Classifying Quick draw dataset using Neural Networks

Prem Shanker Mohan, Pooja Chakrabarthy
School of Computer Science, University of Windsor
Windsor, ON, Canada
Mohan31@uwindsor.ca, Chakrabp@uwindsor.ca

Abstract – **Quick Draw dataset is a collection of more than 50 million drawing across 345 different classes, collected from the players of the game, Quick Draw. It is challenging to identify them because it is hand-drawn under 20 seconds. We choose 6 different classes of drawings and classify them using CNN. We also are using very limited training dataset to emulate real world scenario and due to lack of computational resources. To overcome these difficulties, we use transfer learning (VGG 16) to increase the model accuracy. Then we use ensembling or soft voting to increase the model's accuracy.**

Keywords—transfer learning; CNN; Quick Draw Dataset;

I. Introduction

In this project we perform classification of the Quick Draw dataset using Convolutional Neural Networks (CNN's). *Quick, Draw!* is an online game developed by Google (very similar to the game Pictionary) where the players are challenged to draw or doodle a picture of an object or idea and then uses artificial neural networks to guess the drawing. Google's AI learns from each drawing and thus increases its potential to guess the drawings correctly in future. The game Quick, Draw! also limit their player's time to draw the picture to 20 seconds. The mechanism of the gameplay, the player doddles an object (for example it may say "Draw a bowtie in 20 sec"). Then the player has 19 seconds to doodle the object. Based on their doodle the, the AI guesses the drawing. When the drawing is close enough to the object, it will say "I know, it's a bowtie!" and the player is moved on to the next round. There are in total six round and after the game (all 6 rounds) of Quick, Draw! it shows what other people have doodled in the categories the players didn't draw successfully.

Convolutional Neural Networks [1] is a feed forward neural network (Fig 1) that is extensively used to classify image datasets. They are regularized versions (a process where you add information to avoid overfitting in our model) of multi-layer perceptron. The reason they work so well is thanks to their ability to automatically extract meaningful features to make decisions that classify the images with high probability.

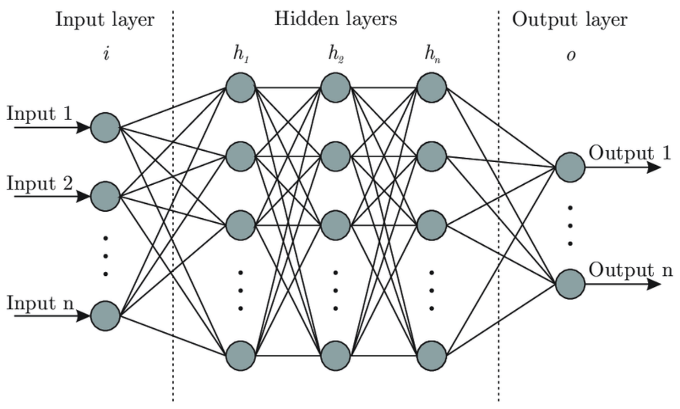


Fig. 1. Neural Network

As the features extracted by the CNN are spatial interaction between the pixels, they perform better compared to the features defined in algorithm based method. Another reason for choosing CNN is they ability to scale well compared to conventional algorithms.

This paper is distributed into the following sections. Section II is the literature review it discusses previous work and background experience. Section III dwells more into the components of the dataset, methodology, details about the layers of the conventional neural network and the hyperparameters. Section IV explains how the project works. In Section V, we discuss the optimizer and loss functions that are used and their influence on the accuracy. In Section VI contains experiments and results. Finally in Section VII we present the conclusion of this project.

II. Realted Works

Over the years, there has been a lot of development in the area of feature detectors and descriptors. Many Algorithms and techniques have been developed for classification. There is an abundance of work in the literature of object detection and classification[2]. Convolutional Neural Networks are mostly used in the ImageNet Challenge with various combinations of datasets of sketches[3][4]. Ye Xu et al.[5], identified topological properties of four different complex networks using convolutional neural network. In this paper, initially, complex networks adjacency matrix is transformed into square shaped JPEG Image named as network adjacent matrix image. The image is given as an input into 10 layers consisting of convolutional layers, pooling layers and softmax dense layer. Then, the features are extracted and further classified. Y Zang et al [6], introduced biomedical relation extraction can automatically extract high-quality biomedical relations from biomedical texts, which is a vital step for the mining of biomedical knowledge hidden in the literature. Recurrent neural networks (RNNs) and convolutional neural networks (CNNs) are two major neural network models for biomedical relation extraction. Neural network-based methods for biomedical relation extraction typically focus on the sentence sequence and employ RNNs or CNNs to learn the latent features from sentence sequences separately. Zeng et al. [7], used PPI network and gene expression profiles to train the proposed model in their study. They developed a DeepEP method by introducing node2vec technique to find the topological representation for features with low dimensional vectors from the PPI networks. Then, Multi-scale convolutional neural network was used to achieve the patterns derived from images of gene expression profiles. Also, since learning of PPI networks is an imbalanced process, sampling method was used to reduce the biasness of the results while selecting essential and non-essentials protein samples in the training process. N Sharma et al.[8], focused on evaluating three networks, AlexNet, GoogleLeNet and ResNet50. They used three layers of CNN and pooling layers. The results showed that trained networks with transfer learning performed well than the previous ones and showed higher rates of accuracy.

III. Components

A. *Dataset*

We use Quick draw dataset which is a collection of about 50 million of drawings across 345 categories contributed by the players who play the game. There are various forms of dataset types you can download like the raw dataset (every stroke and the stroke order is captured), simplified drawing files, binary files, and numpy bitmap files. In this project, we choose 6 categories, namely, baseball, dog, dolphin, golf club, Eiffel tower, and bowtie. The dataset we choose is the numpy bitmap file where the drawing is rendered into a 28x28 grayscale bitmap in numpy (.npy) format. Our training set consists of 4600 data points in each class and in total our dataset has 27600 datapoints to work with. Our test set contains 60,000 points in each class and in total we have 360,000 points.

B. *Convolutional Neural network*

Convolutional Neural Network (CNN) is composed of single or multiple blocks of convolution and sub-sampling layers, one or more fully connected layers and an output layer. The principle of CNN is having locally connected Neural network with a shared kernal and sliding the kernal across the image and then performing maxpooling, dropouts, and analyse the hyper parameters. We added a dense layers after that and use Softmax Activation fuction in the output layer. We use 5 CNN model pretrained using VGG, trained the dataset on the ensembled model. Finally we select the models that have the highest probability. When we added more convolutional layers, we had problems with overfitting. Therefore, we restricted outself to using 5 convolutional layers by max pooling. We also used few nodes in the convultional layer (32, 32, 64, 128, 256) as increasing the number of nodes resulted in overfitting.

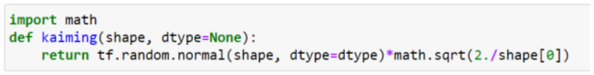
C. *Transfer Learning*

Transfer learning is the process where we use domain knowledge of a problem in another problem[4]. For instance, knowledge gained by recognizing cars could be used for recognizing trucks. In this project we used Neural Network trained on ImageNet weights which we implement through Keras' VGG-16, and froze layers with existing weights and we re-trained the result after feature extraction. In our project, freezing it for 3 blocks gave the best results.

D. *Weight Initialization*

The aim of this technique is to prevent layer activation outputs from exploding or vanishing during the forward pass in our deep neural network. When there are a lot of layers, due the multiplication, some weights can become exponentially large to the point of even the computer not recognizing them. Due to this, the weight may explode. Sometimes if the weight is small, they might become smaller and may become so small that computer recognizes them as zero. To avoid this, we use weight initializations. Since we are using ReLU as our activation function, we use Kaiming function [9]. The way this weight initialization strategy is deployed is as follows (Fig 3).

1. Firstly, a tensor is created with similar dimension as the weight matrix of the initial layer. It is based on randomly number chosen from a standard distribution.
2. Secondly, each randomly chosen number is then multiplied by using $\sqrt{2}/\sqrt{n}$, where n is number of connections from the initial given layer which is received from the previous output layer.
3. Finally, the Bias tensors are initialized to zero.



```
import math
def kaiming(shape, dtype=None):
    return tf.random.normal(shape, dtype=dtype)*math.sqrt(2./shape[0])
```

Fig. 3. Kaiming Layer

E. *Model Configuration*

These were the parameters that was considered in order to achieve the desired results which are as follows:

a) **BatchNormalization**: It is a techniques used to make convolutional neural network more stable by re-centering and rescaling the input layer. In our situation, we noticed adding BatchNormalization after the VGG main layers did not impact the model and in some cases the cross validation dropped a bit. Therefore we decided to drop Batch Norm from our VGG model after some tests.

b) **Image width & Image height**: The shape of the images in our dataset is 28 x 28 pixels (the height and the width of the pixels is 28). We modified the dimensions to 80 x 80 pixels since VGG is trained for higher dimensional data.

c) **Number of channels**: We also converted the input image into RGB (makes 3 channels) so that our VGG model can process the data.

d) **Loss function**: They used in the training process to adjust the weight to increase the accuracy of the model prediction. In this project, we use categorical cross entropy loss function.

e) **Number of classes and epochs**: There are totally 6 classes in our project and we used 50 epochs in our main model. However, after 25-30 epochs the model was overfitting. Therefore for other VGG-16 models we set number of epochs to 25.

f) **Optimizer**: We used Adamax optimizer in this project which we use to update weights

g) **Validation Split**: We split the data into 80/20, meaning we split data in such a way where we use 80 % for training and 20% for testing.

h) **Layers Frozen**: We froze the weights in our VGG model till block 3 (Fig 2). Training the last 2 layers (Block 4 & Block 5) works the best in our case.

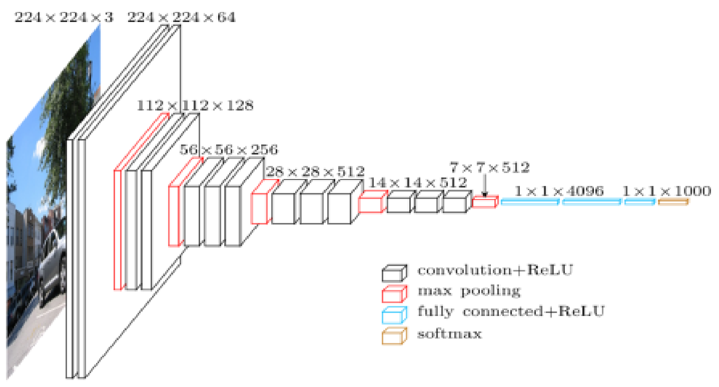


Fig. 2. VGG 16 Architecture

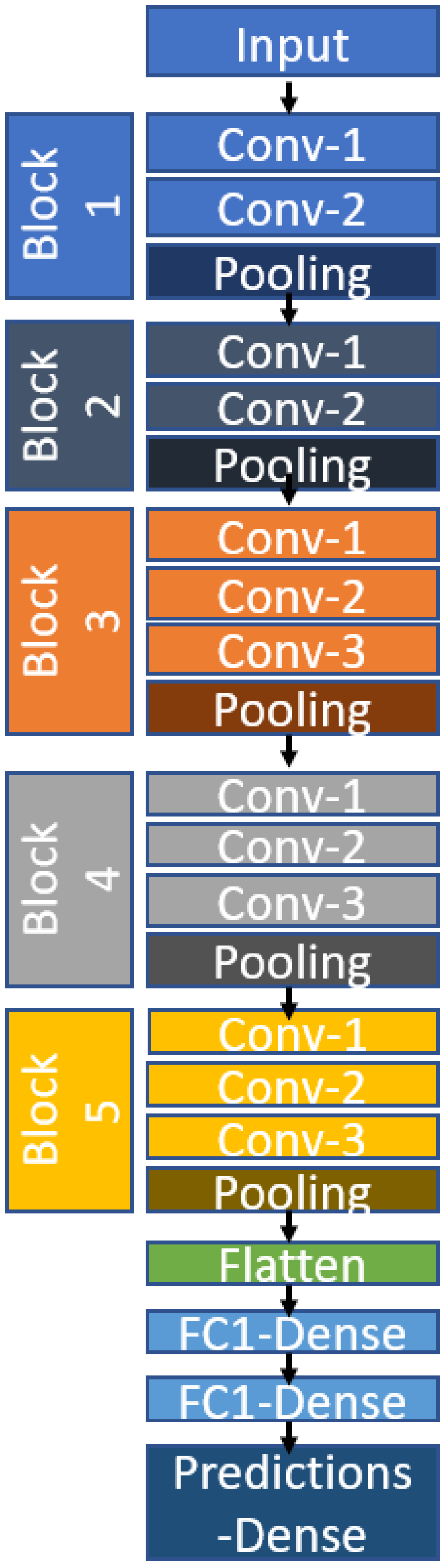


Fig. 2. VGG 16 Architecture

F. *Dropout layer*

The most common problem while training neural network is overfitting. Drop out is a form of regularization that helps us reduce this problem in our neural network. The perks of this type of regularization is that it needs very less computational resourses is executed. In this method, some of the neurons are disabled with a certain probablity. This can be used in a combination with many types of layers like recurrent layer, convolution layer, or LSTM (Long Short Term Memory) layer.

The most important part of dropouts is the hyperparameter that determines the probablity of dropping the outputs generated by the node. This probility is given by us while creating the model. The value ranges from 0 to 1, 1 meaning nearly all the output generated by the previous layer whereas a value close to 0 means that very less output generated from the nodes in the previous layer is retainied.

In our project, we compared different dropout and mapped out the value that gives the maximum accuracy. Out of all the values, 0.3 gave us the highest accuracy (Fig 4).

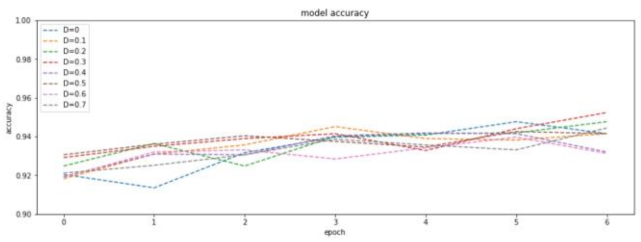


Fig. 4. *Dropout*

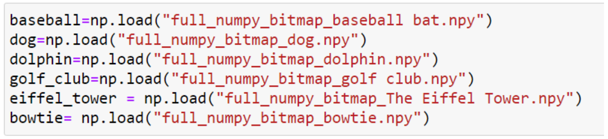
IV. Methodology

In order to express the functionality of our model, we first download the dataset from the github, then we preprocess the dataset, create the models and perform ensembling to increase the accuracy.

A. *Data Loading and manipulation*

The dataset in downloaded from the official github page. The format of the files are Numpy Bitmap Files (.npy).

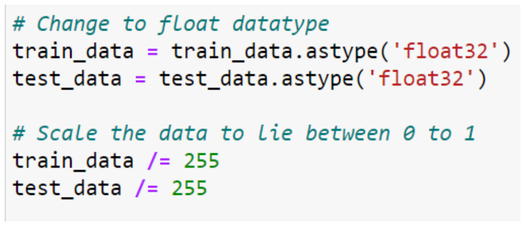
Our dataset has 6 classes, namely, baseball, dog, dolphin, golf club, Eiffel tower, and bowtie. After we download them from github, we extract them using the load function from the numpy library as shown in Fig 5.



```
baseball=np.load("full_numpy_bitmap_baseball bat.npy")
dog=np.load("full_numpy_bitmap_dog.npy")
dolphin=np.load("full_numpy_bitmap_dolphin.npy")
golf_club=np.load("full_numpy_bitmap_golf club.npy")
eiffel_tower = np.load("full_numpy_bitmap_The Eiffel Tower.npy")
bowtie= np.load("full_numpy_bitmap_bowtie.npy")
```

Fig. 5. Loading the dataset

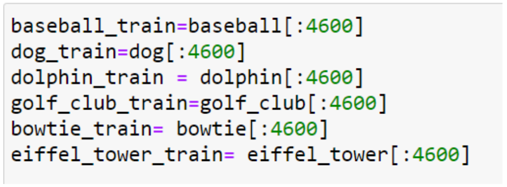
After loading the data using numpy, we split them into training and test set (Fig 7). In training set we take 4600 points in each class and in total our training set contains 27600 points. Now we standardize our data by converting it into float 32 format and normalizing the data. The main reason we perform this step is so that all images are treated as a single unit and can be processed which increases the speed of our process and we normalize the data so that it lies between 0 and 1 as shown in Fig 8.



```
# Change to float datatype
train_data = train_data.astype('float32')
test_data = test_data.astype('float32')

# Scale the data to lie between 0 to 1
train_data /= 255
test_data /= 255
```

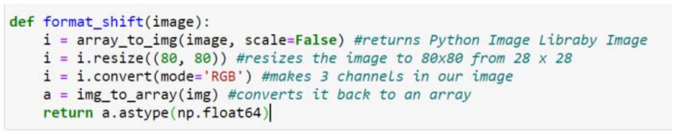
Fig. 7. Standardizing the dataset



```
baseball_train=baseball[:4600]
dog_train=dog[:4600]
dolphin_train = dolphin[:4600]
golf_club_train=golf_club[:4600]
bowtie_train= bowtie[:4600]
eiffel_tower_train= eiffel_tower[:4600]
```

Fig. 6. Splitting the dataset

As mentioned before, we are using VGG-16 as our transfer learning model. However the format of the image is 28 x 28 cannot be used as an input to VGG-16 as it is trained on higher dimensional data. Therefore we preprocess the data by shifting the format of the image to the size 80 x 80 and adding 3 channels. The code is shown in the image below in Fig 7.



```
def format_shift(image):
    i = array_to_img(image, scale=False) #returns Python Image Libraby Image
    i = i.resize((80, 80)) #resizes the image to 80x80 from 28 x 28
    i = i.convert(mode='RGB') #makes 3 channels in our image
    a = img_to_array(img) #converts it back to an array
    return a.astype(np.float64)
```

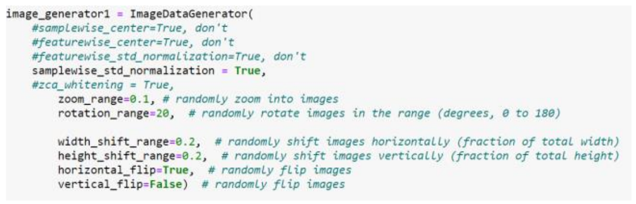
Fig. 7. Loading the dataset

After this, we have used the ImageDataGenerator function from Keras (Fig 8). Using this function you can generate random augmentation to the image as it passes to the model. Image augmentation is a technique where you apply different augmentations to the image which would result in new copies of the image in different transformations. Here are the following transformations that can be applied using this function:

- *Random rotations*: This is the most widely used image augmentation techniques used and allows the model to become uniform to the orientation of the image.
- *Random Shifts:* The object might not be the center of the image all the time. Therefore this feature shifts the pixels of the image either

horizontally or vertically by a constant value. In the function you have height_shift_range for vertical shift and width_shift_range for horizontal shift of the image.

- *Random flips:* Flipping images is also an augmentation technique where you flip image. In our project, this doesn't serve the dataset as flipping the images would change the meaning of the data.
- *Random Zoom:* Here either randomly zoom into the image or randomly zoom out of the image.



```
image_generator1 = ImageDataGenerator(
    #samplewise_center=True, don't
    #featurewise_center=True, don't
    #featurewise_std_normalization=True, don't
    samplewise_std_normalization = True,
    #zca_whitening = True,
        zoom_range=0.1, # randomly zoom into images
        rotation_range=20,  # randomly rotate images in the range (degrees, 0 to 180)

        width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.2,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

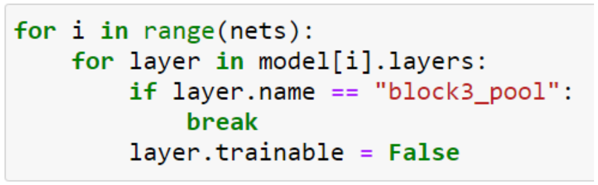
Fig. 8. Image Data Generator

B. *Model Architechture*

In order to construct the model we have to design it in a way that gives us the maximum accuracy. VGG-16 architecture has total 16 layers split into 5 blocks. There architecture contains 3 parts, namely, convolutional layer, pooling layer, and fully connected layer.

This model was introduced in 2014 and it is trained on millions of images and classifies them with high accuracy. Instead of building the VGG model from scratch, we use transfer learning where we preserve knowledge by maintaining weights, and features of the previously trained VGG model that was trained on millions of images.

Here we first block till block 3 and train the last 2 blocks of the neural network and the code for that is shown in the figure below (Fig 9).



```
for i in range(nets):
    for layer in model[i].layers:
        if layer.name == "block3_pool":
            break
        layer.trainable = False
```

Fig. 8. Image Data Generator

After importing the VGG16 model (Fig 9), we flatten the output using Flatten() function from Keras. After we flatten our model, we apply a dropout layer with the probability of 0.3 since that gives the highest accuracy.

The next layer we add is a dense layer with 256 neurons. We have tested all the configurations of this layer, i.e. a dense layer with 0, 32, 64, 128, 256, 512, 1024, 2048 neurons (fig 10).

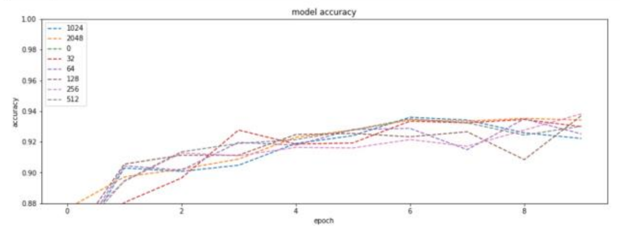
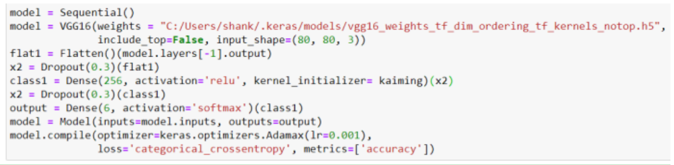


Fig. 10. Dense layer configuration

As you can notice from Fig. 10, out of all the neurons, 256 gives the higest accuray. So we use that configuration in training our model. We initialize the kernal using kaiming and is activated using ReLU activation function.

Then we add and another dropout layer with probablity of 0.3 as that gives the highest accuracy in out model.

Finally, we add our output layer with 6 neurons to predict the classes. We use softmax activation funtion in this layer. The function of softmax activation function is to convert the output into a probability distribution whose sum is 1.

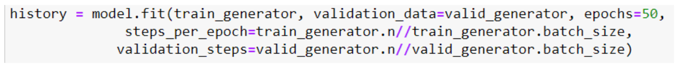


```
model = Sequential()
model = VGG16(weights = "C:/Users/shank/.keras/models/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5",
              include_top=False, input_shape=(80, 80, 3))
flat1 = Flatten()(model.layers[-1].output)
x2 = Dropout(0.3)(flat1)
class1 = Dense(256, activation='relu', kernel_initializer= kaiming)(x2)
x2 = Dropout(0.3)(class1)
output = Dense(6, activation='softmax')(class1)
model = Model(inputs=model.inputs, outputs=output)
model.compile(optimizer=keras.optimizers.Adamax(lr=0.001),
              loss='categorical_crossentropy', metrics=['accuracy'])
```

Fig. 9. Model used in the project

C. *Compiling & Model fitting*

The Skeleton of the model is ready and it need to be complied first by using the function call of model compile. This function includes parameters like optimizers and loss functions. Once the model is complied it is ready to be executed.



```
history = model.fit(train_generator, validation_data=valid_generator, epochs=50,
            steps_per_epoch=train_generator.n//train_generator.batch_size,
            validation_steps=valid_generator.n//valid_generator.batch_size)
```

Fig. 11. Fit the model

D. *Evaluation Metrics and Visualizaiton*

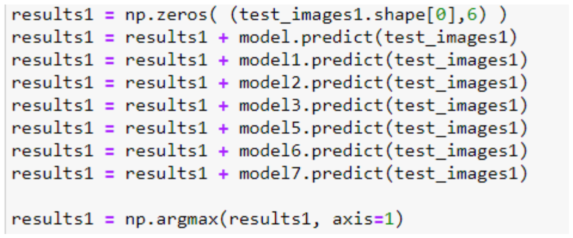
This piece of code allows us to visualize the results that is compiled by the model. At first we ran the model for 200 epochs. However, after 20 epochs, the model started to overfit as shown in Fig 12.

We then proceed to train 7 other VGG models. In model 7, we add another layer to improve the performance in certain classes.

Finally when you ensemble all the 8 models, you get an accuracy of 94.52% as shown in Fig 13.

Ensemble Learning is a machine learning technique where we use multiple algorithm to obtain more accurate prediction. This technique was discovered during the Netflix competition when the two top performers utilized this technique to create the most accurate model and win the competition.

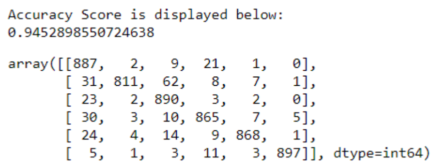
Here, we stack the prediction of all 8 models and you track the prediction from each model. Finally, our prediction will be the highest voted class in the combined prediction. In the code, as shown in Fig 14, we use argmax from the numpy library to accomplish this.



```
results1 = np.zeros( (test_images1.shape[0],6) )
results1 = results1 + model.predict(test_images1)
results1 = results1 + model1.predict(test_images1)
results1 = results1 + model2.predict(test_images1)
results1 = results1 + model3.predict(test_images1)
results1 = results1 + model5.predict(test_images1)
results1 = results1 + model6.predict(test_images1)
results1 = results1 + model7.predict(test_images1)

results1 = np.argmax(results1, axis=1)
```

Fig. 14. Ensemble Learning



```
Accuracy Score is displayed below:
0.9452898550724638

array([[887,   2,   9,  21,   1,   0],
       [ 31, 811,  62,   8,   7,   1],
       [ 23,   2, 890,   3,   2,   0],
       [ 30,   3,  10, 865,   7,   5],
       [ 24,   4,  14,   9, 868,   1],
       [  5,   1,   3,  11,   3, 897]], dtype=int64)
```

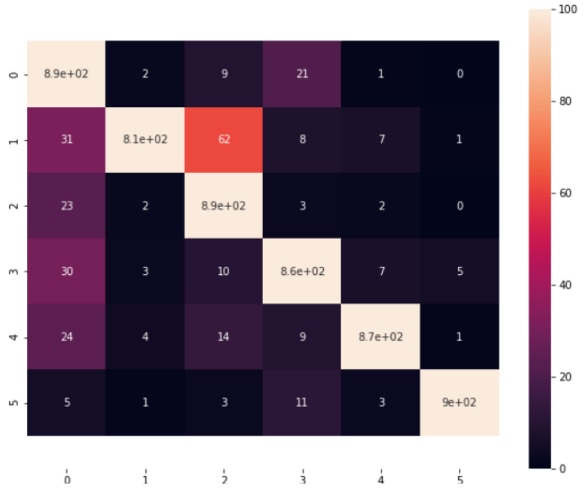


Fig. 13. Confusion Matrix for ensemble model

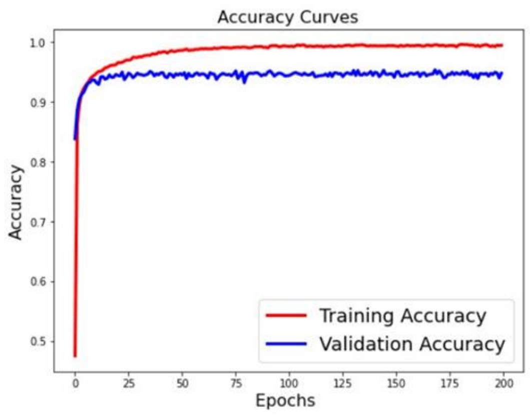


Fig. 12. Accuracy Curve

E. *Optimizers and Loss Functions*

We use different optimizers in this project to perform a comparative study on which are as follows:

- Adamax Optimizer
- Stochastic Gradient Decent Optimizer
- Adam Optimizer

1) *Adam Optimizer:* This is a replacement optimization algorithm for stochastic gradient decent for training models tranied on deep learning. This is relatively easy to configure as the default parameters perform well on most models. The name Adam is derived from adaptive moment estimation.

2) *Stochastic Gradient Optimizer:* This uses a linear model and implements stocastic gradient decent with mini batch leaning techniques. In high dimentional optimization, this reduces the computational burden while achieving faster iterations in trade for lower convergence rate.

3) *Adamax Optimizer:* This a varient of Adam based on the infinity norm. This model is perform better especially in models with embeddings.

In our project, we used implemented all 3 optimizers on our model for 15 epochs and checked which one gave the highest accuracy. As you can see in Fig 15, Adamax optimizer performed better than SDG and Adam optimizer. Therefore, we choose to use Adamax in our project.

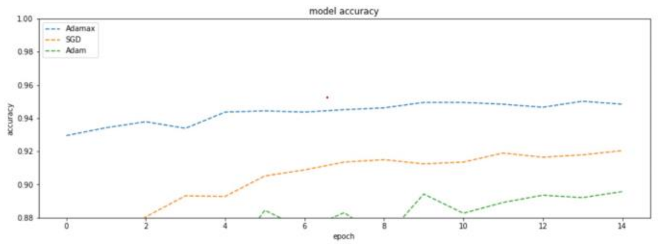


Fig. 15. Loss Curve

V. EXPERIMENTS AND RESULTS

For the experimentation, the first model we tested was a convolutional neural network architecture. To improve the accuracy we used transfer learning where we imported the weights from VGG-16 and trained from block 3. Finally, we create 8 such models (tweaking parameters) and apply ensemble learning.

Let’s explore the accuracy of the CNN architecture. In Fig 16, the red line represents training accuracy and the blue line tracks validation accuracy.

Notice in Fig 16, we observe the accuracy of our CNN model. We observe a sharp increase in the accuracy, and it increased up to 92% after which it plateaued at that level.

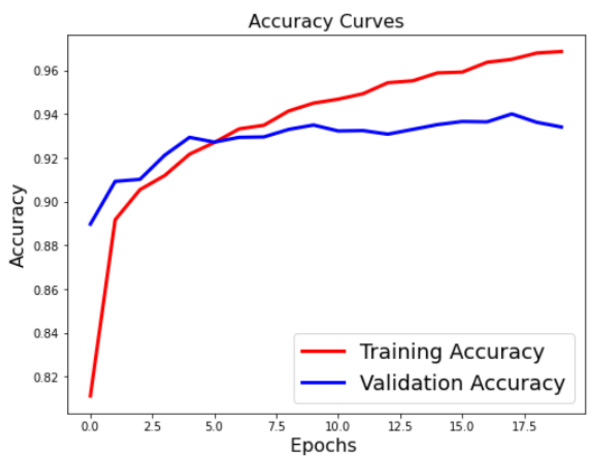
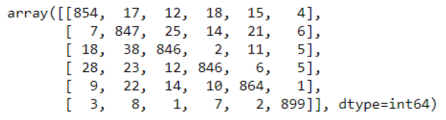


Fig. 16. CNN model

Now let's look at the confusion matrix generated by the CNN model in figure 17. In our project, there are 6 classes namely class 0,1,2,3,4,5,6 represents baseball, dog, dolphin, golf club, bowtie and Eiffel Tower respectively. This matrix shows the classification made on the test data. The diagonal elements are the correctly classified element and others the misclassification.



```
array([[854,  17,  12,  18,  15,   4],
       [  7, 847,  25,  14,  21,   6],
       [ 18,  38, 846,   2,  11,   5],
       [ 28,  23,  12, 846,   6,   5],
       [  9,  22,  14,  10, 864,   1],
       [  3,   8,   1,   7,   2, 899]], dtype=int64)
```

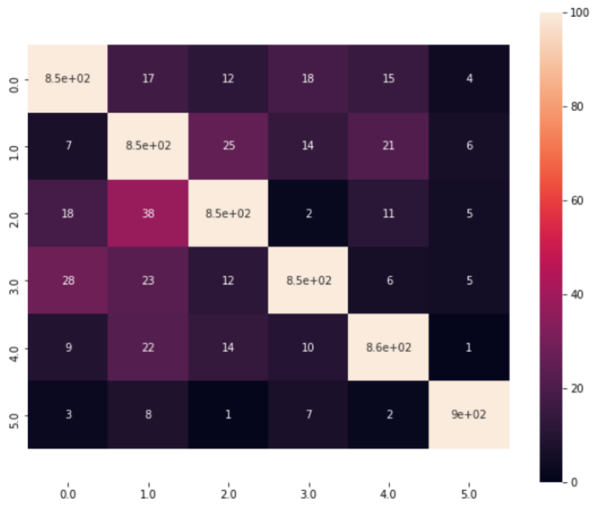
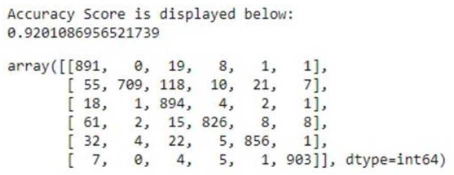


Fig. 17. CNN model

Moving on to our VGG trained model, in Figure 17 we show the confusion matrix of our model. The accuracy is comparable to our CNN model, however, our model has better classification on class1, class 3, class 4, and class 5 compared to CNN.



```
Accuracy Score is displayed below:
0.9201086956521739

array([[891,   0,  19,   8,   1,   1],
       [ 55, 709, 118,  10,  21,   7],
       [ 18,   1, 894,   4,   2,   1],
       [ 61,   2,  15, 826,   8,   8],
       [ 32,   4,  22,   5, 856,   1],
       [  7,   0,   4,   5,   1, 903]], dtype=int64)
```

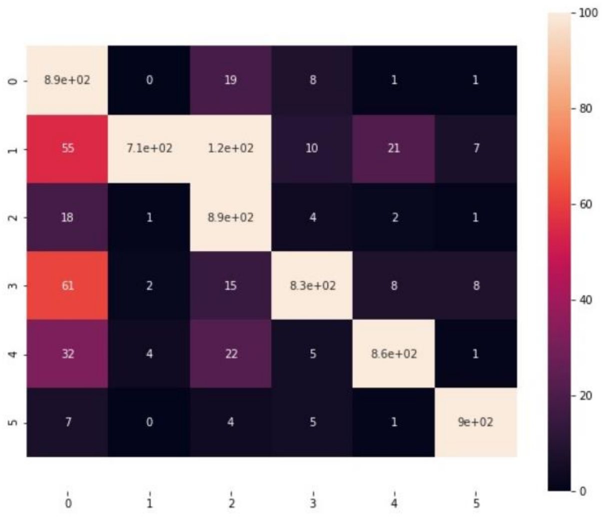


Fig. 17. VGG model

To improve the accuracy on class 1 and class 2, we use ensemble learning (Fig 13), we have achieved an accuracy of 94.52% compared to previous accuracy which is 92%. Our model now has better classification on class 1 and class 3. However, our model had a hard time generalizing between class 2 (dolphin) and class 1(dog). After trying and tuning different parameters, this is the best we result we could achieve.

VI. Conclusion

In this project, we classify hand drawn images from Quick, Draw! dataset. There are 6 classes namely class 0,1,2,3,4,5,6 represents baseball, dog, dolphin, golf club, bowtie and Eiffel Tower respectively. We work with a limited dataset where each class has 4600 datapoints (27600 datapoints in total). We first use CNN and achieve an accuracy of 92%. To increase our accuracy, we first use ImageDataGenerator function from Keras to increase the size of the dataset by augmenting the images. We now use transfer learning where we import VGG-16 model and trained it which improved its performance across class1, class 3, class 4, and class 5 compared to CNN. To improve the accuracy even further, we used ensemble learning and stacked 8 VGG-16 pretrained models which increased the accuracy to 94% compared to previous 92%.

Acknowledgment

We would like to thank Professor Alioune Ngom who guided us though these tough times like this and provide us quality information. This project wouldn't be possible with his sincere love and support.

References

[1] R. Xin, J. Zhang, and Y. Shao, “Complex network classification with convolutional neural network,” *Tsinghua Sci. Technol.*, vol. 25, no. 4, pp. 447–457, 2020.

[2] L. J. Li, H. Su, Y. Lim, and L. Fei-Fei, “Objects as attributes for scene classification,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6553 LNCS, no. PART 1, pp. 57–69, 2012.

[3] M. Eitz, J. Hays, and M. Alexa, “How do humans sketch objects?,” *ACM Trans. Graph.*, vol. 31, no. 4, 2012.

[4] Q. Qi, Q. Huo, J. Wang, H. Sun, Y. Cao, and J. Liao, “Personalized Sketch-Based Image Retrieval by Convolutional Neural Network and Deep Transfer Learning,” *IEEE Access*, vol. 7, pp. 16537–16549, 2019.

[5] Y. Xu, Y. Chi, and Y. Tian, “Deep Convolutional Neural Networks for Feature Extraction of Images Generated from Complex Networks Topologies,” *Wirel. Pers. Commun.*, vol. 103, no. 1, pp. 327–338, 2018.

[6] Y. Zhang *et al.*, “A hybrid model based on neural networks for biomedical relation extraction,” *J. Biomed. Inform.*, vol. 81, pp. 83–92, 2018.

[7] M. Zeng, M. Li, F. X. Wu, Y. Li, and Y. Pan, “DeepEP: A deep learning framework for identifying essential proteins,” *BMC Bioinformatics*, vol. 20, no. Suppl 16, pp. 1–10, 2019.

[8] N. Sharma, V. Jain, and A. Mishra, “An Analysis of Convolutional Neural Networks for Image Classification,” *Procedia Comput. Sci.*, vol. 132, no. Iccids, pp. 377–384, 2018.

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1026–1034, 2015.