

Rahul K Gupta  
May 14, 2017

# **PROJECT REPORT**

Handwritten Number Recognition:  
Deep Learning using ConvNets

## Contents

1.	DEFINITION .....	3
	<i>Project Overview</i> .....	3
	<i>Problem Statement</i> .....	3
	<i>Metrics</i> .....	3
2.	ANALYSIS .....	4
	<i>Data Exploration</i> .....	4
	<i>Exploratory Visualization</i> .....	5
	<i>Algorithms and Techniques</i> .....	6
	<i>Benchmark</i> .....	7
3.	METHODOLOGY .....	8
	<i>Data Preprocessing</i> .....	8
	<i>Implementation</i> .....	9
4.	RESULTS .....	9
	<i>Model Evaluation and Validation</i> .....	9
	<i>Justification</i> .....	10
5.	CONCLUSION.....	10
	<i>Free-Form Visualization</i> .....	10
	<i>Reflection</i> .....	11
	<i>Improvement</i> .....	11
	<i>References</i> .....	12

# 1. Definition

## *Project Overview*

Computer vision is a field of computer science that works on enabling computers to see, identify and process images in the same way that human vision does, and then provide appropriate output.

Computer vision is closely linked with artificial intelligence, as the computer must interpret what it sees, and then perform appropriate analysis or act accordingly and provide useful results based on the observation.

MNIST ("Modified National Institute of Standards and Technology") is the de facto "hello world" dataset of computer vision. It classic dataset of handwritten images has served as the basis for benchmarking classification algorithms.

## *Problem Statement*

In this project we will develop a deep learning model on the MNIST handwritten number recognition in Python using the Keras deep learning library running Theano in the backend.

My motivation of choosing this project as my Capstone project is to use this as an opportunity and getting hands on with deep learning using neural network.

## *Metrics*

The most important feature of any computer vision application is that how accurately it can predict input provided to it. Accurate prediction of a handwritten number is our most important goal in this project. Based on the characteristics of the problem, we will use accuracy matrix to measure performance of a model or result in our project.

## 2. Analysis

Computer vision is like imparting human intelligence and instincts to a computer. In reality though, it is a difficult task to enable computers to recognize images of different objects. The difficulty of visual pattern recognition becomes apparent if you attempt to write a computer program to recognize handwritten digits. What seems easy when we do it ourselves suddenly becomes extremely difficult.

### *Data Exploration*

The dataset we are using in this project was constructed from a number of scanned document dataset available from the National Institute of Standards and Technology (NIST). This is where the name for the dataset comes from, as the Modified NIST or MNIST in short. It consists of images of handwritten digits like these:



It also includes labels for each image, telling us which digit it is. For example, the labels for the above images are 5, 0, 4, and 1.

In this project, we're going to train a model to look at images and predict what digits they are.

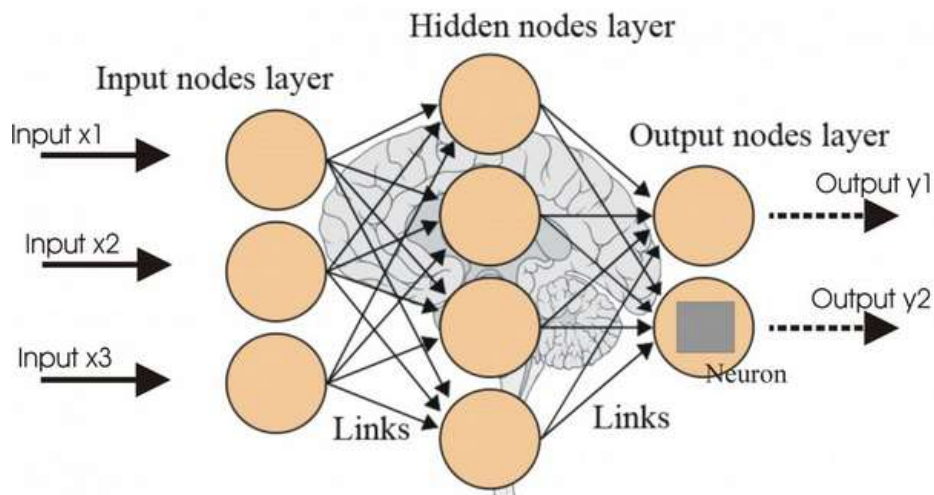
This is a multi-class classification problem as there are 10 digits (0 to 9) or 10 classes to predict. Results are reported using prediction error, which is nothing more than the inverted classification accuracy.

MNIST is a modified subset of two datasets collected by the U.S. National Institute of Standards and Technology. It contains 70,000 labeled images of handwritten digits. The MNIST database of handwritten digits, 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. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

### *Exploratory Visualization*

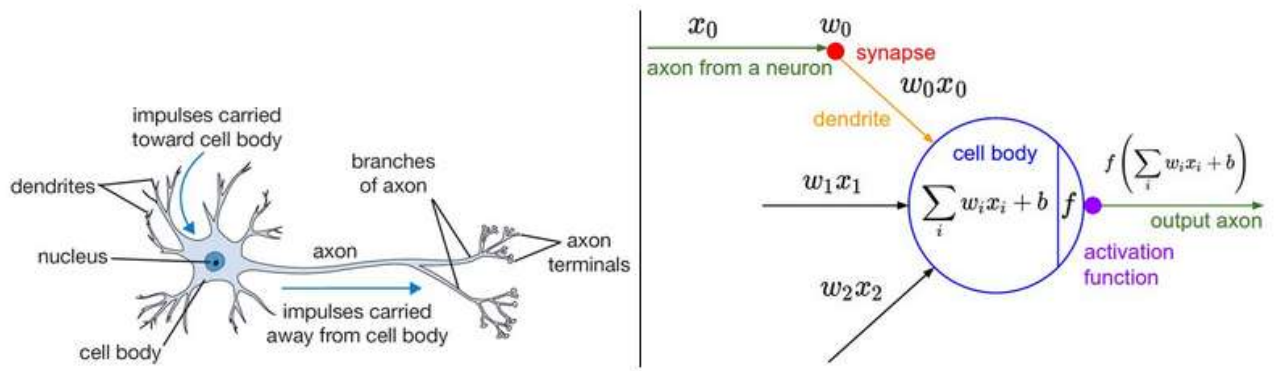
Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing.



*Basic structure of an Artificial Neural Network (ANN)*  
(Source: [FutureHumanEvolution.com](http://FutureHumanEvolution.com))

A neural network is a biologically-inspired programming paradigm which enables a computer to learn from observational data. This artificial intelligence technique mimics the operation of the human brain and comprises of densely interconnected computer processors working simultaneously. The key feature of neural networks is that they are programmed to 'learn' by sifting data repeatedly, looking for relationships to build mathematical models, and automatically correcting these models to refine them continuously.

Deep learning is a powerful set of techniques for learning and implementing the neural networks.



*Drawing of a biological neuron (left) and its mathematical model (right)*  
 (Source: [stanford.edu](https://stanford.edu))

In this project we are exploring Convolutional Neural Network neural networks with dropout.

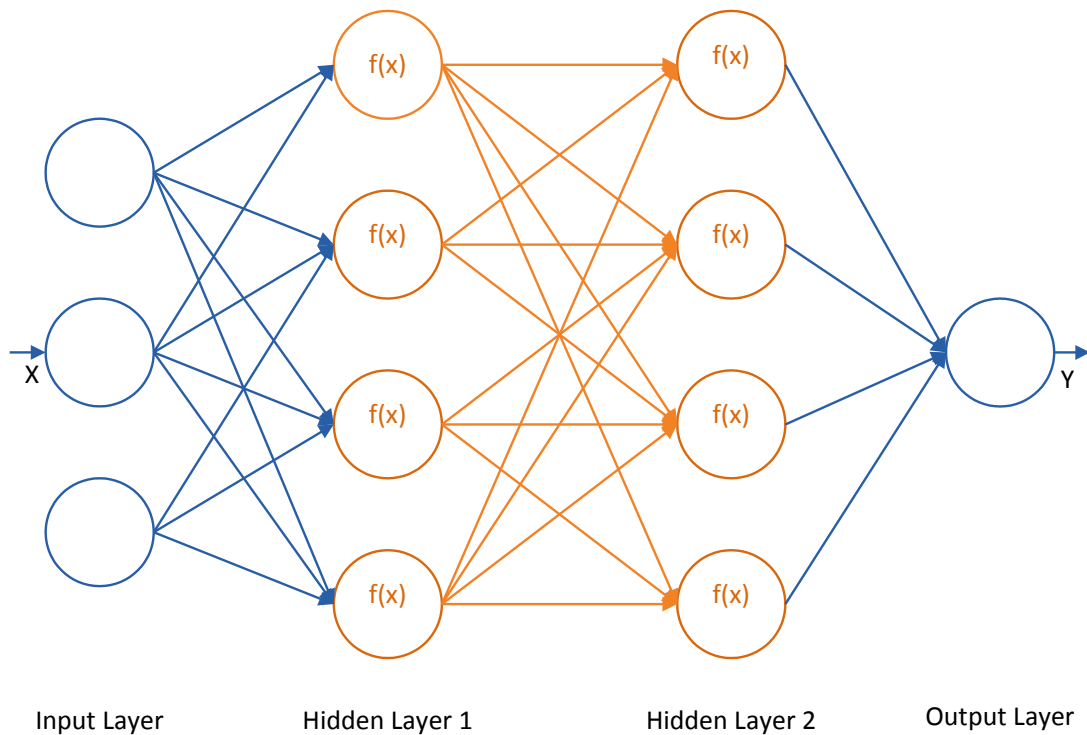
### *Algorithms and Techniques*

## Feed-forward Neural Networks

Feed-forward Neural Networks are the simple form of Artificial Neural Networks.

These networks have 3 types of layers: Input layer, hidden layer and output layer. In these networks, data moves from the input layer through the hidden nodes (if any) and to the output nodes.

Below is an example of a fully-connected feed-forward neural network with 2 hidden layers. "Fully-connected" means that each node is connected to all the nodes in the next layer.



## Activation Functions

Activation functions transform the weighted sum of inputs that goes into the artificial neurons. These functions should be non-linear to encode complex patterns of the data. The most popular activation functions are Sigmoid, Tanh and ReLU. ReLU is the most popular activation function in deep neural networks.

## Convolutional Neural Networks

Convolutional neural networks also referred as ConvNets or CNN are a special type of feed-forward networks. These models are designed to emulate the behavior of a visual cortex. CNNs perform very well on visual recognition tasks. CNNs have special layers called convolutional layers and pooling layers that allow the network to encode certain images properties. We will be using ConvNets approach in order to recognize the handwritten digit images provided in MNIST dataset.

ConvNet Layers:

- Convolutional Layer
- Pooling Layer
- Normalization Layer
- Fully-Connected Layer

### *Benchmark*

There are multiple historical models are available in public domain to compare results and provide benchmark. Following are few of them:

Classifier	Description	Preprocessing	Test Error	Reference
Linear classifiers	linear classifier (1-layer NN)	none	12%	LeCun et al. 1998
Nearest Neighbors	K-nearest-neighbors, Euclidean (L2)	none	3.09%	Kenneth Wilder, U. Chicago
Non Linear Classifiers	1000 RBF + linear classifier	none	3.6	LeCun et al. 1998
SVMs	SVM, Gaussian Kernel	none	1.4%	
Neural Nets	3-layer NN, 500+300 HU, softmax, cross entropy, weight decay	none	1.2%	Hinton, unpublished, 2005
Convolutional Nets	Large conv. net, random features	none	0.89%	Ranzato et al., CVPR 2007

(Source: [github.com](https://github.com))

We will be considering Convolutional Nets (Test Error: 0.89%) from above table as benchmark for this project.

## 3. Methodology

### *Data Preprocessing*

The Keras deep learning library provides a convenience method for loading the MNIST dataset. The training dataset is structured as a 3-dimensional array of instance, image width and image height. For a neural net model we must reduce the images down into a vector of pixels. In this case the 28×28 sized images will be 784 pixel input values. We can do this transform using the `reshape()` function on the NumPy array.

The pixel values are gray scale between 0 and 255. It is a good idea to perform some scaling of input values because the scale is well known and well behaved. We will normalize the pixel values to the range 0 and 1 by dividing each value by the maximum of 255.

Finally, the output variable is an integer from 0 to 9. Because this is a multi-class classification problem, we will use a one hot encoding of the class values, transforming the vector of class integers into a binary matrix. We will do this using the built-in `np_utils.to_categorical()` helper function in Keras.



We will define our model. The model is a simple neural network with one hidden layer with the same number of neurons as there are inputs (784). A rectifier activation function is used for the neurons in the hidden layer. A softmax activation function is used on the output layer to turn the outputs into probability-like values and allow one class of the 10 to be selected as the model's output prediction. Logarithmic loss is used as the loss function (called `categorical_crossentropy` in Keras) and the efficient ADAM gradient descent algorithm is used to learn the weights.

We can now fit and evaluate the model. The model is fit over 10 epochs with updates every 200 images.

Finally, the test dataset is used to evaluate the model.

## *Implementation*

On the top of above program structure discussed under aforementioned baseline neural net model, we will create a CNN for MNIST that will include Convolutional, Pooling and Dropout layers.

Below summarizes the Convolutional neural network architecture:

1. The first hidden layer is a convolutional layer called a Convolution2D.
2. Pooling layer that takes the max called MaxPooling2D.
3. Convolutional layer with feature maps.
4. Pooling layer.
5. A regularization layer using dropout to randomly exclude certain of neurons in the layer in order to reduce over fitting.
6. Next layer converts the 2D matrix data to a vector called Flatten. It allows the output to be processed by standard fully connected layers.
7. Fully connected layer of neurons and rectifier activation function.
8. Fully connected layer of neurons and rectifier activation.
9. The output layer of neurons for the classes and a softmax activation function to output probability-like predictions for each class.

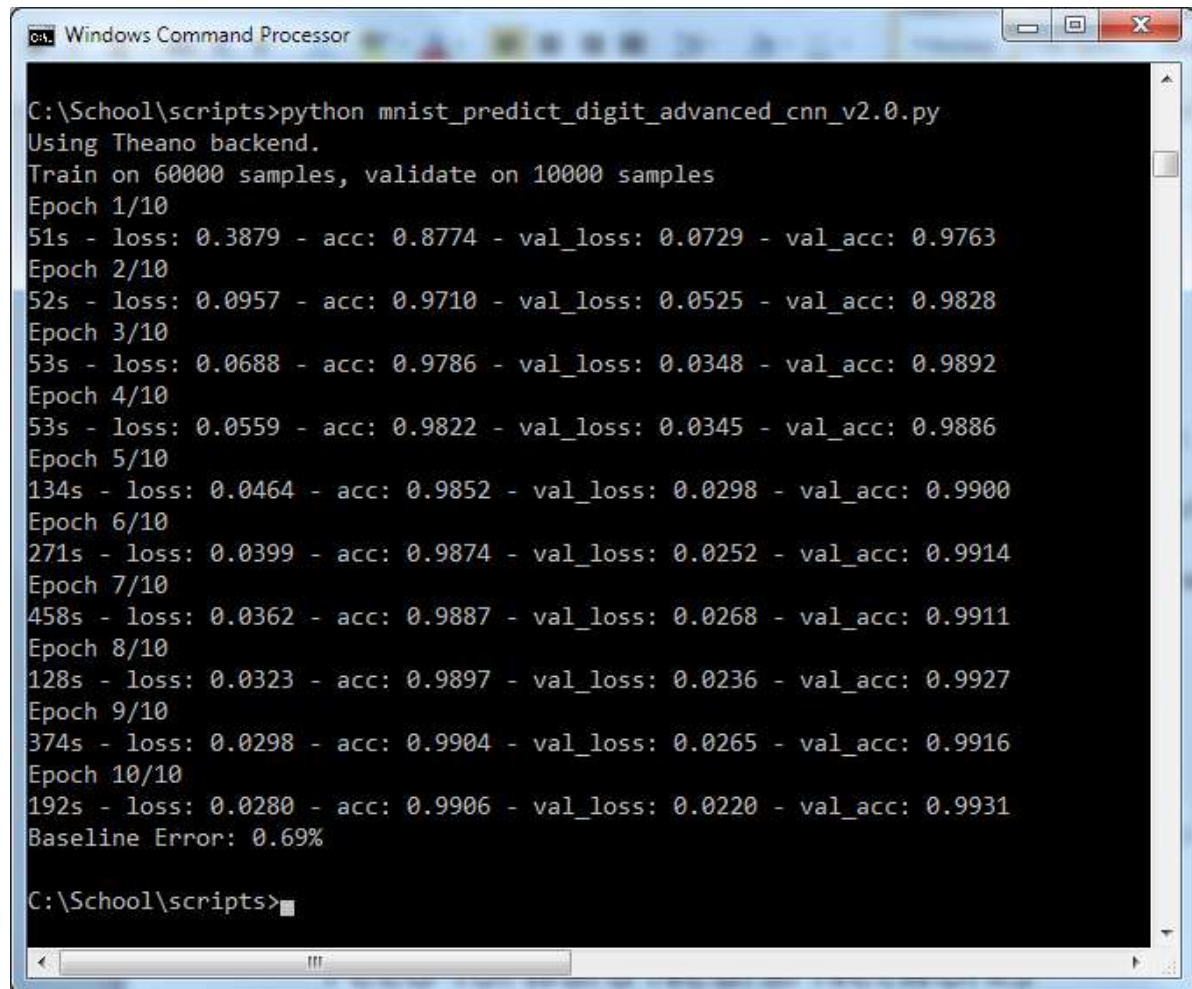
The model will be trained using logarithmic loss and the ADAM gradient descent algorithm.

And finally we will fit the model over 10 epochs with a batch size of 200.

# 4. Results

## *Model Evaluation and Validation*

Execution returns and displays the the loss value & metrics values for the model in test mode. On evaluate the model the network achieves an error rate of 0.69%, which is better than benchmark we have set earlier.



```
C:\School\scripts>python mnist_predict_digit_advanced_cnn_v2.0.py
Using Theano backend.
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
51s - loss: 0.3879 - acc: 0.8774 - val_loss: 0.0729 - val_acc: 0.9763
Epoch 2/10
52s - loss: 0.0957 - acc: 0.9710 - val_loss: 0.0525 - val_acc: 0.9828
Epoch 3/10
53s - loss: 0.0688 - acc: 0.9786 - val_loss: 0.0348 - val_acc: 0.9892
Epoch 4/10
53s - loss: 0.0559 - acc: 0.9822 - val_loss: 0.0345 - val_acc: 0.9886
Epoch 5/10
134s - loss: 0.0464 - acc: 0.9852 - val_loss: 0.0298 - val_acc: 0.9900
Epoch 6/10
271s - loss: 0.0399 - acc: 0.9874 - val_loss: 0.0252 - val_acc: 0.9914
Epoch 7/10
458s - loss: 0.0362 - acc: 0.9887 - val_loss: 0.0268 - val_acc: 0.9911
Epoch 8/10
128s - loss: 0.0323 - acc: 0.9897 - val_loss: 0.0236 - val_acc: 0.9927
Epoch 9/10
374s - loss: 0.0298 - acc: 0.9904 - val_loss: 0.0265 - val_acc: 0.9916
Epoch 10/10
192s - loss: 0.0280 - acc: 0.9906 - val_loss: 0.0220 - val_acc: 0.9931
Baseline Error: 0.69%

C:\School\scripts>
```

Test Loss Rate: 0.02%  
Test Accuracy Rate: 99.31%  
Test Error Rate: 0.69%

The model is robust as it finishes the execution in little more or less in same execution time and the result found after validating 10000 images are accurate around 99.31% of times. We can say the model can be trusted and lives the expectation

### *Justification*

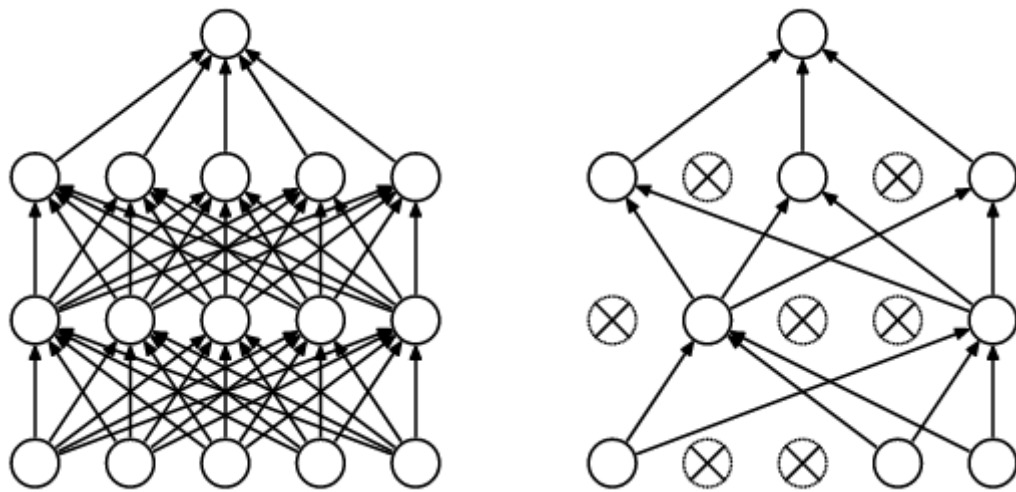
The result of above execution is better than compared to the result of benchmark chosen earlier (Test Error 0.089%) and so the solution is significant enough to have adequately solved the problem.

## 5. Conclusion

### *Free-Form Visualization*

The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random.

Dropout is a technique prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently.



*Standard Model*

*Dropout Model*

*(Source: Journal of ML Research by Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov)*

This is dropout Neural Net Model. The image in left is a standard neural net with two hidden layers. The image in right is an example of a thinned neural net produced by applying dropout to the network on the left. In the right image the units represented as crossed have been dropped.

## *Reflection*

This is my first project on the subject of Computer Vision as well as Deep Learning, so there was quite a bit of learning was involved here while doing this project. It was pretty interesting to see that there are range of standard libraries available including Keras, Tensarflow. As finding optimal hyperparameters is a daunting task I feel the most difficult aspect of the project was define a straight-of-art network model that produces best results at the same time have efficient train-and-test execution.

## *Improvement*

There are further improvements that could be made on the algorithms or techniques have been used in this project. Being my first deep learning and neural net, the algorithms and techniques I have used here are simple one and there is still a lot of margin creating more complex model and for parameter tuning that will give much better than what we have achieved here.

## References

As the solution design of this project is inspired by Jason's Deep Learning MOOC and examples provided by a Deep Learning library Keras author François Chollet, I would like to thank both of them providing great insight about deep learning using neural network.

Besides Udacity's Machine Learning Nanodegree engineering, following are some further references helped me understanding the overall Machine Learning fundamental concepts and Convolutional Neural Networks specific concepts used while doing this project:

- Machine Learning Fundamentals
  - CS 7641: Machine Learning
  - C Isbell, M Littman
  - College of Computing, Georgia Institute of Technology
  
  - Machine Learning With Big Data
  - M Nguyen, I Altintas
  - University of California, San Diego
- Convolutional Neural Networks
  - CS231n: Convolutional Neural Networks for Visual Recognition (Spring 2017)
  - L. Fei-Fei, J Johnson, S Yeung
  - Stanford Vision Lab, Stanford University
- Convolutional Neural Networks with Dropout
  - Dropout: A Simple Way to Prevent Neural Networks from Overfitting (published June 2014)
  - N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov
  - Department of Computer Science, University of Toronto