Optical Character Recognition using Deep Learning

Vatsal Goel
IIT Guwahati
vatsal29@iitg.ac.in

Abstract

I have created a character level and a word level neural network to recognize hand-written text. I used 2 datasets for this purpose, EMNIST Dataset [1] for character level recognition and IAM Handwriting Dataset [2] for word level recognition. The character level model uses a CNN architecture and achieved an accuracy of 88% . The word level model uses a CNN architecture which feeds into a LSTM layer and this model achieved an accuracy of 79%.

1. Dataset and Pre-processing

1.1 Character recognition

I used the EMNIST Balanced Dataset [1] for character level recognition. This dataset consists of 131,600 images classified upon 47 classes. The images have been pre-processed into high contrast greyscale images of size 28 x 28 pixels. Since the size of the dataset is already quite large, I didn't use data augmentation, but it can always be used to better performance.

I downloaded the csv files for this dataset from Kaggle [3]. Thus I was directly able to import the images and labels as numpy arrays of size (m, n) and (m, 1) using pandas library, where m is the total number of examples and n is 28 x 28. Then I reshaped my image array to size (m, 28, 28, 1) and converted the labels array to one-hot vectors which gave me an array of size (m, 47), 47 being the number of classes. The last step was to divide the image pixels by 255 for the purpose of normalizing.

1.2 Word recognition

For word level recognition, I used the IAM Handwriting Dataset [2]. Keeping segmentation aside, I used the images of separated words for my image dataset. This dataset consists of 115,320 segmented words as PNG files of varying shapes.

Since the words had varying shapes, first I resized all the images to a fixed size. I added white padding to all the images till they square shaped and then resized them to 64 x 64 pixels using the openCV library. Thus I was able to import my images into a Numpy array of size (m, 64, 64, 1) and then divided the pixels by 255 to normalize. The code for this can be found in the file Images.py.

I extracted the labels from the xml meta-data file using BeautifulSoup [4]. I padded all the shorter words with spaces to have a constant length. By doing this I was able to split the words into characters and then store them as one-hot vectors. Thus the final shape of my label vector was (m, 10, 53). I kept the length of longest word as 10 so I made all words of length 10, and I used 53 classes, 26 lowercase letters, 26 uppercase and 1 class for all special characters like “,/. etc.

2. Models

2.1 Character Recognition

For character recognition, I used a CNN architecture, the model consisted of 3 layers of CNN, Batch Normalization and 2 layers of Max Pooling. The first 2 layers of CNN used filters of size (3, 3), padding as same and channels of 32 and 64 respectively. Both layers of Max Pooling used filters of size (2, 2). The last layer of CNN used filters of size (4, 4), padding as zero and channels equal to 128. This model was followed by 2 Dense layers of size 256 and then 47. The activations used for all layers were ReLu except for the last layer which used Softmax. I added dropout to the dense layers with p = 0.5 to avoid overfitting.

2.2 Word Recognition

The word recognition model used a CNN architecture followed by a LSTM layer. I used the CNN model to extract features from the input word image before passing it to the RNN model. There were 3 layers of CNN, Batch Normalization an Max Pooling of filters (3, 3) and increasing channels of 16, 32 and 64 respectively. The output was then passed through 10 layers of 64 Long Short Term Memory (LSTM) cells. Each of the LSTM layers output was then passed through a fully connected layer with softmax activation which gave the probabilities for all characters.

2.3 Training

I used the categorical cross-entropy loss function for both of the models. For training I used the Adam optimizer with its default values of hyperparameters to update the weights. Also I trained my dataset in minibatches of size 512.

3. Results

Using the above defined models I was able to achieve 88% accuracy for character recognition after running on 40 epochs and 79% accuracy on word level recognition after running on 60 epochs.

The plots for loss and accuracy for both models are given below in Figure 1 and Figure 2 respectively.

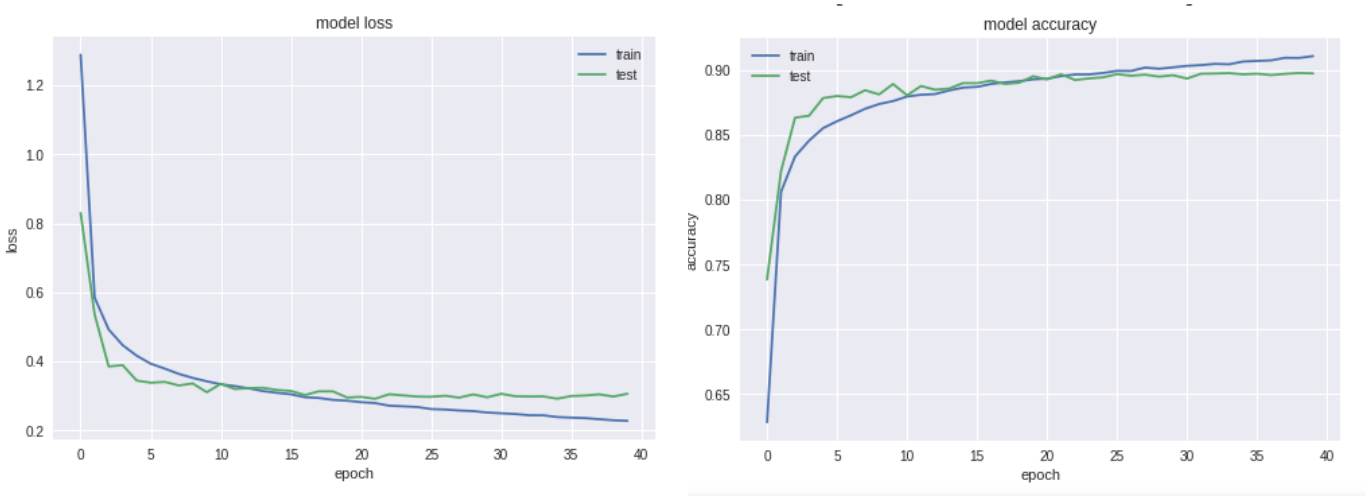

Figure 1: Character Recognition Loss and Accuracy Variation

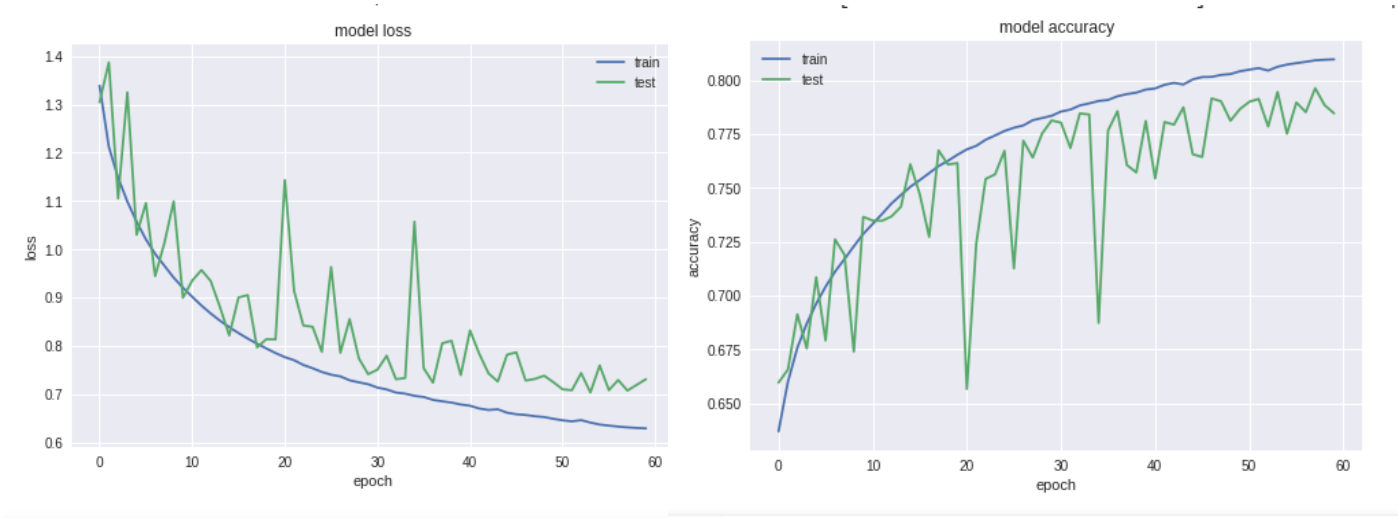

Figure 2: Word level Recognition Loss and Accuracy Variation

4. Future Work

I would be continuing my work on Optical Character Recognition, searching for new models to improve accuracy, and also use segmentation techniques to separate words from sentences or a page of text, which would be more practical and can have quite useful applications.

References

[1] EMNIST Dataset - www.nist.gov/itl/iad/image-group/emnist-dataset

[2] IAM Handwriting Dataset - www.fki.inf.unibe.ch/databases/iam-handwriting-database

[3] Kaggle link for EMNIST dataset - www.kaggle.com/crawford/emnist

[4] Beautiful Soup Documentation - www.crummy.com/software/BeautifulSoup/bs4/doc/