Handwritten Letter Recognition

Jervis Muindi*
Department of Computer Science
Stanford University
Palo Alto, CA 94305
jmuindi@cs.stanford.edu

Abstract

In this work, we explore applying machine learning techniques to the task of handwritten letter recognition. Our work covers logisitic regression and traditional classifiers such as SVMs (Support Vector Machines) and the more modern deep learning techniques including a trial of transfer learning. We find that CNN perform best on this task and are able to achieve an accuracy of more than 90% on the test set.

1 Introduction

Although a lot communication is digital and electronic, there is still some that occurs over traditional analog means. To that end, we would like to build a system that can recognize handwritten letters. Such a system would be useful for example to automatically parse and the mailing address on a postal letters so that they may be routed accordingly. It would also be useful as a component in indexing handwritten notes which has many numerous applications. For example, we can leverage this technology to automatically recognize handwritten medical doctor notes which is useful in gathering data and building a medical profile for a patient. This can serve as a means for gathering training data from medical notes for medical related ML applications.

Thus, this motivates our goal to build a model that can recognize handwritten letters.

2 Related Work

2.1 Digit Recognition

The task of recognizing handwritten letters is similar to that of recognizing handwritten numbers. In this domain, the seminal work was done by LeCun et al[17] where by they proposed using gradient based learning techniques for this instead of relying on manual heuristics. In that work in which they published the canonical modified NIST (MNIST) dataset[3], they found that convolutional neural networks can work especially well for recognizing digits.

2.2 Letter Recognition

On letter recognition we have work by Ganapathy et al[12] that used regular Neural Networks made of 3 layers (an input layer, a hidden layer, output layer) and were able to attain 85% recognition accuracy for english letters in their dataset.

There has been work to recognize handwriting from languages other than english and one such example is the work by Abandah et al [10] for Arabic script. For that task, they used recurrent neural networks (RNN) and at the time were able to build a system strong enough to win the ICDAR 2009 Arabic handwritten script challenge.

*Author also reachable via jervisfm [at] gmail dot com

Figure 1: EMNIST sample training data visulization

3 Dataset

The primary dataset that we use for our project is the Extended MNIST dataset. This is a dataset that consists of training images of handwritten english letters as well as their appropriate labels. There are a few variations of it and the one we use is the EMNIST letters variant. This specific dataset variant has 124,800 training examples split evenly across the letter classes and 20800 examples in a held out test set.

As the original dataset does not include a validation/dev set, we create one from the training data equal in size to the test set. While doing so, care is done to ensure that we still maintain balance across the various classes. Thus, our final split is 104,000 training examples, 20800 dev/validation examples, and 20800 test examples and we use the source data in the MATLAB format.

We show a visualization of ten randomly chosen examples from the training dataset for each of the letter classes in Figure 1

4 Methods

We look at classical machine learning techniques such as support vector machines as well as more modern techniques with Deep learning. Our code is available on Github[7]

4.1 Baseline: Logistic Regression with Softmax

In regular logistic regression, the log likelihood [18] to be optimized is given by:

$$l(\theta) = \sum_{i=1}^{m} y^{(i)} log\, h_\theta(x^{(i)}) + (1 - y^{(i)}) log\, (1 - h_\theta(x^{(i)}))$$

where

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

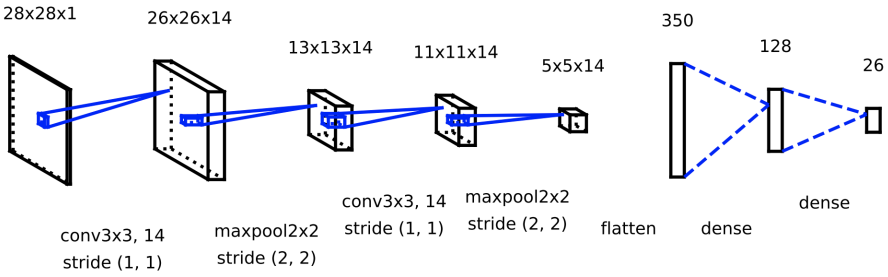


Figure 2: Our CNN (Convolution Neural Network) Model architecture

As we have more than 2 classes, we apply multinomial logistic regression with softmax that uses cross entropy loss. Cross entropy loss is given by:

$$CE_{loss} = -\sum_{c=1}^{C} y_c log\hat{y}_c$$

4.2 Support Vector Machine

Support Vector Machines are a classical machine learning technique. These are maximum margin classifiers and they try to find an optimal hyperplane that best separates the data whilst keeping some margin. In SVMs, the objective to minimize[19] is given by:

$$min_{w,b} \frac{1}{2} \|w\| \, s.t \, y^{(i)}(w^T x^{(i)} + b) >= 1, i = 1, ..., m$$

We explore using SVMs with various kernels. The kernels are linear kernel, polynomial kernel, RBF kernel, and sigmoid kernel.

4.3 Neural Networks

We look at using a few different types of neural networks discussed below.

4.3.1 Feed Forward Neural Network

The feed forward neural network consists of an input layer, hidden layer, and output layer. The input layer is the input image as a vector, and we use 5 hidden layers with (512,256,128,64, 32) number of hidden units respectively. We use ReLu activation and softmax for output.

4.3.2 Convolutional Neural Network (CNN)

We use a simple CNN network that consists of 2 convolution layers along with max pooling. The input to the CNN is a 28x28 image of a letter and we perform 2 convolutions and max pooling operation before flattening and adding a single fully connected 128-unit dense layer. Output layer has 26 classes and we use softmax. Again, we use ReLu activation for the hidden layers/units. A visualization of the CNN model architecture is given in figure 2.

4.3.3 Transfer Learning

Transfer learning is the idea of taking model trained on one task and then leveraging it to perform another (related) task[11]. We look at transfer learning from the ImageNet ResNet[14] model. Here we bootstrap model with pretrained weights and add a fully connect layer with 128 units along with softmax output. For training, we keep the initial layers fixed and only tune our custom added layers.

5 Results and Discussion

We now discuss the results of our experiments on the dev set. As we have a balanced dataset, our core evaluation metric we used was accuracy in recognizing the letters. We may however report additional metrics to paint a holistic picture of model performance.

5.1 Baseline

The baseline model was able to attain an accuracy of 64.5%. Precision, recall, and F1 score were 64%, 65% and 64% respectively. Looking at the confusion matrix[6], one of more prevalent errors for this model was confusing 'G' with 'Q'.

5.2 Support Vector Machine

We tested a few different kernels for the SVM which were implemented using SciKit Learn[20] machine learning toolkit. For the polynomial kernel, we used a degree value of 3. The kernel coefficient gamma for (RBF, polynomial, sigmoid) kernels was set to $1/numfeatures$. Results are shown in table 1 below.

Table 1: SVM Results

SVM Kernel	Accuracy (%)	Training Time[1](s)
Linear Kernel	41.6	521
RBF Kernel	70.4	619
Polynomial Kernel	73.0	556
Sigmoid Kernel	37.9	620

[1] Ran for 100 iterations.

The linear kernel had a low accuracy score of 41%. This is not surprising as the mapping for this recognition task is likely non-linear. Interestingly, the sigmoid kernel had lower performance of 37.9%. We did not perform any significant parameter tuning so this may be a contributing factor here. Overall, the best performing kernel was the polynomial kernel which attained an accuracy of 73%. This is an increase of about 13 percentage points over the baseline.

5.3 Neural Network

Neural network was implemented using Keras[2] and Tensorflow[9]. We used a Google Cloud Compute n1-highmem-2 machine[1] instance with Nvidia Tesla K80 GPU along with the Deep learning VM image[13] for training. The machine had about 12GB of system memory and 10GB GPU RAM. As of June 2019, the cost for running this machine comes out to about $14 a day. The results for the neural network experiments are shown in table 2.

Table 2: Neural Network Results

Neural Network Model	Accuracy (%)	Training Time[1](s)
Feedforward Network	90.3	1671
Convolutional Neural Network	91.1	1294
Transfer Learning (ResNet)	13.6	73823

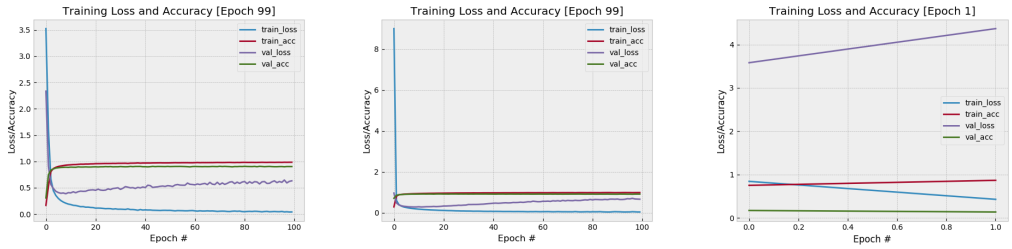
[1] Ran for 100 iterations, except for transfer learning which had 2 iterations.

We see that the best performing neural network is our simple Convolutional Neural Network that consisted of 2 convolutional layers along with max pooling attaining an accuracy of 91.1%. The Feed forward network of 5 hidden layers also had similar performance of 90.3% although it took a little longer to train due to the dense fully connected layers.

The transfer learning model had very poor performance on the dev set attaining an accuracy of 13.6%. While this is better than random guessing, it is much lower than our baseline model. One challenge

here is that training the model took an extraordinary long time with the resources at hand we were able to complete two training epochs that took 20.5 hours to complete.

Looking at the training plot graph in figure 3c[8] for the ResNet transfer learning model, it is clear that the model had overfit to the training dataset but failed to generalize to the dev set. Thus, one avenue here is to fix the variance problem with this model and apply some regularization. However, due to the prohitive cost of training these model we did not persue this option. We did also try other models such as VGG19[21] and MobileNet[15] and faced similar challenges. Some recent research also suggests that transfer learning from ImageNet models as done here may not always work[16].



(a) Feedforward Network (b) Convolutional Neural Network (c) Transfer learning: ResNet 50

Figure 3: Training plot graphs for the neural networks. They show training accuracy and dev accuracy over time in addition to loss

Hyperparameter tuning: Overall, the best performing neural network model was our CNN model. As this model had a gap of about 7% between training accuracy and dev accuracy[5], we did some hyperparameter tuning applying L2 regularization. Results from this exploration as shown in table 3. Slowly increasing the L2 regularization penalty improved the model performance a bit. However, we see that too heavy a penalty eventually reduces dev set accuracy. The optimal value for the L2 penalty is 0.01

Table 3: Hyperparameter tuning CNN model Results

L2 regulariation penalty	**Accuracy (%)**	**Training Time[1](s)**
0.001	92.1	1267
0.01	92.8	1264
0.05	90.1	1263
0.1	87.7	1260

[1] Ran for 100 iterations

Error analysis: Looking at the confusion matrix for our CNN model[4], we find that the most common error that the model makes is confusing 'I' and 'L'. This would make sense as these letters are very similar to each other, especially when written in lowercase ('i' vs 'l').

Test set performance: At the end of the project, we took our best model and tested it on the held out test set. We find that the model generalizes well and achieves an accuracy score of 92.5%.

6 Conclusion and Future Work

We have explored different machine learning techniques for the task of handwritten letter recognition. Our experiments show that the best performing model both in terms of accuracy and training time to be a convolutional neural network. This model is able to achieve 92.8% accuracy on the dev set and 92.5% on the test set and does not take too long to train. Our experiments with transfer learning from ImageNet on the other hand did not yield positive results as the models were cost prohibitive to fine-tune.

Future work: One area for future work is to continue working through the most common errors that the model is making. In addition, it would be interesting to explore using this model as a foundation in a system to recognize handwritten words all together.

Acknowledgments

We work would to thank the CS229A teaching staff Andrew Ng et al. for a great quarter. Thanks also to Paul de La Villehuchet for project advice.

References

[1] Google cloud n1-highmem-2 machine type. https://cloud.google.com/compute/docs/machine-types#highmem. Accessed: 2019-06-09.

[2] Keras: The python deep learning library. https://keras.io/. Accessed: 2019-06-09.

[3] Mnist dataset. http://yann.lecun.com/exdb/mnist/. Accessed: 2019-06-08.

[4] Our cnn model confusion matrix. https://github.com/jervisfm/cs229a-project/blob/master/imgs/cm_imgs/confusion_matrix_cnn_iter=100_l2reg=0.01.png. Accessed: 2019-06-09.

[5] Our cnn model training plot accuracy graph. https://github.com/jervisfm/cs229a-project/blob/master/results/training_plots/cnn_model_iter=100/cnn_model_iter=100_training_plot_accuracy.png. Accessed: 2019-06-09.

[6] Our cs229a baseline model confusion matrix. https://github.com/jervisfm/cs229a-project/blob/master/imgs/cm_imgs/confusion_matrix_logistic_regression_baseline.png. Accessed: 2019-06-09.

[7] Our cs229a project source code. https://github.com/jervisfm/cs229a-project. Accessed: 2019-06-09.

[8] Our transfer learning resnet model training plot graph. https://github.com/jervisfm/cs229a-project/blob/master/results/training_plots/transfer_learning_model_transfer_learning_with_ResNet50_iter=2/transfer_learning_model_transfer_learning_with_ResNet50_iter=2_training_plot_loss_accuracy.png. Accessed: 2019-06-09.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] G. A. Abandah, F. T. Jamour, and E. A. Qaralleh. Recognizing handwritten arabic words using grapheme segmentation and recurrent neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(3):275–291, Sep 2014.

[11] CS231n. Convolutional neural networks for visual recognition: Transfer learning. http://cs231n.github.io/transfer-learning/, 2019. Accessed: 2019-06-09.

[12] V. Ganapathy and K. L. Liew. Handwritten character recognition using multiscale neural network training technique. 2008.

[13] Google. Cloud deep learning vm image. 2018. https://cloud.google.com/deep-learning-vm/.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? *CoRR*, abs/1805.08974, 2018.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[18] A. Ng. Supervised learning lecture notes. 2018. `http://cs229.stanford.edu/notes/cs229-notes1.pdf`.

[19] A. Ng. Support vector machines. 2018. `http://cs229.stanford.edu/notes/cs229-notes3.pdf`.

[20] 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.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.