Objective - Create a ML system that detects and classifies hand-written digits

Import modules and prepare dataset

In [1]:

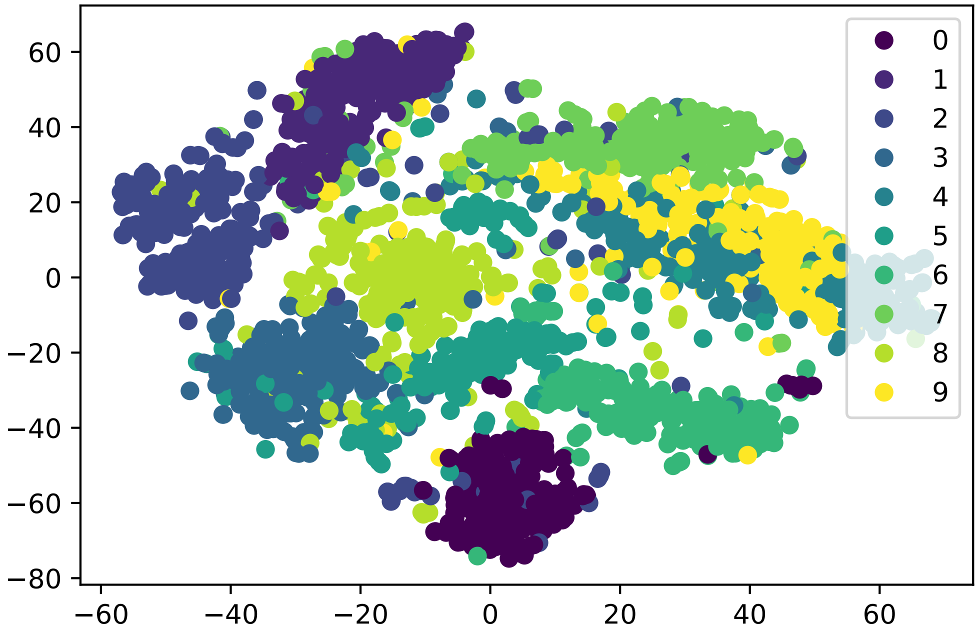
```
# Import package.
from package.package import *

# Import dataset.
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# initializing data type as unsinged int (non-negative integer).
X_train = X_train.reshape(X_train.shape[0], -1).astype('uint8')
X_test = X_test.reshape(X_test.shape[0], -1).astype('uint8')

# X represents the hand written digits which are 28 x 28 in size.
print(f'X_train.shape: {X_train.shape}')    # -> (60,000, 784)

# Y is the actual digits they represent.
print(f'y_train.shape: {y_train.shape}')    # -> (60,000,)

print(f'X_test.shape: {X_test.shape}')    # -> (10,000, 784)
print(f'y_test.shape: {y_test.shape}')    # -> (10,000,)
```

```
X_train.shape: (60000, 784)
y_train.shape: (60000,)
X_test.shape: (10000, 784)
y_test.shape: (10000,)
```

Visualize the data using TSNE dimensionality technique

In [3]:

```
# Initializing data reduction algorithm
tsne = TSNE()

# Change the data accordingly to the algorithm.
X_test_trans = tsne.fit_transform(X_test[:2500])

# Scatter the data.
scatter = plt.scatter(X_test_trans[:, 0], X_test_trans[:, 1], c=y_test[:2500])
plt.legend(*scatter.legend_elements())
plt.show()

# Data is formed in clusters and looks to be linearly seperable.
```



Data analysis

In [2]:

```
# Concatenating both train and test datasets.
data = np.concatenate((X_train, X_test))
target = np.concatenate((y_train, y_test))

# Check % of data that's 0.
percent_of_zeros = np.sum(data == 0)/data.size    # -> 80%
percent_of_non_zeros = np.sum(data != 0)/data.size    # -> 20%

# Check for null values.
check_null = np.isnan(np.sum(data))    # -> False

# Create scaler.
scaler = MinMaxScaler()

# Keep sample for comparison.
sample = X_train[0]

# Scale the data.
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Compare the two samples and their values
print('Before data transformation: {}'.format(sample[np.where(sample != 0)][:5]))
print('After data transformation:  {}'.format(X_train[np.where(X_train != 0)][:5]))

# Change to categorical data.
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Change data to image data.
X_train = X_train.reshape((60000, 28, 28, 1)).astype('float32')
X_test = X_test.reshape((10000, 28, 28, 1)).astype('float32')
```

```
Before data transformation: [  3  18  18  18 126]
After data transformation:  [0.01176471 0.07058824 0.07058824 0.07058824 0.49411765]
```

Apply linear model to the data

In [3]:

```
# Creating simple linear model to learn more about data.
perceptron = Perceptron(max_iter=99999999)    # Perceptron only works when data is line

# NOTE: Perceptron is a neural network with only one hidden-layer.

# Create function that reforms the data for the algorithm.
def reform_sklearn(*arrays):

    old_arrays = arrays
    reformed_arrays = []
    print('Reforming arrays...')

    for array in arrays:

        # Array is a data variable.
        if array.ndim != 2:
            reformed_arrays.append(array.reshape(len(array), -1))

        # Array is a target variable
        else:
            reformed_arrays.append(np.array([np.argmax(sample) for sample in array]))

    print('Old array shapes: {}'.format([array.shape for array in old_arrays]))
    print('New array shapes: {}\n'.format([array.shape for array in reformed_arrays]))
    return reformed_arrays

per_X_train, per_X_test = reform_sklearn(X_train, X_test)
per_y_train, per_y_test = reform_sklearn(y_train, y_test)

# NOTE: perceptrons are not good for image processing,
#       meaning the data must be 1D, not 2D.

start_time = time()

perceptron.fit(per_X_train, per_y_train)

print(f'Perceptron train score: {perceptron.score(per_X_train, per_y_train)}')
print(f'Perceptron test score: {perceptron.score(per_X_test, per_y_test)}')
print(f'Iterations used: {perceptron.n_iter_}')
print(f'Elapsed: {(time() - start_time)/60:.2f} min.')

# NOTE: Based on the results on the perceptron and that the model used little iteration
#       to fit the data, we can assume the data is linearly seperable.
```

```
Reforming arrays...
Old array shapes: [(60000, 28, 28, 1), (10000, 28, 28, 1)]
New array shapes: [(60000, 784), (10000, 784)]

Reforming arrays...
Old array shapes: [(60000, 10), (10000, 10)]
New array shapes: [(60000,), (10000,)]

Perceptron train score: 0.9024666666666666
Perceptron test score: 0.8951
Iterations used: 24
Elapsed: 0.29 min.
```

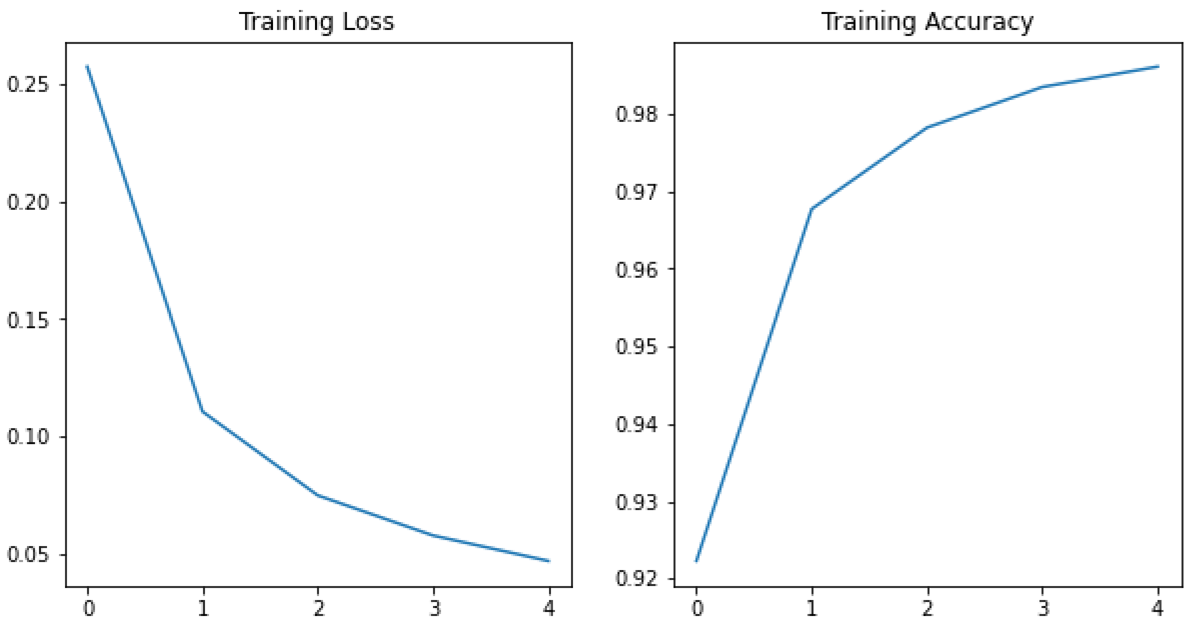
Apply model to the data

In [4]:

```python
# Create convolutional neural network.

# NOTE: Convolutional neural networks (CNNs) are used as a machine learning model for p
#       This algorithm goes thorugh different steps of processing the images in differe
#       eventually finding a pattern in the data.

# Begin cited code:
# https://rb.gy/yykaxm
def model():

    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', in
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))

    # Compile model.
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# End cited code.

# NOTE: Sequential() models allow you to make models layer-by-layer and is much simpler
#       Conv2D() is an input layer for converting an image in to a matrix.
#       MaxPooling2D() is another input layer.
#       Flatten() converts multi-dimensional data into a single vector to be processed.
#       Dense() is the main hidden layer.
#       SGD() is the stochastic gradient descent optimization function to update weight
#       relu activation function is a nonlinear function that is good for learning comp
#       softmax activation function is used to quantify the output in classification ex

# Create timer.
start_time = time()

model = model()
results = model.fit(X_train, y_train, epochs=5, batch_size=128).history

# Clear the output.
clear_output()

# NOTE: batch_size param means that the model will be tested on 64 samples at a time.
#       This save a ton of RAM during the training process.

print(train_score := 'CNN Train Score: {:.2f}'.format(model.evaluate(X_train, y_train,
print(test_score := 'CNN Test Score: {:.2f}'.format(model.evaluate(X_test, y_test, verb
print(elapsed := f'Elapsed: {(time() - start_time)/60:.2f} min.')

# Notify when done.
notification.notify(
    title='Neural Network Training Results',
    message=f'{train_score}\n{test_score}\n{elapsed}',
    app_icon='python_icon.ico'
)

fig, (ax_1, ax_2) = plt.subplots(1, 2, figsize=(10, 5))

ax_1.plot(range(5), results['loss'])
ax_1.set_title('Training Loss');
```

```
ax_2.plot(range(5), results['accuracy'])
ax_2.set_title('Training Accuracy');
```

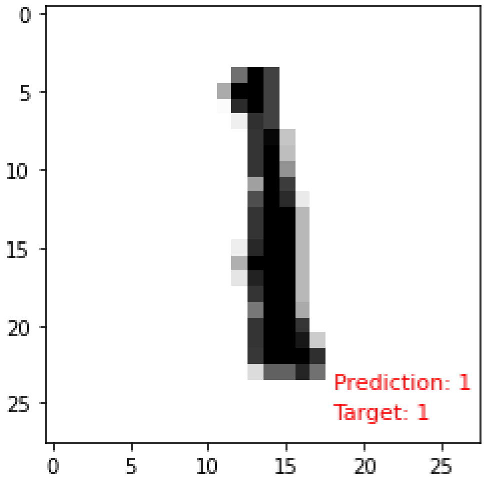
```
CNN Train Score: 99.02
CNN Test Score: 98.39
Elapsed: 1.99 min.
```



Test the model through visualizations

In [6]:

```
# Ask user for specific number to test model.
def pick_num(num):
    try:
        number = int(num)
    except ValueError:
        raise ValueError('Please choose a number')
    else:
        return number

# Call function
number = pick_num(input('Choose a number to test the model on -> '))

# Revert the target data from to_categorical()
og_y_test = np.array([np.argmax(y, axis=None, out=None) for y in y_test])

# Get the random index for the sample and the sample target.
index = choice(np.where(og_y_test == number)[0])
sample = X_test[index].reshape(1, 28, 28, 1)
sample_target = og_y_test[index]

prediction = np.argmax(model.predict(sample))

sample_img = sample.reshape(28, 28)

plt.imshow(sample_img, cmap='binary')
plt.text(18, 24, f'Prediction: {prediction}', fontsize=11, color='red');
plt.text(18, 26, f'Target: {sample_target}', fontsize=11, color='red');
```



List of dependencies

- https://www.tensorflow.org/
- https://bit.ly/3pWvxTz
- https://bit.ly/3stGnlr
- https://bit.ly/2NGYyFG
- https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
- https://bit.ly/2NJ1kdp
- https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD
- https://bit.ly/2O6ySC1
- https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist
- https://ipython.readthedocs.io/en/stable/interactive/plotting.html
- https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html
- https://numpy.org/
- https://docs.python.org/3/library/time.html
- https://github.com/kivy/plyer/blob/master/plyer/facades/notification.py
- https://docs.python.org/3/library/random.html
- https://bit.ly/3svyQml