In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries ins
talled
# It is defined by the kaggle/python docker image: https://github.com/kaggl
e/docker-python
# For example, here's several helpful packages to load in

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline
import tensorflow as tf
np.random.seed(2)

from sklearn.model_selection import  train_test_split
from sklearn.metrics import confusion_matrix
import itertools

from keras.utils.np_utils import to_categorical # convert to one-hot-encod
ing
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will
 list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

```
/kaggle/input/digit-recognizer/train.csv
/kaggle/input/digit-recognizer/test.csv
/kaggle/input/digit-recognizer/sample_submission.csv


Using TensorFlow backend.
```

In [2]:
```
train=pd.read_csv('/kaggle/input/digit-recognizer/train.csv')
test=pd.read_csv('/kaggle/input/digit-recognizer/test.csv')
```

In [3]:
```
train.shape
```

Out[3]:
```
(42000, 785)
```

In [4]:
```
test.shape
```

Out[4]:
```
(28000, 784)
```

In [5]:
```
X_train = train.drop('label', axis=1)
y_train = train['label']
X_test = test
```

In [6]:
```
X_train=X_train/255
X_test=X_test/255
```

In [7]:
```
y_train
```

Out[7]:
```
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
```

In [8]:
```
y_train=to_categorical(y_train, num_classes = 10)
```

In [9]:
```
X_train = X_train.values.reshape(-1,28,28,1)
X_test = X_test.values.reshape(-1,28,28,1)
```

In [10]:
```
X_train.shape,y_train.shape
```

Out[10]:
```
((42000, 28, 28, 1), (42000, 10))
```

In [11]:
```
x_train, x_val, y_train, y_val = train_test_split(X_train, y_train, test_
size = 0.1, random_state = 2)
```

In [12]:
```
model = Sequential()

model.add(Conv2D(filters = 20, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters = 40, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters = 150, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Flatten())
model.add(Dense(10, activation = "softmax"))
```

In [13]:
```
model.compile(optimizer = 'rmsprop' , loss = "categorical_crossentropy",
metrics=["accuracy"])
```

In [14]:

```
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size = 128,
                    validation_data = (x_val, y_val))
```

```
Train on 37800 samples, validate on 4200 samples
Epoch 1/20
37800/37800 [==============================] - 5s 144us/step - loss:
0.2112 - accuracy: 0.9332 - val_loss: 0.0903 - val_accuracy: 0.9705
Epoch 2/20
37800/37800 [==============================] - 2s 46us/step - loss: 0.
0537 - accuracy: 0.9833 - val_loss: 0.0559 - val_accuracy: 0.9826
Epoch 3/20
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0357 - accuracy: 0.9885 - val_loss: 0.0433 - val_accuracy: 0.9888
Epoch 4/20
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0265 - accuracy: 0.9915 - val_loss: 0.0327 - val_accuracy: 0.9900
Epoch 5/20
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0190 - accuracy: 0.9935 - val_loss: 0.0353 - val_accuracy: 0.9898
Epoch 6/20
37800/37800 [==============================] - 2s 55us/step - loss: 0.
0157 - accuracy: 0.9953 - val_loss: 0.0323 - val_accuracy: 0.9905
Epoch 7/20
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0120 - accuracy: 0.9962 - val_loss: 0.0315 - val_accuracy: 0.9919
Epoch 8/20
37800/37800 [==============================] - 2s 46us/step - loss: 0.
0094 - accuracy: 0.9971 - val_loss: 0.0329 - val_accuracy: 0.9921
Epoch 9/20
37800/37800 [==============================] - 2s 51us/step - loss: 0.
0082 - accuracy: 0.9973 - val_loss: 0.0355 - val_accuracy: 0.9926
Epoch 10/20
37800/37800 [==============================] - 2s 48us/step - loss: 0.
0065 - accuracy: 0.9980 - val_loss: 0.0307 - val_accuracy: 0.9924
Epoch 11/20
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0041 - accuracy: 0.9986 - val_loss: 0.0589 - val_accuracy: 0.9876
Epoch 12/20
37800/37800 [==============================] - 2s 50us/step - loss: 0.
0043 - accuracy: 0.9986 - val_loss: 0.0581 - val_accuracy: 0.9886
Epoch 13/20
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0035 - accuracy: 0.9988 - val_loss: 0.0431 - val_accuracy: 0.9924
Epoch 14/20
```

```
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0040 - accuracy: 0.9989 - val_loss: 0.0471 - val_accuracy: 0.9914
Epoch 15/20
37800/37800 [==============================] - 2s 48us/step - loss: 0.
0020 - accuracy: 0.9994 - val_loss: 0.0547 - val_accuracy: 0.9926
Epoch 16/20
37800/37800 [==============================] - 2s 48us/step - loss: 0.
0027 - accuracy: 0.9992 - val_loss: 0.0421 - val_accuracy: 0.9917
Epoch 17/20
37800/37800 [==============================] - 2s 46us/step - loss: 0.
0014 - accuracy: 0.9996 - val_loss: 0.0588 - val_accuracy: 0.9912
Epoch 18/20
37800/37800 [==============================] - 2s 48us/step - loss: 0.
0016 - accuracy: 0.9995 - val_loss: 0.0653 - val_accuracy: 0.9910
Epoch 19/20
37800/37800 [==============================] - 2s 46us/step - loss: 9.
3602e-04 - accuracy: 0.9997 - val_loss: 0.0621 - val_accuracy: 0.9936
Epoch 20/20
37800/37800 [==============================] - 2s 47us/step - loss: 0.
0022 - accuracy: 0.9992 - val_loss: 0.0758 - val_accuracy: 0.9907
```

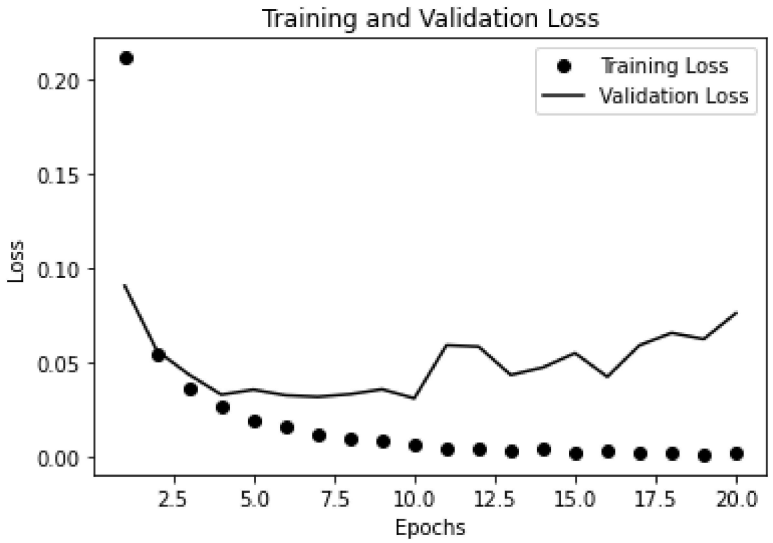
In [15]:

```
loss = history.history['loss']
val_loss = history.history['val_loss']
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, 21)

plt.plot(epochs, loss, 'ko', label = 'Training Loss')
plt.plot(epochs, val_loss, 'k', label = 'Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
```

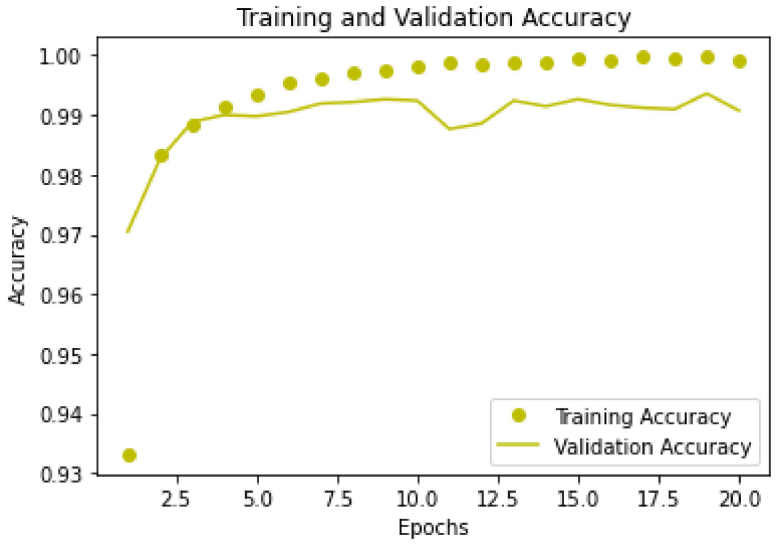
Out[15]:

```
<matplotlib.legend.Legend at 0x7f9513c18358>
```



In [16]:
```
plt.plot(epochs, acc, 'yo', label = 'Training Accuracy')
plt.plot(epochs, val_acc, 'y', label = 'Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
```

Out[16]:
```
<matplotlib.legend.Legend at 0x7f951202a940>
```



In [17]:
```
# predict results
results = model.predict(X_test)
```

In [18]:
```
results = np.argmax(results,axis = 1)
results = pd.Series(results,name="Label")
```

In [19]:
```
results
```

Out[19]:
```
0        2
1        0
2        9
3        9
4        3
        ..
27995    9
27996    7
27997    3
27998    9
27999    2
Name: Label, Length: 28000, dtype: int64
```

In [20]:
```
submission = pd.concat([pd.Series(range(1, 28001), name = 'ImageId'), res
ults], axis = 1)
submission.to_csv("MNIST_Dataset_Submissions.csv", index = False)
```

In [ ]:
```

```