Limited Hand-drawn Sketch Recognition

Vansh Khandelwal, Utkarsh Singh

Abstract

In this report, we have discussed the complete machine learning pipeline used while implementing the project. We have talked about all the preprocessing steps involved. We have discussed the results obtained by using different models with and without applying dimensionality reduction techniques. We have compared the different models using various cross validation techniques and have reported the findings in this report.

I. Introduction

The basic goal of Sketch Recognition is to allow non-artists to be able to draw visual content and then get their drawn image classified into a class of objects. Our goal in this project is to develop different Machine Learning and Deep Learning Models that help us to efficiently classify Sketches into their respective classes. We have used the Sketchy Database which provides us with a huge amount of supervised data for efficient training of various models.

There are various models like the Google-Net Sketch model that have performed well in this domain. Our task is to try and build the best models we can using the techniques learnt in the course. We also use different image and Data pre-processing techniques to feed good trainable data into our created models.

II. ABOUT THE DATASET

The database used in the project is the Sketchy Database. It is the first large-scale application of Sketch-Photo Pairs. Crowd workers were asked to sketch particular photographic objects sampled from 125 categories and acquire 75,471 sketches of 12,500 objects. We were given SVG images of 640 X 480 pixels dimension corresponding to these sketches. Due to computational constraints, we decided to use the first 30 classes for classification. Owing to this transformation, our reduced data contains 15,524 images corresponding to 30 classes of sketches. The classes predicted in the transformed dataset are:

A. Classes Predicted

1. bicycle, 2. couch, 3. blimp, 4. knife, 5. banana, 6. Pineapple, 7. pretzel, 8. castle, 9. trumpet, 10. flower, 11. church, 12. Hourglass, 13. hat, 14. fan, 15. spoon, 16. umbrella, 17. skyscraper, 18. Bench, 17. saw, 18. car(sedan), 19. shoe, 20. hamburger, 21. hammer, 22. Hot-air balloon, 25. hot-dog, 26. eyeglasses, 27. helicopter, 28. harp, 29. geyser, 30. mushroom

III. PRE-PROCESSING THE DATA

In this part we will discuss the pipeline used by us to generate trainable data from the raw zip file downloaded from the database website. The steps that we took for pre-processing the data were:

1. So, firstly we downloaded the raw zip files available on the dataset website into our google drive. This was followed by extracting the contents of the zip file into google drive. By doing this, we were able to extract the folder containing all the 75,471 sketches that were segregated according to their classes into different folders. The images obtained were in SVG format and the dimension of each image was 640 X 480 pixels.
2. The next task was to process these images and then create .csv files corresponding to trainable data for each class.
3. We used the cairosvg library to convert SVG images to PNG format as it is easier to process images in this format with the opencv library.
4. The next task was to read the PNG image with opencv library in BGRA format which is the default for PNG images.
5. As the SVG images do not contain any background, it was essential to add a white background to these images which was done in the next step.
6. The next task was to convert the images into grayscale as the only colors that the images contained were black and white and there was absolutely no need of storing these images using 4 channels. So, we converted them to grayscale which indeed reduced space consumption by a lot.
7. Now, the current condition of the image was that there was a white background and the sketch was black in color. So, it still was consuming a lot of storage space as most of the image was white (255 pixel value) and the essential part was black (0 pixel value). So, we decided to invert the color scheme so that it reduces the space consumption and helps in better training as well. 8. The final step was to resize the image to a uniform value of 160 X 160 pixels. This was necessary because the previous dimension of the image was still large enough to cause RAM issues while training.

These were the pre-processing steps that we used on the SVG images obtained from extracting the zip files and we

created the .csv files which contained trainable data from each class. We save the .csv files in ‘uint8’ data type to reduce storage space. Now we just had to use this trainable data and create various models and evaluate their performances.

IV. MODELS USED

We have used a total of four models. These include:

A. Random Forest

Random Forest Classifier is one of the models used by us. It is a classifier that uses the ensembling technique of Bagging. We have implemented it using the sklearn library. We have also tuned its hyperparameters which includes the ‘n estimators’ hyperparameter. It is basically the number of weak Decision Tree Classifiers that Random Forest model trains and makes the average of predictions on. We trained the model using different values for ‘n estimators’ and concluded that ‘n estimators’ = 1000 gave best results.

B. Multi-Layer Perceptron

The next model used by us is the Multi-Layer Perceptron Model. It is a model that combines different neurons together to form a larger and in most cases, a better model. We have implemented this model using the Tensorflow library. Again, in this model also, we have tried to tune its hyperparameters to get optimized results. The various hyperparameters that can be tuned include the number of layers in the model, the number of neurons in each layer, the activation function for each layer, the optimizer used and the loss metric used. We trained the model using different combinations of these hyperparameters and tried to optimize them.

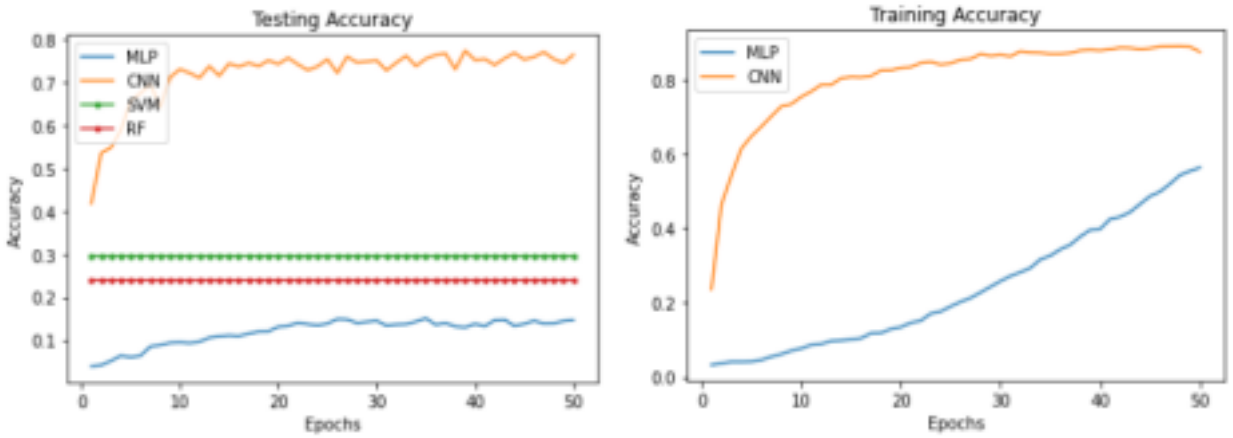
C. Support Vector Machine

The next model is Support Vector Machine. It is an algorithm that tries to find the best hyperplane that divides the classes. Support Vector Machines only work for binary classification problems. For multi-class classification, we have to use approaches like One Vs One Classification (OVO) and One Vs All Classification (OVA). Here also, we have optimized its hyperparameters which include ‘C’, ‘gamma’, ‘kernel’, etc. by conducting various experiments. This model is implemented using the sklearn library. We have used the dimensionality reduction technique of Principal Component Analysis with this model.

D. Convolutional Neural Network

The final model used is the CNN. It is a deep learning algorithm best suited for image classification tasks. As the name suggests, it works by processing the image pixels by multiplying them with suitable matrices and using the information learned from these pixels to classify the image. Similar to the Multi-Layer Perceptron Model, here also we have to optimize the architecture of the model which we have done using various experiments. We have implemented this model using Tensorflow.

Comparing the performance of these Models



V. SELECTING THE BEST MODEL

The final model selected was the CNN model due to its high overall performance in all the evaluation metrics. The CNN model used had the following architecture:

Before training the dataset through CNN model, we first selected 30 classes out of the given 125 classes (due to the limited RAM capacity), we then had to normalize the data which was done by dividing the features by 255 to get binary data. We then trained the data on a variety of CNN models and got an improvement from 40 percent validation accuracy (on a naive CNN model) to finally a 76 percent validation accuracy on the architecture displayed in figure 1. Designing the architecture and key insights:

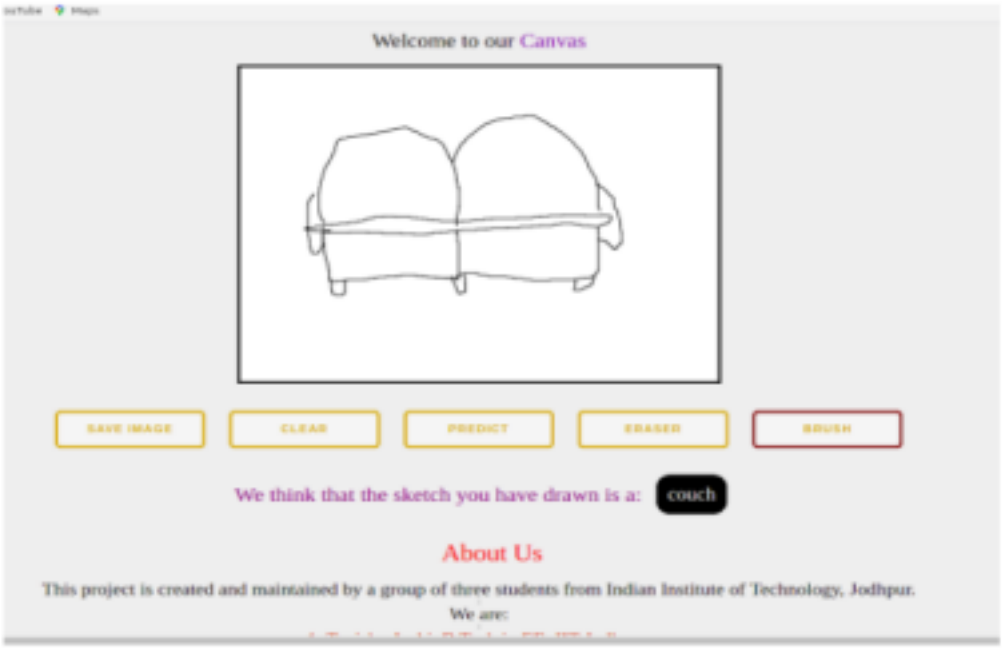
- The architecture was built from scratch and was inspired by popular CNN models for image based classification like VGG, which involve blocks of Conv2d layer followed by a max pooling layer with 2-3 fully connected layers at the end.
- VGG was found to be a very complex model for our dataset thus we used only 2 blocks of CNN layers = 4 Conv2d layers. • We also found out that Dropout layers improve the model to a great extent, this can be explained by the nature of the data since most of the pixels are never used therefore dropouts ensure the equity between the pixels which explains why we have used such high dropouts in our model.

We also tried different dimension reduction techniques like PCA but the performance was reduced greatly due to information loss, this shows that the model is well fit.

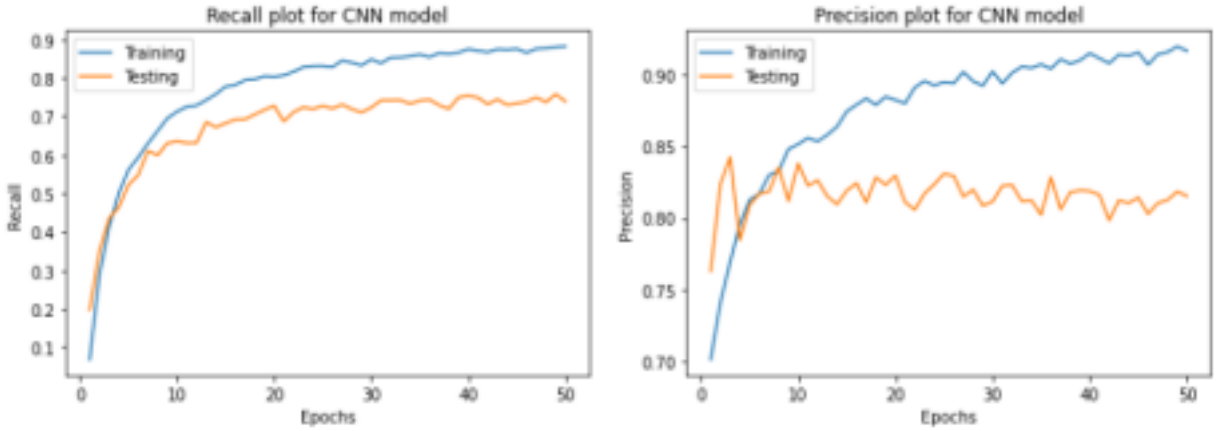
We then deployed the final model (via heroku) on an interactive website made by us to test the

model. Link to the website : https://flamboyant-jepsen-6e91fc.netlify.app/

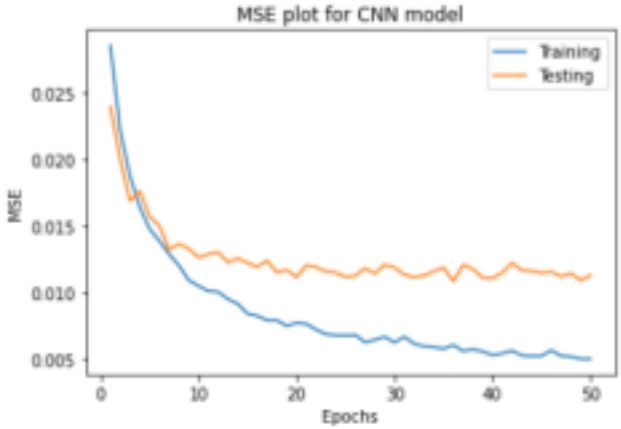
Note: It works best if the sketches are big and centered like the sample one below:

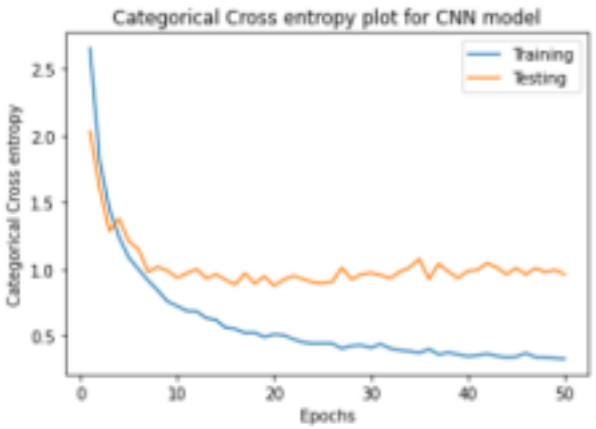


VI. RESULTS AND CONCLUSIONS



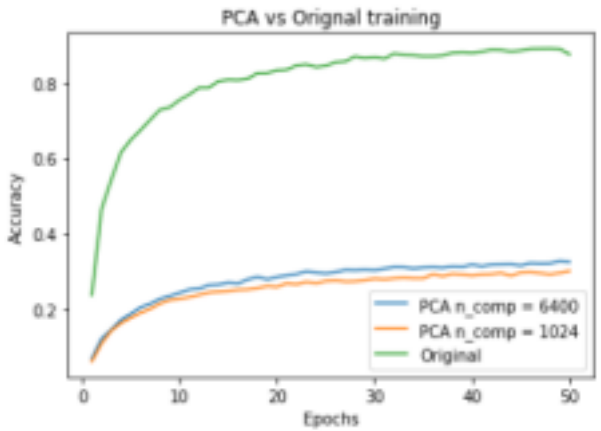
As we can observe from the above plots, the curves of precision and recall increase as the number of epochs increase for training data.
For testing data we observe that recall increases with increase in epochs which indicates a decrease in false negative rate, while the precision first increase then oscillates (effectively constant) this indicates constant false positive rate which is common in classification problems since correctly classifying the datapoint reaches saturation after a certain point.
Below are the plots of loss function and mse with respect to epochs:

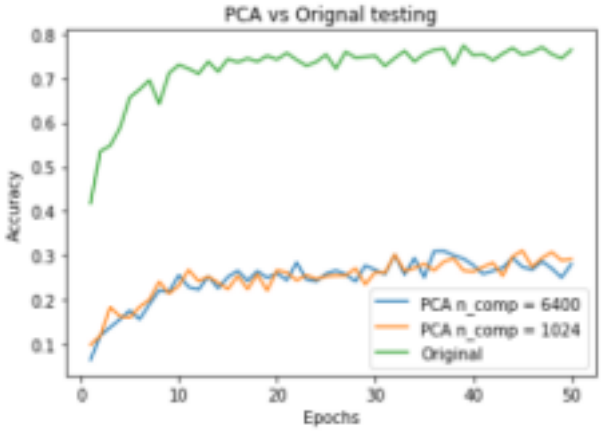




We can see that the loss and error rate decreases exponentially till the 10 th epoch, after which the error rate and loss becomes constant for the testing data while it still decreases for the training data indicating that the model is fit properly and would start to overfit after 50 epochs.

Below are the results of training and testing accuracy using PCA:





We can observe that PCA does not perform well on the given dataset, this can be explained by the fact that there is a lot of information loss while using PCA with components being far less than original dimensions, also PCA is not good at classification tasks.

We also tried to implement LDA but due to RAM limitations we were unable to implement it (LDA caused session crashes even when n components was equal to 1).

We can also conclude that since we do not face any overfitting or underfitting problems as evident from the metrics' graphs, we do not need any dimensionality reduction technique and the model obtained has a good fit.

REFERENCES

[1] https://sketchy.eye.gatech.edu/paper.pdf