Logistic Regression for a Binary Classification of Images

import library

In [1]:

```
import numpy as np
import matplotlib.image as img
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib import ticker, cm
import os
```

load data

In [2]:

```
directory_data  = './'
filename_data   = 'assignment_02_data.npz'
path_data       = os.path.join(directory_data, filename_data)
data            = np.load(path_data)

x = data['x']
y = data['y']

im_0    = x[0,:,:]
im_1    = x[1,:,:]

label_0 = y[0]
label_1 = y[1]
```

In [3]:

```
print('size of data :', x.shape)
print('number of images :', x.shape[0])
print('size of image :', x[0,:,:].shape)
print('size of label :', y.shape)

print('data type of image :', x.dtype)
print('data type of label :', y.dtype)

print('number of image :', x.shape[0])
print('height of image :', x.shape[1])
print('width of image :', x.shape[2])
```

```
size of data : (2, 28, 28)
number of images : 2
size of image : (28, 28)
size of label : (2,)
data type of image : float64
data type of label : float64
number of image : 2
height of image : 28
width of image : 28
```

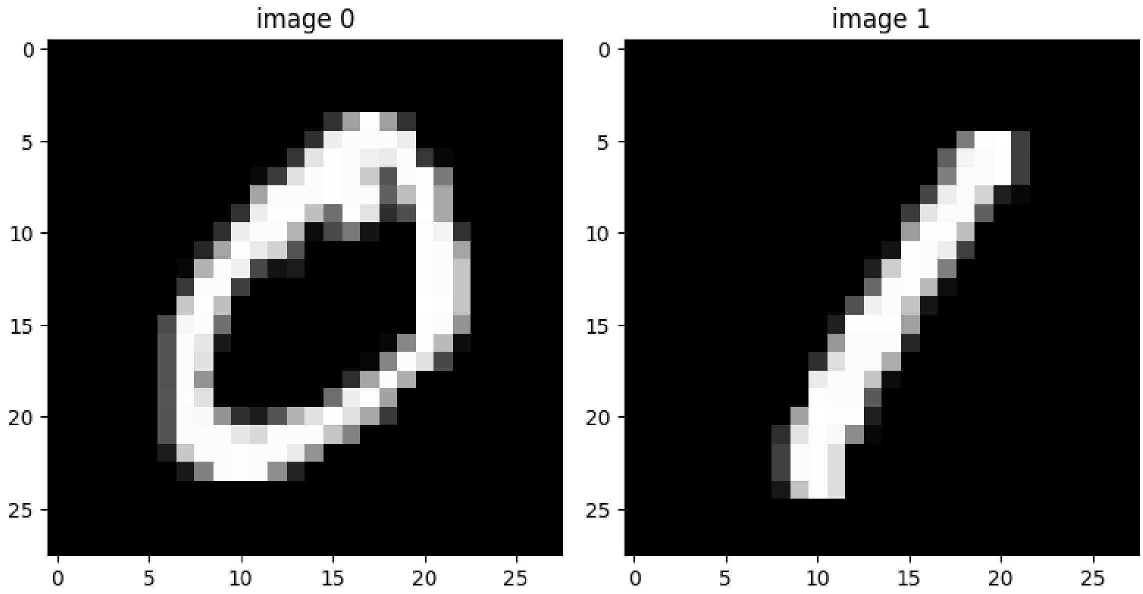
plot grey image

In [4]:

```
def plot_image_gray(title1, data1, title2, data2):

    size_width  = 8
    size_height = 4

    fig = plt.figure(figsize=(size_width, size_height))

    rows = 1
    cols = 2

    ax1 = fig.add_subplot(rows, cols, 1)
    ax2 = fig.add_subplot(rows, cols, 2)

    ax1.set_title(title1)
    ax1.imshow(data1, cmap='gray', vmin=0, vmax=1)

    ax2.set_title(title2)
    ax2.imshow(data2, cmap='gray', vmin=0, vmax=1)

    plt.tight_layout()
    plt.show()
```

In [5]:

```
plot_image_gray('image 0', im_0, 'image 1', im_1)
```



create color images from gray ones

In [6]:

```
im_0_red    = np.zeros((im_0.shape[0], im_0.shape[1], 3))
im_0_green  = np.zeros((im_0.shape[0], im_0.shape[1], 3))
im_0_blue   = np.zeros((im_0.shape[0], im_0.shape[1], 3))

im_1_red    = np.zeros((im_1.shape[0], im_1.shape[1], 3))
im_1_green  = np.zeros((im_1.shape[0], im_1.shape[1], 3))
im_1_blue   = np.zeros((im_1.shape[0], im_1.shape[1], 3))

im_0_red[:,:,0]     = im_0
im_0_green[:,:,1]   = im_0
im_0_blue[:,:,2]    = im_0

im_1_red[:,:,0]     = im_1
im_1_green[:,:,1]   = im_1
im_1_blue[:,:,2]    = im_1
```

In [7]:

```
print('size of red image for 0 : ', im_0_red.shape)
print('size of green image for 0 : ', im_0_green.shape)
print('size of blue image for 0 : ', im_0_blue.shape)
```

```
size of red image for 0 :  (28, 28, 3)
size of green image for 0 :  (28, 28, 3)
size of blue image for 0 :  (28, 28, 3)
```

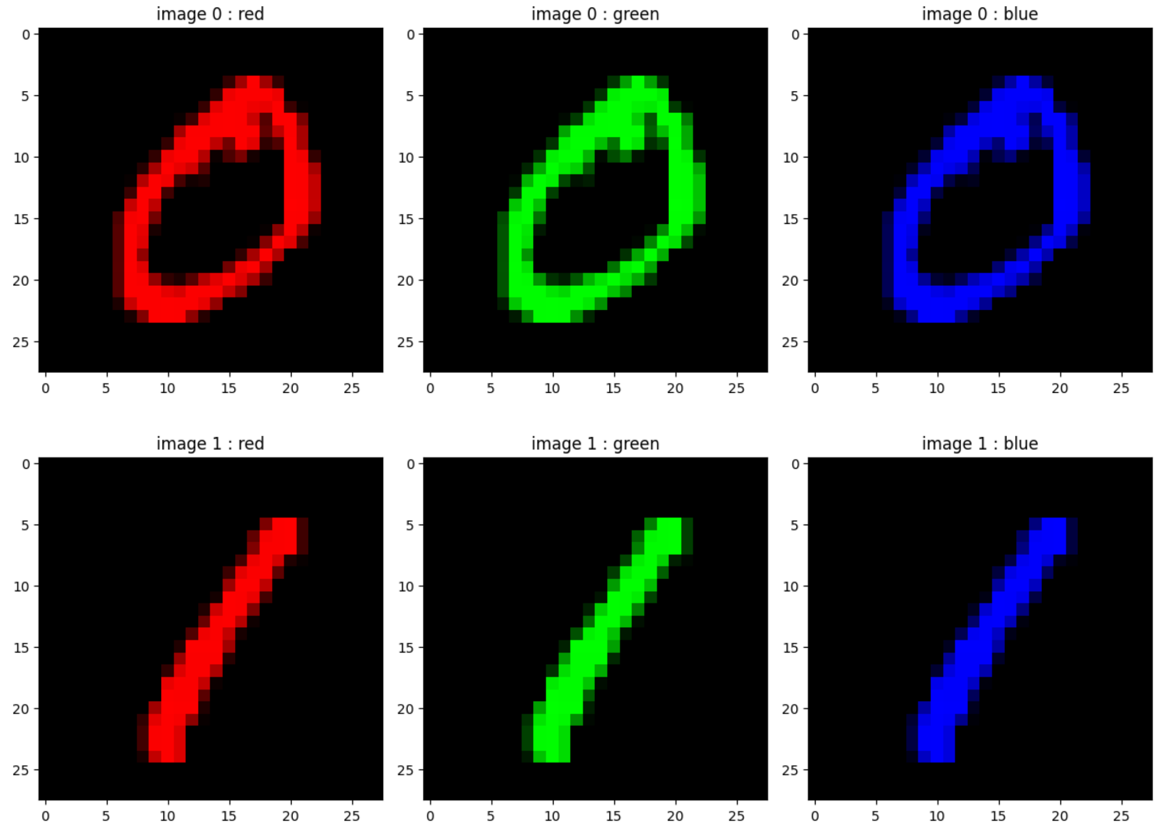
plot color image

In [8]:

```
def plot_image_color(title1, data1, title2, data2, title3, data3):

    size_width  = 12
    size_height = 4

    fig = plt.figure(figsize=(size_width, size_height))

    rows = 1
    cols = 3

    ax1 = fig.add_subplot(rows, cols, 1)
    ax2 = fig.add_subplot(rows, cols, 2)
    ax3 = fig.add_subplot(rows, cols, 3)

    ax1.set_title(title1)
    ax1.imshow(data1, vmin=0, vmax=1)

    ax2.set_title(title2)
    ax2.imshow(data2, vmin=0, vmax=1)

    ax3.set_title(title3)
    ax3.imshow(data3, vmin=0, vmax=1)

    plt.tight_layout()
    plt.show()
```

In [9]:

```
plot_image_color('image 0 : red', im_0_red, 'image 0 : green', im_0_green, 'image 0 : blue', im_0_blue)
plot_image_color('image 1 : red', im_1_red, 'image 1 : green', im_1_green, 'image 1 : blue', im_1_blue)
```



initialize input data

- vectorize input image matrices into vectors (row-major order)

In [82]:

```
# ==================================================
# fill up the blank
#
vec_0    = im_0.flatten()
vec_1    = im_1.flatten()
#
# ==================================================
```

In [83]:

```
print('size of vec 0 : ', vec_0.shape)
print('size of vec 1 : ', vec_1.shape)
```

```
size of vec 0 :  (784,)
size of vec 1 :  (784,)
```

initialize a linear layer for the neural network

- initialize the weights in a fully connected layer (zero matrix)
- dimension
  - input : length of input data
  - output : 1

In [84]:

```
size_input  = len(vec_0)
size_output = 1

# ==================================================
# fill up the blank
#
weight = np.zeros((size_output, size_input))
#
# ==================================================
```

In [85]:

```
print('size of weight : ', weight.shape)
```

```
size of weight :  (1, 784)
```

define a linear layer

- input
  - input data
  - dimension : $m \times 1$
- weight
  - weight associated with the layer of the neural network
  - dimension : $p \times m$
- return
  - output of the layer
  - dimension : $p \times 1$

In [86]:

```
def layer_linear(input, weight):

# ==================================================
# fill up the blank
#
    output = np.matmul(weight, input)
#
# ==================================================

    return output
```

define sigmoid function

In [87]:

```
def activation_sigmoid(input):

# ==================================================
# fill up the blank
#
    output = 1. / (1. + np.exp(-1. * input))
#
# ==================================================

    return output
```

define forward propagation

In [88]:

```
def propagation_forward(input, weight):

# ==================================================
# fill up the blank
#
    output      = layer_linear(input, weight)
    prediction  = activation_sigmoid(output)
#
# ==================================================

    return prediction
```

define the loss function

- cross entropy between the ground truth and the prediction
- cross entropy : $-y\log(h) - (1-y)\log(1-h)$
    - y : grount truth
    - h : prediction
- output dimension : 1×1

In [89]:

```
def compute_loss(prediction, label):

# ==================================================
# fill up the blank
#
    loss = -1. * label * np.log(prediction) - (1. - label) * np.log(1. - predict
ion)
#
# ==================================================

    return loss
```

compute the gradient of the loss with respect to the model parameter

In [90]:

```
def compute_gradient(input, prediction, label):

# ==================================================
# fill up the blank
#
    residual    = prediction - label
    gradient    = residual * input
#
# ==================================================

    return gradient
```

gradient descent algorithm

- hyper-parameters

In [91]:

```
number_iteration    = 500
learning_rate       = 0.01
```

In [92]:

```
loss_iter   = np.zeros(number_iteration)
loss_iter_0 = np.zeros(number_iteration)
loss_iter_1 = np.zeros(number_iteration)

pred_iter_0 = np.zeros(number_iteration)
pred_iter_1 = np.zeros(number_iteration)
```

run the gradient descent algorithm

In [93]:

```
for i in range(number_iteration):

# ==================================================
# fill up the blank
#
    pred_0  = propagation_forward(vec_0, weight)
    loss_0  = compute_loss(pred_0, label_0)
    grad_0  = compute_gradient(vec_0, pred_0, label_0)

    pred_1  = propagation_forward(vec_1, weight)
    loss_1  = compute_loss(pred_1, label_1)
    grad_1  = compute_gradient(vec_1, pred_1, label_1)

    loss    = np.mean([loss_0, loss_1])
    grad    = np.mean([grad_0, grad_1], axis=0)
    weight  = weight - learning_rate * grad
#
# ==================================================

    loss_iter_0[i]  = loss_0
    loss_iter_1[i]  = loss_1
    loss_iter[i]    = loss

    pred_iter_0[i]  = pred_0
    pred_iter_1[i]  = pred_1
```

functions for presenting the results

In [97]:

```
def function_result_01():

    title           = 'loss for image 0'
    label_axis_x    = 'iteration'
    label_axis_y    = 'loss'

    plt.figure(figsize=(8, 6))
    plt.title(title)

# ==================================================
# fill up the blank
#

    plt.plot(loss_iter_0, color='red')
    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

#
# ==================================================

    plt.tight_layout()
    plt.show()
```

In [99]:

```
def function_result_02():

    title           = 'loss for image 1'
    label_axis_x    = 'iteration'
    label_axis_y    = 'loss'

    plt.figure(figsize=(8, 6))
    plt.title(title)

# ==================================================
# fill up the blank
#

    plt.plot(loss_iter_1, color='blue')
    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

#
# ==================================================

    plt.tight_layout()
    plt.show()
```

In [102]:

```
def function_result_03():

    title           = 'total loss'
    label_axis_x    = 'iteration'
    label_axis_y    = 'loss'

    plt.figure(figsize=(8, 6))
    plt.title(title)

# ==================================================
# fill up the blank
#

    plt.plot(loss_iter_0, color='red', label='image 0')
    plt.plot(loss_iter_1, color='blue', label='image 1')
    plt.plot(loss_iter, color='green', label='total')
    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)
    plt.legend()


#
# ==================================================

    plt.tight_layout()
    plt.show()
```

In [105]:

```
def function_result_04():

    title           = 'prediction for image 0'
    label_axis_x    = 'iteration'
    label_axis_y    = 'prediction'

    plt.figure(figsize=(8, 6))
    plt.title(title)

# ==================================================
# fill up the blank
#

    plt.plot(pred_iter_0, color='red')
    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

#
# ==================================================

    plt.tight_layout()
    plt.show()
```

In [107]:

```
def function_result_05():

    title           = 'prediction for image 1'
    label_axis_x    = 'iteration'
    label_axis_y    = 'prediction'

    plt.figure(figsize=(8, 6))
    plt.title(title)

# ==================================================
# fill up the blank
#

    plt.plot(pred_iter_1, color='blue')
    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

#
# ==================================================

    plt.tight_layout()
    plt.show()
```

In [109]:

```
def function_result_06():

    title           = 'prediction'
    label_axis_x    = 'iteration'
    label_axis_y    = 'prediction'

    plt.figure(figsize=(8, 6))
    plt.title(title)

# ==================================================
# fill up the blank
#

    plt.plot(pred_iter_0, color='red', label='image 0')
    plt.plot(pred_iter_1, color='blue', label='image 1')
    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)
    plt.legend()

#
# ==================================================

    plt.tight_layout()
    plt.show()
```

In [110]:

```
def function_result_07():

    print('prediction = %12.10f' % (pred_iter_0[0]))
```

In [111]:

```
def function_result_08():

    print('prediction = %12.10f' % (pred_iter_1[0]))
```

In [112]:

```
def function_result_09():

    print('prediction = %12.10f' % (pred_iter_0[-1]))
```

In [113]:

```
def function_result_10():

    print('prediction = %12.10f' % (pred_iter_1[-1]))
```

In [114]:

```
def function_result_11():

    print('loss = %12.10f' % (loss_iter_0[-1]))
```

In [115]:

```
def function_result_12():

    print('loss = %12.10f' % (loss_iter_1[-1]))
```

In [116]:

```
def function_result_13():

    print('loss = %12.10f' % (loss_iter[-1]))
```

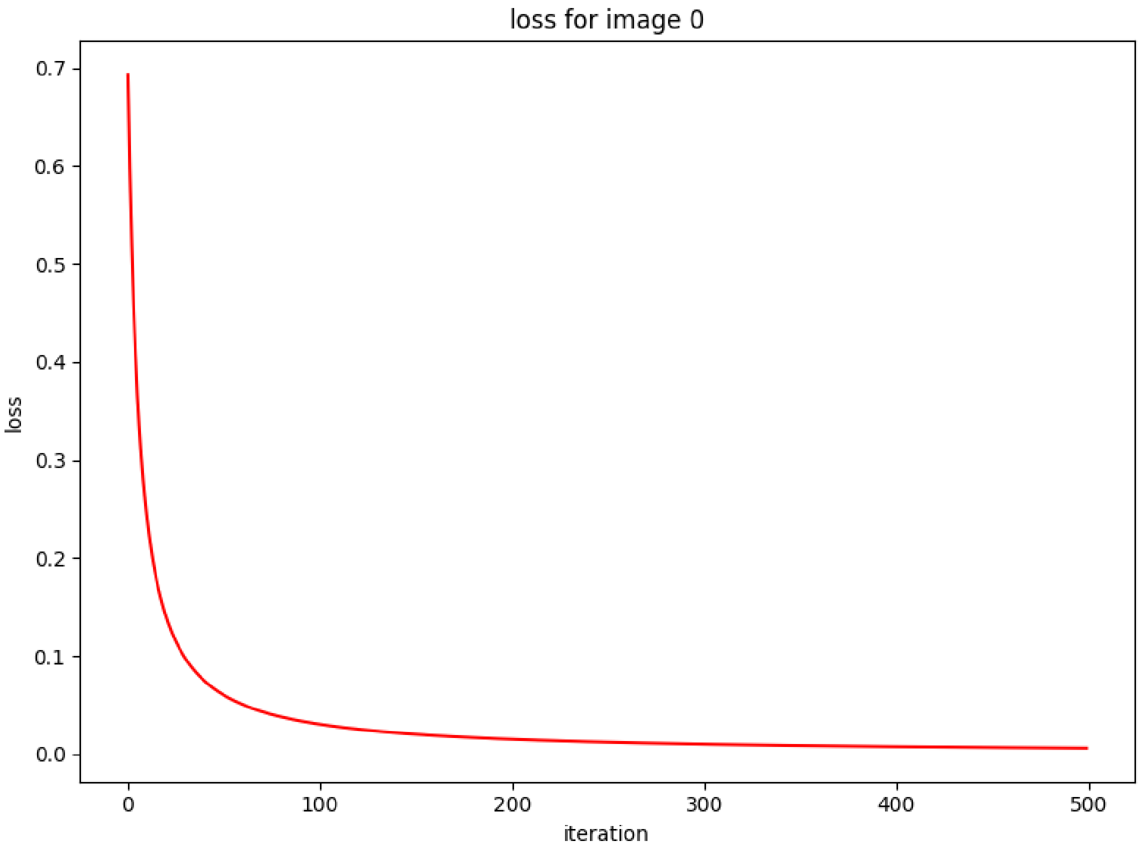
---

results

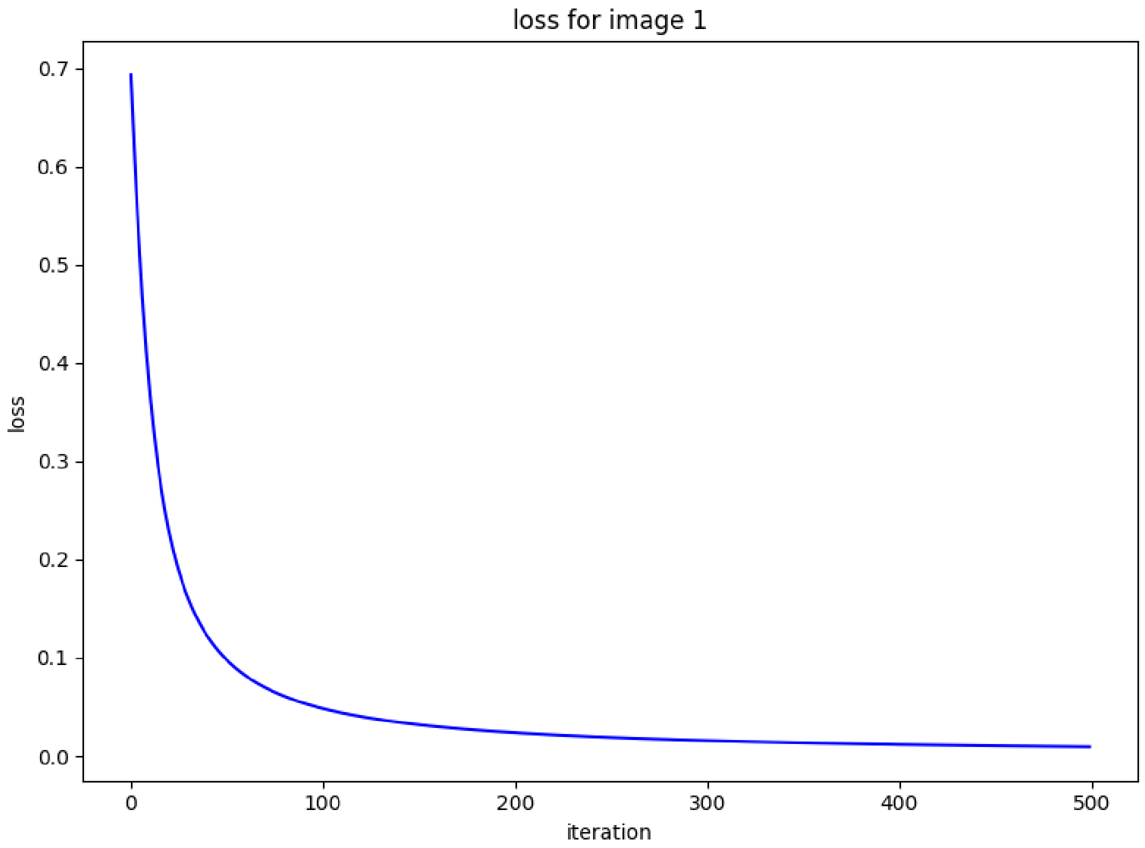
In [117]:

```
number_result = 13

for i in range(number_result):

    title           = '# RESULT # {:02d}'.format(i+1)
    name_function   = 'function_result_{:02d}()'.format(i+1)

    print('')
    print('################################################################################')
    print('#')
    print(title)
    print('#')
    print('################################################################################')
    print('')

    eval(name_function)
```

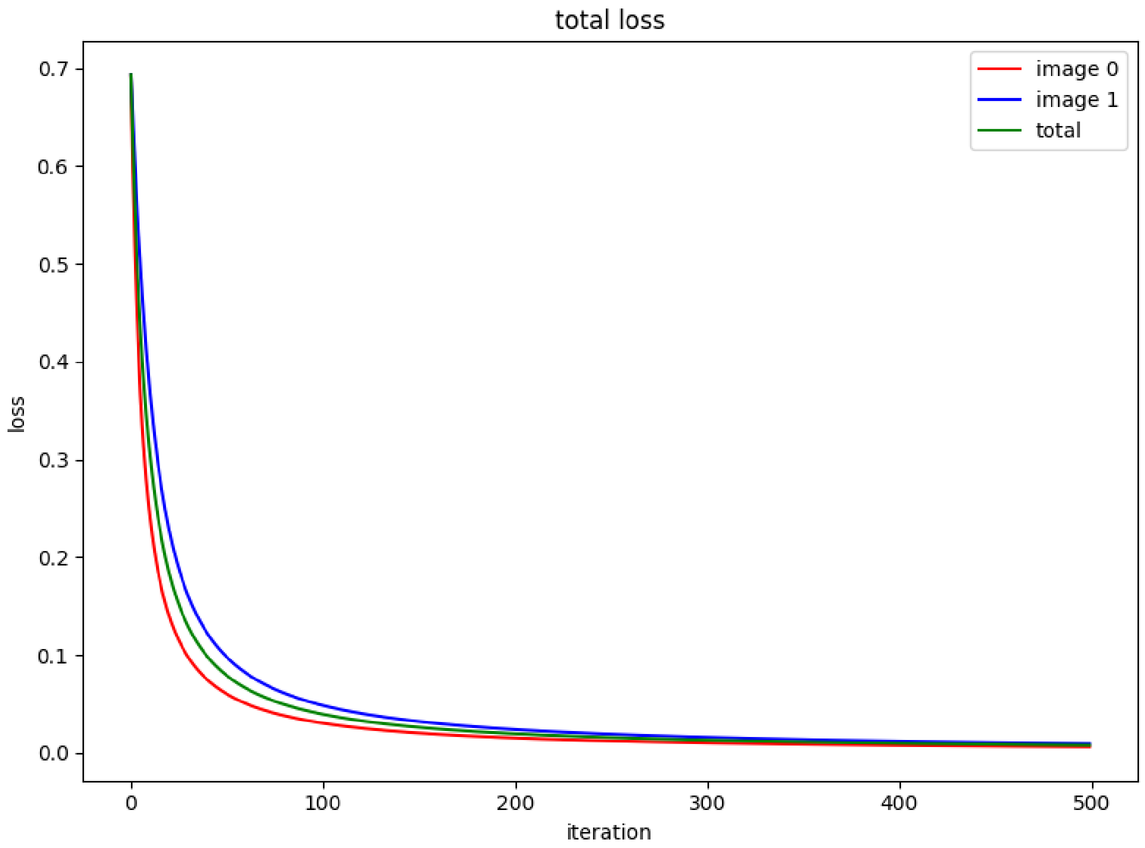
```
####################################################################
############
#
# RESULT # 01
#
####################################################################
############
```



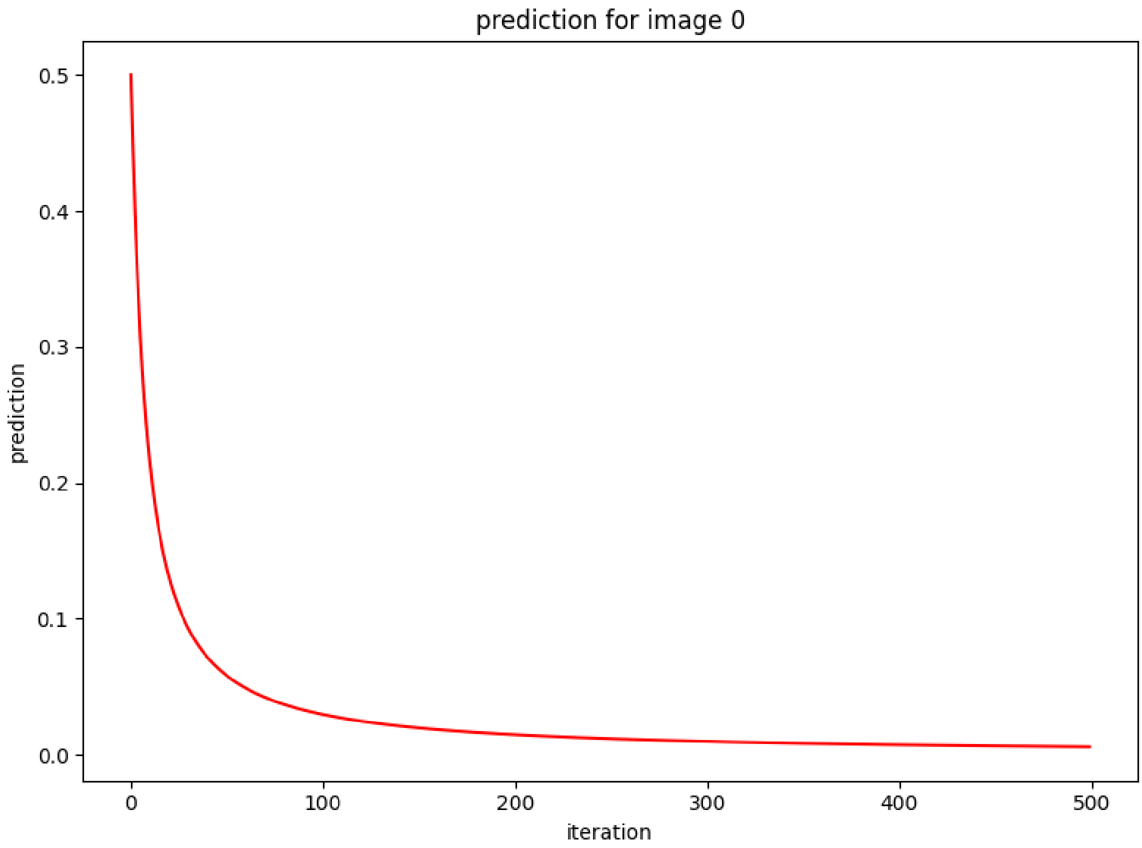

```
####################################################################
############
#
# RESULT # 02
#
####################################################################
############
```


loss for image 1
loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
100
200
300
400
500
iteration

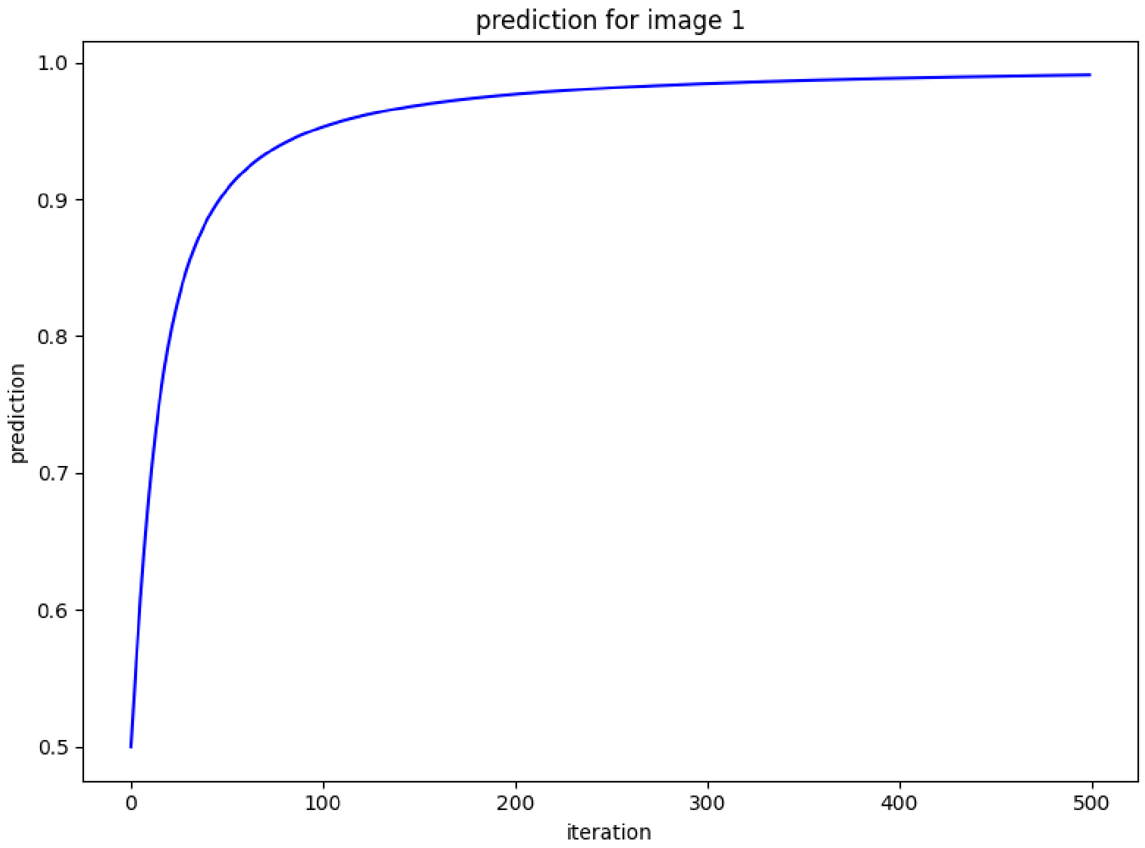
```
####################################################################
############
#
# RESULT # 03
#
####################################################################
############
```


total loss
image 0
image 1
total
loss
iteration
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
100
200
300
400
500

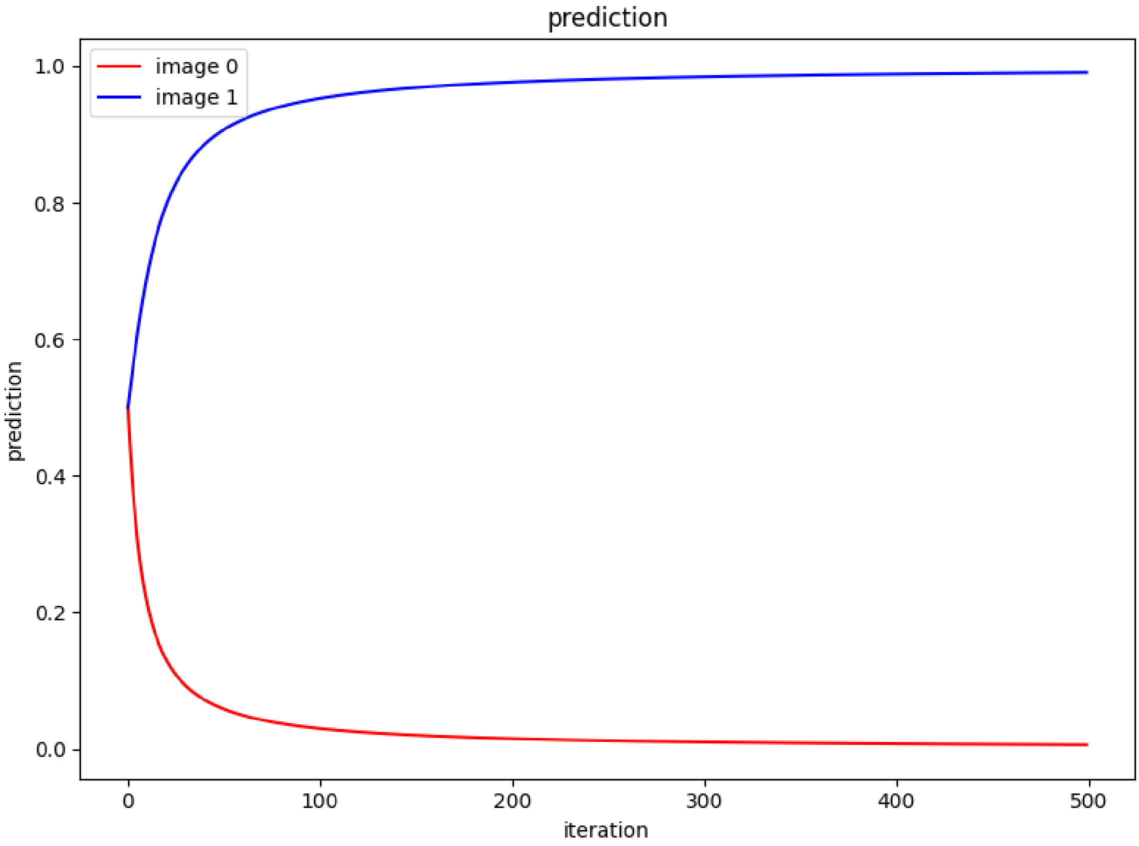
```
######################################################################
############
#
# RESULT # 04
#
######################################################################
############
```


prediction for image 0
prediction
iteration
0.5
0.4
0.3
0.2
0.1
0.0
0
100
200
300
400
500

```
####################################################################
############
#
# RESULT # 05
#
####################################################################
############
```


prediction for image 1
prediction
iteration
1.0
0.9
0.8
0.7
0.6
0.5
0
100
200
300
400
500

```
######################################################################
############
#
# RESULT # 06
#
######################################################################
############
```


prediction
image 0
image 1
1.0
0.8
0.6
0.4
0.2
0.0
prediction
0
100
200
300
400
500
iteration

```
####################################################################
############
#
# RESULT # 07
#
####################################################################
############

prediction = 0.5000000000

####################################################################
############
#
# RESULT # 08
#
####################################################################
############

prediction = 0.5000000000

####################################################################
############
#
# RESULT # 09
#
####################################################################
############

prediction = 0.0058987888

####################################################################
############
#
# RESULT # 10
#
####################################################################
############

prediction = 0.9907446562

####################################################################
############
#
# RESULT # 11
#
####################################################################
############

loss = 0.0059162553

####################################################################
############
#
# RESULT # 12
#
####################################################################
############

loss = 0.0092984407
```

```
######################################################################
############
#
# RESULT # 13
#
######################################################################
############

loss = 0.0076073480
```

In [ ]: