ÖZYEĞİN UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

CS 454

2021 Fall

Homework3

Perceptrons

By
Tuğcan Hoşer
S015561

Supervised By
Ethem Alpaydın

PART A

1) Intro

First of all, the data in the training.csv file and testing.csv in the project was separated according to their classes. There are 10 classes and each class has 100 data.

2) Weights

A weights of 100 columns of 10 rows was randomly created with numbers between -0.01 and +0.01.

3) Single Layer Perception Formulas

$$y_i \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = w_i^T x$$

W= weights
X = inputs
The weight and input values are multiplied by the dot product.

4) Classification

$$O_i = w_i^T x$$

Dot products are made to each class. Oi is initialized.

5) Softmax

$$y_i = \frac{exp\ o_i}{\Sigma_k\ exp\ O_k}$$

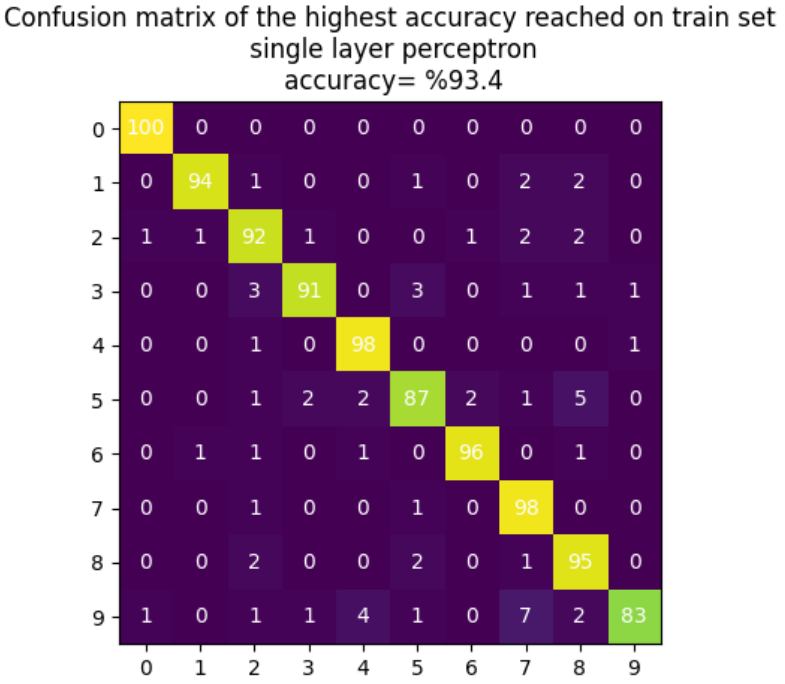
Oi values have individual softmax values. The highest value is our prediction.

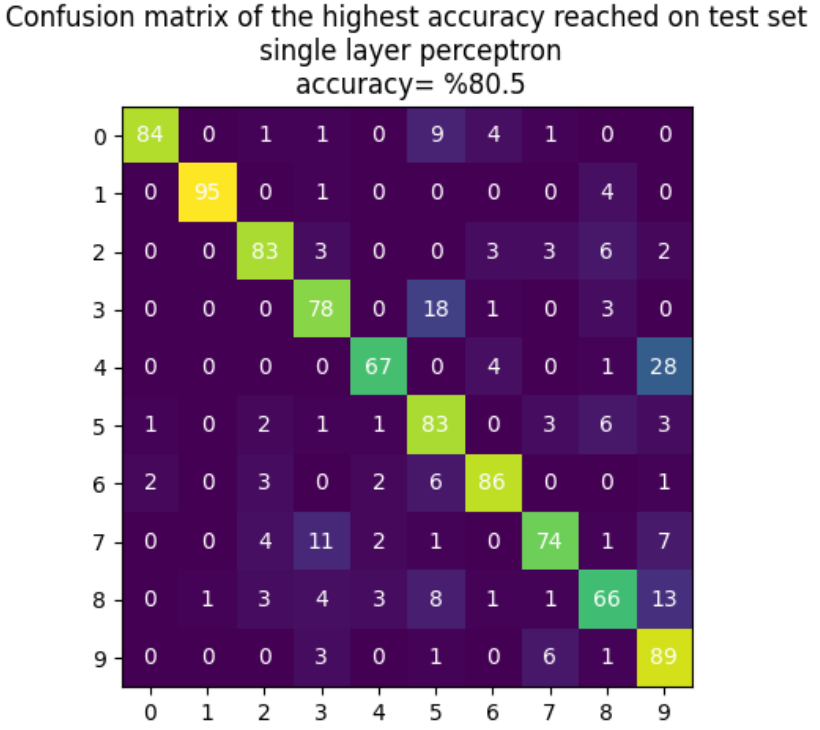
6) Training

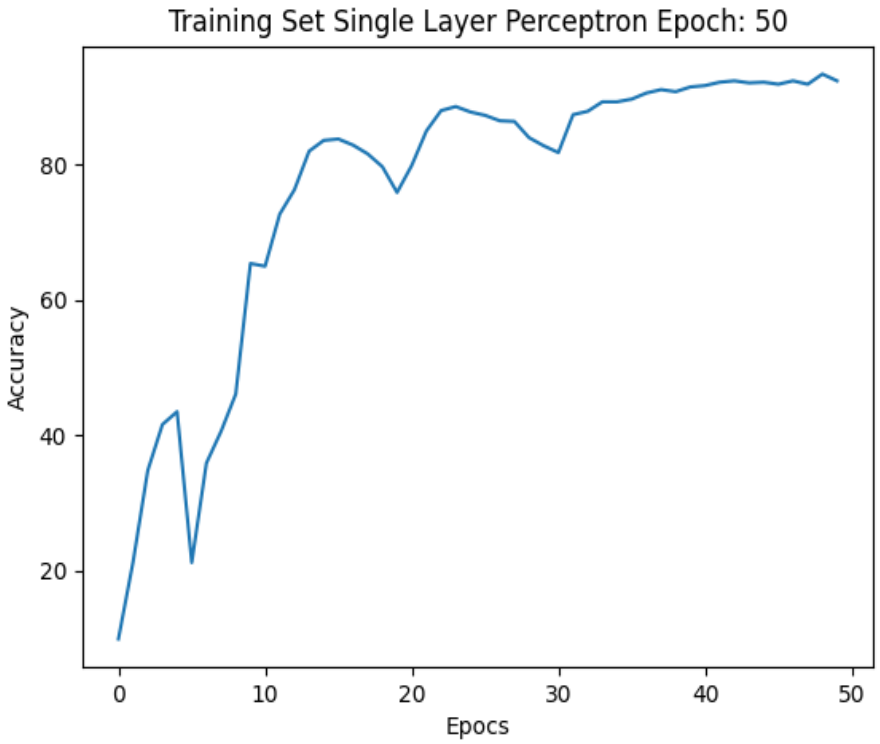
$$\Delta w_{ij}^t = \eta\left(r_i^t - y_i^t\right)x_j^t$$

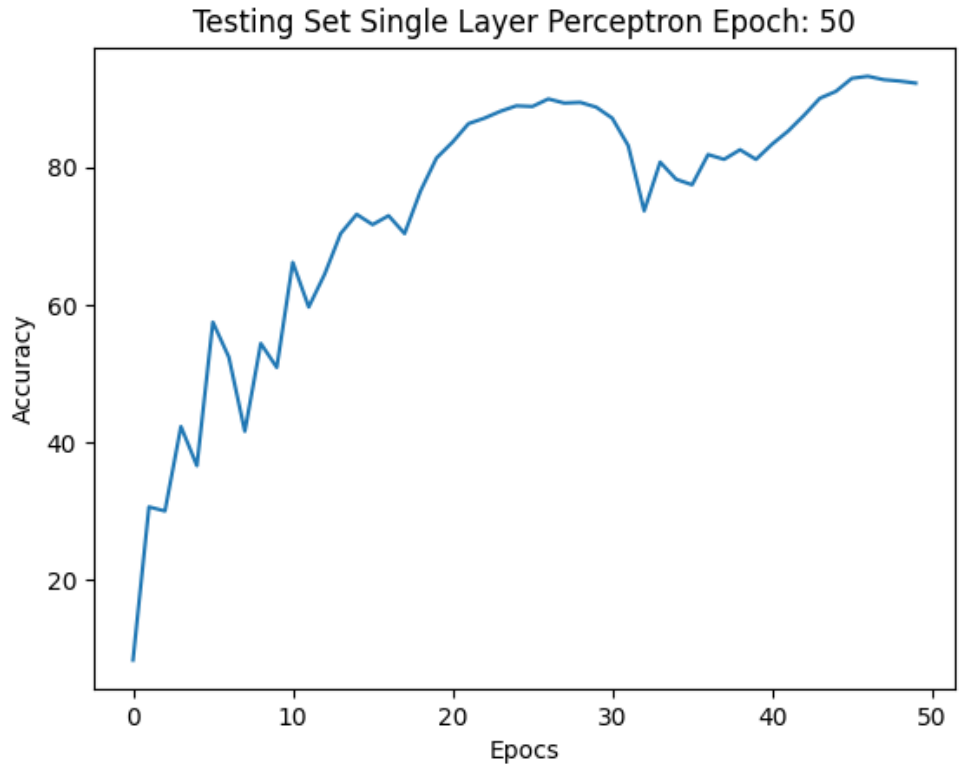
Update = LearningFactor (Desired Output-Actual Output) * Input

8)Results






Training Set Single Layer Perceptron Epoch: 50
Accuracy
Epocs
20
40
60
80
0
10
20
30
40
50


Testing Set Single Layer Perceptron Epoch: 50
Accuracy
Epocs
20
40
60
80
0
10
20
30
40
50

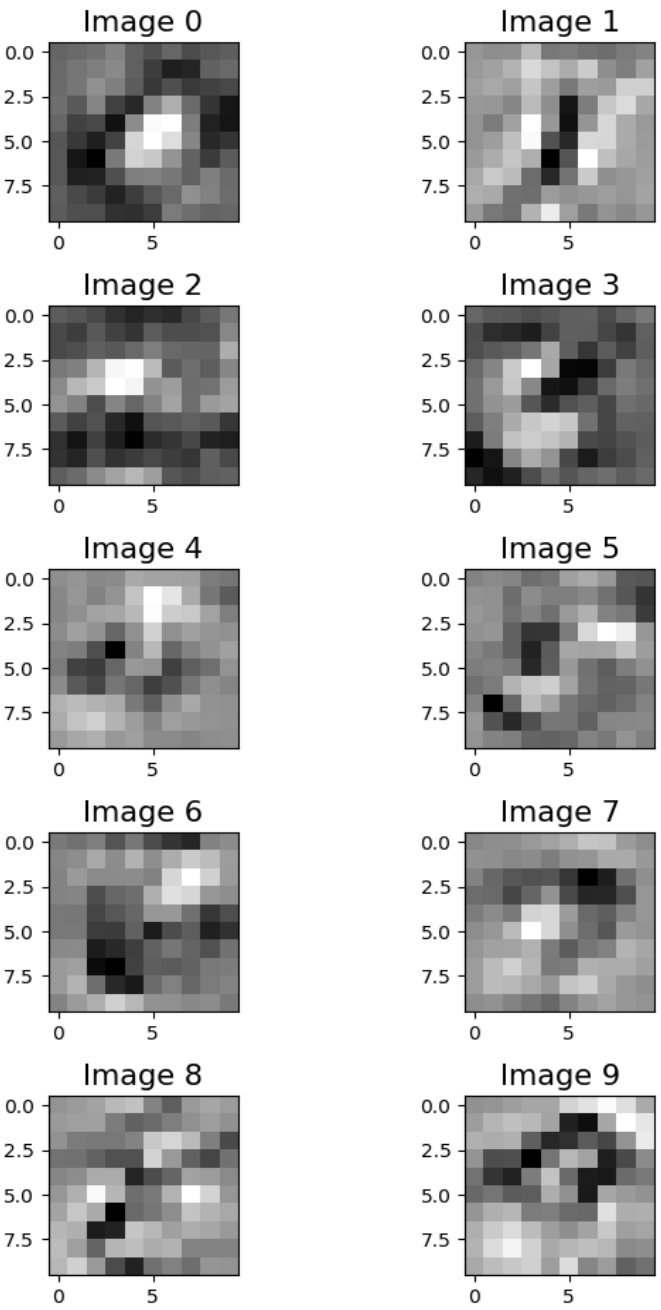


Image 0
Image 1
Image 2
Image 3
Image 4
Image 5
Image 6
Image 7
Image 8
Image 9
0.0
2.5
5.0
7.5
0
5

PART B

1) Multi Layer Perceptron

Initialize the weights randomly

w = [H = (0, 10, 25, 50, 75), Dimension = 100]

v = [K = 10, H = (25, 50, 75)]

$$o_i = \sum_{h=1}^{H} v_{ih}\omega_n^t + v_{i0}$$

v = output layer weights

w = input layer weights

h = hidden layer neurons

$$y_i^t = \frac{e^{o_i^t}}{\Sigma_k e_k^t}$$

k = classes

t = sample number

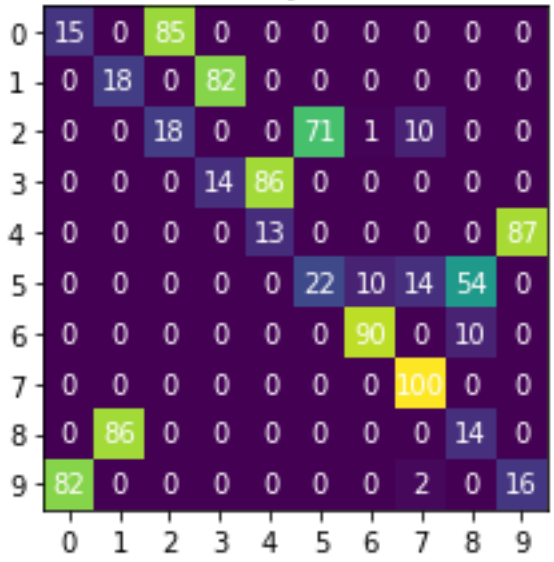
$$E = (\omega, v|x) = -\sum_t \sum_i r_i^t \, log \, y_i^t$$

$$\Delta v_{ih} = \alpha \sum_t \left(r_i^t - y_i^t\right) z_h^t$$

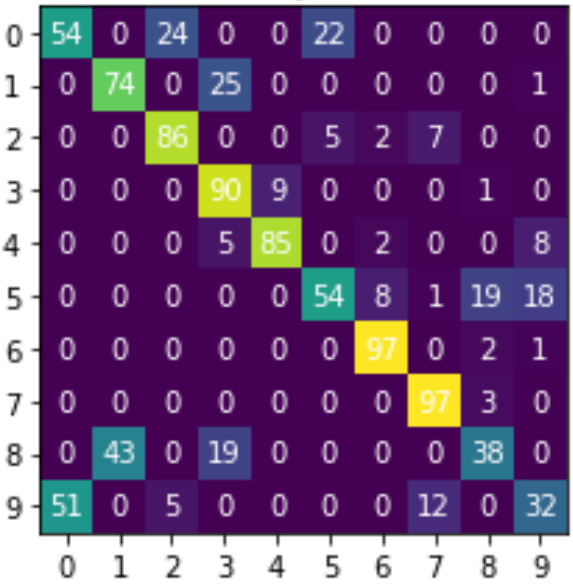
$$\Delta w_{h_j} = \alpha \sum_t \left[\sum_i \left(\Gamma \frac{1}{i} - y_i^t\right) v_i h\right] z \frac{1}{h}(1 - z_h^t) x_j^t$$

2) Results epocs = 15

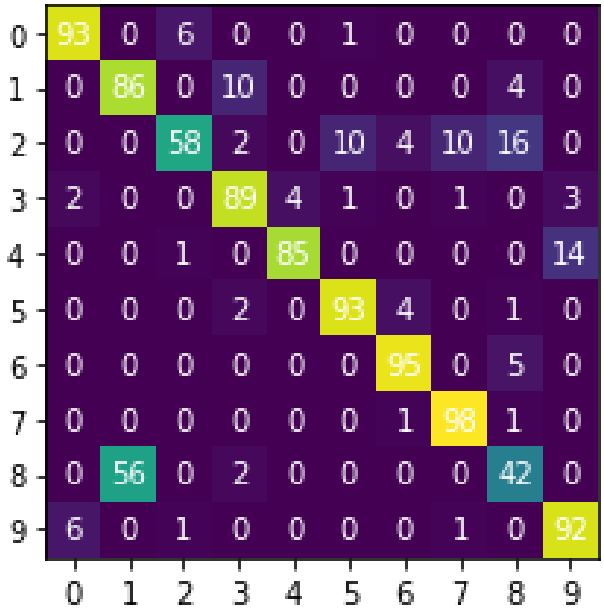
Confusion matrix of the highest accuracy reached on train set
multi layer perceptron
h= 5
accuracy= %32.0



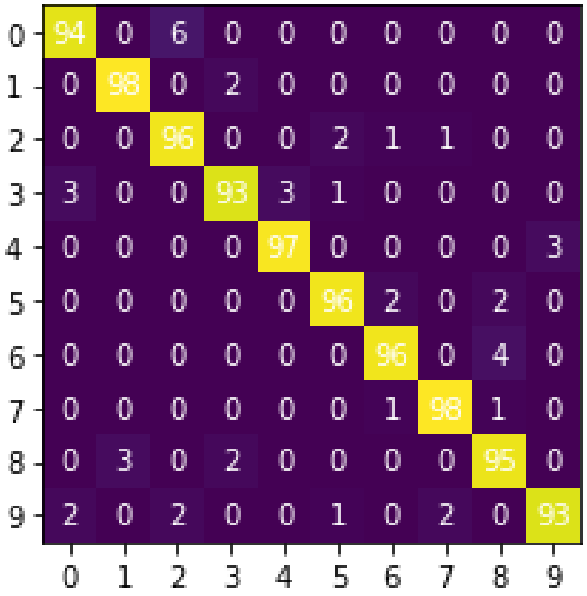
Confusion matrix of the highest accuracy reached on train set
multi layer perceptron
h= 10
accuracy= %70.7

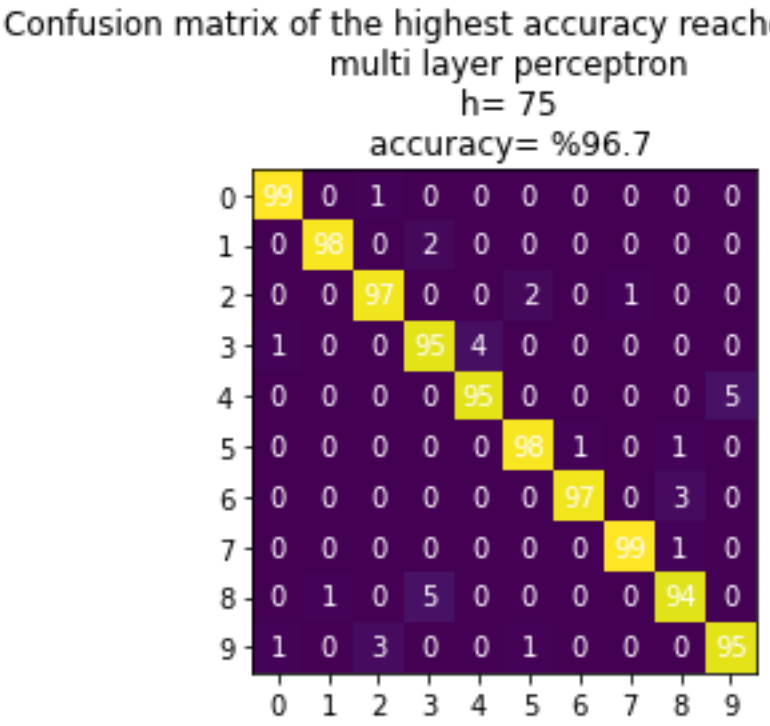


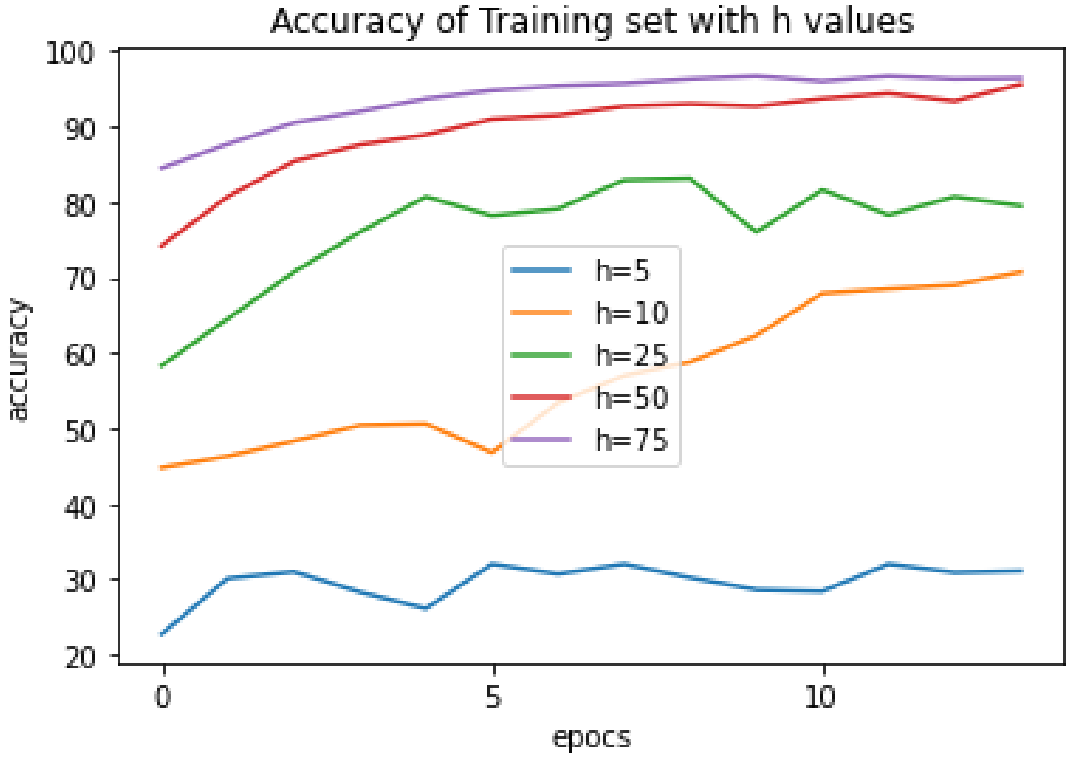
Confusion matrix of the highest accuracy reached on train set
multi layer perceptron
h= 25
accuracy= %83.1

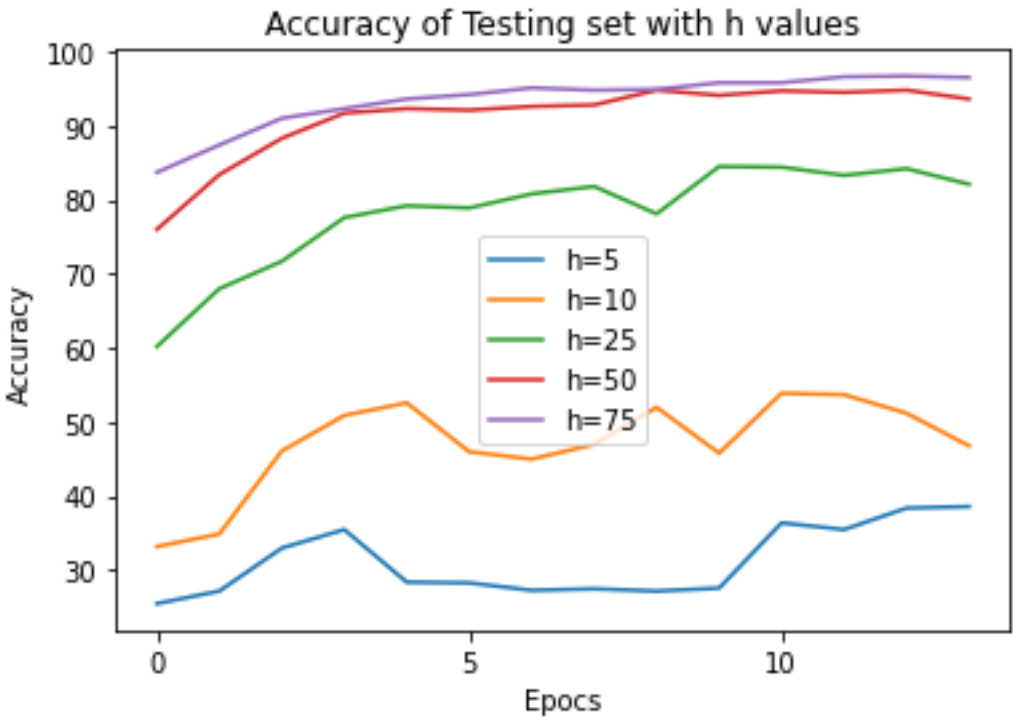


Confusion matrix of the highest accuracy reached on train set
multi layer perceptron
h= 50
accuracy= %95.6

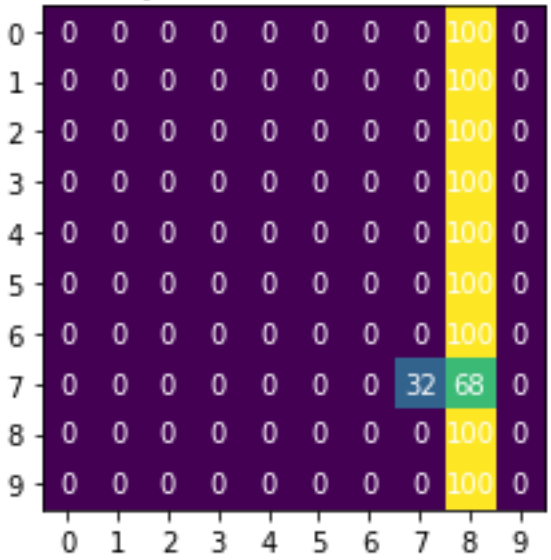



Confusion matrix of the highest accuracy reached on train set
multi layer perceptron
h= 75
accuracy= %96.7
0 99 0 1 0 0 0 0 0 0 0
1 0 98 0 2 0 0 0 0 0 0
2 0 0 97 0 0 2 0 1 0 0
3 1 0 0 95 4 0 0 0 0 0
4 0 0 0 0 95 0 0 0 0 5
5 0 0 0 0 0 98 1 0 1 0
6 0 0 0 0 0 0 97 0 3 0
7 0 0 0 0 0 0 0 99 1 0
8 0 1 0 5 0 0 0 0 94 0
9 1 0 3 0 0 1 0 0 0 95
0 1 2 3 4 5 6 7 8 9

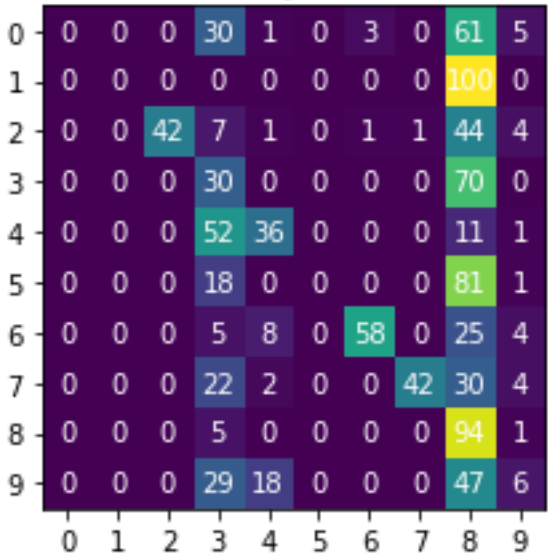

Accuracy of Training set with h values
accuracy
epocs
h=5
h=10
h=25
h=50
h=75
100
90
80
70
60
50
40
30
20
0
5
10



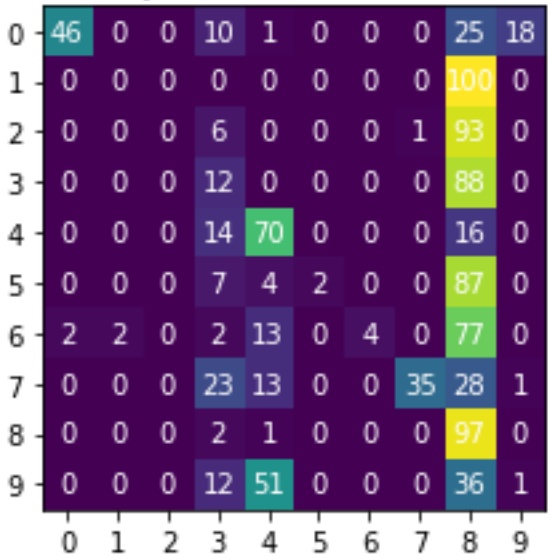
Confusion matrix of the highest accuracy reached on test set
multi layer perceptron
h= 5
accuracy= %13.200000000000001



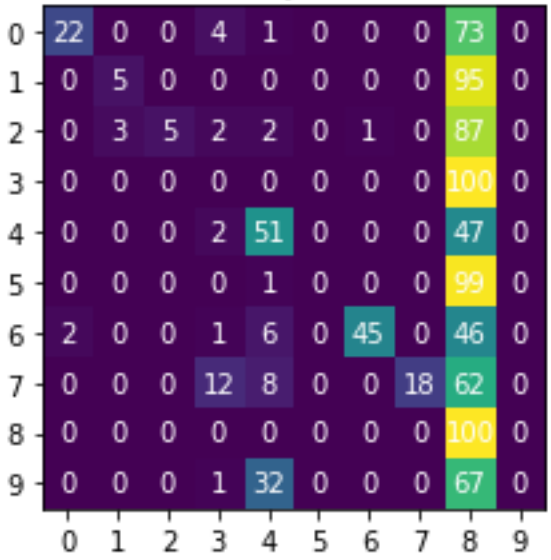
Confusion matrix of the highest accuracy reached on test set
multi layer perceptron
h= 10
accuracy= %30.8



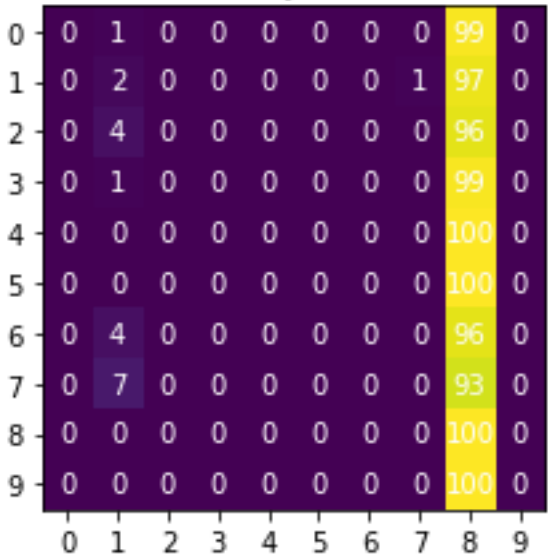
Confusion matrix of the highest accuracy reached on test set
multi layer perceptron
h= 25
accuracy= %26.700000000000003



Confusion matrix of the highest accuracy reached on test set
multi layer perceptron
h= 50
accuracy= %24.6



Confusion matrix of the highest accuracy reached on test set
multi layer perceptron
h= 75
accuracy= %10.2



CODING PART

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import random

training_file = pd.read_csv("training.csv", header=None)
testing_file = pd.read_csv("testing.csv", header=None)

training_labels = training_file.iloc[:, 0]
training_images = training_file.iloc[:, 1:]

testing_labels = testing_file.iloc[:, 0]
testing_images = testing_file.iloc[:, 1:]

total_index = len(training_images)  ## 1000
classes_length = len(np.unique(training_labels))  ## 10
dimensions = len(training_images.columns)  ## 100

# PART A

def random_weights():
    weights = np.zeros([classes_length, dimensions])
    for i in range(classes_length):
        for j in range(dimensions):
            rd = random.uniform(-0.01, 0.01)
            weights[i][j] = rd

    return weights


def compute_confusion_matrix(true, pred):
    K = len(np.unique(true))  # Number of classes
    result = np.zeros((K, K))

    for i in range(len(true)):
        result[true[i]][pred[i]] += 1

    return result


def accuracy_percentage(matrix, number):
    temp = 0
    length = len(matrix)
    for i in range(length):
        temp = temp + matrix[i][i]

    return (temp / number) * 100
```

```
def confusion_matrix_table(conf_matrix, slp, train, h):
    numbers = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

    fig, ax = plt.subplots()
    im = ax.imshow(conf_matrix)

    ax.set_xticks(np.arange(len(numbers)))
    ax.set_yticks(np.arange(len(numbers)))

    ax.set_xticklabels(numbers)
    ax.set_yticklabels(numbers)

    plt.setp(ax.get_xticklabels())

    for i in range(len(numbers)):
        for j in range(len(numbers)):
            text = ax.text(j, i, conf_matrix[i, j],
                           ha="center", va="center", color="w")

    if slp and train:
        acc = accuracy_percentage(conf_matrix, 1000)
        ax.set_title(
            "Confusion matrix of the highest accuracy reached on train
set\n single layer perceptron\n accuracy= %" + str(
                acc))
    if slp and not train:
        acc = accuracy_percentage(conf_matrix, 1000)
        ax.set_title(
            "Confusion matrix of the highest accuracy reached on test set\n
single layer perceptron\n accuracy= %" + str(
                acc))
    if not slp and train:
        acc = accuracy_percentage(conf_matrix, 1000)
        ax.set_title(
            "Confusion matrix of the highest accuracy reached on train
set\n multi layer perceptron\n h= " + str(
                h) + '\n accuracy= %' + str(acc))
    if not slp and not train:
        acc = accuracy_percentage(conf_matrix, 1000)
        ax.set_title(
            "Confusion matrix of the highest accuracy reached on test set\n
multi layer perceptron\n h= " + str(
                h) + '\n accuracy= %' + str(acc))

    fig.tight_layout()
    plt.show()


weights = random_weights()  ## Using for single_layer_perceptron_training
and testing
```

```
def single_layer_perceptron_training(number_epocs):
    weights_training = weights
    accuracy_rates = []
    epoc = 0
    best_weights = np.zeros([classes_length, dimensions])
    max_accuracy = 0
    max_accuracy_confusion_matrix = []

    while epoc < number_epocs:

        orig = weights_training

        d_w = np.zeros([classes_length, dimensions])

        true_prediction = 0
        predictions = []

        for n in range(total_index):

            label_index = training_labels[n]
            o = np.zeros(classes_length)
            images_per_row = training_images.loc[n]
            labels = np.zeros(classes_length)
            labels[label_index] = 1

            for i in range(classes_length):
                dot_product = np.dot(images_per_row, weights_training[i])
                o[i] = o[i] + dot_product

            y = np.zeros(classes_length)

            ##Softmax
            Omax = np.max(o)
            a = o - Omax
            for i in range(classes_length):
                numerator = np.exp(o[i] - Omax)
                denominator = np.sum(np.exp(a))
                y[i] = numerator / denominator

            prediction = np.argmax(y)
            predictions.append(prediction)

            if prediction == label_index:
                true_prediction = true_prediction + 1

            for i in range(classes_length):
                diff = labels[i] - y[i]
                d_w[i] = d_w[i] + np.dot(diff, images_per_row)

        weights_training = weights_training + 0.1 * d_w  ## 0.1 is Learning
factor

        accuracy_rate = (true_prediction / total_index) * 100
        accuracy_rates.append(accuracy_rate)

        if accuracy_rate > max_accuracy:
            max_accuracy_confusion_matrix =
compute_confusion_matrix(training_labels, predictions)
            max_accuracy = accuracy_rate
            best_weights = weights_training
```

```
            print('Best weights has been changed')

        print('Epoc = ' + str(epoc) + '\t' + ' Accuracy= ' +
str(accuracy_rate) + '\n')

        epoc = epoc + 1

    return best_weights, max_accuracy_confusion_matrix, accuracy_rates



epocs_count = 50
best_weights_slp_training, max_accuracy_confusion_matrix_training,
accuracy_rates_training = single_layer_perceptron_training(
    epocs_count)
confusion_matrix_table(max_accuracy_confusion_matrix_training.astype('int32
'), True, True, h=0)

plt.plot(np.arange(0, 50), accuracy_rates_training)
plt.xlabel('Epocs')
plt.ylabel('Accuracy')
plt.title('Training Set Single Layer Perceptron Epoch: ' +
str(epocs_count))
plt.show()


def single_layer_perceptron_testing(number_epocs):
    weights_testing = weights
    accuracy_rates = []
    epoc = 0
    best_weights = np.zeros([classes_length, dimensions])
    max_accuracy = 0
    max_accuracy_confusion_matrix = []

    while epoc < number_epocs:

        orig = weights_testing

        d_w = np.zeros([classes_length, dimensions])

        true_prediction = 0
        predictions = []

        for n in range(total_index):

            label_index = training_labels[n]
            o = np.zeros(classes_length)
            images_per_row = testing_images.loc[n]
            labels = np.zeros(classes_length)
            labels[label_index] = 1

            for i in range(classes_length):
                dot_product = np.dot(images_per_row, weights_testing[i])
                o[i] = o[i] + dot_product

            y = np.zeros(classes_length)

            ##Softmax
            Omax = np.max(o)
            a = o - Omax
            for i in range(classes_length):
```

```
                numerator = np.exp(o[i] - Omax)
                denominator = np.sum(np.exp(a))
                y[i] = numerator / denominator

            prediction = np.argmax(y)
            predictions.append(prediction)

            if prediction == label_index:
                true_prediction = true_prediction + 1

            for i in range(classes_length):
                diff = labels[i] - y[i]
                d_w[i] = d_w[i] + np.dot(diff, images_per_row)

        weights_testing = weights_testing + 0.1 * d_w  ## 0.1 is Learning
factor

        accuracy_rate = (true_prediction / total_index) * 100
        accuracy_rates.append(accuracy_rate)

        epoc = epoc + 1

    return accuracy_rates



epocs_count = 50
accuracy_rates_testing = single_layer_perceptron_testing(epocs_count)

plt.plot(np.arange(0, 50), accuracy_rates_testing)
plt.xlabel('Epocs')
plt.ylabel('Accuracy')
plt.title('Testing Set Single Layer Perceptron Epoch: ' + str(epocs_count))
plt.show()

def single_layer_perceptron_prediction(best_weights):
    n_test = len(testing_images)

    predictions_test = []
    for n in range(n_test):

        x = testing_images.loc[n]
        o = np.zeros(classes_length)

        for i in range(classes_length):
            dot_product = np.dot(x, best_weights[i])
            o[i] = o[i] + dot_product

        y = np.zeros(classes_length)
        ##Softmax
        Omax = np.max(o)
        a = o - Omax
        for i in range(classes_length):
            top = np.exp(o[i] - Omax)
            bottom = np.sum(np.exp(a))
            y[i] = top / bottom

        predicted = np.argmax(y)
        predictions_test.append(predicted)

    return predictions_test
```

```
predictions_test =
single_layer_perceptron_prediction(best_weights_slp_training)

confusion_matrix = compute_confusion_matrix(testing_labels,
predictions_test)

confusion_matrix_table(confusion_matrix.astype('int32'), True, False, h=0)

w = 10
h = 10


fig = plt.figure(figsize=(10, 10))
columns = 2
rows = 5
for i in range(1, columns * rows + 1):
    img = best_weights_slp_training[i - 1].reshape(10, 10)
    fig.add_subplot(rows, columns, i)
    plt.title('Image ' + str(i - 1), fontsize=(15))
    plt.imshow(img, cmap='binary')
    plt.tight_layout()

plt.show()


### PART B

number_train_inputs = len(training_images)


def random_mlp_weights(h_size):
    v = np.zeros([classes_length, h_size])
    w = np.zeros([h_size, dimensions])

    for i in range(h_size):
        for j in range(dimensions):
            random_num = random.uniform(-0.01, 0.01)
            w[i][j] = random_num

    for i in range(classes_length):
        for j in range(h_size):
            random_num = random.uniform(-0.01, 0.01)
            v[i][j] = random_num

    return w, v


def mlp_training(epocs, h_len):
    w, v = random_mlp_weights(h_len)

    epoc = 0
    max_accuracy = 0
    best_v = np.zeros([classes_length, h_len])
    best_w = np.zeros([h_len, dimensions])

    accuracy_mlp = []
    highest_acc_conf_matrix = []

    while epoc < epocs:
        correct_scores = 0
        predictions = []
```

```
        for t in range(number_train_inputs):

            x = training_images.loc[t]
            actual = training_labels[t]
            r = np.zeros(classes_length)
            r[actual] = 1

            z = np.zeros(h_len)
            for i in range(h_len):
                w_t = np.transpose(w[i])
                o = np.dot(w_t, x)
                sigmoid = 1 / (1 + np.exp(-o))
                z[i] = sigmoid

            y = np.zeros(classes_length)
            for i in range(classes_length):
                v_t = np.transpose(v[i])
                y[i] = np.dot(v_t, z)

            dis_v = np.zeros([classes_length, h_len])
            for i in range(classes_length):
                diff = r[i] - y[i]
                dis_v[i] = 0.001 * np.dot(diff, z)

            dis_w = np.zeros([h_len, dimensions])
            for h in range(h_len):
                sum_of = np.zeros(classes_length)
                for i in range(classes_length):
                    diff = r[i] - y[i]
                    sum_of[i] = diff * v[i][h]

                sum_total = np.sum(sum_of)
                inner_1 = np.dot(sum_total, z[h])
                inner_2 = np.dot(inner_1, 1 - z[h])
                inner_3 = np.dot(inner_2, x)
                dis_w[h] = 0.001 * inner_3

            predicted = np.argmax(y)
            predictions.append(predicted)
            if predicted == actual:
                correct_scores = correct_scores + 1

            v = v + dis_v
            w = w + dis_w

        accuracy = (correct_scores / number_train_inputs) * 100
        accuracy_mlp.append(accuracy)

        if accuracy > max_accuracy:
            max_accuracy = accuracy
            best_v = v
            best_w = w
            highest_acc_conf_matrix =
compute_confusion_matrix(training_labels, predictions)
            print('highest accuracy has been changed')

        epoc = epoc + 1
        print('Epoc= ' + str(epoc) + '\t' + 'Accuracy= %' + str(accuracy) +
'\n')
```

```
    return best_v, best_w, highest_acc_conf_matrix, accuracy_mlp



best_v_h_5, best_w_h_5, highest_acc_conf_matrix_h_5, accs_mlp_h_5 =
mlp_training(15, 5)
best_v_h_10, best_w_h_10, highest_acc_conf_matrix_h_10, accs_mlp_h_10 =
mlp_training(15, 10)
best_v_h_25, best_w_h_25, highest_acc_conf_matrix_h_25, accs_mlp_h_25 =
mlp_training(15, 25)
best_v_h_50, best_w_h_50, highest_acc_conf_matrix_h_50, accs_mlp_h_50 =
mlp_training(15, 50)
best_v_h_75, best_w_h_75, highest_acc_conf_matrix_h_75, accs_mlp_h_75 =
mlp_training(15, 75)


confusion_matrix_table(highest_acc_conf_matrix_h_5.astype('int32'), False,
True, h=5)
confusion_matrix_table(highest_acc_conf_matrix_h_10.astype('int32'), False,
True, h=10)
confusion_matrix_table(highest_acc_conf_matrix_h_25.astype('int32'), False,
True, h=25)
confusion_matrix_table(highest_acc_conf_matrix_h_50.astype('int32'), False,
True, h=50)
confusion_matrix_table(highest_acc_conf_matrix_h_75.astype('int32'), False,
True, h=75)


plt_1 = plt.plot(np.arange(0, 14), accs_mlp_h_5[1:], label='h=5')
plt_2 = plt.plot(np.arange(0, 14), accs_mlp_h_10[1:], label='h=10')
plt_3 = plt.plot(np.arange(0, 14), accs_mlp_h_25[1:], label='h=25')
plt_4 = plt.plot(np.arange(0, 14), accs_mlp_h_50[1:], label='h=50')
plt_5 = plt.plot(np.arange(0, 14), accs_mlp_h_75[1:], label='h=75')


plt.xlabel('epocs')
plt.ylabel('accuracy')
plt.title('Accuracy of Training set with h values')
plt.legend()
plt.xticks(np.arange(0, 14, 5))
plt.show()


def mlp_testing(epocs, h_len):
    w, v = random_mlp_weights(h_len)

    epoc = 0
    max_accuracy = 0
    best_v = np.zeros([classes_length, h_len])
    best_w = np.zeros([h_len, dimensions])

    accuracy_mlp = []
    highest_acc_conf_matrix = []

    while epoc < epocs:
        correct_scores = 0
        predictions = []

        for t in range(number_train_inputs):
```

```
            x = testing_images.loc[t]
            actual = testing_labels[t]
            r = np.zeros(classes_length)
            r[actual] = 1

            z = np.zeros(h_len)
            for i in range(h_len):
                w_t = np.transpose(w[i])
                o = np.dot(w_t, x)
                sigmoid = 1 / (1 + np.exp(-o))
                z[i] = sigmoid

            y = np.zeros(classes_length)
            for i in range(classes_length):
                v_t = np.transpose(v[i])
                y[i] = np.dot(v_t, z)

            dis_v = np.zeros([classes_length, h_len])
            for i in range(classes_length):
                diff = r[i] - y[i]
                dis_v[i] = 0.001 * np.dot(diff, z)

            dis_w = np.zeros([h_len, dimensions])
            for h in range(h_len):
                sum_of = np.zeros(classes_length)
                for i in range(classes_length):
                    diff = r[i] - y[i]
                    sum_of[i] = diff * v[i][h]

                sum_total = np.sum(sum_of)
                inner_1 = np.dot(sum_total, z[h])
                inner_2 = np.dot(inner_1, 1 - z[h])
                inner_3 = np.dot(inner_2, x)
                dis_w[h] = 0.001 * inner_3

            predicted = np.argmax(y)
            predictions.append(predicted)
            if predicted == actual:
                correct_scores = correct_scores + 1

            v = v + dis_v
            w = w + dis_w

        accuracy = (correct_scores / number_train_inputs) * 100
        accuracy_mlp.append(accuracy)

        if accuracy > max_accuracy:
            max_accuracy = accuracy
            best_v = v
            best_w = w
            highest_acc_conf_matrix =
compute_confusion_matrix(training_labels, predictions)
            print('highest accuracy has been changed')

        epoc = epoc + 1
        print('Epoc= ' + str(epoc) + '\t' + 'Accuracy= %' + str(accuracy) +
'\n')

    return best_v, best_w, highest_acc_conf_matrix, accuracy_mlp
```

```
best_v_h_5_test, best_w_h_5_test, highest_acc_conf_matrix_h_5_test,
accs_mlp_h_5 = mlp_training(15, 5)
best_v_h_10_test, best_w_h_10_test, highest_acc_conf_matrix_h_10_test,
accs_mlp_h_10 = mlp_training(15, 10)
best_v_h_25_test, best_w_h_25_test, highest_acc_conf_matrix_h_25_test,
accs_mlp_h_25 = mlp_training(15, 25)
best_v_h_50_test, best_w_h_50_test, highest_acc_conf_matrix_h_50_test,
accs_mlp_h_50 = mlp_training(15, 50)
best_v_h_75_test, best_w_h_75, highest_acc_conf_matrix_h_75_test,
accs_mlp_h_75 = mlp_training(15, 75)


plt_1 = plt.plot(np.arange(0, 14), accs_mlp_h_5[1:], label='h=5')
plt_2 = plt.plot(np.arange(0, 14), accs_mlp_h_10[1:], label='h=10')
plt_3 = plt.plot(np.arange(0, 14), accs_mlp_h_25[1:], label='h=25')
plt_4 = plt.plot(np.arange(0, 14), accs_mlp_h_50[1:], label='h=50')
plt_5 = plt.plot(np.arange(0, 14), accs_mlp_h_75[1:], label='h=75')

plt.xlabel('Epocs')
plt.ylabel('Accuracy')
plt.title('Accuracy of Testing set with h values')
plt.legend()
plt.xticks(np.arange(0, 14, 5))
plt.show()



def predict_mlp(best_v, best_w, h_len):
    n_test = len(testing_images)

    predictions_test = []
    for t in range(n_test):
        x = testing_images.loc[t]

        z = np.zeros(h_len)
        for i in range(h_len):
            w_t = np.transpose(best_w[i])
            o = np.dot(w_t, x)
            sigmoid = 1 / (1 + np.exp(-o))
            z[i] = sigmoid

        y = np.zeros(classes_length)
        for i in range(classes_length):
            v_t = np.transpose(best_v[i])
            y[i] = np.dot(v_t, z)

        predicted = np.argmax(y)
        predictions_test.append(predicted)

    return predictions_test
```

```
predictions_test_h_5 = predict_mlp(best_v_h_5, best_w_h_5, 5)
predictions_test_h_10 = predict_mlp(best_v_h_10, best_w_h_10, 10)
predictions_test_h_25 = predict_mlp(best_v_h_25, best_w_h_25, 25)
predictions_test_h_50 = predict_mlp(best_v_h_50, best_w_h_50, 50)
predictions_test_h_75 = predict_mlp(best_v_h_75, best_w_h_75, 75)

confusion_matrix_h_5 = compute_confusion_matrix(testing_labels,
predictions_test_h_5)
confusion_matrix_h_10 = compute_confusion_matrix(testing_labels,
predictions_test_h_10)
confusion_matrix_h_25 = compute_confusion_matrix(testing_labels,
predictions_test_h_25)
confusion_matrix_h_50 = compute_confusion_matrix(testing_labels,
predictions_test_h_50)
confusion_matrix_h_75 = compute_confusion_matrix(testing_labels,
predictions_test_h_75)

confusion_matrix_table(confusion_matrix_h_5.astype('int32'), False, False,
h=5)
confusion_matrix_table(confusion_matrix_h_10.astype('int32'), False, False,
h=10)
confusion_matrix_table(confusion_matrix_h_25.astype('int32'), False, False,
h=25)
confusion_matrix_table(confusion_matrix_h_50.astype('int32'), False, False,
h=50)
confusion_matrix_table(confusion_matrix_h_75.astype('int32'), False, False,
h=75)
```