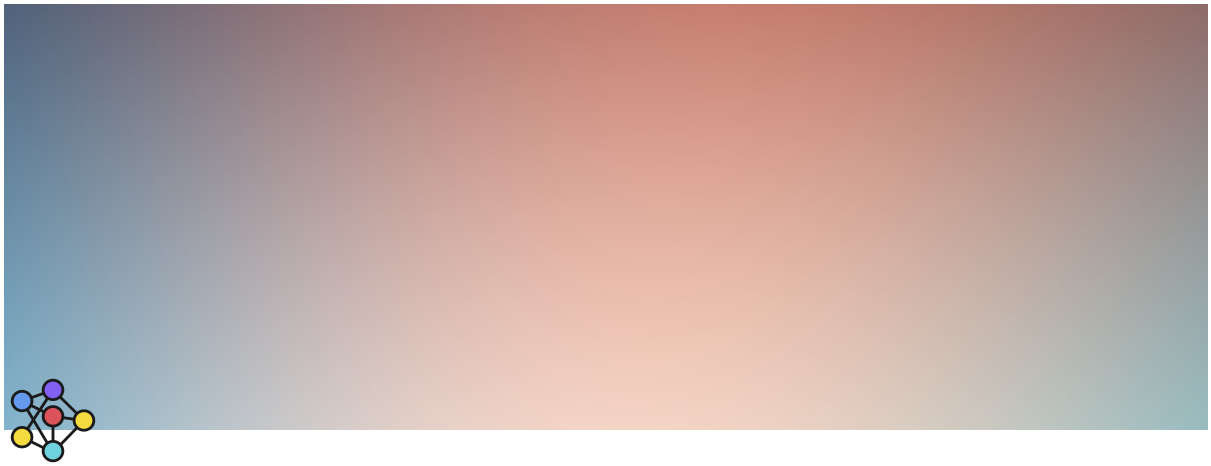


ICS381 PA02

Table of Contents

Omar Jarallah Alghamdi. 201855000. ICS381-02

Problem Statement

It is required to develop a python program that prompts the user to insert the USPS training dataset and generate a model that has only one hidden layer to recognize Arabic handwritten digits. The user should be able to spcify the number of hidden neurons and the learning rate. After the user has trained the model, the program should prompt the user to insert the USPS testing dataset. The program should show the following data: loss vs epochs graph, confusion matrix, training time.

Codes

GUI/main/MyApp()

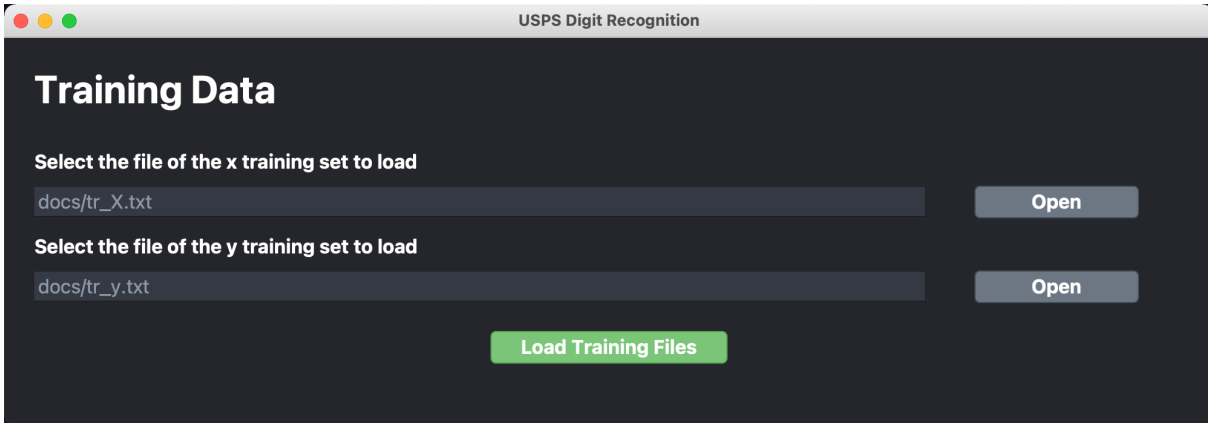
MyApp class is the class that defines the tkinter based aplication and is responsible for running it.

```
def __init__(self):
        self.cntrl = Controller.Controller()
        root = tk.Tk()
        root.geometry('1000x320')
        root.update()
        root.configure(bg="#25262B")
        root.title('USPS Digit Recognition')
        training_data_label = tk.Label(root, text="Training Data",
                                       font=("Great Vibes", 32, "bold"), foreground='#FFFFFF', background='#25262
B')
        training_data_label.place(x=20, y=20)
        # load train_x section
        trx_load_label = tk.Label(root, text="Select the file of the x training set to load",
                                  font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262B')
        trx_load_label.place(x=20, y=90)
        trx_path_field = tk.Entry(root, width=73, font=("Great Vibes", 16), highlightbackground='#25262B',
                                  foreground='#8f98a6', background='#353943')
        trx_path_field.place(x=22, y=120)
        trx_path_field.insert(0, 'docs/tr_X.txt')
        trx_open_button = Button(root, text="Open", command=lambda: self.fill_text_field(trx_path_field),
                                 font=("Great Vibes", 16, "bold"), width=140, height=30, bg='#6D7784', fg='#FFFFF
F',
                                 borderless=True, activeforeground='#FFFFFF', activebackground='#88929D',
                                 focusthickness=0)
        trx_open_button.place(x=800, y=120)
        # load train_y section
        try_load_label = tk.Label(root, text="Select the file of the y training set to load",
                                  font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262B')
        try_load_label.place(x=20, y=160)
        try_path_field = tk.Entry(root, width=73, font=("Great Vibes", 16), foreground='#8f98a6',
                                  background='#353943', highlightbackground='#25262B')
        try_path_field.place(x=22, y=190)
        try_path_field.insert(0, 'docs/tr_y.txt')
        try_open_button = Button(root, text="Open", command=lambda: self.fill_text_field(try_path_field),
                                 font=("Great Vibes", 16, "bold"), width=140, height=30, bg='#6D7784', fg='#FFFFF
F',
                                 borderless=True, activeforeground='#FFFFFF', activebackground='#88929D',
                                 focusthickness=0)
        try_open_button.place(x=800, y=190)
        tr_load_button = Button(root, text="Load Training Files",
                                command=lambda: self.load_training(trx_path_field.get(), try_path_field.get(), roo
t),
                                font=("Great Vibes", 16, "bold"), width=200, height=30, bg='#7AC578', fg='#FFFFF
F',
                                borderless=True, activeforeground='#FFFFFF', activebackground='#AAD1A9',
                                focusthickness=0)
        tr_load_button.place(x=400, y=240)
        root.mainloop()
```

`__init__` is the constructor or the method that initialize the program. The program will prompt the user to enter directory of the training dataset so it can load them. The program will have the dataset previously so the user

will only need to load them. The program also provide the `open` functionality to make the file navigation easier. The the user will click on `Load training Files` to proceed.

Image of the result of this method:



Fill text field function

```
def fill_text_field(self, tf):
        path = self.cntrl.open_directory()
        tf.delete(0, tk.END)
        tf.insert(0, path)
        return
```

`fill_text_field` is the function taht takes a `tkinter Entry` and insert a directory to the file slected by `open_directory()` function into it.

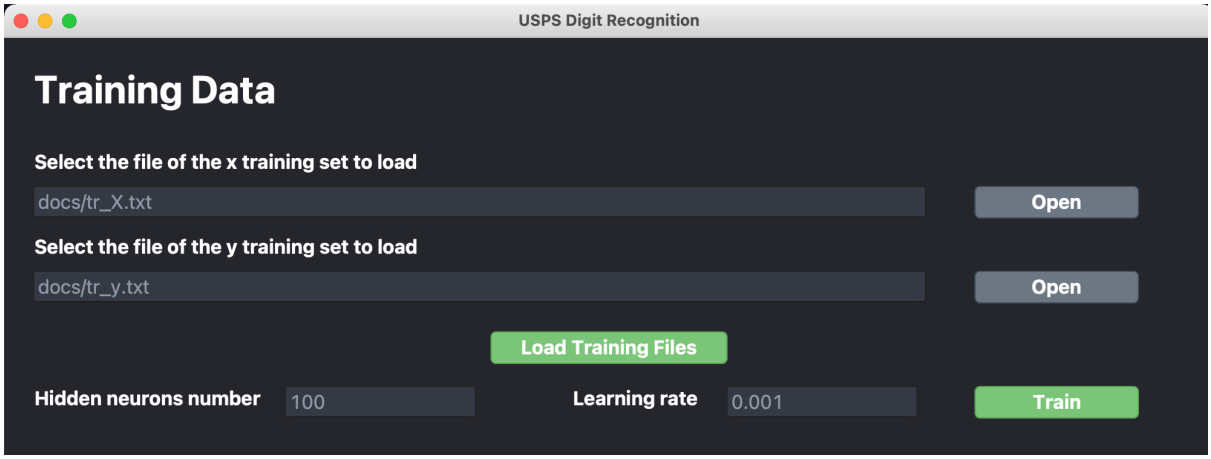
Load training function

```
def load_training(self, trx_path, try_path, root):
        if len(trx_path) != 0 and len(try_path) != 0:
            self.cntrl.load_training_files(trx_path, try_path)
            root.geometry('1000x350')
            root.update()
            hidden_neurons_label = tk.Label(root, text="Hidden neurons number",
                                            font=("Great Vibes", 16, "bold"), foreground='#FFFFFF',
                                            background='#25262B')
            hidden_neurons_label.place(x=20, y=285)
            hidden_neurons_field = tk.Entry(root, width=15, font=("Great Vibes", 16), foreground='#8f98a6',
                                            background='#353943', highlightbackground='#25262B')
            hidden_neurons_label.place(x=20, y=285)
            hidden_neurons_field.place(x=230, y=285)
            hidden_neurons_field.insert(0,100)
            learning_rate_label = tk.Label(root, text="Learning rate",
                                           font=("Great Vibes", 16, "bold"), foreground='#FFFFFF',
                                           background='#25262B')
            learning_rate_label.place(x=465, y=285)
            learning_rate_field = tk.Entry(root, width=15, font=("Great Vibes", 16), foreground='#8f98a6',
                                           background='#353943', highlightbackground='#25262B')
            learning_rate_field.insert(0, 0.001)
            learning_rate_field.place(x=595, y=285)
            train_button = Button(root, text="Train", command=lambda: self.train(hidden_neurons_field.get(),
```

```
                                                                learning_rate_field.get(), root),
                                font=("Great Vibes", 16, "bold"), width=140, height=30, bg='#7AC578', fg='#FFFFFF',
                                borderless=True, activeforeground='#FFFFFF', activebackground='#AAD1A9',
                                focusthickness=0)
            train_button.place(x=800, y=285)
```

`load_training` is the function that takes the `tr_X.txt` and `tr_y.txt` pathes, read contents of them and convert them to a format that `sklearn.MLPClassifier` can accept them then save them to `Controls/Controller` class so they can be used later. Then the program will update the scale of the window to add the remaining fields that prompt the user to insert the `neuron units` (default = 100) and the `learning rate` (default = 0.001). The the user will click on `Train` to proceed.

Image of the result of this method:

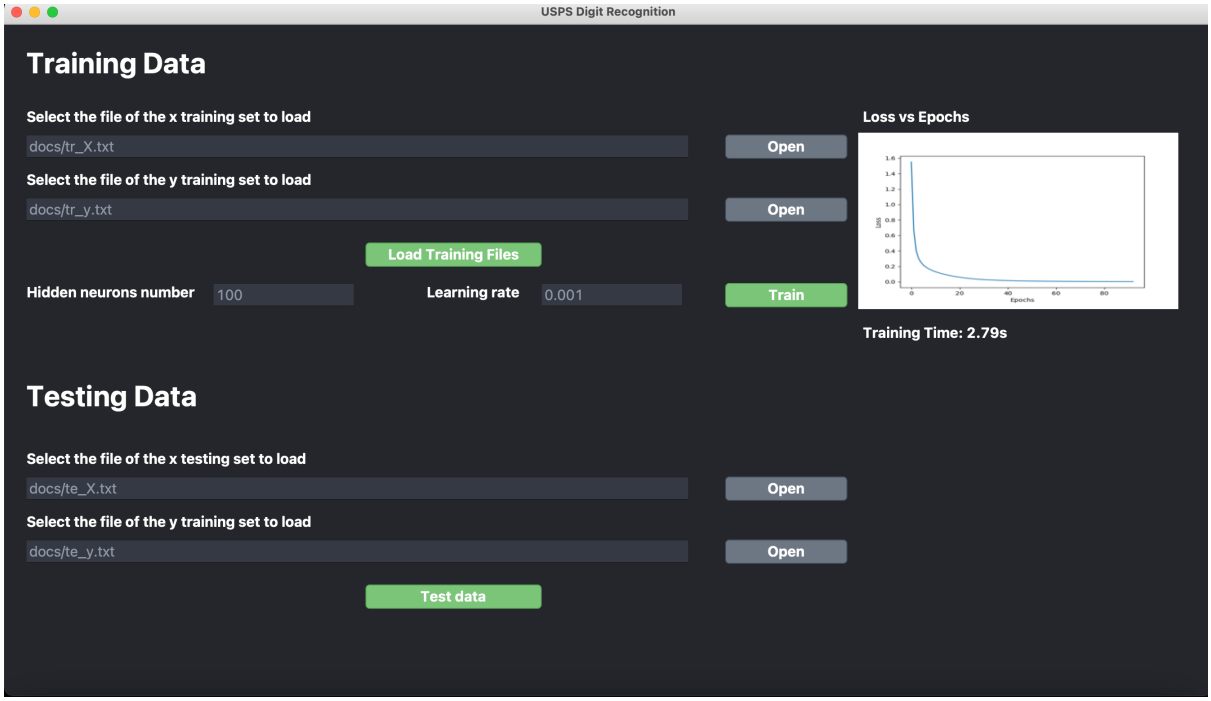


Train function

```
def train(self, hidden_neurons, learning_rate, root):
        if len(hidden_neurons) != 0 or len(learning_rate) != 0 or learning_rate == 0 or hidden_neurons == 0:
            time = self.cntrl.train(hidden_neurons, learning_rate)
            self.plot(root, time)
            self.test(root)
```

`train` is the function taht takes the `hidden neurons` and `learning rate` values from the user then sends them to `Controlrs/Controller.train` function to generate a model train it abd retrieve the training time. Then it will call `plot` funtion to plot the `Loss vs Epochs` graph using `matlibplot` library. Finally it will call the `test` function which will include the test files text field (Entry in tkinter) to make the user fill their directories. Then it will call `plot` funtion.

Image of the result of this method:



Plot function

```
def plot(self, root, time):
        loss_vs_epoch_label = tk.Label(root, text="Loss vs Epochs",
                                       font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262
B')
        loss_vs_epoch_label.place(x=950, y=90)
        training_time_label = tk.Label(root, text="Training Time: " + time,
                                       font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262
B')
        training_time_label.place(x=950, y=330)
        root.geometry("1370x900")
        root.update()
        image = Image.open("assets/loss_vs_epoch.png")
        resized = image.resize((350, 190))
        img = ImageTk.PhotoImage(resized)
        panel = tk.Label(root, image=img)
        panel.image = img
        panel.place(x=950, y=120)
```

`plot` funtion is the function taht takes the `time` and will plot the `Loss vs Epochs` and the `Time` that are generated from `train` function.

Test function

```
def test(self, root):
        testing_data_label = tk.Label(root, text="Testing Data",
                                      font=("Great Vibes", 32, "bold"), foreground='#FFFFFF', background='#25262
B')
        testing_data_label.place(x=20, y=390)
        # load test_x section
        tex_load_label = tk.Label(root, text="Select the file of the x testing set to load",
                                  font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262B')
        tex_load_label.place(x=20, y=470)
        tex_path_field = tk.Entry(root, width=73, font=("Great Vibes", 16), highlightbackground='#25262B',
                                  foreground='#8f98a6', background='#353943')
```

```
        tex_path_field.place(x=22, y=500)
        tex_path_field.insert(0, 'docs/te_X.txt')
        tex_open_button = Button(root, text="Open", command=lambda: self.fill_text_field(tex_path_field),
                                 font=("Great Vibes", 16, "bold"), width=140, height=30, bg='#6D7784', fg='#FFFFF
F',
                                 borderless=True, activeforeground='#FFFFFF', activebackground='#88929D',
                                 focusthickness=0)
        tex_open_button.place(x=800, y=500)
        # load test_y section
        tey_load_label = tk.Label(root, text="Select the file of the y training set to load",
                                  font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262B')
        tey_load_label.place(x=20, y=540)
        tey_path_field = tk.Entry(root, width=73, font=("Great Vibes", 16), foreground='#8f98a6',
                                  background='#353943', highlightbackground='#25262B')
        tey_path_field.place(x=22, y=570)
        tey_path_field.insert(0, 'docs/te_y.txt')
        tey_open_button = Button(root, text="Open", command=lambda: self.fill_text_field(tey_path_field),
                                 font=("Great Vibes", 16, "bold"), width=140, height=30, bg='#6D7784', fg='#FFFFF
F',
                                 borderless=True, activeforeground='#FFFFFF', activebackground='#88929D',
                                 focusthickness=0)
        tey_open_button.place(x=800, y=570)
        te_load_button = Button(root, text="Test data",
                                command=lambda: self.load_testing(tex_path_field.get(), tey_path_field.get(), roo
t),
                                font=("Great Vibes", 16, "bold"), width=200, height=30, bg='#7AC578', fg='#FFFFF
F',
                                borderless=True, activeforeground='#FFFFFF', activebackground='#AAD1A9',
                                focusthickness=0)
        te_load_button.place(x=400, y=620)
```

`test` is the function that displays the text fields that are for testing files directories. it will prompt the user to enter the dire

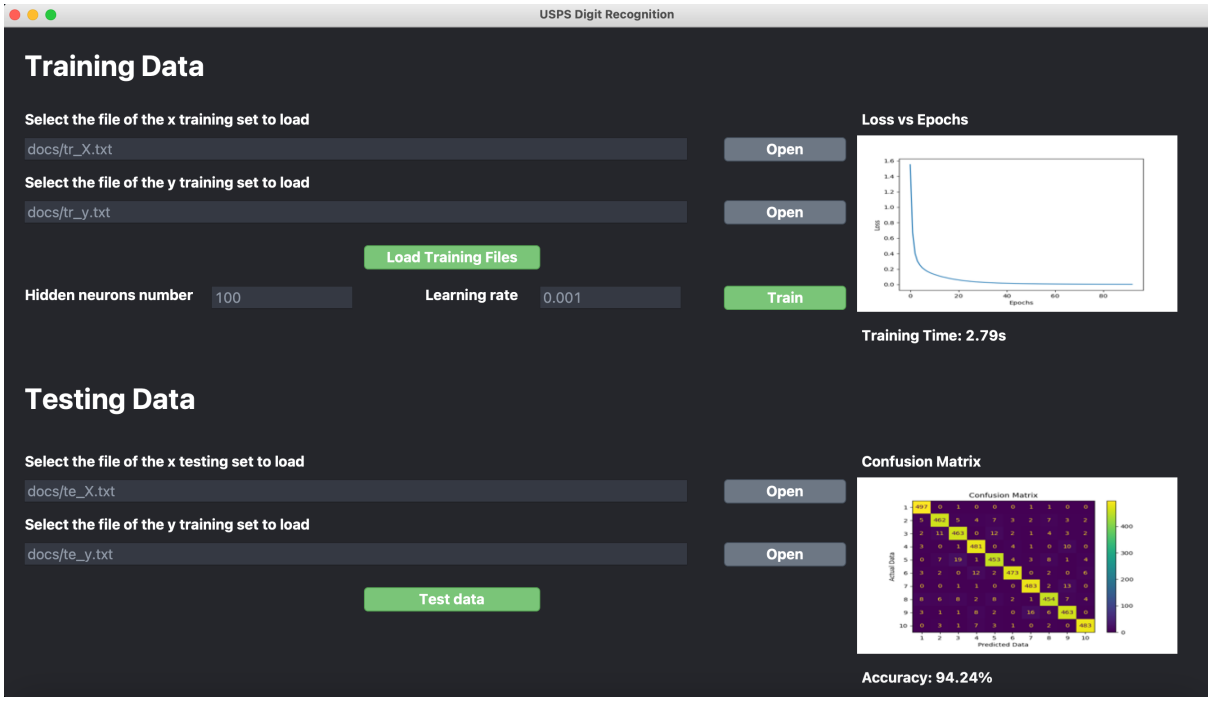
Load testing function

```
def load_testing(self, tex_path, tey_path, root):
        self.cntrl.load_testing_files(tex_path, tey_path)
        accuracy = self.cntrl.test()
        confusion_matrix_label = tk.Label(root, text="Confusion Matrix",
                                          font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262
B')
        confusion_matrix_label.place(x=950, y=470)
        accuracy_label = tk.Label(root, text="Accuracy: " + accuracy,
                                          font=("Great Vibes", 16, "bold"), foreground='#FFFFFF', background='#25262
B')
        accuracy_label.place(x=950, y=710)
        image = Image.open("assets/Confusion_Matrix.png")
        resized = image.resize((350, 190))
        img = ImageTk.PhotoImage(resized)
        panel = tk.Label(root, image=img)
        panel.image = img
        panel.place(x=950, y=500)
        tr_load_button = Button(root, text="Load Training Files",
                                command=lambda: self.reset(root),
                                font=("Great Vibes", 16, "bold"), width=200, height=30, bg='#7AC578', fg='#FFFFF
F',
                                borderless=True, activeforeground='#FFFFFF', activebackground='#AAD1A9',
                                focusthickness=0)
        tr_load_button.place(x=400, y=240)
        train_button = Button(root, text="Train", command=lambda: self.reset(root),
                              font=("Great Vibes", 16, "bold"), width=140, height=30, bg='#7AC578', fg='#FFFFFF',
                              borderless=True, activeforeground='#FFFFFF', activebackground='#AAD1A9',
                              focusthickness=0)
        train_button.place(x=800, y=285)
        te_load_button = Button(root, text="Test data",
                                command=lambda: self.reset(root),
                                font=("Great Vibes", 16, "bold"), width=200, height=30, bg='#7AC578', fg='#FFFFF
```

```
F',
                                borderless=True, activeforeground='#FFFFFF', activebackground='#AAD1A9',
                                focusthickness=0)
        te_load_button.place(x=400, y=620)
```

`load_testing` is the function that takes the `te_X` and `te_y` pathes, read contents of them and convert them to a format that `sklearn.MLPClassifier` can accept them then save them to `Controls/Controller` class call `Controller.test` method to test the samples, plot the `confusion matrix` and get the `accureccy`.

Image of the result of this method:



Reset function

```
def reset(self, root):
        root.destroy()
        MyApp()
```

`reset` function will kill the app then start it again.

Controls/Controller

Controller is the class that acts like a layer between the GUI and the functionalities of the program.

```
def __init__(self):
        self.train_x = None
        self.train_y = None
        self.test_x = None
```

```
        self.test_y = None
        self.model = None
```

`__init__` is the constructor or the method that initialize the variables that hold the datasets.

Open directory function

```
def open_directory(self):
        path = filedialog.askopenfilename()
        return path
```

`open_directory` is the functions that open the directory window then asks the user to select one file only to return its `path`.

Load training files function

```
def load_training_files(self, train_x_path, train_y_path):
        self.train_x = fr.read(train_x_path)
        self.train_y = fr.read(train_y_path)
        return
```

`load_training_files` is the functions that saves the data training data set and convert them to csv format by calling `Controls/FileReding.read()` then save them to use them later.

Load testing files function

```
def load_testing_files(self, test_x_path, test_y_path):
        self.test_x = fr.read(test_x_path)
        self.test_y = fr.read(test_y_path)
        return
```

`load_testing_files` is the functions that saves the data testing data set and convert them to csv format by calling `Controls/FileReding.read()` then save them to use them later.

Train function

```
def train(self, units, lr):
        time, model = nn.train(self.train_x, self.train_y, units, lr)
        self.model = model
        return time
```

`train` is the functions that calls `Neural_Networks/NN.train` to `create`, `train` and return the `model` and the training `time`.

Test function

```
def test(self):
        accuracy = nn.test(self.test_x, self.test_y, self.model)
        return accuracy
```

`test` is the functions that calls `Neural_Networks/NN.test` to test the training `acccurecy` , return it and to plot the `confusion matrix`.

Controls/FileReading

Read function

```
def read(path):
    data = open(path, 'rt')
    reader = csv.reader(data, delimiter=',', quoting=csv.QUOTE_NONE)
    lister = list(reader)
    data = np.array(lister, dtype=float)
    return data
```

`read` is the functions that convert the dataset from csv format to arrays.

Neural_Networs/NN

Train function

```
def train(train_x, train_y, units, lr):
    classifier = MLPClassifier(learning_rate_init=float(lr), hidden_layer_sizes=(int(units),), max_iter=200)
    start = time.time()
    model = classifier.fit(train_x, np.ravel(train_y))
    end = time.time()
    plt.plot(classifier.loss_curve_)
    print(classifier.get_params())
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.xticks()
    plt.savefig('assets/loss_vs_epoch.png')
    return f'{end - start:.2f}s', model
```

`train` is the function that will take the `train_x` , `train_y` datasets , neurons `units` and the `learning rate` and will create the model according to them. Then it will record the start time and start fitting then the end time. Finally it will plot the `loss vs epochs` graph and will return the training `time`.

Test function

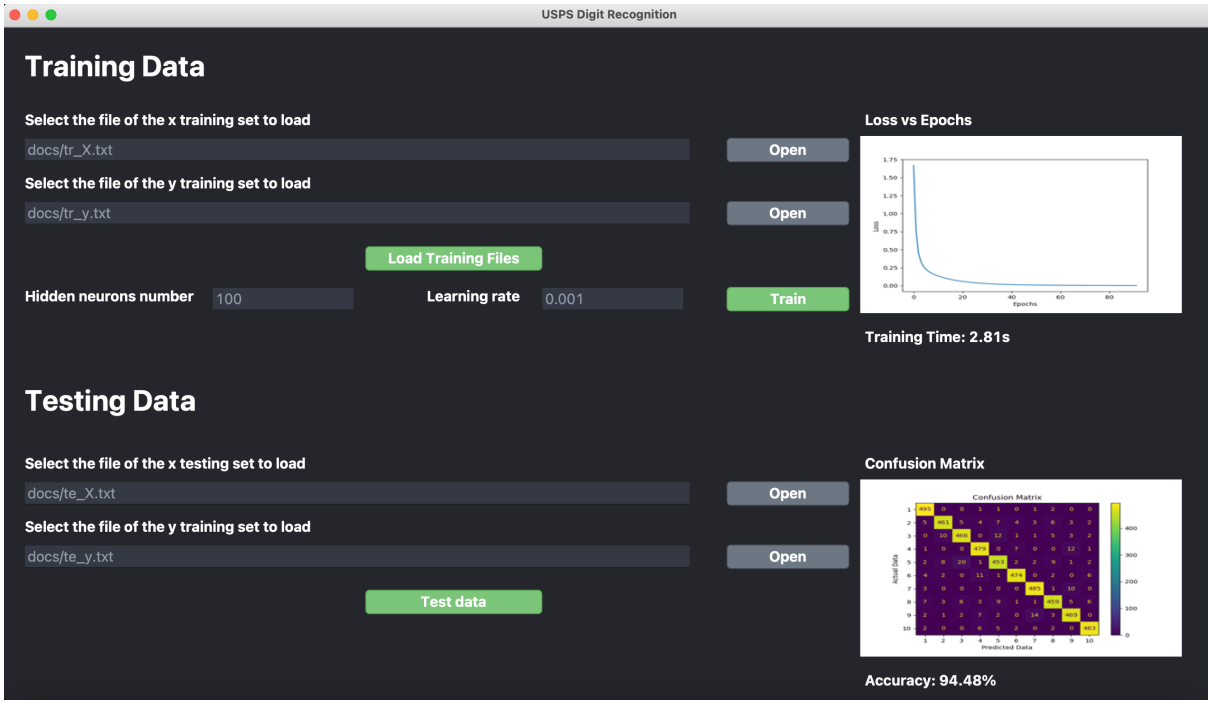
```
def test(test_x, test_y, model):
    classifier = model
    score = classifier.score(test_x, test_y)
    predicted_y = classifier.predict(test_x)
    cm = confusion_matrix(test_y, predicted_y)
    cmd = ConfusionMatrixDisplay(cm, display_labels=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    cmd.plot()
```

```
    cmd.ax_.set(
        title='Confusion Matrix',
        xlabel='Predicted Data',
        ylabel='Actual Data')
    plt.savefig('assets/Confusion_Matrix.png')
    plt.show()
    return f'{score * 100}%'
return f'{score * 100}%'
```

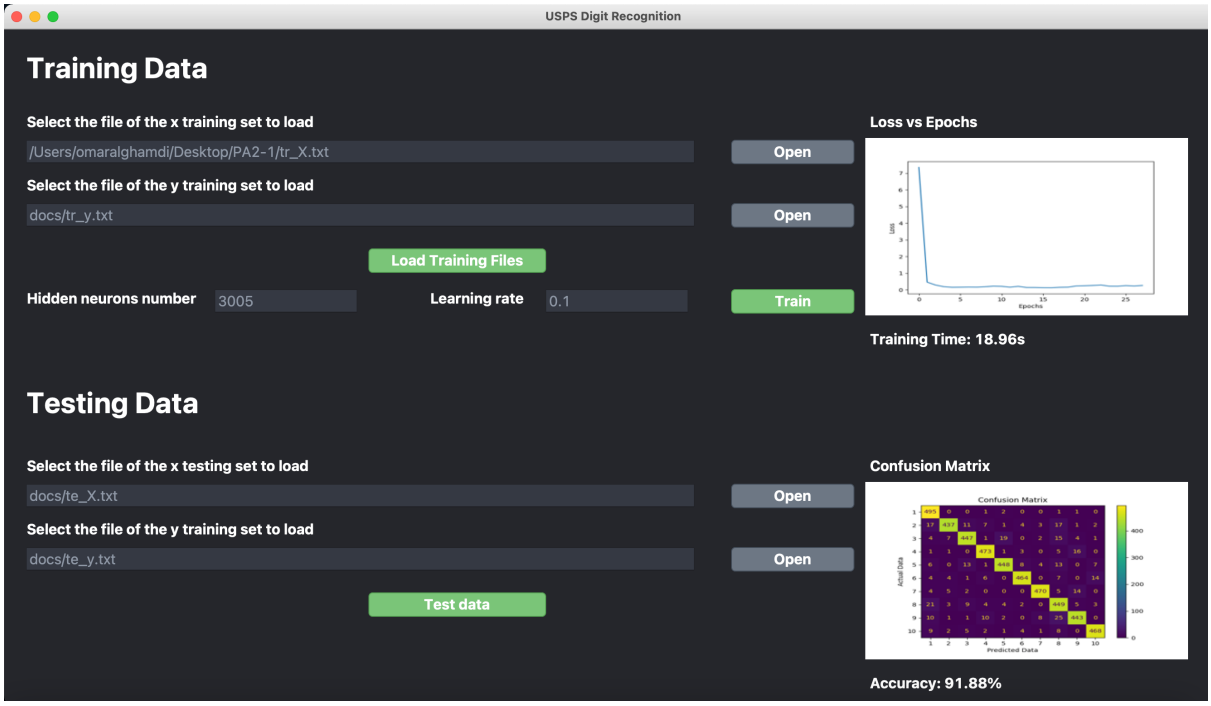
`test` is the function that will take the `test_x` , `test_y` datasets and `model` created in the `train` function then it will calculate the `accuracy` , plot the `confution matrix` and will return the `accuracy.`

Test Cases

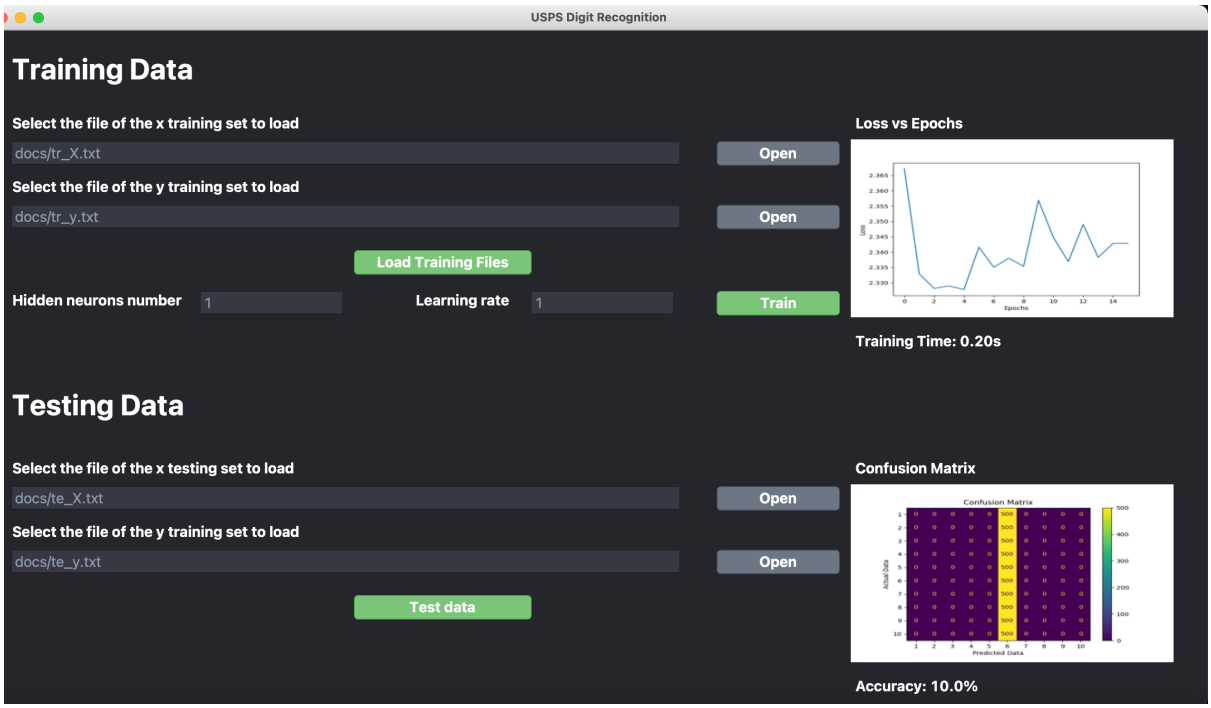
Test Case 1



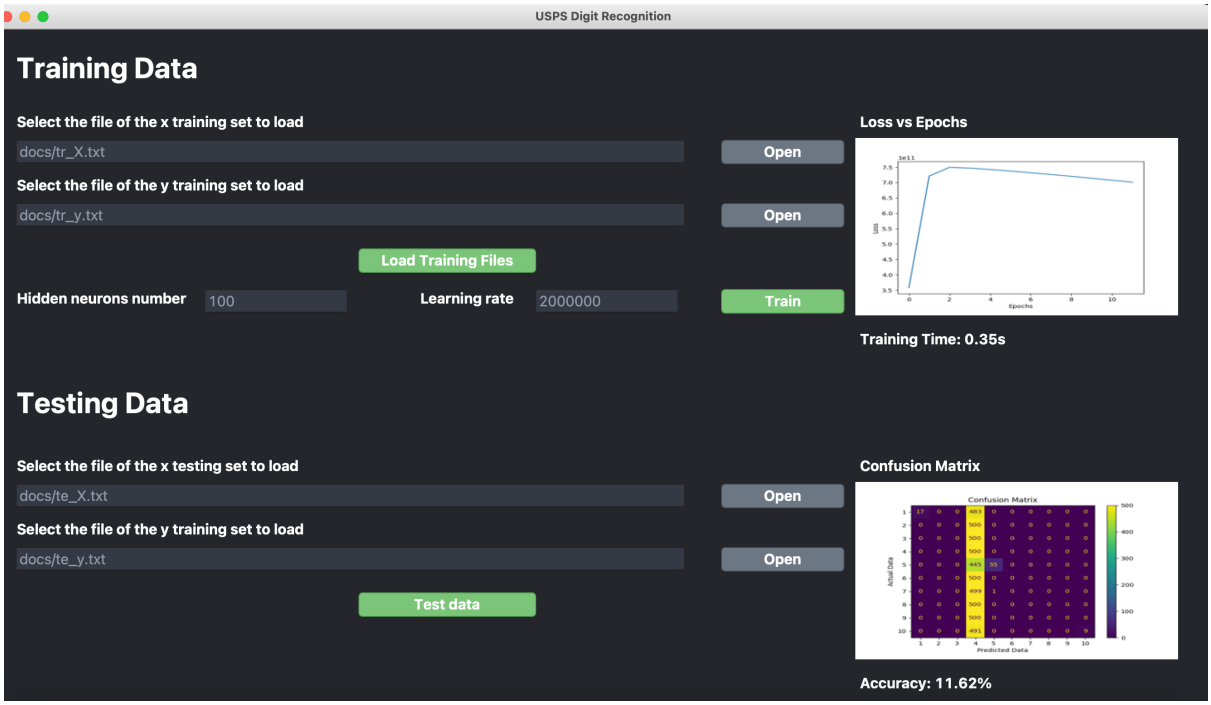
Test Case 2



Test Case 3



Test Case 4


USPS Digit Recognition
Training Data
Select the file of the x training set to load
docs/tr_X.txt
Open
Select the file of the y training set to load
docs/tr_y.txt
Open
Load Training Files
Hidden neurons number
100
Learning rate
2000000
Train
Loss vs Epochs
Training Time: 0.35s
Testing Data
Select the file of the x testing set to load
docs/te_X.txt
Open
Select the file of the y training set to load
docs/te_y.txt
Open
Test data
Confusion Matrix
Accuracy: 11.62%