

Handwriting Recognition using different Machine Learning Techniques

- Darshan Tank - 150020012
- Chinmay Talegaonkar - 15D070046
- Kavya Prudhvi - 15D070014
- Tanya Gupta - 150040010



INDEX

- Introduction to the project
- CNN architecture
- Capsule networks
- Comparison of CNNs and Capsule networks
- Accuracies using all the algorithms - Confusion matrix
- Robustness of Capsule networks to affine transforms



INTRODUCTION

Using different ML algorithms like CNNs, Capsule networks, Logistic Regression, SVMs to predict digits using the MNIST training and test datasets.

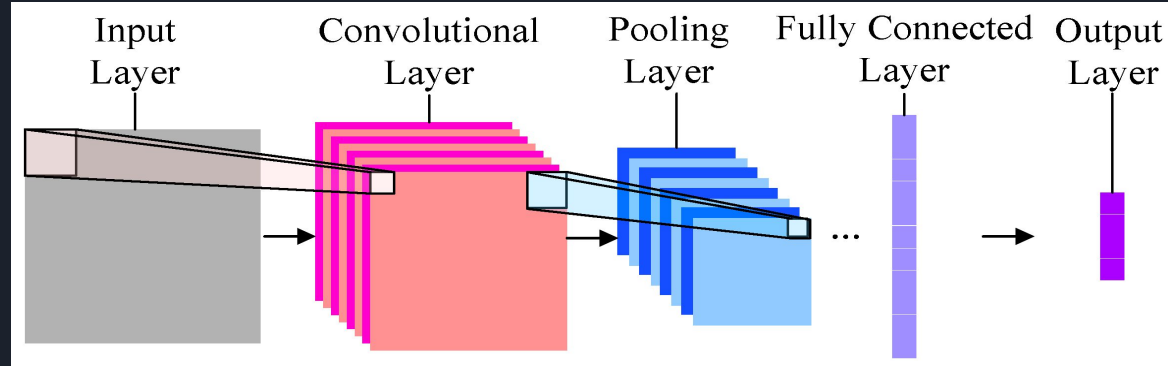
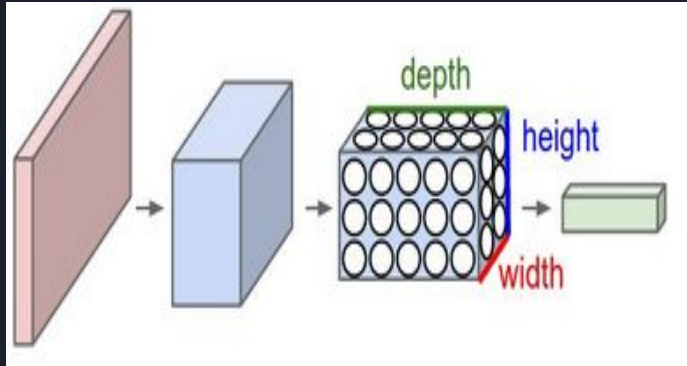
The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image

Also computed the confusion matrix to identify existence of errors

Convolutional neural networks

- Layers transform input 3D volume to output 3D volume using differentiable functions
- There are a few distinct types of layers (CONV, RELU, POOL, FC) each having different functions
- Higher layers abstract complicated features, lower layers act as spatial filters
- We have used the following architecture -:

CONV -> RELU -> MAX POOL -> CONV -> RELU -> MAXPOOL -> FC -> OUTPUT



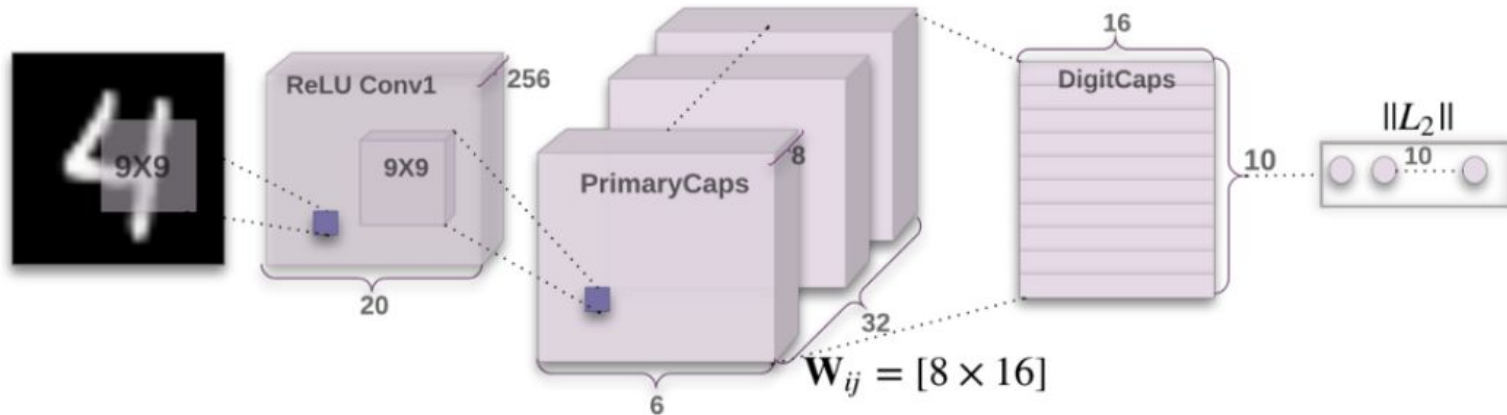
Capsule Networks Introduction

- They incorporate relative relationships between objects and it is represented numerically as a 4D pose matrix
- Work on the idea of dynamic routing to send information across capsules in different layers
- Capsules store and output information in the form of a vector, and uses dynamic routing to find the weights c_{ij}

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector(\mathbf{u}_i)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j i} = \mathbf{W}_{ij}\mathbf{u}_i$	—
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\ \mathbf{s}_j\ ^2}{1 + \ \mathbf{s}_j\ ^2} \frac{\mathbf{s}_j}{\ \mathbf{s}_j\ }$	$h_j = f(a_j)$
Output		vector(\mathbf{v}_j)	scalar(h_j)

Capsule network Architecture

- Comprises of an Encoder and decoder
- Encoder takes 28x28 image and converts it to 16 dimensional vector
- Decoder takes a 16 dimensional vector and converts it back to a 28x28 image using 2 CONV layers, and one FC layer





CNN Drawbacks

- Internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects
- Max pooling loses information
- Capsule nets are more robust to affine transforms. Trained a CNN and capsule network on MNIST dataset and then tested both on affNIST dataset



Confusion Matrix - Explained through an example

n = 165	Predicted: No	Predicted: Yes	
Actual: No	Tn =50	FP=10	60
Actual: Yes	Fn=5	Tp=100	105
	55	110	



Recall:

Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN)


Precision:

$$\text{Recall} = \frac{TP}{TP + FN}$$

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (small number of FP).

Precision is given by the relation:

$$\text{Precision} = \frac{TP}{TP + FP}$$



High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)



For the example confusion matrix

Classification Rate/Accuracy:

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN}) = (100+50) / (100+5+10+50) = 0.90$$

Recall: Recall gives us an idea about when it's actually yes, how often does it predict yes.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN}) = 100 / (100+5) = 0.95$$

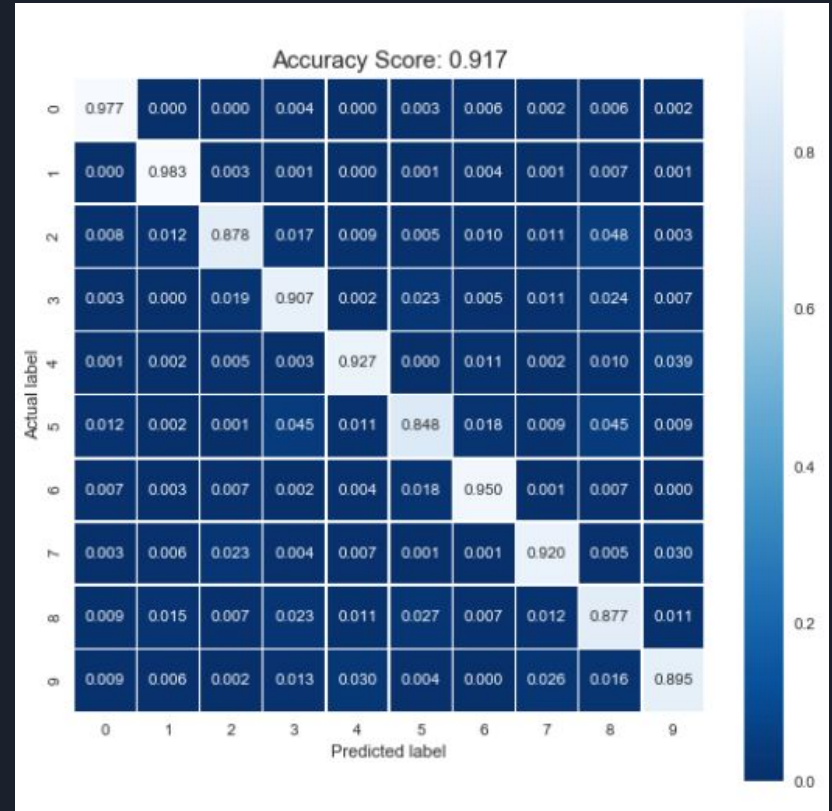
Precision: Precision tells us about when it predicts yes, how often is it correct.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP}) = 100 / (100+10) = 0.91$$

Logistic Regression

Used the scikit learn library to import already existing functions related to logistic regression and the seaborn for plotting

ACCURACY: 91.7%





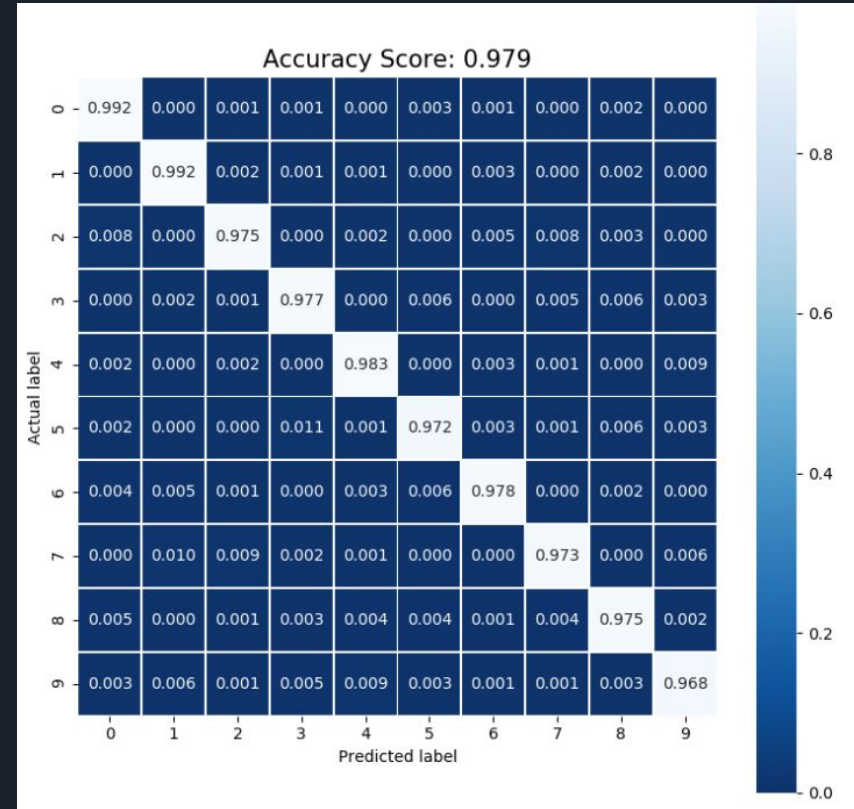
Logistic Regression Drawbacks

- It can only have a linear predictor and hence cannot classify the XOR type data at all.
- Needs a very large set of training data to have a stable and decent accuracy.

Support Vector Machines

Used the scikit learn library to import already existing functions related to svms and the pylab library for plotting (3 degree polynomial)


ACCURACY: 97.87%





Drawbacks: SVMs

- The most serious problem with SVMs is the high algorithmic complexity and extensive memory requirements of the required quadratic programming in large-scale tasks
- Abysmally slow in test phase
- Important practical question that is not entirely solved, is the selection of the kernel function parameters



Accuracy Achieved on Affine transformed MNIST

We trained our CNN and Capsule Nets on MNIST Data and tested it on Affine transformed MNIST

Following Accuracies were obtained:

CNN: Trained and Tested on MNIST: 98%

Trained on MNIST and Tested on AffNIST: 18%

Capsule Nets: Trained and Tested on MNIST: 99.2%

Trained on MNIST and Tested on AffNIST: 30%