Neural Networks

Ghazal Kalhor

***Abstract* — In this computer assignment, we want to classify given photos from clothes into ten classes using Neural Netwoks that we learnt in Artificial Intelligence. We will build our neural network from scratch using method of NumPy library.** ***Keywords* — Artificial Intelligence, NumPy, Neural Netwoks**

Introduction

The aim of this computer assignment is to implement a neural network from scratch and learn its hyper parameters to achieve the highest accuracy in our model.

Importing Libraries

In this part, some of the necessary libraries were imported in order to use their helpful functions.

In [2]:

```
import numpy as np
from PIL import Image
import matplotlib
import matplotlib.pyplot as plt
```

Importing Data

Contents of files *trainData.csv*, *trainLabels.csv*, *testData.csv*, and *testLabels.csv* were read using *pd.read_csv* and then stored in following dataframes.

In [3]:

```
train_data = np.loadtxt('trainData.csv', delimiter=",")
test_data = np.loadtxt('testData.csv', delimiter=",")
```

In [4]:

```
train_labels = np.loadtxt('trainLabels.csv', delimiter=",")
test_labels = np.loadtxt('testLabels.csv', delimiter=",")
```

Phase 1: Visualization and Preprocessing

Step #1

Plotting a Sample of each Class

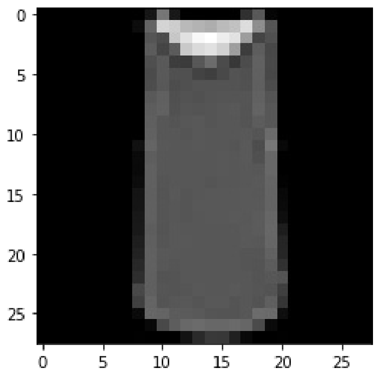
In [5]:

```
def plot_original_image(flatten_vect, label):
    print("Class No.", label)
    img = Image.fromarray(flatten_vect.reshape(28, 28))
    plt.imshow(img, interpolation = 'nearest')
    plt.show()
```

In [6]:

```
index = 2
plot_original_image(train_data[index], train_labels[index])
```

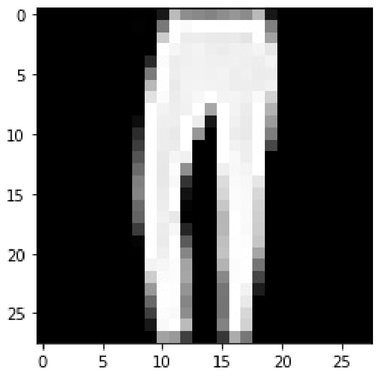
```
Class No. 0.0
```



In [7]:

```
index = 1399
plot_original_image(train_data[index], train_labels[index])
```

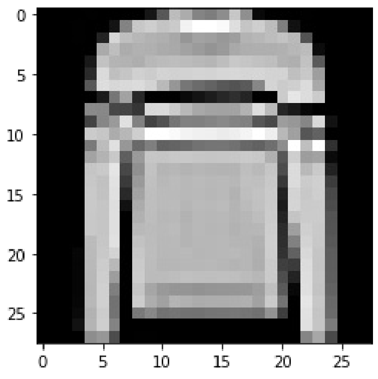
```
Class No. 1.0
```



In [8]:

```
index = 5
plot_original_image(train_data[index], train_labels[index])
```

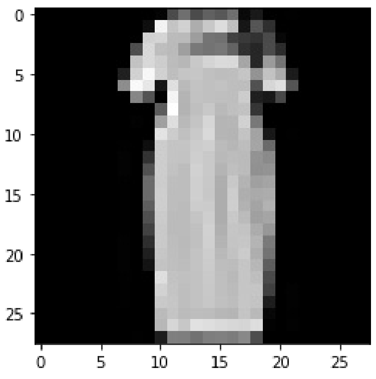
```
Class No. 2.0
```



In [9]:

```
index = 1378
plot_original_image(train_data[index], train_labels[index])
```

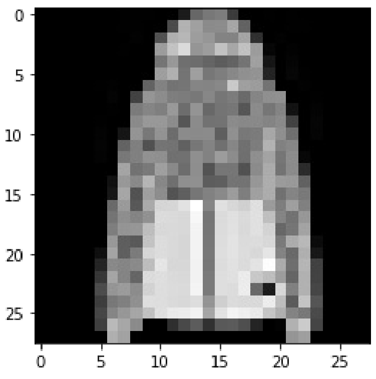
```
Class No. 3.0
```



In [10]:

```
index = 22
plot_original_image(train_data[index], train_labels[index])
```

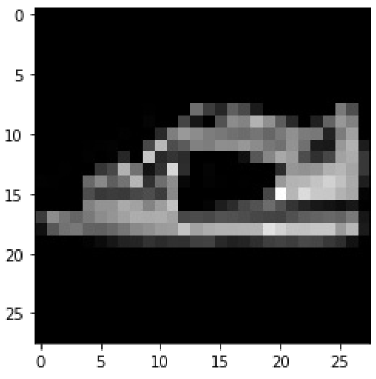
```
Class No. 4.0
```



In [11]:

```
index = 13
plot_original_image(train_data[index], train_labels[index])
```

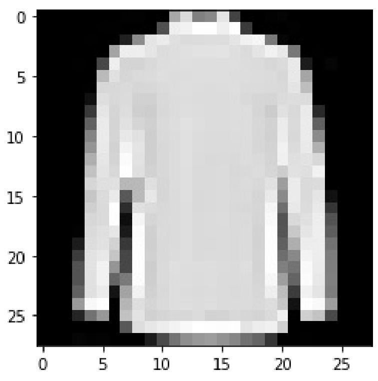
```
Class No. 5.0
```



In [12]:

```
index = 2021
plot_original_image(train_data[index], train_labels[index])
```

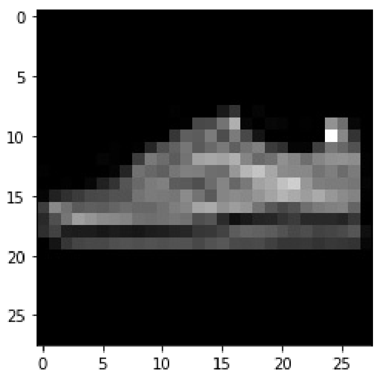
```
Class No. 6.0
```



In [13]:

```
index = 666
plot_original_image(train_data[index], train_labels[index])
```

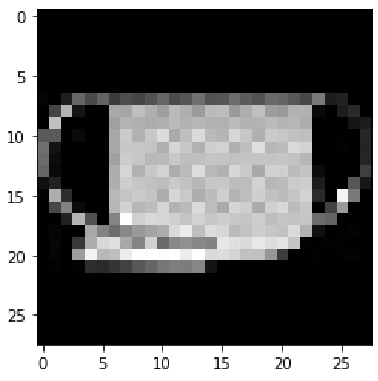
```
Class No. 7.0
```



In [14]:

```
index = 1999
plot_original_image(train_data[index], train_labels[index])
```

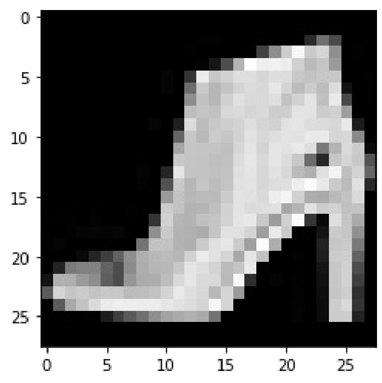
```
Class No. 8.0
```



In [15]:

```
index = 1341
plot_original_image(train_data[index], train_labels[index])
```

```
Class No. 9.0
```

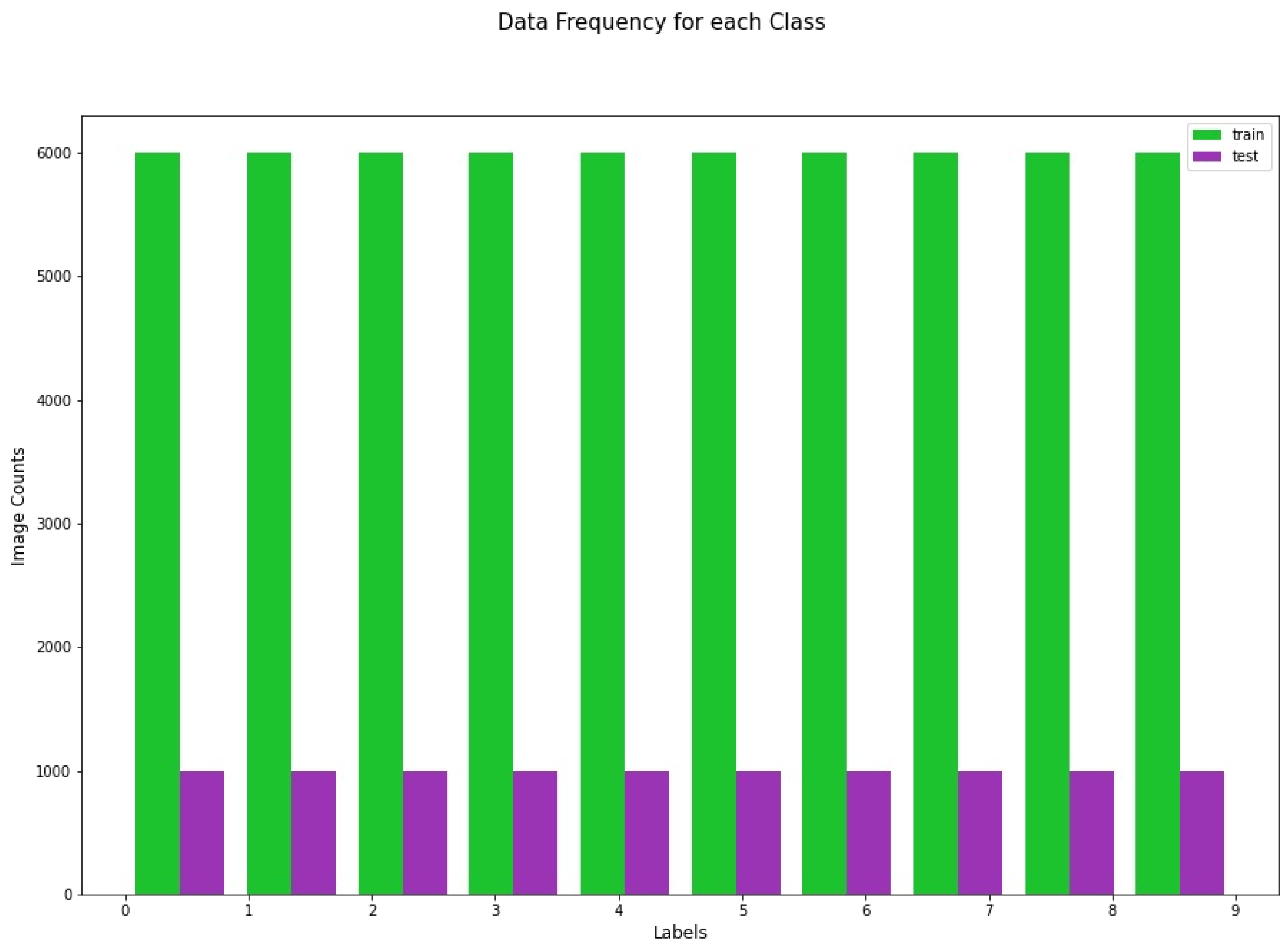


Step #2

Plotting Label Distributin in Tran and Test

In [92]:

```
fig = plt.figure(figsize=(15, 10))
labels = (train_labels, test_labels)
plt.hist(labels, label=['train', 'test'], color=['#1cc32f', '#9a34b5'])
plt.xlabel('Labels', fontsize = 12)
plt.ylabel('Image Counts', fontsize = 12)
plt.suptitle('Data Frequency for each Class', fontsize = 15)
plt.legend()
plt.xticks((0, 1, 2, 3, 4, 5, 6, 7, 8, 9))
plt.show()
```



Step #3

Normalizing Image Vectors

Normalization refers to rescaling real valued numeric attributes into the range 0 and 1. Neural networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process. As such it is good practice to normalize the pixel values so that each pixel value has a value between 0 and 1.

In [17]:

```
train_data /= 255
test_data /= 255
```

Phase 2: Completing the Given Code for Neural Network

Dataloader

In this part, we completed *onehot* and *shuffle_dataset* methods of Dataloader class. This class is used for preparing the train and test data for the learning process of our neural network.

onehot method is used to perform one-hot encoding on class labels.

shuffle_dataset method is used to shuffle data and labels simultaneously. It is very important that dataset is shuffled well to avoid any element of bias/patternsin the split datasetsbefore training the model.

In [18]:

```
class Dataloader:

    def __init__(self, data, labels, n_classes, batch_size=None, shuffle=False):

        assert len(data)==len(labels)
        self.__n_classes = n_classes
        self.__batch_size = batch_size
        self.__shuffle = shuffle
        self.__data = data
        self.__onehot_labels = self.__onehot(labels, self.__n_classes)

    def __onehot(self, labels, n_classes):
        # TODO: Implement: Done:)=
        labels = labels.astype(int)
        onehot_vectors = np.zeros((labels.size, n_classes))
        onehot_vectors[np.arange(labels.size), labels] = 1.0
        return onehot_vectors

    def __shuffle_dataset(self):
        # TODO: Implement Done:)=
        total_data = np.c_[self.__data.reshape(len(self.__data), -1), \
                           self.__onehot_labels.reshape(len(self.__onehot_labels), -1)]
        np.random.shuffle(total_data)
        data = total_data[:, :self.__data.size//len(self.__data)].reshape(self.__data.shape)
        self.__onehot_labels = total_data[:, self.__data.size//\
                                          len(self.__data):].reshape(self.__onehot_labels.shape)
        self.__data = data

    def __iter__(self):

        if self.__shuffle:
            self.__shuffle_dataset()

        if self.__batch_size==None:
            yield (np.matrix(self.__data), np.matrix(self.__onehot_labels))
            return

        for idx in range(0, len(self.__data), self.__batch_size):
            yield (np.matrix(self.__data[idx:idx+self.__batch_size]),
                   np.matrix(self.__onehot_labels[idx:idx+self.__batch_size]))

    def get_one_hot(self):
        return np.argmax(self.__onehot_labels, axis = 1)
```

Activation Functions

In this part, we completed *val* and *derivative* methods of activation function classes. Also. we added a class for **Tanh** activation function. We used the following formulas to implement these activation functions.

$$Relu(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \quad \text{and} \quad Relu'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

$$LeakyRelu(x) = \begin{cases} x & x > 0 \\ 0.01x & x \leq 0 \end{cases} \quad \text{and} \quad LeakyRelu'(x) = \begin{cases} 1 & x > 0 \\ 0.01 & x \leq 0 \end{cases}$$

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \quad \text{and} \quad Sigmoid'(x) = Sigmoid(x)(1 - Sigmoid(x))$$

$$Tanh'(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \text{and} \quad Tanh'(x) = 1 - Tanh(x)^2$$

In [96]:

```python
class Identical:

    def __init__(self): pass

    def __val(self, matrix):
        identical_value = np.matrix(matrix, dtype=float)
        return identical_value

    def derivative(self, matrix):
        temp = np.matrix(matrix, dtype=float)
        identical_derivative = np.matrix(np.full(np.shape(temp), 1.))
        return identical_derivative

    def __call__(self, matrix):
        return self.__val(matrix)


class Relu:

    def __init__(self): pass

    def __relu(self, matrix):
        # TODO: Implement: Done:)=
        temp = np.matrix(matrix, dtype=float)
        relu_value = np.matrix(np.maximum(temp, 0.))
        return relu_value

    def derivative(self, matrix):
        # TODO: Implement: Done:)=
        relu_derivative = np.matrix(matrix, dtype=float)
        relu_derivative[relu_derivative <= 0] = 0.
        relu_derivative[relu_derivative > 0] = 1.
        return relu_derivative

    def __call__(self, matrix):
        return self.__relu(matrix)


class LeakyRelu:

    def __init__(self, negative_slope=0.01):
        self.negative_slope = 0.01

    def __val(self, matrix):
        # TODO: Implement: Done:)=
        temp = np.matrix(matrix, dtype=float)
        leacky_relu_value = np.matrix(np.where(temp > 0, temp, temp * self.negative_slope))
        return leacky_relu_value

    def derivative(self, matrix):
        # TODO: Implement: Done:)=
        temp = np.matrix(matrix, dtype=float)
        leacky_relu_derivative = np.ones_like(temp)
        leacky_relu_derivative[temp <= 0] = self.negative_slope
        return leacky_relu_derivative
    def __call__(self, matrix):
        return self.__val(matrix)


class Sigmoid:

    def __init__(self): pass

    def __val(self, matrix):
        # TODO: Implement: Done:)=
        temp = np.matrix(matrix, dtype=float)
        sigmoid_value = np.matrix(1.0/(1.0 + np.exp(-temp)))
        return sigmoid_value

    def derivative(self, matrix):
        # TODO: Implement: Done:)=
        sigmoid_derivative = np.multiply(self.__val(matrix), (1. - self.__val(matrix)))
        return sigmoid_derivative

    def __call__(self, matrix):
        return self.__val(matrix)


class Tanh:

    def __init__(self): pass

    def __val(self, matrix):
        # TODO: Implement: Done:)=
        temp = np.matrix(matrix, dtype=float)
        tanh_value = np.matrix((np.exp(temp) - np.exp(-temp))/(np.exp(temp) + np.exp(-temp)))
        return tanh_value

    def derivative(self, matrix):
        # TODO: Implement: Done:)=
        tanh_derivative = 1 - np.power(self.__val(matrix), 2)
        return tanh_derivative

    def __call__(self, matrix):
        return self.__val(matrix)


class Softmax:

    def __init__(self): pass

    def __val(self, matrix):
        # TODO: Implement: Done:)=
        temp = np.matrix(matrix, dtype=float)
        softmax_value = np.exp(temp - np.max(temp))
        softmax_value /= softmax_value.sum(axis=1)
        return softmax_value

    def __call__(self, matrix):
        return self.__val(matrix)
```

Loss Function

In this part, we completed *val* and *derivative* methods of CrossEntropy classes. It is assumed that Sofmax is applied before using the following formulas.

$$CrossEntropy(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$$

$$CrossEntropy'(y, \hat{y}) = y - \hat{y}$$

In [97]:

```
class CrossEntropy: #(with softmax)

    def __init__(self): pass

    def __val(self, true_val, expected_val):
        assert np.shape(true_val)==np.shape(expected_val)
        # TODO: Implement: Done:)=
        softmax = Softmax()
        cross_entropy_value = np.sum(-np.multiply(np.matrix(expected_val, dtype=float), \
                                                  np.log(softmax(true_val))), axis=1)
        return cross_entropy_value

    def derivative(self, true_val, expected_val):
        assert np.shape(true_val)==np.shape(expected_val)
        # TODO: Implement: Done:)=
        softmax = Softmax()
        cross_entropy_derivative = softmax(true_val) - expected_val
        return cross_entropy_derivative

    def __call__(self, true_val, expected_val):
        return self.__val(true_val, expected_val)
```

Layer

In this part, we completed *forward*, *update_weights*, *uniform_weight*, and *normal_weight* methods of Layer class. Also we added *he_weight* that implements **he initialization** method for **Relu** activation function to avoid vanishing and exploiding in performing gradient descent. We used the following formulas to update weights at the end of each batch.

$$\frac{\partial L}{\partial W} = x^T \frac{\partial L}{\partial y}$$

$$\frac{\partial L}{\partial b} = 1 \frac{\partial L}{\partial y}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} W^T$$

In [98]:

```
class Layer:

    DEFAULT_LOW, DEFAULT_HIGH, DEFAULT_MEAN, DEFAULT_VAR = 0, 0.05, 0., 1.

    def __init__(self, input_size, output_size,
                 activation=Identical(), initial_weight='uniform', **initializing_parameters):

        self.__weight_initializer_dict = {'uniform':self.__uniform_weight, 'normal':self.__normal_weight,
                                          'he':self.__he_weight}

        assert type(initial_weight)==str, 'Undefined activation function!'
        assert initial_weight in self.__weight_initializer_dict, 'Undefined weight initialization function!'

        self.__n_neurons = output_size
        self.__fan_in = input_size
        self.__fan_out = output_size
        weight_initializer = self.__weight_initializer_dict[initial_weight]
        self.__weight = weight_initializer(input_size, self.__n_neurons, **initializing_parameters)
        self.__bias = weight_initializer(1, self.__n_neurons, **initializing_parameters)
        self.__activation = activation

        self.__last_input = None
        self.__last_activation_input = None
        self.__last_activation_output = None
        self.__last_activation_derivative = None

    def forward(self, layer_input):
        assert np.ndim(layer_input)==2
        assert np.size(self.__weight,0) == np.size(layer_input,1)
        # TODO: Implement: Done:)=
        self.__last_input = layer_input
        self.__last_activation_input = np.add(np.dot(self.__last_input, self.__weight), self.__bias)
        self.__last_activation_output = self.__activation(self.__last_activation_input)
        self.__last_activation_derivative = self.__activation.derivative(self.__last_activation_output)

        return self.__last_activation_output

    def update_weights(self, backprop_tensor, lr):
        assert np.ndim(backprop_tensor)==2
        assert np.size(backprop_tensor,0) == np.size(self.__last_activation_derivative,0)
        assert np.size(backprop_tensor,1) == self.__n_neurons
        # TODO: Implement: Done:)=
        backprop_tensor = np.multiply(backprop_tensor, self.__last_activation_derivative)
        self.__weight -= (lr * np.dot(self.__last_input.T, backprop_tensor))
        self.__bias -= (lr * np.dot(np.ones((1, backprop_tensor.shape[0])), \
                                                              backprop_tensor))
        backprop_tensor = np.dot(backprop_tensor, self.__weight.T)
        return backprop_tensor

    def __uniform_weight(self, dim1, dim2, **initializing_parameters):
        low, high = self.DEFAULT_LOW, self.DEFAULT_HIGH
        if 'low' in initializing_parameters.keys(): low = initializing_parameters['low']
        if 'high' in initializing_parameters.keys(): high = initializing_parameters['high']
        # TODO: Implement: Done:)=
        weights = np.random.uniform(low, high=high, size=(dim1, dim2))
        return weights

    def __he_weight(self, dim1, dim2, **initializing_parameters):
        r = np.sqrt(12./(self.__fan_in, self.__fan_out))
        low, high = -r, r
        if 'low' in initializing_parameters.keys(): low = initializing_parameters['low']
        if 'high' in initializing_parameters.keys(): high = initializing_parameters['high']
        # TODO: Implement: Done:)=
        weights = np.random.uniform(low, high=high, size=(dim1, dim2))
        return weights

    def __normal_weight(self, dim1, dim2, **initializing_parameters):
        mean, var = self.DEFAULT_MEAN, self.DEFAULT_VAR
        if 'mean' in initializing_parameters.keys(): mean = initializing_parameters['mean']
        if 'var' in initializing_parameters.keys(): var = initializing_parameters['var']
        # TODO: Implement: Done:)=
        weights = np.random.normal(loc=mean, scale=var, size=(dim1, dim2))
        return weights

    @property
    def n_neurons(self): return self.__n_neurons

    @property
    def weight(self): return self.__weight

    @property
    def bias(self): return self.__bias

    @property
    def activation(self): return self.__activation
```

Feed Forward Neural Network

In this part, we completed the methgod of FeedForwardNN class.

In [99]:

```
class FeedForwardNN:

    def __init__(self, input_shape):

        self.__input_shape = input_shape
        self.__output_shape = None

        self.__layers_list = []

        self.__lr = None
        self.__loss = None

        self.__before_last = None
        self.__before_last_train = None
        self.__before_last_test = None

        self.__train_labels = None
        self.__test_labels = None

        self.__colors = ['mediumvioletred', 'darkviolet', 'gold', 'turquoise', 'blue', 'springgreen', \
                         'magenta', 'tan', 'chocolate', 'darkgreen']
        self.__classes = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
    def add_layer(self, n_neurons, activation=Relu(), initial_weight='uniform', **initializing_parameters):

        assert type(n_neurons)==int, "Invalid number of neurons for the layer!"
        assert n_neurons>0, "Invalid number of neurons for the layer!"

        n_prev_neurons = self.__input_shape if len(self.__layers_list)==0 else \
        self.__layers_list[-1].n_neurons
        new_layer = Layer(n_prev_neurons, n_neurons, activation, initial_weight, \
                          **initializing_parameters)
        self.__layers_list.append(new_layer)
        self.__output_shape = self.__layers_list[-1].n_neurons


    def set_training_param(self, loss=CrossEntropy(), lr=1e-3):
        assert self.__layers_list, "Uncomplete model!"
        self.__loss = loss
        self.__lr = lr


    def forward(self, network_input):
        assert type(self.__output_shape) != None, "Model is not compiled!"
        # TODO: Implement: Done:)=
        network_output = network_input
        for i in range(len(self.__layers_list)):
            network_output = self.__layers_list[i].forward(network_output)
            if i == (len(self.__layers_list)-2):
                self.__before_last = network_output
        return network_output


    def fit(self, epochs, trainloader, testloader=None, print_results=True):

        assert type(self.__output_shape) != None, "Model is not compiled!"
        assert type(self.__lr) != None and type(self.__loss) != None, "Training paramenters are not set!"

        log = {"train_accuracy":[], "train_loss":[], "test_accuracy":[], "test_loss":[]}

        for epoch in range(1, epochs+1):

            self.__before_last_train = None
            self.__before_last_test = None

            if print_results:
                print('Epoch {}:'.format(epoch))

            average_accuracy, average_loss = self.__train(trainloader)
            log['train_accuracy'].append(average_accuracy)
            log['train_loss'].append(average_loss)
            if print_results:
                print('\tTrain: Average Accuracy: {}\tAverage Loss: {}'.format(average_accuracy,\
                                                                    average_loss))

            if type(testloader) != type(None):
                average_accuracy, average_loss = self.__test(testloader)
                log['test_accuracy'].append(average_accuracy)
                log['test_loss'].append(average_loss)
                if print_results:
                    print('\tTest: Average Accuracy: {}\tAverage Loss: {}'.format(average_accuracy,\
                                                                    average_loss))

        return log

    def early_stopping(self, trainloader, testloader, print_results=True):

        assert type(self.__output_shape) != None, "Model is not compiled!"
        assert type(self.__lr) != None and type(self.__loss) != None, "Training paramenters are not set!"

        log = {"train_accuracy":[], "train_loss":[], "test_accuracy":[], "test_loss":[]}

        epoch = 1

        while True:

            self.__before_last_train = None
            self.__before_last_test = None

            if print_results:
                print('Epoch {}:'.format(epoch))

            train_accuracy, train_loss = self.__train(trainloader)
            log['train_accuracy'].append(train_accuracy)
            log['train_loss'].append(train_loss)

            if print_results:
                print('\tTrain: Average Accuracy: {}\tAverage Loss: {}'.format(train_accuracy,\
                                                                    train_loss))

            test_accuracy, test_loss = self.__test(testloader)
            log['test_accuracy'].append(test_accuracy)
            log['test_loss'].append(test_loss)
            if print_results:
                print('\tTest: Average Accuracy: {}\tAverage Loss: {}'.format(test_accuracy,\
                                                                     test_loss))
            if epoch > 1 and test_accuracy < log['test_accuracy'][len(log['test_accuracy'])-2]:
                break

            epoch += 1

        return log


    def __train(self, trainloader):
        bach_accuracies, batch_losses = [], []
        for x_train, y_train in trainloader:
            batch_accuracy, batch_loss = self.__train_on_batch(x_train, y_train)
            bach_accuracies.append(batch_accuracy)
            batch_losses.append(batch_loss)
        self.__train_labels = trainloader.get_one_hot()
        return np.mean(bach_accuracies), np.mean(batch_losses)


    def __test(self, testloader):
        bach_accuracies, batch_losses = [], []
        for x_test, y_test in testloader:
            batch_accuracy, batch_loss = self.__test_on_batch(x_test, y_test)
            bach_accuracies.append(batch_accuracy)
            batch_losses.append(batch_loss)
        self.__test_labels = testloader.get_one_hot()
        return np.mean(bach_accuracies), np.mean(batch_losses)


    def __train_on_batch(self, x_batch, y_batch):
        # TODO: Implement: Done:)=
        output = self.forward(x_batch)
        self.__update_weights(output, y_batch)
        batch_accuracy = self.__compute_accuracy(output, y_batch)
        batch_average_loss = np.mean(self.__loss(output, y_batch))
        if type(self.__before_last_train) is not np.matrix:
            self.__before_last_train = self.__before_last
        else:
            self.__before_last_train = np.concatenate((self.__before_last_train, \
                                                       self.__before_last), axis=0)
        return (batch_accuracy, batch_average_loss)


    def __test_on_batch(self, x_batch, y_batch):
        # TODO: Implement: Done:)=
        output = self.forward(x_batch)
        batch_accuracy = self.__compute_accuracy(output, y_batch)
        batch_average_loss = np.mean(self.__loss(output, y_batch))
        if type(self.__before_last_test) is not np.matrix:
            self.__before_last_test = self.__before_last
        else:
            self.__before_last_test = np.concatenate((self.__before_last_test, \
                                                      self.__before_last), axis=0)
        return (batch_accuracy, batch_average_loss)


    def __get_labels(self, outputs):
        # TODO: Implement: Done:)=
        labels = np.argmax(outputs, axis = 1)
        return labels


    def __compute_accuracy(self, output, expected_output):
        # TODO: Implement: Done:)=
        accuracy = (self.__get_labels(output) == self.__get_labels(expected_output)).sum()/output.shape[0]
        accuracy *= 100
        return accuracy


    def __update_weights(self, output, y_train):
        # TODO: Implement: Done:)=
        for i in reversed(range(len(self.__layers_list))):
            if i == (len(self.__layers_list) - 1):
                backprop_tensor = self.__layers_list[i].\

                update_weights(self.__loss.derivative(output, y_train), self.__lr)
            else:
                backprop_tensor = self.__layers_list[i].update_weights(backprop_tensor, self.__lr)
        return


    def plot_reduced_dimension_trian(self):
        features = np.array(self.__before_last_train)
        labels = self.__train_labels
        x = features[:,0]
        y = features[:,1]
        fig = plt.figure(figsize=(15, 10))
        plt.scatter(x, y, c=labels, cmap=matplotlib.colors.ListedColormap(self.__colors))
        plt.title('Train Reduced Dimension', fontsize = 15)
        plt.ylabel('2nd Dimension', fontsize = 12)
        plt.xlabel('1st Dimension', fontsize = 12)

        cb = plt.colorbar()
        loc = np.arange(0, max(labels), max(labels)/float(len(self.__colors)))
        cb.set_ticks(loc)
        cb.set_ticklabels(self.__classes)
        plt.show()

    def plot_reduced_dimension_test(self):
        features = np.array(self.__before_last_test)
        labels = self.__test_labels
        x = features[:,0]
        y = features[:,1]
        fig = plt.figure(figsize=(15, 10))
        plt.scatter(x, y, c=labels, cmap=matplotlib.colors.ListedColormap(self.__colors))
        plt.title('Test Reduced Dimension', fontsize = 15)
        plt.ylabel('2nd Dimension', fontsize = 12)
        plt.xlabel('1st Dimension', fontsize = 12)

        cb = plt.colorbar()
        loc = np.arange(0, max(labels), max(labels)/float(len(self.__colors)))
        cb.set_ticks(loc)
        cb.set_ticklabels(self.__classes)
        plt.show()
```

Phase 3: Classifying Data

Step #1

In [100]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
EPOCHS = 30
BATCH_SIZE = 32
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=Relu(), weight_initializer='he')
network.add_layer(10, activation=Relu(), weight_initializer='he')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 35.26833333333333      Average Loss: 1.7270650013465823
        Test: Average Accuracy: 57.897364217252395      Average Loss: 1.1760277681872764
Epoch 2:
        Train: Average Accuracy: 63.635 Average Loss: 0.9729168544947738
        Test: Average Accuracy: 66.38378594249201       Average Loss: 0.8684203135665756
Epoch 3:
        Train: Average Accuracy: 70.415 Average Loss: 0.7931485822329905
        Test: Average Accuracy: 72.17452076677316       Average Loss: 0.7583906425862659
Epoch 4:
        Train: Average Accuracy: 74.17666666666666      Average Loss: 0.7110941353626407
        Test: Average Accuracy: 75.3694089456869        Average Loss: 0.6944423287536822
Epoch 5:
        Train: Average Accuracy: 76.66666666666667      Average Loss: 0.6552827166475541
        Test: Average Accuracy: 77.09664536741214       Average Loss: 0.651829023188389
Epoch 6:
        Train: Average Accuracy: 78.39666666666666      Average Loss: 0.614323117859118
        Test: Average Accuracy: 78.04512779552715       Average Loss: 0.6224772724808407
Epoch 7:
        Train: Average Accuracy: 79.58  Average Loss: 0.5836763894875798
        Test: Average Accuracy: 78.4844249201278        Average Loss: 0.6016709443558008
Epoch 8:
        Train: Average Accuracy: 80.47  Average Loss: 0.5606124229365076
        Test: Average Accuracy: 79.06349840255591       Average Loss: 0.5861143277343346
Epoch 9:
        Train: Average Accuracy: 81.09  Average Loss: 0.5430543318595225
        Test: Average Accuracy: 79.56269968051119       Average Loss: 0.5738431778185031
Epoch 10:
        Train: Average Accuracy: 81.53333333333333      Average Loss: 0.5293739153834736
        Test: Average Accuracy: 79.99201277955271       Average Loss: 0.5639477932111422
Epoch 11:
        Train: Average Accuracy: 81.88666666666667      Average Loss: 0.518420601244861
        Test: Average Accuracy: 80.39137380191693       Average Loss: 0.5560464917730891
Epoch 12:
        Train: Average Accuracy: 82.17666666666666      Average Loss: 0.5094430481375236
        Test: Average Accuracy: 80.61102236421725       Average Loss: 0.5498744641178958
Epoch 13:
        Train: Average Accuracy: 82.42833333333333      Average Loss: 0.5019502692242261
        Test: Average Accuracy: 80.77076677316293       Average Loss: 0.5451738596629426
Epoch 14:
        Train: Average Accuracy: 82.60833333333333      Average Loss: 0.4956132657299996
        Test: Average Accuracy: 80.870607028754 Average Loss: 0.5416959042693508
Epoch 15:
        Train: Average Accuracy: 82.80833333333334      Average Loss: 0.49020468384741483
        Test: Average Accuracy: 80.90055910543131       Average Loss: 0.5392313200940752
Epoch 16:
        Train: Average Accuracy: 82.98666666666666      Average Loss: 0.4855641408938002
        Test: Average Accuracy: 80.91054313099042       Average Loss: 0.5376173220754847
Epoch 17:
        Train: Average Accuracy: 83.175 Average Loss: 0.48157469699497635
        Test: Average Accuracy: 80.99041533546325       Average Loss: 0.5367336034528385
Epoch 18:
        Train: Average Accuracy: 83.255 Average Loss: 0.4781503676023217
        Test: Average Accuracy: 80.83067092651757       Average Loss: 0.5365002302859008
Epoch 19:
        Train: Average Accuracy: 83.36333333333333      Average Loss: 0.47522379825665306
        Test: Average Accuracy: 80.90055910543131       Average Loss: 0.5368548642533802
Epoch 20:
        Train: Average Accuracy: 83.50333333333333      Average Loss: 0.47274460885864744
        Test: Average Accuracy: 80.870607028754 Average Loss: 0.5377582925206683
Epoch 21:
        Train: Average Accuracy: 83.59666666666666      Average Loss: 0.4706774307365159
        Test: Average Accuracy: 80.8905750798722        Average Loss: 0.5391944135797546
Epoch 22:
        Train: Average Accuracy: 83.71333333333334      Average Loss: 0.4689841505849608
        Test: Average Accuracy: 80.78075079872204       Average Loss: 0.5411300915828654
Epoch 23:
        Train: Average Accuracy: 83.80833333333334      Average Loss: 0.4676448055701705
        Test: Average Accuracy: 80.60103833865814       Average Loss: 0.5435603437089908
Epoch 24:
        Train: Average Accuracy: 83.83333333333333      Average Loss: 0.4666424860707082
        Test: Average Accuracy: 80.5111821086262        Average Loss: 0.5464777777882742
Epoch 25:
        Train: Average Accuracy: 83.865 Average Loss: 0.46596582441395284
        Test: Average Accuracy: 80.43130990415335       Average Loss: 0.5498840275943331
Epoch 26:
        Train: Average Accuracy: 83.9   Average Loss: 0.46560883942863557
        Test: Average Accuracy: 80.37140575079871       Average Loss: 0.5537839144789992
Epoch 27:
        Train: Average Accuracy: 83.91166666666666      Average Loss: 0.4655708182268282
        Test: Average Accuracy: 80.22164536741214       Average Loss: 0.5581871283028007
Epoch 28:
        Train: Average Accuracy: 83.92666666666666      Average Loss: 0.465856314608251
        Test: Average Accuracy: 80.10183706070288       Average Loss: 0.5631079774223084
Epoch 29:
        Train: Average Accuracy: 83.96666666666667      Average Loss: 0.46647592646819297
        Test: Average Accuracy: 79.99201277955271       Average Loss: 0.568571058965018
Epoch 30:
        Train: Average Accuracy: 83.98833333333333      Average Loss: 0.46744462650968316
        Test: Average Accuracy: 79.85223642172524       Average Loss: 0.5745883419767477
```

Step #2

Learning Rate X 0.1

In this part, when we multiplied the optimum learning rate by 0.1, the speed of the learning process decreased and it seems that we need more epochs to converge to the highest accuracy.

In [101]:

```
INPUT_SHAPE = 784
LEARNING_RATE = (1e-4) * 0.1
EPOCHS = 30
BATCH_SIZE = 32
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=Relu(), weight_initializer='he')
network.add_layer(10, activation=Relu(), weight_initializer='he')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 10.611666666666666     Average Loss: 2.260236689910968
        Test: Average Accuracy: 11.142172523961662      Average Loss: 2.246001333834595
Epoch 2:
        Train: Average Accuracy: 12.546666666666667     Average Loss: 2.2264843453038585
        Test: Average Accuracy: 12.290335463258787      Average Loss: 2.2019746619744374
Epoch 3:
        Train: Average Accuracy: 18.915 Average Loss: 2.157437190466935
        Test: Average Accuracy: 22.234424920127797      Average Loss: 2.1007224140855145
Epoch 4:
        Train: Average Accuracy: 31.895 Average Loss: 2.0120976747227917
        Test: Average Accuracy: 44.18929712460064       Average Loss: 1.9139581235406313
Epoch 5:
        Train: Average Accuracy: 42.85333333333333      Average Loss: 1.8048451691218588
        Test: Average Accuracy: 49.371006389776355      Average Loss: 1.704333582608462
Epoch 6:
        Train: Average Accuracy: 53.3   Average Loss: 1.613049476760501
        Test: Average Accuracy: 53.66413738019169       Average Loss: 1.535863903339014
Epoch 7:
        Train: Average Accuracy: 57.14  Average Loss: 1.4646242012680781
        Test: Average Accuracy: 56.30990415335463       Average Loss: 1.407027844219716
Epoch 8:
        Train: Average Accuracy: 58.906666666666666     Average Loss: 1.348988089493812
        Test: Average Accuracy: 58.516373801916934      Average Loss: 1.3047139036886626
Epoch 9:
        Train: Average Accuracy: 60.321666666666665     Average Loss: 1.2553714879226818
        Test: Average Accuracy: 60.14376996805112       Average Loss: 1.2206732723717646
Epoch 10:
        Train: Average Accuracy: 61.605 Average Loss: 1.1776179396719997
        Test: Average Accuracy: 61.88099041533546       Average Loss: 1.1504933805927313
Epoch 11:
        Train: Average Accuracy: 62.74666666666667      Average Loss: 1.1124055251570466
        Test: Average Accuracy: 62.819488817891376      Average Loss: 1.091607484714988
Epoch 12:
        Train: Average Accuracy: 63.87166666666667      Average Loss: 1.0575190966640042
        Test: Average Accuracy: 63.57827476038339       Average Loss: 1.042050175647081
Epoch 13:
        Train: Average Accuracy: 64.76333333333334      Average Loss: 1.0111524365178783
        Test: Average Accuracy: 64.5167731629393        Average Loss: 1.0001659642883656
Epoch 14:
        Train: Average Accuracy: 65.60333333333334      Average Loss: 0.9718016741205058
        Test: Average Accuracy: 65.02595846645367       Average Loss: 0.9645823903942848
Epoch 15:
        Train: Average Accuracy: 66.30333333333333      Average Loss: 0.9382110165187859
        Test: Average Accuracy: 65.69488817891374       Average Loss: 0.9341461764713077
Epoch 16:
        Train: Average Accuracy: 66.96  Average Loss: 0.9093384787741249
        Test: Average Accuracy: 66.19408945686901       Average Loss: 0.9079070333182738
Epoch 17:
        Train: Average Accuracy: 67.545 Average Loss: 0.884331023413762
        Test: Average Accuracy: 66.82308306709265       Average Loss: 0.8851083928307528
Epoch 18:
        Train: Average Accuracy: 68.17333333333333      Average Loss: 0.8624857121317565
        Test: Average Accuracy: 67.34225239616613       Average Loss: 0.865114833350778
Epoch 19:
        Train: Average Accuracy: 68.76833333333333      Average Loss: 0.8432236529234037
        Test: Average Accuracy: 67.72164536741214       Average Loss: 0.847410248890222
Epoch 20:
        Train: Average Accuracy: 69.42  Average Loss: 0.8260712284648958
        Test: Average Accuracy: 68.3905750798722        Average Loss: 0.8315698558668853
Epoch 21:
        Train: Average Accuracy: 69.94666666666667      Average Loss: 0.8106427949183272
        Test: Average Accuracy: 68.97963258785943       Average Loss: 0.8172529491709007
Epoch 22:
        Train: Average Accuracy: 70.60333333333334      Average Loss: 0.7966293050312383
        Test: Average Accuracy: 69.63857827476038       Average Loss: 0.8041879307381649
Epoch 23:
        Train: Average Accuracy: 71.16166666666666      Average Loss: 0.7837841844433879
        Test: Average Accuracy: 70.30750798722045       Average Loss: 0.7921581519014695
Epoch 24:
        Train: Average Accuracy: 71.74166666666666      Average Loss: 0.7719113202373667
        Test: Average Accuracy: 70.66693290734824       Average Loss: 0.7809926673128937
Epoch 25:
        Train: Average Accuracy: 72.225 Average Loss: 0.7608551651848414
        Test: Average Accuracy: 71.19608626198082       Average Loss: 0.770556897563626
Epoch 26:
        Train: Average Accuracy: 72.71333333333334      Average Loss: 0.7504925148790702
        Test: Average Accuracy: 71.61541533546325       Average Loss: 0.7607454385336822
Epoch 27:
        Train: Average Accuracy: 73.14333333333333      Average Loss: 0.740726302511766
        Test: Average Accuracy: 72.08466453674122       Average Loss: 0.7514749340663955
Epoch 28:
        Train: Average Accuracy: 73.60166666666667      Average Loss: 0.7314794586844252
        Test: Average Accuracy: 72.54392971246007       Average Loss: 0.7426788579609483
Epoch 29:
        Train: Average Accuracy: 74.00666666666666      Average Loss: 0.722688369684692
        Test: Average Accuracy: 72.87340255591054       Average Loss: 0.7343020435449654
Epoch 30:
        Train: Average Accuracy: 74.355 Average Loss: 0.7143060291609384
        Test: Average Accuracy: 73.19289137380191       Average Loss: 0.7263023655483968
```

Learning Rate X 10

In this part, when we multiplied the optimum learning rate by 10, we faced with the problem of dying Leaky-Relu and its obvious because weight will increase and it can cuase in vanishing in gradients. We can see that there is a small change between accuracies of different epochs.

In [102]:

```
INPUT_SHAPE = 784
LEARNING_RATE = (1e-4) * 10
EPOCHS = 30
BATCH_SIZE = 32
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=Relu(), weight_initializer='he')
network.add_layer(10, activation=Relu(), weight_initializer='he')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 9.99   Average Loss: 2.3883120647417364
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 2:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 3:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 4:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 5:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 6:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 7:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 8:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 9:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 10:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 11:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
Epoch 12:
        Train: Average Accuracy: 10.0   Average Loss: 2.302585092994046
        Test: Average Accuracy: 9.984025559105431       Average Loss: 2.302585092994046
```

```
Epoch 13:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 14:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 15:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 16:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 17:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 18:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 19:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 20:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 21:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 22:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 23:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 24:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 25:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 26:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 27:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 28:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 29:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
Epoch 30:
	Train: Average Accuracy: 10.0	Average Loss: 2.302585092994046
	Test: Average Accuracy: 9.984025559105431	Average Loss: 2.302585092994046
```

Step #3

Why Leaky-Relu is better than Relu?

Firstly, Leaky-Relu has a small slope for negative values, instead of altogether zero. So, this modification will fix the “dying ReLU” problem, as it doesn’t have zero-slope parts.

Secondly, it is faster than Relu in training speeds up training, because having the “mean activation” be close to 0 makes training faster.

Finally, unlike Relu, Leaky-Relu is more “balanced,” and may therefore learn faster.

Why Sigmoid and Tanh do not work well?

First Problem with **Sigmoid** is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Therefore, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, we must pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. If the initial weights are too large then most neurons would become saturated and the network will barely learn.

Second Problem with **Sigmoid** is that its outputs are not zero-centered . This is undesirable since neurons in later layers of processing in a Neural Network would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive, then the gradient on the weights w will during backpropagation become either all be positive, or all negative. This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, notice that once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue. Therefore, this is an inconvenience but it has less severe consequences compared to the saturated activation problem above.

Tanh squashes a real-valued number to the range [-1, 1]. Like the Sigmoid neuron, its activations saturate, but unlike the Sigmoid neuron its output is zero-centered. Therefore, in practice the Tanh non-linearity is always preferred to the sigmoid nonlinearity.

Using Leaky-Relu

In [103]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
EPOCHS = 30
BATCH_SIZE = 32
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=LeakyRelu(), weight_initializer='uniform')
network.add_layer(10, activation=LeakyRelu(), weight_initializer='uniform')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 34.64  Average Loss: 1.7565450622794638
        Test: Average Accuracy: 58.66613418530351       Average Loss: 1.1706641184127702
Epoch 2:
        Train: Average Accuracy: 63.88333333333333      Average Loss: 0.9648554939944207
        Test: Average Accuracy: 66.85303514376997       Average Loss: 0.859489450084683
Epoch 3:
        Train: Average Accuracy: 70.685 Average Loss: 0.7856874578230076
        Test: Average Accuracy: 72.51397763578275       Average Loss: 0.7504397609557348
Epoch 4:
        Train: Average Accuracy: 74.33666666666667      Average Loss: 0.7039351446774033
        Test: Average Accuracy: 75.39936102236422       Average Loss: 0.6872892325514437
Epoch 5:
        Train: Average Accuracy: 76.82  Average Loss: 0.6486018161843424
        Test: Average Accuracy: 77.2064696485623        Average Loss: 0.6455075324011997
Epoch 6:
        Train: Average Accuracy: 78.49333333333334      Average Loss: 0.6081594421256649
        Test: Average Accuracy: 78.1349840255591        Average Loss: 0.6169222679815584
Epoch 7:
        Train: Average Accuracy: 79.69  Average Loss: 0.5781399272903787
        Test: Average Accuracy: 78.65415335463258       Average Loss: 0.5968569824714275
Epoch 8:
        Train: Average Accuracy: 80.635 Average Loss: 0.5558577407747289
        Test: Average Accuracy: 79.22324281150159       Average Loss: 0.5819978638758591
Epoch 9:
        Train: Average Accuracy: 81.215 Average Loss: 0.5391290549258874
        Test: Average Accuracy: 79.54273162939297       Average Loss: 0.5703806817773789
Epoch 10:
        Train: Average Accuracy: 81.64166666666667      Average Loss: 0.5262389922216119
        Test: Average Accuracy: 79.94209265175719       Average Loss: 0.5611274973002737
Epoch 11:
        Train: Average Accuracy: 81.96833333333333      Average Loss: 0.5159981391082645
        Test: Average Accuracy: 80.27156549520767       Average Loss: 0.5538381342567462
Epoch 12:
        Train: Average Accuracy: 82.265 Average Loss: 0.5076448679107269
        Test: Average Accuracy: 80.5111821086262        Average Loss: 0.548214194701519
Epoch 13:
        Train: Average Accuracy: 82.53  Average Loss: 0.5006864222227734
        Test: Average Accuracy: 80.74081469648563       Average Loss: 0.5439733067364648
Epoch 14:
        Train: Average Accuracy: 82.73  Average Loss: 0.4948017354931971
        Test: Average Accuracy: 80.86062300319489       Average Loss: 0.5408656279271704
Epoch 15:
        Train: Average Accuracy: 82.91  Average Loss: 0.48977514778108283
        Test: Average Accuracy: 80.85063897763578       Average Loss: 0.5386879000583269
Epoch 16:
        Train: Average Accuracy: 83.07333333333334      Average Loss: 0.4854536473879928
        Test: Average Accuracy: 80.98043130990415       Average Loss: 0.537293081638653
Epoch 17:
        Train: Average Accuracy: 83.21  Average Loss: 0.48172995430874743
        Test: Average Accuracy: 80.98043130990415       Average Loss: 0.5365732882814256
Epoch 18:
        Train: Average Accuracy: 83.34833333333333      Average Loss: 0.4785209907439719
        Test: Average Accuracy: 81.05031948881789       Average Loss: 0.5364686779297317
Epoch 19:
        Train: Average Accuracy: 83.46333333333334      Average Loss: 0.47576647742794415
        Test: Average Accuracy: 80.98043130990415       Average Loss: 0.5369140808694652
Epoch 20:
        Train: Average Accuracy: 83.54333333333334      Average Loss: 0.4734246156915109
        Test: Average Accuracy: 80.86062300319489       Average Loss: 0.537877915828281
Epoch 21:
        Train: Average Accuracy: 83.66166666666666      Average Loss: 0.47146066829666977
        Test: Average Accuracy: 80.82068690095846       Average Loss: 0.539337880667134
Epoch 22:
        Train: Average Accuracy: 83.72666666666667      Average Loss: 0.4698486997821265
        Test: Average Accuracy: 80.7008785942492        Average Loss: 0.5412800001114175
Epoch 23:
        Train: Average Accuracy: 83.77666666666667      Average Loss: 0.46856978559442164
        Test: Average Accuracy: 80.5011980830671        Average Loss: 0.5436985758210009
Epoch 24:
        Train: Average Accuracy: 83.84  Average Loss: 0.46761089354381846
        Test: Average Accuracy: 80.41134185303514       Average Loss: 0.5465928943616067
Epoch 25:
        Train: Average Accuracy: 83.88833333333334      Average Loss: 0.4669642889389237
        Test: Average Accuracy: 80.3314696485623        Average Loss: 0.5499673614222973
Epoch 26:
        Train: Average Accuracy: 83.87333333333333      Average Loss: 0.4666273170866861
        Test: Average Accuracy: 80.35143769968052       Average Loss: 0.5538306706446873
Epoch 27:
        Train: Average Accuracy: 83.91166666666666      Average Loss: 0.46660212398320655
        Test: Average Accuracy: 80.29153354632588       Average Loss: 0.5581947808220817
Epoch 28:
        Train: Average Accuracy: 83.915 Average Loss: 0.4668957133662736
        Test: Average Accuracy: 80.11182108626198       Average Loss: 0.5630767752073818
Epoch 29:
        Train: Average Accuracy: 83.93  Average Loss: 0.4675202772321911
        Test: Average Accuracy: 80.04193290734824       Average Loss: 0.5684967451101681
Epoch 30:
        Train: Average Accuracy: 83.965 Average Loss: 0.4684947416939751
        Test: Average Accuracy: 79.90215654952077       Average Loss: 0.5744833209132603
```

Using Tanh

In [114]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
EPOCHS = 30
BATCH_SIZE = 32
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=Tanh(), weight_initializer='normal')
network.add_layer(10, activation=Tanh(), weight_initializer='normal')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 9.973333333333333      Average Loss: 2.302978394510847
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3028611472069294
Epoch 2:
        Train: Average Accuracy: 9.868333333333334      Average Loss: 2.3029365067888006
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3028237948411827
Epoch 3:
        Train: Average Accuracy: 9.865  Average Loss: 2.302868242386226
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.302728347596802
Epoch 4:
        Train: Average Accuracy: 9.87   Average Loss: 2.302725447416856
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3025390424257752
Epoch 5:
        Train: Average Accuracy: 9.918333333333333      Average Loss: 2.3024578683164667
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3021892638253867
Epoch 6:
        Train: Average Accuracy: 10.075 Average Loss: 2.3019520766313515
        Test: Average Accuracy: 10.023961661341852      Average Loss: 2.301483589220706
Epoch 7:
        Train: Average Accuracy: 10.496666666666666     Average Loss: 2.300550721502764
        Test: Average Accuracy: 10.303514376996805      Average Loss: 2.298437009235665
Epoch 8:
        Train: Average Accuracy: 19.871666666666666     Average Loss: 2.2323306178339126
        Test: Average Accuracy: 25.93849840255591       Average Loss: 2.0717675566617673
Epoch 9:
        Train: Average Accuracy: 31.723333333333333     Average Loss: 1.9627276508338483
        Test: Average Accuracy: 38.11900958466454       Average Loss: 1.8638086965259937
Epoch 10:
        Train: Average Accuracy: 41.755 Average Loss: 1.7838999572836114
        Test: Average Accuracy: 40.92452076677316       Average Loss: 1.7366158910382443
Epoch 11:
        Train: Average Accuracy: 43.31333333333333      Average Loss: 1.7027718717362286
        Test: Average Accuracy: 40.31549520766773       Average Loss: 1.6820352124938964
Epoch 12:
        Train: Average Accuracy: 43.181666666666665     Average Loss: 1.6615116643053374
        Test: Average Accuracy: 42.442092651757186      Average Loss: 1.6504251435399782
Epoch 13:
        Train: Average Accuracy: 43.30833333333333      Average Loss: 1.6324459211067215
        Test: Average Accuracy: 42.18250798722045       Average Loss: 1.6244022566390557
Epoch 14:
        Train: Average Accuracy: 42.473333333333336     Average Loss: 1.606318402788496
        Test: Average Accuracy: 41.423722044728436      Average Loss: 1.5980450445603058
Epoch 15:
        Train: Average Accuracy: 44.035 Average Loss: 1.5774536498133664
        Test: Average Accuracy: 49.65055910543131       Average Loss: 1.5686621602835766
Epoch 16:
        Train: Average Accuracy: 47.391666666666666     Average Loss: 1.5475228301979407
        Test: Average Accuracy: 45.37739616613418       Average Loss: 1.5428521778540891
Epoch 17:
        Train: Average Accuracy: 45.865 Average Loss: 1.5251216171633704
        Test: Average Accuracy: 45.57707667731629       Average Loss: 1.5250605589452666
Epoch 18:
        Train: Average Accuracy: 46.11  Average Loss: 1.510667550310688
        Test: Average Accuracy: 45.82667731629393       Average Loss: 1.5142977991414999
Epoch 19:
        Train: Average Accuracy: 45.82833333333333      Average Loss: 1.4986022907675187
        Test: Average Accuracy: 45.48722044728434       Average Loss: 1.4973788485703359
Epoch 20:
        Train: Average Accuracy: 45.585 Average Loss: 1.486293460378758
        Test: Average Accuracy: 45.397364217252395      Average Loss: 1.4852089600466056
Epoch 21:
        Train: Average Accuracy: 45.13333333333333      Average Loss: 1.4793319120122974
        Test: Average Accuracy: 44.98801916932907       Average Loss: 1.4791817228860813
Epoch 22:
        Train: Average Accuracy: 44.931666666666665     Average Loss: 1.4753250057586371
        Test: Average Accuracy: 50.489217252396166      Average Loss: 1.4746859847858464
Epoch 23:
        Train: Average Accuracy: 50.04  Average Loss: 1.4709327984925802
        Test: Average Accuracy: 50.039936102236425      Average Loss: 1.4719708682154775
Epoch 24:
        Train: Average Accuracy: 50.083333333333336     Average Loss: 1.4653585058278895
        Test: Average Accuracy: 50.3694089456869        Average Loss: 1.466597480497699
Epoch 25:
        Train: Average Accuracy: 50.23833333333334      Average Loss: 1.4587734905267444
        Test: Average Accuracy: 50.069888178913736      Average Loss: 1.4594653040803025
Epoch 26:
        Train: Average Accuracy: 49.98166666666667      Average Loss: 1.4410673286612903
        Test: Average Accuracy: 49.900159744408946      Average Loss: 1.4339656471855986
Epoch 27:
        Train: Average Accuracy: 49.766666666666666     Average Loss: 1.4242586946291307
        Test: Average Accuracy: 49.50079872204473       Average Loss: 1.4307523167757428
Epoch 28:
        Train: Average Accuracy: 49.555 Average Loss: 1.4274586027301428
        Test: Average Accuracy: 49.840255591054316      Average Loss: 1.4351264640437806
Epoch 29:
        Train: Average Accuracy: 50.291666666666664     Average Loss: 1.4313638188790878
        Test: Average Accuracy: 50.44928115015974       Average Loss: 1.4385938351691336
Epoch 30:

/snap/jupyter/6/lib/python3.7/site-packages/ipykernel_launcher.py:87: RuntimeWarning: overflow encou
ntered in exp
/snap/jupyter/6/lib/python3.7/site-packages/ipykernel_launcher.py:87: RuntimeWarning: invalid value
encountered in true_divide

        Train: Average Accuracy: 38.07833333333333      Average Loss: nan
        Test: Average Accuracy: 9.984025559105431       Average Loss: nan
```

Using Sigmoid

In [105]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
EPOCHS = 30
BATCH_SIZE = 32
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=Sigmoid(), weight_initializer='uniform')
network.add_layer(10, activation=Sigmoid(), weight_initializer='uniform')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 10.001666666666667     Average Loss: 2.3025623211933173
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3024330445096193
Epoch 2:
        Train: Average Accuracy: 9.911666666666667      Average Loss: 2.3024518079966776
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.302393997914763
Epoch 3:
        Train: Average Accuracy: 9.831666666666667      Average Loss: 2.3024208493756895
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.302366261135662
Epoch 4:
        Train: Average Accuracy: 9.78   Average Loss: 2.3023895481903836
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3023327162248766
Epoch 5:
        Train: Average Accuracy: 9.838333333333333      Average Loss: 2.3023531971787956
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.30229464762954
Epoch 6:
        Train: Average Accuracy: 9.821666666666667      Average Loss: 2.302313042778187
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3022531399305644
Epoch 7:
        Train: Average Accuracy: 9.82   Average Loss: 2.302269592545211
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.302208358355521
Epoch 8:
        Train: Average Accuracy: 9.836666666666666      Average Loss: 2.3022227274823077
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.302160020962459
Epoch 9:
        Train: Average Accuracy: 9.855  Average Loss: 2.3021720351244497
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.302107639536882
Epoch 10:
        Train: Average Accuracy: 9.87   Average Loss: 2.302116949753958
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3020506008794412
Epoch 11:
        Train: Average Accuracy: 9.891666666666667      Average Loss: 2.30205679133409
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3019881808375886
Epoch 12:
        Train: Average Accuracy: 9.91   Average Loss: 2.30199076196172
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.301919529635685
Epoch 13:
        Train: Average Accuracy: 9.953333333333333      Average Loss: 2.301917921731497
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.301843640649264
Epoch 14:
        Train: Average Accuracy: 9.953333333333333      Average Loss: 2.301837149348538
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3017593037596744
Epoch 15:
        Train: Average Accuracy: 9.968333333333334      Average Loss: 2.3017470853639783
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3016650387571898
Epoch 16:
        Train: Average Accuracy: 9.995  Average Loss: 2.3016460509230634
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3015589992695626
Epoch 17:
        Train: Average Accuracy: 10.023333333333333     Average Loss: 2.3015319290989766
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.301438830684601
Epoch 18:
        Train: Average Accuracy: 10.038333333333334     Average Loss: 2.301401986611192
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3013014536063867
Epoch 19:
        Train: Average Accuracy: 10.061666666666667     Average Loss: 2.3012525972097526
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.301142722593142
Epoch 20:
        Train: Average Accuracy: 10.08  Average Loss: 2.3010787970760647
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.3009568684334054
Epoch 21:
        Train: Average Accuracy: 10.128333333333334     Average Loss: 2.3008735424670315
        Test: Average Accuracy: 9.994009584664537       Average Loss: 2.30073555038454
Epoch 22:
        Train: Average Accuracy: 10.185 Average Loss: 2.3006264189892836
        Test: Average Accuracy: 10.023961661341852      Average Loss: 2.3004661782391533
Epoch 23:
        Train: Average Accuracy: 10.245 Average Loss: 2.300321302306631
        Test: Average Accuracy: 10.043929712460065      Average Loss: 2.3001288173194623
Epoch 24:
        Train: Average Accuracy: 10.318333333333333     Average Loss: 2.2999319489136973
        Test: Average Accuracy: 10.043929712460065      Average Loss: 2.299690268356238
Epoch 25:
        Train: Average Accuracy: 10.413333333333334     Average Loss: 2.299413443368272
        Test: Average Accuracy: 10.043929712460065      Average Loss: 2.299092513810644
Epoch 26:
        Train: Average Accuracy: 10.613333333333333     Average Loss: 2.298685649254178
        Test: Average Accuracy: 10.123801916932907      Average Loss: 2.298230755515051
Epoch 27:
        Train: Average Accuracy: 11.038333333333334     Average Loss: 2.297603956425433
        Test: Average Accuracy: 10.223642172523961      Average Loss: 2.2969179369880686
Epoch 28:
        Train: Average Accuracy: 12.976666666666667     Average Loss: 2.295924011080525
        Test: Average Accuracy: 11.631389776357828      Average Loss: 2.2948580806889725
Epoch 29:
        Train: Average Accuracy: 18.048333333333332     Average Loss: 2.2933169052718743
        Test: Average Accuracy: 17.751597444089455      Average Loss: 2.2917187508986907
Epoch 30:
        Train: Average Accuracy: 20.641666666666666     Average Loss: 2.289522067710178
        Test: Average Accuracy: 20.027955271565496      Average Loss: 2.2873380088492863
```

Step #4

What is the philosophy behind the use of batches?

We use batches instead of the whole data for the following reasons:

1- It requires less memory. As we train our network using fewer samples, the overall training process requires less memory. This is very important when we are not able to fit the whole dataset in our machine's memory.

2- Typically networks train faster with mini-batches, because we update the weights after each propagation. If we used all samples during propagation, we would make only 1 update for the network's parameter and this can lead to underfitting.

What is the problem with extremely small batches?

There is no doubt that the smaller the batch is the less accurate the estimation of the gradient will be. If we use extremely small batch size, we would lose the effectiveness of vectorization. Moreover, It will produce a noisier gradient descent. As a result, we prefer to use bigger batch size.

Setting Batch-Size to 16

In this part, when we set the batch-size to 16, the speed of learning increased and our model reached to the desirable accuracy faster. It is obvious that the more batches we train our model, the more weight updates are performed and we get better results.

In [106]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
EPOCHS = 30
BATCH_SIZE = 16
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=LeakyRelu(), weight_initializer='uniform')
network.add_layer(10, activation=LeakyRelu(), weight_initializer='uniform')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 35.915 Average Loss: 1.744707502195137
        Test: Average Accuracy: 58.29   Average Loss: 1.1432801330785543
Epoch 2:
        Train: Average Accuracy: 63.916666666666664     Average Loss: 0.9608971289877295
        Test: Average Accuracy: 67.72   Average Loss: 0.8499426383919035
Epoch 3:
        Train: Average Accuracy: 70.49333333333334      Average Loss: 0.7863745017530492
        Test: Average Accuracy: 72.92   Average Loss: 0.7456677127700301
Epoch 4:
        Train: Average Accuracy: 74.61833333333334      Average Loss: 0.7025278780256119
        Test: Average Accuracy: 75.97   Average Loss: 0.6798142975200804
Epoch 5:
        Train: Average Accuracy: 77.12666666666667      Average Loss: 0.6450549453740927
        Test: Average Accuracy: 77.69   Average Loss: 0.6343844527306115
Epoch 6:
        Train: Average Accuracy: 78.90666666666667      Average Loss: 0.6036528673438222
        Test: Average Accuracy: 78.64   Average Loss: 0.6027784781520771
Epoch 7:
        Train: Average Accuracy: 80.11833333333334      Average Loss: 0.573388421545949
        Test: Average Accuracy: 79.55   Average Loss: 0.5804011999190097
Epoch 8:
        Train: Average Accuracy: 80.97166666666666      Average Loss: 0.5510252722291531
        Test: Average Accuracy: 80.2    Average Loss: 0.5639903843436455
Epoch 9:
        Train: Average Accuracy: 81.46666666666667      Average Loss: 0.5341724508733596
        Test: Average Accuracy: 80.61   Average Loss: 0.5513818669175871
Epoch 10:
        Train: Average Accuracy: 81.855 Average Loss: 0.5210575137195942
        Test: Average Accuracy: 80.86   Average Loss: 0.5413355119616221
Epoch 11:
        Train: Average Accuracy: 82.22333333333333      Average Loss: 0.51050166720331
        Test: Average Accuracy: 81.13   Average Loss: 0.5331823436758006
Epoch 12:
        Train: Average Accuracy: 82.44333333333333      Average Loss: 0.5017545949012525
        Test: Average Accuracy: 81.3    Average Loss: 0.5264977894296721
Epoch 13:
        Train: Average Accuracy: 82.69833333333334      Average Loss: 0.4943487933932043
        Test: Average Accuracy: 81.5    Average Loss: 0.5209750476205826
Epoch 14:
        Train: Average Accuracy: 82.915 Average Loss: 0.487967498740787
        Test: Average Accuracy: 81.7    Average Loss: 0.516383179781912
Epoch 15:
        Train: Average Accuracy: 83.11833333333334      Average Loss: 0.4824068290319446
        Test: Average Accuracy: 81.75   Average Loss: 0.5125529760071933
Epoch 16:
        Train: Average Accuracy: 83.28333333333333      Average Loss: 0.47752274360680746
        Test: Average Accuracy: 81.81   Average Loss: 0.5093538127741529
Epoch 17:
        Train: Average Accuracy: 83.46  Average Loss: 0.473207744850723
        Test: Average Accuracy: 81.87   Average Loss: 0.5066866813395839
Epoch 18:
        Train: Average Accuracy: 83.61166666666666      Average Loss: 0.4693795550623317
        Test: Average Accuracy: 81.99   Average Loss: 0.504474810518272
Epoch 19:
        Train: Average Accuracy: 83.77666666666667      Average Loss: 0.4659735469123208
        Test: Average Accuracy: 81.98   Average Loss: 0.5026552242905353
Epoch 20:
        Train: Average Accuracy: 83.88333333333334      Average Loss: 0.46293313060227265
        Test: Average Accuracy: 82.12   Average Loss: 0.5011831448577846
Epoch 21:
        Train: Average Accuracy: 83.98666666666666      Average Loss: 0.4602174818145361
        Test: Average Accuracy: 82.09   Average Loss: 0.5000181756726851
Epoch 22:
        Train: Average Accuracy: 84.07833333333333      Average Loss: 0.45778921195608674
        Test: Average Accuracy: 82.14   Average Loss: 0.499125336266436
Epoch 23:
        Train: Average Accuracy: 84.15166666666667      Average Loss: 0.4556108578034651
        Test: Average Accuracy: 82.23   Average Loss: 0.4984782793262334
Epoch 24:
        Train: Average Accuracy: 84.25  Average Loss: 0.45366084802759676
        Test: Average Accuracy: 82.25   Average Loss: 0.4980559029424401
Epoch 25:
        Train: Average Accuracy: 84.29333333333334      Average Loss: 0.4519162537313247
        Test: Average Accuracy: 82.3    Average Loss: 0.49783894246813654
Epoch 26:
        Train: Average Accuracy: 84.34833333333333      Average Loss: 0.4503574155694988
        Test: Average Accuracy: 82.35   Average Loss: 0.49781154984578824
Epoch 27:
        Train: Average Accuracy: 84.41  Average Loss: 0.44896775618250967
        Test: Average Accuracy: 82.3    Average Loss: 0.497959033053868
Epoch 28:
        Train: Average Accuracy: 84.415 Average Loss: 0.4477330693006329
        Test: Average Accuracy: 82.34   Average Loss: 0.4982693187654825
Epoch 29:
        Train: Average Accuracy: 84.485 Average Loss: 0.4466412602174919
        Test: Average Accuracy: 82.27   Average Loss: 0.4987318076739843
Epoch 30:
        Train: Average Accuracy: 84.525 Average Loss: 0.445682063146415
        Test: Average Accuracy: 82.28   Average Loss: 0.49933787573007893
```

Setting Batch-Size to 128

In this part, when we set the batch-size to 128, the speed of learning decreased. So, it is better to use smaller batch size to make the process of learning faster.

In [107]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
EPOCHS = 30
BATCH_SIZE = 128
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE, shuffle=True)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE, shuffle=True)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=LeakyRelu(), weight_initializer='uniform')
network.add_layer(10, activation=LeakyRelu(), weight_initializer='uniform')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 10.250088841506752     Average Loss: 2.340221710371429
        Test: Average Accuracy: 10.03757911392405       Average Loss: 2.2989253166781203
Epoch 2:
        Train: Average Accuracy: 9.955246090973704      Average Loss: 2.298057653787562
        Test: Average Accuracy: 9.889240506329115       Average Loss: 2.295701465886236
Epoch 3:
        Train: Average Accuracy: 9.699271499644633      Average Loss: 2.2938159631836728
        Test: Average Accuracy: 9.731012658227849       Average Loss: 2.2921471449945
Epoch 4:
        Train: Average Accuracy: 9.864738805970148      Average Loss: 2.2923725743463446
        Test: Average Accuracy: 10.314477848101266      Average Loss: 2.2908231261905474
Epoch 5:
        Train: Average Accuracy: 9.952469793887703      Average Loss: 2.289488244755758
        Test: Average Accuracy: 9.889240506329115       Average Loss: 2.287670181550202
Epoch 6:
        Train: Average Accuracy: 10.278962331201136     Average Loss: 2.2880905372643556
        Test: Average Accuracy: 10.245253164556962      Average Loss: 2.2860076846671658
Epoch 7:
        Train: Average Accuracy: 10.9397210376688       Average Loss: 2.2850117678665023
        Test: Average Accuracy: 17.335838607594937      Average Loss: 2.2836027013682565
Epoch 8:
        Train: Average Accuracy: 9.964130241648897      Average Loss: 2.2829977982444714
        Test: Average Accuracy: 10.027689873417721      Average Loss: 2.282410098290485
Epoch 9:
        Train: Average Accuracy: 10.119047619047619     Average Loss: 2.28064641071483
        Test: Average Accuracy: 10.096914556962025      Average Loss: 2.2818843916083202
Epoch 10:
        Train: Average Accuracy: 10.714285714285714     Average Loss: 2.27930703626554
        Test: Average Accuracy: 9.889240506329115       Average Loss: 2.277934797478933
Epoch 11:
        Train: Average Accuracy: 11.283426616915422     Average Loss: 2.277280607671861
        Test: Average Accuracy: 12.589003164556962      Average Loss: 2.2762367416004348
Epoch 12:
        Train: Average Accuracy: 10.395011549395878     Average Loss: 2.2759490312019865
        Test: Average Accuracy: 10.027689873417721      Average Loss: 2.2751929082713884
Epoch 13:
        Train: Average Accuracy: 12.484452736318408     Average Loss: 2.2743940980140187
        Test: Average Accuracy: 14.349287974683545      Average Loss: 2.273512059866415
Epoch 14:
        Train: Average Accuracy: 10.673751776830134     Average Loss: 2.273053548403202
        Test: Average Accuracy: 9.958465189873417       Average Loss: 2.272247716474317
Epoch 15:
        Train: Average Accuracy: 10.928615849324803     Average Loss: 2.271575122868152
        Test: Average Accuracy: 18.730221518987342      Average Loss: 2.2704987525701967
Epoch 16:
        Train: Average Accuracy: 11.26843461265103      Average Loss: 2.271773122933735
        Test: Average Accuracy: 14.102056962025317      Average Loss: 2.270239930479319
Epoch 17:
        Train: Average Accuracy: 11.766502309879176     Average Loss: 2.2694836322340444
        Test: Average Accuracy: 10.166139240506329      Average Loss: 2.2689258467983016
Epoch 18:
        Train: Average Accuracy: 12.300661869225303     Average Loss: 2.2677024447803356
        Test: Average Accuracy: 14.992088607594937      Average Loss: 2.266045488856361
Epoch 19:
        Train: Average Accuracy: 15.768812189054726     Average Loss: 2.266485270183369
        Test: Average Accuracy: 15.081091772151899      Average Loss: 2.2657367943209437
Epoch 20:
        Train: Average Accuracy: 15.338486140724946     Average Loss: 2.265504001505502
        Test: Average Accuracy: 14.072389240506329      Average Loss: 2.264358207844802
Epoch 21:
        Train: Average Accuracy: 14.642190831556503     Average Loss: 2.2650322741757205
        Test: Average Accuracy: 11.303401898734178      Average Loss: 2.264790171150547
Epoch 22:
        Train: Average Accuracy: 14.41286869225302      Average Loss: 2.263702110009995
        Test: Average Accuracy: 12.005537974683545      Average Loss: 2.262830027043513
Epoch 23:
        Train: Average Accuracy: 13.426172707889126     Average Loss: 2.2625436091147946
        Test: Average Accuracy: 12.440664556962025      Average Loss: 2.2611991062250385
Epoch 24:
        Train: Average Accuracy: 12.547752309879176     Average Loss: 2.260813904577467
        Test: Average Accuracy: 11.61985759493671       Average Loss: 2.259610278712804
Epoch 25:
        Train: Average Accuracy: 14.474502487562187     Average Loss: 2.2594289058610397
        Test: Average Accuracy: 15.29865506329114       Average Loss: 2.259878458464369
Epoch 26:
        Train: Average Accuracy: 12.533870824449183     Average Loss: 2.258206682697295
        Test: Average Accuracy: 14.527294303797468      Average Loss: 2.258428246034738
Epoch 27:
        Train: Average Accuracy: 13.085798685145699     Average Loss: 2.256952484540913
        Test: Average Accuracy: 11.224287974683545      Average Loss: 2.2566351126426696
Epoch 28:
        Train: Average Accuracy: 11.941409026297086     Average Loss: 2.2591862829525895
        Test: Average Accuracy: 11.748417721518987      Average Loss: 2.2560273991064275
Epoch 29:
        Train: Average Accuracy: 13.636616027007818     Average Loss: 2.2557155225005543
        Test: Average Accuracy: 12.509889240506329      Average Loss: 2.2544277173825757
Epoch 30:
        Train: Average Accuracy: 13.894811656005686     Average Loss: 2.2534741196218913
        Test: Average Accuracy: 10.452927215189874      Average Loss: 2.2528345456976626
```

Step #5

Why we use multiple epochs in training Neural Networks?

It is undeniable that we want to get good performance on non-training data and usually that takes more than one pass over the training data. Moreover, it is typical that gradient descent does not reach a global or local minimum after the first epoch. So, training just one epoch can lead to underfitting. As a result, we use multiple epochs.

Overfitting in Neural Networks

One of the signs of overfitting in neural networks is that the test accuracy starts to decrease and we use early stoppung technique to prevent it. When we train our model in many epochs, out model starts to classify the noises in the training data and we shoud stop the process of learning in this situation. Another situation that leads to overfitting is that we consider a very complex architecture for our neural network in a simple problem.

In [108]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
BATCH_SIZE = 16
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(40, input_shape=INPUT_SHAPE, activation=LeakyRelu(), weight_initializer='uniform')
network.add_layer(10, activation=LeakyRelu(), weight_initializer='uniform')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.early_stopping(TRAINLOADER, TESTLOADER)
```

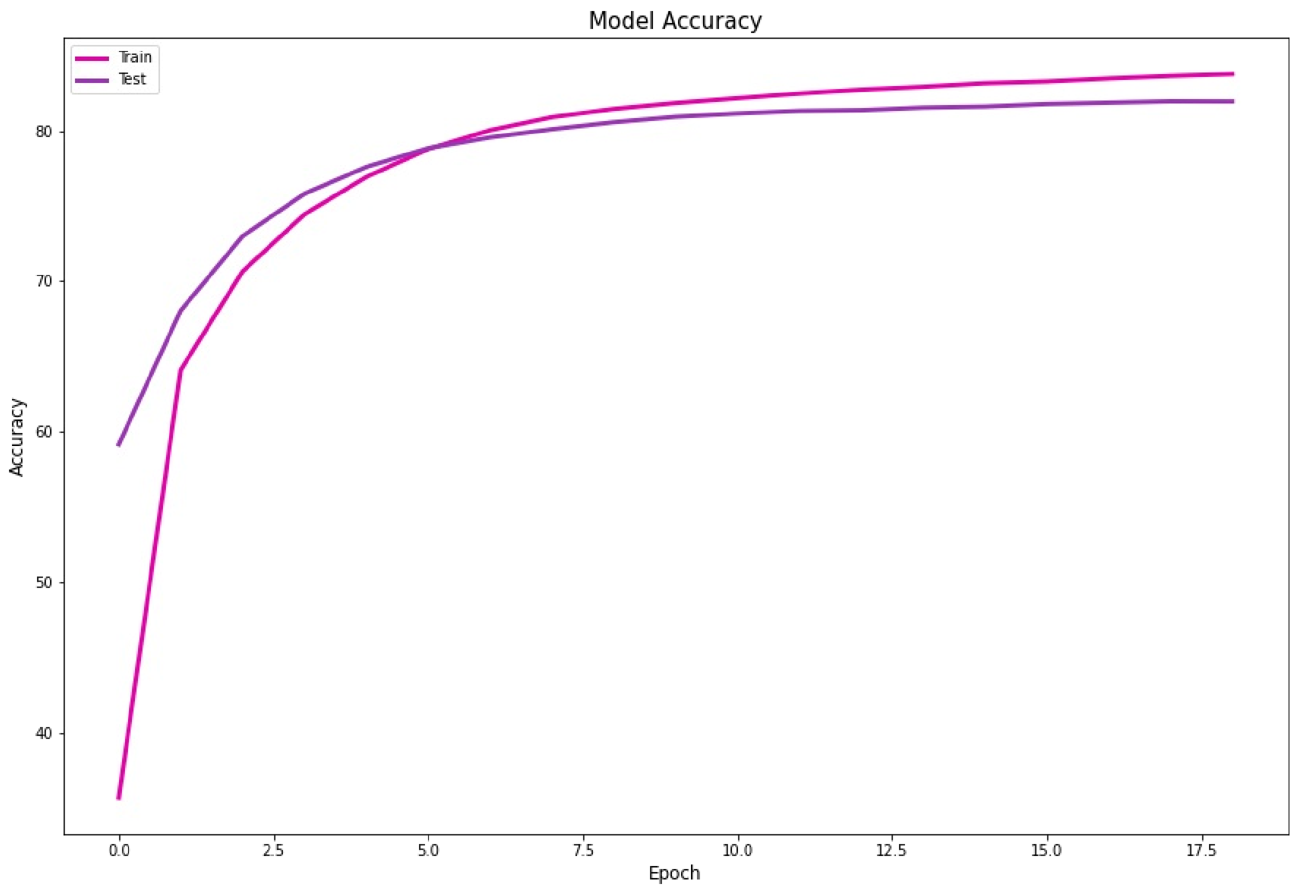
```
Epoch 1:
        Train: Average Accuracy: 35.66833333333334      Average Loss: 1.756776884764874
        Test: Average Accuracy: 59.15   Average Loss: 1.153408016323865
Epoch 2:
        Train: Average Accuracy: 64.09  Average Loss: 0.9639426341238299
        Test: Average Accuracy: 68.04   Average Loss: 0.8476625167933842
Epoch 3:
        Train: Average Accuracy: 70.645 Average Loss: 0.7842946892053245
        Test: Average Accuracy: 72.99   Average Loss: 0.7446588570140661
Epoch 4:
        Train: Average Accuracy: 74.44166666666666      Average Loss: 0.7024318488055302
        Test: Average Accuracy: 75.81   Average Loss: 0.6813653727492837
Epoch 5:
        Train: Average Accuracy: 76.92666666666666      Average Loss: 0.6465723936701218
        Test: Average Accuracy: 77.59   Average Loss: 0.637107505106073
Epoch 6:
        Train: Average Accuracy: 78.78166666666667      Average Loss: 0.6055091410148158
        Test: Average Accuracy: 78.84   Average Loss: 0.6054182859450948
Epoch 7:
        Train: Average Accuracy: 80.035 Average Loss: 0.5747351641205418
        Test: Average Accuracy: 79.56   Average Loss: 0.5824733988338754
Epoch 8:
        Train: Average Accuracy: 80.90333333333334      Average Loss: 0.5515817005465626
        Test: Average Accuracy: 80.1    Average Loss: 0.5655753016285082
Epoch 9:
        Train: Average Accuracy: 81.45333333333333      Average Loss: 0.534047100466523
        Test: Average Accuracy: 80.57   Average Loss: 0.5526731146295094
Epoch 10:
        Train: Average Accuracy: 81.84666666666666      Average Loss: 0.5204501167651053
        Test: Average Accuracy: 80.93   Average Loss: 0.542441562015447
Epoch 11:
        Train: Average Accuracy: 82.165 Average Loss: 0.5095777675654143
        Test: Average Accuracy: 81.15   Average Loss: 0.5341451288010001
Epoch 12:
        Train: Average Accuracy: 82.475 Average Loss: 0.5006286589794716
        Test: Average Accuracy: 81.32   Average Loss: 0.5273369572831873
Epoch 13:
        Train: Average Accuracy: 82.73333333333333      Average Loss: 0.4930886736556193
        Test: Average Accuracy: 81.37   Average Loss: 0.521716466372895
Epoch 14:
        Train: Average Accuracy: 82.92166666666667      Average Loss: 0.4866333318376033
        Test: Average Accuracy: 81.54   Average Loss: 0.5170482026417592
Epoch 15:
        Train: Average Accuracy: 83.16  Average Loss: 0.4810383450917966
        Test: Average Accuracy: 81.61   Average Loss: 0.5131544333916374
Epoch 16:
        Train: Average Accuracy: 83.27666666666667      Average Loss: 0.4761445611450315
        Test: Average Accuracy: 81.78   Average Loss: 0.5099011435463661
Epoch 17:
        Train: Average Accuracy: 83.47666666666667      Average Loss: 0.4718357898587529
        Test: Average Accuracy: 81.87   Average Loss: 0.5071855785673575
Epoch 18:
        Train: Average Accuracy: 83.64333333333333      Average Loss: 0.46802353710851674
        Test: Average Accuracy: 81.96   Average Loss: 0.504928986569173
Epoch 19:
        Train: Average Accuracy: 83.77166666666666      Average Loss: 0.46463731348940146
        Test: Average Accuracy: 81.95   Average Loss: 0.5030682380823875
```

In [109]:

```
fig = plt.figure(figsize=(15, 10))
plt.plot(log['train_accuracy'], linewidth = 3, color = '#e500a9')
plt.plot(log['test_accuracy'], linewidth = 3, color = '#9a34b5')
plt.title('Model Accuracy', fontsize = 15)
plt.ylabel('Accuracy', fontsize = 12)
plt.xlabel('Epoch', fontsize = 12)
plt.legend(['Train', 'Test'], loc='upper left')
```

Out[109]:

```
<matplotlib.legend.Legend at 0x7f678b7826a0>
```

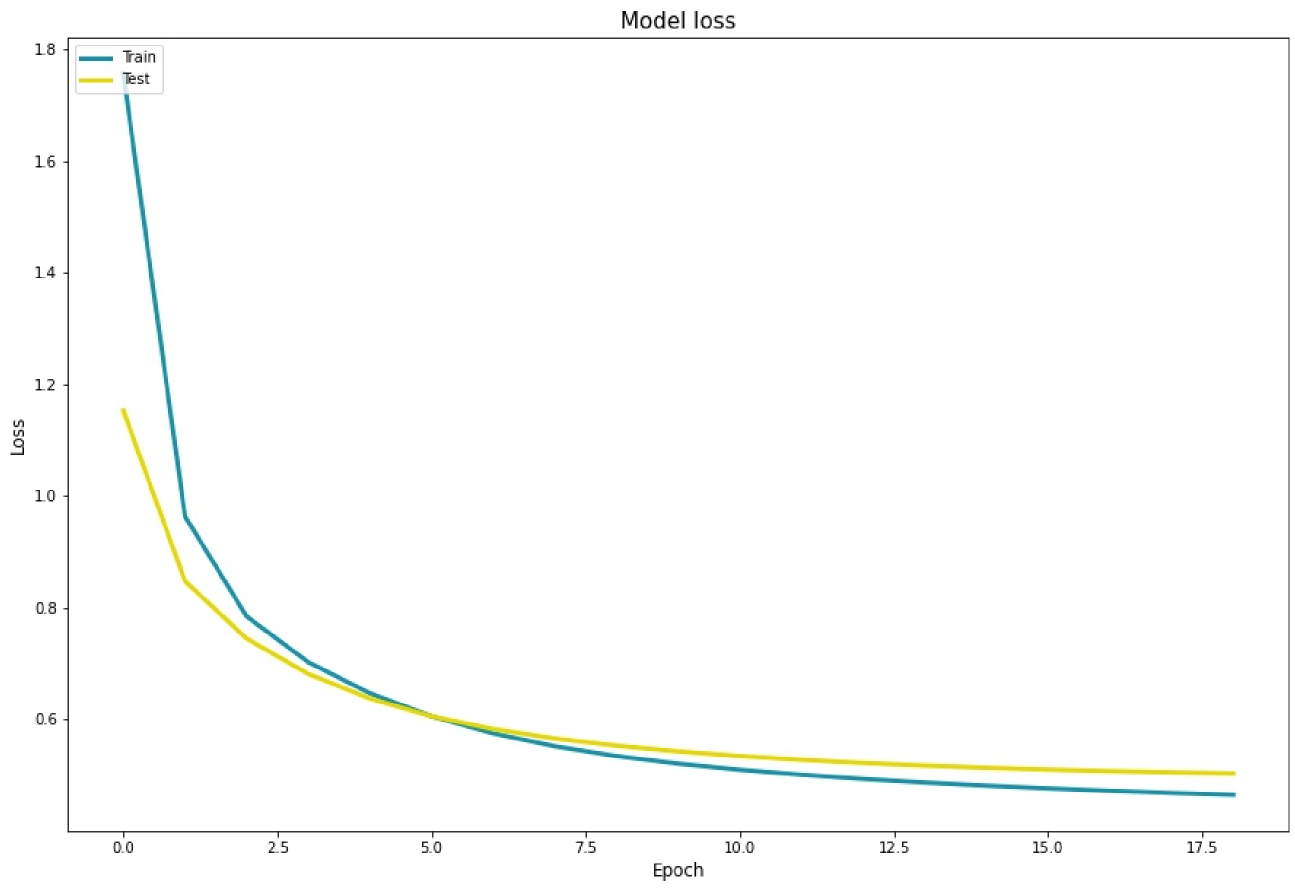


In [110]:

```
fig = plt.figure(figsize=(15, 10))
plt.plot(log['train_loss'], linewidth = 3, color = '#1691a7')
plt.plot(log['test_loss'], linewidth = 3, color = '#e5d700')
plt.title('Model loss', fontsize = 15)
plt.ylabel('Loss', fontsize = 12)
plt.xlabel('Epoch', fontsize = 12)
plt.legend(['Train', 'Test'], loc='upper left')
```

Out[110]:

```
<matplotlib.legend.Legend at 0x7f678b840668>
```



Phase 4: Plotting Reduced Dimension

Based on the results, we can see that our model works well in classifying photos of shoes (class no. 7 and 9) from the other types of clothes. This obvious, because there are many differences between these photos.

Also, we have the same scenario for photos of the bags (class no. 8).

Moreover, our model can classify photos of pants from other clothes (class no. 1).

But, our model gets a little bit confused in classifying other photos because they have a lot of similarities and it seems that we need a more complex architecture for our model to increase its power in classification.

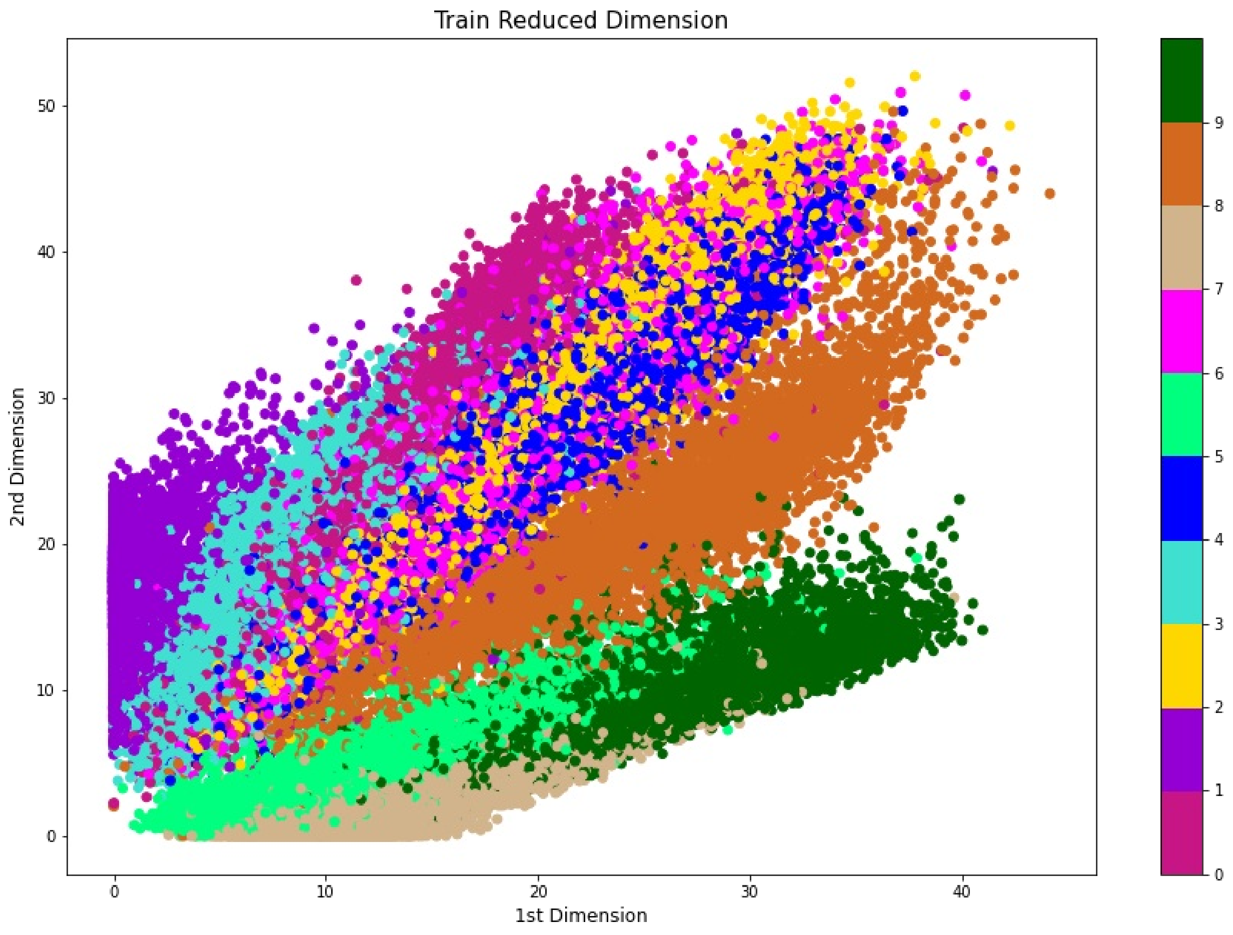
In [111]:

```
INPUT_SHAPE = 784
LEARNING_RATE = 1e-4
EPOCHS = 20
BATCH_SIZE = 16
TRAINLOADER = Dataloader(train_data, train_labels, 10, BATCH_SIZE, shuffle=True)
TESTLOADER = Dataloader(test_data, test_labels, 10, BATCH_SIZE, shuffle=True)

network = FeedForwardNN(INPUT_SHAPE)
network.add_layer(2, input_shape=INPUT_SHAPE, activation=LeakyRelu(), weight_initializer='uniform')
network.add_layer(10, activation=LeakyRelu(), weight_initializer='uniform')
network.set_training_param(loss=CrossEntropy(), lr=LEARNING_RATE)

log = network.fit(EPOCHS, TRAINLOADER, TESTLOADER)
```

```
Epoch 1:
        Train: Average Accuracy: 17.226666666666667     Average Loss: 2.0999771889759993
        Test: Average Accuracy: 27.48   Average Loss: 1.7207062557892772
Epoch 2:
        Train: Average Accuracy: 43.11833333333333      Average Loss: 1.431832586171235
        Test: Average Accuracy: 47.54   Average Loss: 1.2566495714843735
Epoch 3:
        Train: Average Accuracy: 53.42666666666667      Average Loss: 1.1838155270561237
        Test: Average Accuracy: 55.15   Average Loss: 1.1408121650277594
Epoch 4:
        Train: Average Accuracy: 56.72833333333333      Average Loss: 1.09979237918151
        Test: Average Accuracy: 56.91   Average Loss: 1.083583560165818
Epoch 5:
        Train: Average Accuracy: 58.038333333333334     Average Loss: 1.0509894532309134
        Test: Average Accuracy: 58.14   Average Loss: 1.044870247515816
Epoch 6:
        Train: Average Accuracy: 59.51166666666666      Average Loss: 1.0175791938559111
        Test: Average Accuracy: 61.42   Average Loss: 1.0192211509806566
Epoch 7:
        Train: Average Accuracy: 60.53666666666667      Average Loss: 0.9933646701549779
        Test: Average Accuracy: 60.68   Average Loss: 0.9904918770977931
Epoch 8:
        Train: Average Accuracy: 61.31666666666667      Average Loss: 0.9742313842043622
        Test: Average Accuracy: 62.34   Average Loss: 0.9732914250911207
Epoch 9:
        Train: Average Accuracy: 62.255 Average Loss: 0.957931490127191
        Test: Average Accuracy: 63.34   Average Loss: 0.9543157065300569
Epoch 10:
        Train: Average Accuracy: 62.821666666666665     Average Loss: 0.9441934284046437
        Test: Average Accuracy: 63.89   Average Loss: 0.9429332074084806
Epoch 11:
        Train: Average Accuracy: 63.305 Average Loss: 0.9318619153443084
        Test: Average Accuracy: 64.22   Average Loss: 0.9642065633166123
Epoch 12:
        Train: Average Accuracy: 63.69  Average Loss: 0.9218252215231411
        Test: Average Accuracy: 64.11   Average Loss: 0.938494056381391
Epoch 13:
        Train: Average Accuracy: 64.28166666666667      Average Loss: 0.9130211363883461
        Test: Average Accuracy: 62.41   Average Loss: 0.9369098510150359
Epoch 14:
        Train: Average Accuracy: 64.48333333333333      Average Loss: 0.9044989326165587
        Test: Average Accuracy: 64.09   Average Loss: 0.9047364105841538
Epoch 15:
        Train: Average Accuracy: 64.915 Average Loss: 0.8979687563407743
        Test: Average Accuracy: 64.82   Average Loss: 0.9035047253849873
Epoch 16:
        Train: Average Accuracy: 65.14  Average Loss: 0.8906474727504909
        Test: Average Accuracy: 64.7    Average Loss: 0.8934264140858137
Epoch 17:
        Train: Average Accuracy: 65.3   Average Loss: 0.8852887510887483
        Test: Average Accuracy: 63.07   Average Loss: 0.9056552500978209
Epoch 18:
        Train: Average Accuracy: 65.58833333333334      Average Loss: 0.8802255712256322
        Test: Average Accuracy: 66.66   Average Loss: 0.8872962150748728
Epoch 19:
        Train: Average Accuracy: 65.64833333333333      Average Loss: 0.8753662973281414
        Test: Average Accuracy: 65.54   Average Loss: 0.8875391663275337
Epoch 20:
        Train: Average Accuracy: 65.97166666666666      Average Loss: 0.8702385694609366
        Test: Average Accuracy: 65.35   Average Loss: 0.8857335487822112
```

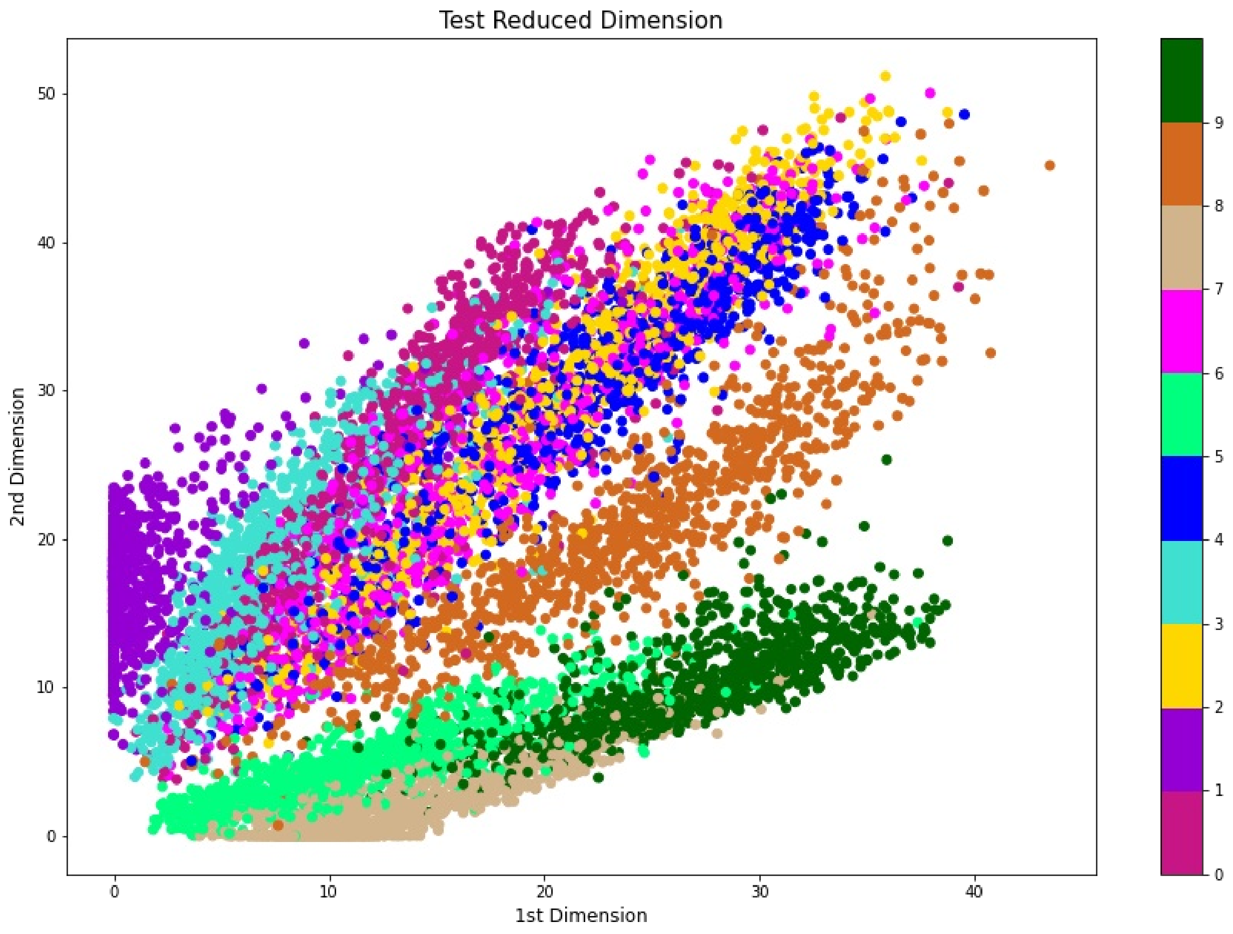
In [112]:

```
network.plot_reduced_dimension_trian()
```



In [113]:

```
network.plot_reduced_dimension_test()
```



Conclusion

In this computer assignment we learned that neural networks are good methods to solve image classification problems. Also, we were learned to some linear algebra methods to implements formulas in a neural network.