Recognizing Handwritten Digits and Characters

Vishnu Sundaresan
Stanford University
vishnu@stanford.edu

Jasper Lin
Stanford University
jasperlin@stanford.edu

Abstract

Our project is meant to implement a full pipeline for recognizing the text of any input image. We separate our project into two parts, the segmentation of an image into individual characters, followed by classifying these images into their respective character labels. Our approach builds upon Lecun et. al. in digit recognition, and applies the techniques to characters recognition (a-z, A-Z, 0-9). In particular, we use architectures involving shallow as well as deep neural networks that expands on the classifiers trained by de Campos et. al.

1. Introduction

Our project tackles the problem of character classification in natural handwritten images. Previous work in the area include utilizing several supervised learning algorithms such as K nearest neighbors, support vector machine, and stochastic gradient descent to classify the images. They do not create any two layer or deep convolutional neural networks, so this project extends their work by testing several architectures in an attempt to decrease the classification errors.. LeCun et al. (1998) on the other hand does use several Convolutional Neural Networks (CNNs) for their digit recognition work, giving us a basis off of which to refer to and apply it to a more complicated problem. In addition to the recognition of handwritten characters individually, we create a pipeline that allows any image of handwritten text to be passed in and segmented into separate characters. Our final project will allow essentially any image of a document or note to be segmented and translated to a digitized version.

2. Background and Related Work

2.1. LeCun et. al. (1998)

This paper is the original inspiration and basis for the improvements in accuracy we made to the character dataset. The paper describes the process they used to achieve up to a 99.1% accuracy on the MNIST dataset, using both a 3-layer convolutional network and a 5-layer network that failed to outperform the former. Additionally, the paper discusses the problem of segmentation, and how it cannot be decoupled from the recognition of isolated characters. Essentially, making the decision to segment an image with multiple characters before the recognition of individual characters is not optimal, as the process should be parallelized to test multiple hypotheses at the same time. In terms of feature extraction, LeCun et. al. argues that humans cannot possibly capture all relevant information from images and even coming close requires expert knowledge on the subject, so instead they resorted to utilizing Gradient-Based Learning to learn the useful features for the images.

2.2. De Campos et. al. (2009)

This paper introduces the Chars74K dataset that we used to train and evaluate our convolutional network. While the authors here did not implement conv nets for their own problem, we will compare our results using the techniques and architectures described in LeCun et. al. in order to outperform the results presented in this paper. Some of the features that DeCampos et. al. use to train their classifiers include geometric blur, spin image, as well as affine transformations. With a maximum accuracy of 55.25% achieved by the paper, we can immediately see that the problem of character recognition in addition to simply digits is a much harder problem that potentially requires a different approach.

2.3. Choudhary (2014)

The second part of our proposed problem is extending our solution from beyond just individual characters and recognizing words and multi-digit numbers. The classical approach to optical character recognition (OCR) is finding the next character image, extract distinguishing attributes of the character image, and matching the given symbol to a best match and outputting its identity. With the previously designed neural network from the first part of the problem, we will already have the solution to the last part of the algorithm.

From reviewing literature, there are three main strategies for segmentation, plus numerous hybrid approaches. These three approaches are dissection, recognition based, and holistic methods. Dissection is the process of cutting the image into meaningful components and passed individually into our classifier. When using dissection, the segmentation becomes the most crucial step in the recognition process. By far the most common approach used by researchers, this approach tries to find the presence of ligatures (interconnections between characters) and cut the word image through these ligatures. Recognition based segmentation searches the image for components that match a predetermined alphabet. These approaches take advantage of the Hidden Markov model (HMM), bypassing the need for complex dissection algorithms. This approach has also been called "segmentation-free" recognition.

3. Technical Approach

3.1. Character Recognition

For training on the MNIST dataset, we will implement different neural networks and convolutional neural networks architectures and compare the accuracy of the different architectures. The large collection of examples and potentially complex characters due to variability in handwriting makes using multi-layer networks trained with gradient descent an obvious candidate for recognition tasks. LeCun et al. (1998) implemented a seven layer convolutional neural network not including the input layer that had an error rate of 0.9% on the MNIST dataset. We will try to at least meet this error rate and possibly improve on the result by implementing newer concepts like dropout in our network. Like LeCun et al., we will also implement a basic linear classifier and k-nearest neighbors classifier to establish a minimum baseline for our neural networks. We expect the digits classifiers to outperform the characters classifiers due to only 10 different classifications instead of 62.

Moving on to the character dataset, we plan on implementing the supervised classifier models used in de Campos et. al. (2009) and attain a similar accuracy, as well as incorporate the techniques used in LeCun et al. with respect to shallow and deep convolutional neural networks. In particular, we can implement the convnet architecture used in the Lecun et al. paper, use transfer learning with a larger network, as well as implement and train a common architecture discussed in class.

3.2. Convolutional Neural Networks

Our first goal was to use the same LeNet architecture used for training and evaluating the MNIST dataset. This architecture consisted of a (conv -> pool)x2 -> FC -> relu -> FC -> Softmax (Figure 1 on next page) convnet that is 2 layers deep, meant to recognize characters from the 10 digit classes. Next, we chose to try a deeper, 8 layer AlexNet, which would solve some of the problems related to the shallower LeNet meant to classify only 10 digits. Due to the large training time associated with such a large network, we used transfer learning and fine-tuning to use the weights of the pretrained AlexNet, and input our own character images. We realize that the hyper parameters and dropout rates used in AlexNet were likely not optimal for our Chars74K dataset, so we tuned the learning rate, momentum, and dropout percentage with a smaller set of images, as well as between layers of the convnet.

Finally, we came to the realization that the smaller LeNet could be improved in order to better recognize the larger number of classes, as well as the larger AlexNet was unnecessary and over fitted the training data due to the fewer number of features associated with characters as opposed to natural images. We then created our own network that was based from models discussed in class, specifically the (conv -> relu -> pool)x3 -> FC -> Softmax architecture (Figure 2 on next page). As before, we tuned the hyper parameters, experimented with dropout rates, and used data augmentation as well as rotation to best fit the model to our dataset and capture the relevant features.

3.3. Segmentation

3.3.1. Image Preprocessing

Slope and slant correction of handwritten words are necessary to reduce the variations in handwriting styles. Careful estimation of the slope and slant can make the following segmentation process much simpler. In the ideal scenario, a word is written horizontally with ascenders and descenders aligned in the vertical direction. However, this is rarely the case. Slope is defined as the angle between the horizontal direction and the direction of the line the word is aligned. Slant is defined as the angle between vertical direction and the direction of strokes supposed to be vertical. Before segmentation occurs, we should work to eliminate both of these angles.

To handle the slant estimation, we referred to the work done by Papandreou et. al. First, the word's core region is found using black run profiles. From there, we can divide the core region into vertical strips and determine the centroid for each region and fit a straight line through the centroids. In addition, this calculation also provides an estimation of the average thickness of the stroke, which will be used in slant correction.

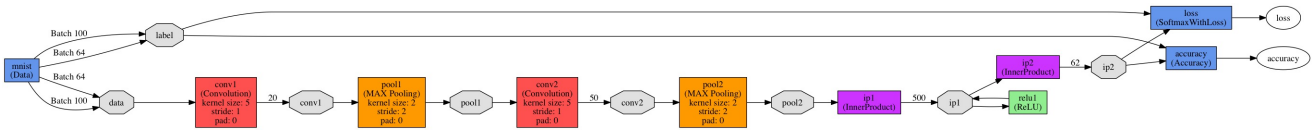


Figure 1 - LeNet Architecture

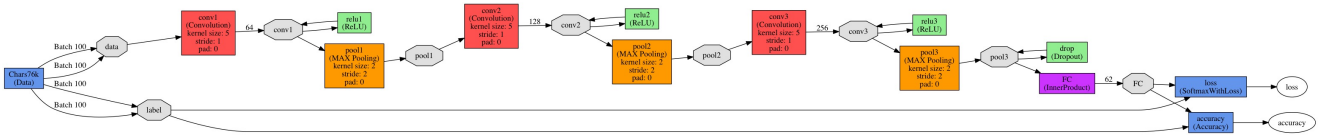
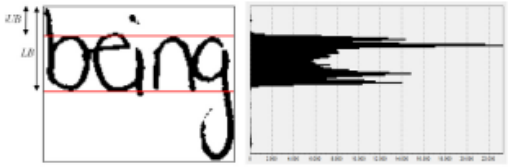


Figure 2 – Our Architecture



a **b**

Figures A and B – stroke thickness

Given a binary image, the core-region of a word is defined as the region between the upper baseline and the lower baseline (Figure A). To detect the core region, we count the number of horizontal black runs in each line. The horizontal black run profile H(y) is defined as follows:

$$H(y) = |B(y)|^2 \sum_{i=1}^{B(y)} \sum_{j=0}^{L(iy)} j$$

Where B(y) is the number of black runs in horizontal line (y) and L(i,y) is the length of the ith black run of line y. We can convert this value to a Boolean horizontal profile as follows:

$$H_B(y) = 1 \mid H(y) > T_h$$

$$T_h = \frac{0.15}{I_y} \sum_{y=0}^{I_y - 1} H(y)$$

The result of this thresholding is shown in Figure B. Lastly, we can determine the upper baseline (UB) and lower baseline (LB) as:

$$\{UB, LB\} = \{s_k, e_k\}$$

$$k = \arg\max_i \left(\sum_{y=s_i}^{e_i} H(y) \right)$$

Through running these computations, we can also determine the character stroke width by determining the modal value of the lengths of horizontal black runs.

Next, we can correct the slant in a word by convolving Gabor filters with our images. We can use the Gabor filter to create different filters corresponding to different length scales and orientations to determine the slant in the word.

A Gabor filter is a linear filter that is often used for edge detection and is the product of a Gaussian function and a harmonic function:

$$G(x, y; \lambda, \theta, \sigma_x, \sigma_y, \phi) = \exp\{-.5(\frac{x_\theta^2}{\sigma_x^2} + \frac{y_\theta^2}{\sigma_y^2})\} \cos(2\frac{\pi}{\lambda} x_\theta + \phi)$$

$$\sigma_x = \frac{\lambda}{\pi} \sqrt{\frac{\log(2)}{2}} [1 + 2^{BW}]$$

$$\sigma_y = \frac{\sigma_x}{aspect\ ratio}$$

$$x_\theta = (x \cos\theta) + (y \sin\theta)$$

$$y_\theta = (-x \sin\theta) + (y \cos\theta)$$

λ is equivalent to the height of the core region of the word and BW, the bandwidth, is assumed to be 1. We test different Gabor filters from ±60 degrees for each word.

a. b.

c.

Figure C – Gabor filter correction

3.3.2 Word Segmentation

We chose to use a vertical projection of each image to determine where the segmentation points in the word are. The vertical projection is determined by the sum of all the white pixels along a line in the vertical direction, accomplished by passing the average thickness of each stroke to the algorithm and using that as the threshold to determine cutoffs. If, at any point in the image, there are less black pixels in one vertical run than the stroke thickness, this is likely the tailing end of a character or white space. This technique has been previously explored by Jagannathan et. al (2013) on datasets of license plates.

4. Experiments

4.1. Dataset

We are currently using the MNIST handwritten digit database for the digit recognition, and the Chars74K dataset for character recognition (sample characters shown in Figure 3). All images will be of size 28x28 (256x256x3 for the character dataset), and we will use transfer learning to train a neural network on the smaller number of digits classes before training on the character dataset. We have split each dataset into a train and test portion, as well as running cross validation on 10% of the train set in order to tune our hyper parameters and determine which model works the best.

Figure 3 – sample images from Chars74K dataset

In addition, we will implement segmentation algorithms and our trained CNN on the Street View House Numbers (SVHN) dataset to test the recognition of numbers from a harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

4.2. Expected Results and Evaluation Criteria

We will evaluate our results based mostly on the accuracy of character recognition for each of the different algorithms we implement. Qualitatively, this will translate these different accuracies and plot them with different constraints such as size of training and test sets, learning over time, and limiting to lowercase/uppercase alphabetic characters. For each of our different models, we will evaluate both a training error that utilizes cross validation, as well as the test error on a separate set to avoid over fitting our model. For the convolutional neural network architectures that we are experimenting with, we will also take into account the performance as a possible factor given the time period of our project.

4.3. Linear Classifiers

We created a linear classifier that utilizes stochastic gradient descent, taking in 30000 images from the MNIST dataset and training the model with each image's respective digit label. We then took a different set of 5000 images also in our dataset as the test set, and calculated the accuracy on both the train and test set of data. We achieved a train accuracy of **88.03%**, and a test accuracy of **88.69%.** This is a reasonable baseline for us to begin with, with the possibility of increasing this accuracy. In terms of the Chars74K dataset, we split the total of 8000 naturally occurring character images and split them into 6400 train and 1600 test images. This resulted in a test accuracy of **30.15%**, clearly far below that of the MNIST dataset.

4.4. K-Nearest Neighbors

We created a k-nearest neighbors classifier that computed the nearest 100 neighbors to each image, and classifies the point based on those neighbors. Similar to the linear classifier described above, we first fit the model using 30000 training images and labels, before testing on a different set of 5000 images. We achieved a train accuracy of **93.73%**, and a test accuracy of **93.28%.** Likewise for the Chars74K dataset, we were able to achieve a test accuracy of **35.47%**. Comparing these results in the linear and KNN classifiers to that in DeCampos, our lack of extensive feature extraction besides augmenting images and rotating to correct for slant can clearly be seen. Each of our test accuracies underperformed their respective counterparts by around 10%, leaving a huge room for improvement when implementing our convolutional networks.

4.4. LeNet Architecture

The most similar problem to the one of character recognition is that of digits, which was tackled by LeCun with respect to the MNIST digits dataset. The convolutional network presented in their paper is a 2-layer network, tuned and fit to best represent the 10 classes of digits in the dataset. We can see from Figure 4 that the training and test error quickly reach a maximum of **99.05%** accuracy in 10000 iterations for the MNIST dataset. When using this same architecture (Figure 1) and passing in the Chars74K dataset as input instead, we get an accuracy of **45.36%,** slightly better than the linear and KNN classifier results above. The two-layer network without any dropout is clearly is not deep enough to capture all the features of each class of character, suggesting that a deeper network is necessary to best solve this problem

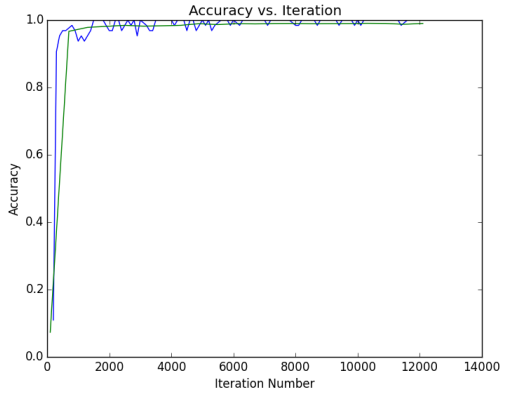


Figure 4 – MNIST accuracy
(train = blue, test = green)

4.5. AlexNet Architecture (Transfer Learning)

Going off the realization that the LeNet architecture was too shallow and simple to fit the character dataset, we looked next to use transfer learning and fine-tuning to use the pretrained AlexNet model and utilize it for our character recognition. Figure 5 below shows the plot of training and test accuracy over 10000 iterations with weights being initialized to that of the pretrained AlexNet. The test accuracy a peak of **63.38%,** but even with a dropout ratio of 0.8 the model is clearly overfitting our training data heavily. This suggests that the convnet is larger than necessary, and that a smaller network may result in a higher accuracy given that the problem of overfitting can be resolved.

Another observation we made is that while pictures of naturally occurring objects and creatures may have many aspects that change from image to image, something that AlexNet is meant to work well with, characters are less diverse and follow some of the same patterns regardless of how they occur. With such a deep network, certain features of characters in the training set could have been learned that are not indicative of the class overall, leading to overfitting and misclassification of test data.

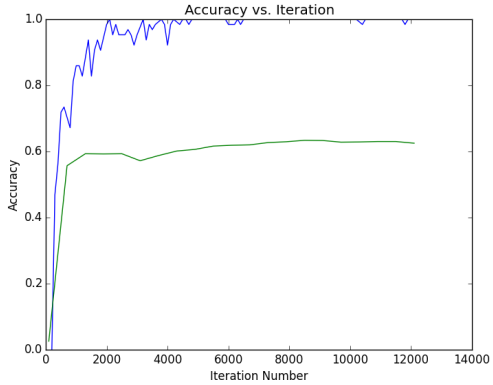


Figure 5 – Chars74K using AlexNet architecture
(train = blue, test = green)

4.6. Three-Layer Custom Architecture

We chose to implement the 3-layer architectures in Figure 1, and trained it using the Chars74K dataset. In tuning the hyper parameters, we first began with a coarse granularity search for an appropriate learning rate. Figure 6 shows our experiment, with the loss decreasing most rapidly around a learning rate of up to 0.001, and becoming infinite at a higher rate.

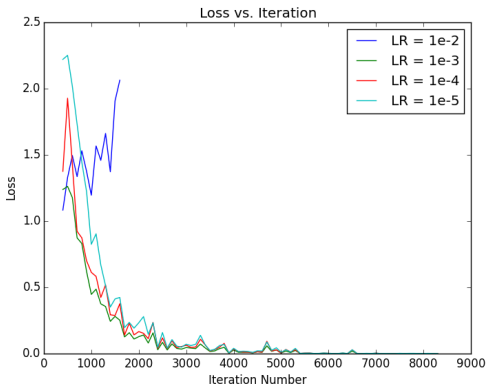


Figure 6 – Learning Rate Tuning

In order to augment and extract the most useful features from our dataset, we used Caffe's built in data augmentation ability, as well as creating additional images that accounted for their slant using techniques incorporated from the segmentation aspect of our project. Initially, training our convnet for 10000 iterations produced a similar result to that of the fine-tuned AlexNet (Figure 7 below), but without any dropout layers implemented.

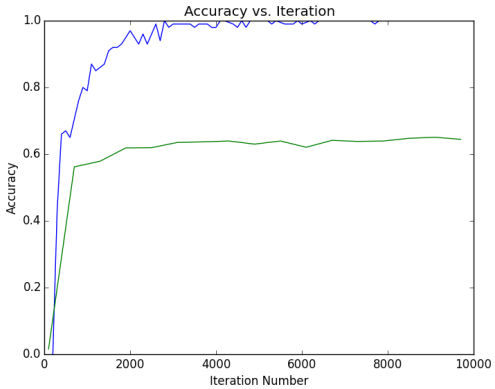


Figure 7 – Dropout Rate = 0%

We then added a dropout layer before our fully connected layer, and slowly increased the number of neurons being zeroes out until the training accuracy began to align with the validation accuracy (Figures 8 and 9).

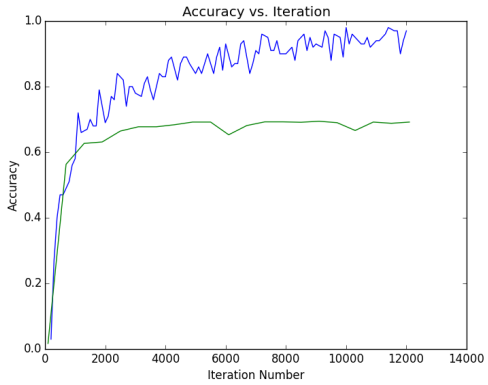


Figure 8 – Dropout Rate = 50%

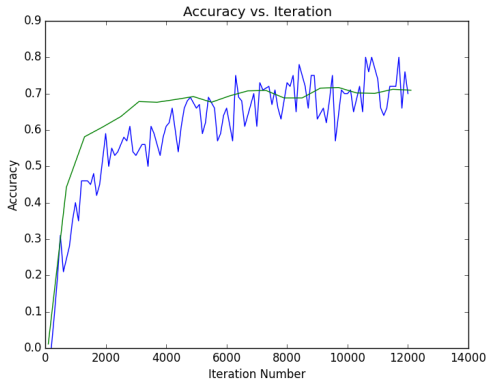


Figure 9 – Dropout Rate = 80%

Finally after close tuning of the learning rate, momentum, dropout percentage, and appropriate data augmentation, we achieved a maximum test accuracy of **71.69%**, over 15% higher than the best results obtained by the DeCampos paper. Below (Figure 10) are some of the weights outputted from the first convolutional layer. There is a clear structure to these weights, and the shapes of certain characters can even be discerned, reinforcing the notion that characters of the same class have very similar structure and features. Additionally, if we use this same architecture and train the convnet using the MNIST digits dataset, we do not reach the same accuracy as the LeNet architecture, thus supporting our observation that certain architectures lend themselves better to certain problems.

4.7. Segmentation

The first step to processing images in order to get reasonable segments is the slant correction. Figure 11 shows a few examples of slope and slant corrected images

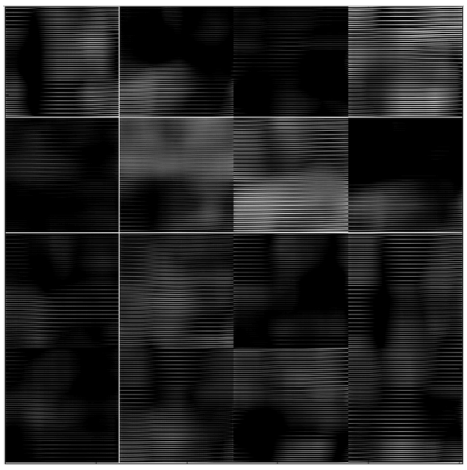


Figure 10 – Learned Weights
(First conv layer, first 16 outputs)

from the IAM dataset. These applied corrections are essential to the segmentation process. Looking at the "should", we can see that without the slant correction, the vertical projection would have only segmented the image at the space between the "s" and the "h." With the corrected image, each character is better defined and the vertical projection can more accurately distinguish the characters.

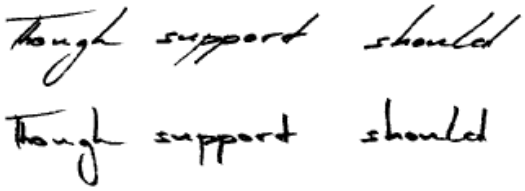


Figure 11 – sample slant correction

One of the problems that arise with input word images is the overlapping of characters. In Figure 12, we see that the word "MOVE" is correctly segmented into four separate letters. However, the algorithm only finds one segmentation point in "Life." This a problem due to the presence of many overlapping letters in the handwritten words. The lack of uniformity in handwritten text makes it difficult to apply a simple vertical projection algorithm.

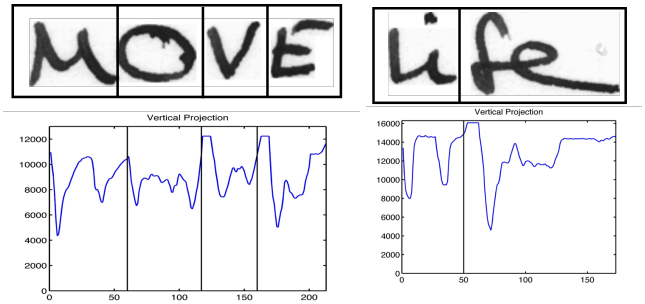


Figure 12 – sample segmentation

As a result of this possible source of error, many of our segmented character images under-segmented the word images. However, to test if the general segmentation algorithm was correct, we filtered our image segmentation outputs to only include words that had the same number of segmentation points as number of characters before we tested the individual images on our trained convnet. Of the 1500 segmented characters that we tested, we had a **55.47%** accuracy. This lower percentage in comparison to the accuracy of the Chars74K data makes sense because the learned weights from our convnet were directly used, without any additional fine-tuning or backwards passes to fit the network to our new data.

5. Conclusions

	MNIST	Chars74K
KNN	0.9328	0.3547
Linear Classifier	0.8869	0.3015
LeNet	**0.9905**	0.4536
AlexNet	~	0.6338
Our Convnet	0.9812	**0.7169**
DeCampos	~	0.5526

Table 1 – Aggregation of Test Accuracies

Experimenting with different possible optimal architectures has left us with the realization that even for very similar datasets, the choice of a proper architecture is extremely important. We saw that when using the LeNet architecture on the Chars74K dataset, the performance was far below the best accuracy we obtained. We also saw that blindly using a deep and complicated network is not the best choice either. While fine-tuning and data augmentation, as well as feature extraction in slant correction did improve our results, the complexity of the network either in being unable to capture all features or capturing unimportant features will result in the largest improvements.

	Accuracy
Segmentation Accuracy	0.2316
Recognition Given Correct Segmentation	0.5547

Table 2 – Segmentation Pipeline Accuracies

One of the most important takeaways from this project was discovering the importance of not separating the segmentation and recognition processes. It is far easier to develop a pipeline that utilizes the recognition system to determine where the characters are located in a word image. One solution would be to pass data to the neural network by utilizing a sliding window. Using a softmax classifier, the neural network return a certain probability that a character exists. By finding a local maximum for the probability in some search small space, we can determine where the characters are.

6. References

[1] 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.

[2] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition.

[3] Choudhary, A. A Review of Various Character Segmentation Techniques for Cursive Handwritten Words Recognition. International Journal of Information & Computation Technology. 2014.

[4] Gupta, J., Chanda, B. Novel Methods for Slope and Slant Correction of Off-line Handwritten Text Word. Third International Conference on Emerging Applications of Information Technology. 2012.

[5] Papandreou, A., Gatos, B., Slant estimation and core detection for handwritten Latin words. Pattern Recognition Letters. 2014.