Exercise 6: Build a neural network that will read the image of a digit and correctly identifies the digit.

Recognizing hand-written digits -- The activity is to learn from training dataset of images of hand-written digits from 0-9.

Digits dataset

The digits dataset consists of 8x8 pixel images of digits. The `images` attribute of the dataset stores 8x8 arrays of grayscale values for each image. The `target` attribute of the dataset stores the digit each image represents.

```
# Importing datasets
from sklearn import datasets

# loading digit image dataset
digits = datasets.load_digits()
```

```
# The dir() function returns all properties and methods
#                          of the specified object, without the values.
dir(digits)
```

```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

```
# Let us describe the dataset
print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998
```

```
This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

```
# Let us check the feature names
print(digits.feature_names)
```

```
['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5', 'pixel
```

```
# type() function returns the datatype of the object.
type(digits.images)
```

```
numpy.ndarray
```

```
# Let us check the target names
print(digits.target_names)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

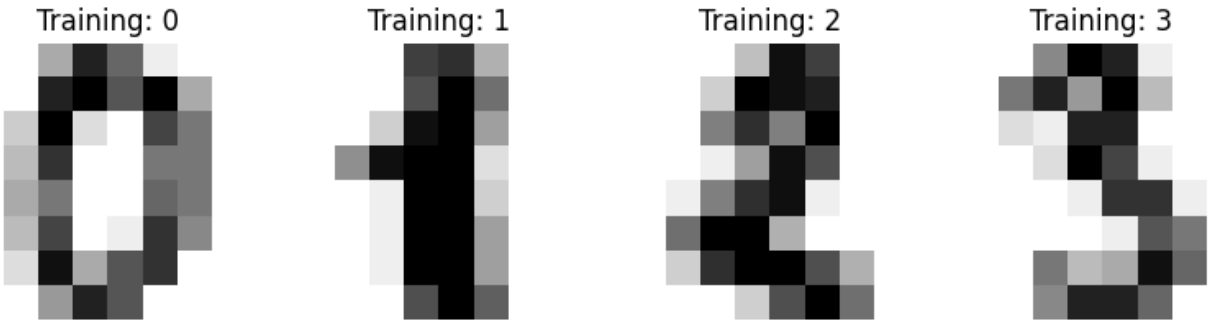
```
# Let us check the shape of the dataset
digits.images.shape
```

```
(1797, 8, 8)
```

```
# Let us check how an image is represented
digits.images[0]
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

```
# Let us display first two images using matplotlib

import matplotlib.pyplot as plt

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```



Classification using Multilayer Perceptron Networks and Backpropagation Algorithm

A simple neural network, with three layers is being constructured

(1) An input layer, with 64 nodes, one node per pixel in the input images. Nodes are neurons that actually do nothing. They just take their input value and send it to the neurons of the next layer.

(2) Two hidden layers with 128 and 64 neurons respectively. We could choose a different number, and also add more hidden layers with different numbers of neurons.

(3) An output layer with 10 neurons corresponding to our 10 classes of digits, from 0 to 9. This is a fully connected neural network, which means that each node in each layer is connected to all nodes in the previous and next layers.

To apply a classifier on this data, we need to flatten the images, turning each 2-D array of grayscale values from shape `(8, 8)` into shape `(64,)`. Subsequently, the entire dataset will be of shape `(n_samples, n_features)`, where `n_samples` is the number of images and `n_features` is the total number of pixels in each image.

Further data can be split into train and test subsets and fit an MLP classifier on the training samples. The fitted classifier can subsequently be used to predict the value of the digit for the samples in the test subset.

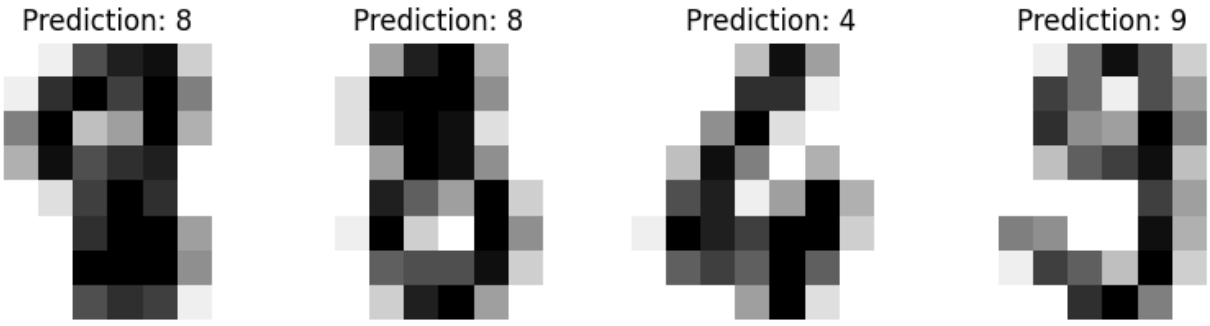
```
# flattening the images
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
```

```
from sklearn import metrics
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

# Spliting data into 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(data,
                                                    digits.target,
                                                    test_size = 0.5,
                                                    shuffle = False
                                                    )

# Creating a classifier: an MLP classifier with 2 hidden layers
#                         - hidden layer 1 with 128 neurons
#                         - hidden layer 2 with 64 neurons

mlp = MLPClassifier(hidden_layer_sizes = (128 , 64), activation = 'logistic', alpha =  1e
                    solver = 'sgd', tol = 1e-4, random_state = 1,
                    learning_rate_init = .1, verbose = True)

# Training the MLP with training data
mlp.fit(X_train, y_train)

# Predicting the value of the digit on the test data
y_predicted = mlp.predict(X_test)
```

```
Iteration 78, loss = 0.00751589
Iteration 79, loss = 0.00738729
Iteration 80, loss = 0.00725937
Iteration 81, loss = 0.00714130
Iteration 82, loss = 0.00704057
Iteration 83, loss = 0.00690226
Iteration 84, loss = 0.00681652
Iteration 85, loss = 0.00670753
Iteration 86, loss = 0.00657451
Iteration 87, loss = 0.00648019
Iteration 88, loss = 0.00638242
Iteration 89, loss = 0.00629340
Iteration 90, loss = 0.00620439
Iteration 91, loss = 0.00612568
Iteration 92, loss = 0.00602496
Iteration 93, loss = 0.00595090
Iteration 94, loss = 0.00585264
Iteration 95, loss = 0.00575984
Iteration 96, loss = 0.00570582
Iteration 97, loss = 0.00561369
Iteration 98, loss = 0.00553325
Iteration 99, loss = 0.00546445
Iteration 100, loss = 0.00539742
Iteration 101, loss = 0.00533112
Iteration 102, loss = 0.00526372
Iteration 103, loss = 0.00520110
```

```
# Below we visualize the first 4 test samples and show their predicted digit value in th
```

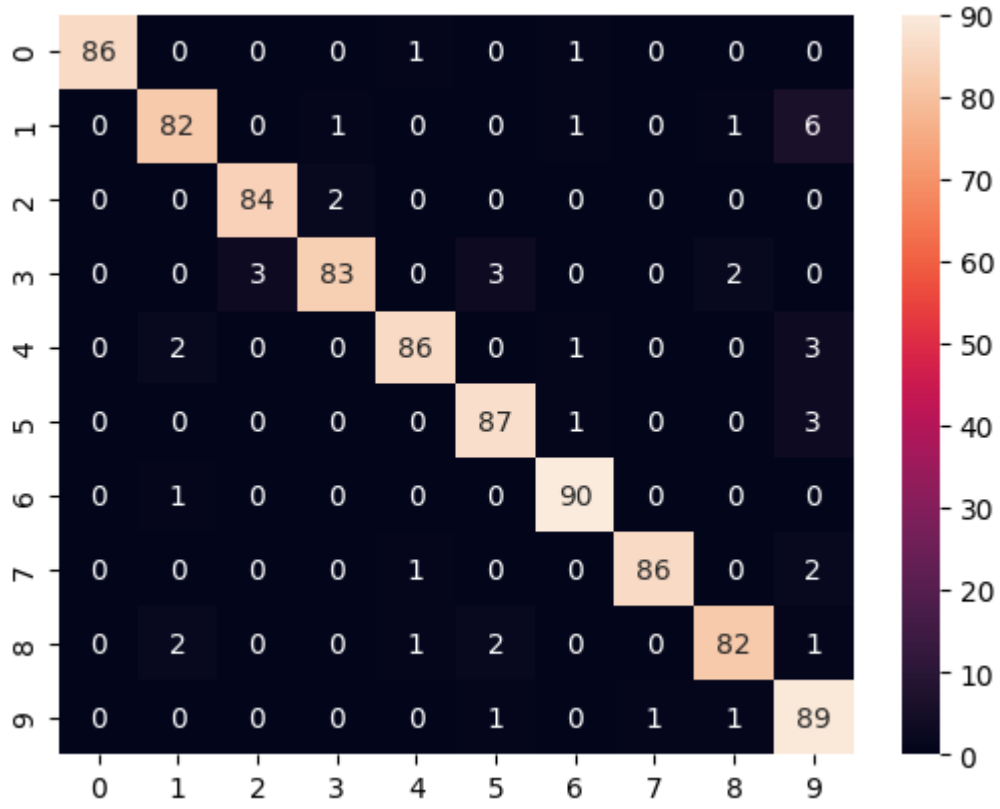
```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, y_predicted):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap = plt.cm.gray_r, interpolation = "nearest")
    ax.set_title(f"Prediction: {prediction}")
```



We can plot a `confusion matrix <confusion_matrix>` of the true digit values and the predicted digit values in the form of a heatmap.

```
from sklearn.metrics import confusion_matrix , accuracy_score
import seaborn as sns

# computing a confusion matrix
cm = confusion_matrix(y_test, y_predicted)

# displaing confusion matrix as a heatmap
sns.heatmap(cm, annot = True , fmt = 'd')

# computing and displaying the the accuracy of the KNN model
print('The % of Accuracy is :', accuracy_score(y_test , y_predicted)*100)
```

```
The % of Accuracy is : 95.10567296996662
```




```
# Visualizing weights of MLP

print("weights between input and first hidden layer:")
print(mlp.coefs_[0])

print("\nweights between first hidden and second hidden layer:")
print(mlp.coefs_[1])
```

```
weights between input and first hidden layer:
[[-0.01693263  0.04495979 -0.1020075  ...  0.00792393  0.01077894
   0.0697954 ]
 [-0.10832321 -0.05075487  0.01556038 ...  0.09515672  0.01825767
   0.05450046]
 [-0.06968199 -0.08100316 -0.02565275 ...  0.00246374  0.1093701
  -0.12351096]
```

```
 ...
 [ 0.0448075   0.10889104 -0.05133974 ...  0.03960139  0.00773755
   0.16546269]
 [ 0.13746521 -0.09864133  0.03776946 ...  0.09360828 -0.00222922
  -0.0966712 ]
 [ 0.02722291 -0.06329276 -0.08054808 ... -0.08376382  0.09110037
   0.02449374]]

weights between first hidden and second hidden layer:
[[-0.20563751  0.10088131  0.06349793 ...  0.08711656 -0.13983782
  -0.22816924]
 [ 0.14888228 -0.08867514  0.04615003 ... -0.14848029  0.01351488
   0.29328764]
 [-0.05430794  0.04978504 -0.09730345 ...  0.11565929 -0.02747184
   0.02509967]
 ...
 [-0.0275512   0.00269346 -0.07988297 ...  0.04356784 -0.10349571
   0.08501737]
 [-0.09235084 -0.05238574 -0.04346453 ...  0.08302493 -0.09771271
  -0.03994741]
 [-0.05631216 -0.05575696  0.05710291 ... -0.03214848 -0.09666771
   0.1159155 ]]
```