Springboard

READING BENGALI

USING PYTORCH TO INTERPRET HANDWRITTEN BENGALI LETTERS

Introduction

Unlike English, Bengali letters are not separated symbols, but rather a combination of different types of symbols. Bengali 'letters' are a combination of at least one of the following: consonant, vowel diacritic, and a consonant diacritic. The combination of these can result in approximately 13,000 possible graphemes.

The aim of this project is to classify the dataset of handwritten graphemes. This allows computer vision to read Bengali words, with an alphabet that (to me, a native English-speaker) is much more complicated than the latin alphabet.

Check out the github for this project here.

Data Wrangling

Dataset

Bengali.AI provides monochromatic image data that was collected by asking participants to write individual letters on a piece of paper.

The image data is already split into test and training data provided by kaggle. The image data is provided in .parquet Apache data structure. Each image is 137 x 236 pixels Metadata for both training and testing is also provided in CSVs:

- Source: https://www.kaggle.com/c/bengaliai-cv19
- Size: 200876 rows and images, with 7 columns of metadata for each
- Timeframe: N/A, currently running Kaggle competition
- Metadata CSV Columns:
    - Image ID - string ID of the image
    - Grapheme_root - integer (0, 168)
    - Vowel_diacritic - integer indicating the vowel diacritic (0, 10)
    - Consonant_diacritic - integer (0, 6)
    - Grapheme - printed bengali script letter

Below are some examples of a few images from the .parquet files:

Processing the Data

The Apache Parquet files provided by Bengali.AI are quite large and difficult to load to a Pandas dataframe. Instead, the data was loaded to the GPU using PyTorch.

To load the data into the PyTorch model, I constructed a custom dataset class that transformed the image from a rectangle into a square. A square shape simplifies some of the convolutional neural network settings.

Exploratory Data Analysis

To begin with, I investigated the frequency of each grapheme root, vowel diacritic, and consonant diacritic.

Grapheme Root Frequency

There are 168 possible grapheme roots. Figure 1 shows the distribution of each root.

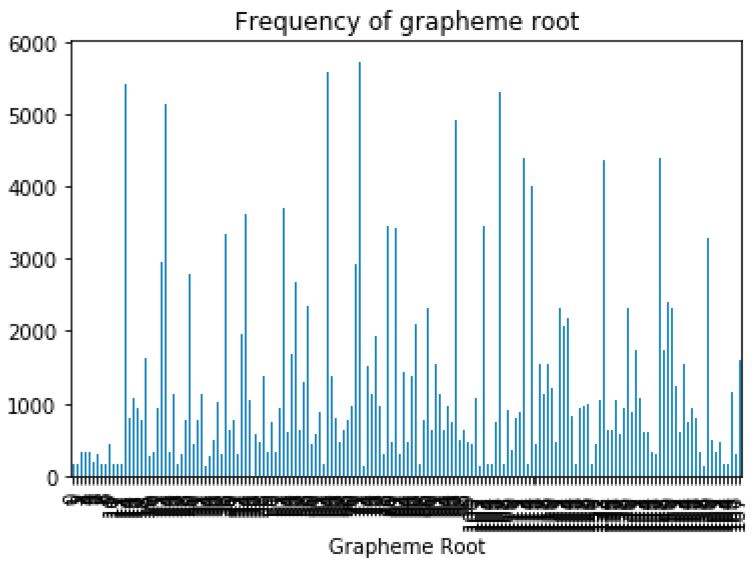


Figure 1--Frequency of grapheme roots

It’s clear that some roots occur more often than others; however, none of the roots are completely unrepresented.

Vowel Diacritic Frequency

Vowel diacritic frequency can be seen in Figure 2. Note that a value of ‘0’ in this case indicates the absence of a vowel diacritic from the grapheme.

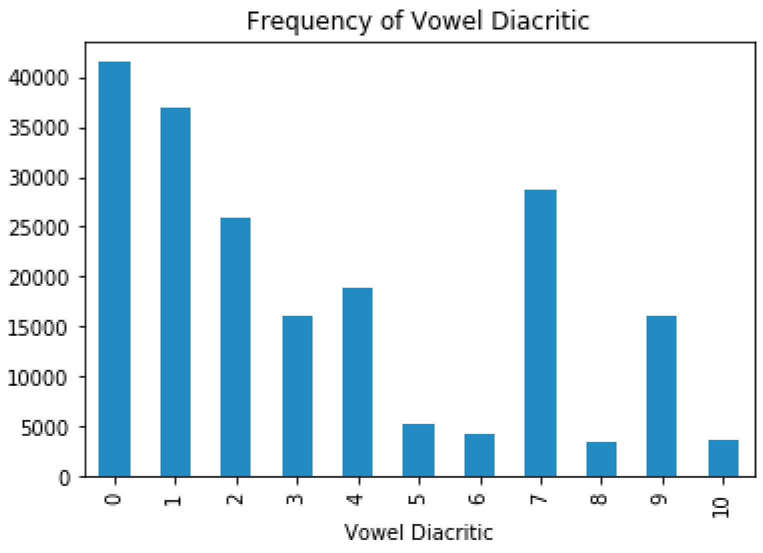


Figure 2--Frequency of vowel diacritics

Consonant Diacritic Frequency

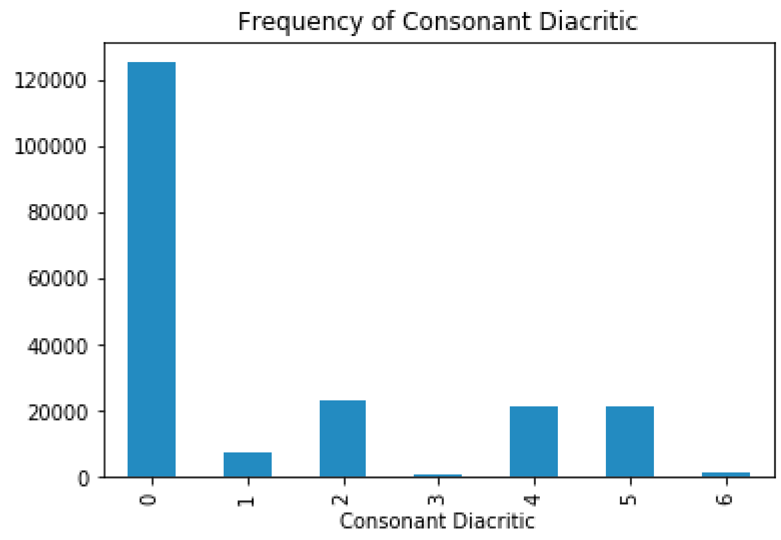


Figure 3--Consonant Diacritic

In Figure 3, most graphemes do not include a consonant diacritic (i.e. item ‘0’).

Deep Learning

Custom Convolutional Neural Network

First, I created a custom Convolutional Neural Network to establish a baseline.

Network Structure

Figure 4 shows the structure of the neural network:

- 2 Convolutional layers (each transformed with a rectified linear unit and max pooling)
- 3 Fully connected linear layers

The figure shows each layer and transformation, finally outputting a value for the most likely category. In this example, the image is Grapheme Root 37 (ন্ধ).

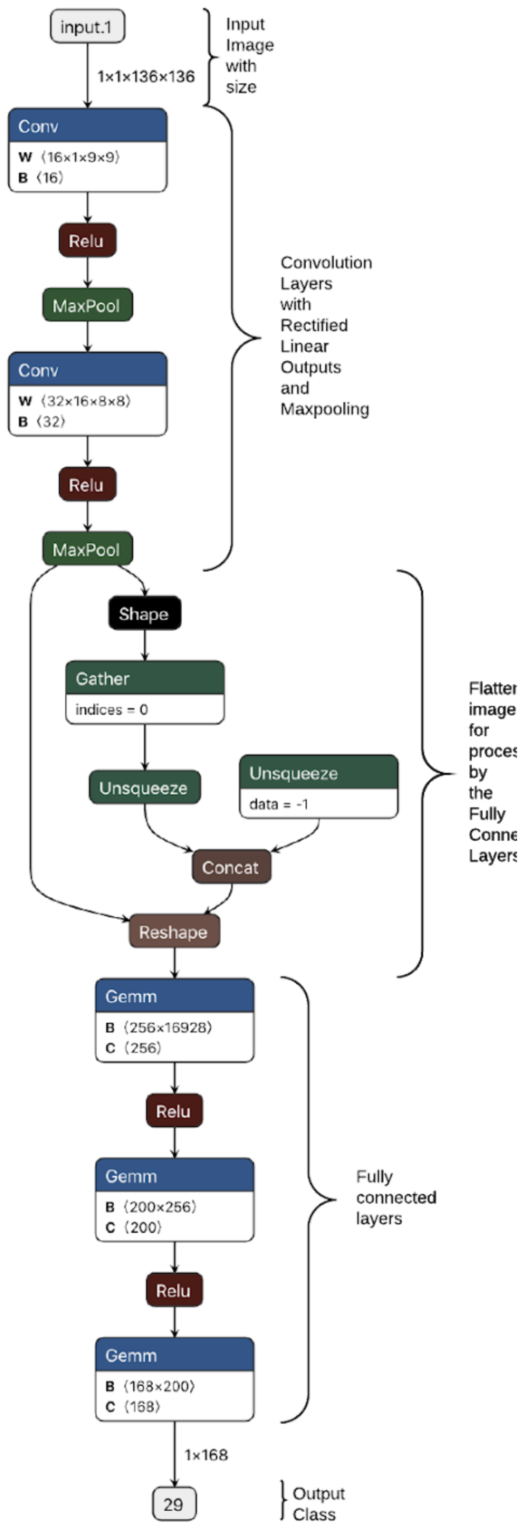


Figure 4--Neural Network Structure

Accuracy and Loss

Over 60 epochs, the network achieved the following validation accuracies:

Grapheme Root	70%
Vowel Diacritic	88%
Consonant Diacritic	91%

Figures 5, 6 and 7 show the loss over the epochs for the Grapheme Root, Vowel Diacritic, and Consonant Diacritic Respectively. For each model, most of the loss reduction occurs in the first ten epochs; however, the next 50 epochs still show small decreases.

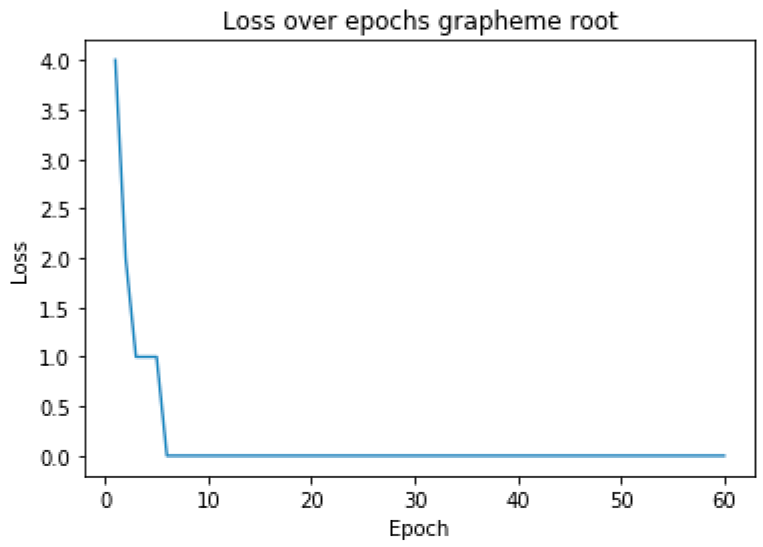


Figure 5--Grapheme root loss

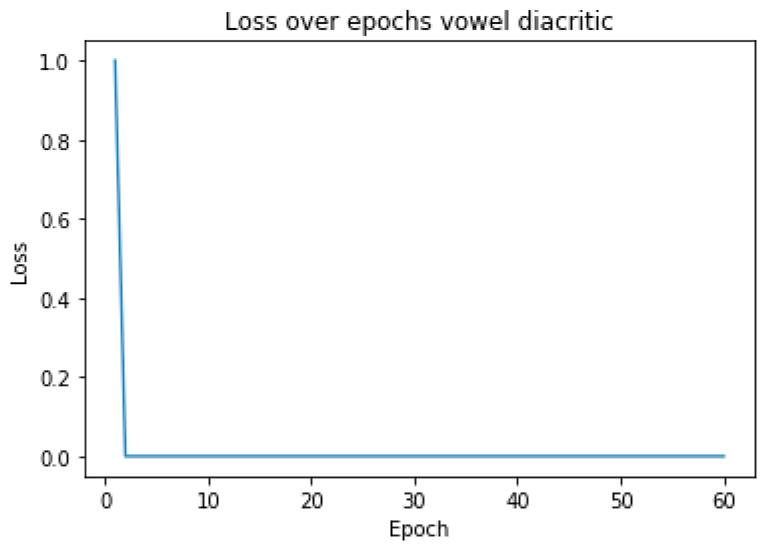


Figure 6--Vowel diacritic loss

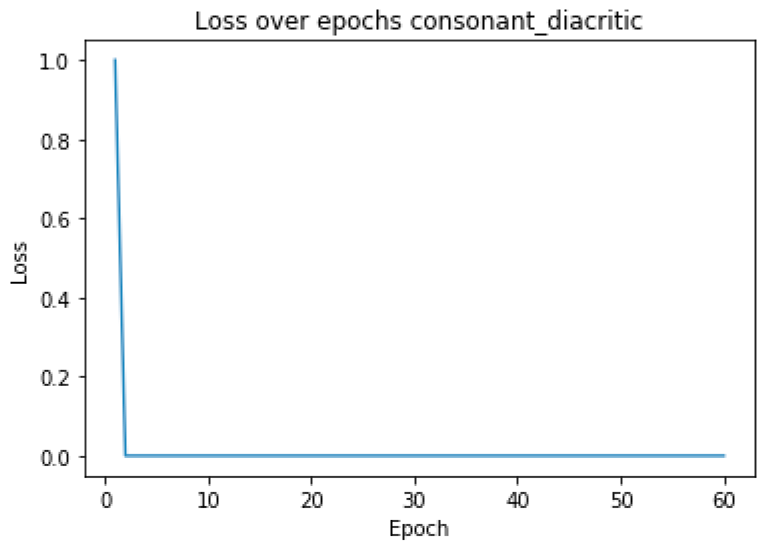


Figure 7--Consonant Diacritic loss

Potential Model Improvements

Future efforts and improvements to this model could include:

- Transfer learning from other models with similar shapes (such as Hindi).
- Further adjustment of Model Parameters
    - Additional Convolutional layers
    - Additional Fully Connected layers
    - Optimized batch size
    - Different Optimize strategies (ie Adam vs SGD)

ResNet

Using a Residual Network to feed the results of layers into later layers as described in this paper I was able to greatly improve the accuracy of the model over just 10 epochs.

The structure of the ResNet as shown in Figure 8 repeats multiple times. In this case, I used "ResNet18" loaded from PyTorch. This pattern repeats 3 times, with 3 additional convolution layers (not shown).

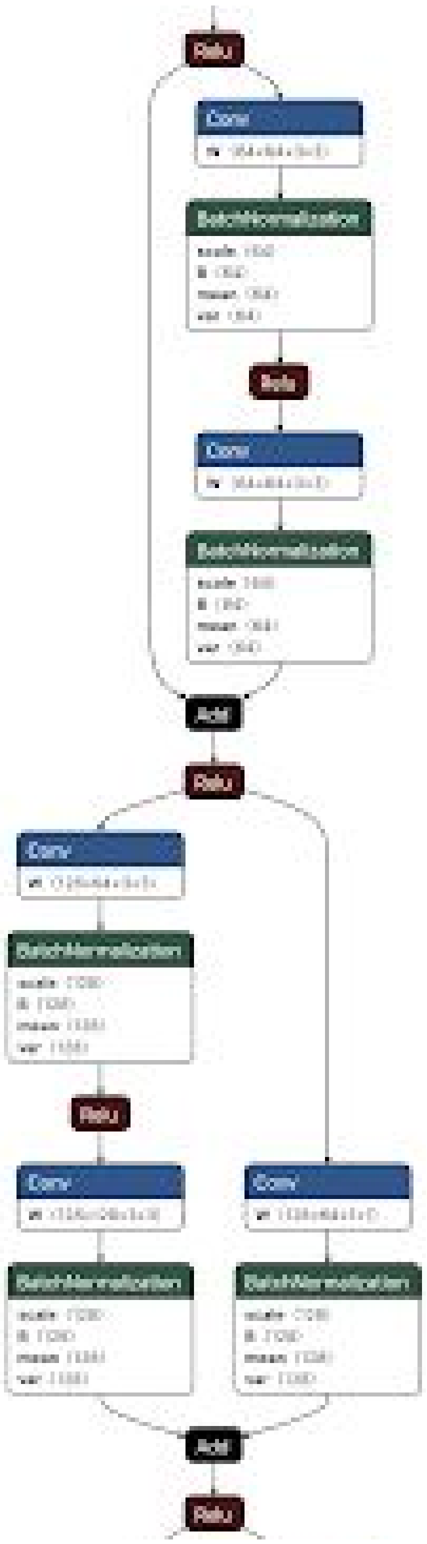


Figure 8 - Repeating Pattern in ResNet

Model Evaluation

Accuracy

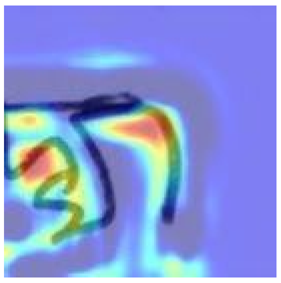
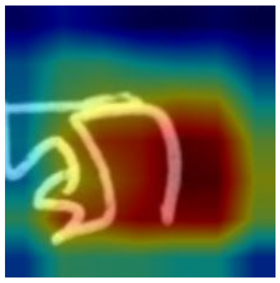
The below table shows the results from both models

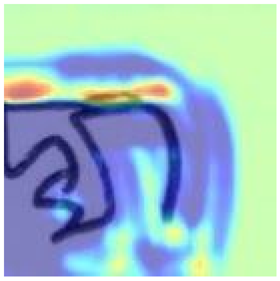
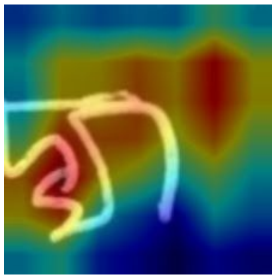
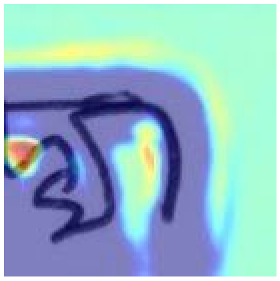
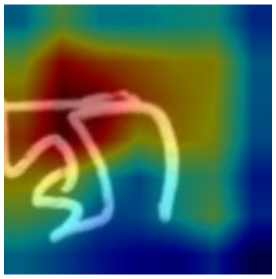
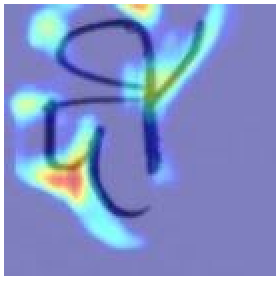
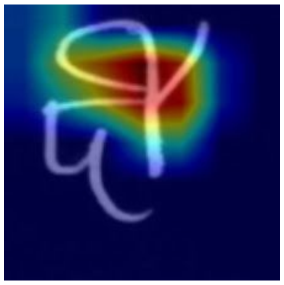
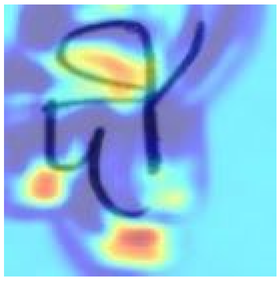
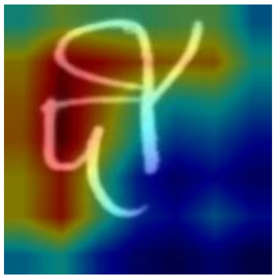
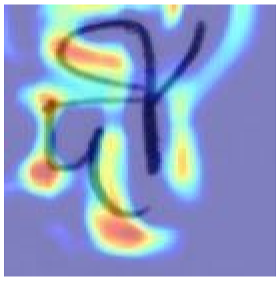
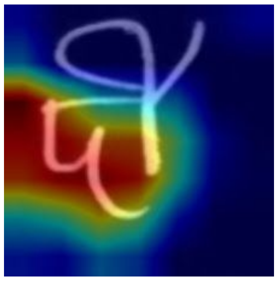
Character type	My CNN Accuracy	ResNet Accuracy
Grapheme Root	70%	92%
Vowel Diacritic	88%	95%
Consonant Diacritic	91%	96%

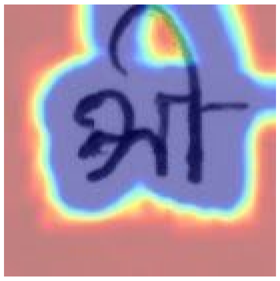
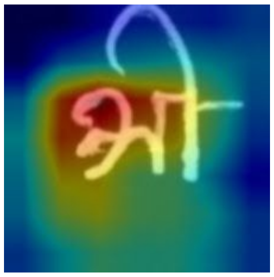
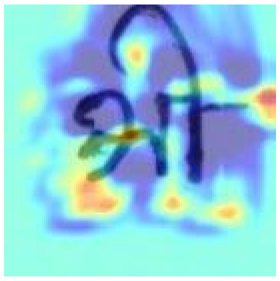
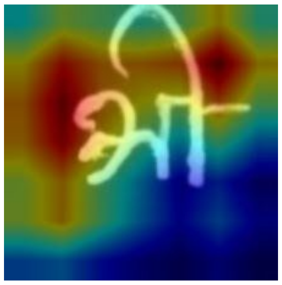
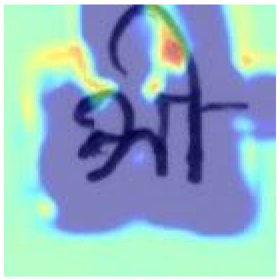
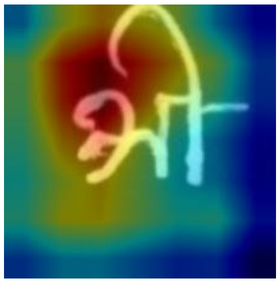
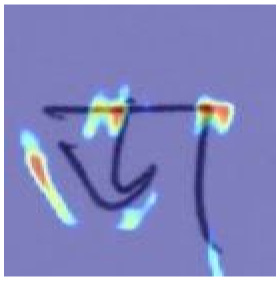
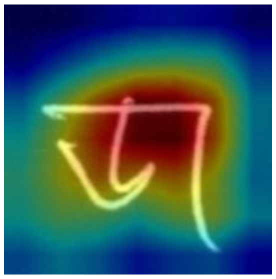
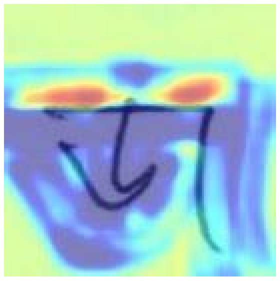
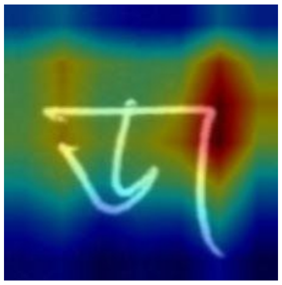
GradCam

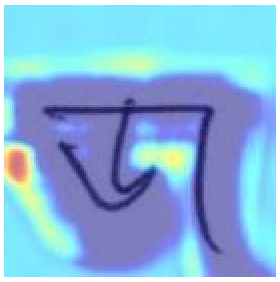
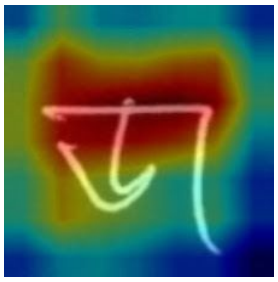
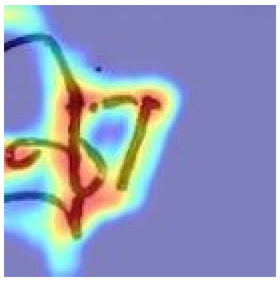
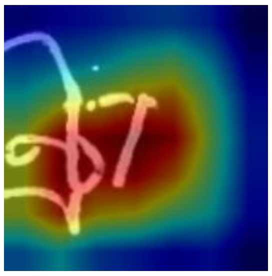
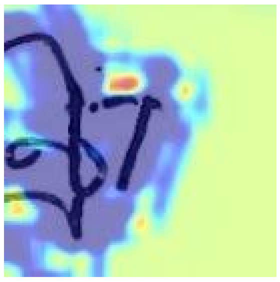
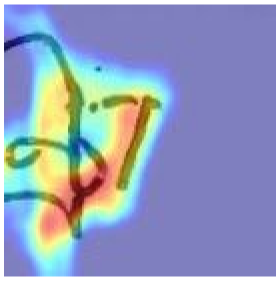
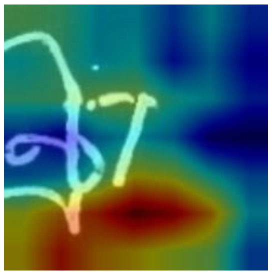
Grad-Cam

I modified Jacob Gildenblat's PyTorch Implementation of Grad-Cam (as described by Selvaraju, et al.) to visualize the 'hotspots' in images processed by my neural network. Below are 5 examples of the results for each character types.

Label: **Grapheme Root - GR** **Vowel Diacritic - VD** **Consonant Diacritic - CD**	**My CNN**	**ResNet18**
GR-73 ক্ষ		

VD-1 ◌া		
CD-0		
GR-72 দ		
VD-3 ◌ী		
CD-2 র্		

GR-95 ন্স		
VD-3 ◌ী		
CD-0		
GR-56 ড		
VD-1 ◌া		

CD-0		
GR-83 ন্ট		
VD-1 ◌া		
CD-4 ◌্য		

The hotspots indicate areas of the image that most affected the classification. For my CNN, these appear somewhat noisy, but they do show the network looking at different parts of the image to make the identification. With ResNet18's deeper network the hotspots become more defined.

Note that even the class 0 consonant diacritics show hotspots, because 0 was coded as a category, so the network appears to be identifying the curves where a consonant diacritic might be.

Conclusion

Unsurprisingly, the deeper ResNet18 model performed overwhelmingly better than my Convolutional Neural Network. Using Grad-Cam helped to visualize the inner workings of both models, and the keys to identify the images.

Future efforts and improvements both my model and the ResNet implementation could include:

- Transfer learning from other models with similar shapes (such as Hindi).
- Further adjustment of Model Parameters
  - Additional Convolutional layers for my CNN
  - Additional Fully Connected layers
  - Optimize test more batch sizes
  - Different optimization strategies (ie Adam vs SGD)
  - Testing to find optimum number of layers and the speed tradeoff. (ResNet18 vs ResNet34 vs. ResNet 108)

Resources

Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. *Automatic differentiation in PyTorch. “NIPS-W”. 2017*

Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun. D*eep Residual Learning for Image Recognition* 2015,