ECE - 1512

Final Project

Equation's Image to LaTeX Markup generation

Anshul Verma

1004730703

Date due: 02^{nd} December 2019

Date handed in: 02^{nd} December 2019

Contents

List of Figures

1 Introduction

1.1 Motivation

The task of generating text from an input image is considered to be very challenging because it involves Natural Language Processing and Reinforcement Learning. The problem has been solved by a few researchers in the community and the best possible model for the problem to date is a seq2seq model with features like attention and beam search in the decoder. A seq2seq model is basically a encoder-decoder model [4]. The problem of generating the latex-markup from an equations image is similar to that of generating captions for an image and it has been tackled earlier by a lot of people the reader can have a lot at these [3], [5] for more information on this problem.

The solutions to the problem are all Deep Neural Network approaches which requires a lot of computational power and training data to produce sensible result. Even after that it doesn't work well for an image which does not have a lot of equations similar to them in the training dataset. The biggest available dataset for this purpose is im2latex-100k which has around 100k images of equations and their latex-markup. But this dataset still doesn't cover all the equations existing in all the fields. Now its very likely that a researcher who is willing to use this facility has a lot of different character and equations which doesn't have anything similarity with the dataset used to train the network (for example, a Physicist doing string theory). Since there are not a lot of big-data set of equations common to a particular field I have tried to tackle this problem of converting the equation's image to their latex-markup by using transfer learning to reduces the number of training parameters in the final seq2seq model approach. Along with this an OCR based approach for generating the latex-markup from an image has also been implemented and explained.

1.2 Problem Statement

For the purpose of this project two different approaches to convert an equation's image to their latex markup has been discussed and implemented. One of the approaches is a Optical Character Recognition based technique where the image is segmented into different characters in an equation after that a trained CNN is used to predict character in each of the segment(block) of the original image and the final output is then generate using these predictions of character in each segment and the order of occurrence of the segments in the image.

The second approach is a Seq2Seq model with a convolutional encoder and LSTM-decoder. For the encoder a CNN which has been trained to classify 83 different characters on a separate-dataset is used and the first fully connected layer of the network is considered to generate the feature vectors which are then transferred to the decoder. All the current state of art solution to the problem have attention and beam-search like feature in their decoder. But the final trained Seq2Seq model didn't give produce consistent and good results so I have decided not to discuss in great depth. I will only brief the model that I implemented and will also share with you the code for it if asked. There are a few possible reasons as to why the Seq2Seq model didn't perform well for me. One can obviously be that the model that I used was too simplistic another one can be that I didn't have sufficient data and possibly the feature vectors which I thought are actually useful to the decoder ended up being not so useful. The usual trend for such a

Seq2Seq model is a convnet-encoder with GRU-decoder [6] where the entire model is trained together on the data-set. Since I only had limited data for hand-written equations I decided to use transfer learning and used the classifying convnet's fully-connected layer 1 as the encoded sequence.

1.3 Data-set

For the purpose of this project I am using CHORME data-set, which is a handwritten equation data-set which is even harder to convert to latex-markup than the printed equations because of different styles and types of handwriting in the images and its a relatively smaller data-set compare to im2latex-100k as it has only 10k images of equations but in .inkml format. Which I needed to convert to .png to be able to process these images for my task.

The data-set has a wide variety of equations from single character to more complicated equations with lot of fractions, superscript and subscript. The LaTeX formulas for the codes are available online and were download along with the dataset.

For training the Convolutional Neural Network I used the the Handwritten Math Symbol dataset provided by Kaggle it has around 378k images of 83 different math symbols which covers most(not all) of the characters that occur in CHORME dataset's equations.

1.4 Preprocessing

The dataset came with a set of instructions which was followed. Since the data was in .inkml format, inkml2img.py was used (available in github) to transform these images to .png format and save them in a folder which was then used later for the purpose of this project. (*Note*: Can provide with inkml2img.py if required)

2 Classifier for Transfer Learning

2.1 Classifier model

I looked at different available trained Convolutional Neural Network classifiers to classify different math symbols but none of the models were built to classify a big enough set of characters. So I decided to train a classifier for my own purpose on a data-set which has characters closely realted to the CHORME dataset. Handwritten Math Symbol dataset provided by Kaggle was used to train the classifier but it doesn't have all the characters that occur in the equation's data-set.

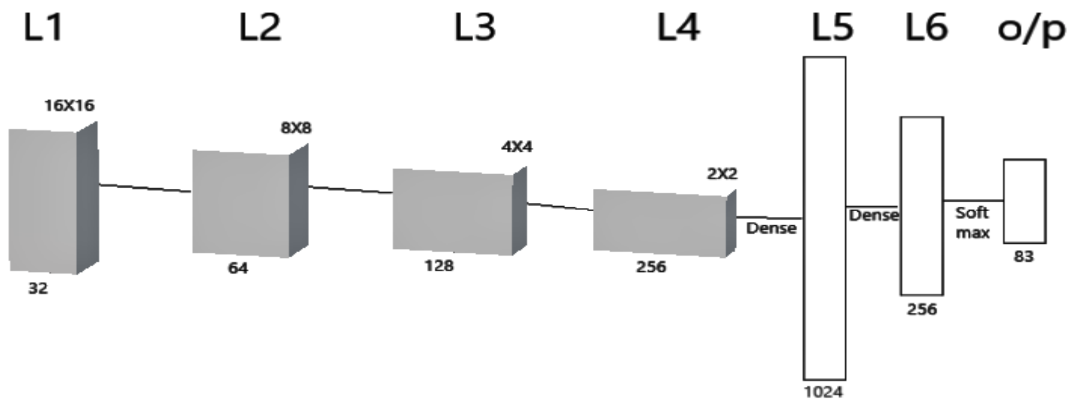
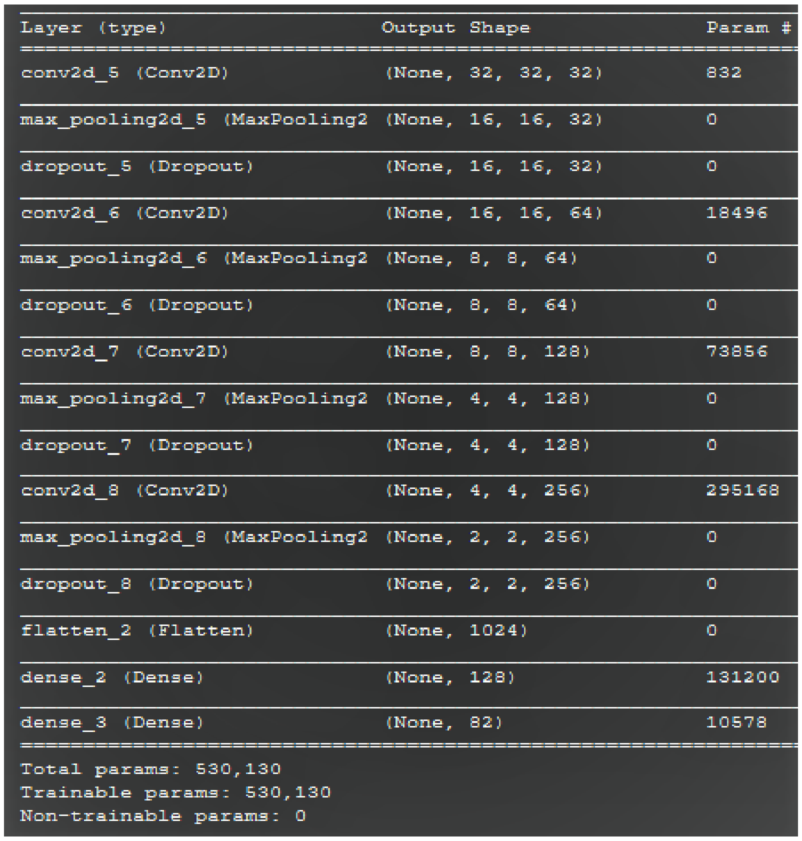


Figure 1: CNN Visualization.

I used a CNN with 4-convolutional layers and 2-fully connected layers for the purpose of classification of math symbols the architecture of the model is inspired by CIFAR-10 model [2]. I have used one less convolutional layer with slight variations to the model's structure and then used it for my purpose of building a classifier.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 32, 32, 32)        832
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
dropout_8 (Dropout)          (None, 2, 2, 256)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               131200
_________________________________________________________________
dense_3 (Dense)              (None, 82)                10578
=================================================================
Total params: 530,130
Trainable params: 530,130
Non-trainable params: 0
```

Figure 2: Details of used CNN model.

2.2 CNN Training

Since the task of classifying a character on a white background is easier than the task of classifying an object in a varying environment. Binary images after thresholding were used as inputs to the model. Binarizing the input image didn't lead to any significant drop in performance but it did shorten the duration of training significantly. With binary inputs the model took around an hour to train for 20 epochs. Whereas for a normal image it took around 4hours to train the CNN for just 10 epochs.

(*Note*: The CNN used for the results in the presentation was not for a binary image and had lower performance than this even with more convolutional layers, it had 6 convolutional layers and 2 fully connected layers. The results for the CNN used to generate the results in the slides can be provided if required.)

This model work with binary images of size (32, 32, 1), I produce this binary image by using threshold_mean() filter of sklearn.filter module. Training accuracy for this model is 92% whereas the validation accuracy is 93% which are better than results obtained for any of the other CNN model that was train on a RGB-image.

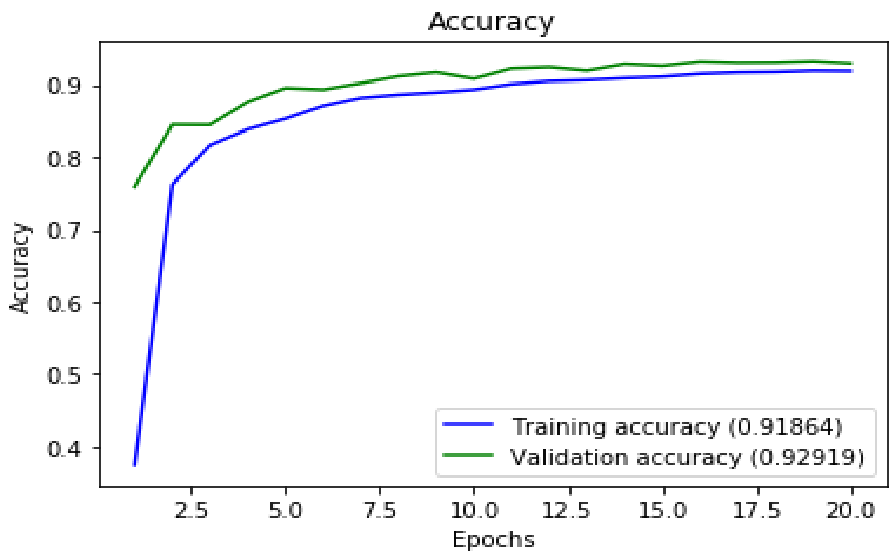


Figure 3: Accuracy of the CNN model with epoch.

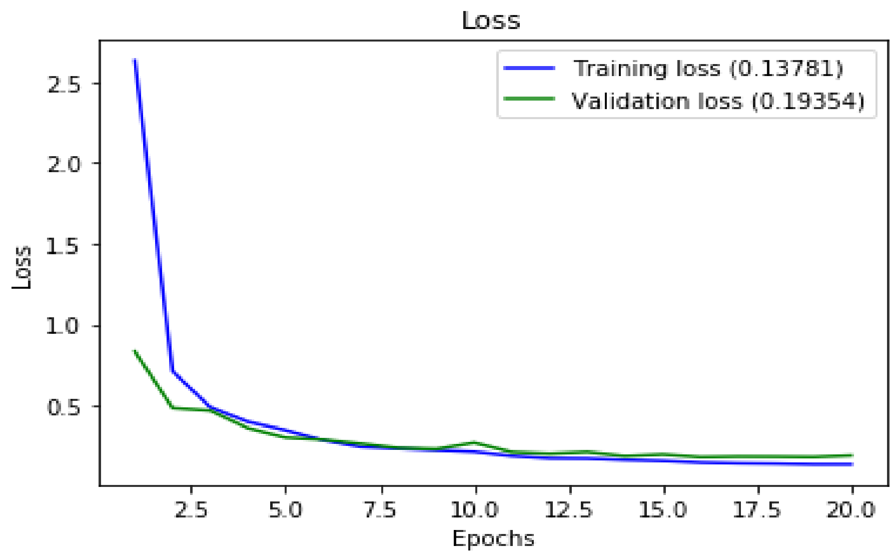


Figure 4: Loss of the CNN model with epoch.

2.3 Test Performance

The data-set does not have same weights for all the classes and the class-weight is very sparse. For some of the classes there are 50 images whereas for some others there are 5k images. So to try and avoid missing any of the classes in the test-set I manually splitted the data-set into a small testing and training subset. Making sure that the test-set has atleast one good image for all the classes. This is the reason why the test accuracy is higher than the training accuracy. The accuracy stat is still more useful than confusion matrix or roc-curve in this case because it ensures that almost all the classes (atleast the good images in each class) are classifiable. But the test-set is too small and although the model gets confused between some of the classes like z and 2 and q and 9 because they look similar. But the test set is split manually and consists of good images mainly so these confusions are not very evident in the Confusion

Matrix.

The classifier achieves 93% accuracy on the test-set and the confusion matrix and roc_curve for the classifier are shown below. (*Note:* The classifier does gets confused between similar looking classes but because the test-set is manually made and it is very small and consist of good images mainly is the reason why these confusions are not very evident in the Confusion Matrix.)

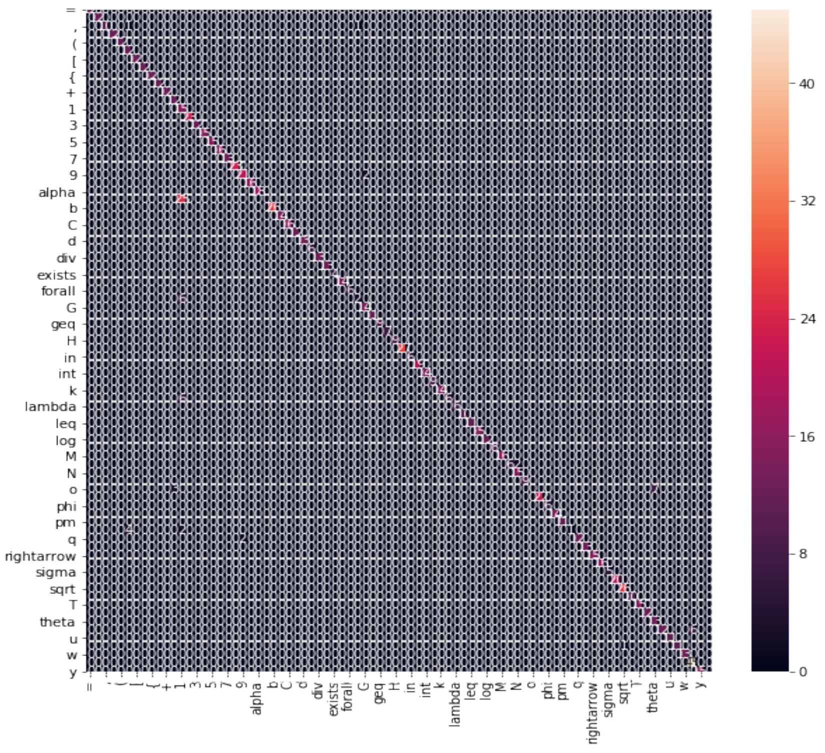


Figure 5: Confusion Matrix

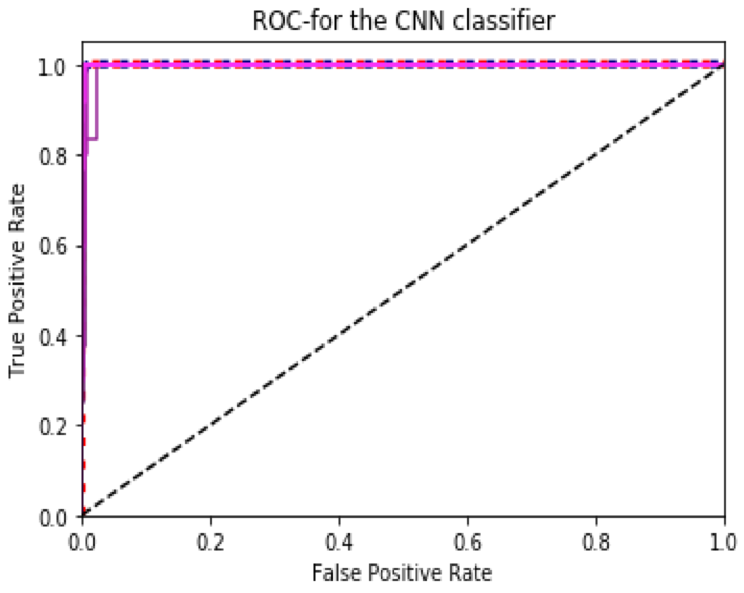


Figure 6: ROC for all the classes

3 Optical Character Recognition Approach

There are a lot of ways of finding different characters in an image and for the project an Image Processing based approach which is one of the easier approaches has been implemented. Despite being an easy approach it still work's in this case because recognizing characters in a constant background is much easier than doing the same in a varying environment and the problem that the project concerns with is characterizing black characters over a white background.

3.1 Image Processing Based Character Segmentation

3.1.1 Noise Reduction

The input image can have patches of dark spots and varying brightness to account for all such variation at first noise reduction is implement. There are several ways to reduce noise but the one that has been used in the final approach is a Gaussian Blur with a 5X5-kernel of constant. Since the data-set used doesn't have a lot of noise (beacuse its a conversion of .inkml to .png images) this works fine. But images taken using a phone or from some other source will need to be filtered appropriately. One can use different types of noise reduction approach and for that very reason in the code the kernel k has been defined as an argument which if is not passed is set to a constant kernel of size of (5, 5) for Gaussian blur.

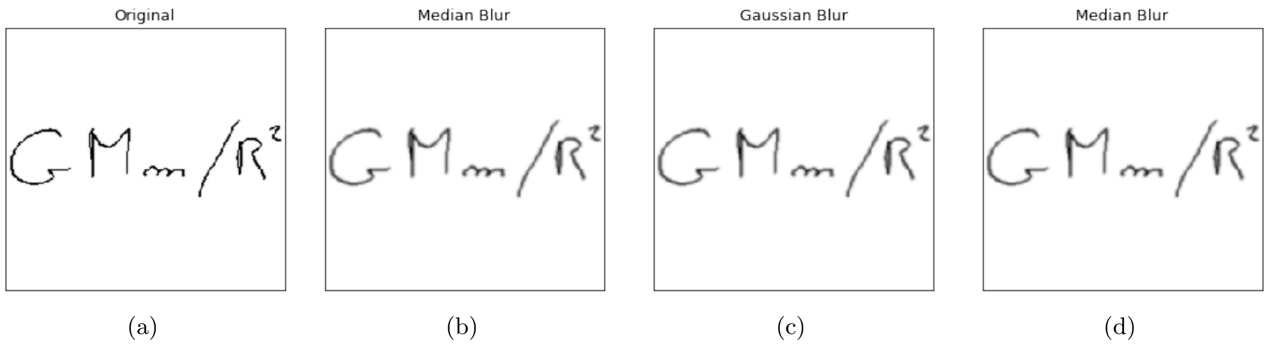


Figure 7: (a) Original Image, (b) Averaging (Mean Blurred for kernel (5,5)), (c) Gaussian Blurred for kernel (5,5) and (d) Median Blurred (kernel size 5X5).

3.1.2 Adaptive Thresholding

After noise removal the image is binarized to ease the process of finding contours in the original image. A normal threshold would work well in this case because we only have black images on a white background.

Another thresholding approach is called Adaptive thresholding and it takes a look at neighbourhood of a pixel and then decides the threshold value of that particular pixel this is a crucial since it allows us to use an image which has varying brightness. Adaptive thresholding is preferred over normal thresholding because of possible brightness variation in the image. For the purpose of this project binary inverse adaptive thresholding has been used. Also the image which goes as an input to the CNN is also thresholded.

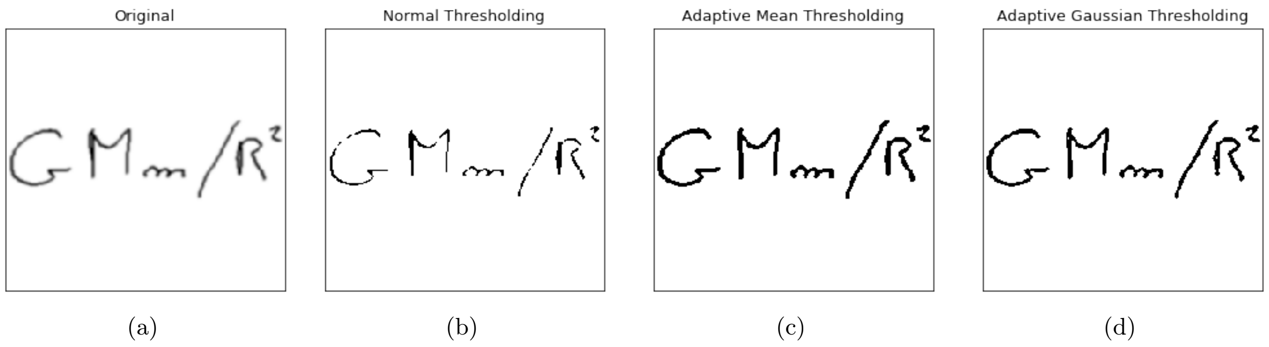


Figure 8: (a) Gaussian Blurred Image, (b) Mean Thresholding, (c) Adaptive Mean Thresholding and (d) Adaptive Gaussian Thresholding.(For Inverse Thresholding every white pixel becomes black and every black becomes white)

3.1.3 Dilating

It can be seen in the example above that after thresholding the characters in an image might have a slight disconnected point.Since these images are to be used to find contours of connected region its important that the image is dilated so that all the characters are completely connected and the identified segments have full but not partial character.

For my purpose I am using a dilating kernel of (5, 5) but (3, 3) also gave decent result when it was used in the dilating step. But (5, 5) gave some '=' sign in a same segment so it was selected to be the kernel for the rest of the data-set. But again this kernel is a function argument and can be changed.

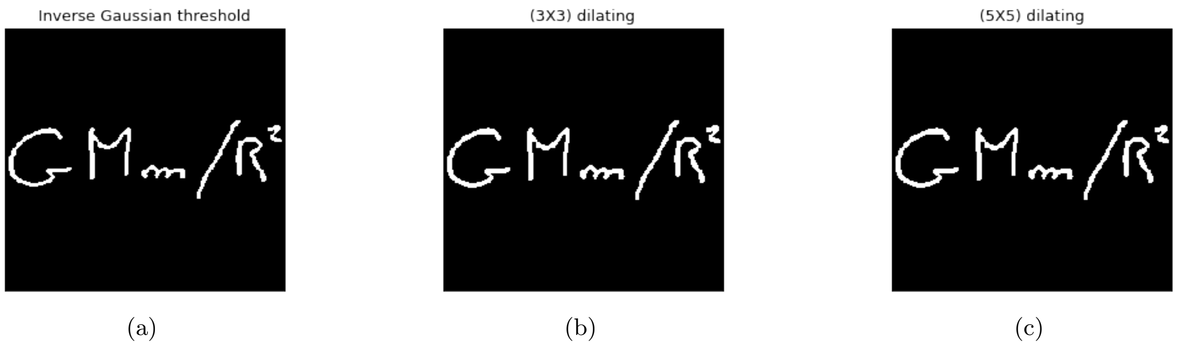


Figure 9: (a) Inverse Adaptive Gaussian Threshold, (b) Dilating with kernel (3X3) and (c) Dilating with kernel (5X5)

3.1.4 Rectangular Contours

Then the dilated image is used to find rectangular contours of the connected regions in the original image. These contours are then considered as segments of characters in the image. But some symbols like $\geq$, $=$, *and* ! will almost never be in one segment and this is one of flaw of using this segmentation approach.

But for basic equations this works pretty well and identifies the segments with single characters as long as they are written properly and are well spaced in the image like in the example 10 shown below

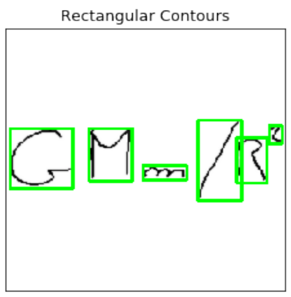


Figure 10: Caption

3.2 Character Recognition

The found segments in an image are then ordered based on their position in the original image. This segment is then padded with zeros around its boundary to make it bit more centered and then it is binarized and changed to the input size of the CNN. CNN then predicts the class of character in each segment which results in a sequential output corresponding to the order of characters in the equation which in this case is also considered to be the predicted latex-equation.

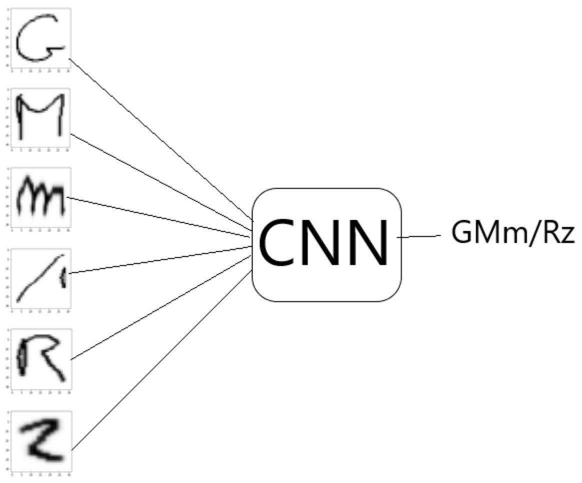


Figure 11: Character Recognition in sequence of their occurrence in the Image

3.3 Flaws

So this approach will not be able to detect any of the special latex character like ∧ , __, \frac, {, }, \left, \right etc. Other than these the model will also have difficulty in understanding terms like sin, cos, and tan in the equation because it is designed to characterise a single character and these terms are usually group of characters.

One of the measure to test accuracy of the strings is the BLUE score of similarity of the two strings. The final model gets 0.076 BLUE Score of similarity. The used data-set has a lot of equation with above features which results in a overall low similarity between the predicted and the actual latex-markup. Other than that even difference in spacing between the ground truth equation and the predicted-equation will lead a drop in similarity and as a result the overall similarity is quite low for this approach. But in the result section it can clearly be seen that the process work well for basic equations. Which has characters well spaced and written clearly.

4 Seq2Seq Model

This will only discuss this briefly discussed because the implemented model didn't give any satisfactory and consistent result. The architecture and the idea of the implemented approach will be discussed although the results were inconsistent and thus will not be added. (i.e. everytime I was getting a different and wrong predictions). (*Note*: Code for this section hasn't been added to the Appendix because it has not been used for any of the results presented in the report but can be provided if asked.)

4.1 Introduction

All the Seq2Seq models approaches available for this purpose are trained on im2latex dataset (for printed equations) which is a very large dataset, allowing training the entire model in the dataset [1]. However I am working with handwritten equations and the dataset which I am using has like $\left(\frac{1}{10}\right)^{th}$ the amount of data in im2latex dataset. So to account for this gap in the amount of available data, transfer learning was used.

4.2 Model Architecture

Character classifying CNN was used as a convolutional encoder. The first fully connected layer of the CNN were considered to be the feature vectors which are passed onto the decoder. These feature vectors are firstly connected to an intermediate dense layer with 256 cells and "tanh" activation. This layer is then passed as initial state of the LSTM layer with internal state-size of 256 cells in the decoder . The other input to these LSTM-decoder is the partial input equation's embedded vector. So if we want to predict the third word we do consider the second character as the input to the decoder and pass on the embedded vector for that word into the decoder as input so that it can predict the next character. The output of the LSTM layer is then passed onto a dense layer which learns embedding of the new character and reverses it to give us the next character.

4.3 Possible reasons for under-performance of Model

The possible reasons are simplicity of the decoder. But I could not add more layer because I have a limitation on the amount of data available. One possible way to work around this loop is to use data-augmentation.

Another possible reason for the poor performance can be that the feature vectors which I am transferring to the decoder are not passing in any useful information to the decoder and that is why the decoder is not learning anything. But this seems highly unlikely because the CNN was trained to classify the handwritten characters so it must be learning different features which are essential to differentiate between different characters.

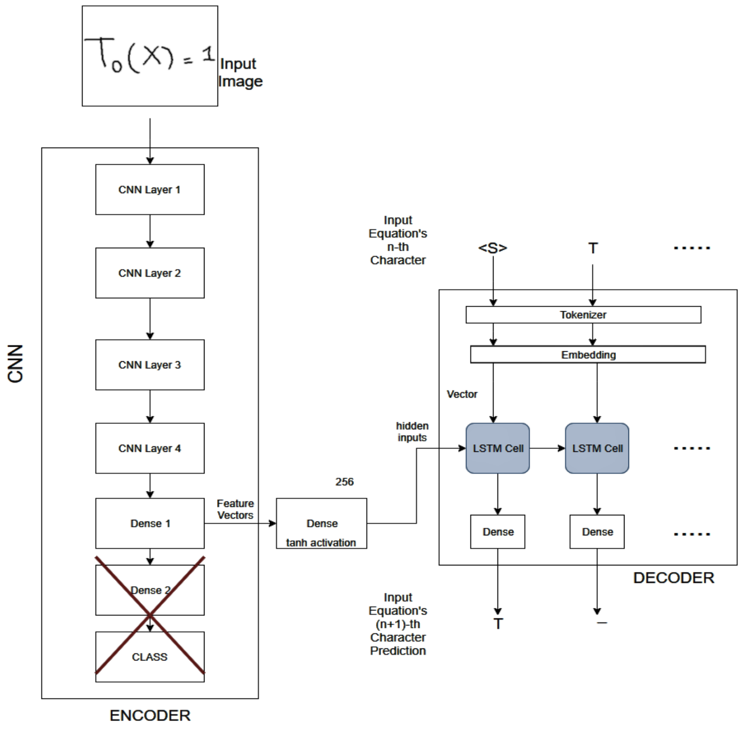
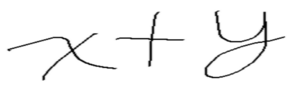
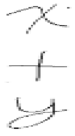
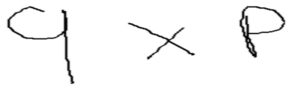
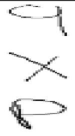
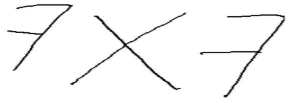
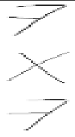
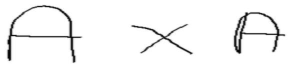
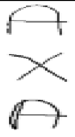
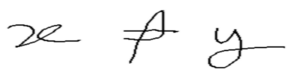
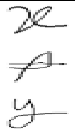
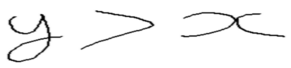
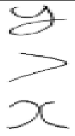
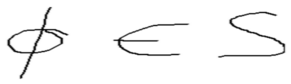
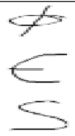
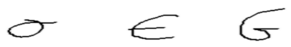
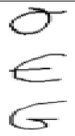


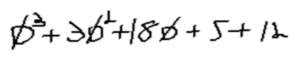
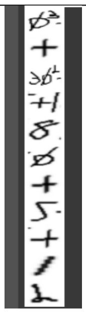
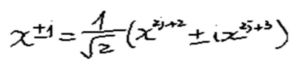
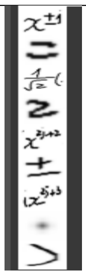
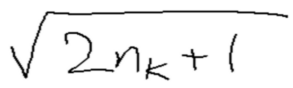
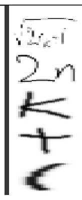
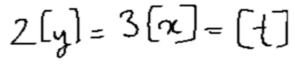
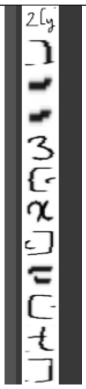
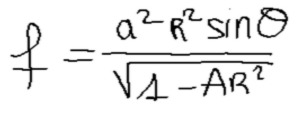
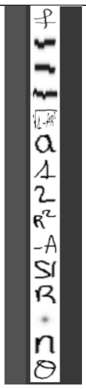
Figure 12: Architecture of the Seq2Seq I attempted to train. (i.e. Input image is binarized (32, 32, 1) Since the CNN used takes binary images to classify characters)

5 Results of the OCR approach

5.1 Basic Equations

Original Images	Bounding Boxes Order	LaTeX- Markup
		**Predicted :** X + y **Actual :** x + y
		**Predicted :** 9 times p **Actual :** q times p
		**Predicted :** 7 X 7 **Actual :** 7 times 7
		**Predicted :** A X A **Actual :** A times A
		**Predicted :** X + y **Actual :** x neq y
		**Predicted :** y gt x **Actual :** y gt x
		**Predicted :** phi in S **Actual :** phi in S
		**Predicted :** v in G **Actual :** sigma in G

5.2 Complex Equations non-ideal spacing

Original Images	Bounding Boxes Order	LaTeX- Markup
		**Predicted :** phi + X 4 X d + X + p R **Actual :** phi∧ 3 + 3 phi ∧ 2 + 18 phi + 5 + 12
		**Predicted :** X = tan z - pm X j y **Actual :** x ∧ { \pm j } = \frac { 1 } { \ { 2 } } (x ∧ { 2 j + 2 } \pm j x ∧ { 2 j + 3 })
		**Predicted :** pi lim k + 6 **Actual :** \sqrt { 2 n _ k + 1 }
		**Predicted :** 4 X log phi 3 pi M A log C 1 1 **Actual :**2 [y] = 3 [x] = [t]
		**Predicted :**f log tan tan lim A d 2 pi log G phi j N theta **Actual :** f = \frac { a ∧ 2 R ∧ 2 sin \theta } { \sqrt { 1 - A R ∧ 2 } }

6 Conclusion

For the project techniques to tackle the huge data requirement of Deep Neural Network are attempted to generate latex-markup from their equation's image. The model is not perfect and it has ita flaws as it only works well for basic equations as shown above and for more complicated equations with superscript, subscript fractions etc. the suggested OCR based model under performs.

Whereas the trained Seq2Seq model using a pre-trained CNN for encoding the sequence turned out be too simplistic to learn how to decode the encoded sequence and ended up giving random outputs. This approach of transfer learning can be suggest approach can be improved upon by using data-augmentation for handwritten equations as it will allow us to add more layers to the decoder. But overall the use of transfer learning will still require less amount of data compare to the current approaches where the entire model is trained together.

References

[1] Guillaume Genthial and Romain Sauvestre. Image to latex. 2016.

[2] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. Unpublished manuscript, 40(7):1–9, 2010.

[3] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7008–7024, 2017.

[4] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: A fully end-to-end text-to-speech synthesis model. arXiv preprint arXiv:1703.10135, 2017.

[5] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4651–4659, 2016.

[6] Zhengqing Zhou, Junwen Zheng, and Zhongnan Hu. Image to ixt) k: A neural network approach.