Difficult Handwritten Digit Classification

Code available at: `https://github.com/slflmm/Miniproject-3`

COMP 598 – Miniproject 3 – Team X

Narges Aghakazemjourabbaf 260460855

Stephanie Laflamme 260376691

Benjamin La Schiazza 260531181

ABSTRACT

In order to revive the challenge of digit classification, researchers at Université de Montréal have built a new, challenging dataset by applying transformations to MNIST images. We provide baseline results from a perceptron, a fully-connected neural network, and a linear SVM model, using both pixel features and features based on Gabor filters. Moreover, we present impressive results from a convolutional neural network, obtaining a test accuracy of 92.73%. We discuss the implications of our results and suggest further avenues of research.

1. INTRODUCTION

The MNIST dataset–a collection of labelled handwritten digits, obtained from American Census Bureau employees and high school students[12]–has been widely used as a benchmark in machine learning. However, handwritten digit recognition in the style of the MNIST dataset can very well be considered a solved problem; indeed, the best learners achieve near-human performance at correctly labelling new digits[4], obtaining a peak accuracy of 99.79%[19].

Bringing back the challenge of handwritten digit recognition, researchers at Université de Montréal introduce noise to MNIST images to form a new dataset for difficult handwritten digit classification. Images are enlarged from 28×28 to 48×48 pixels, digits are embossed, textures from a large variety are added to the images' backgrounds, and digits are rotated by random amounts.

In the context of an in-class competition, we tackle this dataset. In this paper, we report results with four classifiers; a perceptron, a fully-connected neural network, a linear SVM model, and a convolutional neural network. Moreover, when appropriate, we consider two feature sets; raw pixels and features derived from Gabor filters. We describe our learners and features in depth in the early sections of this report, and follow-up with a description of our methodology and results. We end with a discussion of the implications of our findings.

2. ALGORITHM SELECTION

As our preprocessing treatment and feature sets vary between our choices of learning algorithms, we introduce the four here; we use a perceptron, a fully-connected neural network, a linear SVM, and a convolutional neural network.

2.1 Perceptron

We implement a multiclass perceptron as a baseline classifier. Given n examples with m features each, and given k classes, the perceptron learns a matrix W of $(m+1) \times k$ weights (one weight for each feature-class combination, and a bias vector) by gradient descent on the error

$$Err(x) = (y - f(W^T x)),$$

where f is the Heaviside step function, x is an example, and y is its class.

2.2 Fully-Connected Neural Network

The fully-connected neural network, also known as a multilayer perceptron, concatenates layers of perceptrons, but uses a softer activation function than the step function. The advantage of this arrangement over a single layer is that complex, non-linear functions can be learned; for instance, a linear learner cannot encode the XOR function, but a neural network can do it with only two layers. Our implementation uses the easily differentiable sigmoid, written

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

The weights of each layer are learned by gradient descent on the error,

$$Err(x) = y - h_w(x),$$

where $h_w(x)$ is the network's current guess for the label of example x and y is its true label.

2.3 Linear SVM

We use the scikit-learn [16] library's 'SVC' implementation of a linear SVM (support vector machine). This learning algorithm solves the quadratic optimization problem written as

$$\arg\min_{w,b} \|w\|^2$$

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

where w is the normal vector to the hyperplane separating two classes. It handles the dataset's multiple classes using a 'one-vs-the-rest' approach.

2.4 Convolutional Neural Network

Although fully-connected neural networks tend to perform very well at machine learning tasks, their ability to classify images is sensitive to shifts in position or shape distortions– they can be bad at identifying invariant patterns. In the early '80s, Fukushima introduced convolution layers to remedy that problem by handling geometrical similarities regardless of position.[6] A convolution applies a filter to an image; a convolution layer contains many such filters, whose values are learned as the neural network is trained. This serves as an implicit, learned feature extraction at the start of the neural network, typically followed by hidden layers.

The use of convolution layers seem particularly appropriate given the difficulty of our dataset, which adds noise-inducing modifications to the MNIST dataset. Indeed, LeCun has shown that a convolutional neural network trained with MNIST images continues to predict digits correctly regardless of rotations and noise.[11]

We note that neural networks with many layers tend to overfit; there may be many complicated relationships between examples and their labels, and enough hidden units to model these relationships in multiple ways. Hinton et al.[8] introduced the concept of dropout to alleviate this issue. During training, hidden units are randomly omitted with probability p, which helps prevent complex co-adaptations on training data.

The application of dropout to convolutional neural network has been successfully applied; Krizhevsky et al. applied dropout to the hidden layer weights of their convolutional neural network, and vastly outperformed other methods on an ImageNet classification competition.[10]

We also consider rectifier linear units (ReLUs) as activations, written

$$f(x) = \max(0.0, x).$$

This activation function is known to perform better than bounded activations (e.g. tanh) in deep neural nets for image-related tasks, presumably because they preserve intensity information about incoming information.[15]

In light of these findings, we implement our own convolutional neural network. Its architecture consists of a variable number of convolution layers, followed by a variable number of hidden layers and the output layer. We consider both tanh and ReLU activations, and apply dropout to the hidden layers. We also experiment with the use of learning rate decay and momentum. Our network is trained using minibatch SGD.

3. PREPROCESSING

The raw pixels are normalized. In the case of the perceptron, fully-connected neural network, and linear SVM learners, they are standardized. For each example i in the training set, and each feature j, we replace features with their standardized value

$$x'_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j},$$

where μ_j is the average value for feature j and σ_j is its standard deviation. These values for μ_j and σ_j are subsequently used to apply the same transformation to the test set examples.

For the convolutional neural network, we apply local contrast normalization; given a 48×48 image X from the dataset, we let

$$X = \frac{(X - X_{min})}{X_{max} - X_{min}},$$

where X_{max} and X_{min} are the maximum and minimum values in X's pixels, respectively. This ensures that pixel intensity falls between 0 and 1, which allows us to safely use rectifier linearities as activation functions.

4. FEATURE DESIGN AND SELECTION

When appropriate for the learner, we consider two feature sets; standardized raw pixels and Gabor filter-based features. For the open method, we consider contrast-normalized raw pixels and the addition of rotation-perturbed examples.

4.1 Pixels

We use the post-preprocessing pixel information as a baseline feature set.This produces feature vectors of length 2304.

4.2 Gabor

Gabor filters are linear filters used for edge detection and are thought to be similar to the early stages of visual processing in humans. Their frequency and orientations correspond roughly to simple cells in the visual cortex of humans and other mammals; these cells can be modelled with Gabor functions.[9][14] As these filters are well-suited to image processing, we attempt to translate them into features.

We use two kinds of Gabor filters: for the perceptron and fully-connected neural network, we use a simplified expression due to lack of time and memory; we describe these features in more detail below. For the linear SVM model, we take advantage of scikit-learn's online learning capacity and use as features the pixels of the results of convolving each filter with an image.

Previous research has used the energy of the convolution between a Gabor filters and image (a measure of how strongly the filter responds to that image[7]) as a feature by summing its magnitudes in the image.[1] This is equivalent to using the Frobenius norm of the convolved image as a feature.

Using the Scikit-learn library[16], we generate 16 Gabor filters with 8 equidistant θ values, $\sigma = \{1, 3\}$, and frequencies $= \{0.05, 0.25\}$. Figure 1 illustrates the effects of θ, σ, and frequency with a sample of our filters. We form the feature vector of an image in the dataset by collecting the Frobenius norms of the image convolved with each filter.

4.3 Rotation perturbations

We introduce a new training set by perturbing a copy of each training example with a random rotation. This doubles the

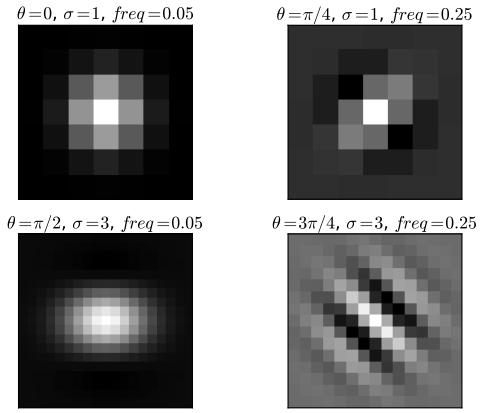

Figure 1: Some Gabor filters

size of our training data; we predict that this dataset will perform better with the convolutional neural network, as these learners tend to benefit from an increase in the number of examples.

5. OPTIMIZATION

Given the vast amount of time required to train a convolutional neural network, we implement ours to run on a GPU. This reduces the training time by approximately a factor of 10. We use Theano[3], a symbolic Python library that compiles computation-heavy functions in C and can execute them on a GPU. We use the cuda-convnet implementation of convolutions[10], which offers a 3-fold speed improvement relative to Theano's convolution function.

6. PARAMETER SELECTION METHOD

For each learner, we fix the hyperparameters to compare the performance of appropriate feature sets, and perform a preliminary manual inspection to identify viable hyperparameters, followed up by either the traditional gridsearch, or a random search. In cases where there are many hyperparameters and training is long, random search offers a faster alternative to gridsearch for finding good hyperparameters. It has been shown to obtain results as good as or better than gridsearch[2]. The details of parameter selection are discussed in this section.

6.1 Perceptron

The perceptron can use pixel or Gabor features. We initially fix the hyperparameters to identify the most promising feature set using 5-fold cross-validation. We then consider its two hyperparameters–learning rate and number of learning iterations–and perform gridsearch for $\alpha \in [10^{-4}, 10^{-1}]$ and the number of iterations ranging form 10 to 35; the averaged validation accuracy from 5-fold cross-validation is used as a measure of success for the hyperparameter setting.

6.2 Fully-Connected Neural Network

The neural network can also use pixel or Gabor features. As we did for the perceptron, we fix what appear to be reasonable hyperparameters and identify the best feature set for this classifier; we use results from 2-fold cross-validation, as neural networks take a long time to train. Then, we perform random search over 15 hyperparameter settings; we use mean accuracy from 2-fold cross-validation to determine the most promising parameter setting. Table 1 lists the hyperparameters and the range of values we consider.

Hyperparameter	**Values**
Number of layers	{1, 2}
Hidden units per layer	{10, 15, ..., 30}
Learning rate	$[0.0001, 0.1]$

Table 1: Neural network hyperparameters and their considered values

6.3 Linear SVM

As we do for the perceptron and fully-connected neural network, we used reasonable hyperparameters to determine which feature set between pixels and Gabor features had the most potential with this classifier. Using results from a 5-fold cross validation run with the full Gabor feature set and the standardized pixel data, determine the best feature set for htis classifier. We then perform cross-validation over two hyperparameters; the learning rate $\alpha \in [10^{-4}, 10^{-1}]$ and the number of iterations (between 1 and 50).

6.4 Convolutional Neural Network

The only appropriate features for a convolutional neural network are pixel features. We must optimize many hyperparameters; in addition to those for the fully-connected neural network, we must also consider the dropout rate, as well as the number of convolution layers, the number of filters per layer, and the filter sizes. We must also consider activation functions, learning rate decay, and momentum rate.

Due to time constraints, we can only sample a small portion of this very large parameter space. Hence, we select hyperparameters with manual search and train models until they reach a validation accuracy plateau to compare their performance. It should be noted that, due to time concerns, we simply use 10% of the training data as a validation set rather than our usual cross-validation method.

We start from the parameters used in Theano's convolutional network tutorial, where they train a network on the MNIST dataset. That is, we start with two convolutional layers with 20 and 50 filters respectively, each with filter sizes 5×5. We use a learning rate $\alpha = 0.1$, without momentum, learning rate decay, or the extended dataset. From there, we apply modifications to each parameter and attempt to find a good model.

Luckily, we can use heuristics to reduce search when it comes to the size of hidden layers and the size and number of filters per convolution layer. We keep in mind tips and tricks offered by Theano's deep learning tutorials[13]:

- When filter size increases, number of filters decrease; this is to preserve information across convolution layers.

- 5×5 filters in the first convolution layer work well with the MNIST dataset of size 28×28 pixels, while filter sizes of 12×12 or 15×15 work well with images with hundreds of pixels in each dimension.

7. TESTING AND VALIDATION

7.1 Perceptron

In a primary step, we performed 5-fold cross-validation to compare raw pixels and Gabor features using a fixed learning rate ($\alpha = 0.01$) and number of iterations (15). As shown in Table 2, pixel features were the most performant, with a validation accuracy of 26.282%.

Features	Pixels	Gabor
Accuracy	26.282	10.398

Table 2: Mean validation accuracy of perceptron using different features

Using pixel features, we performed 5-fold cross-validation over the learning rate α and the number of training iterations of the perceptron model. Figure 2 shows the results of gridsearch with both parameters. The precise values of the best parameters found by the gridsearch cross-validation procedure were $\alpha = 0.0005$ with 25 training iterations, yielding a mean validation accuracy of 26.598%.[1]

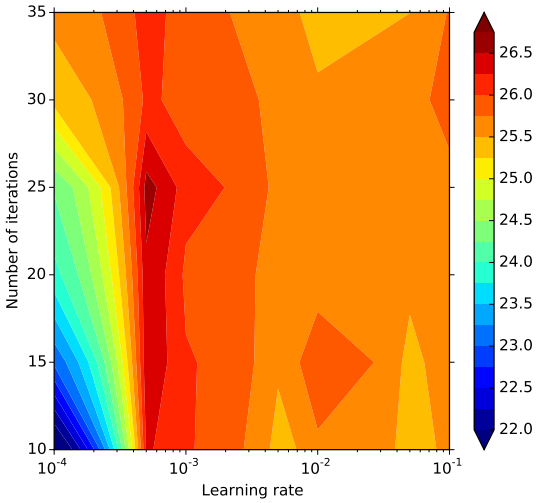

Figure 2: Mean cross-validation accuracy as a function of parameters α and number of iterations

After training our perceptron on the complete training set using these parameters, we submitted our results to Kaggle and obtained a test accuracy of 27.420%. As an approximation of the test confusion matrix, we provide the confusion matrix for the combined validation sets in Figure 3. Note the perceptron's greater ability to identify 0s and 1s compared to other digits.

[1]Additional results showing training error vs validation error are shown in Appendix A

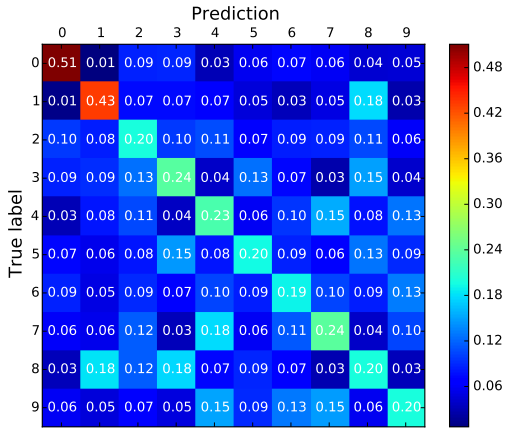

Figure 3: Validation confusion matrix for perceptron

7.2 Fully-Connected Neural Network

To select the best feature set for this classifier, we performed a quick 2-fold cross-validation with each feature set, setting $\alpha = 0.01$, with one layer of 15 hidden units, and training for 10 epochs. Table 3 shows the mean validation accuracies of each feature set; standardized pixel features performed best with 32.570% accuracy.

Features	Pixels	Gabor
Accuracy	32.570	12.944

Table 3: Mean validation accuracy of neural network using different features

Using standardized pixel features, we performed 2-fold cross-validation over the learning rate α, the number of layers, and the number of hidden units per layer using a random search technique. As we found during manual search that validation accuracies level off at 20 training epochs, we train our models for that duration. From our 20 samples, shown in Table 4 (and Figure 11 in Appendix A), we see that the model with learning rate 0.01 and one hidden-layer with 25 hidden units performed best.

We train our neural network using these hyperparameters and submit test predictions to Kaggle; this yielded a test accuracy of 37.23%. Once more, we show the cross-validation confusion matrix in Figure 4.

7.3 Linear SVM

We gathered results from two datasets; the original standardized pixel features, and pixel features resulting from convolving Gabor filters with the example images. Table 5 shows the results of a 5-fold cross-validation, which we used to determine the best feature set.

The standardized pixels used as features yielded the best validation accuracy with the linear SVM model. Using this feature representation, we performed cross-validation over the learning rate α and the number of learning iterations. We show the results of cross-validation in Figures 12 and 13 of Appendix A. Our best model used $\alpha = 10^{-4}$ and 5

Learning rate	Layer Sizes	Accuracy
0.001	[2304, 10, 10, 10]	21.562
0.1	[2304, 25, 20, 10]	27.241
0.005	[2304, 15, 10]	29.065
0.1	[2304, 20, 10]	29.971
0.0005	[2304, 10, 10]	28.0548
0.005	[2304, 30, 10, 10]	28.748
0.01	[2304, 10, 25, 10]	29.412
0.0005	[2304, 15, 10]	28.619
0.01	[2304, 25, 10]	29.151
0.01	[2304, 20, 10]	29.565
0.001	[2304, 25, 25, 10]	26.832
0.1	[2304, 20, 30, 10]	29.736
0.01	[2304, 30, 30, 10]	31.259
0.1	[2304, 15, 15, 10]	31.000
0.05	[2304, 10, 10]	31.320
0.005	[2304, 25, 15, 10]	31.609
0.01	[2304, 25, 10]	31.886
0.001	[2304, 15, 10, 10]	30.329
0.01	[2304, 10, 30, 10]	30.678
0.05	[2304, 10, 10]	30.903

Table 4: Cross-validation accuracies for the neural network

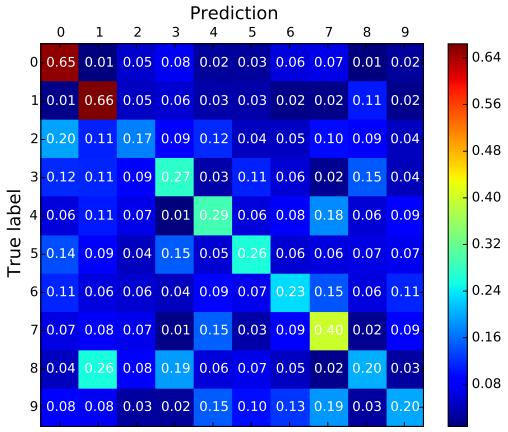

Figure 4: Validation confusion matrix for fully connected neural network

Features	Pixels	Gabor
Accuracy	34.5	30.0

Table 5: Mean validation accuracy of linear SVM using different feature sets

learning iterations.

This model achieved a test accuracy of 35.8% on the Kaggle public leaderboard. As before, we show the confusion matrix using the validation set in Figure 5.

7.4 Convolutional Neural Network

We experimented with a number of hyperparameter settings. Figure 6 shows the validation accuracy of several settings. We obtained our best validation results with a convolutional neural network comprised of 4 convolution layers and one

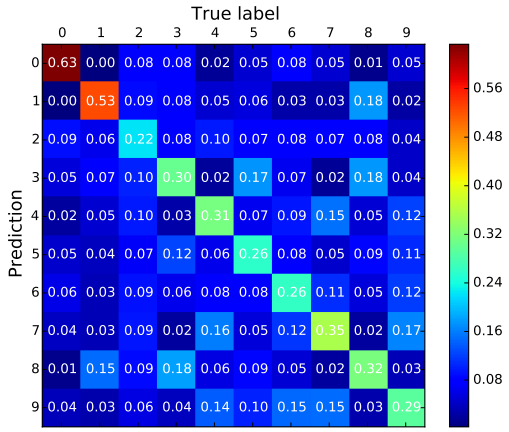

Figure 5: Validation confusion matrix for Linear SVM classifier with standardized pixel dataset.

hidden layer, using a weight decay factor of 0.995, and rectified linear units (ReLU) for all neurons.

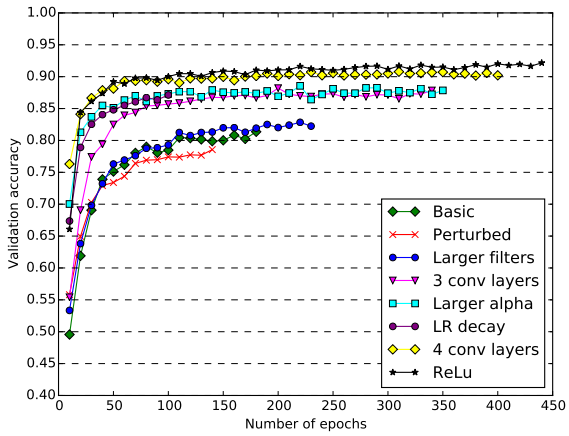

Figure 6: Validation scores of convolutional neural network at different hyperparameter settings. Appendix A elaborates on the specific hyperparameters used in the trials showed here.

Our best model, which obtained a validation accuracy of 92.64% after 750 training epochs, achieved a test accuracy of 92.73% on the public Kaggle leaderboard. Figure 7 shows this convolutional neural network's architecture. Figure 8 displays an estimate of the model's confusion matrix using validation results at epoch 100.

8. DISCUSSION

We trained four models on the difficult digits dataset; a perceptron, a fully-connected neural network, a linear SVM, and finally, a convolutional neural network, which yielded our best test result of 92.73% on the Kaggle leaderboard, vastly outperforming our other classifiers. Considering the strength of convolutional neural networks for image classification, and their resistance to noise, this is an unsurprising

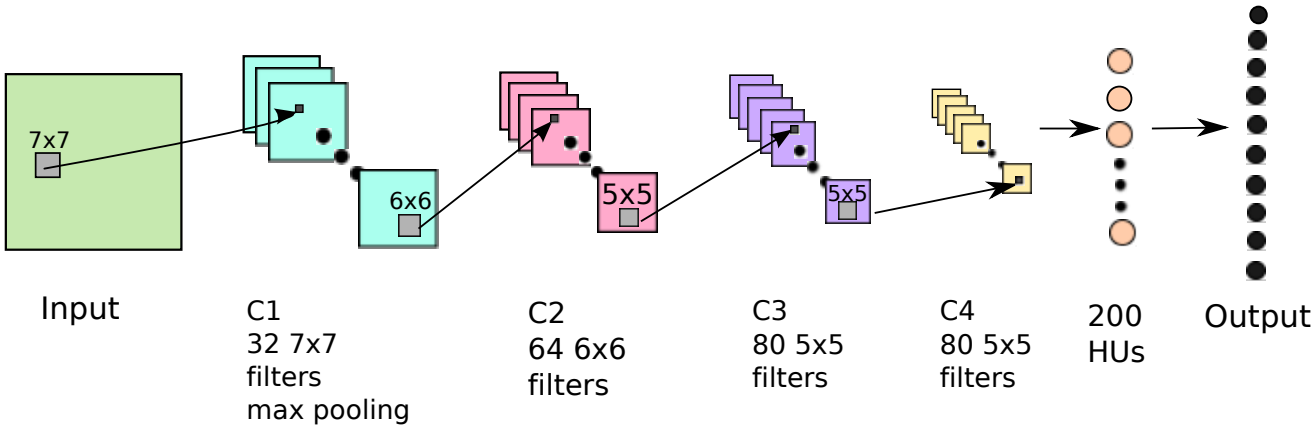

Figure 7: Final convolutional neural network architecture. Drawn to scale.

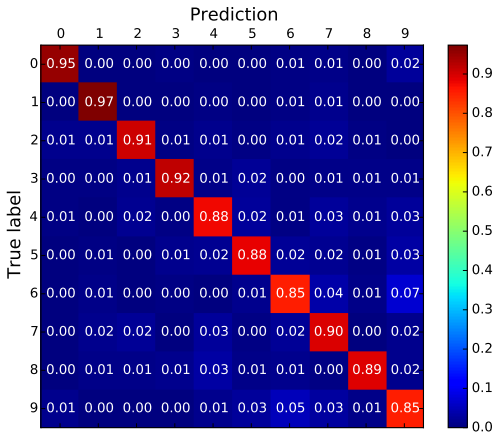

Figure 8: Confusion matrix on validation set for convolutional neural network after 100 training epochs

result.

Interestingly, both variations of Gabor features failed to produce good results. The smaller representation used for the perceptron and neural network presumably did poorly because it had too few features, or lost too much information relative to the pixel representation. The larger Gabor representation, used for the linear SVM, also did poorly–we believe that this was caused by an opposite problem; this was simply too large of a feature set for these classifiers to handle.

Our use of Gabor filters as a source of features had its strength in its domain-specific application; as we discussed earlier, they hold similarities to the way the brain processes visual information. However, it was difficult to translate convolutions of filters with images into features. An alternative would have been to use Gabor filters as a kernel with the SVM model[18], or using an entirely different feature representation generated from an auto-encoder.

The inclusion of rotationally perturbed examples for training the convolutional neural network did not improve validation accuracy. This is likely because we did not include perturbations of the validation set; a better approach might have been to pool the prediction of the original example with the prediction for its perturbed version. We might have also tried including more perturbed examples, perhaps tripling or quadrupling the size of the training set. Unfortunately, due to time constraints, this was not feasible. Another alternative might have been to pretrain the convolutional neural network with enlarged upright images from the MNIST dataset.[5][17]

Our confusion matrices reveal some interesting details; classifiers seem to have much more ease with 0s and 1s than with any other number. An inspection of the dataset did not reveal flagrant differences in the number of examples per class; there were perhaps 500 more examples of 1s than of other numbers, but this seems relatively minor. It seems more likely that the difference is due to their easily recognizable, and fairly universal shape across handwritings and orientations. We also observed that the neural network frequently labeled 8s as 1s; we suggest that this may be due to some individuals writing 8s so narrow that they resemble 1s.

As a whole, our results emphasize the difficulty of working with the difficult digits dataset, and the necessity of extremely descriptive feature representations; we achieved impressive results with convolutional neural networks, which implicitly extract such features before learning. Future research might consider reviewing the notion of introducing perturbed examples, or larger convolutional neural networks containing more layers–indeed, we noted best results with larger filters, and a sizeable gain in accuracy with each new layer added. Researchers might also attempt the method described by Wan et al.[19], who implemented drop-connect, a version of dropout where connections are dropped rather than neurons; they obtained the best results with the MNIST dataset by pooling the predictions of multiple convolutional neural networks trained with drop-connect in their hidden layers.

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

9. REFERENCES

[1] T. C. Bau. *Using Two-Dimensional Gabor Filters for Handwritten Digit Recognition*. PhD thesis, M. Sc. thesis, University of California, Irvine.

[2] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[3] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[5] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[6] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[7] S. E. Grigorescu, N. Petkov, and P. Kruizinga. Comparison of texture features based on gabor filters. *Image Processing, IEEE Transactions on*, 11(10):1160–1167, 2002.

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[9] J. P. Jones and L. A. Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of neurophysiology*, 58(6):1233–1258, 1987.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Y. LeCun. Lenet-5, convolutional neural networks. `http://yann.lecun.com/exdb/lenet/index.html`, 2004. Accessed: 2014-10-22.

[12] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. `http://yann.lecun.com/exdb/mnist/`. Accessed: 2014-11-2.

[13] LISA. Convolutional neural networks (lenet). `http://www.deeplearning.net/tutorial/lenet.html#lenet`, 2008. Accessed: 2014-10-25.

[14] S. Marčelja. Mathematical description of the responses of simple cortical cells*. *JOSA*, 70(11):1297–1300, 1980.

[15] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[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] H. A. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 38–44. IEEE, 1998.

[18] M. Sabri and P. Fieguth. A new gabor filter based kernel for texture classification with svm. In *Image Analysis and Recognition*, pages 314–322. Springer, 2004.

[19] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

APPENDIX
A. ADDITIONAL RESULTS

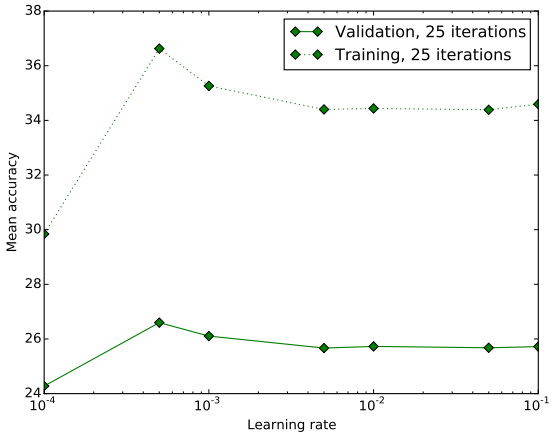

Figure 9: Cross-validation over α with perceptron, keeping # iterations optimal

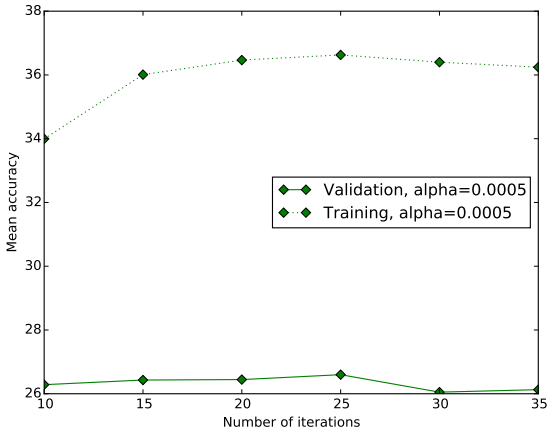

Figure 10: Cross-validation over # of iterations with perceptron, keeping α optimal

s

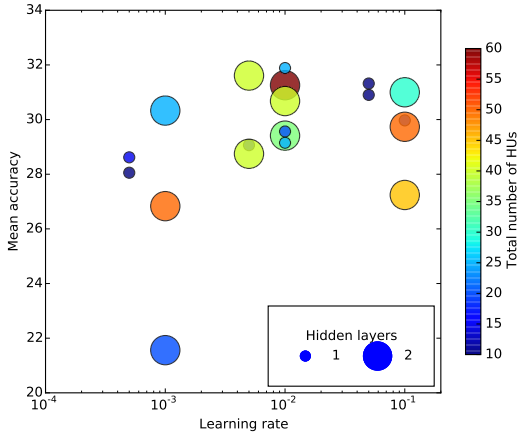

Figure 11: Cross-validation results for the neural network

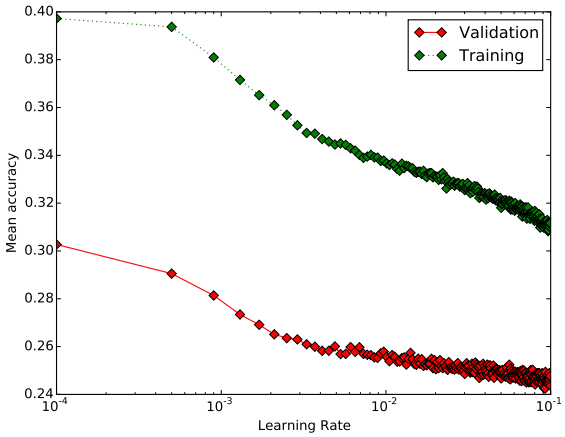

Figure 12: Cross-validation over learning rate for the linear SVM

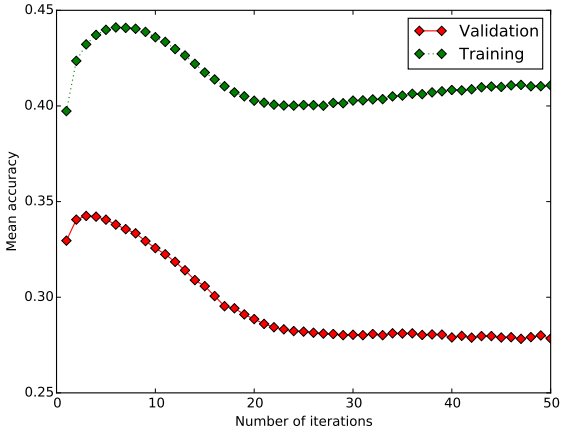

Figure 13: Cross-validation over number of iterations for the linear SVM

Label	**Conv**	**Hidden**	**Filter size**	**# Filters**	**# HUs**	**Batch size**	**Alpha**	**Drop**	**Mom**	**Perturb**
Basic	2	1	5ds,5ds	20,50	200	500	0.1	0.5	No	No
Perturbed	2	1	5ds,5ds	20,50	200	500	0.1	0.5	No	Yes
Larger filt	2	1	7ds,6ds	20,50	200	500	0.1	0.5	No	No
3 conv	3	1	7ds,6ds,4	20,50,70	200	500	0.1	0.5	No	No
Larger α	3	1	7ds,6ds,4	20,50,70	200	512	0.25	0.5	No	No
LR decay	3	1	7ds,6ds,4	32,64,80	200	512	0.2, dec 0.995	0.5	No	No
4 conv	4	1	7ds,6,5,5	32,64,80,80	200	512	0.2, dec 0.995	0.5	No	No
ReLU	4	1	7ds,6,5,5	32,64,80,80	200	512	0.2, dec 0.995	0.5	No	No
9	3	1	7ds,6ds,4	32,64,80	200	512	0.2	0.5	0.1	No
10	3	1	7ds,6ds,4	32,64,80	200	512	0.2	0.5	0.9	No
11	3	1	7ds,6ds,4	20,50,70	200	256	0.25	0.5	No	No

Table 6: Some hyperparameters tested for convolutional neural network. Those labelled with numbers are not shown in Figure 6 because they show little of interest. The notation 'ds' refers to downsampling after filter application. ReLU was our best model.