Handwriting Generation and Fraud Detection

December 16, 2020

1 Rishabh Bhonsle (17217) (EECS)

2 Prithvi Poddar (17191) (EECS)

3 Sourav Yadav (17285) (EECS)

Part_1_Report_Handwriting_Generation

December 16, 2020

1 Report for Handwriting Generation

In this part, we have worked on converting a digital string of text into ahandwritten image of the same text. Following are the details of the procedure we followed and the details of the codes.

2 Our approach

To convert the digital text into handwritten text in the image format, we trained individual gans to generate individual letters of the english alphabet. Then we break the text into it's constituent characters and generate the image for each character and then finally append all these images to get a continuous text.

3 Data Preprocessing

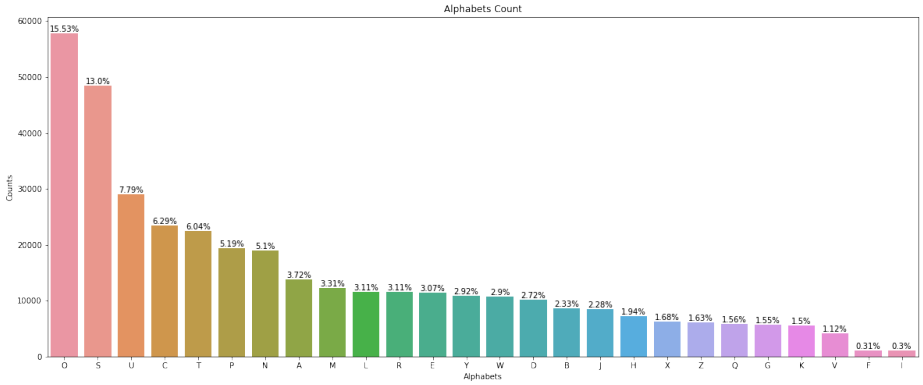
The data pre processing has been done in the **data_preprocessing.ipynb** notebook. The dataset was acquired from Kaggle A-Z handwritten characters

There are **372450 images in total** for all the characters from A to Z. The number of images for each character is as follows:

```
Number of images of letter a is: 13869
Number of images of letter b is: 8668
Number of images of letter c is: 23409
Number of images of letter d is: 10134
Number of images of letter e is: 11440
Number of images of letter f is: 1163
Number of images of letter g is: 5762
Number of images of letter h is: 7218
Number of images of letter i is: 1120
Number of images of letter j is: 8493
Number of images of letter k is: 5603
Number of images of letter l is: 11586
Number of images of letter m is: 12336
Number of images of letter n is: 19010
Number of images of letter o is: 57825
Number of images of letter p is: 19341
Number of images of letter q is: 5812
```

```
Number of images of letter r is: 11566
Number of images of letter s is: 48419
Number of images of letter t is: 22495
Number of images of letter u is: 29008
Number of images of letter v is: 4182
Number of images of letter w is: 10784
Number of images of letter x is: 6272
Number of images of letter y is: 10859
Number of images of letter z is: 6076
```

The distribution of these images can be seen in this graph below:



We separated the images for each letter and saved them as numpy arrays (i.e. **a.npy**, **b.npy** ...). These images were saved as numpy arrays so that it would be later easier for us to load them in the data loader.

4 The model

4.1 The generator:

Our generator takes a 1000 dimensional array as the input and produces a single channel 28x28 image of a character. The generator model has been summarized below:

```
DataParallel(
  (module): Generator(
    (main): Sequential(
      (0): ConvTranspose2d(1000, 512, kernel_size=(7, 7), stride=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): ConvTranspose2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU(inplace=True)
      (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=Fal
      (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): ReLU(inplace=True)
      (9): ConvTranspose2d(128, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (11): ReLU(inplace=True)
      (12): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False
      (13): Tanh()
    )
  )
)
```

4.2 The discriminator:

The discriminator takes 28x28 image and calssifies it as real or fake image

```
DataParallel(
  (module): Discriminator(
    (main): Sequential(
      (0): Conv2d(1, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (1): LeakyReLU(negative_slope=0.1, inplace=True)
      (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (4): LeakyReLU(negative_slope=0.1, inplace=True)
      (5): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (7): LeakyReLU(negative_slope=0.1, inplace=True)
      (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (10): LeakyReLU(negative_slope=0.1, inplace=True)
      (11): Conv2d(512, 1, kernel_size=(7, 7), stride=(1, 1), bias=False)
      (12): Sigmoid()
    )
  )
)
```

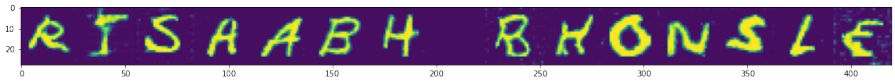
The batch size was 128 with learning rate of 0.0002. Optimizer used was ADAM with beta1 = 0.5. The loss function used was BinaryCrossEntropy. The weights were randomly initialized from a normal distribution with a mean of 0.0 and standard deviation of 0.2
We trained each GAN for 100 epochs

The plots of losses of the GANS while training can be found at the /plots directory. The videos of every GAN's training can be found in the /videos directory

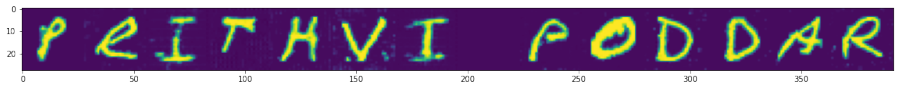
We trained our model on 2 GeForce RTX 2080 Super GPU's with data parallelism. The training code has been provided in the working directory. Kindly go through the README for more information on that.

5 Outputs:

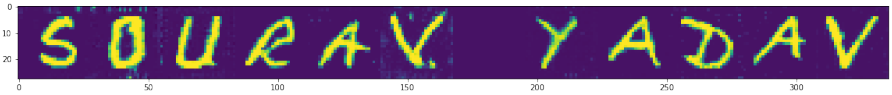
Here are a few outputs that we generated from our trained model:



Rishabh Bhonsle



Prithvi Poddar



Sourav Yadav

6 Current limitations:

Due to the constrained time period, we were only able to traine on capitalised english alphabets. Hence our model can't generate numbers as of now. But we will soon train t on numbers and lower case English alphabets as well.