Handwritten Tamil Recognition using a Convolutional Neural Network

Prashanth Vijayaraghavan
MIT Media Lab
pralav@mit.edu

Misha Sra
MIT Media Lab
sra@mit.edu

Abstract

We classify characters in Tamil, a south Indian language, using convolutional neural networks (ConvNets) into 35 different classes. ConvNets are biologically inspired neural networks. Unlike other vision learning approaches where features are hand designed, ConvNets can automatically learn a unique set of features in a hierarchical manner. We augment the ConvNetJS library for learning features by using stochastic pooling, probabilistic weighted pooling, and local contrast normalization to establish a new state-of-the-art of 94.4% *accuracy on the IWFHR-10 dataset. Furthermore, we describe the different pooling and normalization methods we implemented and show how well they work in our experiments.*

1. Introduction

Tamil is one of the oldest languages in the world with several million speakers in the southern Indian state of Tamil Nadu and Sri Lanka. Tamil is written in a non-Latin script and has 156 characters including 12 vowels and 23 consonants (see Figure 1). Compared to Latin character recognition, isolated Tamil character recognition is a much harder problem because of the larger category set and potential confusion due to similarity between handwritten characters. Previous approaches in classifying Tamil characters have used a variety of hand-crafted features [15] and template-matching [16]. In contrast, ConvNets learn features from pixels all the way to the classifier [14]. The superiority of learned features over hand-designed ones was demonstrated by [12] and also shown by others obtaining best performance in traffic sign classification using ConvNets [14, 3]. The deep learning network structure implicitly extracts relevant features, by restricting the weights of one layer to a local receptive field in the previous layer. By reducing the spatial resolution of the feature map, a certain degree of shift and distortion invariance is achieved [5]. The number of free parameters decreases significantly due to using the same set of weights for all features in a feature map [9]. Their ability to exploit the spatially local correlation

Figure 1: 32×32 cropped samples from the classification task of the IWFHR-10 Tamil character dataset. The dataset has samples for 156 handwritten characters classes.

has helped in achieving extremely high performance on the popular MNIST handwritten digits dataset [10]. ConvNets were also able to achieve a 97.6% facial expression recognition rate on 5,600 still images of more than 10 individuals [11].

Handwritten character recognition can be online or offline. In this context, online recognition involves conversion of digital pen-tip movements into a list of coordinates, used as input for the classification system whereas offline recognition uses images of characters as input. Some of the earlier works apply shallow learning with hand-designed features on both online and offline datasets. Examples of hand-designed features include pixel densities over regions of image, character curvature, dimensions, and number of horizontal and vertical lines. Shanthi et al. [15] use pixel densities over different zones of the image as features for an SVM classifier. Their system achieved a recognition rate of 82.04% on a handwritten Tamil character database. Sureshkumar et. al. [16] use a neural network based algo-

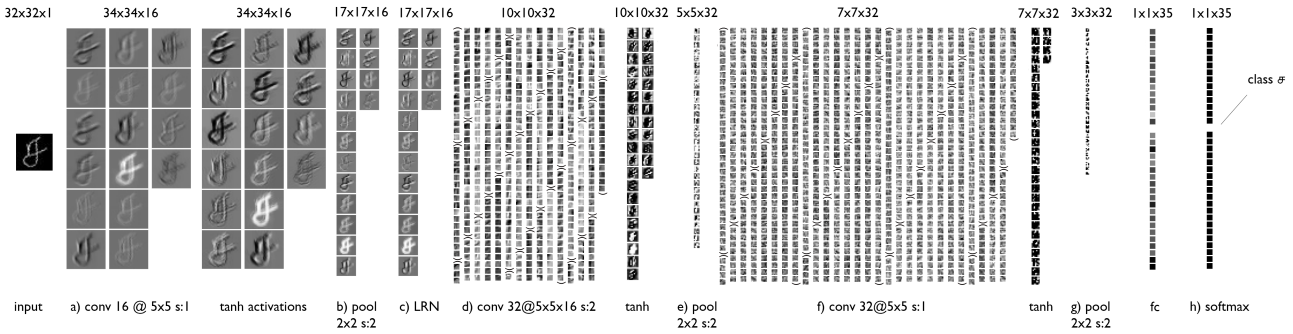

Figure 2: An input image (or a feature map) is passed through a non-linear filterbank, followed by $tanh$ activation, local contrast normalization and spatial pooling/sub-sampling. a) First convolutional layer with 16 filters of size 5×5. b) Max-pooling layer of size 2×2 with stride 2. c) Local response normalization layer with $alpha = 0.1$ and $beta = 0.75$. d) Second convolutional layer with 32 filters of size 5×5. e) Max-pooling of size 2×2 with stride 2. f) Third convolutional layer with 32 filters of size 5×5. g) Max-pooling of size 2×2 with stride 2. h) 35-way softmax classification layer. We train using stochastic gradient descent with an adaptive learning rate.

rithm where features like number of horizontal and vertical arcs and width and height of each character are extracted during pre-processing. These features are then passed to an SVM, a Self Organizing Map, an RCS, a Fuzzy Neural Network, and a Radial Basis Network. They achieve an accuracy of 97% on test data but their approach is not invariant to deformations or different writing styles as their algorithms are highly dependent on the form of the character. Unfortunately, they provide little to no detail on their dataset. Ramakrishnan et al. [13] derive global features from discrete Fourier transform (DFT), discrete cosine transform (DCT), wavelet transform to capture overall information about the data and feed into an SVM with a radial basis function (RBF) kernel. They obtain 95% accuracy on an online test set. Though there has been a lot of research in Tamil handwriting recognition, most of it has been with online datasets [1, 8], or with online and offline hybrid classifiers, and limited research with offline datasets. To the best of our knowledge, we have not seen previous attempts with ConvNets for our particular dataset. We employ the traditional ConvNet architecture augmented with different pooling methods and local contrast normalization. This work is implemented with the open source ConvNetJS library [1].

2. The Dataset

We train the offline IWFHR-10 Tamil character dataset from the HP Labs India website [2]. The dataset contains approximately 500 samples for each of the 156 Tamil characters written by native Tamil writers. The characters are made available for download as TIFF files. We resize the original unequally sized rectangular images into 32×32 square images and save them as JPG files. The resized JPG images are exported and saved as rows in a large CSV file where the first column of each row is added as the image class. This is done in MATLAB. A simple Python script is used to shuffle this large CSV file and split it into two smaller CSV files, one for the training set and another for the test set containing approximately $60K$ and $10K$ images each. We read both CSV files into the ConvNetJS library by implementing a CSV parser using Papaparse [3].

3. Architecture

The input to the convolutional neural network is a 32×32 image passed through a stack of different kinds of layers as follows: $n \times 32 \times 32 - 16C5 \times 5 - P2 \times 2 - L3 \times 3 - 32C5 \times 5 - P2 \times 2 - 32C5 \times 5 - P2 \times 2 - 35N$. This represents a net with n input images of size 32×32, a convolutional layer with 16 maps and filters of size 5×5, a max-pooling layer over non-overlapping regions of size 2×2, a convolutional layer with 32 maps of size 5×5, a max-pooling layer over non-overlapping regions of size 2×2 and a fully connected output layer with 35 neurons, one neuron per class (see Figure 2). We use a non-linear hyperbolic tangent activation function, where the output f is a function of input x such that $f(x) = tanh(x)$ for the convolutional layers, a linear activation function for the max-pooling layers, and a softmax activation function for the output layer. We train

[1] http://cs.stanford.edu/people/karpathy/convnetjs/

[2] http://lipitk.sourceforge.net/datasets/tamilchardata.htm

[3] http://papaparse.com/

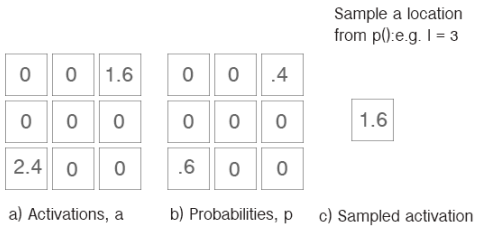

Figure 3: Example illustrating stochastic pooling. a) Resulting activations within a 3×3 pooling region. b) Probabilities based on the activations. c) Sampled activation.

using stochastic gradient descent with an adaptive learning rate[17].

The input layer has $N \times N$ neurons corresponding to the size of the input image. If we use an $m \times m$ filter λ, the output of the convolutional layer will be of size $(N - m + 1) \times (N - m + 1)$. The input at any neuron at a particular point in time x_{ij}^l is the sum of the weighted contributions from the neurons in the previous layer such that:

$$x_{ij}^l = \sum_{p}^{m-1} \sum_{q}^{m-1} \lambda_{pq} y_{(i+p)(j+q)}^{(l-1)} \tag{1}$$

A non-linear function is then applied element-wise to each feature map and the resulting activations are passed to the pooling layer.

$$y_{ij}^l = \sigma(x_{ij}^l) \tag{2}$$

The pooling layer (i.e. subsampling) outputs local averages of the feature maps in the previous convolutional layer to produce pooled feature maps (of smaller size) as output. Max-pooling layers take a $K \times K$ region as input and output a single value based on the maximum value in that region. If the input layer is $N \times N$, the output from the pooling layers is of size $NK \times NK$ since all the $K \times K$ blocks are reduced to a single value. Because pooling is an average over a local region, the output feature maps are less sensitive to precise locations of features in the image than the first layer of feature maps. Beyond the conventional deterministic forms of pooling like average and max, Zeiler et al. [18] introduce stochastic pooling and probabilistic weighting which we implement in the ConvNetJS library for our experiments.

3.0.1 Stochastic Pooling

In stochastic pooling, a sample activation from the multinomial distribution of activations from each pooling region is selected. More precisely, the probabilities p for every region R_j (total regions n_r) are calculated after normalizing the activations within the region (see Figure 3).

$$p_i = \frac{x_i}{\sum_{k \in R_j} x_k} \tag{3}$$

Sampling the multinomial distribution based on p to pick a location t within the pooling region is simply:

$$s_j = x_t; t \sim P(p_1 \cdots p_{n_r}) \tag{4}$$

3.0.2 Probabilistic Weighting

Probabilities are computed similar to stochastic pooling. But the activations in each region are weighted by the probability p_i and summed. It can be called probabilistic weighted averaging as it is a variation of the standard average pooling. Stochastic pooling causes performance degradation during test but probabilistic averaging can boost the performance when applied during test time.

$$s_j = \sum_{k \in R_j} p_i x_i \tag{5}$$

The next pair of convolutional and subsampling layers work in the same manner. The convolutional layer takes in the output from the pooling layer as input and extracts features are that are increasingly invariant to local changes in the input images. The second convolutional layer has 32 feature maps which increases the feature space but reduces the spatial resolution. The last layer is the classification layer.

Figure 4: Output from the LRN layer for one character, rotated for display.

3.0.3 Local response normalization

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron [7]. Even though we got better results using $tanh()$ over ReLU, we still found that the following local normalization schemes aided generalization (see Figure 4).

This layer computes the function:

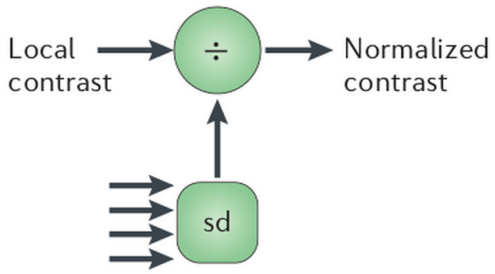

Figure 5: Local contrast normalization

$$f(u_f^{x,y}) = \frac{u_f^{x,y}}{1 + \frac{\alpha}{N} region_{xy}} \tag{6}$$

where $u_f^{x,y}$ is the activity of a unit in map f at position x, y prior to normalization, S is the image size, and N is the size of the region to use for normalization. The output dimensionality of this layer is always equal to the input dimensionality.

$$region_{xy} = \sum_{x'=max(0,x-N/2)}^{min(S,x-N/2+N)} \sum_{y'=max(0,y-N/2)}^{min(S,y-N/2+N)} (u_f^{x',y'})^2)^\beta \tag{7}$$

This layer is useful when using neurons with unbounded activations (e.g. rectified linear neurons) as it permits the detection of high-frequency features with a big neuron response, while damping responses that are uniformly large in a local neighborhood. It is a type of regularizer that encourages "competition" for big activities among nearby groups of neurons [4].

3.0.4 Local contrast normalization

Local contrast Normalization (LCN) can be either subtractive or divisive. Subtractive LCN subtracts from every value in the feature a gaussian weighted average of its neighbors. Divisive LCN divides every value in a layer by the standard deviation of its neighbors over space and over all feature maps. The mean and variance of an image around a local neighborhood are made consistent (see Figure 5)and that is useful for correcting non-uniform illumination or shading artifacts [5].

We implement this layer to compute the LCN function:

$$f(u_f^{x,y}) = \frac{u_f^{x,y}}{1 + \frac{\alpha}{N} region_{xy}} \tag{8}$$

where $m_f^{x',y'}$ here is the mean of all $u_f^{x,y}$ in the 2D neighborhood defined by the summation bounds below.

$$region_{xy} = \sum_{x'=max(0,x-N/2)}^{min(S,x-N/2+N)} \sum_{y'=max(0,y-N/2)}^{min(S,y-N/2+N)} (u_f^{x',y'} - m_f^{x',y'})^2)^\beta \tag{9}$$

This layer is similar to the response normalization layer except we compute the variance of activities in each neighborhood, rather than just the sum of squares (correlation).

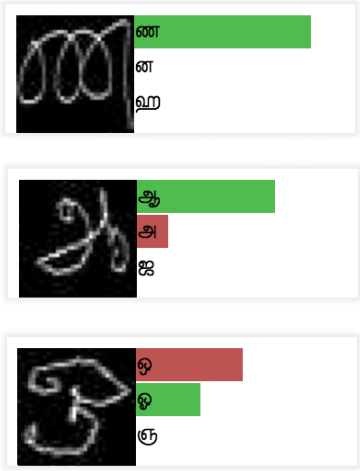

Figure 6: The first two images are perfectly classified with high confidence scores despite existence of similar characters shown as second best predictions for each image.

4. Experiments

ConvNets have a large set of hyper-parameters and finding the perfect configuration for each dataset is a challenge. We explored different configurations of the network and attempted to optimize the parameters based on the validation set accuracy. For our recognition task, we focus on 35 Tamil characters that include all vowels and consonants with a dataset of 18,535 images.

4.1. Data Preparation

The IWFHR-10 classification dataset contains randomly sized images. After resizing all images to 32 samples, we split the dataset is into three subsets: train set, validation

[4] https://code.google.com/p/cuda-convnet/wiki/LayerParams

[5] http://bigwww.epfl.ch/sage/soft/localnormalization/

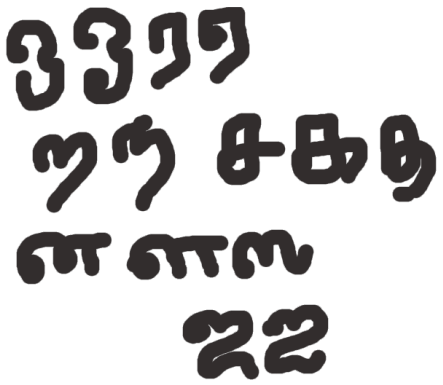

Figure 7: Pairs of similar looking characters in the Tamil dataset.

set and test set. Since we are given no information about how the sampling of these images was done, we shuffle our datasets before use. We initially experimented with rectangular images of size 35×28 selected to maintain the aspect ratio of most of the sample images in the original dataset. However, the results were not promising due to limited training examples. A neural network model for pattern recognition should make predictions that are invariant to variations of the same class of patterns [4]. This is usually done by training the neural network using a dataset with enough variations of the patterns. However, with limited data, the neural network training can over-fit and hurt the classification robustness. One way to deal with this problem is data augmentation where the training set is artificially augmented by adding samples with transformations that preserve the class labels. Data augmentation is widely used in image recognition tasks where transformations have led to significant improvements in recognition accuracy [4]. To increase the size of our dataset, we first created complementary images and normalized them to values between 0 and 1 and then applied rotational deformations for improving spatial invariance resulting in a dataset $70,524$ images.

4.2. Character Recognition

Our initial experiments gave us a 93% training accuracy while the test accuracy was 89.3%. We explored techniques to prevent overfitting (see Tables 1 and 2) by applying different regularization methods. In a convolutional neural network there may be many different settings of weights that can model the training set well, especially with limited labeled training data. Each of these weight vectors will make different predictions on test data and most likely not do as well as it did on training data because the feature detectors have been tuned to work well together on the training set but not on the test set [6]. Dropout is a regularization technique where on each presentation of each training case, feature detectors are deleted with probability p and the remaining weights are trained by backpropagation [2].

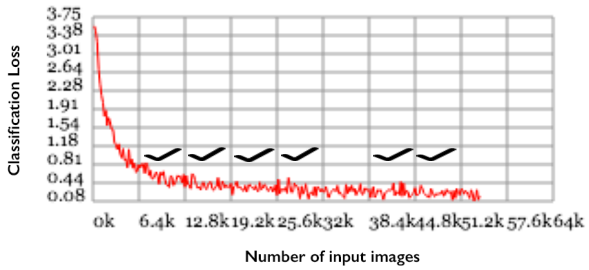

Figure 8: The classification function calculates a negative log likelihood loss and back propagates L1-loss.

For improving generalization, we implement a data augmentation function for our network, where we initially input a 35×28 image and crop a random 31×24 window from that image before training on it. Similarly, to do prediction, 4 random crops are sampled and the probabilities across all crops are averaged to produce final predictions [6]. For square input images of size 28×28, the ConvNetJS library already has a data augmentation implementation as described above.

We implemented and experimented with the following generalization techniques:

- Stochastic Pooling during training and testing
- Probabilistic Weighting during training and testing
- Stochastic Pooling during training + Probabilistic Weighting during testing
- Dropout in a convolutional layer
- Dropout in a fully connected layer
- Data augmentation function

Applying these techniques did not provide a big boost in performance but we did see marginal increases in comparison with configurations without them. With the implementation of local contrast normalization, the test accuracy improved to 94.1%. However, the local response normalization outperformed all other configurations and we were able to achieve a high test accuracy of 94.4% with a training accuracy of 99%. The effect of local contrast normalization

[6] http://cs.stanford.edu/people/karpathy/convnetjs/

Pooling	Activation	Classifier	Train Acc.	Test Acc.	Others
Max	Tanh	Softmax	91%	87.39%	NA
Max	ReLu	Softmax	69%	48.2%	NA
Max	Tanh	SVM	84%	80.4%	NA
Stochastic	Tanh	Softmax	86%	87.72%	NA
Stochastic	Tanh	Softmax	84%	57.92%	FC (dropout:0.1)
Stochastic	Tanh	SVM	80%	65.99%	NA
Stochastic+Prob Wt	Tanh	Softmax	84%	86.5%	NA

Table 1: Experiments with 35×28 images. FC: fully connected layer. Dropout: drop activations with probability 0.1.

Pooling	Activation	Classifier	Train Acc.	Test Acc.	Others
Max	Tanh	Softmax	99%	94.4%	LRN
Max	Tanh	Softmax	95%	94.1%	LCN
Max	Tanh	Softmax	97%	93.5%	NA
Max	Tanh	Softmax	93%	89.3%	FC (neurons:1024)
Max	ReLu	Softmax	97%	93.2%	NA
Max	Tanh	SVM	97%	90.3%	NA
Max	Tanh	Softmax	82%	73%	FC (dropout:0.2)
Stochastic	Tanh	Softmax	94%	93.3%	NA
Stochastic	ReLu	Softmax	95%	92.5%	NA
Stochastic	Tanh	SVM	90%	89.6%	NA
Stochastic+Prob Wt	ReLu	Softmax	94%	92.6%	NA

Table 2: Experiments with 32×32 images. FC: fully connected layer. Dropout: drop activations with probability 0.2.

can be seen in Figure 2 with the approximate whitening of the image. Experiments using Dropout in a convolutional layer as well as in a fully connected layer led to a surprising drop in accuracy and we conjectured this was likely due to a small total number of activations possibly leading to loss of important activations.

The convolutional neural network architecture was trained with different set of configurations for rectangular and square images. The hyperparameters were optimized based on a validation set of images and results are shown in Tables 1 and 2.

Figure 8 shows the drop in the classification loss as the number of training examples increases. Classification loss came down from approximately 3.39 to 0.08 as the training progressed. Exploiting the ability of convolutional neural networks to learn invariances to scale, rotation, and translation the characters were recognized with high accuracy. Figure 6 shows the images and the top three predictions of our classification. The first two images are perfectly classified with high confidence scores though they have very similar looking characters as the second best predictions for each image. In the third image, our ConvNet fails to correctly classify the character but the confidence score is really close to the second best guess. We would like to note that it is difficult even for a human being to distinguish between these two characters as they are very similar to each other.

5. Acknowledgements

We would like to thank Professors William Freeman and Antonio Torralba for giving us the opportunity to learn about ConvNets through course 6.869. We are also thankful to the course TAs Carl Vondrick and Andrew Owens for their help and support. Lastly, we would like to thank HP Labs India for making the dataset available for research.

6. Discussion

Our results show that a convolutional neural network is capable of achieving record breaking results on the Tamil dataset using purely supervised learning. It is notable that our network's performance degrades if a single convolutional layer is removed and does not improve much with the addition of another convolutional + pooling layer pair. To simplify our experiments, we did not use any unsupervised pre-training even though we expect that it will help. We explored auto encoders but since our system is running in the browser, we had trouble saving large amounts of data to file from the browser to be used as input for the ConvNet.

References

[1] K. Aparna, V. Subramanian, M. Kasirajan, G. V. Prakash, V. Chakravarthy, and S. Madhvanath. Online handwriting recognition for tamil. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 438–443. IEEE, 2004.

[2] P. Baldi and P. J. Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013.

[3] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE, 2011.

[4] X. Cui, V. Goel, and B. Kingsbury. Data augmentation for deep neural network acoustic modeling. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 5582–5586. IEEE, 2014.

[5] S. Haykin. Self-organizing maps. *Neural networks-A comprehensive foundation, 2nd edition, Prentice-Hall*, 1999.

[6] 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.

[7] 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.

[8] R. Kunwar and A. Ramakrishnan. Online handwriting recognition of tamil script using fractal geometry. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1389–1393. IEEE, 2011.

[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, 1998.

[10] Y. LeCun and C. Cortes. The mnist database of handwritten digits.

[11] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5):555–559, 2003.

[12] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4, 2011.

[13] A. G. Ramakrishnan and K. B. Urala. Global and local features for recognition of online handwritten numerals and tamil characters. In *Proceedings of the 4th International Workshop on Multilingual OCR*, MOCR '13, pages 16:1–16:5, New York, NY, USA, 2013. ACM.

[14] P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2809–2813. IEEE, 2011.

[15] N. Shanthi and K. Duraiswamy. A novel svm-based handwritten tamil character recognition system. *Pattern Analysis and Applications*, 13(2):173–180, 2010.

[16] C. Sureshkumar and T. Ravichandran. Handwritten tamil character recognition and conversion using neural network. *Int J Comput Sci Eng*, 2(7):2261–67, 2010.

[17] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[18] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.