assignment09

November 24, 2018

1 Binary Classification with Different Features

Name: ZHU GUANGYU
Student ID: 20165953
Github Repo: assignment09

In a *classification problem*, the outcome takes on only a finit number of values. In the simplest case, outcome has only two values, for example TRUE or FALSE. This is called the *binary classification problem*.

As in real-valued data fitting, we assume that an approxomate relation ship of the form $y \approx f(x)$ holds, where $f : R^n \rightarrow -1, +1$. The model $\hat{f}$ is called a *classifier*.

For a given data point x, y with predicted outcome $\hat{y} = \hat{f}(x)$, there are four possibilities:

- *True positive*: $y = +1$ and $\hat{y} = +1$.
- *True negative*: $y = -1$ and $\hat{y} = -1$.
- *False positive*: $y = -1$ and $\hat{y} = +1$.
- *False negative*: $y = +1$ and $\hat{y} = -1$.

Continue ith assignmen08, we still use *least squares classifer* to separate 0 and other numbers in MNIST data set.

Sign function is same:

$$sign(x) = \begin{cases} +1 & if x \geq 0 \\ -1 & if x < 0 \end{cases}$$

But this time, we change feature function to see the differencies.

We define new feature functions as $f_i = r_i^T x, r_i \sim N(0, \sigma)$, and try with varing the number of parameter p with the standard deviation $\sigma = 1$ of the random feature vectore r.

1.1 Create Classifier

1.1.1 First, import data sets

We have two data sets, one for training, one for testing. Each element is a image that has height 28 and width 28 pixels.

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np

        file_data_train = "mnist_train.csv"
        file_data_test  = "mnist_test.csv"

        h_data_train    = open(file_data_train, "r")
        h_data_test     = open(file_data_test, "r")

        data_train      = h_data_train.readlines()
        data_test       = h_data_test.readlines()

        h_data_train.close()
        h_data_test.close()

        size_row    = 28    # height of the image
        size_col    = 28    # width of the image

        num_train   = len(data_train)   # number of training images
        num_test    = len(data_test)    # number of testing images

        # number of training images: 60000
        # number of testing images: 10000
```

To reduce the bias, we need to normalize the data.

```
In [2]: #
        # normalize the values of the input data to be [0, 1]
        #
        def normalize(data):

            data_normalized = (data - min(data)) / (max(data) - min(data))

            return(data_normalized)
```

Normalize each pixel, and put image data into a 784*num_image matrix.

```
In [3]: #
        # make a matrix each column of which represents an images in a vector form
        #
        list_image_train    = np.empty((size_row * size_col, num_train), dtype=float)  # 784 *
        list_label_train    = np.empty(num_train, dtype=int)
```

```
list_image_test     = np.empty((size_row * size_col, num_test), dtype=float)
list_label_test     = np.empty(num_test, dtype=int)

count = 0

for line in data_train:

    line_data   = line.split(',')
    label       = line_data[0]
    im_vector   = np.asfarray(line_data[1:])  # convert to float type
    im_vector   = normalize(im_vector)

    list_label_train[count]      = label
    list_image_train[:, count]   = im_vector  # each column is a image

    count += 1

count = 0

for line in data_test:

    line_data   = line.split(',')
    label       = line_data[0]
    im_vector   = np.asfarray(line_data[1:])
    im_vector   = normalize(im_vector)

    list_label_test[count]       = label
    list_image_test[:, count]    = im_vector

    count += 1
```

1.1.2 Define feature functions generator

This is the core part of this assignment.

Depends on the given number of parameters we should generate corresponde feature functions r. Each element of r is a vector with length $28 * 28$. The element of r_i is random number from a normal distribution.

```
In [4]: def generate_features(n, size):
            """Generate feature functions

            Argumengs:
                n(int): number of parameters
                size: number of elements of each vector
            Return:
                functions(2d matrix): feature function matrix
            """
```

```
    functions = []

    # for n, generate vectore with #size elements
    mean, sigma = 0, 1

    for _ in range(n):
        ri = np.random.normal(mean, sigma, size)
        functions.append(ri)

    return np.array(functions)
```

1.1.3 Generate the matrix of feature funtions and data

Since we have got feature functions, now we can apply them on input data.

```
In [5]: def generate_tilde_matrix(feature_func, data, num_data):
            """Create matrix of feature function on data

            Arguments:
                feature_func(2d matrix): feature function matrix
                data: input image data
                num_data: number of input data
            Return:
                matrix of feature functions applied on image data
            """

            A = []

            for i in range(num_data):
                img = data[:, i]  # ith image(column)
                row = np.inner(feature_func, img)
                A.append(row)

            return np.array(A)
```

1.1.4 Compute θ

Depends on $A\theta = y$, while A is the matrix of feature function apply on data set, θ is perameters, and y is the label.

Because we just want to separate 0 and other numbers, we need to process label y which gives 0's image $+1$ and other number's image -1.

Through pseudo inverse $(A^T A)^{-1}A$ we can compute the θ. Here we use `np.linalg.pinv` to get A^{-1}.

```
In [6]: # process label array
        def process_label(labels):
            result = []
            for label in labels:
                if label == 0:
```

```
                result.append(1)
            else:
                result.append(-1)

        return result


    def compute_theta(A, y):
        A_inv = np.linalg.pinv(A)
        theta = np.inner(A_inv, y)

        return theta
```

1.1.5 Define classifier $\hat{f}(x)$

Now we have had feature functions, parameters, so we can create our classifier $\hat{f}(x) = sign(\tilde{f}(x))$, where

$$sign(x) = \begin{cases} +1 & if x \geq 0 \\ -1 & if x < 0 \end{cases}$$

$$\tilde{f}(x) = \theta_1 f_1(x) + \theta_2 f_2(x) + \cdots + \theta_p f_p(x)$$

sign function

```
In [7]: def sign_func(x):
            if x >= 0:
                return 1
            else:
                return -1
```

Combine all components

```
In [8]: def classifier(input_data, data_num, feature_funcs, theta):
            """Given classify result

            Argument:
                input_data(2d matrix): testing image data
            Return:
                determine result array
            """

            # generate tilde f matrix
            tilde_matrix = generate_tilde_matrix(feature_funcs, input_data, data_num)

            # get classify result
            f_tilde = np.inner(tilde_matrix, theta)
            f_hat = list(map(sign_func, f_tilde))
```

```
            return np.array(f_hat)


    def eles_classifier(num_paras, size, data_train, num_data, labels):
        """Generate feature functions, compute parameters

        Arguments:
            num_paras(int): number os parameters
            size(int): size of input data
            data_train(2d matrix): training data set
            num_data(int): number of data
            labels(array): label of each training data
        Returen:
            (tuple): feature function, theta

        """

        # generate feature functions
        feature_funcs = generate_features(num_paras, size)

        # generate tilde f matrix
        A = generate_tilde_matrix(feature_funcs, data_train, num_data)

        # process label
        label_list = process_label(labels)

        # compute parameter
        theta = compute_theta(A, label_list)

        return feature_funcs, theta
```

1.1.6 Count predicted outcome: TP, FP, TN, and FN

We have already got the prediction by our classifier $\hat{f}(x)$, now let's compare it with the label of testing data set to check how it works.

The outcome are

- *True positive*: $y = +1$ and $\hat{y} = +1$.
- *True negative*: $y = -1$ and $\hat{y} = -1$.
- *False positive*: $y = -1$ and $\hat{y} = +1$.
- *False negative*: $y = +1$ and $\hat{y} = -1$.

```
In [9]: def outcomes(labels, prediction):
            """count outcomes of prediction

            Input:
                label(array): correct labels
```

```
        prediction(array): prediction of classifier
    Return:
        A dictionary contains the indices of each outcome type
        tp: true positive
        tn: true negative
        fp: false positive
        fn: flase negative
    """

    # process labels, let 0 == +1, others == -1
    label_processed = process_label(labels)

    length = len(label_processed)
    tp = []
    fp = []
    tn = []
    fn = []

    for i in range(length):
        if label_processed[i] == 1 and prediction[i] == 1:
            tp.append(i)
        elif label_processed[i] == 1 and prediction[i] == -1:
            fn.append(i)
        elif label_processed[i] == -1 and prediction[i] == -1:
            tn.append(i)
        else:
            fp.append(i)

    outcome = {'TP': tp,
               'FP': fp,
               'TN': tn,
               'FN': fn}

    return outcome
```

1.1.7 Define F_1 score funtion

$$F_1 score = 2 \cdot \frac{precision \cdot recall}{precision + recall},$$

where $precision = \frac{true\ positives}{true\ positives\ +\ false\ positives}$, $recall = \frac{true\ positives}{false\ negative\ +\ true\ positive}$

```
In [10]: def f1_score(outcome):

             tp = len(outcome['TP'])
             fp = len(outcome['FP'])
             tn = len(outcome['TN'])
             fn = len(outcome['FN'])
```

```
        precision = tp / (tp + fp)
        recall = tp / (fn + tp)

        return 2 * precision * recall / (precision + recall)
```

1.1.8 Plotting funcions

```
In [11]: def average_img(data, indices):
             # compute the average value of one outcome type

             size = 28 * 28
             sum_img = np.zeros(size)

             for index in indices:
                 img = data[:, index]
                 sum_img += img

             num_img = len(indices)

             return sum_img / num_img


         def plot_all(data, outcomes):

             labels = ['TP', 'FP', 'TN', 'FN']

             for i in range(4):

                 label = labels[i]
                 imgs = average_img(data, outcomes[label])
                 img_matrix = imgs.reshape((28, 28))

                 plt.subplot(2, 2, i+1)
                 plt.title('Average Image of ' + label)
                 plt.imshow(img_matrix, cmap='Greys', interpolation='None')

                 frame   = plt.gca()
                 frame.axes.get_xaxis().set_visible(False)
                 frame.axes.get_yaxis().set_visible(False)

             plt.show()
```

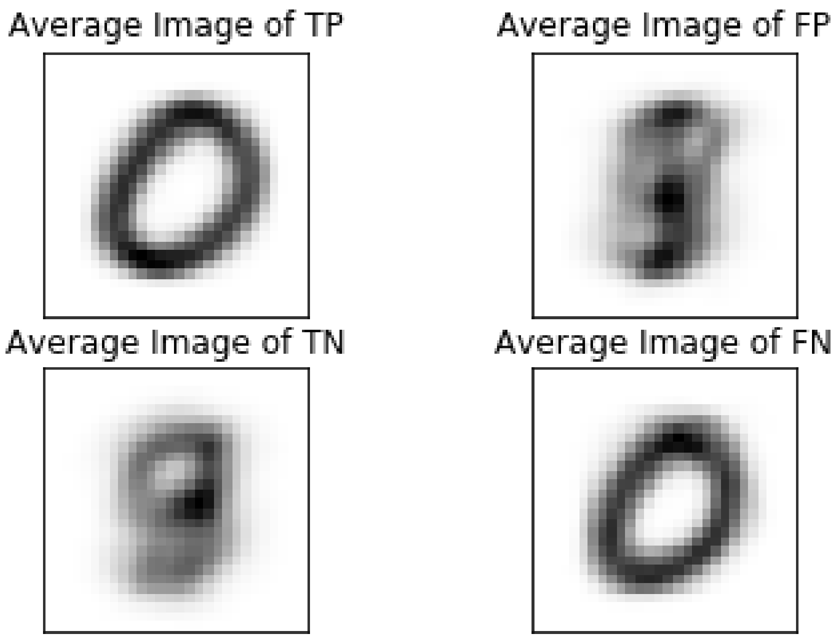
1.2 Try different number of parameters

Now let's try different number parameters.

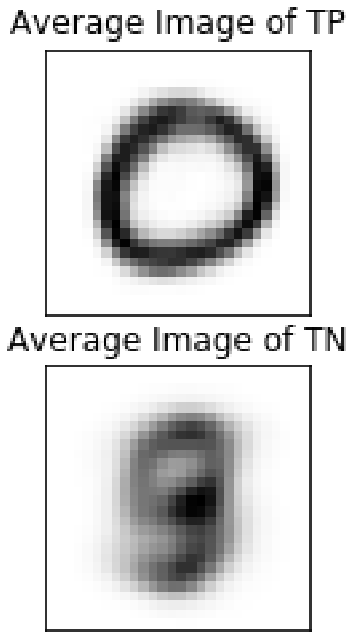
Here we use logarithm $2^n, n \in [1, 10]$ number of parameters.

```
In [17]: num_paras = [2**n for n in range(1, 11)]

         f1_history = []

         for p in num_paras:

             features, theta = eles_classifier(p, 28*28, list_image_train, num_train, list_labe

             prediction = classifier(list_image_test, num_test, features, theta)
             outcome = outcomes(list_label_test, prediction)

             f1_history.append(f1_score(outcome))

             print("\n {} parameters' images".format(p))
             plot_all(list_image_test, outcome)
```

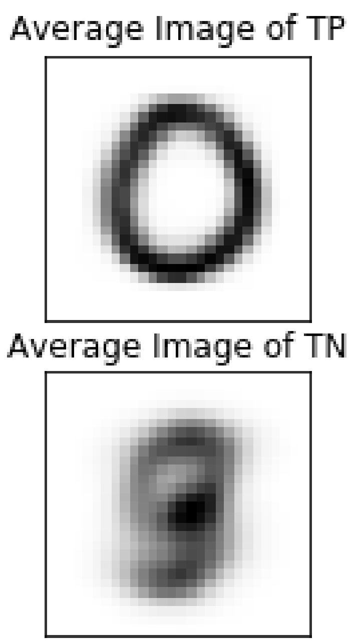
```
 2 parameters' images
```



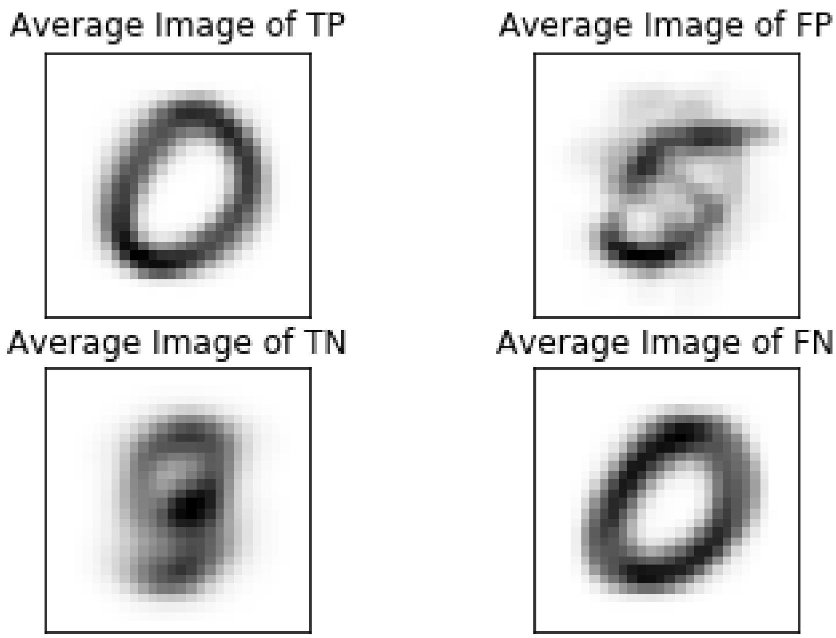

```
 4 parameters' images
```



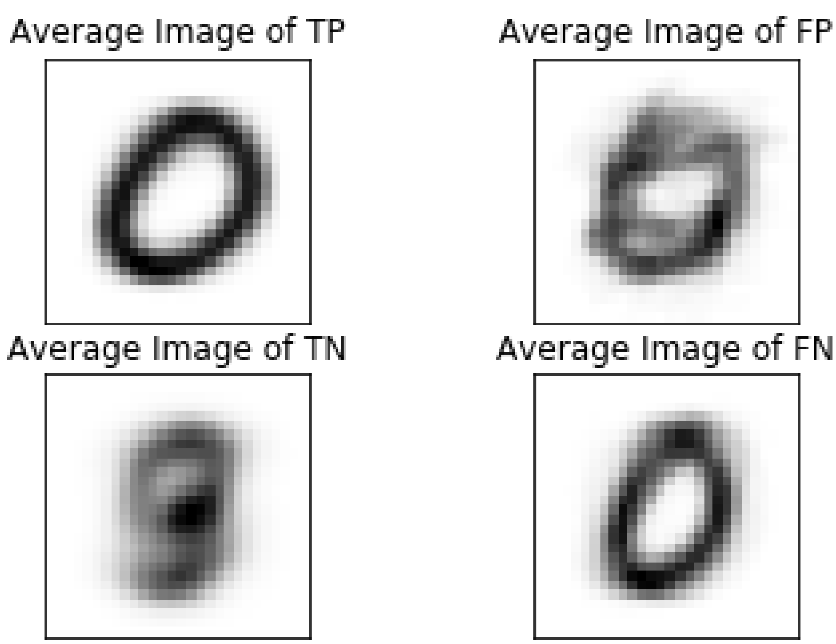

```
8 parameters' images
```



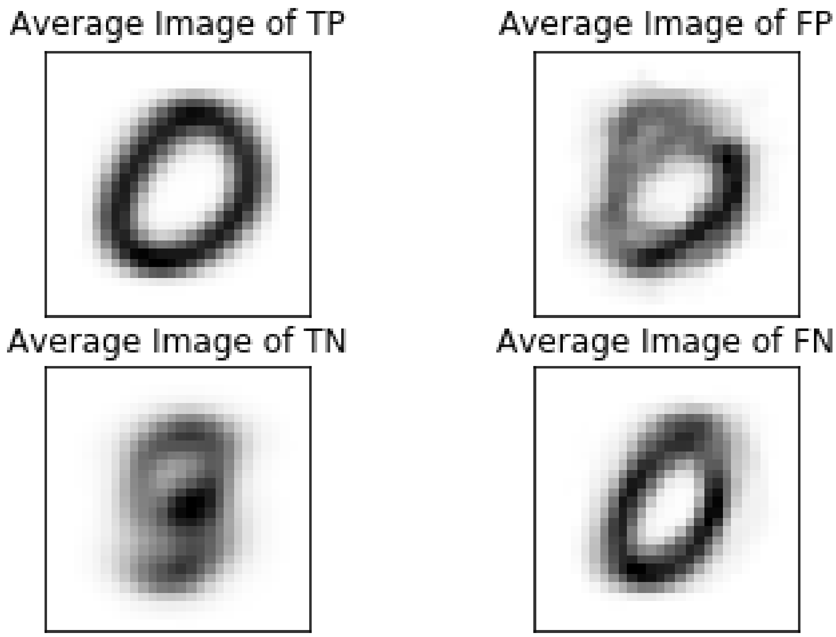
16 parameters' images



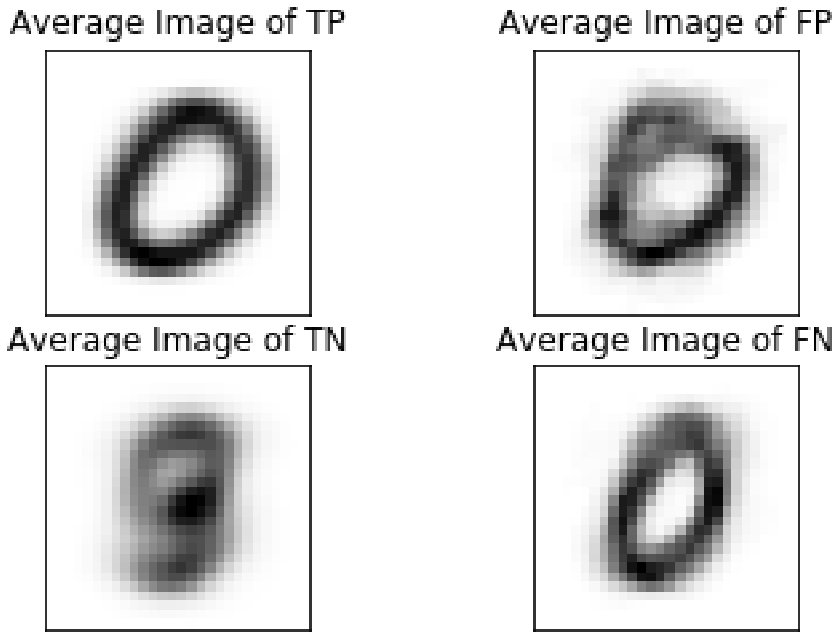
32 parameters' images



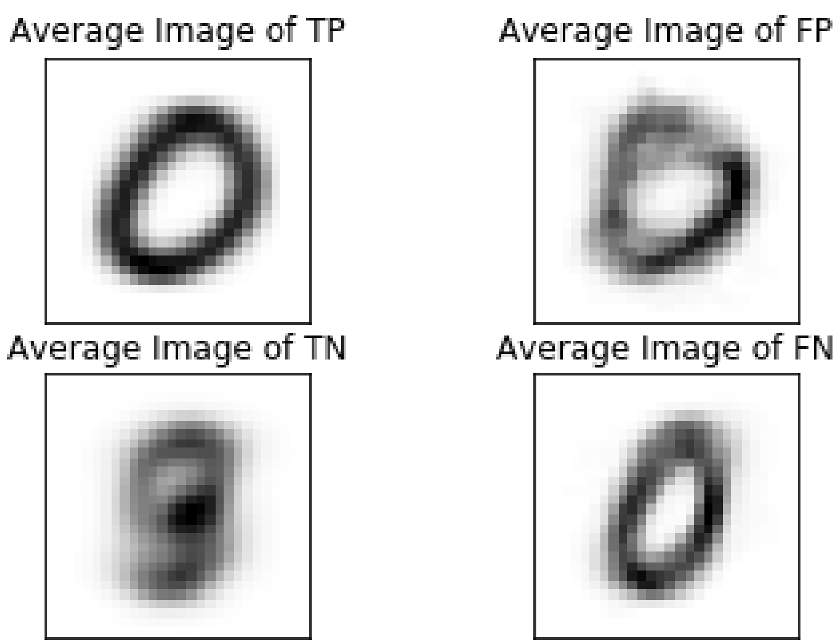
64 parameters' images



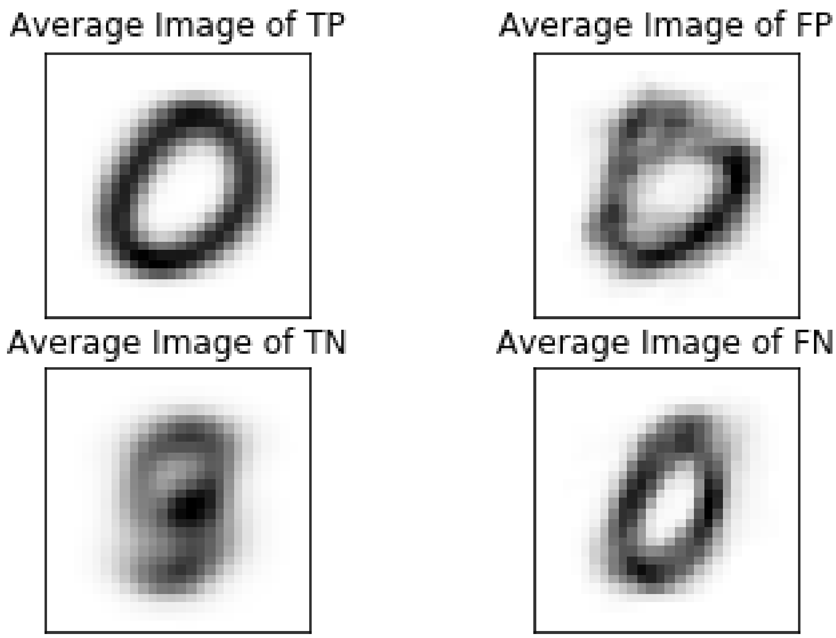
128 parameters' images



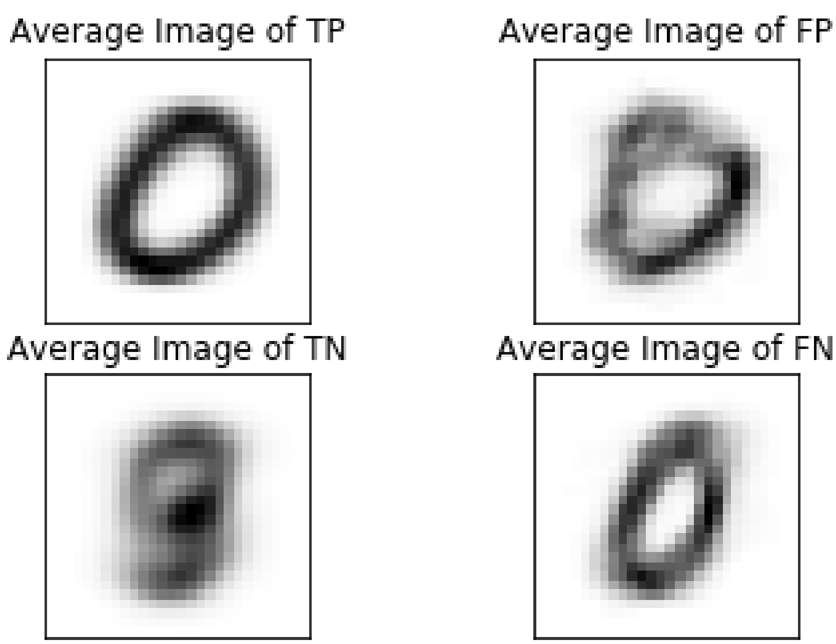
256 parameters' images



512 parameters' images



1024 parameters' images



```
In [20]: plt.title("F1 Score History")
         plt.plot(num_paras, f1_history, 'b-')
         plt.xlabel('number of parameters')
         plt.ylabel('F1 score')
         plt.show()
```