Approaches to Generalized Character Recognition

Alex X Chen Harini Kannan Nitya Subramanian
Massachusetts Institute of Technology
Cambridge, MA
{axc, hkannan, nityas}@mit.edu

Abstract

Generalized optical character recognition is a challenging but very useful task. Using help from open source libraries, we implemented and tested three existing methods to perform generalized character recognition. All classifiers were tested on two datasets, the first being handwritten characters and the second being characters that appeared in natural photographs. First, we tried a logistic regression classifier using HOG feature vectors as input, achieving an accuracy of 52.6% on handwritten characters and 53.9% on photographed characters. Next, we used an artificial neural network to classify characters, also using HOG feature vectors (55.5% accurate for handwritten characters and 63.5% accurate for photographed ones). Lastly, we used a convolutional neural network to classify the character images, giving an accuracy of 66.4% for handwritten characters and 64.6% for photographed ones. We also implemented character segmentation to complete the pipeline.

1. Introduction

Generalized optical character recognition is one of the most important and broadly applicable problems in computer vision. For example, simple document scanning was one of the earliest applications of OCR, which is widely used everywhere, from automated check deposits to license plate recognition on traffic camera photos. However, the generalized problem of OCR is much more complicated, as document processing is a very constrained domain. General character recognition focuses on detecting words in natural images and handwriting, where the letters could come in different shapes, sizes, or forms. They could be warped, blurred, slanted, subjected to background noise, or otherwise obscured. The applications of generalized OCR are immense - for example, self-driving cars could read street signs in order to navigate roads safely. A picture of professor's notes on a whiteboard could even be automatically transcribed to a pdf in LaTeX.

Convolutional neural networks (CNNs) are particularly well-suited to the problem of object detection, as they take advantage of the spatial context of pixels that are close together. A convolutional neural network is complex machine learning model built from layers of neurons, or computational nodes, as shown in Figure 1.

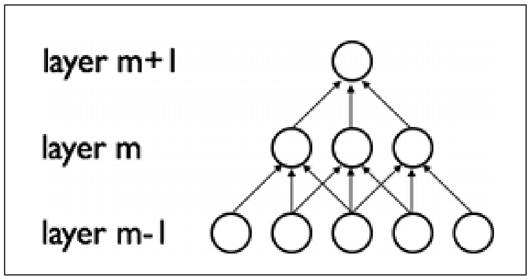

Figure 1. Three example layers in a CNN comprised of neurons (from `http://deeplearning.net/tutorial/lenet.html`).

Each layer takes its input from the layer below and outputs it to the layer above. In this figure, each node only depends on the three nodes below it, and each set of three nodes is a receptive field, or a subregion of the input image. At each layer, the CNN convolves the relevant receptive field with a linear filter. By convolving subregions of the input image, a CNN takes advantage of the spatial context of pixels grouped together, allowing it to identify objects in images. Additionally, the use of the convolution operator allows for the input transformation that takes place at each layer to be computed as a convolution rather than a full matrix multiplication. This additional capability makes CNNs more computationally efficient than their other neural net counterparts and therefore well-suited to massive-scale, deep learning neural networking tasks, such as generalized OCR.

2. Existing Methods in the Field

A significant amount of research has been done to investigate both generalized OCR and convolutional neural networks.

2.1. Past Research in CNNs

Krizhevsky *et al.* [8] introduced a massive scale convolutional neural net built to classify the then-largest dataset of labeled images, the ImageNet LSVRC-2010, into 1000 different classes. The CNN built to perform this task was comprised of 60 million parameters and 650,000 neurons organized into five convolutional layers and three fully-connected layers terminating with a 1000-way softmax. Due to the large scale of this neural net, a number of innovative optimizations were made to reduce training time. Namely, non-saturating neurons were employed along with convolution performed by a GPU operation. Results from this neural net structure were positive, yielding better error rates than previous state-of-the-art classification schemes.

Moreover, previous work in the field of object detection has led to insights regarding the optimal structure of a CNN. Simonyan and Zisserman [11] evaluated the effect changing the depth of a CNN has on accuracy. They tested the performance of their CNNs on the ILSVRC-2012 dataset, which is comprised of images in 1000 different classes. The CNNs they built had a stack of convolutional layers followed by three fully connected layers. The first two fully connected layers had 4096 channels each, and the last fully connected layer had 1000 channels, one for each output class. The last layer of the CNN was the softmax layer.

During their experiment, they varied the number of weight layers from 11 (8 convolutional and 3 fully connected) to 19 (16 convolutional and 3 fully connected). They also varied the receptive field from 1x1 to 3x3. From these experiments, they found out that 16-19 weight layers gave the highest accuracy. Secondly, neural nets with 3x3 receptive fields outperformed those with 1x1 receptive field, indicative of the importance of spatial context. Their findings shed light on the optimal configurations for CNNs.

Lee *et al.* [10] introduced an important algorithm for CNNs called probabilistic max-pooling, which reduces the dimensionality of the representation of higher layers, while still keeping information intact with a high probability. Essentially, max-pooling works by computing the maximum activation value for each subregion of the input image. By throwing away values that are not maximal, the technique reduces the computational complexity of the algorithm. Moreover, max-pooling makes the representation of a higher layer more resistant to small translations of an input, making the CNN more robust. For the MNIST data set, the authors obtained a 0.8% test error.

2.2. Methods researched for OCR

Dalal and Triggs [4] experimented with new method to create feature sets, which is useful for OCR. Specifically, they created HOG (histograms of oriented gradient) descriptors, which they showed to significantly outperform currently used feature sets. They performed their experiment by testing types of HOG (rectangular and circular HOG) against other methods, such as the Haar wavelet, PCA-SIFT, and shape context approaches, and found that HOG performed much better on the MIT pedestrian database, as well as on their own INRIA database. Part of their experiment included optimizing HOG by means including realizing the importance of high-quality local contrast normalization before using HOG.

Jimenez and Nguyen [7] explored problems in the field of handwriting recognition. Their research aimed to convert handwritten mathematical notes and symbols to LaTex. Their approach involved using Pyramids of Oriented Gradients (PHOG) in conjunction with an ensemble of one-against-one of SVM classifiers. This approach to be proved effective with respect to the task of recognizing mathematical symbols. The results of their experimentation, trained on the 22,000 character sample CHROHME dataset and tested on a small sample of 75 handwritten mathematical symbols, yielded promising accuracy rates of 92%.

A team from Google Research recently published a paper on a modern-day application of generalized OCR: Google Street View. Goodfellow *et al.* [6] constructed a deep convolutional neural network to recognize multi-digit numbers from Google Street View data. Using a highly parallelized deep neural network to perform this computationally intensive task, they created a network that, after eight days of training achieved 90% accuracy on Google Street View Image Data. The primary approach taken was to train a probabilistic model of letter sequenced given images. This contrasted with previous work, which would attempt to isolate each digit before assigning predictions. The results from this project indicate that deeper neural networks, when adequate computational resources exist to support training, produce higher accuracy and generalization scores compared to shallower networks with fewer layers.

3. System Design and Approach

Our primary approach to generalized OCR was to conduct a survey of multiple machine learning models and compare the performance on each. As a starting point for our data collection we used the MNIST handwriting recognition dataset. Digit recognition is a problem well-solved using a convolutional neural net(CNN) approach. This is largely due to the uniformity of the dataset– all input digits in the MNIST dataset belong to one of 10 (0-9) classes and appear as unobscured black digits on a white background. In contrast, the generalized character recognition dataset we used consisted of characters in 62 distinct classes which were often obscured, illegible, or otherwise obfuscated such that reading each character needed to be done in a more generalized manner than with MNIST.

The three approaches to generalized character recognition that we chose to implement were (1) Multiclass Logistic Regression, (2) Artificial Neural Networks, and (3) Convolutional Neural Networks. These approaches each built upon each other by adding successive layers of complexity to character classification, and a comparison of the results obtained under each approach provides insight into the benefits and drawbacks of each approach. We used the logistic regression model and the ANN as benchmarks to compare the performance of the CNN. The system flow of our project is detailed in Figure 2.

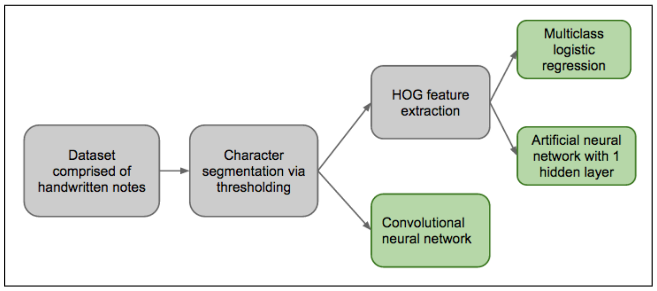
Dataset comprised of handwritten notes
Character segmentation via thresholding
HOG feature extraction
Multiclass logistic regression
Artificial neural network with 1 hidden layer
Convolutional neural network

Figure 2. System flow chart detailing approach.

We started out with a dataset of handwritten class notes, and then applied a character segmentation algorithm. The output of this algorithm was a set of characters, which were then classified with a logistic regression model, ANN, and CNN. Both logistic regression and the ANN used HOG feature extractions to compute features, while the inputs to the CNN were the images themselves.

4. Algorithms

4.1. Character Segmentation

For images which mostly consist of text on a plain background, we implemented a character segmentation algorithm based roughly on the work of Jimenez and Nguyen [7]. This particular implementation works for usage cases such as for detecting characters written on paper or on a blackboard, or for identifying the characters in a CAPTCHA.

In the first step, the algorithm blurs the image using a Gaussian blur. This helps reduce the effects of noise. Next, the image is adaptively thresholded to detect which parts of the image might be letters. Normally, thresholding consists of finding which pixels have colors above (or below) a certain threshold and marking those as the foreground and everything else as the background. Unfortunately, this is sensitive to changes in illumination. To combat this, we apply adaptive thresholding, which compares each pixel to a central measure of its neighboring pixels. After adaptive thresholding, the foreground pixels are dilated in order to connect potentially disconnected character parts. Finally, a connected component search (using contour-finding) is used to segment the individual characters. An example of this process can be seen in Figure 3 and another sample result can be seen in 4.

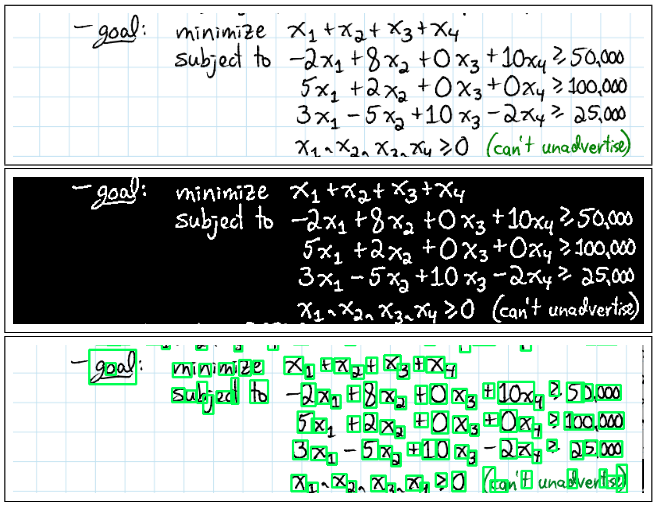

Figure 3. The top image is a page from MIT lecture notes, handwritten, scanned, and published online. The second image shows the result after applying blurring and adaptive thresholding. The last image shows the result of the segmentation.

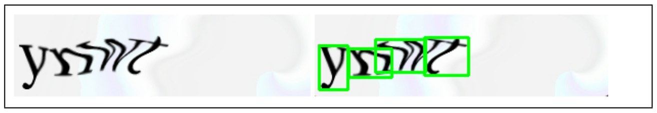

Figure 4. An example of the results of character segmentation on a CAPTCHA image. Segmenting the CAPTCHA uses different parameters from segmenting other types of images.

As a proof of concept, we used this character segmentation algorithm to segment characters from our own pictures, such as of school notes, and used our classifiers to classify them.

4.2. HOG Feature Extraction

Certain character classification methods required us to generate feature vectors from the character samples. For this, we used histograms of oriented gradients (HOG) [4]. This involved dividing the image into 16x16 cells. Over each cell, a histogram of the gradient orientations is computed. Finally, the histograms are concatenated together in order to form the feature vector.

Because the training and testing images came in different sizes, the feature vectors from HOG would not be of the same length without image preprocessing. To solve this problem, black strips were added to make each input image square. The number of cells in the HOG representation were kept constant at 144 cells across all the now square images, which yielded feature vectors of the same length (1152).

We chose to use HOG for its simplicity and its usefulness in object recognition. HOG focuses on gradient orientations, which are intuitively useful for distinguishing between characters. While HOG has often been used to classify more concrete objects such as humans [4], there have also been studies that have used variants of HOG for character recognition. Su *et al.* [12] used a variant of HOG for character recognition in natural scenes. Advantages of using HOG include its invariance to local geometric and photometric transformations. The former invariant might be useful for text because different peoples handwritings or different fonts may use different angles when writing characters, and HOG would be able to withstand these transformations.

An example of HOG is shown in Figure 5.

One possible optimization, mentioned in research [4] but not done here, is to contrast-normalize all the images before computing the HOG feature vectors. This would have helped decrease the impact from illumination effects.

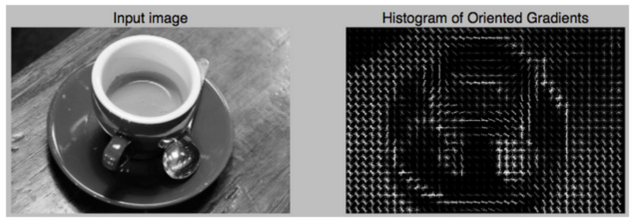

Figure 5. A representation of histogram of oriented gradients computation. The coffee mug image was divided into 16x16 cells, and within each cell a histogram of gradients was computed. Within each cell, each of the gradients seen within that cell has a line segment drawn with that slope.

4.3. Logistic Regression

A logistic regression function is a linear classifier with two parameters: a weight matrix W and a bias vector b. The softmax function determines the most likely class label of an input by computing probabilities of input point placement in each possible subcategory and assigning the input point to the class of which it is most likely to be a member. More formally, we see that the probability that an input x is a member of class i is:

$$\begin{aligned}\Pr(Y = i|x, W, B) &= \text{softmax}_i(Wx + b) \\ &= \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}}\end{aligned} \tag{1}$$

As such, the logistic regression function can be thought of as a simple network with two layers: an input layer that processes the data, and an output layer that applies the softmax function to classify the data.

4.4. Artificial Neural Network

An artificial neural network (ANN) takes a logistic regression classifier and adds an extra layer of complexity. Because logistic regression works best when the inputs are completely linearly separable, the ANN generalizes this process and works on non-linear inputs. Using a single hidden layer, the ANN transforms the input layer with a non-linear transformation, which makes the input linearly separable. This transformation is known as an activation function, and popular choices for the activation function include tanh and the logistic sigmoid function. A depiction of an ANN is shown in Figure 6.

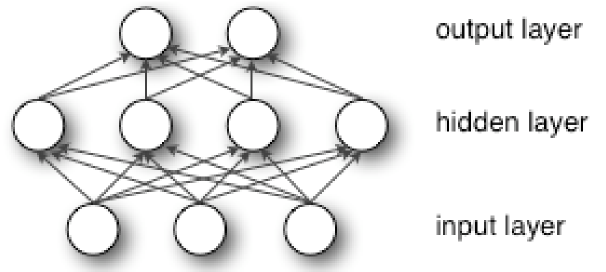

Figure 6. The second layer of this ANN applies an activation function to make the input data linearly separable

The ANN trains on input data, and outputs a vector of class probabilities for each input. Formally, we can write:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))) \tag{2}$$

In the above formula, f(x) is the output vector of class probabilities, s is the activation function, and G is the softmax function.

4.5. Convolutional Neural Network

A convolutional neural network is built by adding at least one convolutional layer before the layers of an artificial neural network. A convolutional layer has a set of linear filters, which are used to obtain feature maps. A feature map is created by applying an activation function to the convolution of subregions of the input image with a linear filter, which is determined by two parameters: W and b.

More formally, given h^k, the kth feature map at a convolutional layer, with filters defined by weights W^k and biases b_k, the following formula holds at pixel coordinates i and j with activation function tanh:

$$h^k_{ij} = \tanh((W^k * x)_{ij} + b_k) \tag{3}$$

The neuron outputs at each layer are then computed from these feature maps, which are analogous to the feature vectors used in simple machine learning models.

Additionally, CNNs typically have maxpooling layers between convolutional layers. The pooling layers take the maximum value from a pxp sub-region of the image in order to reduce the dimensionality of an image. This reduces the computational complexity of the higher layers, and therefore also the computational time. Maxpooling also lends translational invariance to the CNN, which helps make it more robust.

One advantage of the CNN architecture over that of alternate neural network architectures is the capability to take advantage of the 2D structure of input data. Because the feature maps of the CNNs are computed from a 2D convolution, the inherent two-dimensional structure of the input images to be classified is taken into account.

5. Experiments and Results

5.1. Character Segmentation

On a hand-created dataset based on one page of notes from MIT's intermediate algorithms class, the character segmentation algorithm successfully detected and separated 47.7% of the characters on the page. Almost all of the characters were found in some way, but many of them were incorrectly merged with other characters. These were not counted among the successes. The images shown in Figure 3 was part of this dataset.

One part of the experimental design was balancing the dilation step, which is both helpful in merging two parts of a single character but has the downside that some characters that should not be together are combined. As such, our experimental results showed that our character segmentation method often failed to find smaller characters, and sometimes connected two separate characters.

5.2. MNIST

As a proof of concept, we first tested the performance of a convolutional neural network on the MNIST dataset. The MNIST dataset of handwritten digits features segmented handwritten black-and-white digits, all size-normalized and centered [9]. It is a very simple and relatively easy dataset that we used for the initial testing of our CNN performance. The original convolutional neural network paper was tested on this dataset, and by implementing the CNN architecture from that paper[9], we achieved an error rate of 0.92%. It is not unexpected that the error rate is so low because the dataset is very easy. There is no illumination or color effects, all the digits are centered, and everything is already segmented.

The architecture we used was comprised of multiple different layer types. The input layer took in a 28x28 handwritten digit image. This was then fed through a convolutional layer followed by a max-pooling layer. This was then fed through another convolutional layer followed by another max-pooling layer. Finally, there was a fully-connected layer whose outputs were classified using logistic regression. Both convolutional layers used 5x5 feature maps, and the max-pooling layer always cut the image dimensions in half. The cost minimized was the negative log-likelihood of the model. This architecture came directly from the paper [9].

5.3. Overview of Designing Neural Networks for OCR

When designing the ANN and CNN architecture, a general approach was followed. We sought to increase the complexity of the neural network as much as possible, which often resulted in a high training accuracy and a low testing accuracy. Then, we worked on decreasing overfitting as much as possible, until the testing accuracy was almost as high as the training accuracy. We continued this cycle, increasing complexity and then decreasing overfitting, until the testing accuracy was maximized. To increase neural network complexity, we added more layers and increased the number of nodes per layer. To decrease overfitting, we increased the regularization parameter, decreased the learning rate, and removed unnecessary layers.

We tested our neural networks on two datasets: one with handwritten characters, and one with characters taken from nature photographs [5]. For examples of each, see Figures 7 and 8.

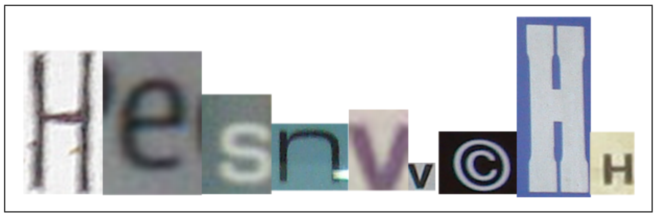

Figure 7. Training dataset 1 - characters taken from natural photographs - H, e, s, n, v, V, C, H, H.

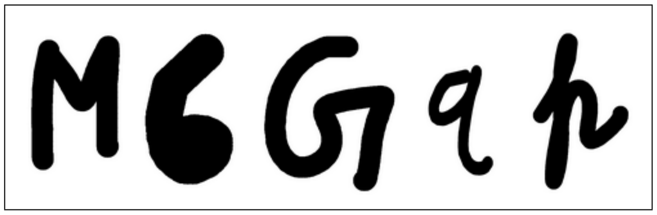

Figure 8. Training dataset 2 - handwritten characters - M, 6, G, q, h.

Because we did not have access to a GPU, testing each neural network architecture took 6 to 10 hours each, and we tested a total of 50 different architectures.

5.4. Benchmark – Logistic Regression

Before experimenting with an artificial neural network and a convolutional neural network, we tested the two character datasets on a multiclass logistic regression model. We used HOG to extract feature vectors, and the training and testing set had 3,852 feature vectors with 1152 features each. Each of the 62 different classes were equally distributed among the training and the testing set. On the handwritten character dataset, the logistic regression performed at a 52.6% accuracy, and on the photographed character dataset, it performed at a 53.9% accuracy.

5.5. Benchmark – Artificial Neural Network Experiments

As the initial design of our artificial neural network had a high training accuracy but low testing accuracy, we focused our design efforts on correcting for overfitting. Four different architecture components were tested to adjust for overfitting. Firstly, the L2 regularization parameter was increased from 0 to 0.01. Secondly, three different activation function for the fully connected hidden layer were compared: logistic, tanh, and rectifier. We also varied the number of hidden layers in the ANN from 1 to 3. Finally, the number of nodes in each hidden layer of the ANN was increased from 5 to 100.

Given D, the length of the feature vector, and F, the number of classes, the final architecture we settled on was the following: input layer with D nodes, fully connected hidden layer with 100 nodes and logistic activation function, output layer with F nodes and softmax activation function.

We used this final architecture to test performance on the handwritten and photographed character datasets. On the handwritten character dataset, the ANN performed at a 55.5% accuracy, and on the photographed character dataset, it performed at a 63.5% accuracy.

5.6. Convolutional Neural Network Experiments

The training process used to construct the Convolutional Neural Nets included 7705 training points representing 62 classes. These images were split into training and test sets with 3852 images each.

A number of different CNN architectures were constructed. These network architectures varied the type, number, node density, and order of respective layers, as well as values of regularization and feature map parameters.

After resizing all the images to 40x40, the final architecture of the CNN was as follows:

1. 40x40 input layer
2. Convolutional layer with 20 feature maps of size 7x7
3. Maxpooling layer with 2x2 pool size
4. Convolutional layer with 50 feature maps of size 7x7
5. Maxpooling layer with 2x2 pool size
6. Fully connected hidden layer with 500 nodes
7. Softmax output layer

This architecture worked out well, as some preliminary calculations show below:

- The filtering in the first convolutional layer reduces the size of the image to (40-7+1 , 40-7+1) = (34, 34).
- The next maxpooling layer reduces this image size more to (34/2, 34/2) = (17, 17).
- So the 4D output after the first convolutional layer and first maxpooling is (batch size, 20, 17, 17), where 20 is the number of feature maps in the convolutional layer.
- The filtering in the second convolutional layer reduces the size of the image to (17-7+1, 17-7+1) = (11, 11).
- Then, the second maxpooling layer reduces the size to (11/2, 11/2) = (5, 5).

As such, the dimensionality has been reduced enough for the output softmax function to handle easily.

The other parameters that were trained were: a learning rate of 0.1, a batch size of 500, and 1000 iterations.

Using this architecture, the CNN yielded an accuracy score of 66.4% on the handwritten character dataset, and an accuracy of 64.6% on the characters in natural photographs dataset.

The final results of our experiments are shown in Tables 1 and 2. For our implementations of convolutional neural networks, we used the Theano library [1] [2] [3].

Method	Accuracy
Logistic regression	52.6%
Artificial neural network	55.5%
Convolutional neural network	66.4%

Table 1. Accuracies on images from handwritten character dataset.

Method	Accuracy
Logistic regression	53.9%
Artificial neural network	63.5%
Convolutional neural network	64.6%

Table 2. Final accuracies for characters taken from natural photographs.

The manner in which training and testing were conducted involved the use of randomized training and test sets

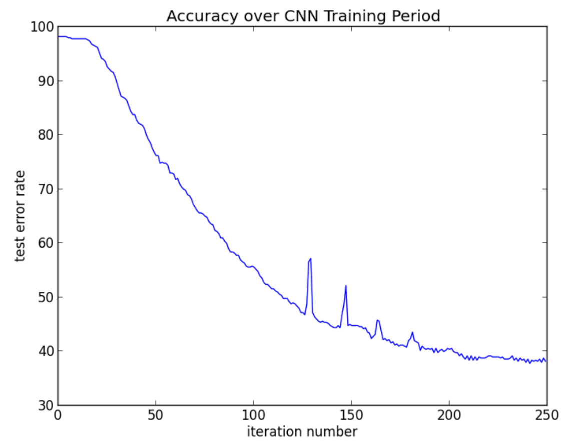

Figure 9. Test error with increasing training iterations on input images from the natural photographs dataset.

with cross-validation. Training occurred as a series of 500 epochs, each of which was comprised of several training iterations.

Regularization parameter increases appear to have a negative impact on training accuracy, but positively impact test accuracy to a point(the optimal value was found to be slightly above 5) before again decreasing accuracy scores. With increases in feature map size, signs of overfitting begin to show with feature map sizes larger than 10x10. At this point, training accuracy continues to increase, but test accuracy begins to drop from the overfitting.

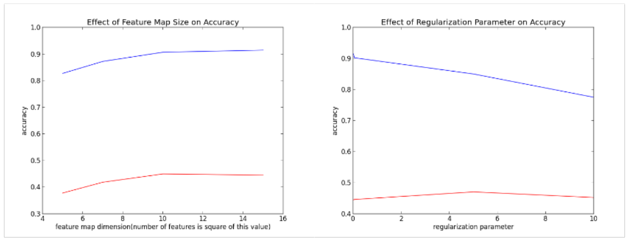

Figure 10. Training(blue) and test(red) accuracy variations with changes to regularization (top) and feature map size (bottom) from tests on generalized character recognition.

See Figures 9 and 10 for graphical depictions of our experimental results.

5.6.1 Convolutional Neural Network Parameters

A variety of parameters were considered in our performance optimization experiments. At the output layer, choice of an algorithm to use for the convolutional layer had a large effect on the resulting classification accuracy of the neural net. Softmax was by far the most successful algorithm for this task, yielding test accuracy scores almost 3x higher than those achieved by simply using logistic regression algorithms on otherwise identical CNN architectures. Logistic regression was, however, quite useful as a probabilistic classifier for assigning final classification weights and assigning inputs to the most likely class.

Another parameter we considered was the number of feature maps in each layer. Our experiments showed optimal generalization at around 40 feature maps per layer, with increasingly large featuremap densities leading to overfitting.

5.7. Discussion and Conclusion

Our survey of approaches to generalized character recognition identifies the successes and shortcomings of several approaches to this unsolved problem. In this survey, we aim to understand the strengths of each approach to better understand where gains may be made in solving this problem. Specifically, by identifying the parameters that most contribute to success at the task of generalized character recognition, we gain insight into why this problem remains unsolved.

Our results are in line with current state-of-the art results for generalized character recognition, which is an important yet unsolved problem with many interesting applications. In addition to straightforward applocations such as captcha-breaking, handwriting recognition and transcription, and document scan-to-text conversion, the problem of generalized character recognition has applications in cutting-edge fields from medical devices for the blind to dynamic visual-world based adaptation capabilities for autonomous vehicles.

References

[1] Convolutional neural networks (lenet). `http://deeplearning.net/tutorial/lenet.html`. Accessed: December, 2014.

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 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] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005*, pages 886–893. ACM, June 2005.

[5] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.

[6] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks.

[7] N. D. Jimenez and L. Nguyen. Recognition of handwritten mathematical symbols with phog features. `http://cs229.stanford.edu/proj2013/JimenezNguyen_MathFormulas_final_paper.pdf`, 2013.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. NIPS, Dec. 2012.

[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, November 1998.

[10] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning*. ACM, June 2009.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Large Scale Visual Recognition Challenge 2014*, Aug. 2014.

[12] B. Su, S. Lu, S. Tian, J. H. Lim, and C. L. Tan. Character recognition in natural scenes using convolutional co-occurrence hog. http://www.comp.nus.edu.sg/ tians/papers/ICPR2014$_{C}onvolution-CoHOG_{S}u.pdf$.