Image Distance

Distance-base functions for image comparisons

Euclidean Distance

It is calculated as the square root of the sum of differences between two images.

$$Euclidean=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

Manhattan Distance

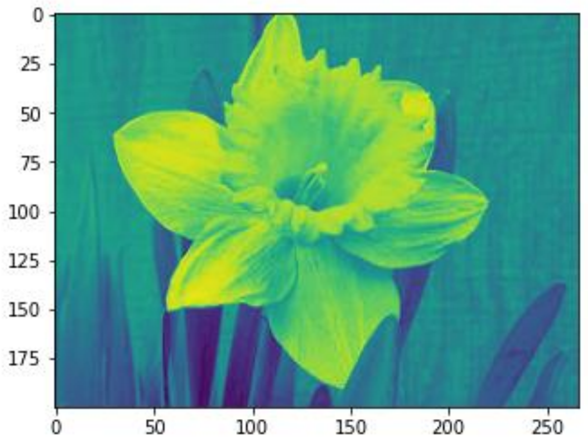
It is calculated as the sum of absolute differences between two images.

$$Manhattan=\sum_{i=1}^{n}|x_i-y_i|$$

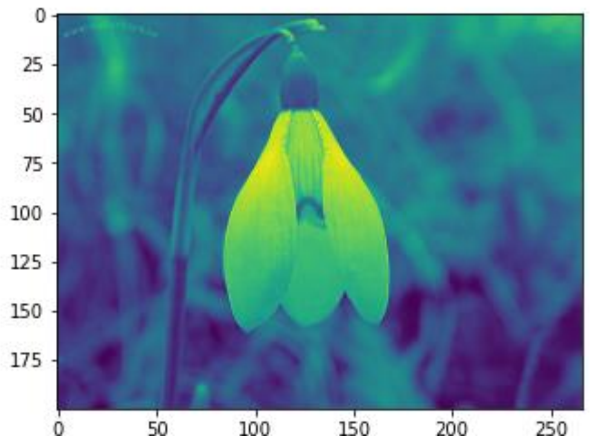
```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv
```

```
# Reading images
image1 = cv.imread("flower.jpg")
image2 = cv.imread("yellowFlower.jpg")
# Displaying images
plt.imshow(image1)
Plt.imshow(image2)
```

<matplotlib.image.AxesImage at 0x5112b3b4c0>



<matplotlib.image.AxesImage at 0x35808a65b0>




```
# Euclidean distance function
def euclidean_distance(x, y):
    distance = 0
    for a, b in zip(x, y):
        distance += pow((int(a)-int(b)), 2)
    return np.sqrt(distance)
```

```
# Manhattan distance function
def manhattan_distance(x, y):
    distance = 0
    for a, b in zip(x, y):
        distance += abs(int(a)-int(b))

    return distance
```

Image Similarity - Jaccard

Measuring the degree of similarity between patterns in two images

Jaccard Similarity

Jaccard similarity is defined as the intersection of sets divided by their union.

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

```
# Jaccard similarity

def jaccard_similarity(x, y):
    intersection = len(set(x).intersection(set(y)))
    union = len(set(x).union(set(y)))

    return intersection / union
```

Testing

```
img1 = image1.flatten()
img2 = image1.flatten()

# Jaccard Similarity test
print("Jaccard similarity: ", jaccard_similarity(img1, img2))
```

Jaccard similarity: 0.8117647058823529

Image Similarity - Cosine

Measuring the degree of similarity between patterns in two images

Cosine Similarity

It measures the cosine angle between the two vectors. If the angle between two vectors increases then they are less similar.

$$similarity(a,b) = \frac{a.b}{||a||.||b||}$$

```
# Cosine similarity function
def cosine(x, y):
    numerator = 0
    sum_x = 0
    sum_y = 0
    for a, b in zip(x, y):
        numerator += (a*b)
        sum_x += (a**2)
        sum_y += (b**2)

    denominator = round(np.sqrt(sum_x) * np.sqrt(sum_y))
    return numerator / denominator
```

Testing on images

```
img1 = image1.flatten()
img2 = image1.flatten()

# Cosine Similarity test
print("Jaccard similarity: ", cosine (img1, img2))
```

Cosine similarity: 0.007010542688668912

Correlation between images

Pearson Correlation

Pearson Correlation

Pearson's correlation measures the statistical relationship, or association, between two continuous variables. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = Correlation coefficient
X(i) = Values of the x-variable in a sample
Xbar = Mean of the values of the x-variable
Y(i) = Values of the y-variable in a sample
Ybar = Mean of the values of the y-variable

```
def correlation(image1, image2):
    img1 = image1.flatten()
    img2 = image2.flatten()
    numerator = 0
    r_denom_x = 0
    r_denom_y = 0
    mean_num_x = 0
    mean_num_y = 0
```

```
# Mean
for a, b in zip(img1, img2):
    mean_num_x += a
    mean_num_y += b

mean_x = mean_num_x / len(img1)
mean_y = mean_num_x / len(img2)
```

```
# Correlation
for a, b in zip(img1, img2):
    numerator += ((a-mean_x) * (b-mean_y))
    r_denom_x += pow((a-mean_x), 2)
    r_denom_y += pow((b-mean_y), 2)

denominator = np.sqrt(r_denom_x * r_denom_y)

return numerator / denominator
```

Correlation between images

Pearson Correlation

```
# Reading images
image1 = cv.imread("flower.jpg", cv.IMREAD_GRAYSCALE)
image2 = cv.imread("yellowFlower.jpg", cv.IMREAD_GRAYSCALE)

# No need to flatten images
# Pearson Correlation between images
print("Pearson Correlation: ", correlation(image1, image2))
```

Pearson Correlation: 0.22270457079177827

Note:
When the value of the correlation lies below + .29, then it is said to be a small correlation (low degree).

Regression Parameter Estimation

Gradient Decent

We have these three equations.

1- The hypothesis equation for line.

$$h(X) = \theta_0 + \theta_1 X$$

2- The cost function I explained above.

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

3- The gradient descends we just saw.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial\theta} J(\theta_0, \theta_1)$$

After substituting 1st in the 2nd, and then 2nd in the 3rd we get:

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{1}^{m}(h(x^{(i)}) - y^{(i)})x^{(i)}$$

Regression Parameter Estimation

Gradient Decent

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

# Reading images
image1 = cv.imread("flower.jpg")
image2 = cv.imread("yellowFlower.jpg")
```

```
# Cumputing cost function
def compute_cost(X, y, theta):
    error = 0

    for i in range(m):
        hypothesis = theta[0] + X[i][1] * theta[1]
        error += (hypothesis - y[i])**2

    #linear algebra way
    # hypothesis = np.dot(X, theta) #dot product
    # error = np.sum((hypothesis - y) ** 2)

    return error / (2 * m)
```

```
# Computing gradient decent function
def gradient_descent(X, y, theta, iterations, alpha):
    m = len(X)
    cost_history = []

    for i in range(iterations):
        hypothesis = np.matmul(X, theta)

        for j in range(theta.shape[0]):
            theta[j] = theta[j] - alpha * np.sum((hypothesis - y) * X[:, j]) / m

        cost_history.append(compute_cost(X, y, theta))

    return theta, cost_history
```

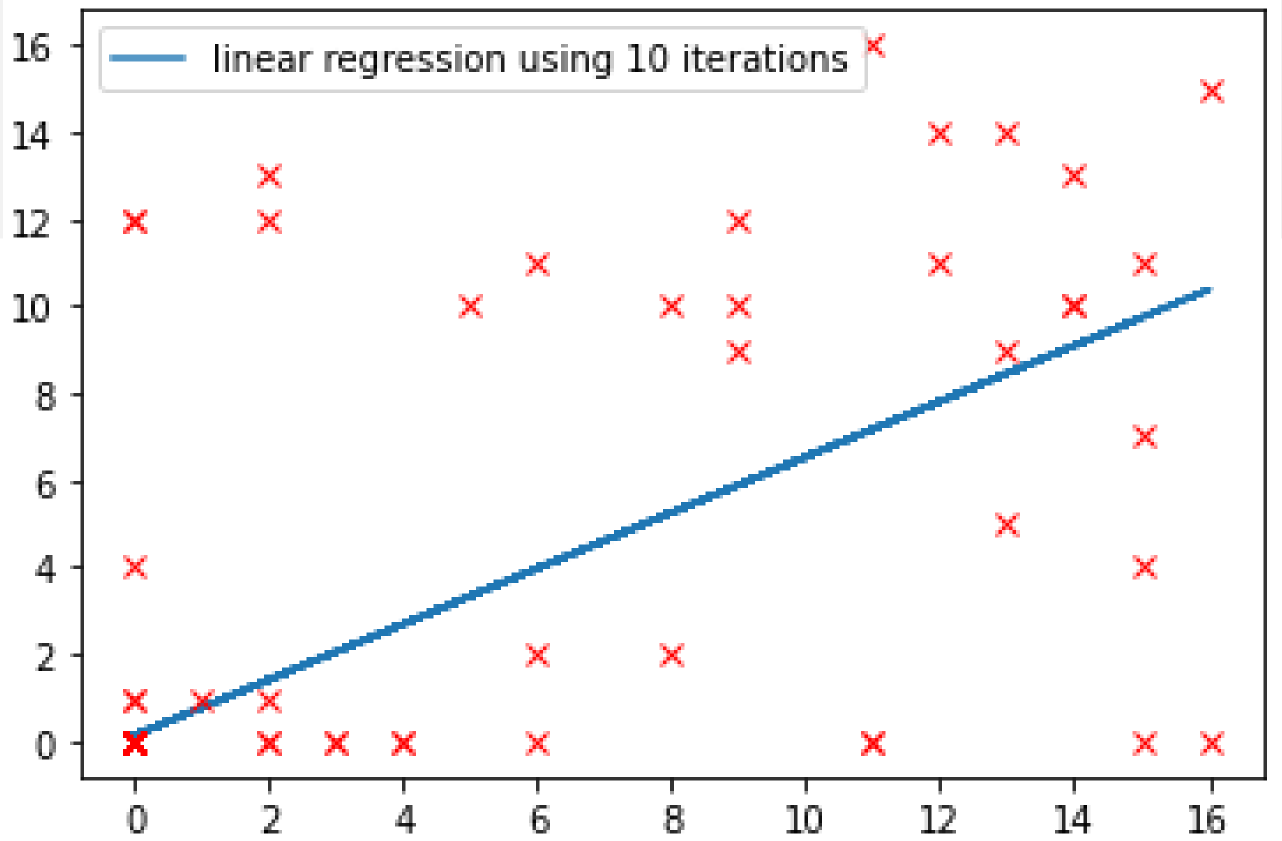
Regression Parameter Estimation

Gradient Decent

```
# Initialize important variables
def function(image1, image2):
   img1 = image1.flatten()
   img2 = image2.flatten()
   m = img1.shape[0]
   alpha = 0.01

   for iterations in [10]:
      theta = np.zeros(2)

      # Evaluation
      # Create the bias column, set to 1
      X = np.stack((np.ones(m), img1), axis=-1)
```

```
      # Second column of data
      y = img2

      theta, cost_history = gradient_descent(X, y, theta, iterations, alpha)

      plt.plot(img1, np.matmul(X,theta), '-', label='LR: %d iterations' % iterations)

      print('Theta found by gradient descent', theta);

   plt.plot(img1, img2, 'rx')

   plt.legend()
   plt.show()
```

Theta found by gradient descent [0.1335297 0.63995682]



Distance and Histogram

Computing direct distance then Histogram (Task 5)

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
import collections
```

```
# Computing distance
def distance_pixels(image1, image2):
    img1 = image1
    img2 = image2
    distance = 0

    for a, b in zip(img1, img2):
        distance += abs(a-b)

    return distance
```

image1 (3x3)

5	10	1
8	9	3
10	100	2

image2 (3x3)

5	10	1
8	9	3
10	100	255

```
print("Distance: ", distance_pixels(image1, image2))
```

Distance: 253

Distance and Histogram

Computing direct distance then Histogram (Task 5)

```
# Computing histogram distance

def hist_distance(img1, img2):
  img1 = img1.flatten()
  img2 = img2.flatten()
  xList = [0]*256
  yList = [0]*256

  distance = 0
  for i in range(len(img1)):
    xList[int(img1[i])] += 1
    yList[int(img2[i])] += 1

  for k in range(len(xList)):
    distance += abs(xList[k] - yList[k])

  return distance
```

image1 (3x3)

5	10	1
8	9	3
10	100	2

image2 (3x3)

5	10	1
8	9	3
10	100	255

```
print(hist_distance(image1, image2))
```

Distance: 2

Distance and Histogram

Image search with Hist function

So we want to search for the images from the digits dataset that are similar to an image source (we've chosen the digit 0 from digits datasets). To do so, we have to compute the histogram between the digit 0 and every other image of the dataset, and then plotting the top 10 images that are most similar.

For more details, you can check the notebook named **08-image-search-hist.ipynb**

MNIST Dataset

Reading and displaying images from MNIST (digits, letters and fashion)

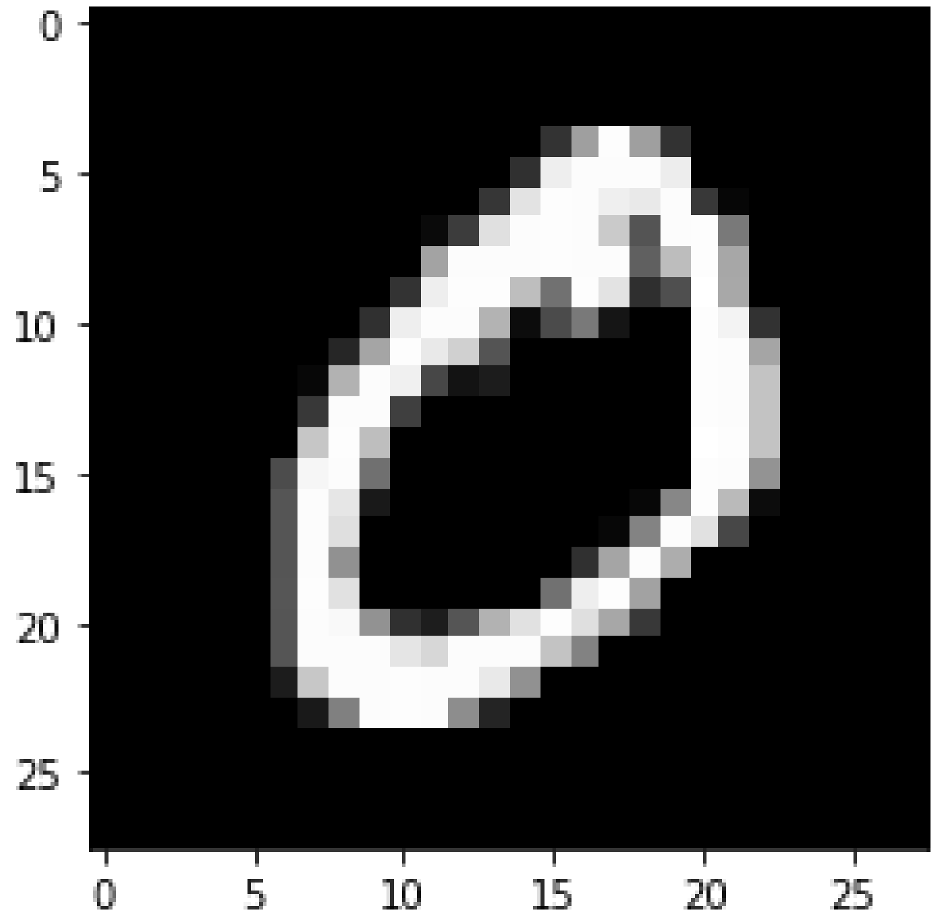
MNIST Digits

The MNIST dataset is a handwritten digit (0 through 9), 28 x 28 collection of images often used by data scientists to evaluate and compare neural network architecture performance within the literature.

```
# Importing the mnist dataset
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
import matplotlib.pyplot as plt
```

```
# Plot individual image from mnist digits
# pick a sample to plot
sample = 1
image = X_train[sample]
# plot the sample
fig = plt.figure
plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f99dbe4b4d0>
```

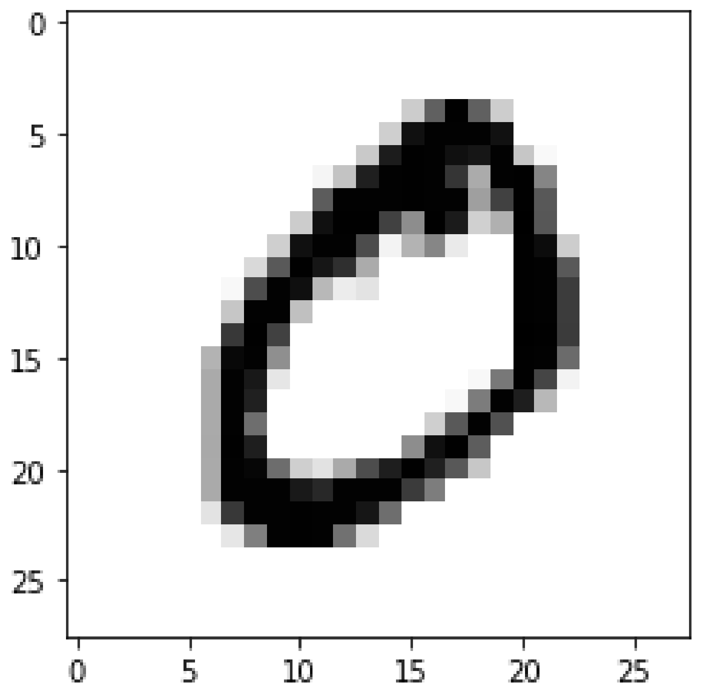


MNIST Digits

Reading and displaying images from MNIST (digits, letters and fashion)

```
# Plotting the sample (white bg)
fig = plt.figure
plt.imshow(image, cmap='gray_r')
```

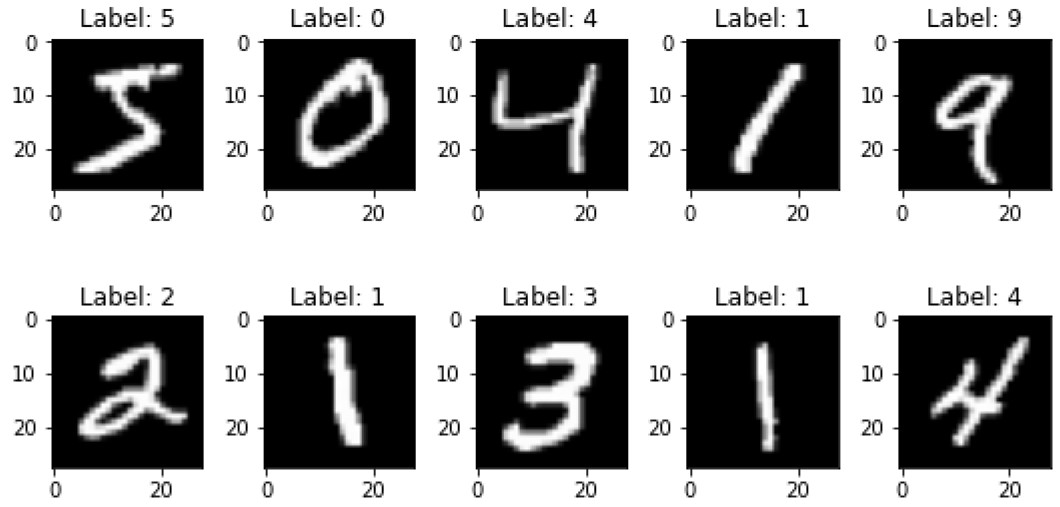
<matplotlib.image.AxesImage at 0x7f99dd10a290>



```
# Labels and images to display
num = 10
images = x_train[:num]
labels = y_train[:num]
```

```
# Display digits and labels
num_row = 2
num_col = 5

# Plotting images
fig, axes = plt.subplots(num_row, num_col, figsize = (1.5*num_col, 2*num_row))

for i in range(num) :
    ax = axes[i//num_col, i%num_col]
    ax.imshow(images[i], cmap='gray')
    ax.set_title('Label: {}'.format(labels[i]))
plt.tight_layout()
plt.show
```

<function matplotlib.pyplot.show>



MNIST Fashion

Reading and displaying images from MNIST (digits, letters and fashion)

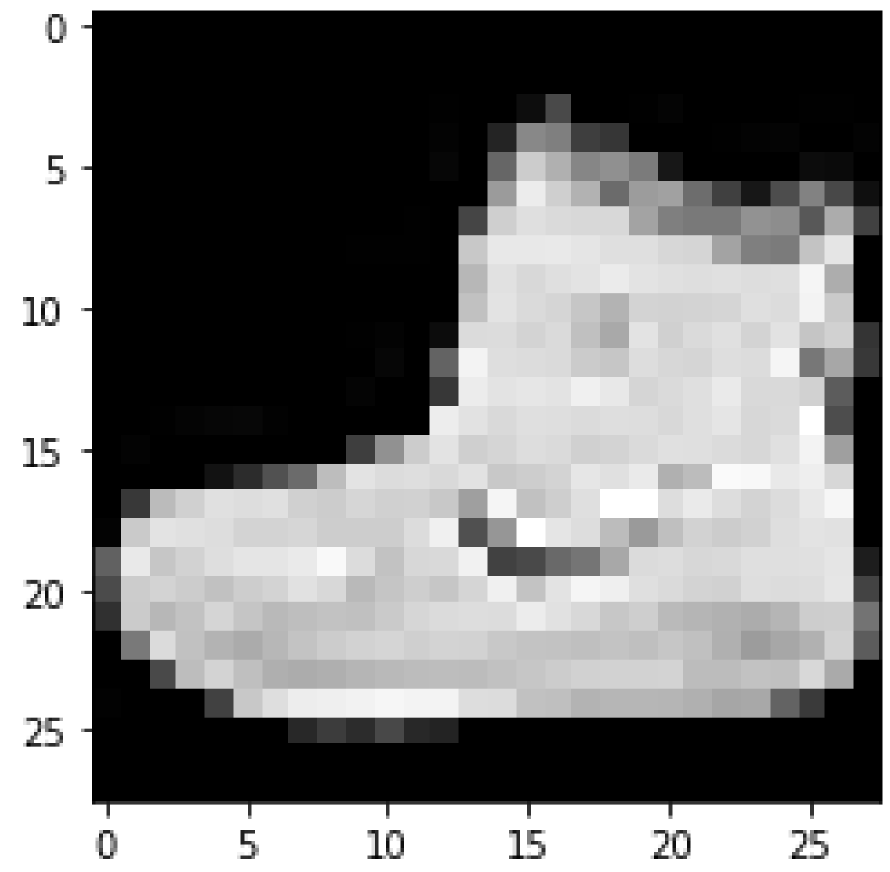
MNIST Fashion

Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

```
# Importing the mnist fasshion dataset
from tensorflow.keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
import matplotlib.pyplot as plt
```

```
# Plot individual image from mnist digits
# pick a sample to plot
sample = 0
image = X_train[sample]
# plot the sample
fig = plt.figure
plt.imshow(image, cmap='gray')
```

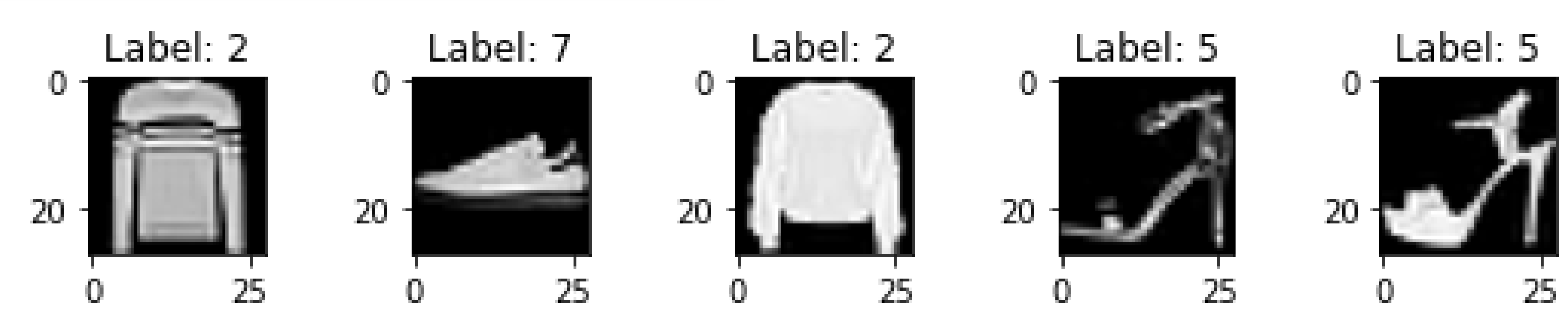
<matplotlib.image.AxesImage at 0x7f954d11c210>



MNIST Fashion

Reading and displaying images from MNIST (digits, letters and fashion)

```
# Labels and images to display
num = 10
images = x_train[:num]
labels = y_train[:num]

# Disply images and labels
num_row = 2
num_col = 5

# Plot images
fig, axes = plt.subplots(num_row, num_col, figsize = (1.5*num_col, 2*num_row))

for i in range(num) :
   ax = axes[1//num_col, i%num_col]
   ax.imshow(images[i], cmap='gray')
   ax.set_title('Label: {}'.format(labels[i]))
plt.tight_layout()
plt.show()
```



MNIST CIFAR10

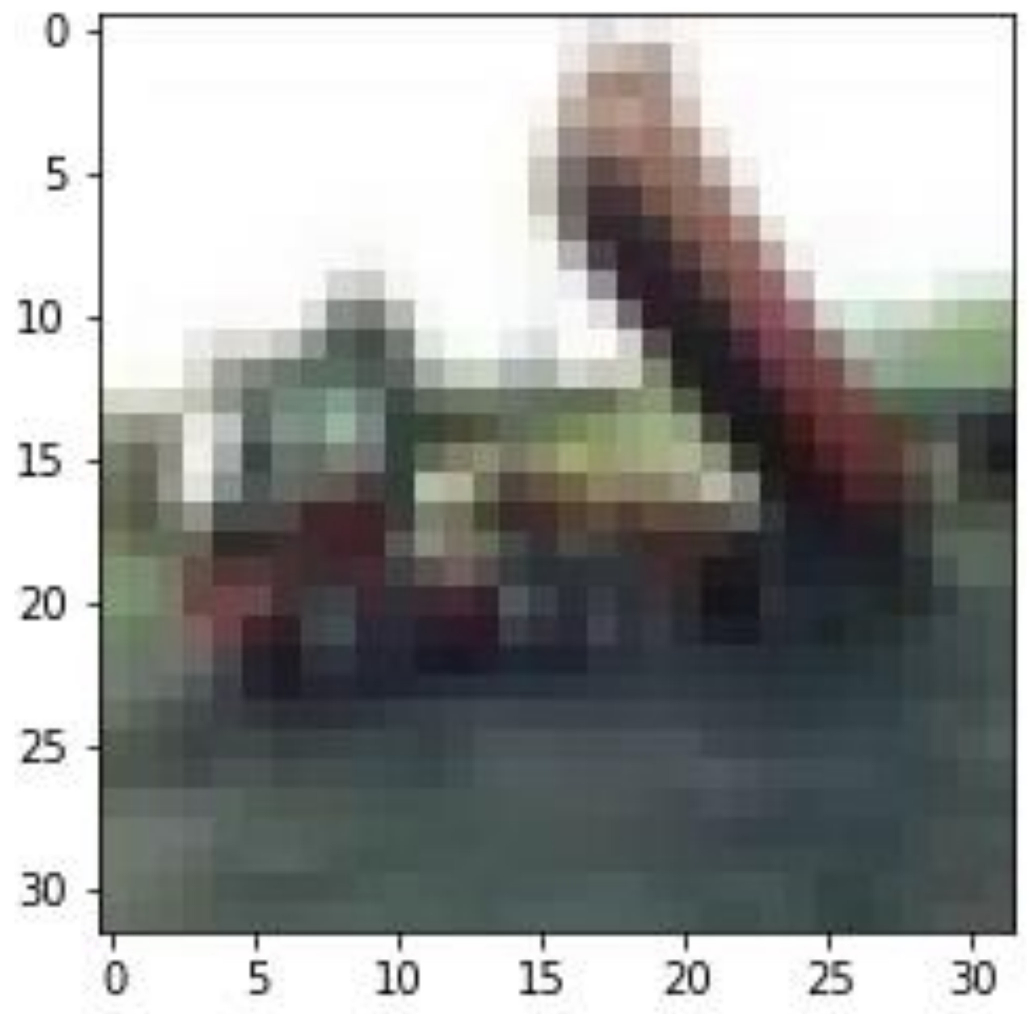
Reading and displaying images from MNIST (digits, letters and fashion)

CIFAR10

CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class.

```
# Importing the cifar10 dataset
from tensorflow.keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
import matplotlib.pyplot as plt
```

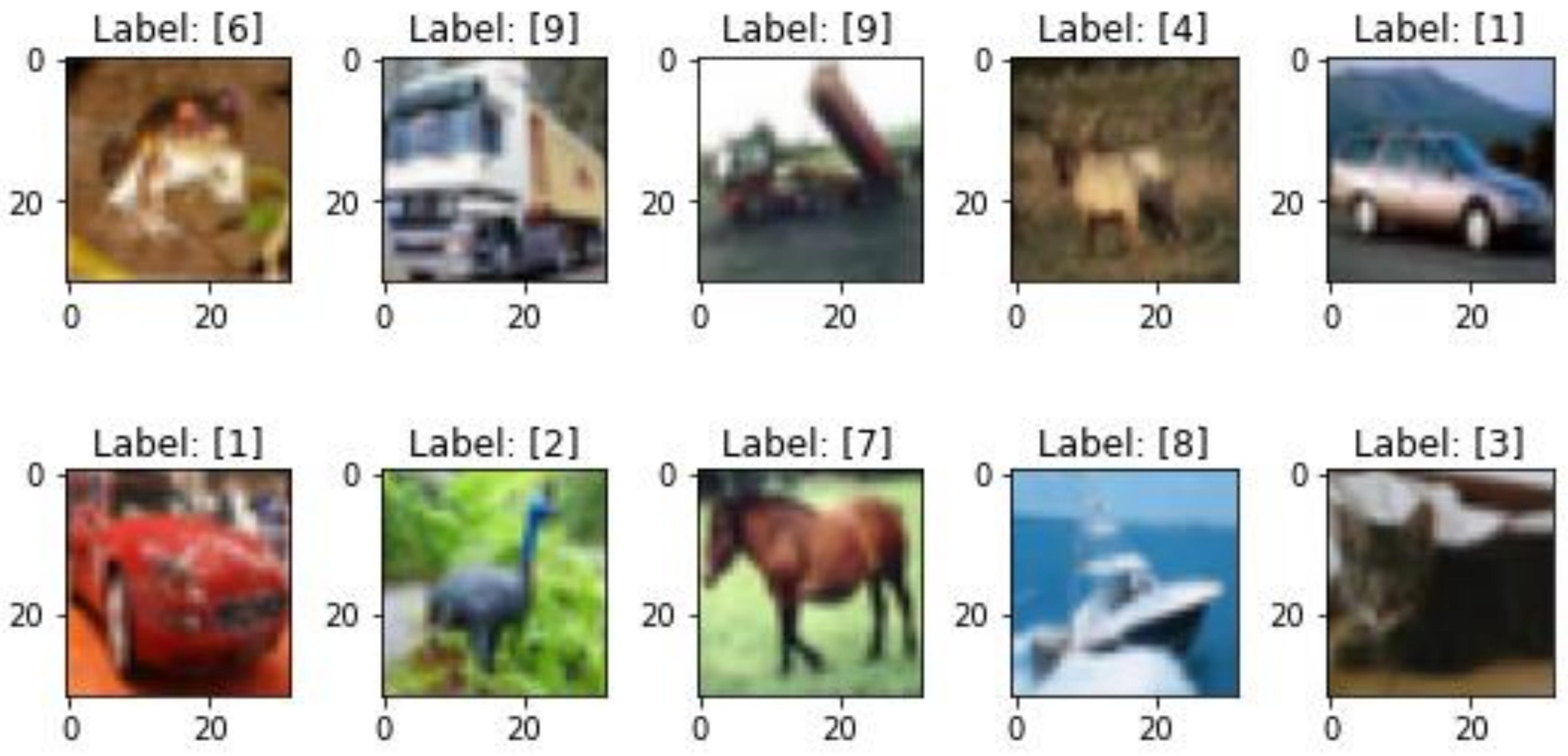
```
# Plot individual image from mnist digits
# pick a sample to plot
sample = 2
image = X_train[sample]
# plot the sample
fig = plt.figure
plt.imshow(image, cmap='gray')
```



MNIST CIFAR10

Reading and displaying images from MNIST (digits, letters and fashion)

```
# labels and images to display
num = 10
images = x_train[:num]
labels = y_train[:num]

# Display digits and labels
num_row = 2
num_col = 5

# plot images
fig, axes = plt.subplots(num_row, num_col, figsize=(1.5*num_col, 2*num_row))

for i in range(num) :
   ax = axes[i//num_col, i%num_col]
   ax.imshow(images[i], cmap='gray')
   ax.set_title('Label: {}'.format(labels[i]))
plt.tight_layout()
plt.show()
```



MNIST CIFAR100

Reading and displaying images from MNIST (digits, letters and fashion)

```
# Importing the cifar100 dataset
from tensorflow.keras.datasets import cifar100
(x_train, y_train), (x_test, y_test) = cifar100.load_data()
import matplotlib.pyplot as plt
```

```
# labels and images to display
num = 10
images = x_train[:num]
labels = y_train[:num]

# Display digits and labels
num_row = 2
num_col = 5

# plot images
fig, axes = plt.subplots(num_row, num_col, figsize=(1.5*num_col, 2*num_row))

for i in range(num) :
    ax = axes[i//num_col, i%num_col]
    ax.imshow(images[i], cmap='gray')
    ax.set_title('Label: {}'.format(labels[i]))
plt.tight_layout()
plt.show()
```