Handwritten Digit Recognition

Project Paper

Zhirong Jian

Abstract: The goal of this project is to compare and analyze common learning algorithms by applying them to handwritten digit recognition problem. Specifically, SVM, KNN, ANN, decision tree and random forest learning algorithms are used, MNIST dataset are utilized as a benchmark. It also aims to provide a reference tutorial on these learning algorithms for beginners. It open sources the project codes in Github and provides overview on these algorithms in terms of both foundational mathematics and intuitive understanding.

jianzhirong@gmail.com
11/2014

Project link in Github:
https://github.com/LevinJ/ML-OCR

Content

1. Introduction

Automatic handwritten digit recognition is of great commercial and academic interests. Post offices use them to sort letters; banks use them to read personal checks. Academically, handwritten digit recognition is considered a "solved" problem and current research on this problem mainly focus on trying and benchmarking various new learning algorithms [1,2,3,4]. MNIST (LeCunet al., 1998) is the most widely used benchmark for isolated handwritten digit recognition [5]. The official MNIST dataset website maintains a list of learning algorithms that has been applied on this dataset and their corresponding error rate, including KNN, SVM, Neural nets, Convolutional nets and etc.

This work aims to accomplish two goals:
1) Compare and analyze the behavior of various learning algorithms on MNIST dataset.
2) Provide a reference tutorial on applying common learning algorithms to handwritten digit classification problems, it features:
   a) Fundamental math behind each learning algorithms, as well intuitive explanation of how they actually work, and why they work
   b) Working C++ source codes which demonstrate how to apply various common machine learning algorithms on a specific classification problem.
   c) All the training and testing can be done in seconds/minutes, instead of hours or days.

2. Dataset

For this project, I used MNIST handwritten digit dataset. MNIST consists of two datasets, one for training (60,000 images) and the other for testing (10,000 images). Each image is a gray scale image with 28 x 8 sizes. The pixel intensity ranges from 0 to 255, and the digit centered in the image. Fig 1 is some representative digits from the dataset. In our experiment, we used the 6000 training samples, and 1000 testing image in an attempt to reduce training/testing time to seconds/minutes, as opposed to hours or days if we use the whole dataset.
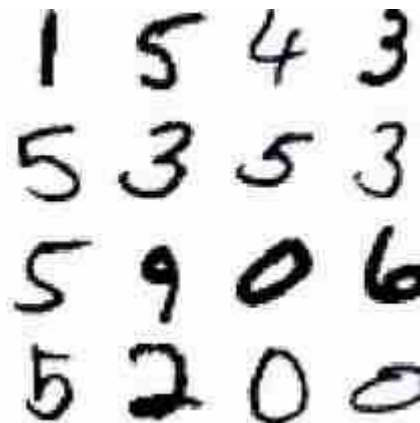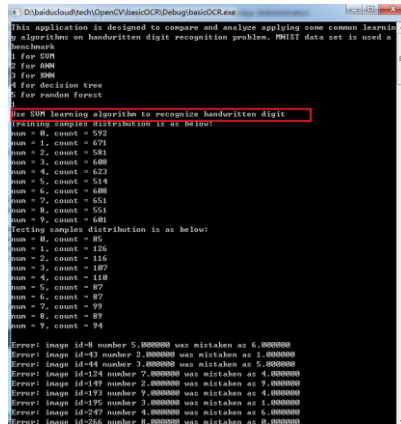

Fig 1. Image of representative digits and characters from the MNIST

3. Experiment result

All the implementations are done with OpenCV and C++. As the main program starts, it prompts users to select learning algorithm to use and then perform the classifications on the MNIST data set accordingly, as illustrated in Fig 2.

Fig2: Main program snapshot

The experiment results (error rate and performance) for the learning algorithms involved are demonstrated in Fig 3 and Fig 4:
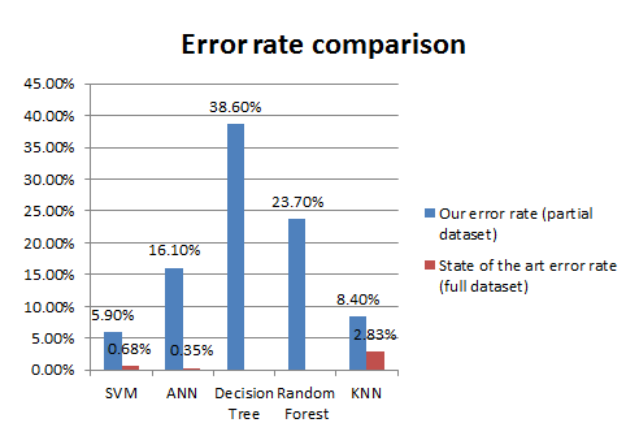

Fig 3. Error rate comparison for the learning algorithms. Our error rate was obtained using only part of the training and testing samples in MNIST dataset. And state of the art error state refers to the best records listed in the MNIST website. Decision tree and random forest has no error rate records in the MNIST website.
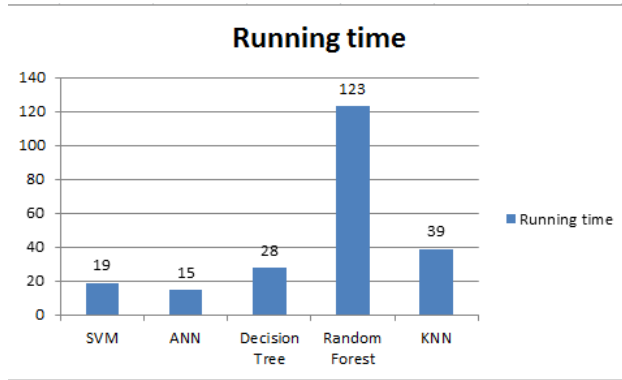
Fig 4. Running time comparison. Running time includes both training   time and test time, and the unit is second(s)

Overall, there are quite a few observations that merit further discussions.

1. Error rate
    SVM is clearly the winner here, which match with its reputation of being easy to use and yet powerful. And not very surprisingly, decision tree turns out to be having the poorest performance; this can be reasonably understood by intuition, as it attempts to make classification decisions by referencing the value of one pixel at a time.

2. Gap with state of the art records
    There is quite a gap between our result and state of the arts. There are at least two obvious causes. First, our experiment only uses part of the dataset (6000 training samples), and yet state of the art uses the full set (60000), this is a serious limitation for effective learning. In the MNIST dataset, ANN has the best records, while here it is not. This could somehow testify that neural network is indeed data hungry. Secondly, in the current implementations, learning parameters and configurations should have some room for fine tuning.

3. Run time
    Random forest is conspicuously slow in this regards, this could attribute to the fact that it has to construct 50 trees with depth of 5.

4. Important pixel positions
    One interesting attribute of decision tree learning is that it can output variable importance. In our case, this is the importance of pixel position. There are 784 (28x 28) positions in all. By intuition, the most important pixels should have the greatest variation among digit images and thus giving them more discriminative power, which is consistent with what we actually observed, they turn out to be around the center of the image. As shown in Fig 5.
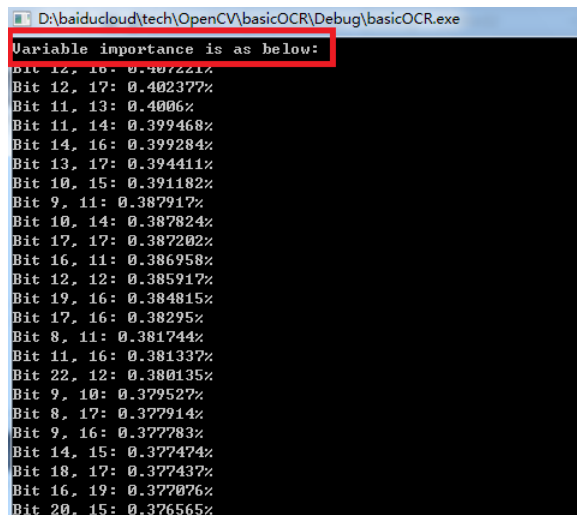
Fig 5: important pixel positions.

4. Improvement/Additional work

There can be a variety of improvement that can be done for the project in the future. This includes:

1. Add more algorithms, like CNN, logistic regression, normal bayes, Adaboost and etc
2. Fine tune the parameters and configurations for the learning algorithms currently used
3. Use the full data set on current implementations and see how they compare with state of the art.
4. Try some preprocessing before the classifications, like PCA，deskewing, normalizations and etc.

5. Conclusion

In this project, I applied various common learning algorithms to handwritten digit recognition problem, compared and analyzed their performance. SVM turns out be most effective algorithm among all the ones experimented. All the implementations are done with OpenCV and C++, and have been open sourced in Github. An overview which consists of both underlying math and intuitive understanding for these learning algorithms are also submitted in Github. The source codes and documentation presented in this project can serve as a reference tutorial for machine learning beginners.

6. Supplementary:

6.1. SVM

With the help of OpenCV, SVM algorithm was applied on the handwritten digit recognition.

6.1.1. SVM classifier function and parameter fitting

Here we have m number of training set, $\{x^{(i)}, y^{(i)}, i=1,...m\}$, for each training sample,$(x^{(i)}, y^{(i)})$, $x^{(i)}$ is a n dimensional vector $[x_1, ..., x_n]^T$, and $y^{(i)}$ is a scalar value which belong to the set {1, -1}.

These m number of $x^{(i)}$ forms a n dimensional space, the basic idea of SVM is to find a superplane $w^T x + b$ to divide this space two parts, where they belong to either 1 0r -1. Mathematically, the classifier function is as below:

$h_{w,b}(x) = g(w^T x + b)$

Here, $g(z) = 1$ if $z \geqslant 0$, and $g(z) = -1$ otherwise.

The idea of fitting parameter **w** and b is to make sure, among the training sample, the distance to the superplance is maximum. In another words, the superplance is as far away from the points of the two sides as possible and intuitively we get a superplane that classify the training samples well, and deduce it will the best chances to classify unseen testing points well.
Mathematically, we could use equal it as an optimization problem is as below:

$$\max{}_{,\mathbf{w},b}$$
$$\frac{(w^T x^{(i)} + b)}{||w||}$$
$$\text{s.t. } y_{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ . \ . \ . \ , m$$

$$\text{where } \gamma = \frac{(w^T x^{(i)} + b)}{||w||}$$

Due to the scaling property of **w**, b (we are bound to find numerous set of **w**,b which satisfy out optimization function requirement as long as we scale w,b), here we can focus on find just one particular set where $w^T x^{(i)} + b = 1$, as a result, we can transform the optimization as below:

$$\max{}_{,\mathbf{w},b}$$

$$\frac{1}{||w||}$$

s.t. $y_{(i)}(w^Tx^{(i)} + b) \geq \gamma$, $i = 1, \ldots, m$

where $\gamma = \dfrac{(w^Tx^{(i)} + b)}{||w||}$

And further into

$\min_{,w,b}$

$1/2 ||w||^2$

s.t. $y^{(i)}(w^Tx^{(i)} + b) \geq 1$, $i = 1, \ldots, m$

We've now transformed the problem into a form that can be efficiently solved. We can solve it with Lagrange duality. For a testing sample **x,** its classifier function can be denoted as below:

$w^Tx+b = \sum_{i=1}^{m} a_iy^{(i)}<x^{(i)}, x> + b$

So after all above derivation, we can classify which group (1 or -1) the testing sample x belong to by calculating above equations.

6.1.2. Using Kernel to map x to higher dimension

Above derivation has a key assumption that we are able to find superplane to divide the training sample space, while in reality this may not be true. This is how kernel function can help. Kernel function can help map existing feature x with n dimensional to higher dimensional space (like n^d) and then have it classified. Common kernels include Polynomial kernel, Radial basis function (RBF), Sigmoid kernel, Linear kernel (no mapping done, linear discrimination is done in the original feature space).

6.1.3. Address outlier

Despite the attempt of mapping original space into higher dimensional, there is always the possibility of some outlier which make it impossible to use a hyper plane to divide the space into the two, besides, taking all outliers strictly into consideration of parameter fitting will make SVM extremely vulnerable to noise. Based on this consideration, we can reformulate our optimization as below:

$\min_{,w,b}$

$1/2 ||w||^2 + C \sum_{i=1}^{m} \xi_i$

s.t. $y^{(i)}(w^Tx^{(i)} + b) \geq 1 - \xi_i$, $i = 1, \ldots, m$ $\xi_{i>0}, i = 1, \ldots, m$

ξ_i is called slack variable, which allows some x(i) 's distance less than 1, but if ξ_i arbitrary big, then any super plane will satisfy the optimization equation, so we add a cost parameter The

parameter C controls the relative weighting between the twin goals of making the $||w||^2$ small (which we saw earlier makes the distance large) and of ensuring that most examples have distance at least 1

6.1.4. Address multiple class classifications

Handwritten digit recognition is a multiclass classification problem in that we need to classify the samples into 10 different categories, but SVM discussed above is only binary classification. We can resolve this issue by reducing the multiple class classification problems into multiple binary classification problems, specifically, one versus all, or one versus one methods can be used.

6.2. ANN

6.2.1. The prediction part

Artificial neural network consists of input layer, output layer, one or more hidden layer. The example below represents a 3-layer perceptron with three inputs, two outputs, and the hidden layer including five neurons:[6]
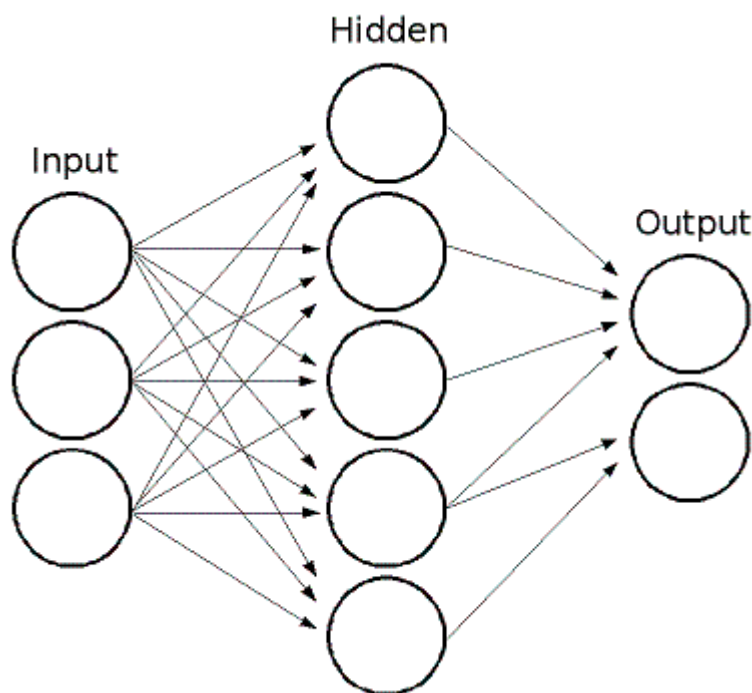
Each node in the hidden layer take the input from input layer, compute a response and pass it to next layer as an input. The computation is as below:

$$u_i = \sum_j (w_{i,j}^{n+1} * x_j) + w_{i,bias}^{n+1}$$

$$y_i = f(u_i)$$

The node would first compute the linear combination of its input, and then further transfer it through non-linear function to model non-linear relationship between input and output.

6.2.2. The training part

The training part of articiaficial neural network is essentially determine the w and f parameters/configurations. batch RPROP algorithm is commonly used to train neural network, more information can be obtained from below link.

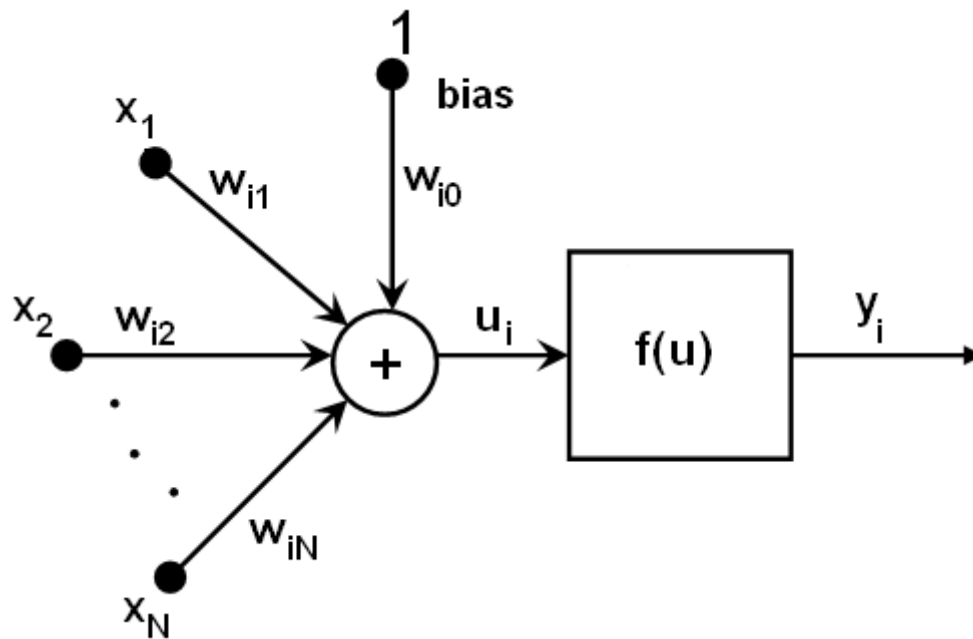http://en.wikipedia.org/wiki/Backpropagation. Wikipedia article about the back-propagation algorithm

6.2.3. Why neural network works

Below is actually a quote from Andrea Klein I encountered in Quora website. I think it gives a ver convincing answer[7].

1. In all the fuss about neurons and brains, let's not forget what neural network diagrams actually represent: each diagram describes a rule for transforming input into output - in other words, a function. The structure of the graph just enforces the order in which you evaluate the sub-functions associated with the nodes: you have to evaluate all the inputs to a node before you can get its output (otherwise, it doesn't make any sense).

Keeping this in mind, you can ask, what class of functions can be represented by rules of this kind? This is a mathematical question with a precise answer. Indeed, if you check out the Universal approximation theorem, you'll find that finite ANNs with even one hidden layer can approximate a huge class of functions*. This property turns out to be fairly robust relative to the exact choice of network: by this I mean, you don't have to have the choice of network exactly right in order to approximate a particular function - there may be a network which is ideal for that function, but you can still do quite well with a similar network (of sufficient complexity).

2. With a new appreciation for the computational power of neural networks, we can return to the problem of learning real-world patterns. Think about the sort of situation in which you'd want to use an ANN. You'd probably think to yourself, "I suspect there is a rule governing how this a value changes as a function of that input. However, the scenario is very complicated, and I'm not sure how to model it with an equation or an algorithm. Maybe an ANN could model this rule for me." For example, you might be interested in developing a self-driving car, which can take an image of the road as input and choose a value from [turn left, turn right] in response. It's reasonable to suppose there is generally a right answer, but it would be pretty hard to write down the exact rules as a function of the pixels in the image.

If you're right that the underlying rule exists, and the rule falls under the class of functions modeled by universal approximators (as defined in the proof of the universal approximation theorem), then there exist neural networks which can do an excellent job of approximating that rule. So, as long as you're happy with the details of the learning process, you need not be so mystified by the successful use of ANNs to model "patterns" - i.e. functions - in the real world.

6.3. KNN

KNN is among one of the simplest of all machine learning algorithm. The training examples are vectors in a multi-dimensional feature space, each with a class label. The training phase of the algorithm consists of only storing the feature vector and class label of the training samples.[8]

In the classification phase, we classify the test sample by finding its nearest K number of neighbor and have them voting for the class label of the test sample.

6.3.1. Distance metric

A commonly used distance metric for continuous vectors is Euclidean distance. For two points in the feature space, $P^{(i)}$ and $P^{(j)}$:

$P^{(i)} = [P_1^{(i)}, \ldots, P_n^{(i)}], \qquad P^{(j)} = [P_1^{(j)}, \ldots, P_n^{(j)}]$

$D(P^{(i)}, P^{(j)}) = \| P^{(i)} - P^{(j)} \| = sqrt((P_1^{(i)} - P_1^{(j)})^2 + (P_2^{(i)} - P_2^{(j)})^2 + \ldots + (P_n^{(i)} - P_n^{(j)})^2 \quad)$

For discreet vector, such as text classification, another metric that can be used, such as overlap

metric or Hamming distance.

The K-nearest neighbor classification performance can often be significantly improved through (supervised) metric learning. Popular algorithms are Neighbourhood components analysis and Large margin nearest neighbor. Supervised metric learning algorithms use the label information to learn a new metric or pseudo-metric.

6.3.2. Parameter K selection

The selection of optimum K value needs to be determined through experiments. If K value is too small, it can be extremely susceptible to accidental misclassified training sample or outlier, and on the other side, if K value is too large, KNN will lose it reasoning foundation and degenerate into finding out which group has the largest number in the training sample in the extreme case when K is equal to the number of training samples.

6.3.3. Curse of dimensionality

For higher dimensional vector space, KNN is susceptible to curse of dimensionality. When the dimensionality of space increases the volume of space increases so much that the available points in feature space becomes sparse and has greater chances of being dissimilar. As a result, distance metrics becomes ineffective in measuring the similarity between points.

The countermeasure of curse of dimensionality is to do feature extraction and reduce the dimensionality of feature space, usually principal component analysis (PCA), linear discriminant analysis (LDA), or canonical correlation analysis (CCA) techniques can be used to avoid this problem.

6.3.4. Voting mechanism

There are two kinds of voting, majority voting and weighted voting. For the majority voting, the algorithm votes the class label of the new test sample by a vote count on the class labels of its K number of nearest neighbor. For weighted voting, the distance between test sample and training samples are taken into account, the closer training samples will have a higher weight in the final voting.

6.4. Decision tree

Using elements in the feature vector of training examples as tree nodes, decision tree algorithm builds a most aggressive tree which allows for the quickest classification of training examples. Each node of the three contains two pieces of information, the element to be computed and the decision rule which determines the next sub node to go to. Once a leaf node is reached, the value assigned to this node is used as the output of the prediction procedure

6.4.1. Split node selection

The construction of decision tree is an iteration process, by choosing an element at each step that best spits the training examples into subsets. These generally measure the purity of the subset, the purer of its subset, the better. There are several metrics which can measure the purity of the subset: Information gain, gini impurity, and variance reduction.

6.4.2. Stop tree growth criteria

At each node the recursive procedure may stop (that is, stop splitting the node further) in one of the following cases:
● Depth of the constructed tree branch has reached the specified maximum value.
● Number of training samples in the node is less than the specified threshold when it is not statistically representative to split the node further.
● All the samples in the node belong to the same class or, in case of regression, the variation is too small.

6.4.3. Address Over fitting

When the training examples contain noise or random error, tree overfitting could occur. For example, a tree has been correctly created, and one of its leaf node has 10 positive class label. Now we deliberately change of the training example's class label to negative, decide tree would proceed to split the node as the node is not pure yet, this will inevitable split the node based on a particular element. If future test samples happen to have the same attributes as that of the noise sample, it will be misclassified. Tree pruning techniques is usually used to avoid overfitting.

6.4.4. Variable importance

Besides the prediction that is an obvious use of decision trees, the tree can be also used for various data analyses. One of the key properties of the constructed decision tree algorithms is an ability to compute the importance (relative decisive power) of each variable. For example, in a spam filter that uses a set of words occurred in the message as a feature vector, the variable importance rating can be used to determine the most "spam-indicating" words and thus help keep the dictionary size reasonable.

Importance of each variable is computed over all the splits on this variable in the tree, primary and surrogate ones. Thus, to compute variable importance correctly, the surrogate splits must be enabled in the training parameters, even if there is no missing data.

6.5. Random forest

Random forest is an ensemble of decision trees. Its classification works as follows: the random trees classifier takes the input feature vector, classifies it with every tree in the forest, and output the class labels that received the majority of votes. In case of regression, the classifier response is the average of the response over all the trees in the forest. [9]

Each tree in the forest grows as follows:

1. If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning

The random forest error rate depends on two things:

- The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob error rate a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

7. References

1. Ciresan, Claudiu Dan; Dan, Ueli Meier, Luca Maria Gambardella, and Juergen Schmidhuber (December 2010). "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition". Neural Computation 22 (12). doi:10.1162/NECO_a_00052. Retrieved 27 August 2013.
2. Wan, Li; Matthew Zeiler; Sixin Zhang; Yann LeCun; Rob Fergus (2013). "Regularization of Neural Network using DropConnect". International Conference on Machine Learning(ICML).
3. Ciresan, Dan Claudiu; Ueli Meier; Luca Maria Gambardella; Jürgen Schmidhuber (2011). "Convolutional neural network committees for handwritten character classification". 2011 International Conference on Document Analysis and Recognition (ICDAR): 1135–1139. doi:10.1109/ICDAR.2011.229. Retrieved 20 September 2013.
4. Kégl, Balázs; Róbert Busa-Fekete (2009). "Boosting products of base classifiers". Proceedings

of the 26th Annual International Conference on Machine Learning: 497–504. Retrieved 27 August 2013

5. MNIST digit database of handwritten digits. Yann LeCun, Corrina Cortes. http://yann.lecun.com/exdb/mnist/

6. OpenCV documentation. http://docs.opencv.org/modules/ml/doc/neural_networks.html

7. Quora discussion on neural networks.
    http://www.quora.com/Why-do-Artificial-Neural-Networks-work-the-way-they-do

8. KNN wiki page. http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

9. Random forest description.
   http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm