```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random

from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

```
np.random.seed(42)
```

```
#get train data from train.csv
train = pd.read_csv('train.csv')
```

```
#get test data from train.csv
test = pd.read_csv('test.csv')
```

```
#get sample data from csv
sample = pd.read_csv('sample_submission.csv')
```

```
#check the train data we have
train.head()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	p
0	1	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	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 785 columns

```
test.head()
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778
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
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 784 columns

```
#get x_train and y_train from train data

x_train = train.drop('label', axis=1)
y_train = train['label']
```

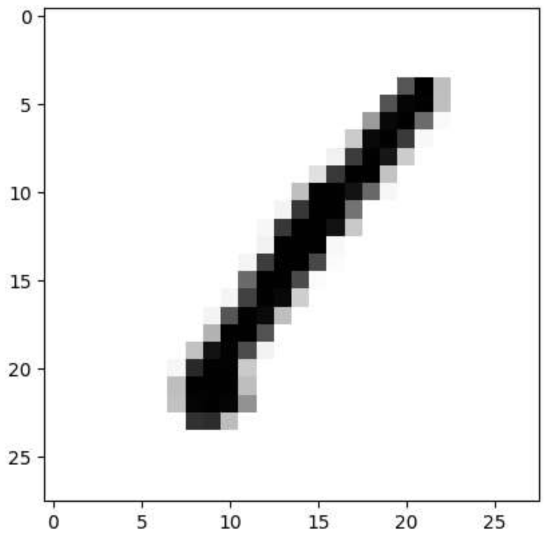
```
x_train = x_train / 255.0
test = test / 255.0
```

```
#reshapre the data and make each varaible became 28 x 28 matrix
x_train = x_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

```
#check the first image label
image_index = 0
y_train[0]
```

```
1
```

```
#check the correspond image
plt.imshow(x_train[image_index], cmap='Greys')
plt.show()
```



```
#split train and test data
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2)
```

```
#transfer the value to categorical variables
num_classes = 10

y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

```
#reshape the train and test data to
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)
```

```
# Initialize the Sequential model
model = Sequential()
# First layer with 50 neurons, ReLU activation, input shape 784 (flattened image)
model.add(Dense(50, input_shape=(784, ), activation='relu', name='dense_1'))
# Second layer with 25 neurons, ReLU activation
model.add(Dense(25, activation='relu', name='dense_2'))
# Output layer with 10 neurons (for 10 classes), softmax activation
model.add(Dense(10, activation='softmax', name='dense_output'))
# Compile the model with categorical crossentropy loss, Adam optimizer, and track accuracy
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 50)                39250

 dense_2 (Dense)             (None, 25)                1275

 dense_output (Dense)        (None, 10)                260

=================================================================
Total params: 40785 (159.32 KB)
Trainable params: 40785 (159.32 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

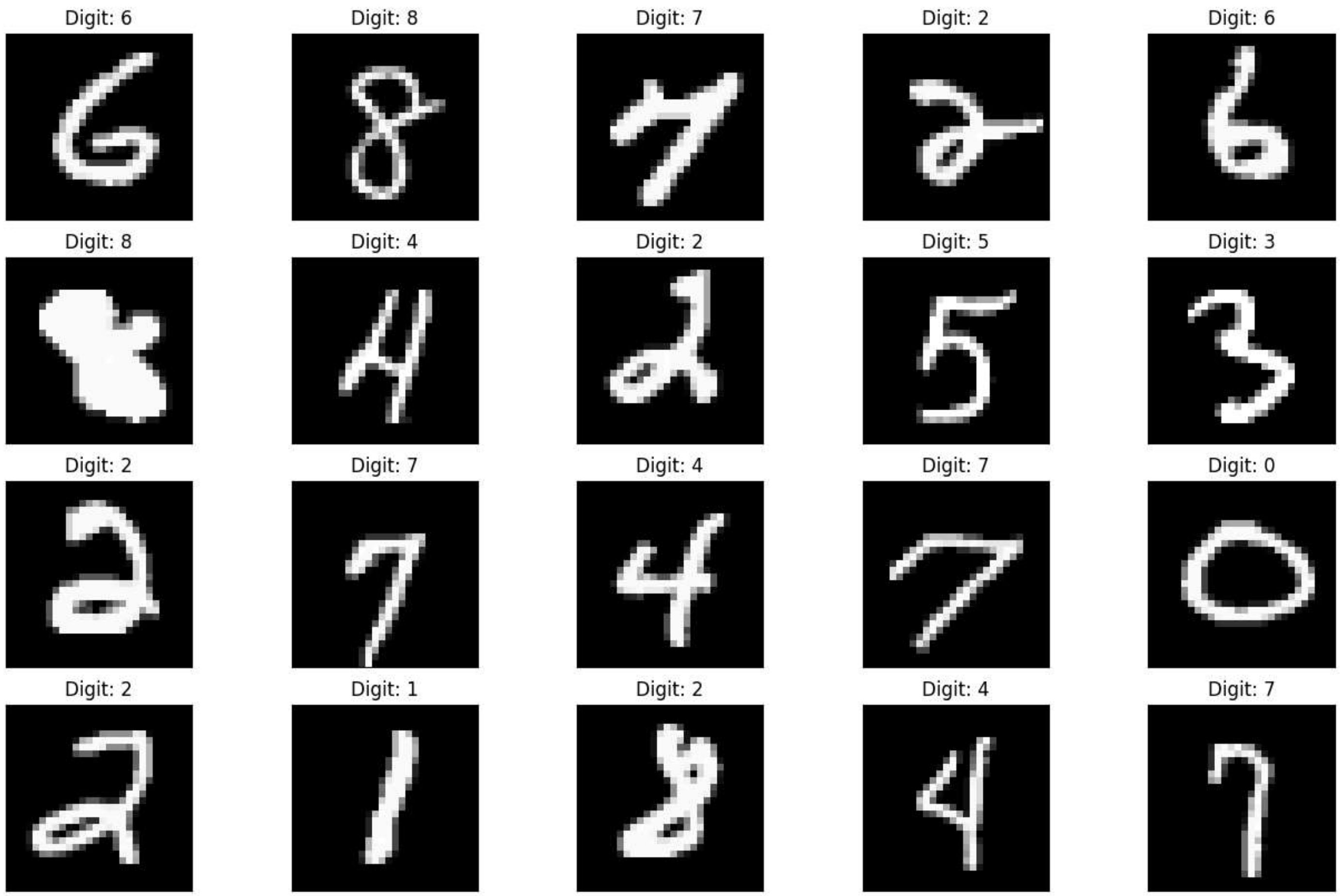
```
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))
```

```
Epoch 1/20
1050/1050 [==============================] - 4s 3ms/step - loss: 0.3971 - accuracy: 0.8850 - val_loss: 0.2338 - val_accuracy: 0.9325
Epoch 2/20
```

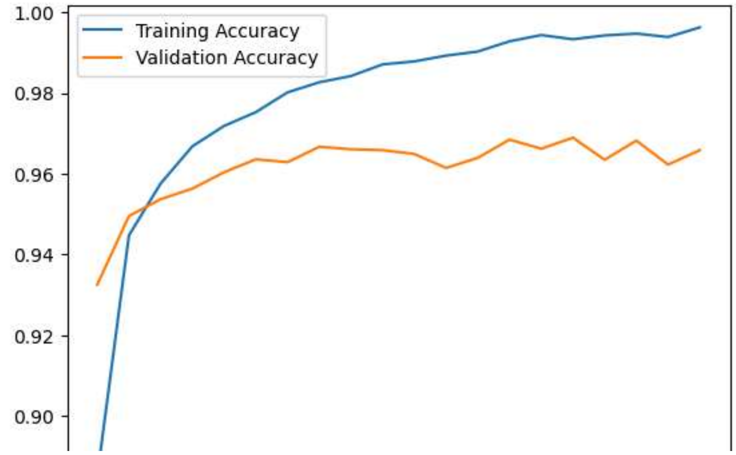
```
1050/1050 [==============================] - 3s 3ms/step - loss: 0.1844 - accuracy: 0.9447 - val_loss: 0.1772 - val_accuracy: 0.9495
Epoch 3/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.1400 - accuracy: 0.9576 - val_loss: 0.1641 - val_accuracy: 0.9537
Epoch 4/20
1050/1050 [==============================] - 3s 2ms/step - loss: 0.1102 - accuracy: 0.9668 - val_loss: 0.1524 - val_accuracy: 0.9563
Epoch 5/20
1050/1050 [==============================] - 4s 4ms/step - loss: 0.0920 - accuracy: 0.9718 - val_loss: 0.1366 - val_accuracy: 0.9604
Epoch 6/20
1050/1050 [==============================] - 6s 6ms/step - loss: 0.0783 - accuracy: 0.9752 - val_loss: 0.1254 - val_accuracy: 0.9636
Epoch 7/20
1050/1050 [==============================] - 5s 5ms/step - loss: 0.0668 - accuracy: 0.9801 - val_loss: 0.1371 - val_accuracy: 0.9629
Epoch 8/20
1050/1050 [==============================] - 6s 6ms/step - loss: 0.0552 - accuracy: 0.9826 - val_loss: 0.1252 - val_accuracy: 0.9667
Epoch 9/20
1050/1050 [==============================] - 6s 6ms/step - loss: 0.0490 - accuracy: 0.9842 - val_loss: 0.1264 - val_accuracy: 0.9661
Epoch 10/20
1050/1050 [==============================] - 4s 4ms/step - loss: 0.0427 - accuracy: 0.9871 - val_loss: 0.1306 - val_accuracy: 0.9658
Epoch 11/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0376 - accuracy: 0.9878 - val_loss: 0.1275 - val_accuracy: 0.9649
Epoch 12/20
1050/1050 [==============================] - 4s 4ms/step - loss: 0.0324 - accuracy: 0.9893 - val_loss: 0.1499 - val_accuracy: 0.9614
Epoch 13/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0288 - accuracy: 0.9903 - val_loss: 0.1392 - val_accuracy: 0.9639
Epoch 14/20
1050/1050 [==============================] - 3s 2ms/step - loss: 0.0241 - accuracy: 0.9928 - val_loss: 0.1323 - val_accuracy: 0.9685
Epoch 15/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0195 - accuracy: 0.9943 - val_loss: 0.1391 - val_accuracy: 0.9662
Epoch 16/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0208 - accuracy: 0.9933 - val_loss: 0.1431 - val_accuracy: 0.9689
Epoch 17/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0180 - accuracy: 0.9943 - val_loss: 0.1599 - val_accuracy: 0.9635
Epoch 18/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0170 - accuracy: 0.9947 - val_loss: 0.1559 - val_accuracy: 0.9682
Epoch 19/20
1050/1050 [==============================] - 3s 2ms/step - loss: 0.0171 - accuracy: 0.9939 - val_loss: 0.1700 - val_accuracy: 0.9623
Epoch 20/20
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0135 - accuracy: 0.9962 - val_loss: 0.1721 - val_accuracy: 0.9658
```

```
def plot_digit(image, digit, plt, i):
    #Arrange plots in 4 rows and 5 columns
    plt.subplot(4, 5, i + 1)
    plt.imshow(image, cmap=plt.get_cmap('gray'))
    plt.title(f"Digit: {digit}")
    plt.xticks([])
    plt.yticks([])

random.seed(5)
# Prepare to display multiple images
plt.figure(figsize=(16, 10))

# Loop to display 20 random digit images from the test set
for i in range(20):
    image = random.choice(x_test).squeeze()
    digit = np.argmax(model.predict(image.reshape((1, 784)))[0], axis=-1)
    plot_digit(image.reshape(28,28), digit, plt, i)

plt.show()
```

```
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
```




```
#test and visualize accuracy of my model

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
plt.plot(accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.legend()
plt.show()
```




```
# accuracy

accuracy[-1]
```

```
0.9962499737739563
```

```
#make prediction base on my model for test data

pred = model.predict(test.reshape(28000, 784))
pred = np.argmax(pred, axis=-1)
sample['label1'] = pred
sample = sample.drop(['label'], axis=1)
sample.head()
sample.to_csv('submission_Yuchen.csv', index=False)
```

```
875/875 [==============================] - 3s 3ms/step
```

Compare with Kneighbor classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.25)
# create a k-nearest neighbors model
knn = KNeighborsClassifier(n_neighbors=5)
# fit the model to the training data
knn.fit(x_train, y_train)
# make predictions on the test data
y_pred = knn.predict(x_test)
# evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```