**Project Report**  
**Freehand Sketch Recognition using CNN***Ronnie Antwiler ID #1  
Nikita Goyal ID #7  
Lab ID #1*

Demo Link: <https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6ec9c27c-877a-4282-af5b-aa50014cfcc5>

Google Colab: <https://drive.google.com/open?id=1U4OKjtp6VSAfn9xk0zbV6iwCo2Hyrkmh>

Github:

1. **Motivation**

Sketching has been a basic way that humans have communicated for millennia. Sketching is still used today through the digital interaction. We can draw images on iPads and laptops using digital pens or pencils. While we can draw digitally, computers cannot always identify those drawings correctly. There has been research and projects on this identification process. We would like to take those existing techniques and try to improve them. Also, we would like to use methods that have not been tried in some of these projects.

1. **Project Idea**

To identify freehand sketches by training and predicting a CNN model using the TU-Berlin Dataset

Significance:

Sketching is a basic ability of humans. Sketching is a way to make a visual representation of objects that are encountered in our world. Sketching does not rely on words to describe an object. Words can be limited to different languages and therefore not easily understood by all. This means that words are not universal. A visual representation of an object is a form of universal communication. No matter the language of the person doing the sketching, everyone who can see the picture can see what is draw and form an idea of what it is based on their own language.

Correctly interpreting the picture can be more difficult. The existing deep learning architectures are geared more towards real image photos instead of sketches. This has led to trying to find more convenient deep learning frameworks. It depends greatly on how well or accurate the artist can sketch. Among the TU-Berlin Dataset, there are 250 classes of objects or animals like airplane, barn or squirrel. Humans can predict the sketches that make up this dataset with an accuracy of 73% [1]. By creating a CNN (convolutional neural network) that can interpret accurately the different sketches that are being drawn, it would be possible to communicate across languages and at a higher accuracy higher than 73%. The first CNN to accomplish this was Sketch-A-Net with an accuracy of 74.9% [2] and was later modified with an increased accuracy of 77.06% [3]. Additional research has been done to further the classification styles and accuracy.

Project Goals:

Our project had two goals. First goal was the construction of the CNN and training of the model to accurately predict sketches from the TU-Berlin Dataset. The Tu-Berlin Dataset [1] is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/). The goal was to create a CNN that could achieve the highest possible validation accuracy we could achieve. The second goal was to create a web page on which new sketches could be made and be predicted.

1. **System Architecture**

Train Model

Training Data

Processing Data

Save Model

Test Model

Processing Data

Validation Data

****

Client

1. **Technical Stack**

Technologies Used:

1. **Implementation**

TU-Berlin Dataset:  
This dataset consists of 20,000 images and 250 classes. We reduced the classes to 50 due to training speed. Google Colab allows for 12 hours of training period. For 250 classes, 100 epochs took 7 to 8 hours while 50 classes took 2 to 3 hours. We split the data set into 75% training and 25% validation. For 50 classes, this resulted in 3000 training images and 1000 validation images for our project.

Models:

We used 5 pretrained models on the TU-Berlin dataset. VGG16, VGG19, ResNet50, InceptionV3, and Xception were the pretrained models we used. We loaded the base models but then changed the outputs layers for all the models because the dataset had a different number of outputs (classes) than the original models. For VGG16 and VGG19, we set the base model layers to untrainable. This means that the original base model minus the output layer was frozen and the weights would not change for those layers. We did add new outputs layers to the VGG16 and VGG19 and these layers were trainable. The other models could be trained without freezing the weights of the base models.

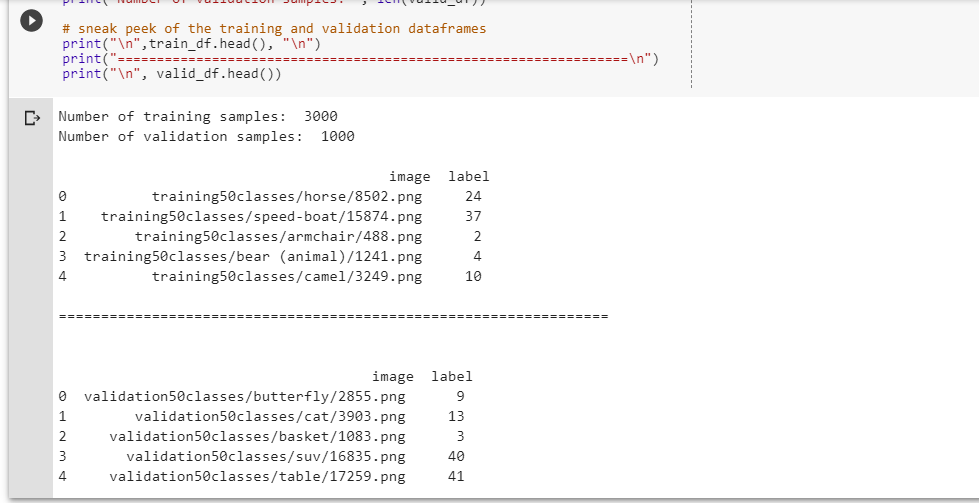
We decided to use VGG16 as our final model because it had the highest accuracy without overfitting. Other pretrained models were overfitting so we did not use them for the final model and the webpage.

1. **Loading the data:** 
   1. We split the data into validation and training set then loaded the data into Google colab.
   2. Then we created a label dictionary by reading the folders in the training directory and storing those in a list. We created a dictionary assigned each folder name a number from 0 to 49.

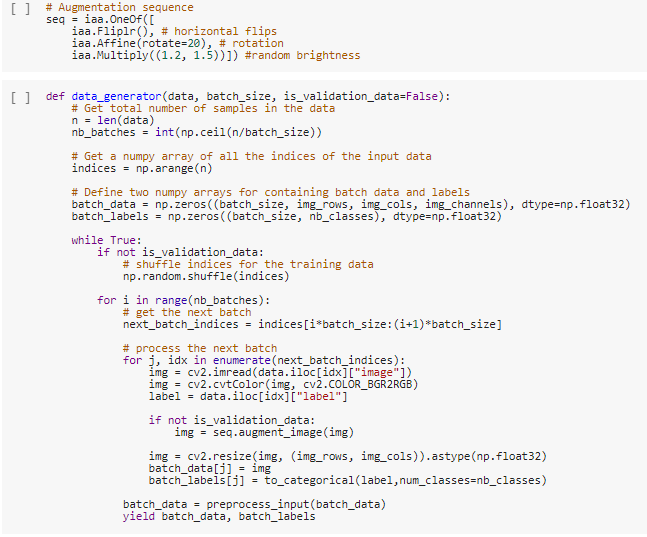


1. **Creating a dataframe:**
   1. We created a validation and a training dataframe in which we stored the image and its associated label.



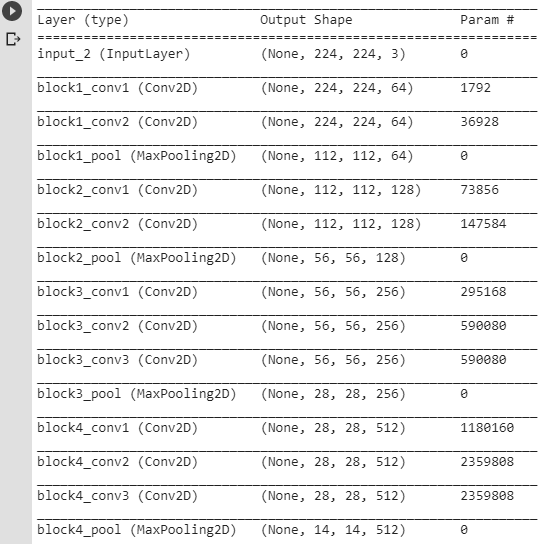


1. **Data Augmentation:**
   1. We did random horizontal flipping, random brightness, and random rotation up to 20 degrees.
2. **Data Generation:**
   1. We then created data generator to use in the model.

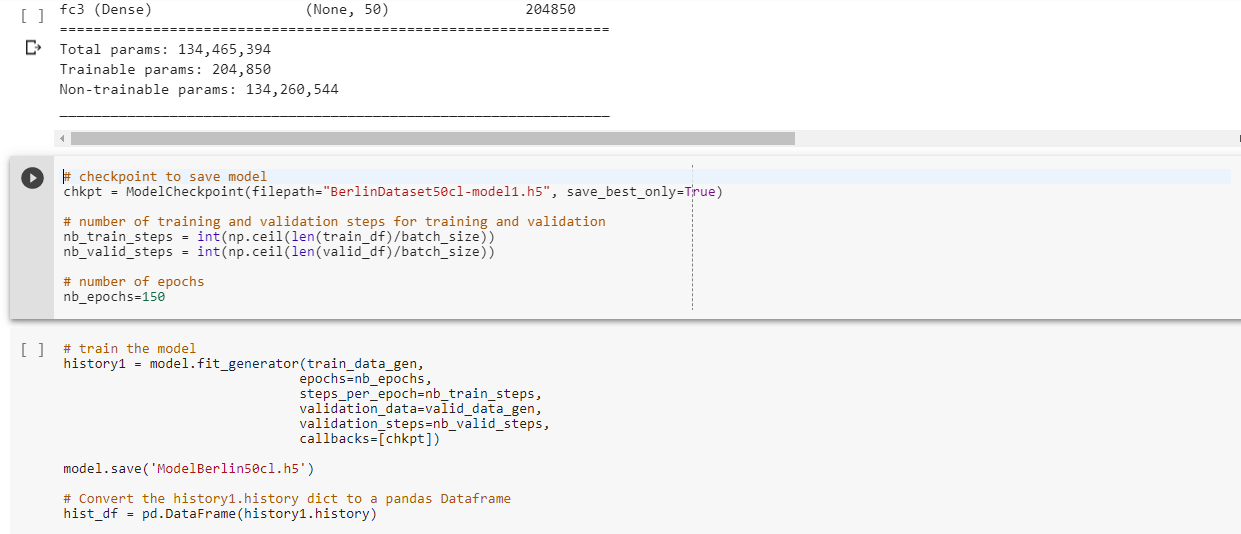


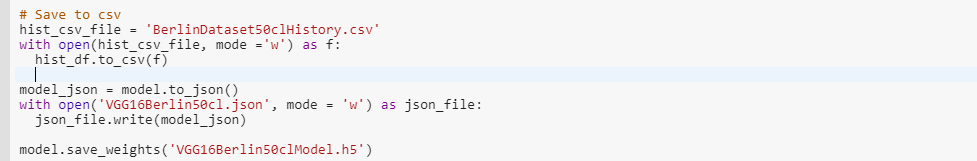
1. **Create model**
   1. We load the base model, VGG16. We remove the output layer then add a new output layer. We combine the original base model and the new output layers into a model. Due to the VGG16 not working with our Dataset, we made the base model layers untrainable. We then compiled the model.

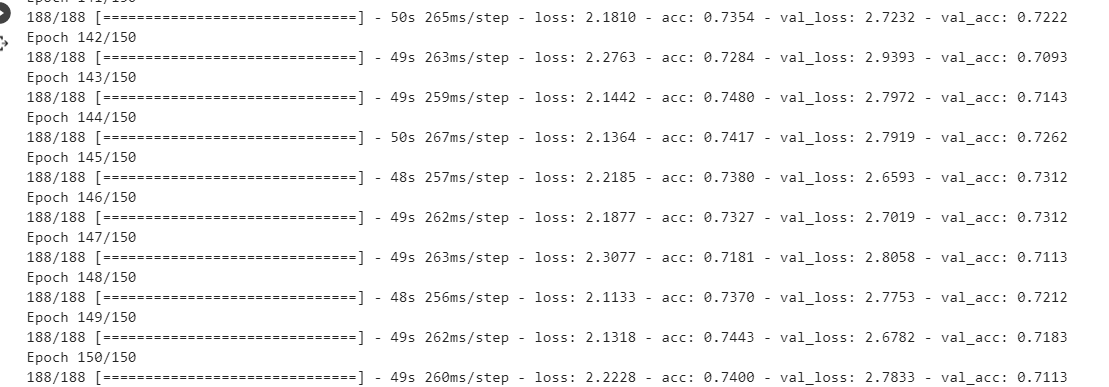




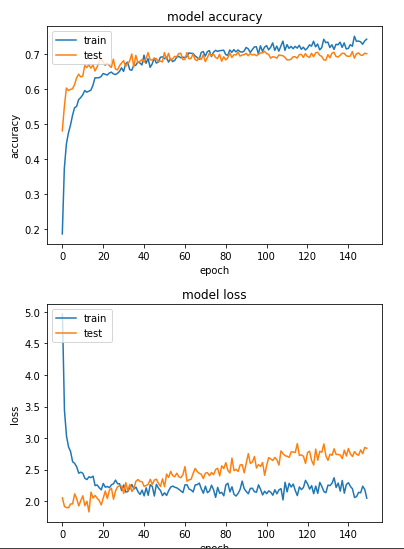
1. **Train and Save the model to h5 file**
   1. We used 150 epochs, 16 batch size and image size of 224 x 224.

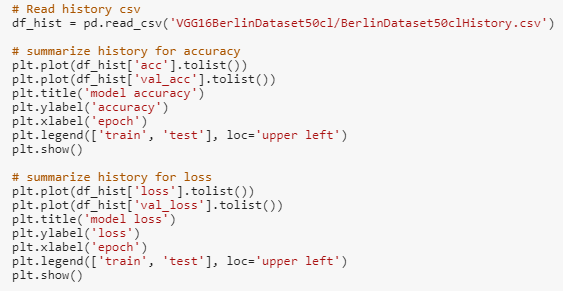


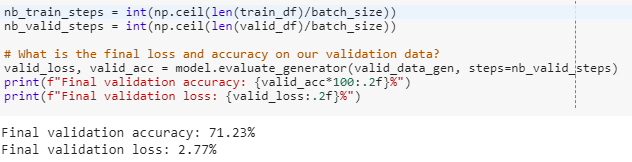




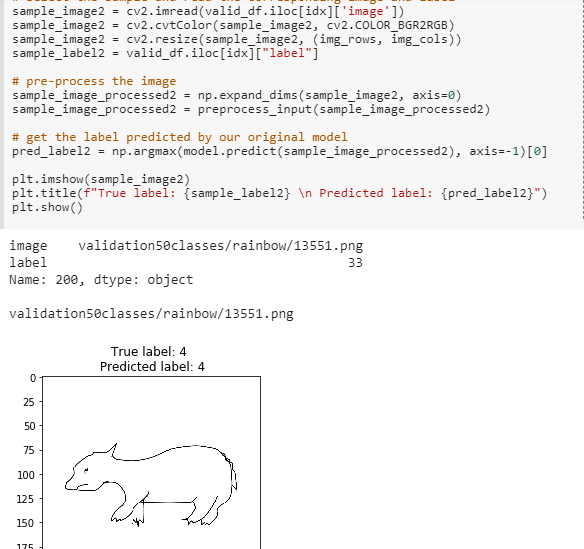
1. **Testing:**
   1. VGG16 validation accuracy: 71.23% and validation loss of 2.77%
   2. Graphs show the accuracies of training and validation and losses of training and validation.





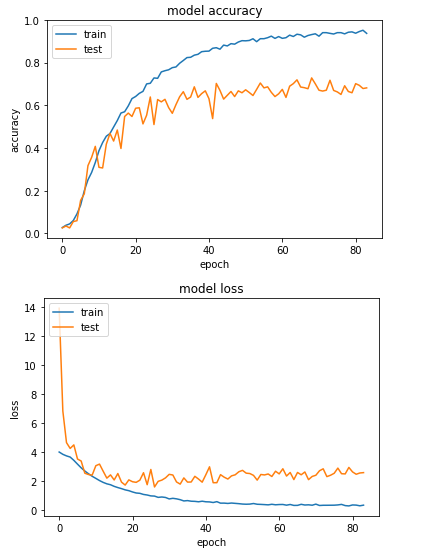


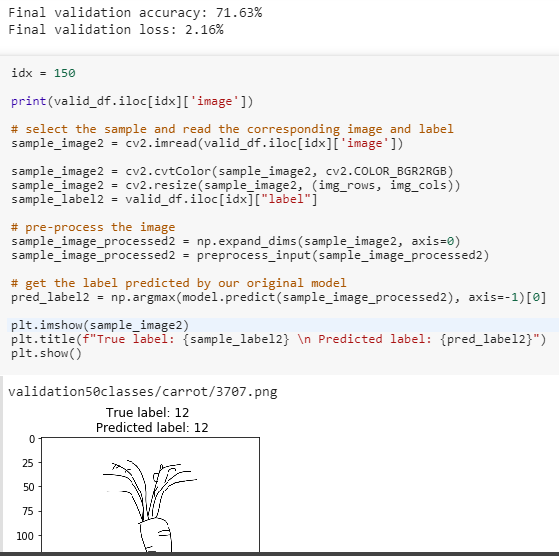
1. **Predict a sketch using the model created.**



**Others Models:  
*Inception:***

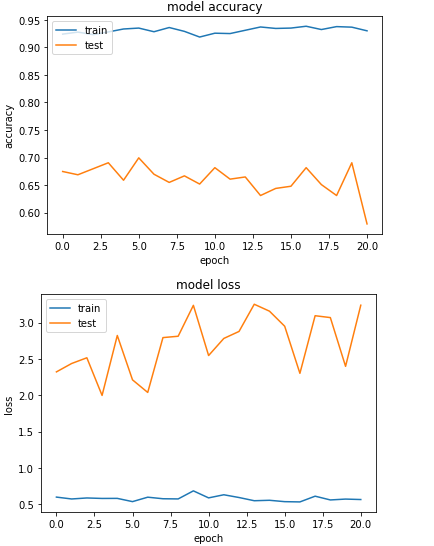
This model has an accuracy of 71.63%. It can be seen to be overfitting on the graph.

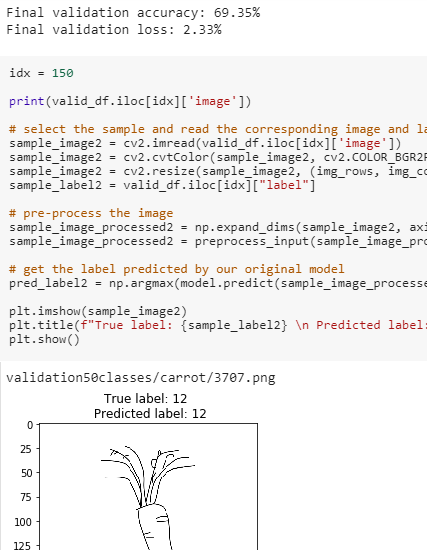




***ResNet50:***

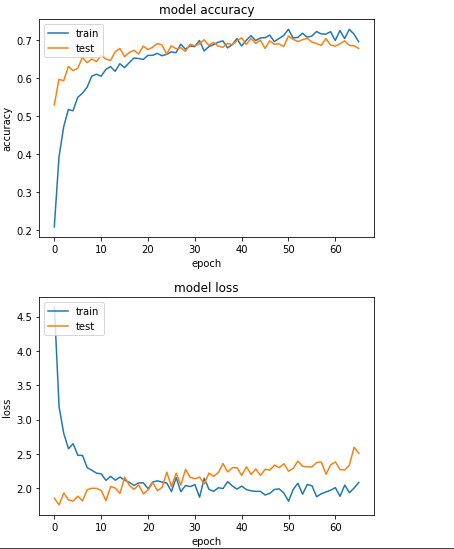
This model had an accuracy of 69.35%. This model is overfitting and not learning on the dataset.

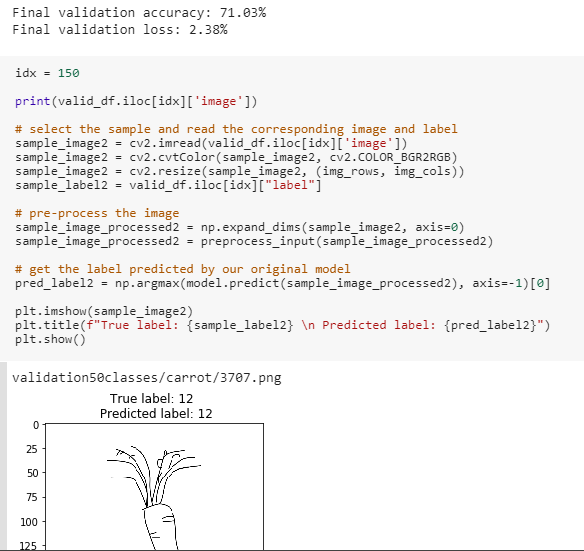




***VGG19:***

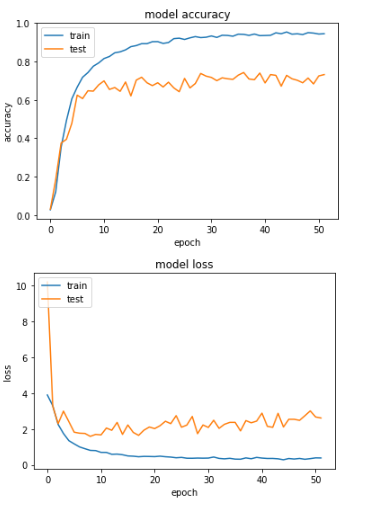
VGG19 had an accuracy of 71.03%. It learns and does not overfit the dataset. However, the accuracy was lower than the VGG16.

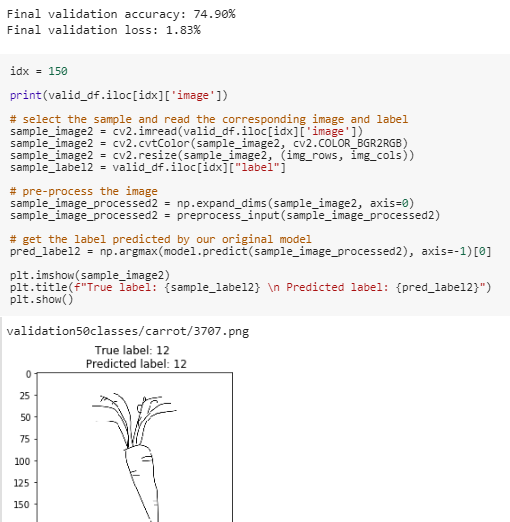




***Xception:***

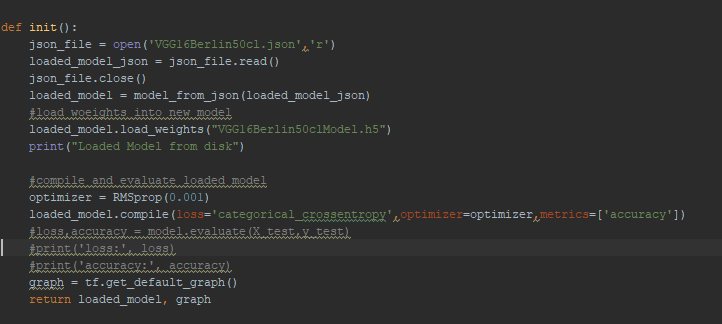
XCeption had an accuracy of 74.90%. But it had overfitting of the model.





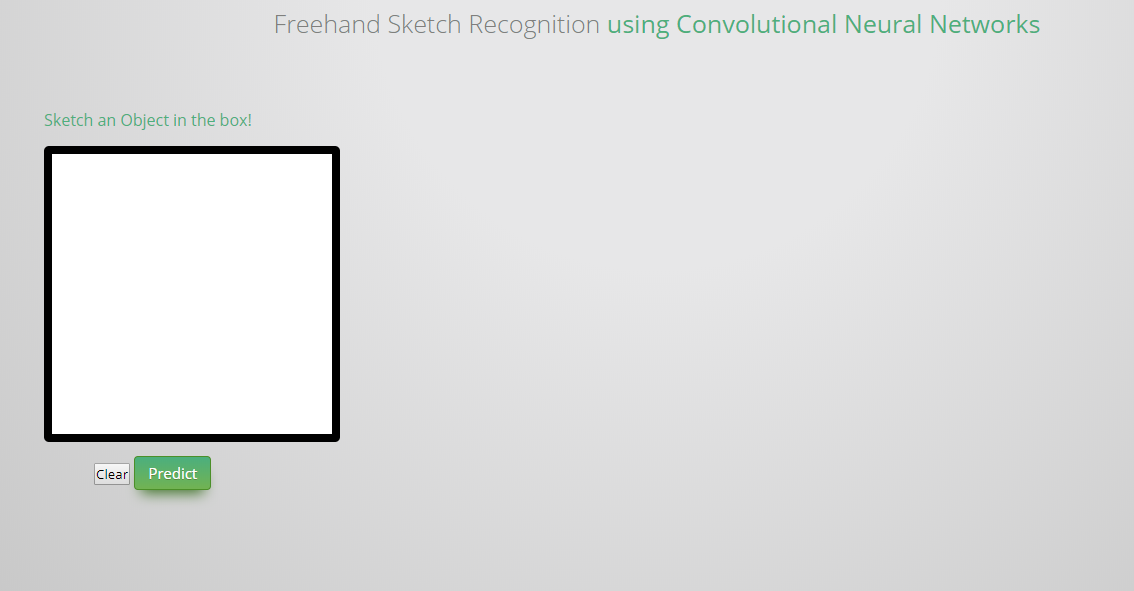
**Deployment of Model  
*Flask***

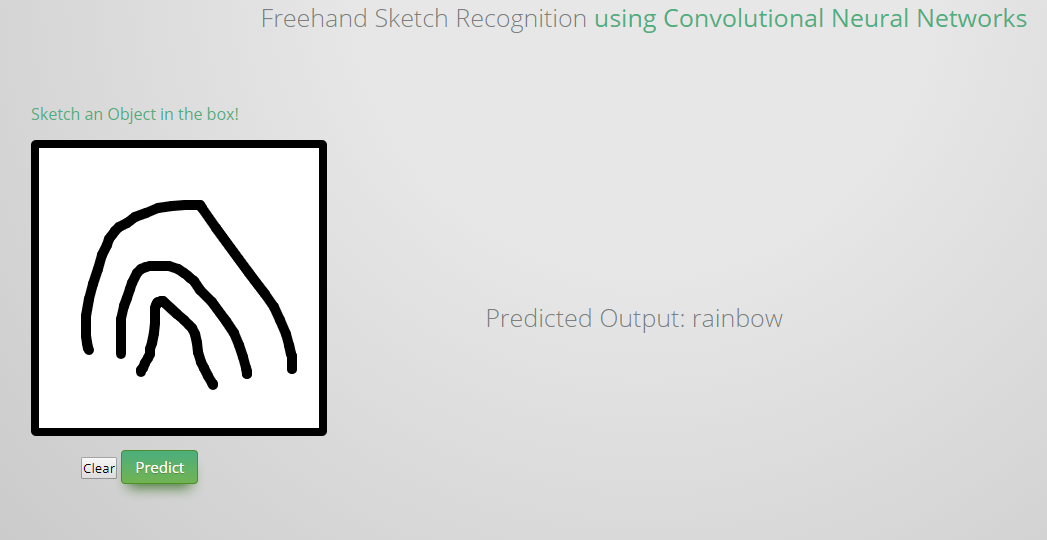
We load the h5 file and the json file of our model in the flask using python. We predict our model in the flask and send back to the client, whenever client requests for it.



***Webpage***

We have created our webpage using HTML, CSS. There are two buttons one is to clear the image and the other one is to predict the drawn image.





**Experimentation**

We tried to create our own model. However, we could not get the model to learn our dataset. We got accuracy around 0.02% and it did not improve even on 150 epochs. We also tried to use other models like Sketch-A-Net. However, they would not learn with this dataset. So, we decided to use the above pretrained models for our project.

1. **Contribution**

We are a two member: Ronnie Antwiler and Nikita Goyal. We did most of the project together. Ronnie did some of the pretrained models alone. Nikita did the flask and part of the web work alone. We both worked on VGG16 together along with our original model and other parts.

1. **References**

[1] Eitz, M. a. Hays, J. a. Alexa and Marc, "How Do Humans Sketch Objects?," ACM Trans. Graph. (Proc. SIGGRAPH), vol. 31, no. 4, pp. 44:1--44:10, 2012.

[2] Yu, Yang, Song, Xiang and Hospedales, "Sketch-a-Net that Beats Humans," Procedings of the British Machine Vision Conference 2015, 2015.

[3] Q. Yu, Y. Yang, F. Liu, Y.-Z. Song, T. Xiang, and T. M. Hospedales. Sketch-a-net: A deep neural network that beats humans. International Journal of Computer Vision, pages 1–15, 2016.

Additional Reference Links

[1] <https://keras.io/applications/>

[2] <https://github.com/suhagba/Final-year-project-deep-learning-models>

[3] <http://cs231n.stanford.edu/reports/2017/pdfs/420.pdf>

[4] <https://www.pytorials.com/deploy-keras-model-to-production-using-flask/>