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

1. INTRODUCTION

- Talk about the dataset and the difficulty with it

- Discuss features

- Discuss learners

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 sum-squared error,

$$Err(x) = \frac{1}{2}(y - h_w(x))^2,$$

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 [14] 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.[5] 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.[10]

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.[7] 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.[9]

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 apply dropout to the hidden layers, and train our network using minibatch SGD. We also experiment with the use of learning rate decay and momentum.

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

$$X = \frac{256 \times (X - X_{min})}{X_{max} - X_{min}},$$

where X_{max} and X_{min} are the maximum and minimum values in X's pixels, respectively.

4. FEATURE DESIGN AND SELECTION

When appropriate for the learner, we consider three feature sets; standardized raw pixels, PCA, 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-standardization pixel information as a baseline feature set.This produces feature vectors of length 2304.

4.2 PCA

Principal component analysis (PCA), developed by Pearson in the early 1900s[13], converts a set of possibly correlated features into a linearly uncorrelated representation, with the resulting features ordered by variance.

We use the implementation of PCA provided by the Scikit-learn library[14], which uses singular value decomposition of the input data matrix.

As removing the least useful features made the results of our baseline classifier worse, we keep full dimensionality. However, we expect PCA features to produce better results given that they are linearly uncorrelated.

4.3 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.[8][12] As these filters are well-suited to image processing, we attempt to translate them into features.

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[6]) 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[14], 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.

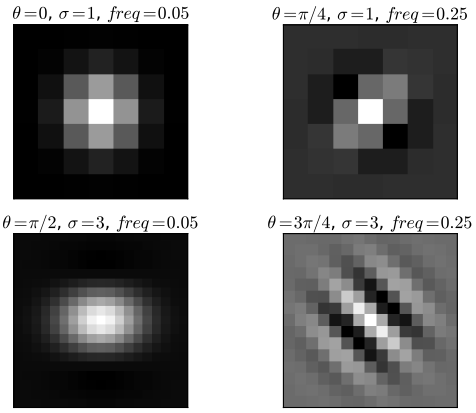

Figure 1: Some Gabor filters

4.4 Perturbed

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[9], 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, PCA, 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, PCA, 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 using results from 5-fold cross-validation. Then, we perform random search for 15 hyperparameter settings selected over the space of hyperparameters; we use mean accuracy from 5-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, ..., 50}
Learning rate	$(0, 0.1]$
Momentum	$[0.1, 0.9]$
Training epochs	$\{50, ..., 200\}$

Table 1: Neural network hyperparameters and their considered values

6.3 Linear SVM

6.4 Convolutional Neural Network

The only appropriate features for a convolutional neural network are pixel features. In this case, rather than using per-feature standardization, we apply contrast normalization within images.

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. Due to time constraints, we can only sample a small portion of this very large parameter space. Hence, we select hyperparameters with manual search.

Luckily, we can use heuristics to reduce our search space when it comes to the size of hidden layers and the size and number of filters per convolution layer. To start with, tips and tricks offered by Theano's deep learning tutorials note that, to preserve information across convolution layers, the value of (number of feature maps $\times$ number of incoming pixels) is usually chosen to be constant. Moreover, it is suggested that 5×5 filters in the first convolution layer work well with the MNIST dataset, while filter sizes of 12×12 or 15×15 work well with images with hundreds of pixels in each dimension.[11] Noting that MNIST images are 28×28 pixels, whereas our dataset's images are 48×48 pixels, we consider filter sizes ranging from 5×5 to 8×8. We begin with 3 convolution layers, with 20 to 50 filters per layer; we adjust the number of filters of later layers as a function of the values for the first layer.

We use one hidden layer; as discussed in class, with binary inputs, the number of hidden units in a hidden layer should be around $\log(n\_inputs)$, where n_inputs is the number of connections going into the layer. Since our inputs are not binary, we start with $10 \times \log(n\_inputs)$ hidden units.

7. TESTING AND VALIDATION

7.1 Perceptron

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

Features	Pixels	PCA	Gabor
Accuracy	25.794	26.282	10.398

Table 2: Mean validation accuracy of perceptron using different features

Using PCA 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]

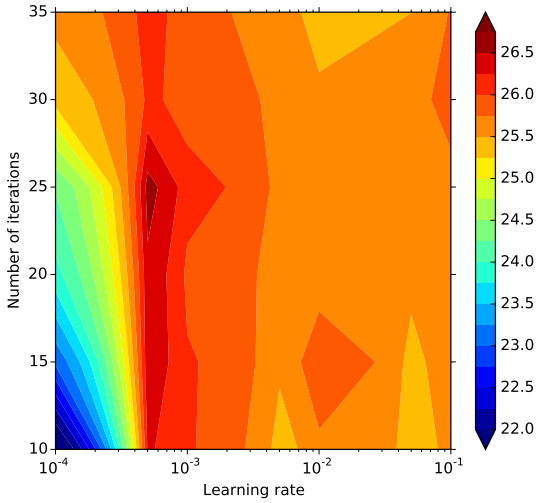

Figure 2: Mean cross-validation accuracy as a function of parameters α and number of iterations

After training our perceptron on the complete training set using these parameters, we submitted our results to Kaggle

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

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.

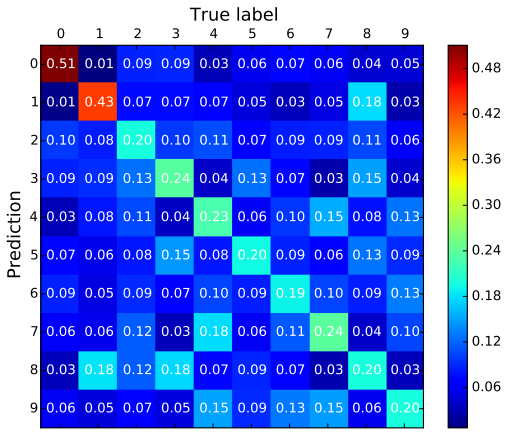

Figure 3: Validation confusion matrix for perceptron

7.2 Fully-Connected Neural Network

Try a baseline net with the three feature sets, compare (5-fold cross-validation).

Use the best feature set to find best hyperparameters with cross-validation. You should probably try random search rather than gridsearch (faster and just as good or better), for some set number of configurations. Use some graph to show the hyperparameter space results. Plot validation confusion matrix. Say the test set (kaggle) result using best model found during crossval.

7.3 Linear SVM

7.4 Convolutional Neural Network

We experimented with a number of hyperparameter settings. Figure 4 shows the validation accuracy of several settings. Appendix B elaborates on the specific hyperparameters used in the trials showed there.

Our best model, whose architecture is shown in Figure 5, obtained a test accuracy of ??? on the Kaggle leaderboard. Figure ??? demonstrates an estimate of the model's confusion matrix using validation results.

(something like this, but the architecture is totally subject to change)

8. DISCUSSION

Why is it better at classifying 0s and 1s? Interestingly, it is not due to different proportions of examples in those classes; the data is evenly distributed.

Using Gabor filters as a kernel rather than feature [16]

Other version of dropout [18]

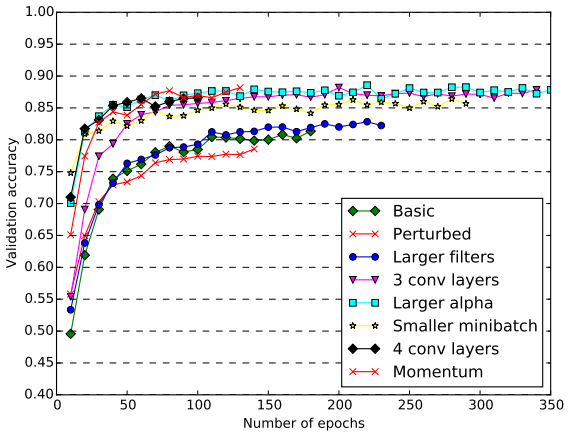

Figure 4: Validation scores of convolutional neural network at different hyperparameter settings

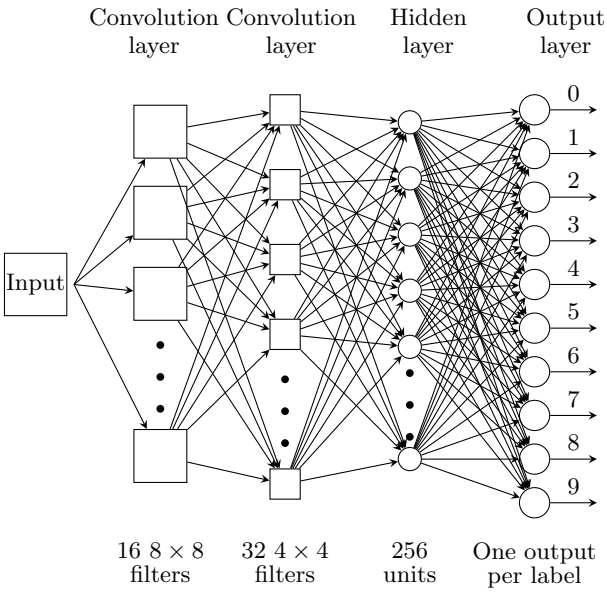

Figure 5: Convolutional neural network final architecture

Pretraining [4]

Others: [15], [17]

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. 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[14] 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.

[15] 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.

[16] 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.

[17] P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition–tangent distance and tangent propagation. In *Neural networks: tricks of the trade*, pages 235–269. Springer, 2012.

[18] 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

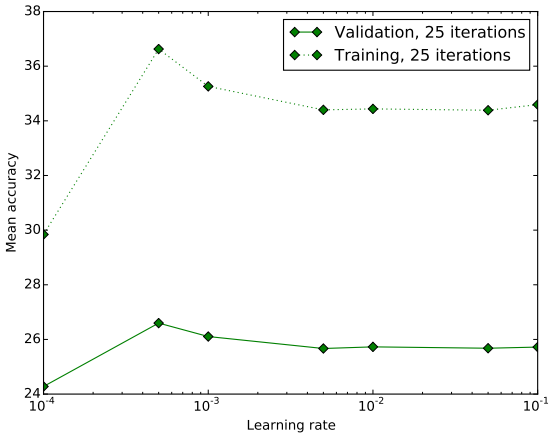

Figure 6: Cross-validation over α with perceptron, keeping # iterations optimal

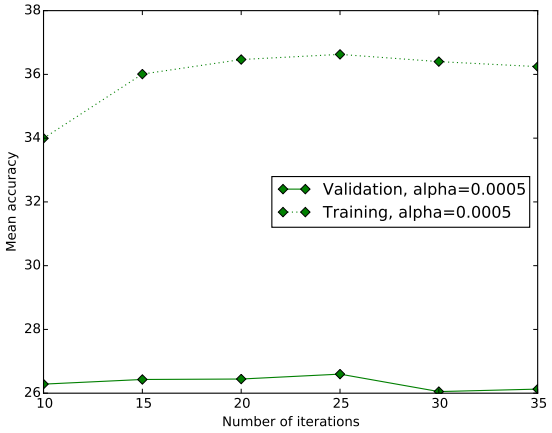

Figure 7: Cross-validation over # of iterations with perceptron, keeping α optimal

B. HYPERPARAMETERS FOR CONVOLUTIONAL NEURAL NETWORK