EE 258

Project 1

Last Name: Saini

First Name: Deepanshu

ID: 010980814

Date:11/12/2017

Implementing Neural Networks to Classify Handwritten Digits

Table of Contents

Methodology

(describe classifiers used, cross-validation method used etc).

1. <u>Linear Classifier</u>

Description

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.

The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

SGD is implemented using *Scikit-Learns'* **SGDClassifier** class. This classifier has the advantage of being capable of handling very large datasets efficiently. This is in part because SGD deals with training instances independently, one at a time (which also makes SGD well suited for online learning).

Cross-Validation Method Used

The cross-validation method used for evaluating the models' performance is *K-fold cross-validation.*

The *cross_val_score()* function imported from the *sklearn.model_selection* class is used to evaluate the SGDClassifier model using K-fold cross-validation, with three folds. The simplest way to use cross-validation is to call the *cross_val_score* helper function on the estimator and the dataset.

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the *training set*), and validating the analysis on the other subset (called the *validation set* or *testing set*). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to estimate a final predictive model.

In summary, cross-validation combines (averages) measures of fit (prediction error) to derive a more accurate estimate of model prediction performance.

2. <u>K-Nearest Neighbor Classifier</u>

Description

In pattern recognition, the **k-Nearest Neighbor's (k-NN)** algorithm is a non-parametric method used for classification and regression. [1] In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k*is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

Scikit-learn implements two different nearest neighbor's classifiers: **KNeighborsClassifier** implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user. RadiusNeighborsClassifier implements learning based on the number of neighbors within a fixed radius r of each training point, where r is a floating-point value specified by the user.

The k-neighbor's classification in **KNeighborsClassifier** is the more commonly used of the two techniques. The optimal choice of the value k is highly data-dependent: in general, a larger k suppresses the effects of noise, but makes the classification boundaries less distinct

Cross-Validation Method Used

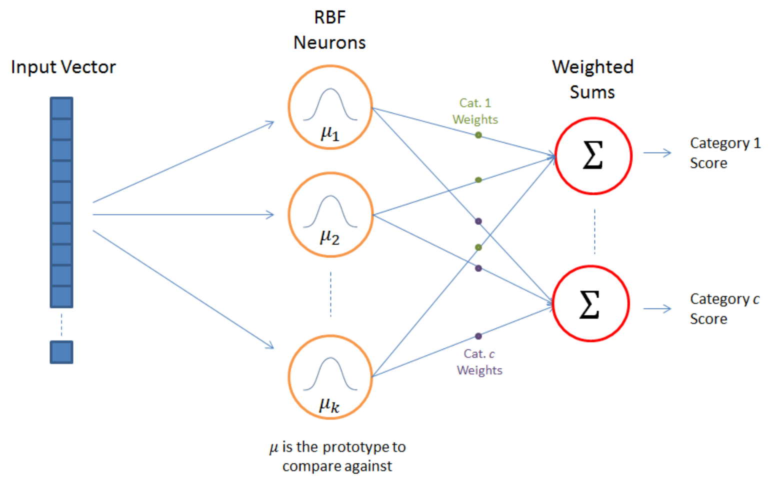
Again, the *K-fold cross-validation method* is used for evaluating the models' performance.

3. Radial Basis Function Neural Network

Description

In the field of mathematical modeling, a **Radial Basis Function Network (RBFN)** is an artificial neural network that uses radial basis functions as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters. Radial basis function networks have many uses, including function approximation, time series prediction, classification, and system control.

An RBFN performs classification by measuring the input's similarity to examples from the training set. Each RBFN neuron stores a "prototype", which is just one of the examples from the training set. When we want to classify a new input, each neuron computes the Euclidean distance between the input and its prototype. Roughly speaking, if the input more closely resembles the class A prototypes than the class B prototypes, it is classified as class A.



The above illustration shows the typical architecture of an RBF Network. It consists of an input vector, a layer of RBF neurons, and an output layer with one node per category or class of data.

Scikit-learn implements RBF using the **sklearn.gaussian_process.kernels.RBF(**aka squared-exponential**)** kernel.

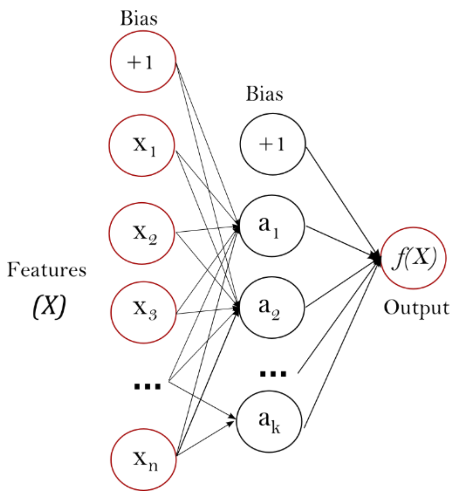
The RBF kernel is a stationary kernel. It is also known as the "squared exponential" kernel. It is parameterized by a length-scale parameter length_scale>0, which can either be a scalar (isotropic variant of the kernel) or a vector with the same number of dimensions as the inputs X (anisotropic variant of the kernel). The kernel is given by:

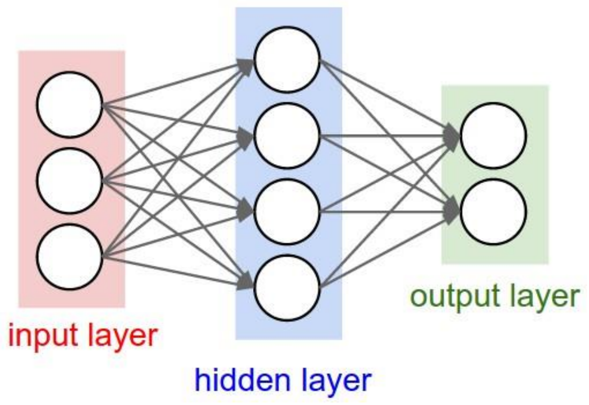
k (x_i, x_j) = exp(-1 / 2 d(x_i / length_scale, x_j / length_scale)^2)

This kernel is infinitely differentiable, which implies that GPs with this kernel as covariance function have mean square derivatives of all orders, and are thus very smooth.

4. <u>One-Hidden Layer Fully Connected Multilayer Neural Network</u>

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features X = {x_1, x_2, ..., x_m} and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. *Figure 1* shows a one hidden layer MLP with scalar output.





Class **MLPClassifier** implements a *multi-layer perceptron* (MLP) algorithm that trains using Backpropagation.

The leftmost layer, known as the input layer, consists of a set of neurons \{x_i | x_1, x_2, ..., x_m\} representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation w_1x_1 + w_2x_2 + ... + w_mx_m, followed by a non-linear activation function like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

MLP trains on two arrays: array X of size (n_samples, n_features), which holds the training samples represented as floating point feature vectors; and array y of size (n_samples,), which holds the target values (class labels) for the training samples.

The module contains the public attributes coefs_ and intercepts_. coefs_ is a list of weight matrices, where weight matrix at index i represents the weights between layer i and layer i+1. intercepts_ is a list of bias vectors, where the vector at index i represents the bias values added to layer i+1.

The advantages of Multi-Layer Perceptron are:

a) Capability to learn non-linear models.
b) Capability to learn models in real-time (on-line learning) using partial_fit.

5. Two-Hidden Layer Fully Connected Multilayer Neural Network

Class **MLPClassifier** implements a *multi-layer perceptron* (MLP) algorithm that trains using Backpropagation.

The leftmost layer, known as the input layer, consists of a set of neurons \{x_i | x_1, x_2, ..., x_m\} representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation w_1x_1 + w_2x_2 + ... + w_mx_m, followed by a non-linear activation function like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

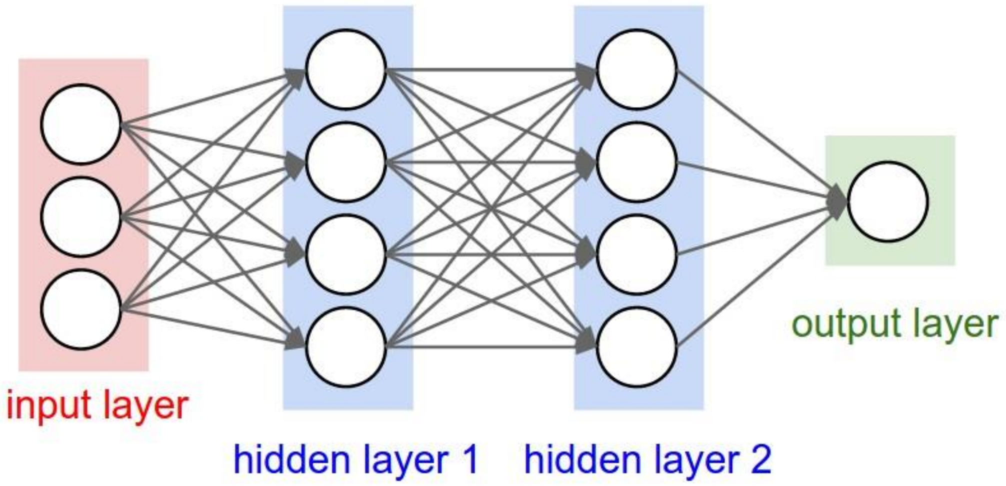
MLP trains on two arrays: array X of size (n_samples, n_features), which holds the training samples represented as floating point feature vectors; and array y of size (n_samples,), which holds the target values (class labels) for the training samples.

The module contains the public attributes coefs_ and intercepts_. coefs_ is a list of weight matrices, where weight matrix at index i represents the weights between layer i and layer i+1. intercepts_ is a list of bias vectors, where the vector at index i represents the bias values added to layer i+1.

The number of hidden layers can be changed by changing the *hidden_layer_sizes* attribute and adding a comma separated integer value in the **MLPClassifier** method to add neurons to the 2nd hidden layer.

The disadvantages of Multi-Layer Perceptron (MLP) include:

a) MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore, different random weight initializations can lead to different validation accuracy.
b) MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.
c) MLP is sensitive to feature scaling.



6. Three-Hidden Layer Fully Connected Multilayer Neural Network

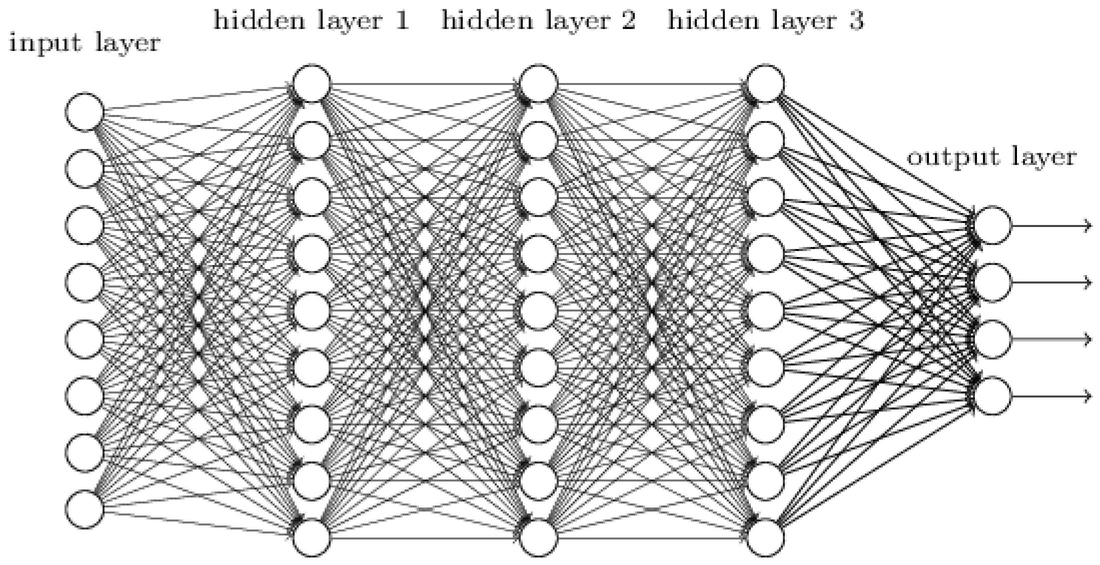
Class **MLPClassifier** implements a *multi-layer perceptron* (MLP) algorithm that trains using Backpropagation.

The leftmost layer, known as the input layer, consists of a set of neurons \{x_i | x_1, x_2, ..., x_m\} representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation w_1x_1 + w_2x_2 + ... + w_mx_m, followed by a non-linear activation function like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

MLP trains on two arrays: array X of size (n_samples, n_features), which holds the training samples represented as floating point feature vectors; and array y of size (n_samples,), which holds the target values (class labels) for the training samples.

The module contains the public attributes coefs_ and intercepts_. coefs_ is a list of weight matrices, where weight matrix at index i represents the weights between layer i and layer i+1. intercepts_ is a list of bias vectors, where the vector at index i represents the bias values added to layer i+1.

The number of hidden layers can be changed by changing the *hidden_layer_sizes* attribute.To add 3rd Hidden layer to the Network add another comma separated integer value in the **MLPClassifier** method of the 2-Hidden Layer Fully Connected Multilayer Neural Network.



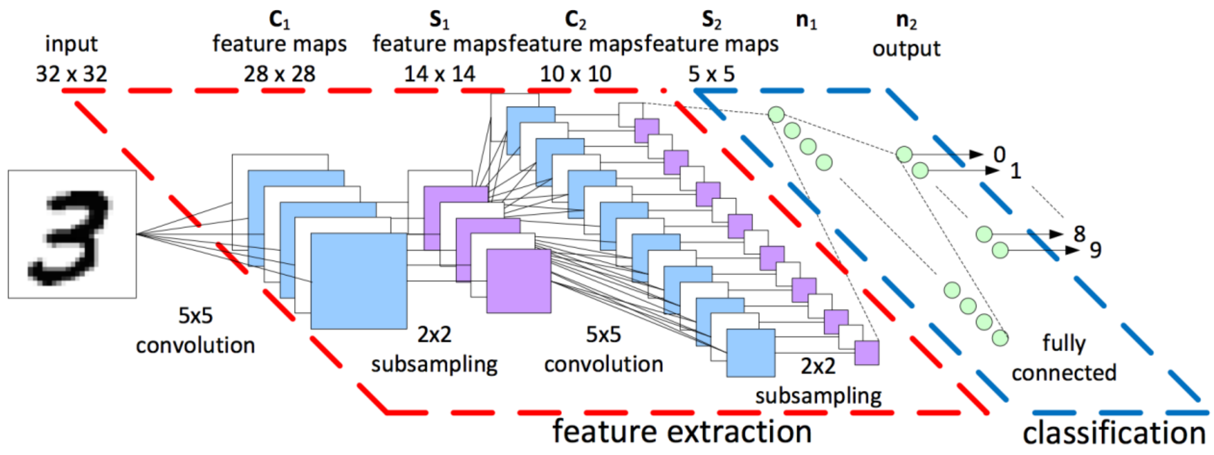
7. Convolutional Neural Network

Convolutional Neural Networks (CNN) is the way to go apply machine learning to image recognition. It has been sweeping the board in competitions for the last several years, but perhaps its first big success came in the late 90's when Yann LeCun used it to solve MNIST with 99.5% accuracy.

CNN implementation in this project is done using Keras, which is a user-friendly neural network library for python. It relies on either tensorflow or theano, so you should have these installed first. Keras is already available here in the kernel and on Amazon deep learning AMI.

A basic CNN consists in successions of convolutional layers (CL) and pooling layers (PL). Convolutional layers allow to extract several feature maps from the input images, while pooling layers perform some subsampling (i.e. dimensionality reduction) on the feature maps. Those successions of CLs and PLs correspond to a step of feature extraction. For image classification, the output layer is a fully connected NN layer with a number of units equal to the number of classes. The output layer activation function is a softmax, so that the ith output unit activation is consistent with the probability that the image belongs to class i. It's also common to see in a CNN, the CLs and PLs being combined with some rectification (non-linearities) and normalization layers that can drastically improve the classification accuracy (Jarrett et al. (2009))

The following scheme, taken from M. Peemen et al. (2011), represents a basic architecture with two convolution+subsampling steps plugged into a classifier:



Data

(describe the dataset)

The **MNIST** database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from *NIST*. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels because of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image to position this point at the center of the 28x28 field.

With some classification methods (particularly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass. If you do this kind of pre-processing, you should report it in your publications.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore, it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000-pattern training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint.

SD-1 contains 58,527-digit images written by 500 different writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in SD-1 is scrambled. Writer identities for SD-1 is available and we used this information to unscramble the writers. We then split SD-1 in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus, we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site. The full 60,000 sample training set is available.

This report presents the results of a comparison analysis of 7 different classifiers methods (SGD, K-NN, RBF, ANN (1-H &2-H) and CNN) trained on the MNIST dataset.

Simulations

(change parameters (number of epochs, activation functions, training set size etc) and observe the effect on the performance, provide plots & tables for both test and training classification errors, confusion matrices, etc)

A. Linear Classifier

1. The following 2 parameters of the **SGDClassifier** were changed to plot the accuracy:
   a) Number of epochs (max_iter)
   b) Learning rate (ETA)
2. For 3 different values of the *ETA*, the classifier run for 5, 10 & 20 epochs.
3. The accuracy & the mean squared error (MSE) was plotted as a function of the epochs
4. The link to the convolution matrix, plots and tables is attached here.

B. K-Nearest Neighbor Classifier

1. The following parameters of the **SVM_RBF** kernel were changed to plot the accuracy:
   a. Number of neighbors (k)
   b. Training Data size (set_size)
2. For 9 different values of the *k,* the classifier run for 30%, 70% & 100% of data size.
3. The accuracy & the mean squared error (MSE) was plotted as a function of the k & set_size
4. The link to the convolution matrix, plots and tables is attached here

C. Radial Basis Function Neural Network

1. The **SGDClassifier** was run by changing the following 2 parameters:
   a) Number of epochs (max_iter)
   b) Kernel coefficient for ‘rbf’(gamma)
2. Different float values were assigned to gamma. If gamma is ‘auto’ then 1/n_features will be used instead.
3. The accuracy & the mean squared error (MSE) was plotted as a function of the epochs
4. The link to the convolution matrix, plots and tables is attached here.

D. One-hidden layer fully connected multilayer neural network

1. The **MLPClassifier** was run by changing the following 2 parameters:
   c) Number of epochs (max_iter)
   d) Learning rate (ETA)
2. For 3 different values of the *ETA*, the classifier run for 5, 10 & 20 epochs.
3. The accuracy & the mean squared error (MSE) was plotted as a function of the epochs
4. The link to the convolution matrix, plots and tables is attached here.

Two-hidden layer fully connected multilayer neural network

1. The **MLPClassifier** was run by changing the following 2 parameters:
   e) Number of epochs (max_iter)
   f) Learning rate (ETA)
2. For 3 different values of the *ETA*, the classifier run for 5, 10 & 20 epochs.
3. The accuracy & the mean squared error (MSE) was plotted as a function of the epochs
4. The link to the convolution matrix, plots and tables is attached here.

E. Three-hidden layer fully connected multilayer neural network
1. The **MLPClassifier** was run by changing the following 2 parameters:
   g) Number of epochs (max_iter)
h) Learning rate (ETA)
2. For 3 different values of the *ETA*, the classifier run for 5, 10 & 20 epochs.
3. The accuracy & the mean squared error (MSE) was plotted as a function of the epochs
4. The link to the convolution matrix, plots and tables is attached here.

F. Convolutional Neural Network
1. The **CNN Classifier** was run by changing the following parameters:
   i) Number of epochs (max_iter)
2. The classifier was run for 5, 10 & 20 epochs.
3. The Annealer learning rate (ETA) was used for all the epoch iterations which is highest stable rate for CNN's.
4. The accuracy & the mean squared error (MSE) was plotted as a function of the epochs
5. The link to the convolution matrix, plots and tables is attached here.

Results

(Discuss your observations, do performance comparison of different classifiers)

The goal of this project is not to achieve the state of the art performance, rather to learn and compare the performance accuracy of 7 different machine learning algorithms namely:

1. Linear Classifier
2. K-Nearest Neighbor Classifier
3. Radial Basis Function Neural Network
4. One-Hidden Layer Fully Connected Multilayer Neural Network
5. Two-Hidden Layer Fully Connected Multilayer Neural Network
6. Two-Hidden Layer Fully Connected Multilayer Neural Network
7. Convolutional Neural Network

Although the solution isn't optimized for high accuracy, the results are quite good (see table below).

Table below shows some results in comparison of the above-mentioned models:

Method	**Accuracy**
Linear Classifier	0.882
K-Nearest Neighbors	0.963
SVM RBF	0.953
1-Hidden Layer NN	0.926
2-Hidden Layer NN	0.947
3-Hidden Layer NN	0.930
Convolutional Neural Network	0.991