In this exercise, you will build a convolutional neural network (CNN) to classify handwritten digits from the MNIST dataset. The steps to build a CNN classifier are outlined in section 20.15 of the Machine Learning with Python Cookbook, but keep in mind that your code may need to be modified depending on your version of Keras.

In [2]:
```
# install packages
! pip install keras
! pip install tensorflow
! pip install opencv-python
```

```
Collecting keras
  Downloading keras-2.8.0-py2.py3-none-any.whl (1.4 MB)
Installing collected packages: keras
Successfully installed keras-2.8.0
Collecting tensorflow
  Downloading tensorflow-2.8.0-cp38-cp38-win_amd64.whl (438.0 MB)
Collecting keras-preprocessing>=1.1.1
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
Requirement already satisfied: h5py>=2.9.0 in c:\users\kadams\anaconda3\lib\site-packages (from tensorflow) (2.10.0)
Collecting gast>=0.2.1
  Downloading gast-0.5.3-py3-none-any.whl (19 kB)
Requirement already satisfied: numpy>=1.20 in c:\users\kadams\anaconda3\lib\site-packages (from tensorflow) (1.20.3)
Collecting protobuf>=3.9.2
  Downloading protobuf-3.19.4-cp38-cp38-win_amd64.whl (895 kB)
Collecting grpcio<2.0,>=1.24.3
  Downloading grpcio-1.44.0-cp38-cp38-win_amd64.whl (3.4 MB)
Collecting tensorflow-io-gcs-filesystem>=0.23.1
  Downloading tensorflow_io_gcs_filesystem-0.24.0-cp38-cp38-win_amd64.whl (1.5 MB)
Collecting absl-py>=0.4.0
  Downloading absl_py-1.0.0-py3-none-any.whl (126 kB)
Collecting libclang>=9.0.1
  Downloading libclang-13.0.0-py2.py3-none-win_amd64.whl (13.9 MB)
Collecting astunparse>=1.6.0
  Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Collecting flatbuffers>=1.12
  Downloading flatbuffers-2.0-py2.py3-none-any.whl (26 kB)
Collecting tensorboard<2.9,>=2.8
  Downloading tensorboard-2.8.0-py3-none-any.whl (5.8 MB)
Collecting opt-einsum>=2.3.2
  Downloading opt_einsum-3.3.0-py3-none-any.whl (65 kB)
Collecting termcolor>=1.1.0
  Downloading termcolor-1.1.0.tar.gz (3.9 kB)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\kadams\anaconda3\lib\site-packages (from tensorflow) (1.12.1)
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in c:\users\kadams\anaconda3\lib\site-packages (from tensorflow) (2.8.0)
Requirement already satisfied: setuptools in c:\users\kadams\anaconda3\lib\site-packages (from tensorflow) (58.0.4)
Requirement already satisfied: six>=1.12.0 in c:\users\kadams\anaconda3\lib\site-packages (from tensorflow) (1.16.0)
Collecting tf-estimator-nightly==2.8.0.dev2021122109
  Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl (462 kB)
Collecting google-pasta>=0.1.1
  Downloading google_pasta-0.2.0-py3-none-any.whl (57 kB)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\kadams\anaconda3\lib\site-packages (from tensorflow) (3.10.0.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\kadams\anaconda3\lib\site-packages (from astunparse>=1.6.0->tensorflow) (0.37.1)
Collecting tensorboard-plugin-wit>=1.6.0
  Downloading tensorboard_plugin_wit-1.8.1-py3-none-any.whl (781 kB)
Collecting google-auth<3,>=1.6.3
  Downloading google_auth-2.6.0-py2.py3-none-any.whl (156 kB)
Requirement already satisfied: werkzeug>=0.11.15 in c:\users\kadams\anaconda3\lib\site-packages (from tensorboard<2.9,>=2.8->tensorflow) (2.0.2)
Collecting markdown>=2.6.8
  Downloading Markdown-3.3.6-py3-none-any.whl (97 kB)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\kadams\anaconda3\lib\site-packages (from tensorboard<2.9,>=2.8->tensorflow) (2.27.1)
Collecting google-auth-oauthlib<0.5,>=0.4.1
  Downloading google_auth_oauthlib-0.4.6-py2.py3-none-any.whl (18 kB)
Collecting tensorboard-data-server<0.7.0,>=0.6.0
  Downloading tensorboard_data_server-0.6.1-py3-none-any.whl (2.4 kB)
Collecting pyasn1-modules>=0.2.1
  Downloading pyasn1_modules-0.2.8-py2.py3-none-any.whl (155 kB)
Collecting rsa<5,>=3.1.4
  Downloading rsa-4.8-py3-none-any.whl (39 kB)
```

```
Collecting cachetools<6.0,>=2.0.0
  Downloading cachetools-5.0.0-py3-none-any.whl (9.1 kB)
Collecting requests-oauthlib>=0.7.0
  Downloading requests_oauthlib-1.3.1-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: importlib-metadata>=4.4 in c:\users\kadams\anaconda3\lib\site-packages (from markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow) (4.8.2)
Requirement already satisfied: zipp>=0.5 in c:\users\kadams\anaconda3\lib\site-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow) (3.7.0)
Collecting pyasn1<0.5.0,>=0.4.6
  Downloading pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\kadams\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\kadams\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\kadams\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\kadams\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow) (1.26.8)
Collecting oauthlib>=3.0.0
  Downloading oauthlib-3.2.0-py3-none-any.whl (151 kB)
Building wheels for collected packages: termcolor
  Building wheel for termcolor (setup.py): started
  Building wheel for termcolor (setup.py): finished with status 'done'
  Created wheel for termcolor: filename=termcolor-1.1.0-py3-none-any.whl size=4848 sha256=97e093c9019d92534d7c1740a202b36b507d47c864d219cf9709c1c96ad258d6
  Stored in directory: c:\users\kadams\appdata\local\pip\cache\wheels\a0\16\9c\5473df82468f958445479c59e784896fa24f4a5fc024b0f501
Successfully built termcolor
Installing collected packages: pyasn1, rsa, pyasn1-modules, oauthlib, cachetools, requests-oauthlib, google-auth, tensorboard-plugin-wit, tensorboard-data-server, protobuf, markdown, grpcio, google-auth-oauthlib, absl-py, tf-estimator-nightly, termcolor, tensorflow-io-gcs-filesystem, tensorboard, opt-einsum, libclang, keras-preprocessing, google-pasta, gast, flatbuffers, astunparse, tensorflow
Successfully installed absl-py-1.0.0 astunparse-1.6.3 cachetools-5.0.0 flatbuffers-2.0 gast-0.5.3 google-auth-2.6.0 google-auth-oauthlib-0.4.6 google-pasta-0.2.0 grpcio-1.44.0 keras-preprocessing-1.1.2 libclang-13.0.0 markdown-3.3.6 oauthlib-3.2.0 opt-einsum-3.3.0 protobuf-3.19.4 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-oauthlib-1.3.1 rsa-4.8 tensorboard-2.8.0 tensorboard-data-server-0.6.1 tensorboard-plugin-wit-1.8.1 tensorflow-2.8.0 tensorflow-io-gcs-filesystem-0.24.0 termcolor-1.1.0 tf-estimator-nightly-2.8.0.dev2021122109
Collecting opencv-python
  Downloading opencv_python-4.5.5.62-cp36-abi3-win_amd64.whl (35.4 MB)
Requirement already satisfied: numpy>=1.17.3 in c:\users\kadams\anaconda3\lib\site-packages (from opencv-python) (1.20.3)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.5.5.62
```

In [177…

```python
# import libraries (pg. 327)
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from keras.datasets import mnist
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras import utils as np_utils
from keras import backend as K
from matplotlib import pyplot
from tensorflow import keras
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay, accuracy_score, precision_score, recall_score, f1_score
```

Load the MNIST data set.

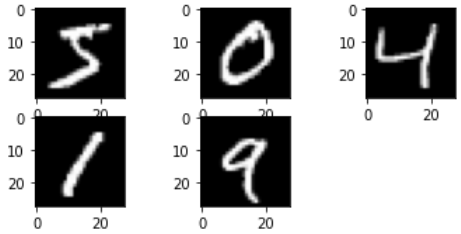
In [147…

```python
# Load data and target from MNIST data
(data_train, target_train), (data_test, target_test) = mnist.load_data()
```

Display the first five images in the training data set (see section 8.1 in the Machine Learning with Python Cookbook). Compare these to the first five training labels.

In [148…

```python
# Display first 5 images in training data set
for i in range(5):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # plot raw pixel data
    pyplot.imshow(data_train[i], cmap=pyplot.get_cmap('gray'))
# show the figure
pyplot.show()
```



In [149…
```
# Display first five labels
pd.DataFrame(target_train[0:5]).set_axis(['label'], axis=1)
```

Out[149…

	label
0	5
1	0
2	4
3	1
4	9

Build and train a Keras CNN classifier on the MNIST training set.

In [150…
```
# Set that the color channel value will be last
K.set_image_data_format("channels_last")
```

In [151…
```
# Set seed
np.random.seed(0)
```

In [152…
```
# Set image information
channels = 1
height = 28
width = 28
```

In [153…
```
# reshape dataset to have a single channel
data_train = data_train.reshape((data_train.shape[0], 28, 28, 1)) # height, width, channel
data_test = data_test.reshape((data_test.shape[0], 28, 28, 1)) # height, width, channel
```

In [154…
```
# Rescale pixel intensity to between 0 and 1
features_train = data_train / 255
features_test = data_test / 255
```

In [155…
```
# One-hot encode target
target_train = keras.utils.to_categorical(target_train)
target_test = keras.utils.to_categorical(target_test)
number_of_classes = target_test.shape[1]
```

In [156…
```
# Start neural network
network = Sequential()
```

In [157…
```
# Add convolutional layer with 64 filters, a 5x5 window, and ReLU activation function
network.add(Conv2D(filters=64,
 kernel_size=(5, 5),
```

```
 input_shape=(height, width, channels),
 activation='relu'))
```

In [158…]
```
# Add max pooling layer with a 2x2 window
network.add(MaxPooling2D(pool_size=(2, 2)))
```

In [159…]
```
# Add dropout layer
network.add(Dropout(0.5))
```

In [160…]
```
# Add layer to flatten input
network.add(Flatten())
```

In [161…]
```
# Add fully connected layer of 128 units with a ReLU activation function
network.add(Dense(128, activation="relu"))
```

In [162…]
```
# Add dropout layer
network.add(Dropout(0.5))
```

In [163…]
```
# Add fully connected layer with a softmax activation function
network.add(Dense(number_of_classes, activation="softmax"))
```

In [164…]
```
# Compile neural network
network.compile(loss="categorical_crossentropy", # Cross-entropy
 optimizer="rmsprop", # Root Mean Square Propagation
 metrics=["accuracy"]) # Accuracy performance metric
```

In [165…]
```
# Train neural network
network.fit(features_train, # Features
 target_train, # Target
 epochs=2, # Number of epochs
 verbose=0, # Don't print description after each epoch
 batch_size=1000, # Number of observations per batch
 validation_data=(features_test, target_test)) # Data for evaluation
```

Out[165…]
```
<keras.callbacks.History at 0x1de20c95880>
```

In [166…]
```
network.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 24, 24, 64)        1664

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_6 (Dropout)         (None, 12, 12, 64)        0

 flatten_3 (Flatten)         (None, 9216)              0

 dense_6 (Dense)             (None, 128)               1179776

 dropout_7 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 10)                1290

=================================================================
Total params: 1,182,730
```

```
Trainable params: 1,182,730
Non-trainable params: 0
_________________________________________________________________
```

In [167…
```
# Use model to make predictions on test set
pred = network.predict(data_test)
```

In [168…
```
# Convert values for confusion matrix
y_preds = np.argmax(pred, axis=1)
```

In [169…
```
# Double check target_test shape
target_test
```

Out[169…
```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 1., ..., 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.]], dtype=float32)
```

In [170…
```
# Convert values for confusion matrix
y_true = np.argmax(target_test, axis=1)
```

Report the test accuracy of your model.

In [171…
```
## Calculate and print stats (from Mike Zoucha)
accuracy = accuracy_score(y_true, y_preds)
precision = precision_score(y_true, y_preds, average='micro')
recall = recall_score(y_true, y_preds, average='micro')
f1 = f1_score(y_true, y_preds, average='micro')
```

In [172…
```
## Print classification report
print(classification_report(y_true, y_preds))
```

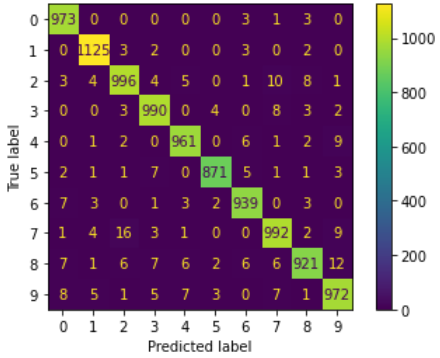
```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.97      0.97      0.97      1032
           3       0.97      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.98      0.98      0.98       958
           7       0.97      0.96      0.97      1028
           8       0.97      0.95      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

Display a confusion matrix on the test set classifications.

In [178…
```
## Visualize confusion matrix
conf_matrix = confusion_matrix(y_true, y_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=[0,1,2,3,4,5,6,7,8,9])
disp.plot()
plt.show()
```



Summarize your results.

The training set is used to perform the initial training of the model and initialize the weights of the neural network. Once the model is trained and parameter/arcitecture selection is complete with the training set, the test set is used to test the predictive accuracy of the trained neural network on previously unseen data. With this particular test set, the accuracy is 97% (i.e., for every 100 test set examples, the model classifies 97 of them correctly).

In regards to the confusion matrix of the test set classifications, the majority of classifications were True Positivities (the predicted value matches the actual/true value) for each label. This aligns with the high test set accuracy of 97%.

In [ ]: