Classification of Digits Obfuscated by Rotation, Texture and Emboss

COMP 598 – Miniproject 3 – Team Numberwangers

Alireza Saberi
260358148

Azar Zandifar
260560394

Ehsan Kia
260481162

ABSTRACT

In this report, authors have approached "hard digit classification" task using different classifiers and preprocessing steps. The raw dataset is a modified version of the popular MNIST digit classification dataset. The performances of logistic regression, feed forward neural network, linear SVM and convolutional neural net have been compared. Convolutional neural network shows the best accuracy 93.69% (submitted on Kaggle), followed by SVM (38.1%), Feed forward neural network (35.37%) and logistic regression (25.58%), respectively.

1. INTRODUCTION

In machine learning, classification is the problem of assigning a new observation to a category that it belongs. In this project, we tackle the problem of classifying hand-written digits (0-9) images from modified Mixed National Institute of Standards and Technology (MNIST) dataset [10]. The MNIST dataset is a subset of a larger dataset of National Institute of Standards and Technology (NIST) that is created by remixing the NIST training dataset and NIST test set together. The training set of MNIST consists of the first half of NIST training dataset and test set while the testing dataset of MNIST contains the second half of NIST training dataset and NIST test set respectively. The MNIST dataset contains a total of 70,000 images; 60,000 training image along with 10,000 test images. The size of each image is normalized to 28 x 28 pixels and digits are centered.

In this project, the several modifications have been done to the MNIST e.g. embossing, random angle rotation, rescaling to 48×48 pixels, overlaying a random texture at the background to make the project more challenging. The new dataset includes 50,000 training examples and 20,000 test examples.

We attempted logistic regression as our baseline learner, fully connected Feed-forward Neural Network, linear Support Vector Machine (SVM), and Convolutional Neural Network (CNN) to tackle this classification. Considering the extra set of obfuscation added to MNIST dataset, we expect substantially reduced accuracy in comparison with the original dataset.

The rest of the paper is organized as follows: Section 2 contains the preprocessing methods that we tried to use. Feature selection and extraction method is described in Section 3 A brief overview algorithm selection is presented in Section 4. The optimization and parameter selection are in Section 5 and 6. The Testing and validation are presented in Section 7. And Finally Section 8 contains the discussion and conclusion part

2. PREPROCESSING

Since we have a data set of noisy images, one way to help the classifiers is to run image processing algorithms on the raw images to clean them up, removing the irrelevant information. We experimented with a couple different transformation in this study, but none of them ended up being used.

2.1 Shape from Shading

The idea of shape from shading is to estimate the surface normal of an object from the shading in the image, knowing the lighting direction. The original algorithm for solving this problem, outlined by Horn 1989 [4], uses partial differential equations to solve this problem.

In our case, we have an embossing filter applied to the digits in our data set, with the same lighting setting in every image. This means that in every image, one side of the digit will be bright and the other will be dark. This will make it much harder for the classifiers, and makes them less tolerant to rotations.

In theory, if we can extract the shape of the digit, the lighting direction won't be an issue anymore. Furthermore, with some thresholding, we might be able to remove some of the smaller bumps in the image introduced by the textured backgrounds.

In practice, these shape for shading algorithms aren't very stable, and the noise in the image makes it very difficult for them properly converge to the correct surface normal. They are also fairly slow, which makes it unrealistic run on our huge data set of 70000 images.

2.2 Image rotations

This isn't exactly image preprocessing, but one idea to help classifier training is to add extra samples by generating 3 extra 90 degree rotations of each of the examples in our dataset. In our study, we found that this doesn't help, and we suspect it's due to the embossing filter applied to the image. As mentioned above, the image has a specific lighting direction, making one side of the digits brighter than the other. By rotating the samples, we confuse the classifier by introducing digits with light coming from different directions.

3. FEATURE SELECTION

While raw image pixels as features is a good starting point, there are many advanced features that can be extract from images which summarize local information better. In this study, we experiment with a few different ones and used cross-validation to see how much they improved our results.

3.1 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) are a powerful tool in computer vision for object detection. In essence, it counts local occurrences of gradient orientations in the image [3].

There exists many variations on HOG features, developed for specific applications. One such derivative we look at is SPHOG feature (spatial pyramid histogram of oriented gradients), which has been used for digit classification. It was shown to get an error rate of 0.79% on the original MNIST dataset when used with an SVM classifier [12].

3.2 Local Binary Patterns

LBP, also known as Local Binary Patterns, is a feature used to describe the local structure of images [15]. It is often used for texture classification tasks, along with SVM classifiers. It has also been shown to give improved results when combined with HOG features [17].

We compute LBP feature vectors for all images in our dataset using `Matlab` and use it along with the HOG features, appending everything together.

3.3 Feature combination

Various combinations of features were tested using SVM as the classifier. As we can see in the table below, each feature set slightly improves the accuracy.

Feature sets	Accuracy
pixels	26.9%
pixels & SPHOG	27.1%
pixels & LBP	28.4%
pixels & LBP & SPHOG	28.6%

4. ALGORITHM SELECTION

4.1 Logistic Regression

Linear classifiers are the simplest classifier which we start with. The classification output is considered as a linear function of the features [9]. However, a linear classifier is not able to model the decision surface between the classes that are not linearly separable. In this study, we used logistic regression classifier, as a base-line method to compare with other more complex classifiers. Logistic regression can be categorized as a discriminative learning algorithm. Given the complexities of our dataset, modeling the conditional probability distribution of classes given features is far more feasible than modeling the joint probability of features and, which is the case in generative learning models. Therefore, logistic regression is used as the basic linear model for approaching this problem.

The MNIST data set shows an error rate of 12 percent with a linear classifier [9]. However, considering the extra set of obfuscation added to our dataset (e.g. textures, rotations and embossing), we expect substantially reduced accuracy in comparison with the original dataset [8].

4.2 Feed-forward Neural Network

Multi-layer neural networks trained by gradient descent are known to have the ability to learn complex, high dimensional and nonlinear models [9]. These characteristics make them one of the most suitable candidates for our digit recognition task. In this study, we tested a multilayer fully connected neural network, trained by back propagation, as one of the simplest implementations of the neural network model. Although theoretical results have shown that any function can be approximated using only one hidden layer [14], some studies showed that adding more hidden layers to the neural net sometimes have positive impact on the total performance of the network [9]. Therefore, we tried to reach the optimal network structure considering both single and double hidden layers.

The original MNIST data set shows 4.7 percent error with a two layer feed-forward neural network [9]. However, as it was mentioned before, considering the complexities added to our specific problem, we again expect substantially reduced accuracy in comparison.

4.3 Linear SVM

We used `scikit-learn` [16] library's implementation of SVM (Support Vector Machine) with a linear kernel. Given data belonging to one of two classes, this classifier tries to find a hyperplane which separates the data. Since there might be many of such hyperplanes, SVM tries to find the one that results in the largest margin. This is done by the following linear optimization:

$$\arg\min_{w,b} \|w\| \quad \text{subject to } y_i(w \cdot x_i - b) \geq 1$$

where w represents the normal vector of the hyperplane separating the two classes. There are many ways of handling more than two classes, such as a "one-vs-all" or a "one-vs-one" method. The `scikit-learn` algorithm we are using implements the "one-vs-one" approach described by Knerr et al. [6]

We don't really expect to get very good results here either as this dataset is likely not linearly separable. A richer feature set will definitely help reduce the error rate though. On the original MNIST data set, linear SVM has been shown to get 11% error rate on the raw pixels, and 3.4% with PHOG (Pyramid Histogram of Oriented Gradients) features [12].

4.4 Convolutional Neural Network

Convolutional neural networks are a type of feed-forward neural network inspired by biological processes [13]. Each layer of the network uses filters to look at local portions of the images, which is why it gives much better results in image classification. In the literature for the MNIST dataset, Convolutional Neural Networks actually achieve the best results, with an error rates as low as 0.23% [2]. In our tests, we used Alex Krizhevsky's `cuda-convnet` library [7] to run the computation on a Titan GPU. This allowed us to run this computationally intensive problem in reasonable times. We expect this classifier to perform the best on our dataset.

5. OPTIMIZATION

5.1 Principal Component Analysis

Since the dataset is really large, and each example has thousands of features, we had to reduce the number of features to be able to run the classifiers in a reasonable amount of that. To that end, we use Principal Component Analysis (PCA) to transform the data and pick components having the largest variance. Using SVM and our feature set (pixels, SPHOG and LBP), we cross-validate the optimal number of components, which was found to be 64 as seen in Figure 1.

5.2 Logistic Regression

Logistic regression is implemented using Newton-Rapshon gradient descent. The Newton-Raphson method is one of the most powerful optimization techniques which have been used successfully with logistic regressions cross entropy minimization [1]. In order to avoid over-fitting, an L2-norm regularization term has been added to the cost function. We used a "one-against-all" strategy for making the logistic regression work on a multi-class problem. Although a logistic

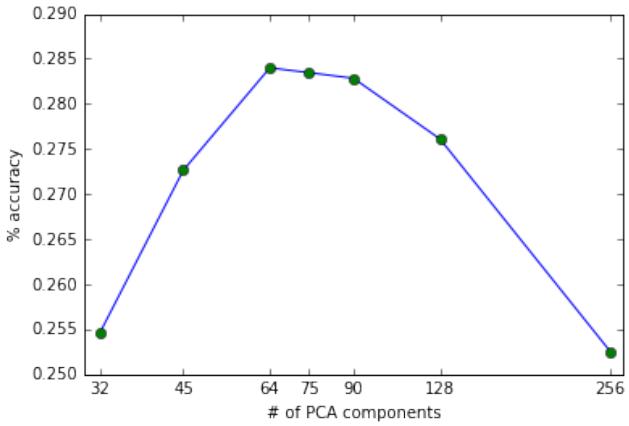

Figure 1: Convolutional Neural Network - Set of computed filters for the first layer of the network

regression can be considered as a neural network without any hidden layer, cost function for the fully connected neural network is the sum of square differences, whereas the cross entropy is minimized in the case of logistic regression.

5.3 Feed-forward Neural Network

The neural network is trained using the back propagation gradient descent. The cost function is the RMS error. In this project we used stochastic gradient descent. Our experiment showed that, for this problem, stochastic gradient descent works better than batch or semi-batch strategies. In the stochastic strategy the network weights are updated each time a new sample reaches the network. An epoch, in the training procedure, means passing through all the samples in the training set once.

A heuristic approach has been taken toward assigning the learning rate: the learning-rate will be divided by an increasing integer each time the net observes 1000 samples. However, this decreasing rate will stop when the learning rate reaches the minimum amount of 0.1. That is, the algorithm begins with a large learning to converge faster and avoid local minima, and as we progress, the learning rate will be decreased to avoid divergence or oscillation around the expected minimum. As for the initial value of learning rate, it is assigned by cross validation.

5.4 Linear SVM

There wasn't any optimization required for the Linear SVM classifier, as the `scikit-learn` library already provides great default values.

5.5 Convolutional Neural Network

Convolutional Neural Networks are known for requiring very little parameter engineering, and performing well on the raw images. To start, we tested our data with the basic network architecture that the library came with, consisting of three convolution and pooling layers, followed by a fully connected and softmax layer. Right off the bat, the GPU based library was able to compute 40 epochs through the dataset in 15 minutes, and achieve a validation accuracy of around 85%.

After a bit of research and experimenting, we found that we received the best results by training the net until it stabilizes (around 70 epochs), then lowering all the learning rates by a factor of ten and training for another 10 epochs. We repeat that last part twice more to make sure we have fully converged to the current minima. As we can see in Figure 3,

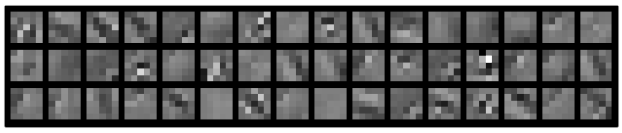

Figure 2: Convolutional Neural Network - Set of computed filters for the first layer of the network

this final step really helped get those few extra percent of accuracy.

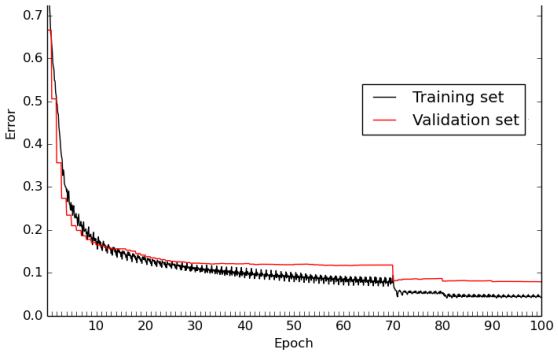

Figure 3: Convolutional Neural Network - Error rate for the training and validation set through the epochs

6. PARAMETER SELECTION

6.1 Logistic Regression

The following experiments were done using the 64 principal components drawn from combined raw pixel data, HOG [3] and LBP features [15], as running the classifier on the full feature set is too computationally expensive. A simple internal cross validation is done to chose the best set of weights for the each logistic regression. That is to say, at the beginning of the optimization procedure, 10 percent of the training data is put aside as a hypothetical test set. The accuracy on this hypothetical is measured using each set of parameters in the optimization procedure. The weights with the maximum corresponding accuracy are set as the final weights for the algorithm.

The regularization hyper parameter (lambda) has been chosen based on a four-fold cross validation. Each fold contains 10000 samples of the data and the final performance has been tested using the remaining 10000 samples. The results show that smaller lambda values give higher prediction accuracy. For the larger regularization coefficient, we obviously expect larger bias and smaller variance. Therefore, it is totally expected that the regularization factor should be set to the smallest value that avoids the algorithm over-fitting. Figure 4 shows the result of cross validation for the logistic regression on different values of lambda. The results are the mean accuracy for each of the folds of our four-fold cross validation. The lambda parameter has been set to 0.1 for the testing results.

6.2 Feed-forward Neural Network

Based on similar reason as the previous section, all the investigations in this section are done using 64 principal components drawn from the raw pixel data combined with HOG [3] and LBP [15] features.

In the optimization procedure, we hypothesize that with increasing the number of epochs, the training accuracy increases constantly and then plateaus, while validation accuracy rises and then falls where the algorithm begin to over

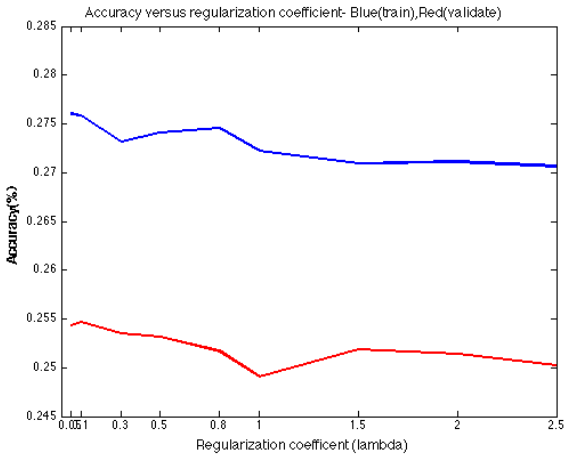

Figure 4: Logistic Regression - Train and validation accuracy for different values of lambda (regularization factor) in blue and red respectively

fit to the training ser. However, considering our limited time and computational power and big size of the data, we could only go through fifteen epochs on the whole training data, for which both the train and validation accuracy tend to rise. Nevertheless, as the number of epochs increases the difference between test and train accuracy tends to rise, which was expected (Figure 5).

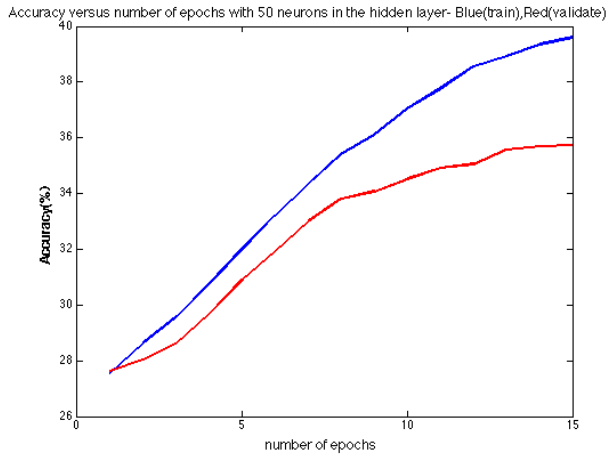

Figure 5: Feed-Forward Neural Network - Train and validation accuracy for different number of epochs in blue and red respectively

Since the training set is sufficiently large, we hypothesize that the training set can be considered as a fair representative of the whole data distribution. Furthermore, considering the large amount of training sample, the computational time is prohibitively long. Therefore, We decided to avoid k-fold cross validation in our study. We train our network using 30000 samples of the training set, validate the network using another 10000 samples, and at last test the final network using the remaining 10000 samples.

The structure of the neural net was determined using cross validation. The neuron count in the first layer is determined by the number of features (input layer), and the number of neurons in the output layer is equal to the number of classes. Therefore, the only parameter to optimize is the number of hidden layers and their corresponding neuron counts. First, we begin by increasing the number of neurons in the first hidden layer from 5 to 60 neurons, with a step size of 5 neurons. Considering our 64 features, we decided to not to go further than 60 neurons in the first hidden layer. The cross validation is done with fifteen epochs and the initial value of learning rate is set to two. Continuous rising pattern have been observed by increasing the number of neurons. However, only slight efficiency difference was observed between 50 neurons and larger numbers such as 55 and 60 (Figure 6). Therefore, we fixed our first hidden layer units to 50 to avoid additional computation burden. Then, another layer is added to the network with both 5 and 10 neurons. The results show that adding the second layer have no positive impact on the performance of the network compared to only single layer network: the accuracy on the validation set falls from 35.74 percent for the single layer to 31.80 and 33.73 percent for 5 and 10 units in the second layer, respectively. The gradient descent, which we use to train our neural network, is not powerful enough to support larger networks, which call for optimization on numerous weights. Therefore, it was expected that the performance of the smaller beat the larger ones for our implemented neural network.

The final parameter, which was set by cross validation, was the learning rate initialization value. Although, the learning rate changes adaptively, the initial value is assigned using cross validation(Figure 7).

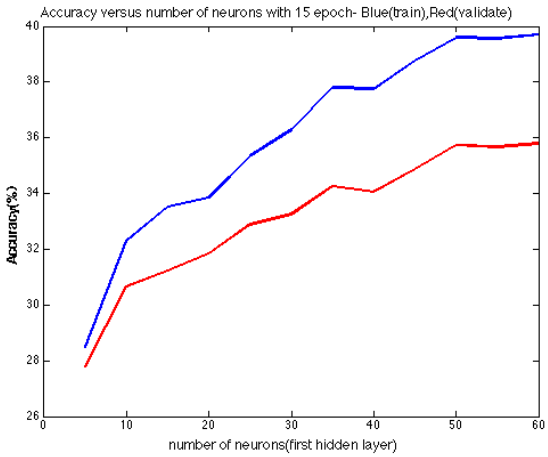

Figure 6: Feed-Forward Neural Network - Train and validation accuracy for different number of neurons in the first hidden layer in blue and red respectively

6.3 Linear SVM

We tried using "L1" penalty, and different multi-class strategies, but nothing performed better than the original parameters. The only hyper-parameter to optimize here is the number of PCA components, but as mentioned before, we had used SVM to optimize that parameter and found the optimal to be 64 components.

6.4 Convolutional Neural Network

There was very little parameter selection involved with the convolutional neural network. The library we used came with all the parameters set to reasonable values, and given the computational intensity of neural networks and the cheer number of parameters, it wasn't really possible to run cross validation on the net.

We did try increasing the number of filters computed at each layer, which doubled the computational time, but also decreased the error rate by another 0.4%. Other than that,

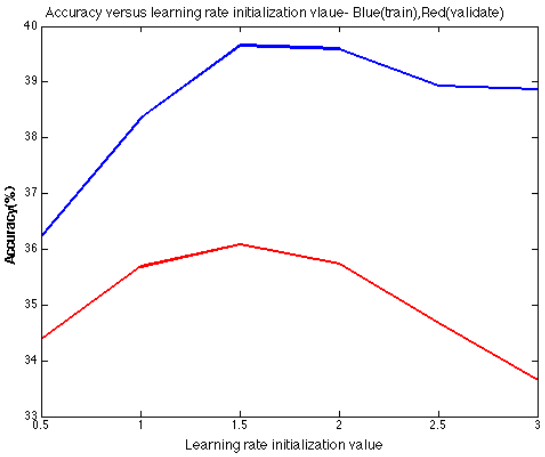

Figure 7: Feed-Forward Neural Network - Train and validation accuracy for different learning rate initialization value in blue and red respectively

we didn't really observe any big differences playing around with the other parameters.

7. TESTING AND VALIDATION

7.1 Logistic Regression

Using the last 10000 samples we have reached accuracy 0f 25.58 percent for the baseline method. Figure 8 shows the confusion matrix for the logistic regression classifier, the matrix has been normalized row-wise to account for between class variability. The results show that the algorithm achieves its best performance for digits 0, and 1. However, other classes show very poor results, number "2" has the worst performance, number "2" is mainly mistaken by "0" .

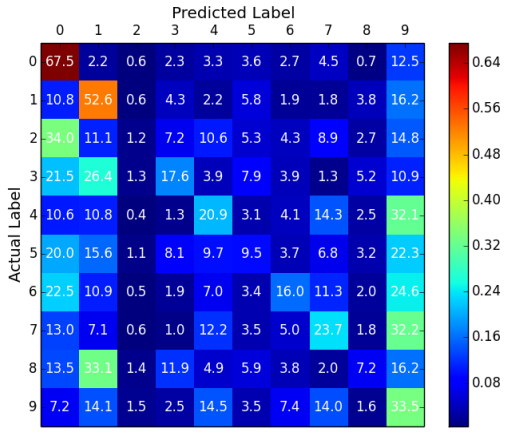

Figure 8: Logistic Regression - Confusion Matrix

7.2 Feed-forward Neural Network

The network has been test on the remainig 10000 samples. The optimum network has 50 hidden neuron in the first layer, and learning rate initializes by 1.5. The network is trained using 40000 and tested using laste 10000 samples. Accuracy on the test set reaches 35.37 % which is significantly higher than the accuracy reported for the linear classifier. Confusion matrix shows that similar to the pattern observed in the logistic regression, the number "2" is the most difficult digit to categorize for our feed-forward neural network (Figure 9).

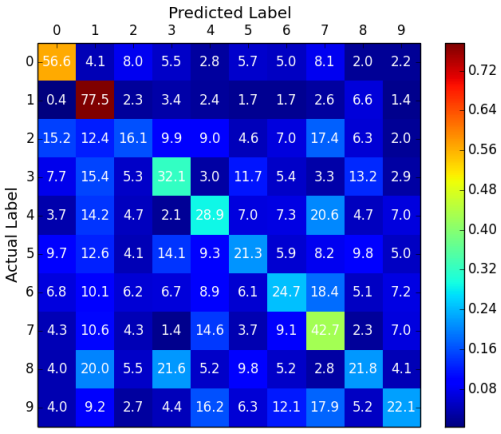

Figure 9: Feed-forward Neural network - Confusion Matrix

7.3 Linear SVM

The classifier was trained on 64 PCA components of the 3 feature sets (pixel, SPHOG and LBP). The first 45000 examples were used for training, and the last 5000 for validation. The higher percent accuracy observed was 38.14%. We can see the confusion matrix in Figure 10. Once again, the number "2" seems to be the hardest to classify.

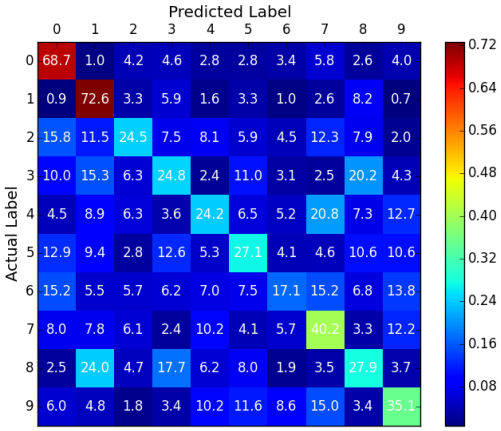

Figure 10: Linear SVM - Confusion Matrix

7.4 Convolutional Neural Network

The network was initially trained on 45000 data samples and the last 5000 were used as a validation set. It achieved up to 96% validation accuracy with the full methodology described above.

Looking at the confusion matrix in Figure 11, we see that there is significant number of misclassification for "9" with "6", which makes sense, since we have rotations of digits in the dataset. Other than that, the other confusions are intuitive too, such as "9" with "4", "5" with "3" and "2" with "7", as they all share similar shapes.

8. DISCUSSION

In this study, we compared multiple classifiers on the Difficult Digits Dataset. The literature shows that the convolutional neural networks are the most successful on the original MNIST dataset [2], followed by the SVM based methods, feed-forward neural nets, k-nearest neighbors and baseline methods. In this study, we observed the same trend of performance in the methods tested on the manipulated data.

Although the pattern of the accuracies follows that of the original MNIST data, the accuracies are substantially re-

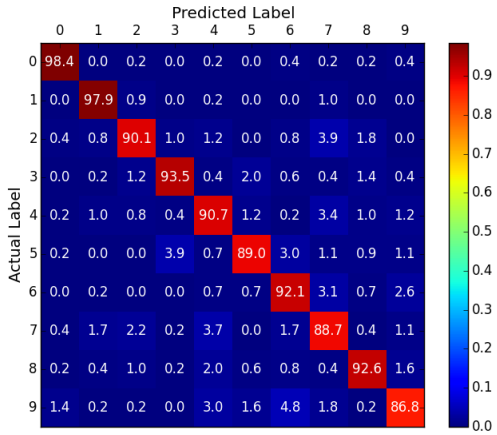

Figure 11: Convolutional Neural Network - Confusion Matrix

duced. This phenomenon was expected, given the distortions applied on the data set. Moreover, it was observed that on the Difficult Digits dataset, the distance between the performance of the convolutional neural nets and the other methods have been hugely increased. The convolutional neural nets are designed to need the minimum ad hoc feature engineering. In other words, there are subsequent feature extraction steps embedded in the network itself [2]. Considering the complexities of the dataset, we believe that the most suitable set of features should be invariant to rotation, scaling, tilting and robust to noise. Apparently, the specific architecture of the convolutional neural networks succeeds in providing such robust spatial features. Particularly, the multiple subsampling steps results in features which are less sensitive to spatial transformations and distortion. Eventually, the deeper layers of these networks are able to detect more global features that are not affected by the spatial manipulations such as those used in the Difficult Digits dataset.

We have tested many other classification techniques such as random forest, decision trees, feed-forward neural nets and SVM with RBF kernel. However, the accuracies were limited between 30-60%, with feed-forward neural nets with best parameter being the weakest at around 35 to 40 percent accuracy, and SVM with RBF kernel using the best parameter the strongest method with 55% accuracy.

Since the properties of the dataset suggest that scale and rotation invariant features might be the ideal set of features, we decided to try SIFT features as well [11]. However, as it has been shown in the literature [5], we observed that the SIFT features are not the appropriate candidates for MNIST digits dataset, while they work reasonably well in dataset of natural images.

We have shown the confusion matrix for each of the discussed methods Figures 8 and Figure 9. The matrices are normalized row-wise to account for classes of different size. The matrices show that "0" and "1" are the easiest digits to classify while "2" and "5" show the poorest classification performance for both logistic regression and feed-forward neural nets.

It is worth mentioning that, although we ran cross validation for all the hyper parameters in our feed-forward neural network implementation, since we have chosen the stochastic gradient descent which requires all the samples to pass through the network sequentially, the running time has increased significantly, which limits us to only 15 epochs on the whole dataset. However, we believe that if we had run more epochs, we would have reached better accuracies.

As expected, the convolutional neural network performed the best amongst all the classifiers we tested. With the full methodology described in the previous sections, we were able to reach an error rate of 6.31%, which is very satisfactory, considering out dataset is much more complex than the original MNIST dataset. In term of usage, it was also one of the simplest one, as we didn't have to compute any extra features or preprocess the images. It just worked right out of the box, and by using a Titan GPU, it ran reasonably fast.

Looking at the confusion matrix in Figure 11, we see that there is significant number of misclassification for "9" with "6", which makes sense, since we have rotations of digits in the dataset. Other than that, the other confusions are intuitive too, such as "9" with "4", "5" with "3" and "2" with "7", as they all share similar shapes.

We hereby state that all the work presented in this report is that of the authors.

9. REFERENCES

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1, June 2005.

[4] B. K. P. Horn. Shape from shading. chapter Obtaining Shape from Shading Information, pages 123–171. MIT Press, Cambridge, MA, USA, 1989.

[5] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. Le-Cun. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1605–1612, June 2009.

[6] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In F. Soulié and J. Hérault, editors, *Neurocomputing*, volume 68 of *NATO ASI Series*, pages 41–50. Springer Berlin Heidelberg, 1990.

[7] A. Krizhevsky. cuda-convnet, https://code.google.com/p/cuda-convnet/, 2012.

[8] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 473–480, New York, NY, USA, 2007. ACM.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[10] Y. Lecun and C. Cortes. 2009.

[11] D. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

[12] S. Maji and J. Malik. Fast and accurate digit classification, 2009.

[13] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda.

Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5Ű6):555 – 559, 2003. Advances in Neural Networks Research: {IJCNN} '03.

[14] H. Mhaskar and C. A. Micchelli. Approximation by superposition of sigmoidal and radial basis functions. *Advances in Applied Mathematics*, 13(3):350 – 373, 1992.

[15] T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision amp; Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 582–585 vol.1, Oct 1994.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] X. Wang, T. Han, and S. Yan. An hog-lbp human detector with partial occlusion handling. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 32–39, Sept 2009.