1 - Import the libraries

```
# Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import cv2
from scipy.io import loadmat
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, LeakyReLU, BatchNormalization
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Set random seed for reproducibility
np.random.seed(7)

# Load data from the mnist dataset and create train and test datasets
mnist = loadmat('/content/drive/MyDrive/mnist-original.mat')
mnist_data = mnist["data"].T
mnist_data = mnist_data.reshape(len(mnist_data), 28, 28, 1)
mnist_label = mnist["label"][0]

X_train = mnist_data[0:60000]
y_train = mnist_label[0:60000]
X_test = mnist_data[60000:70000]
y_test = mnist_label[60000:70000]

n_classes = 10

# Change the labels from categorical to one-hot encoding
# e.g., class '3' transforms into vector [0,0,0,1,0,0,0,0,0,0]
y_train = to_categorical(y_train, n_classes)
y_test = to_categorical(y_test, n_classes)

# Divide the train dataset into train and validation datasets. 80% train / 20% validation
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.2, random_state=123)

# Show example images of digits
n_rows = 2
n_cols = 5

plt.figure(figsize=(10,5))
for i in range(n_rows*n_cols):
    ax = plt.subplot(n_rows, n_cols, i + 1)
    ax.set_yticklabels([])
    ax.set_xticklabels([])
    plt.imshow(X_train[random.randint(0, len(X_train) - 1)], cmap=plt.get_cmap('gray'))
```

```
# Define the Neural Network
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='linear', input_shape = (28,28,1), padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Conv2D(64, (3, 3), activation='linear', padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Conv2D(128, (3, 3), activation='linear', padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Conv2D(256, (3, 3), activation='linear', padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Flatten())

model.add(Dense(256, activation='linear'))
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Dense(64, activation='linear'))
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.25))
model.add(BatchNormalization())

model.add(Dense(n_classes, activation='softmax'))

model.summary()
```

```
 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 batch_normalization_2 (Bat  (None, 4, 4, 128)         512
 chNormalization)

 conv2d_3 (Conv2D)           (None, 4, 4, 256)         295168

 leaky_re_lu_3 (LeakyReLU)   (None, 4, 4, 256)         0

 max_pooling2d_3 (MaxPoolin  (None, 2, 2, 256)         0
 g2D)

 dropout_3 (Dropout)         (None, 2, 2, 256)         0

 batch_normalization_3 (Bat  (None, 2, 2, 256)         1024
 chNormalization)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 256)               262400

 leaky_re_lu_4 (LeakyReLU)   (None, 256)               0

 dropout_4 (Dropout)         (None, 256)               0

 batch_normalization_4 (Bat  (None, 256)               1024
 chNormalization)

 dense_1 (Dense)             (None, 64)                16448

 leaky_re_lu_5 (LeakyReLU)   (None, 64)                0

 dropout_5 (Dropout)         (None, 64)                0

 batch_normalization_5 (Bat  (None, 64)                256
 chNormalization)

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 670538 (2.56 MB)
Trainable params: 668938 (2.55 MB)
Non-trainable params: 1600 (6.25 KB)
_________________________________________________________________
```

```
# Compile the NN
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the NN
history = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_validation, y_validation))
```

```
Epoch 1/20
375/375 [==============================] - 173s 453ms/step - loss: 0.4450 - accuracy: 0.8639 - val_loss: 0.0587 - val_accuracy: 0.9807
Epoch 2/20
375/375 [==============================] - 162s 432ms/step - loss: 0.1119 - accuracy: 0.9684 - val_loss: 0.0474 - val_accuracy: 0.9866
Epoch 3/20
375/375 [==============================] - 160s 427ms/step - loss: 0.0812 - accuracy: 0.9765 - val_loss: 0.0308 - val_accuracy: 0.9908
Epoch 4/20
375/375 [==============================] - 162s 433ms/step - loss: 0.0637 - accuracy: 0.9809 - val_loss: 0.0245 - val_accuracy: 0.9929
Epoch 5/20
375/375 [==============================] - 158s 423ms/step - loss: 0.0567 - accuracy: 0.9831 - val_loss: 0.0392 - val_accuracy: 0.9882
Epoch 6/20
375/375 [==============================] - 158s 422ms/step - loss: 0.0496 - accuracy: 0.9851 - val_loss: 0.0241 - val_accuracy: 0.9931
Epoch 7/20
375/375 [==============================] - 158s 422ms/step - loss: 0.0427 - accuracy: 0.9874 - val_loss: 0.0257 - val_accuracy: 0.9929
Epoch 8/20
375/375 [==============================] - 161s 430ms/step - loss: 0.0399 - accuracy: 0.9883 - val_loss: 0.0216 - val_accuracy: 0.9934
Epoch 9/20
375/375 [==============================] - 156s 416ms/step - loss: 0.0359 - accuracy: 0.9892 - val_loss: 0.0211 - val_accuracy: 0.9944
Epoch 10/20
375/375 [==============================] - 159s 423ms/step - loss: 0.0344 - accuracy: 0.9897 - val_loss: 0.0271 - val_accuracy: 0.9912
Epoch 11/20
375/375 [==============================] - 156s 416ms/step - loss: 0.0297 - accuracy: 0.9914 - val_loss: 0.0234 - val_accuracy: 0.9931
Epoch 12/20
375/375 [==============================] - 159s 423ms/step - loss: 0.0290 - accuracy: 0.9910 - val_loss: 0.0203 - val_accuracy: 0.9942
Epoch 13/20
375/375 [==============================] - 158s 421ms/step - loss: 0.0285 - accuracy: 0.9912 - val_loss: 0.0260 - val_accuracy: 0.9920
Epoch 14/20
375/375 [==============================] - 155s 414ms/step - loss: 0.0268 - accuracy: 0.9917 - val_loss: 0.0262 - val_accuracy: 0.9933
Epoch 15/20
375/375 [==============================] - 160s 427ms/step - loss: 0.0241 - accuracy: 0.9924 - val_loss: 0.0225 - val_accuracy: 0.9944
Epoch 16/20
375/375 [==============================] - 160s 427ms/step - loss: 0.0263 - accuracy: 0.9921 - val_loss: 0.0227 - val_accuracy: 0.9937
Epoch 17/20
```

```
375/375 [==============================] - 161s 428ms/step - loss: 0.0201 - accuracy: 0.9939 - val_loss: 0.0249 - val_accuracy: 0.9932
Epoch 18/20
375/375 [==============================] - 159s 425ms/step - loss: 0.0228 - accuracy: 0.9927 - val_loss: 0.0247 - val_accuracy: 0.9936
Epoch 19/20
375/375 [==============================] - 160s 427ms/step - loss: 0.0200 - accuracy: 0.9935 - val_loss: 0.0230 - val_accuracy: 0.9937
Epoch 20/20
375/375 [==============================] - 158s 420ms/step - loss: 0.0188 - accuracy: 0.9940 - val_loss: 0.0245 - val_accuracy: 0.9933
```

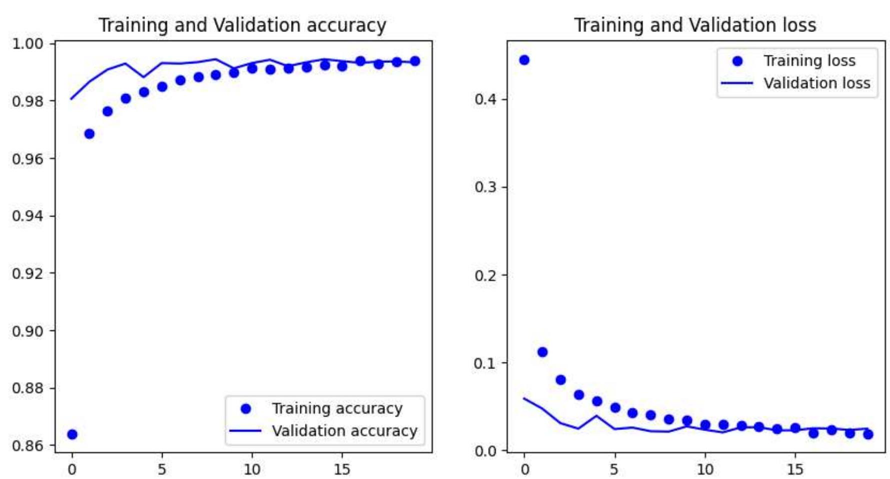
5 - Evaluate the model

```
# Evaluate the model
score = model.evaluate(X_test, y_test, verbose=1)
print('Test Loss:', score[0])
print('Test Accuracy:', score[1])
```

```
313/313 [==============================] - 17s 54ms/step - loss: 0.0201 - accuracy: 0.9943
Test Loss: 0.02005588449537754
Test Accuracy: 0.9943000078201294
```

6 - Show accuracy and loss plots of the model

```
# Show accuracy and loss plots of the model
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(accuracy))

plt.figure(figsize=(10,5))
ax = plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()

ax = plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```



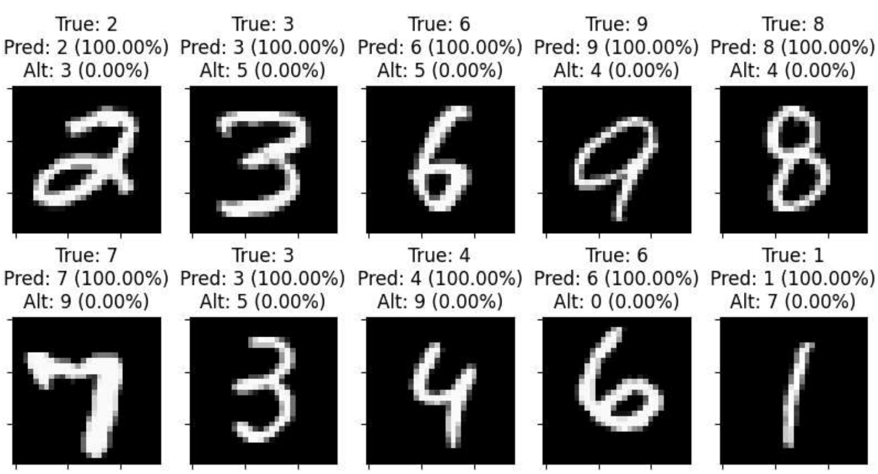
7 - Predict

```
# Obtain predictions. We get the predicted class and the second most probable class, with their probabilities
predicted_class = model.predict(X_test)
predicted_second_class = predicted_class.copy()

predicted_class_probability = np.max(predicted_class, axis=1)*100
predicted_class = np.argmax(predicted_class, axis=1)
true_class = np.argmax(y_test, axis=1)

# In 'predicted_second_class' we set the largest value to 0, to find the second largest value
for i in range(predicted_second_class.shape[0]):
    predicted_second_class[i, np.argmax(predicted_second_class[i])] = 0

predicted_second_class_probability = np.max(predicted_second_class, axis=1)*100
predicted_second_class = np.argmax(predicted_second_class, axis=1)

correct = []
incorrect = []

for i in range(len(predicted_class)):
    if predicted_class[i] == true_class[i]:
        correct.append(i)
    else:
        incorrect.append(i)

print('Correct predictions: ', len(correct))
print('Incorrect predictions: ', len(incorrect))

random.shuffle(correct)
random.shuffle(incorrect)
```

```
313/313 [==============================] - 14s 43ms/step
Correct predictions:  9943
Incorrect predictions:  57
```

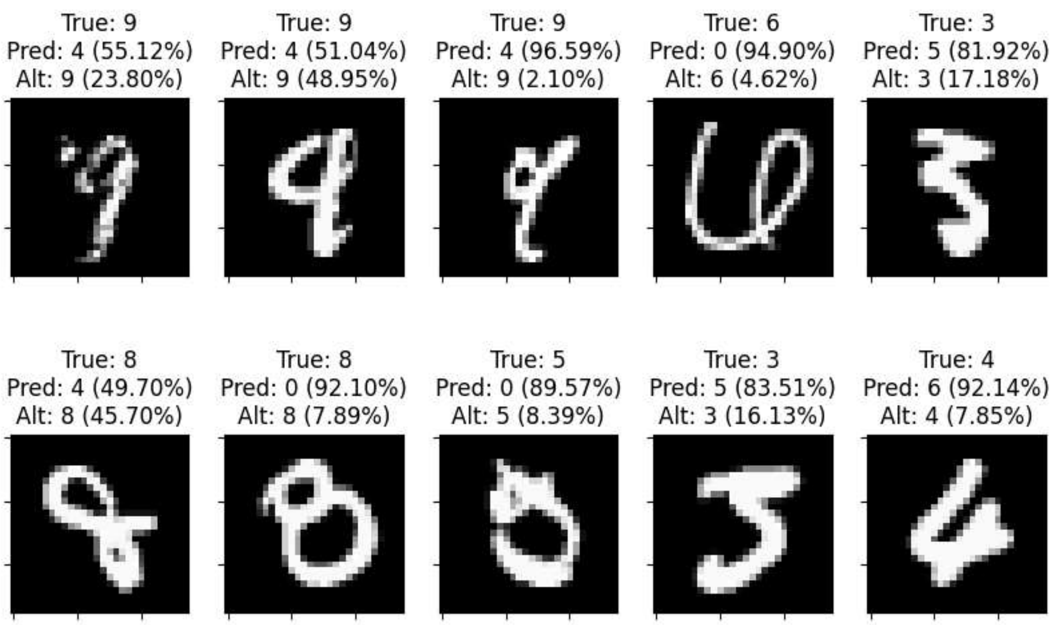
8 - Show CORRECT predictions

```
# Show some CORRECT PREDICTIONS
n_rows = 2
n_cols = 5

plt.figure(figsize=(10,5))
for i in range(n_rows*n_cols):
    ax = plt.subplot(n_rows, n_cols, i + 1)
    ax.set_yticklabels([])
    ax.set_xticklabels([])
    plt.imshow(X_test[correct[i]], cmap=plt.get_cmap('gray'))
    plt.title('True: ' + str(true_class[correct[i]]) +
              '\nPred: ' + str(predicted_class[correct[i]]) + " (%.2f%%)" % predicted_class_probability[correct[i]] +
              '\nAlt: ' + str(predicted_second_class[correct[i]]) + " (%.2f%%)" % predicted_second_class_probability[correct[i]])
```



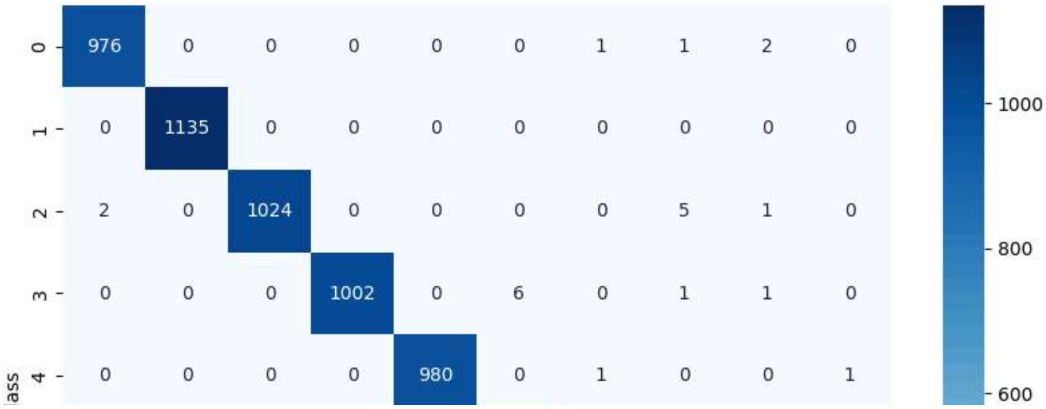
9 - Show INCORRECT predictions

```
# Show some INCORRECT PREDICTIONS
n_rows = 2
n_cols = 5

plt.figure(figsize=(10, 3*n_rows))
for i in range(n_rows*n_cols):
    ax = plt.subplot(n_rows, n_cols, i + 1)
    ax.set_yticklabels([])
    ax.set_xticklabels([])
    plt.imshow(X_test[incorrect[i]], cmap=plt.get_cmap('gray'))
    plt.title('True: ' + str(true_class[incorrect[i]]) +
              '\nPred: ' + str(predicted_class[incorrect[i]]) + " (%.2f%%)" % predicted_class_probability[incorrect[i]] +
              '\nAlt: ' + str(predicted_second_class[incorrect[i]]) + " (%.2f%%)" % predicted_second_class_probability[incorrect[i]])
```



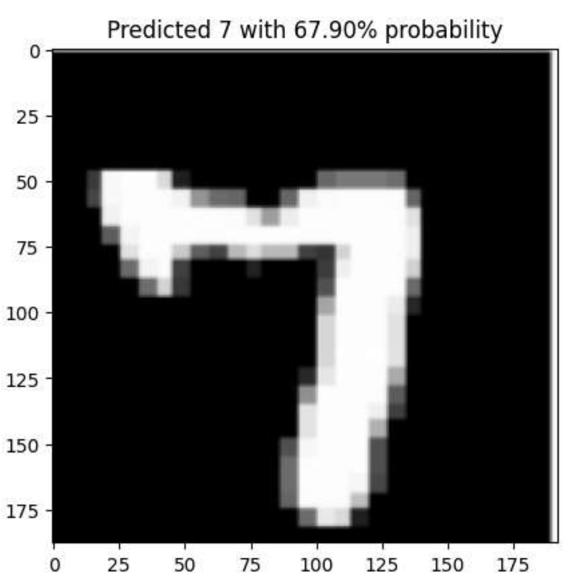
10 - Show Confussion Matrix

```
# Show confussion Matrix
cm = confusion_matrix(true_class, predicted_class)
plt.subplots(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt=".0f", cmap='Blues')
plt.xlabel("Predicted class")
plt.ylabel("True class")
plt.show()
```



11 - Test model with "real" handwritten text

```
# Test model with real handwritten text
files = ['/Screenshot 2024-03-31 160013.png']

for file in files:
    # Open, resize and reshape images
    img = cv2.imread(file)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_gray = cv2.resize(img_gray, (28, 28))
    img_gray = 255 - img_gray
    img_gray = img_gray.reshape(28,28,1)

    # Obtain predictions
    predicted_class = model.predict(img_gray[None,:,:], verbose=0)
    predicted_class_probability = np.max(predicted_class, axis=1)*100
    predicted_class = np.argmax(predicted_class, axis=1)

    # Show original images and predictions
    plt.imshow(img_rgb)
    plt.title('Predicted ' + str(predicted_class[0]) +  ' with %.2f%%' % predicted_class_probability[0] + ' probability')
    plt.show()
```



Start coding or generate with AI.