Handwritten Character Recognition : Using CNNs

I. Abstract

In handwritten character recognition, computers try to interpret the scanned or uploaded images of handwritten texts. HCR is important in improving communication between humans and computers. This task can be strenuous due to ambiguity and unevenness in handwriting styles, strokes for each varying individual. Also the possibility of poor quality of source image makes the task more challenging. Several methods have been used in the past for handwritten character recognition problem. This paper presents a solution for handwritten character recognition using convolutional neural networks - CNN. We have proposed a CNN architecture that uses keras [5] as an interface for tensorflow [6] library. The model has been validated for english as well as devanagari scripts. Dataset put-to-use for english is 'EMNIST_ByClass' [1] and for devanagari is 'devanagari handwritten character dataset - DHCD' [2]. The model achieved 87.20% accuracy for EMNIST_ByClass and 98.19% accuracy for DHCD dataset.

Index terms : CNN - convolutional neural networks, EMNIST - extended MNIST, DHCD - devanagari handwritten character dataset, ReLU - Rectified linear function, HCR - handwritten character recognition.

II. Introduction

Handwritten text recognition is one of the most important and broad topics of research in image processing as well as pattern recognition. It has numerous applications in various domains - digitization of old data, signature recognition in banks and to help the partially sighted and blind by the creation of an interface that converts handwritten text to speech, to name a few. Character recognition is a first step in this direction. The images of handwritten characters are either uploaded or scanned, and to computers images are just numbers. Each of these numbers is represented by a pixel having a value between either '0 to 1' or '0 to 255'. For a grayscale image each pixel has only one value but for colored images each pixel has three values - red, green, blue. So, grayscale images can be represented as 2-dimensional arrays and colored images as 3-dimensional arrays (three 2 dimensional arrays stacked on top of each other). Each image is unique in terms of its features. Traditional way for extracting these features would be to leverage knowledge about different fields and extract them manually, which is inefficient because images can have variations in lighting, illumination, scale, viewpoint, and there is also a possibility of occlusion and background clutter. Our model should be invariant to all of these changes.

Now neural network based approaches allow us to learn different features directly from the data and hierarchy of features to construct a representation of an image internal to the network. Extracting features using fully connected neural networks has limitations as they require 2 dimensional images to be converted into 1 dimensional vectors, which of course loses spatial information and also increases trainable parameters exponentially. CNNs preserve spatial information by connecting only patches of input image and not the entire image to neurons in hidden layers.

A. Convolution: It is an element wise matrix multiplication operation. Where one matrix is of input image and the other is of filter.
B. Filters: In CNN, filters are small matrices of weights. The value of each filter is learned during the training process. So CNNs can find more meanings from images than human defined filters might be able to find.

C. Feature map: Feature map of CNN, captures the result of convolution operation. Whenever the pattern in the filter is conveyed 5in the input image , the value of the feature map is going to be high.
D. Strides: If the image is larger than the filter, we slide the filter to different parts of the image and perform convolution operation. The number of pixels by which we slide the image is called strides.
E. Padding**:** The value of stride is often kept 1 but it can be increased, so the size of image needs to be increased as well by a few pixels in order to fit the filter at the edges of the image.
F. Activation function: Activation function is a node that is put at the end or in between neural networks. They help to decide what is to be fired to the next neuron.
G. Pooling: Pooling layer can be seen between convolution layers in CNN architecture. This layer reduces the number of parameters and computation in the network.

Learning weights of different filters and extracting features using convolution operation is core to CNN.

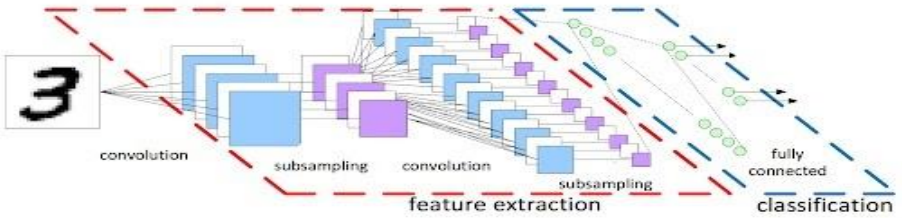

Figure 1 CNN architecture example

III. Literature Review

Even before deep learning, HCR problems have been tried to solve using different techniques, but their accuracies were pretty low. In [1], authors Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik, instead of some state-of-the-art techniques, provided a baseline for validation using a linear classifier and OPIUM (Online Pseudo-Inverse Update Method) based classifier. For the EMNIST_byClass dataset, linear classifier reports an accuracy of 51.80% and OPIUM based classifier 69.71% [1]. CNNs can be incorporated with other methods to get higher accuracy. Peng and Yin [3] presented the application of Markov random field-based CNNs. Sometimes different CNN models are combined into a committee, improving model accuracy [4]. We have created a model with three convolutional layers and two dense layers, which achieves an accuracy of about 87.20% for the EMNIST_byClass dataset and 98.19% accuracy for DHCD dataset.

IV. Methodology

We have used convolutional neural networks to achieve higher accuracy. Before training our model, we needed to apply a few data pre-processing techniques. Since the images in EMNIST are all reversed and rotated we used an inbuilt NumPy [7] transpose function to fix them.

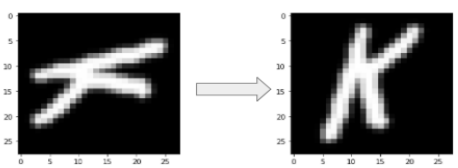

Figure 2

We have used keras [5] as an interface for the tensorflow [6] library to create our CNN model. The most common architecture of CNN models is SEQUENTIAL. Sequential model is a linear stack of layers hence is appropriate when there is only one input tensor and one output tensor. Our defined model has 10 layers.

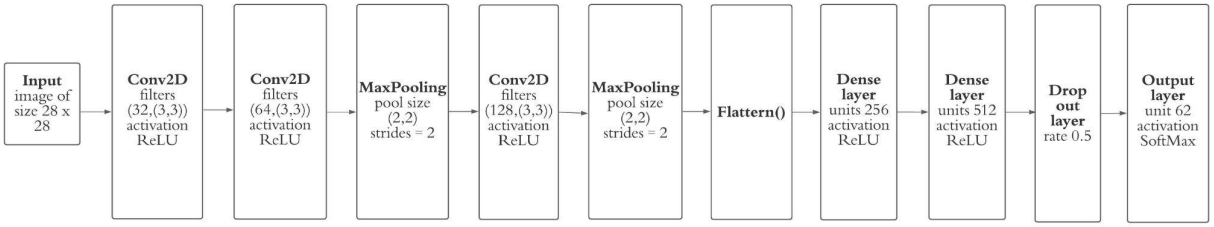

Figure 3 CNN Model

Feature extraction

Layer 1,2,4 - Convolutional layers :

The main function of convolutional layers is to find distinct features of training images. Convolution between filters and input image results feature maps, indicating locations and strengths of detected features in the input image. All the three convolutional layers use filters of size 3 x 3. First layer produces 32 feature maps using 32 different filters, similarly layers 2,4 produces 64 and 128 feature maps respectively. Activation function for all the three layers is the same and that is 'ReLU'. It is a piecewise linear function that will output the input directly if it is positive, otherwise it will output zero. We have used ReLU because it overcomes the vanishing gradient problem,which allows models to learn faster and perform better.

Layer 3,5 - Max Pooling layers :

Max Pooling layers apply max pooling on the feature maps. When images are too large, we need to reduce the no. of trainable parameters. Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus the output after max-pooling layer would be a feature map containing the most prominent features of the input image. This way max pooling reduces the size of the image.

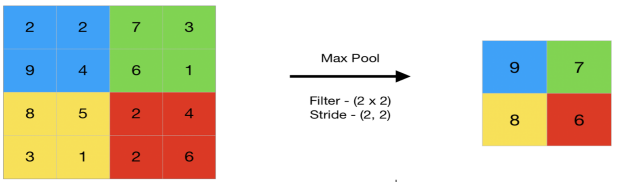

Figure 4 Max Pooling

Layer 6 - Flatten layer :

Flattening is converting data into a 1-dimensional array. This long feature vector is connected to the dense layers.

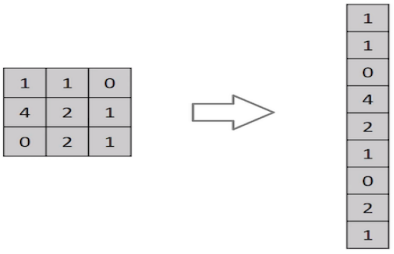

Figure 5 Flattening

Classification

Layer 7,8 - Fully connected (dense) layers :

Here each neuron receives input from ALL the neurons of the previous layer, hence it is called fully connected or dense layer. Each neuron computes the weighted average of its input and this weighted average is passed through activation function (in this case ReLU). Result of this activation function is treated as the output of that neuron. The same process is carried out for all the neurons.

Layer 9 - Dropout layer :

Dropout is a technique used to prevent a model from overfitting by removing neurons that can make the model bulky and increase the training time. Here the rate is 0.5, that suggests ½ fraction of the input units will be dropped.

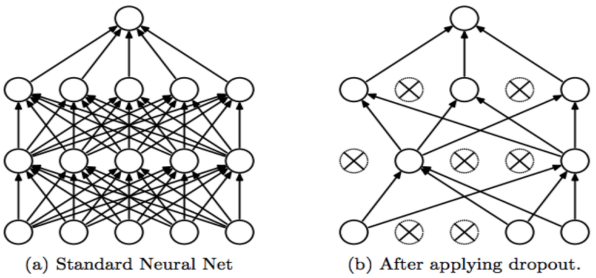

Figure 6 Dropout layer

Layer 10 - Output layer :

Output layer is responsible for producing the final result. Our model has 62 neurons in the output layer corresponding to 62 classes. Here the activation function is 'SOFTMAX'. It will calculate the probability of each 62 class for a particular test image. The class with the highest probability is the output.

V. Dataset

The EMNIST dataset is derived from NIST special database 19. It contains digits as well as alphabets arranged by class [1]. It has 62 classes consisting of [0-9], [A-Z], [a-z] [1]. There are 814,253 images in total with 697,931 training samples and 116,322 testing samples. Each image is 28 x 28 pixels. The data however is uneven, there are far more digit samples (almost half of total samples) than letter samples. Total number of samples per class for letters is similar to the frequency of letters in the english language [1].

The DHCD dataset has 36 classes consisting of 36 consonants [2]. There are 72,000 images in total. Each class has 2000 images [2]. There are 61,200 training samples and 10,800 testing samples. Each image is 32 x 32 pixels, with the actual character centered within 28 x 28 [2].

VI. Results

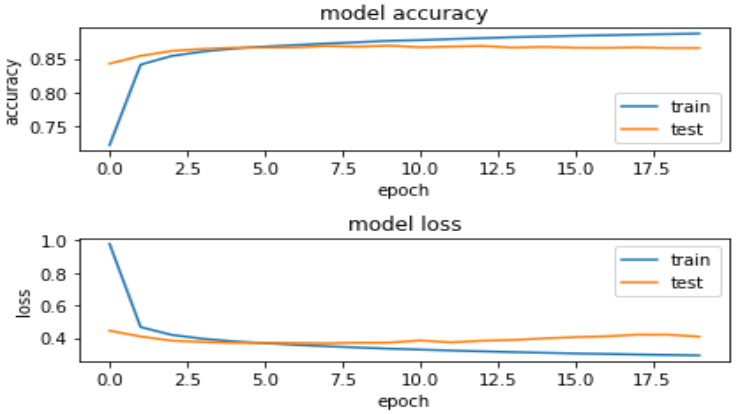

Figure 8 model accuracy and model loss for EMNIST dataset

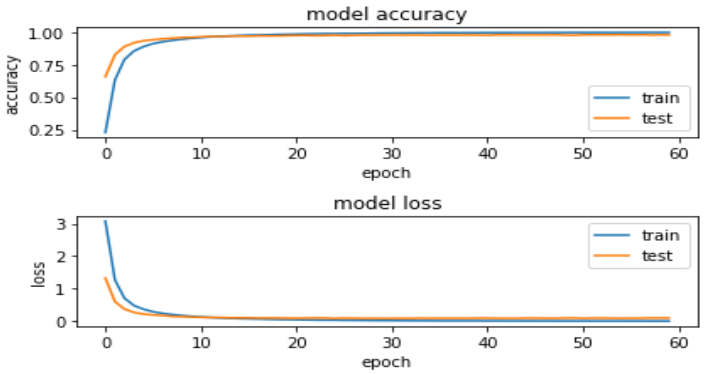

Figure 9 model accuracy and model loss for DHCD dataset

We tested our model with architectures varying in depths and no. of parameters. We trained our model using the 'Adamax' classifier with a learning rate of 0.0001. We tried different combinations for classifier and learning rates but this gave the best accuracy. From the figure above it is observed that the validation accuracy increases exponentially initially, and almost steadily later on. Similarly it is observed that validation loss decreases as no. of epochs increases. The model was evaluated on testing data provided by EMNIST [1] and DHCD [2]. Amongst 116322 testing images for EMNIST, the model predicted 101153 images correctly and 15196 images incorrectly. Amongst 10799 testing images for EMNIST, the model predicted 10604 images correctly and 196 images incorrectly.

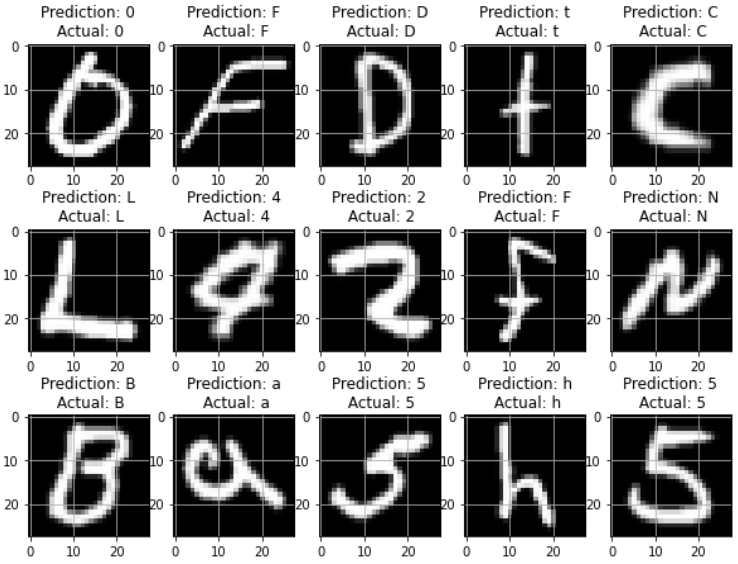

Figure 9 examples of few correct predictions for EMNIST

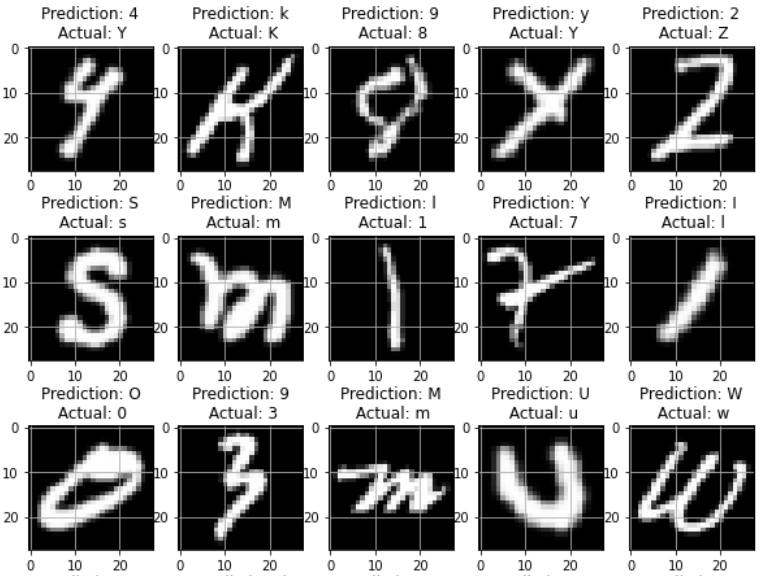

Figure 10 examples of few incorrect predictions for EMNIST

VII. Conclusion

Humans can effectively recognize and read any image or text with a blink of an eye ,without really appreciating it, while in the real world it's a very challenging task. Deep learning is bringing forward the revolution as well as computer vision algorithms and applications to help solve these tasks. Convolutional neural networks provide a more scalable approach for image classification by leveraging principles from linear algebra, specifically matrix multiplication to identify patterns within images. EMNIST datasets, a suit of six datasets, is intended to provide a more challenging alternative to the MNIST dataset. It provides more image samples, more output classes and hence more challenging classification tasks. We used CNNs for the task of HCR. Our model achieved 87.20% accuracy on EMNIST dataset. There is wide scope available in the field of HCR in future as it plays an important role in various fields such as reading postcard addresses, bank check amounts and forms, digital libraries and many more.

VIII. References

1. G. Cohen, S. Afshar, J. Tapson and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2921-2926, doi: 10.1109/IJCNN.2017.7966217.
2. Acharya, Shailesh et al. “Deep learning based large scale handwritten Devanagari character recognition.” *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)* (2015): 1-6.
3. Peng, Y.; Yin, H. Markov Random Field Based Convolutional Neural Networks for Image Classification. In *IDEAL 2017: Intelligent Data Engineering and Automated Learning*; Lecture Notes in Computer, Science; Yin, H., Gao, Y., Chen, S., Wen, Y., Cai, G., Gu, T., Du, J., Tallón-Ballesteros, A., Zhang, M., Eds.; Springer: Guilin, China, 2017; Volume 10585, pp. 387–396.
4. Cireşan, D.C.; Meier, U.; Gambardella, L.M.; Schmidhuber, J. Convolutional Neural Network Committees for Handwritten Character Classification. In Proceedings of the 2011 International Conference on Document Analysis and Recognition, Beijing, China, 18–21 September 2011; pp. 1135–1139.
5. F. Chollet et al., “Keras,” https://github.com/fchollet/keras, 2015.
6. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M.

Kudlur, J. Levenberg, D. Mańe, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Víegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

7. Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 0.1038/s41586-020-2649-2.