ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 7 N. 3 (2018), 47-61
eISSN: 2255-2863
DOI: http://dx.doi.org/10.14201/ADCAIJ2018734761

Design of CNN architecture for Hindi Characters

Madhuri Yadav[a], Ravindra Kr Purwar[a], and Anchal Jain[b]

[a]USIC&T, GGSIPU, Delhi, India
[b]Deptt. of CSE, Indraprastha Engineering College, Ghaziabad, UP, India
madhuri26yadav@gmail.com, ravindra@ipu.ac.in, anchalresearch10@gmail.com

KEYWORD

hindi character recognition; deep learning; CNN; handwritten characters

ABSTRACT

Handwritten character recognition is a challenging problem which received attention because of its potential benefits in real-life applications. It automates manual paper work, thus saving both time and money, but due to low recognition accuracy it is not yet practically possible. This work achieves higher recognition rates for handwritten isolated characters using Deep learning based Convolutional neural network (CNN). The architecture of these networks is complex and plays important role in success of character recognizer, thus this work experiments on different CNN architectures, investigates different optimization algorithms and trainable parameters. The experiments are conducted on two different types of grayscale datasets to make this work more generic and robust. One of the CNN architecture in combination with adadelta optimization achieved a recognition rate of 97.95%. The experimental results demonstrate that CNN based end-to-end learning achieves recognition rates much better than the traditional techniques.

1. Introduction

Automatic character recognition is a process that converts scanned document images into computer understandable format. It can be highly beneficial for various private and public sectors due to its numerous applications such as mail sorting, address recognition, cheque recognition, scene text detection, video text detection, restoring historical documents and so on. Millions of handwritten documents can be processed in seconds using character recognizer. The combination of speech synthesis and character recognition aids visually challenged person to understand documents more easily and effectively. All these applications make handwritten character recognition a vital research area.

In last few years, most of the pattern recognition problems used traditional techniques for problem solving. These techniques follow a basic pipeline of data acquisition, preprocessing, feature extraction, classification and post-processing (optional). One of the most recent work (Madhuri Yadav, 2018), used Hu-geometric moments and histogram of oriented gradients for hindi handwritten characters. The authors exploit the geometric invariant property of moments and used image gradients for spatial correlation. The performance of these features was evaluated on SVM and MLP. In (Deepti Khanduja, 2015), authors exploit the structural properties

Table 1: Summary of literature works of Hindi handwritten character recognition

Methods	Features	Classifiers	Recognition rate (in %)
Madhuri *et al.* (Madhuri Yadav, 2018)	HOG and Hu-moments	SVM and MLP	96.8
Deepti *et al.* (Deepti Khanduja, 2015)	Structural and Quadratic coefficients	MLP	93.4
Ritesh *et al.* (Sarkhel *et al.*, 2017)	CNN	CNN	95.18
Hanmandlu *et al.* (Hanmandlu *et al.*, 2007)	Structural features	Reinforcement learning	90.65
G.K. verma *et al.* (Gyanendra K.Verma, 2011)	Curvelet	k-NN	90
S. Behle *et al.* (Belhe *et al.*, 2012)	Symbol trees	HMM	89
Madhuri *et al.* (Yadav and Purwar, 2017)	Projection profiles	Multiple classifiers	96.6
H.B. Kekre *et al.* (Kekre *et al.*, 2013)	Shape and tecture features	LBG Algorithm	—-
Shitala Prassad *et al.* (Prasad *et al.*, 2012)	Multi-resolution and multi directional	k-NN and SVM	95.4

of a character and use end points, intersection points, branch points, and quadratic polynomial coefficients as features. Hanmandlu *et al.* (Hanmandlu *et al.*, 2007)used reinforcement learning on fuzzy sets. The work in (Gyanendra K.Verma, 2011) explored the curvity of characters and used curvelet transform for feature extraction and k-nearest neighbor (k-nn) as classifier. S.Behle (Belhe *et al.*, 2012) proposed online hindi word recognition system by segmenting words into vowels, matras, syllables etc using Hmm models and giving recognized word probabilities by symbol trees. The work in (Yadav and Purwar, 2017) used projection profile histogram features and compared performance of different classifiers such as MLP, SVM, Bagged trees for Hindi isolated characters. In (Kekre *et al.*, 2013), shape and texture features are extracted from isolated hindi characters using gradient masks and LBG vector quantization, respectively. Shitala (Prasad *et al.*, 2012) proposed multi-lingual character recognition using wavelet, curvelet and ridgelet multi-resolution transforms. SVM and k-nn classifiers were used for classification purposes. These traditional techniques have achieved remarkable accuracies and state of art results, but as of now, they have reached a stagnant point which requires new methodologies to improve accuracy. This stagnancy in accuracy was observed in other pattern recognition problems as well, thus there was a shift from traditional learning to end-to-end learning. Table 1 tabulates the recognition accuracies reported by these works.

Deep learning techniques follow end-to-end learning. The basic workflow in character recognition is preprocessing, automatic feature extraction and classification. The features are not explicitly specified in this kind of learning. Among various deep learning models, Convolutional Neural Networks (CNN) has provided solutions to almost all domains of pattern recognition such as bio-medical imaging, agriculture, speech recognition, object detection, face recognition , scene classification and so on. Ritesh *et al.* (Sarkhel *et al.*, 2017)proposed multi column multi scale deep convolutional network for isolated handwritten characters. Thus, the purpose of this work is to use convolutional neural network for hindi handwritten character recognition. To make CNN learn significant features they should be trained carefully. There are number of decisions that a CNN designer has to make prior to convolutional learning. Some of the decisions are as follows:

- Number of convolutional layer in a network,
- Number of filters and of what size,
- Number of pooling layers with different kernel and stride size,
- Number of hidden neurons in dense layers,

- Which optimization algorithms to be used and with what parameter values, and
- Depth of convolutional network and so on.

This paper tries to answer these questions for Hindi handwritten character recognition. It also investigates the four optimization methods namely Stochastic Gradient Descent (SGD) (Bottou, 2012), Adadelta (Zeiler, 2012), Rmsprop (Tieleman, 2012) and Adam (Kingma and Jimmy, 2014) along with different CNN architectures and tries to identify the one which gives highest recognition accuracy. Section 2 discusses the basic terminologies used in convolutional networks and also discusses the properties of CNN architectures proposed in this work. Section 3 details the databases used and experimental results of different architectures using different optimization methods. Finally, section 4 concludes this article.

2. CNN Architectures

2.1. Basic terminologies used in convolutional networks

The CNN is an ensemble of three basic layers which are: convolution layer, pooling layer or sub-sampling layer and classification layer or dense layer as shown in Figure 1.

The input of this architecture is the image which is to be classified. In traditional methods, the inputs to the network are the extracted features, whereas in CNN, the input is the raw image which may or may not be pre-processed. Thus, CNN are called automatic feature extractor. In mathematical terms, the input to the CNN is matrix Y of dimensions r * r * m where r is the height and width of the image and m is the number of channels present in the image i.e. RGB, grayscale or 0/1 image.

The main features which make CNN robust are shift, scale and distortion invariance for which it uses receptive fields, shared weights and sub-sampling. Each convolution layer has k kernels (or filters) of size n*n*q where $n \text{ Æ } m$ and $q \text{ Æ } r$. Each kernel is convolved over the entire image to form k activation maps for next layers. Each filter has different set of weights and bias so that they can extract different local features. The overlapping portion of weighted kernel k with the input image is called the receptive field. The convolutional layer is actually responsible for feature extraction. With local receptive fields in this layer, the neurons can extract elementary information such as corners, edges, end points etc (Lecun *et al.*, 1998).These features are then fed as input to subsequent layers so that high-level features can be extracted. A convolutional layer contains multiple filters so that multiple features are extracted from an input image at each level. The input image is sequentially scanned over the local receptive field and the convolved output of weights of filter and input image intensities are stored as input for the next layer. In this way the kernel is convolved over entire image and different activation maps are obtained as shown in Figure 2.

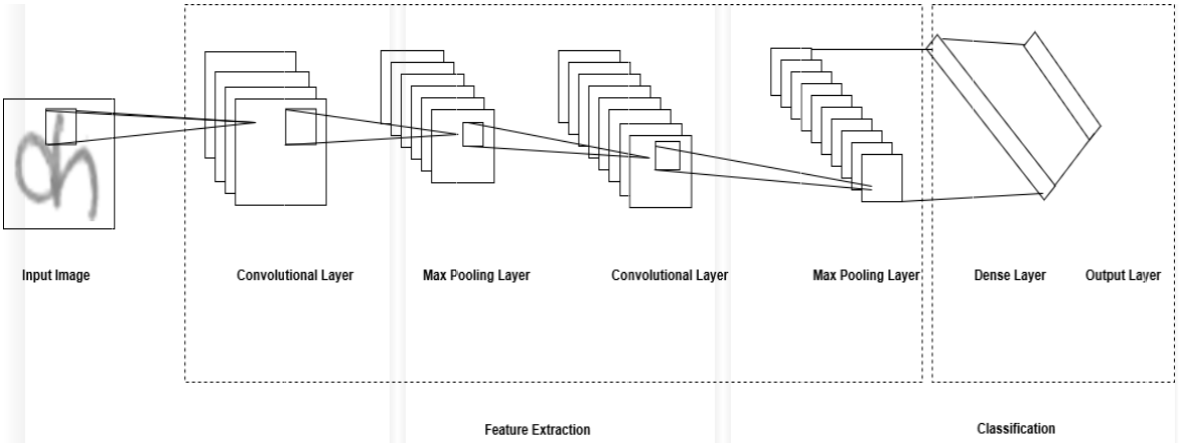


Figure 1: Architecture of basic convolutional neural network

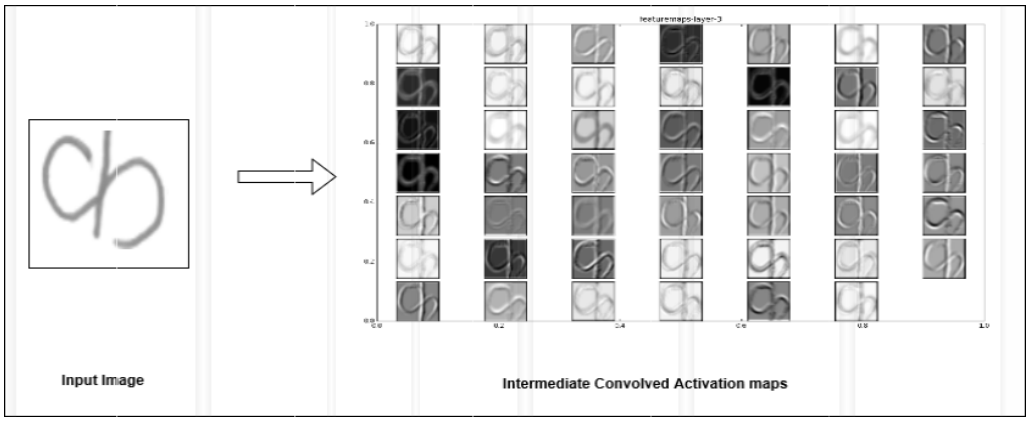


Figure 2: Intermediate activation maps obtained using 48 filters and Adam optimization

The size of the next layer activation maps depends upon two factors i.e. padding and stride. The height and width of the subsequent feature maps are given by Equation 1.

$$\frac{(W \neq F + 2\acute{u}P)}{S} + 1 \tag{1}$$

where W is the size of input image or previous layer activation maps, F is the size of filter, P means padding which is adding of extra zeros to protect the edges of image, S stands for stride, it decides the movement of kernel across the image. The sub-sampling or pooling layer performs local averaging or max pooling and make these networks shift invariance to some extent. This layer also reduces the size of the activation maps. It is based upon the concept of relative positioning; once the feature is detected its exact location becomes irrelevant only the approximate location with respect to other features is important. The features extracted according to exact location are easily vulnerable to slight shift changes in a character. Thus, this layer was introduced to handle shift distortions and reduce resolutions of input images. The fully connected layer or dense layer receives the input from the learned features and flattens them and fed input to classification layer. This layer assigns labels to each class and identifies input images as the corresponding characters. In addition to above discussed layers there are two more layers dropout layer and activation layer. Dropout layer is a regularization technique introduced by (Srivastava *et al.*, 2014), it randomly selects neurons and disable them during training. The randomly selected nodes are dropped-out with a given probability (say 0.5) for each weight update cycle. The nodes are dropped off only at the time of training to avoid regularization and not at the time of evaluation of network. This layer forms an important part of the network as it helps in generalizing the convolutional network so that it can produce accurate results even in case of completely unknown image.

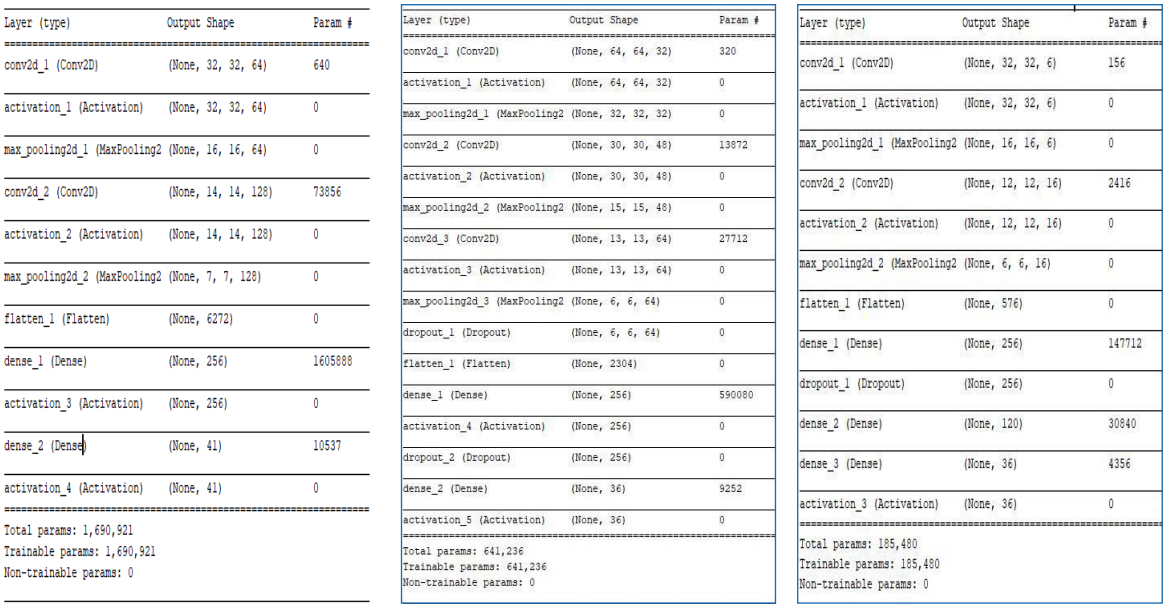
2.2. Architectures Designed in the proposed work

Three architectures have been proposed in this work. Every architecture exhibit different property and trainable parameters. All architectures are discussed in detail in further sections.

2.2.1. *Architecture 1*

The input to this architecture is 32x32 grayscale image as shown in Figure 3(a). The first convolutional layer C11 (labeled as Cxy where x corresponds to architecture number and y represents layer number) with 64 filters of size 3x3 extracts basic features such as end points, corners, edges, intersection points etc. The initial value or weights of these filters can be randomly chosen or by using weight initialization techniques like Xavier filler, proposed by Glorot (Glorot and Bengio, 2010). The weight of each of these filters is different so that they can extract different types of elementary features. The kernel is applied on each 3x3 neighborhood of each activation map. This layer has 640 trainable parameters (3*3*1*64 +64 bias). The next layer is relu activation unit,

it has no parameters. The next layer is max pooling layer with 64 filters. It reduces the size of activation maps by half, using equation (1): Size of activation maps= (32-3+2*1)/2 +1=16. Thus activation maps of 16x16 are obtained in this sub sampling layer. The next convolutional layer C12 has 128 filters, thus 73856 trainable parameters (3*3*64*128+128). The input size of activation maps in this layer is 14x14 which is obtained as



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 64)	640
activation_1 (Activation)	(None, 32, 32, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 14, 14, 128)	73856
activation_2 (Activation)	(None, 14, 14, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 256)	1605888
activation_3 (Activation)	(None, 256)	0
dense_2 (Dense)	(None, 41)	10537
activation_4 (Activation)	(None, 41)	0

Total params: 1,690,921
Trainable params: 1,690,921
Non-trainable params: 0

(a)

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 64, 64, 32)	320
activation_1 (Activation)	(None, 64, 64, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 30, 30, 48)	13872
activation_2 (Activation)	(None, 30, 30, 48)	0
max_pooling2d_2 (MaxPooling2	(None, 15, 15, 48)	0
conv2d_3 (Conv2D)	(None, 13, 13, 64)	27712
activation_3 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 256)	590080
activation_4 (Activation)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 36)	9252
activation_5 (Activation)	(None, 36)	0

Total params: 641,236
Trainable params: 641,236
Non-trainable params: 0

(b)

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 6)	156
activation_1 (Activation)	(None, 32, 32, 6)	0
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 6)	0
conv2d_2 (Conv2D)	(None, 12, 12, 16)	2416
activation_2 (Activation)	(None, 12, 12, 16)	0
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 16)	0
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 256)	147712
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 120)	30840
dense_3 (Dense)	(None, 36)	4356
activation_3 (Activation)	(None, 36)	0

Total params: 185,480
Trainable params: 185,480
Non-trainable params: 0

(c)

Figure 3: Three CNN architectures proposed in this work: (a) Architecture I (b) Architecture II and (c) Architecture III

follows: (16-3+2*0)/1+1=14. The next layer is max pooling layer with input dimensions 7x7 (14-3+2*1)/1+1. It does not have any trainable parameters. The next layer is flatten layer which converts the previous activation map values into format suitable for dense layer. It has 6272 values (7*7*128).

The next layer is dense layer with 256 neurons. Since, it is a fully connected layer with no dropouts it has 6272*256 values. The next dense layer has neurons equivalent to number of classes. This dataset has 41 classes, hence it has 10537 (41*256+41) values. The total trainable parameters in this architecture are1,690,921. The main property of this architecture is that it has lesser numbers of convolutional layers as compared to other architectures, but number of filters is high in each layer. Since, filters are actually responsible for feature extraction, so instead of creating deeper networks the effect of increasing number of filters was exploited in this architecture. There are no dropout layers in this architecture.

2.2.2. *Architecture II*

This architecture has three convolutional layers, three max pooling layers, four activation layers, two dense and dropout layers as represented by Figure 3(b). The number of layers in this architecture is more, as compared to Architecture I, but less number of filters. This architecture was designed to demonstrate the effect of deeper layers with dropout layer. The first convolutional layer C21 has 32 filters of size 3x3 and 320 (3*3*32) trainable parameters. Next layer is relu activation layer for normalizing the values of convolved maps. Neither it has trainable parameters nor dimensionality reduction. The pooling layer helps to achieve shift invariance by max pooling, it reduces the dimensions of activation map to half using (64-3+2*1)/2+1. The next convolutional layer C22 has 48 filters of size 3x3. The input to this layer is activation maps of dimension 30x30 i.e. (32-3+2*0)/1+1. This layer has 13872 parameters (3*3*32*48+48). The next layers are activation and max pooling. Third convolutional layer C23 has 64 filters with 27712 (3*3*48*64+64) parameters. The dropout layer helps in

generalization of network. In this architecture the probability of dropout is set to 0.5 i.e. half of the neurons will be dropped off while training to avoid generalization. The dense layer is a fully connected layer responsible for classification in convolutional network. The first dense layer has 256 neurons with 2304(6*6*64) input values. It has 590080 (256*2304+256) parameters. The next dense layer has 36 neurons according to the character classes in database. The number of trainable parameters in this architecture are 256*36+36 i.e. 9252. The total trainable parameters in this architecture are 641236. It can be noticed that number of filters are increasing as the network is getting deeper; it is because the deeper filters find more advanced features.

2.2.3. *Architecture III*

The architecture represented in Figure 3(c) is architecture III and it is similar to LeCunn architecture (Lecun *et al.*, 1998), except the number of classes in dense layer. This architecture is not much deep and even the number of filters are very less, resulting in less number of trainable parameters and thus low computational cost. The first convolutional layer has 6 filters of size 5x5 hence, the parameters are 156(5*5*6+6). The second convolutional layer has 16 filters with 2416(5*5*6*16+16) trainable parameters. In this architecture the size of the filters has been increased but the number of filters is decreased. The architecture has three dense layers which have 256, 120, and 36 neurons, respectively. The total number of trainable parameters in this architecture is 1,85,480.

2.3. Optimization Algorithms

In deep learning, training is done on large datasets, and hence it consumes lot of time. To reduce training time and to improve the process of learning in deep networks, optimization algorithms are used. The most popular algorithm of artificial neural network i.e. back-propagation with gradient descent is used for convolutional neural network. The focus of this sub section is to introduce its readers with optimization algorithms used in this paper. The details of back-propagation algorithm can be found in (Haykin, 1998). The training can be done using batch or mini-batch gradient descent algorithms. Let us consider a dataset of m samples with training samples as given by Equation 2 with their respective classes as represented by Equation 3.

$$X = [x^{[1]}, x^{[2]}, x^{[3]}, \cdot\cdot \cdots \cdot x^{[m]}] \tag{2}$$

$$Y = [y^{[1]}, y^{[2]}, y^{[3]}, \cdot\cdot \cdots \cdot y^{[m]}] \tag{3}$$

The batch gradient takes the entire set of training samples and tries to optimize the cost function or reach minima. On the other hand, the mini-batch gradient descent algorithms take the mini batches of training samples and try to reach minima. The mini-batches can be represented by Equation 4.

$$X^i = [x^{[1]}, x^{[2]}, x^{[3]}, \cdot\cdot \cdots \cdot x^{[n]}] \tag{4}$$

where $n < m$. Let us consider a training set of 5 million samples, suppose each mini batch has 1000 samples then there can be 5000 mini batches to train. If *mini_batchsize* = *m*, it becomes batch gradient descent, if *mini_batchsize* = 1, it is called stochastic gradient descent (SGD). Algorithm 1 describes the SGD algorithm where, represents the predicted output and actual outputs respectively.

Algorithm 1: Stochastic Gradient Descent at training iteration i

Learning rate —=small constant value , initial parameters (w) and bias (b) are initialized to any random values.;
While stopping criteria do not met;
Each sample from the training set $x^{[1]} \cdots x^{[m]}$ with corresponding targets $y^{[i]}$ forms the minibatch in SGD;
Compute gradient descent for the cost function $J : J\,\Omega\,\frac{1}{\overline{m}}\,\P_{i} f(\hat{y}^{[i]}, y^{[i]})$;
Apply update: $w^{[i]} = w^{[i]} \neq - \text{"}w;\ b^{[i]} = b^{[i]} \neq - \text{"}b$;
end

The next algorithm is Rmsprop which stands for root mean square propagation and works as shown in Algorithm 2.

Adam is combination of Rmsprop and momentum. It is given by Algorithm 3.

3. Experimental Results

3.1. Results on Database I

This dataset consists of 4428 grayscale hindi characters with 108 characters per sample. It has total of 41 classes. The classes of this dataset are equally distributed and thus easy to use. The procedure of creation of this database is explained in (Madhuri Yadav, 2018) [1]. The database is tested on all three CNN architectures as discussed in Section 2. Figure 4(a), 4(b), 4(c) and 4(d) show the performance of this dataset for architecture I on four different optimization algorithms: Adam, SGD, Adadelta, and Rmsprop.

The experiments of the proposed work are performed on Intel dual core i5 processors seventh gen, 8 GB RAM, and a NVIDIA GeForce 750 Ti graphics card with 1TB internal memory, having 640 CUDA cores.

Algorithm 2: RMSProp algorithm

Learning rate⁄, initial parameter weights(w) and bias(b), small constant ", gradient accumulation variable for weight $S_{\text{"}w}$ and bias $S_{\text{"}b}$, a small constant ‘ initialized at $10^{\neq 8}$, constant — initialized at 0.9, Initially $S_{\text{"}b}$, $S_{\text{"}w}$ are 0;
While stopping criteria do not met Take a minibatch of m examples from the training set $x^{[1]} \cdots x^{[m]}$ with corresponding targets $y^{[i]}$;
Compute gradient descent for the cost function $J : J\,\Omega\,\frac{1}{\overline{m}}\,\P_{i} f(\hat{y}^{[i]}, y^{[i]})$;
Accumulate squared gradient for weights: $S_{\text{"}w} = —S_{\text{"}w} + (1 \neq —)\,\text{"}w$;
Accumulate squared gradient for bias : $S_{\text{"}b} = —S_{\text{"}b} + (1 \neq —)\,\text{"}b$;
Compute update: $w^{[i]} = w^{[i]} \neq - \hat{O}\,\frac{\text{"}w}{(S_{\text{"}w} + \text{‘})}$; $b^{[i]} = b^{[i]} \neq - \hat{O}\,\frac{\text{"}b}{(S_{\text{"}w} + \text{‘})}$;
end

Algorithm 3: Adam algorithm

Learning rate, initial parameter weights (w) and bias(b), small constant ", gradient accumulation variable for weight $S_{"w}$ and bias $S_{"b}$, a small constant ' initialized at $10^{\neq 8}$, constant —1 initialized at 1.9 and—2 at 0.999, Initially velocity for bias and weights $V_{"b}$ and $V_{"w}$ Vdw are 0. $S_{"b}$,$S_{"b}$ are 0;

While stopping criteria do not met;

Take a minibatch of m examples from the training set $x^{[1]} \cdots x^{[m]}$ with corresponding targets $y^{[i]}$.

Compute gradient descent for the cost function $J : J \Omega \frac{1}{m} \P_i f(\hat{y}^{[i]}, y^{[i]})$;

Accumulate squared gradient for weights: $S_{"w} = —2 S_{"w} + (1 \neq —2)"w$;

Accumulate squared gradient for bias : $S_{"b} = —2 S_{"b} + (1 \neq —2)"b$;

Compute momentum for weights: $V_{"w} = —1 S_{"w} + (1 \neq —1)"w$;

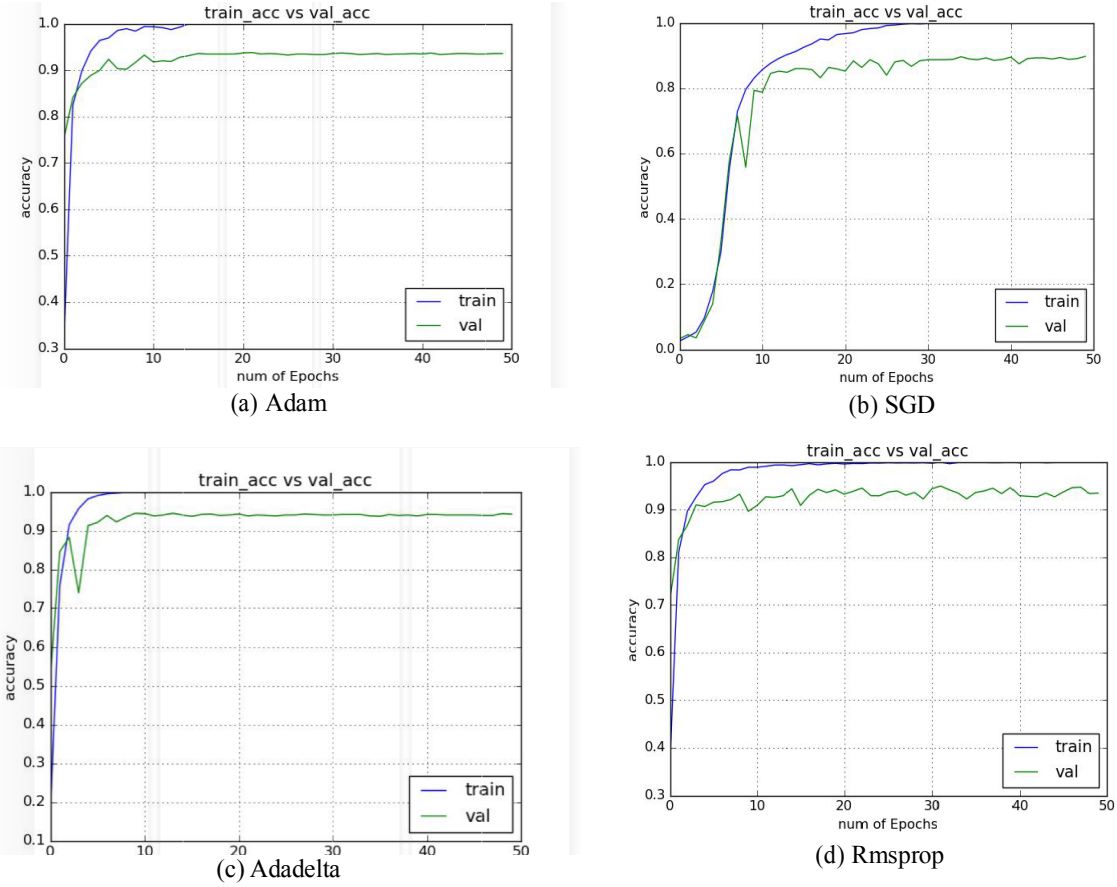
Compute momentum for bias: $V_{"b} = —1 S_{"b} + (1 \neq —1)"b$;

$V_{"w}corrected = \frac{V_{"w}}{1 \neq —1}$; $V_{"b}corrected = \frac{V_{"b}}{1 \neq —1}$;

$S_{"w}corrected = \frac{S_{"w}}{1 \neq —2}$; $S_{"b}corrected = \frac{S_{"b}}{1 \neq —2}$;

Compute update: $w^{[i]} = w^{[i]} \neq -\hat{O} \frac{V_{"w}corrected}{(S_{"w}corrected + ')}$; $b^{[i]} = b^{[i]} \neq -\hat{O} \frac{V_{"b}corrected}{(S_{"w}corrected + ')}$.

end



(a) Adam (b) SGD

(c) Adadelta (d) Rmsprop

Figure 4: Accuracy graphs for CNN architecture I on database I

Table 2: Recognition accuracies (in %) for different optimization methods

Otimization Algorithms	**Dataset I** Architecture I	Architecture II	Architecture III	**Dataset II** Architecture I	Architecture II	Architecture III
SGD	89.72	92.88	89.95	91.34	97.07	96.21
Adadelta	94.24	95.93	93.11	97.89	97.95	96.71
Adam	92.88	95.59	84.65	97.28	97.66	95.73
Rmsprop	93.45	95.37	91.87	90.76	97.53	93.98

Architecture I was designed without dropout layer so it is clearly visible from Fig 4 that there is a need of regularization. The best accuracy is achieved by Adadelta giving an accuracy of 94.24 % with the loss of 0.420. Architecture II gives an accuracy of 95.93 % with Adadelta optimization at loss of 0.229. The accuracy graphs for this architecture are shown in Figure 5.

The architecture II is deeper network than architecture I and the former gives better results. The reason for increased accuracy is that the deeper filters extract more intrinsic features and dropout layer regularizes the network, thus producing better results on testing dataset. Architecture III gives least accuracy of 93.11% with

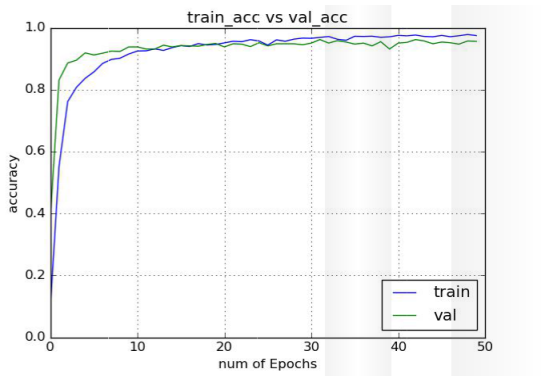
0.40 loss using Adadelta optimization. This architecture had least number of filters, thus the features extracted were not enough to correctly classify the hindi handwritten characters. The accuracy graphs are represented by Figure 6.

3.2. Results on Database II

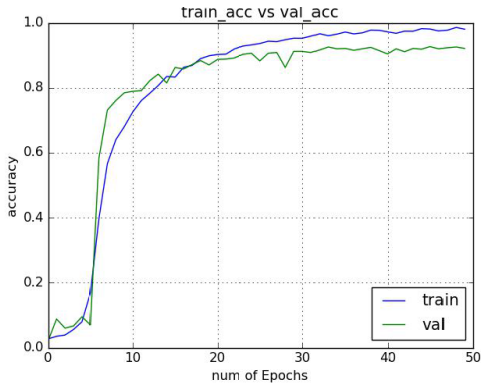
This database is large as compared to Database I. It has total of 6,12,00 images with 1700 images per character and 36 classes. It is also a grayscale database. It was generated by (Acharya *et al.*, 2015). Figures 7,8, and 9 shows experimental accuracy obtained for three CNN architectures on different optimization algorithms. Architecture I was designed without dropout layer so it is clearly visible from Fig. 7 that there is a need of regularization. The best accuracy in this architecture is achieved by Adadelta giving an accuracy of 97.89 % with the loss of 0.420. Architecture II gives an accuracy of 97.95 % with Adadelta optimization at loss of 0.069. The accuracy graphs for this architecture are shown in Figure 8. Architecture III gives least accuracy of 96.71% with 0.13 loss using Adadelta optimization. The accuracy graphs are represented by Figure 9. Table 2 compares the recognition rates of different architectures on different databases.

4. Conclusion

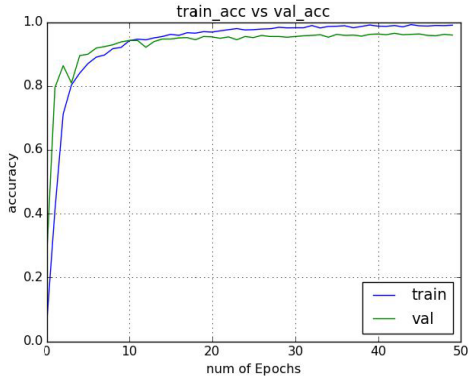
Hindi handwritten character recognition has achieved impressive results using CNN. This work proposed different architectures and experimented on many permutations and combinations of convolutional layers. Two databases have been used for experimental analysis which achieves the highest accuracy of 97.95 %. The experiments prove the significant improvement in accuracy, however, CNN incur high computational cost and high storage space, so there is a need to devise an algorithm which optimize and achieves cost efficient character recognition system



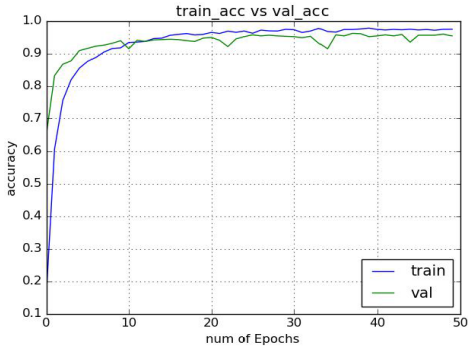
(a) Adam



(b) SGD

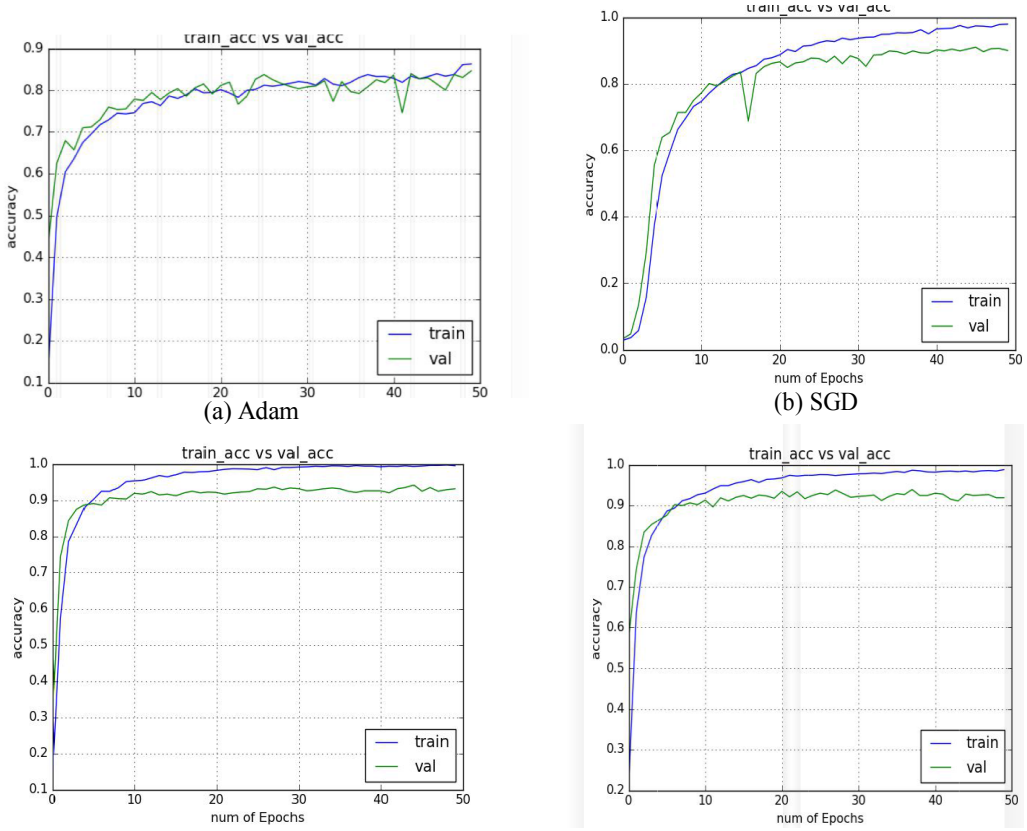


(c) Adadelta



(d) Rmsprop

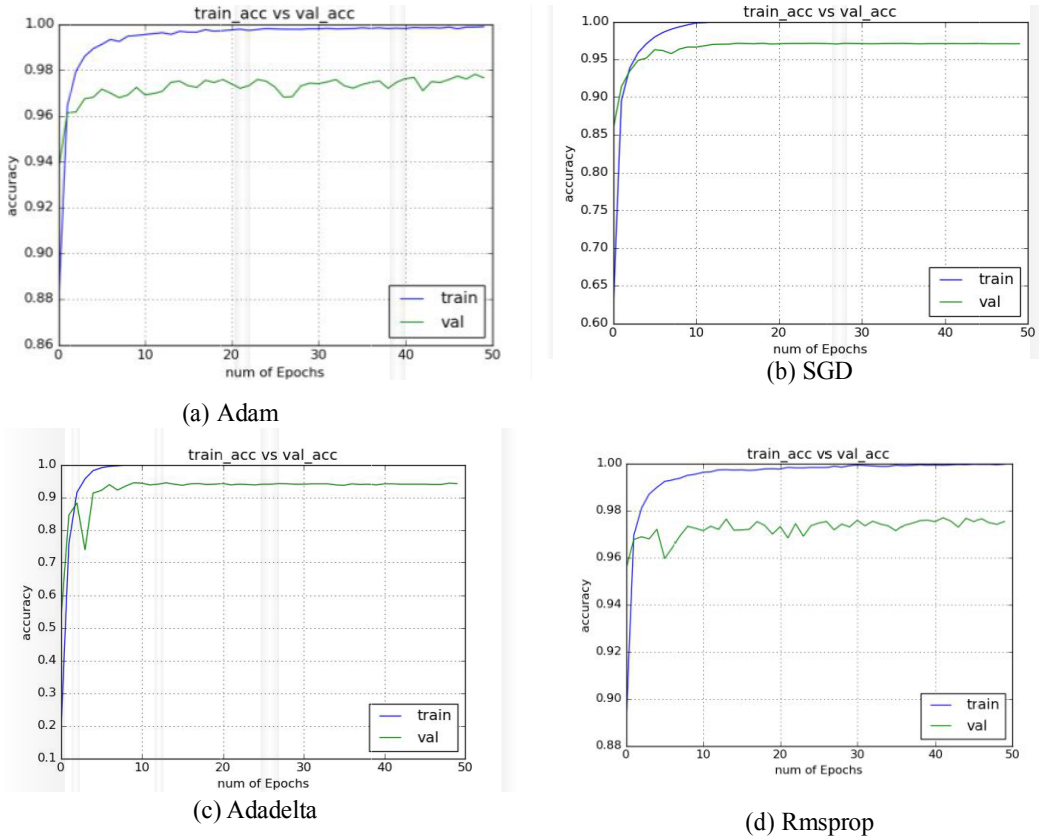
Figure 5: Accuracy graphs for CNN architecture II on database I



(a) Adam (b) SGD

(c) Adadelta (d) Rmsprop

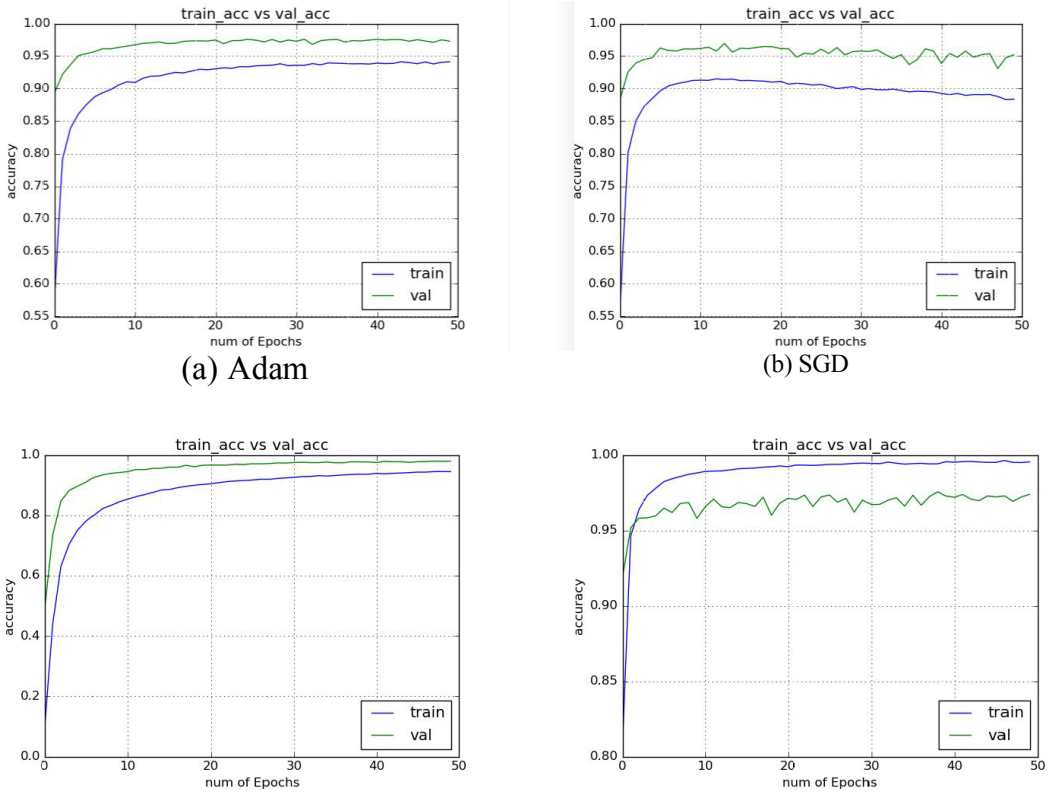
Figure 6: Accuracy graphs for CNN architecture III on database I



(a) Adam (b) SGD

(c) Adadelta (d) Rmsprop

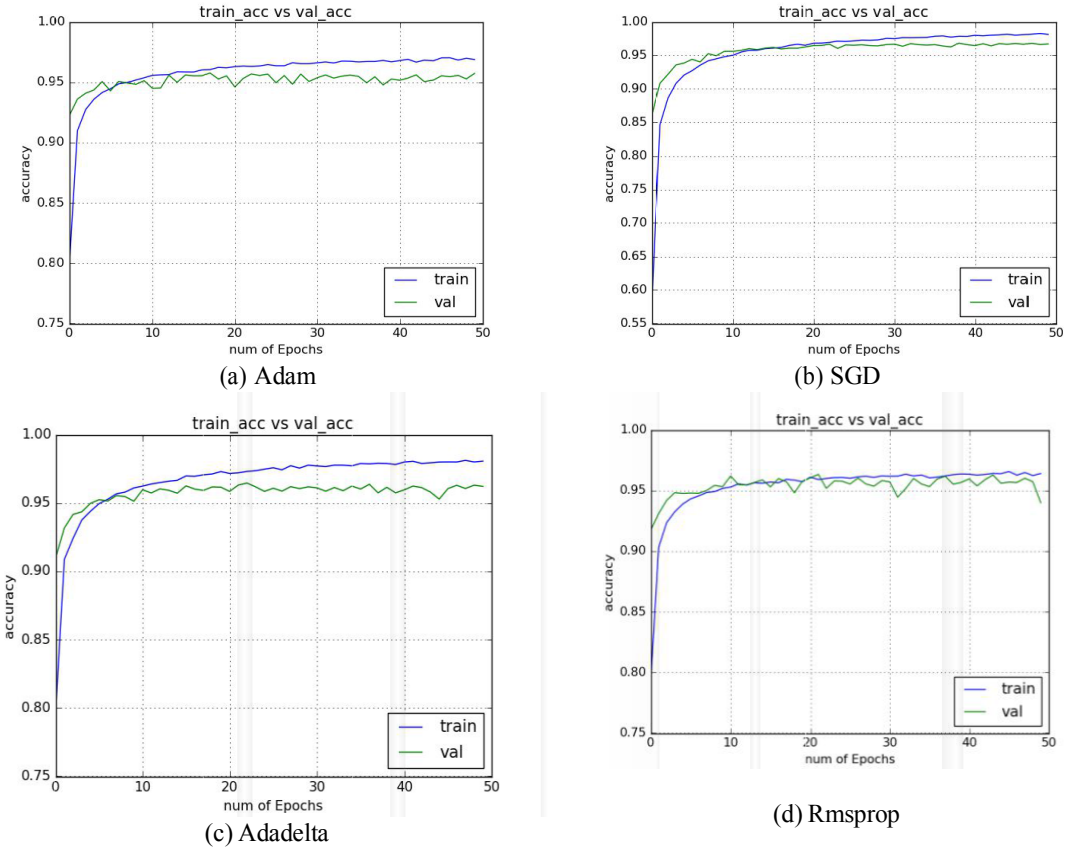
Figure 7: Accuracy graphs for CNN architecture I on database II



(a) Adam (b) SGD

(c) Adadelta (d) Rmsprop

Figure 8: Accuracy graphs for CNN architecture II on database II



(a) Adam (b) SGD

(c) Adadelta (d) Rmsprop

Figure 9: Accuracy graphs for CNN architecture III on database II

based on convolutional neural networks. The results can be further improved by adding linguistic information and generating larger databases.

5. Conflict of interest

Authors declare no conflicts of interest.

6. References

Acharya, S., Pant, A. K., and Gyawali, P. K., 2015. Deep learning based large scale handwritten Devanagari character recognition. In *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 1-6. doi:10.1109/SKIMA.2015.7400041.

Belhe, S., Paulzagade, C., Deshmukh, A., Jetley, S., and Mehrotra, K., 2012. Hindi Handwritten Word Recognition Using HMM and Symbol Tree. In *Proceeding of the Workshop on Document Analysis and Recognition*, pages 9-14. ACM. ISBN 978-1-4503-1797-9. doi:10.1145/2432553.2432556.

Bottou, L., 2012. Stochastic Gradient Descent Tricks. In *Neural Networks: Tricks of the Trade*, volume 7700, pages 421-436. ISBN 9783642352898. doi:10.1007/978-3-642-35289-8.

Deepti Khanduja, S. P., Neeta Nain, 2015. Hybrid Feature Extraction Algorithm for Devanagari Script. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 15:2:1-2:10.

Glorot, X. and Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249-256. PMLR.

Gyanendra K.Verma, P. K., Shitala Prasad, 2011. Handwritten Hindi Character Recognition Using Curvelet Transform. In *Information Systems for Indian Languages*, pages 224-227. Springer.

Hanmandlu, M., Grover, J., Madasu, V. K., and Vasikarla, S., 2007. Input Fuzzy Modeling for the Recognition of Handwritten Hindi Numerals. In *Information Technology, 2007. ITNG '07. Fourth International Conference on*, pages 208-213. IEEE.

Haykin, S., 1998. In *Neural Networks: A Comprehensive Foundation*, 2. Prentice Hall.

Kekre, H. B., Thepade, S. D., Sanas, S. P., and Shinde, S., 2013. Devnagari Handwritten Character Recognition using LBG vector quantization with gradient masks. In *2013 International Conference on Advances in Technology and Engineering (ICATE)*, pages 1-4. doi:10.1109/ICAdTE.2013.6524768.

Kingma, D. and Jimmy, B., 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, pages 1-15.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278-2324. ISSN 0018-9219. doi:10.1109/5.726791.

Madhuri Yadav, R. K. P., 2018. Hindi handwritten character recognition using oriented gradients and Hu- geometric moments. *Journal of Electronic Imaging*, 27(5):051216.1-051216.11.

Prasad, S., Verma, G., Singh, B., and Kumar, P., 2012. Basic handwritten character recognition from multi-lingual image dataset using multi-resolution and multi-directional transform. 10.

Sarkhel, R., Das, N., Das, A., Kundu, M., and Nasipuri, M., 2017. A multi-scale deep quad tree based feature extraction method for the recognition of isolated handwritten characters of popular indic scripts. *Pattern Recognition*, 71: 78-93. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2017.05.022.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15(1): 1929-1958. ISSN 1532-4435.

Tieleman, G. H., 2012. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. In *COURSERA: Neural Networks for Machine Learning*.

Yadav, M. and Purwar, R., 2017. Hindi handwritten character recognition using multiple classifiers. In *2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence*, pages 149-154. doi:10.1109/CONFLUENCE.2017.7943140.

Zeiler, M. D., 2012. ADADELTA: an adaptive learning rate method. In *arXiv preprint arXiv:1212.5701*.