(0, 4):
    train_example = np.random.randint(m_train)
    plt.subplot(220 + (i+1))
    plt.title(train_labels[train_example])
    plt.imshow(train_images[train_example], cmap='gray')
    plt.grid(False)
    plt.axis('off')

    plt.grid(False)

# This line disables the grid lines on the plot. Grid lines are the horizontal and
# vertical lines that can appear on a plot to help align and read values.
# By setting it to False, the plot will not display these lines.
```



View 4 random images of the test set

```
# Plot 4 random images from the test data set, with the corresponding labels
for i in range(0, 4):
    test_example = np.random.randint(m_test)
    plt.subplot(220 + (i+1))
    plt.imshow(test_images[test_example], cmap='gray')
    plt.grid(False)
    plt.axis('off')
plt.grid(False)

#plt.grid(False) turns off the grid lines that might otherwise appear on the plot.
#plt.grid(False) is used to disable the grid lines on the plot.
```

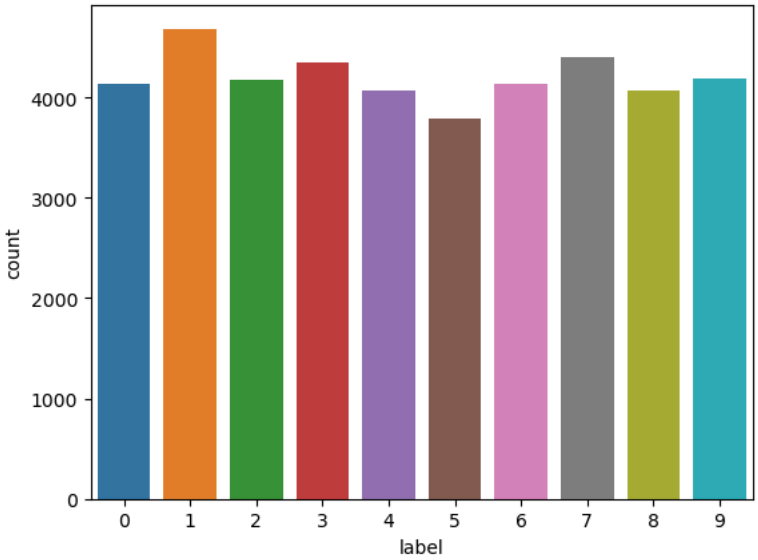


Plot the label distribution

This is to ensure there is a relatively even distribution of digits.

```
sns.countplot(train_labels)
```

```
C:\Users\askpr\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x
  warnings.warn(
<AxesSubplot:xlabel='label', ylabel='count'>
```



Modeling

Now we're ready to start building the sequential model using the Keras framework. A large amount of this was generated using the MNIST example on the Keras Documentation site

Assembling the Model

```
# create a keras sequential model
model = keras.Sequential()

# add convolution layers.
model.add(Conv2D(32, kernel_size=(3,3),
                 activation='relu',
                 input_shape=(28,28,1)))

model.add(Conv2D(64, (3,3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.25))
#Dropout(0.25): This means that 25% of the neurons will be randomly dropped out during training
# to avoid overfitting

model.add(Flatten()) #Flatten layer to convert 2D data to 1D.

model.add(Dense(128, activation='relu'))  #Fully connected layer with 128 neurons and ReLU activation.
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# Fully connected layer with 10 neurons (output layer) and Softmax activation for classification.

#view a summary of the model's different layers
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

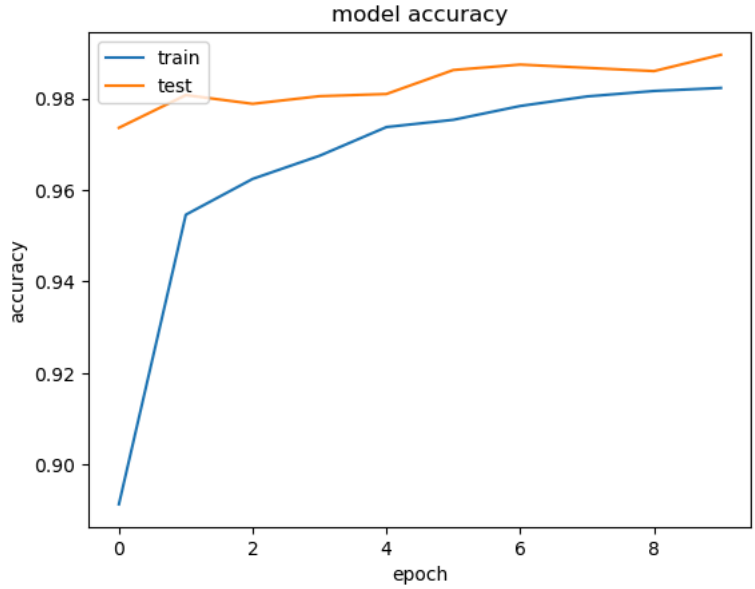
```
# compile the model
model.compile(optimizer='adam', #Adaptive Moment Estimation
              loss=keras.losses.categorical_crossentropy,
              metrics=['accuracy'])
# refer word document for alterative options in parameter


# Reshape the data into a format that can be used by the CNN
X_train = train.to_numpy().reshape(42000, 28, 28,1)
X_test = test.to_numpy().reshape(28000, 28, 28,1)
# Convert the labels into a binary class matrix (also known as one-hot encoding)
y_train = keras.utils.to_categorical(train_labels, 10)


# training the Model. Initializing the 'model_history' variable so we can retrieve some information about how the model performed
model_history = model.fit(X_train, y_train,
                          batch_size=32,
                          epochs=10,
                          verbose=1, #verbose=0:No Output: When set to 0, no output is displayed during training.
                         validation_split=0.1) #validation_split=0.1: 10% of the training data will be used for validation.
```

```
Epoch 1/10
1182/1182 [==============================] - 148s 123ms/step - loss: 0.6370 - accuracy: 0.8913 - val_loss: 0.0996 - val_accuracy: 0
Epoch 2/10
1182/1182 [==============================] - 132s 112ms/step - loss: 0.1692 - accuracy: 0.9508 - val_loss: 0.0822 - val_accuracy: 0
Epoch 3/10
1182/1182 [==============================] - 151s 128ms/step - loss: 0.1241 - accuracy: 0.9636 - val_loss: 0.0664 - val_accuracy: 0
Epoch 4/10
1182/1182 [==============================] - 160s 135ms/step - loss: 0.1085 - accuracy: 0.9675 - val_loss: 0.0574 - val_accuracy: 0
Epoch 5/10
1182/1182 [==============================] - 159s 135ms/step - loss: 0.0910 - accuracy: 0.9719 - val_loss: 0.0532 - val_accuracy: 0
Epoch 6/10
1182/1182 [==============================] - 152s 129ms/step - loss: 0.0823 - accuracy: 0.9752 - val_loss: 0.0552 - val_accuracy: 0
```

```
Epoch 7/10
1182/1182 [==============================] - 158s 134ms/step - loss: 0.0756 - accuracy: 0.9764 - val_loss: 0.0536 - val_accuracy: 0
Epoch 8/10
1182/1182 [==============================] - 157s 133ms/step - loss: 0.0677 - accuracy: 0.9800 - val_loss: 0.0495 - val_accuracy: 0
Epoch 9/10
1182/1182 [==============================] - 1577s 1s/step - loss: 0.0667 - accuracy: 0.9804 - val_loss: 0.0472 - val_accuracy: 0.98
Epoch 10/10
1182/1182 [==============================] - 173s 146ms/step - loss: 0.0593 - accuracy: 0.9826 - val_loss: 0.0425 - val_accuracy: 0
```

```
print(model_history.history.keys())
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

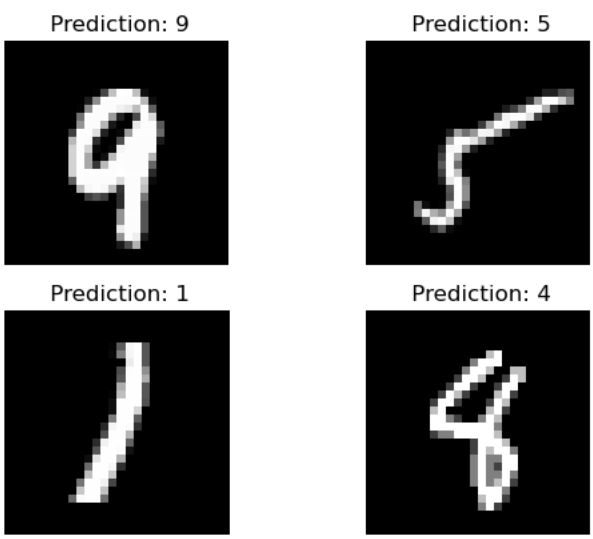
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
# Using the model to generate predicitons
test_predictions = model.predict(X_test)
```

```
875/875 [==============================] - 36s 41ms/step
```

```
# plot a random image from the test data set compared against the prediction_results output

# convert the prediciton probabilities to the actual labels
labels = np.argmax(test_predictions, axis=1)

# plot 4 random test images, with the generated prediction label
for i in range(0, 4):
    index = np.random.randint(m_test)
    plt.subplot(220 + (i+1))
    plt.title(' Prediction: ' + str(labels[index]))
    plt.imshow(test_images[index], cmap='gray')
    plt.grid(False)
    plt.axis('off')
```



Output submission

```
submission = pd.DataFrame({'ImageID': list(range(1, len(test_predictions)+1)), 'Label': labels})
submission.to_csv('MNIST_Digit_Predictions.csv', index=False, header=True)
```