Team 7: Dogpedia

CSC 448: Final Project Report

Johan Delao - Github

Mansij Mishra - Github

Henry Suarez - Github

Table of Contents

1 Abstraction

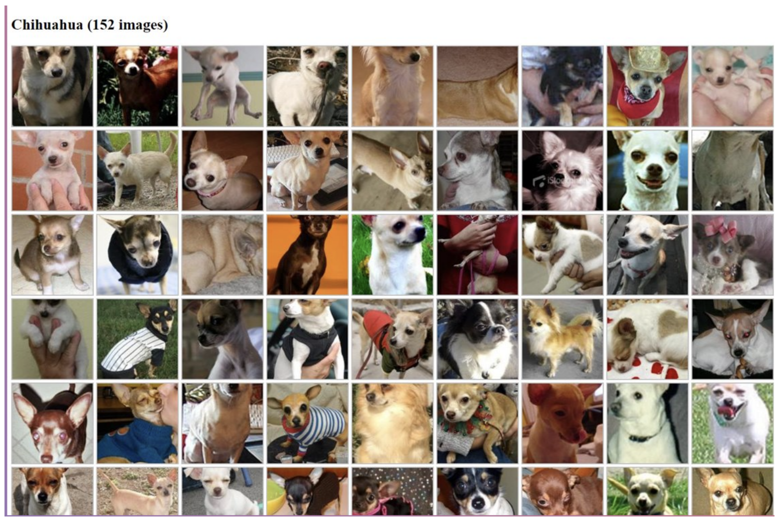
For our project, we wanted to create a solution for people who want to be able to identify dog breeds on the fly and learn more about them. There are over 360 recognized dog breeds in the world. To be able to identify one on your own would be very difficult which is where our solution comes into play. We want users to be able to easily upload an image of a dog and get back facts and descriptions of the dog's breed. We were able to create a web application with React.JS that allows users to upload an image of a dog. After the image is uploaded, our model will identify the dog breed with 93% accuracy and proceed to show the user the breed and general facts/description of the breed.

2 Introduction

The purpose of this application is to utilize neural networks to solve a common problem, identifying the various breeds of dogs. There are many effective approaches, however we decided to further examine an approach using convolutional neural networks. By utilizing techniques such as data augmentation and different types of CNNs, we will be able to determine the best approach for our task. Our main approach is using the Inception-v3 model, a specialized CNN for image recognition which is ultimately the backbone of our model approach. Some other approaches attempted include the VGG16 and EfficientNet models, which ultimately were not successful for reasons we will describe later.

3 Dataset

The dataset utilized for this project was the Stanford Dogs dataset, which is a specially curated selection of 20,580 images of dogs, split evenly between the 120 supported breeds. This dataset is derived from the larger ImageNet dataset, and it features a variety of features including class labels and bounding boxes. Below is an example of the type of data provided for each breed type.



4 Pre-processing

Pre-processing our data is an important step, it ensures that we have a clean and well-structured dataset that can be effectively used for training. When loading our data set of 20,580 images we used various libraries such as cv2, NumPy, and Pickle to reshape, pack into an array, and serialize our images for proper loading for training. Below is a description of each library used and to what extent they were used.

Jupyter Notebook

Jupyter Notebook is the coding environment that we used to write all of our training, modeling, and testing files. It gives us the ability to share live code and visualizations. It also allowed us to code in “blocks” so that we could easily make adjustments as needed.

CV2

Cv2 is an OpenCV library which is a popular library used for image processing and computer vision. In our project, we use Cv2 to load our images and resize them for our training data.

Pickle

Pickle is a Python module that's used for serializing and deserializing objects. In our project, we use Pickle for our preprocessed images to effectively store them and prepare them to be accepted by the model.

NumPy

NumPy is a Python library that's used for data manipulation through arrays and graphs. To pre-process our data we resized the images, reshaped them into NumPy arrays, and then added to a training_data list with its class label.

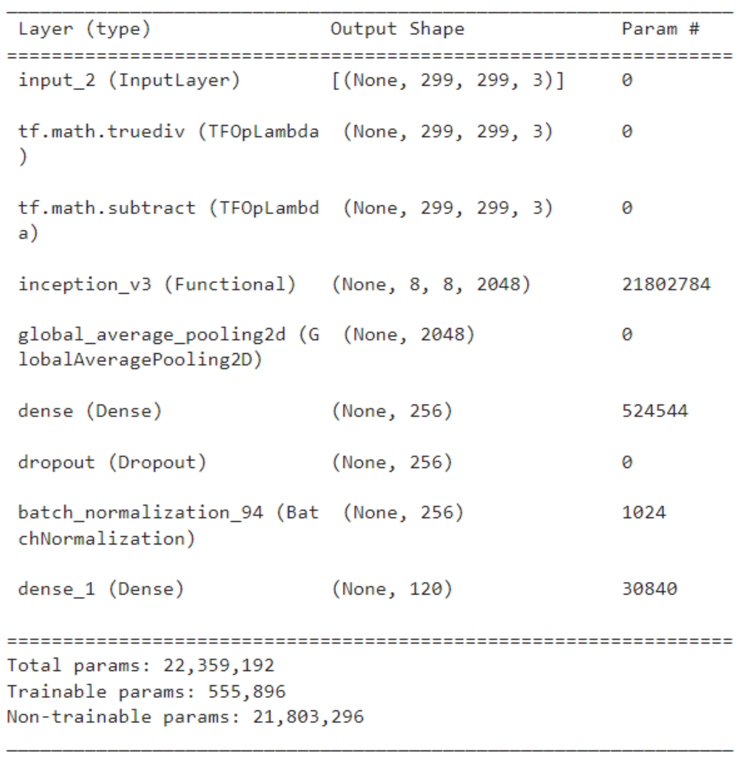
5 Modeling

5.1 Successful Attempts

For our modeling we used InceptionV3 as our base model. To modify its overall architecture we stacked layers on top of the base models. It includes a global average pooling layer, a dense layer with ReLU activation, dropout for regularization, batch normalization for normalization, and a dense output layer with softmax activation for classification. We then used our training data and had a split of 70% for training and 30% for testing. This process spanned 10 epochs each consisting of 32 images.

For our approach, we used Keras, an open-source deep-learning API written in Python. Keras acts as an interface for the TensorFlow library. We utilized Keras to define each layer and connection of our model with the help of high-level functions such as `tf.keras.layers.GlobalAveragePooling2D()` and `tf.keras.layers.Dropout()`. Preprocessing our data and creating data augmentations were other tasks Keras helped us to achieve.

The summary of our model with these added layers is as shown below



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 299, 299, 3)]     0         
                                                                 
 tf.math.truediv (TFOpLambda  (None, 299, 299, 3)      0         
 )                                                               
                                                                 
 tf.math.subtract (TFOpLambd  (None, 299, 299, 3)      0         
 a)                                                              
                                                                 
 inception_v3 (Functional)   (None, 8, 8, 2048)        21802784  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dense (Dense)               (None, 256)               524544    
                                                                 
 dropout (Dropout)           (None, 256)               0         
                                                                 
 batch_normalization_94 (Bat  (None, 256)              1024      
 chNormalization)                                                
                                                                 
 dense_1 (Dense)             (None, 120)               30840     
                                                                 
=================================================================
Total params: 22,359,192
Trainable params: 555,896
Non-trainable params: 21,803,296
_________________________________________________________________
```

Input Layer

- Output Shape: Represents images with a size of 299 x 299 pixels and 3 color channels (RGB)
- Param #: Layer has no trainable parameters

Pre-processing Layers

- Type: TFOpLambda (TensorFlow Operation Lambda)
- Output Shape: (None, 299, 299, 3) - Represents the result of two operations: division and subtraction.
- Param #: Layer has no trainable parameters

InceptionV3 Base Model

- Output Shape: Represents the output of the InceptionV3 base model after feature extraction.
- Param #: 21,802,784 - Total parameters of the InceptionV3 model.

Global Average Pooling Layer

- Output Shape: Represents the spatial average of the 2048 feature maps from the previous layer.
- Param #: Layer has no trainable parameters

Dense Layer (Feature Extraction)

- Output Shape: Represents the result of a dense layer with 256 nodes.

- Param #: 524,544 - Trainable parameters for weights and biases.

Dropout Layer

- Output Shape: Represents the output after applying dropout for regularization.

- Param #: 524,544 - Trainable parameters for weights and biases.

Batch Normalization Layer

- Output Shape: Represents the normalized output after applying batch normalization.

- Param #:1,024 - Trainable parameters for scaling and shifting.

Output Dense Layer

- Output Shape: Represents the final output with 120 nodes for dog breed classification.

- Param #:1,024 - 30,840 - Trainable parameters for weights and biases.

5.2 Unsuccessful Attempts

Data Augmentation Attempt

The first iteration of our model was successful in achieving a low 90% accuracy when it came to testing with 1000 images. Our next inclination was to further improve this model which is where we came into a lot of issues. First, we were limited by the hardware of our laptops. We attempted to augment our data by producing 3 iterations of images off of 1 base image. This would increase our data set size from 20.580 images to nearly 82,320 images. Now this caused issues with our computer memory/RAM and with Numpy's ability to hold these many instances in an array. To try to get around this issue we decided to use batch files to break apart our data and then load each batch into the modeling script. But, once again this proved to be an issue as our program would crash about 2 hours in of running.

VGG16 Attempt

Our original model runs on InceptionV3, so we decided to swap out that model for VGG16 which is another CNN similar to InceptionV3. However, VGG16 has a larger number of parameters which leads to a more computationally intensive model. For reference, Epoch InceptionV3 would run from 5-8 minutes. VGG16 would run for 40-46 minutes per Epoch which for our situation was not viable. This would lead to a total run time of around 8 hours.

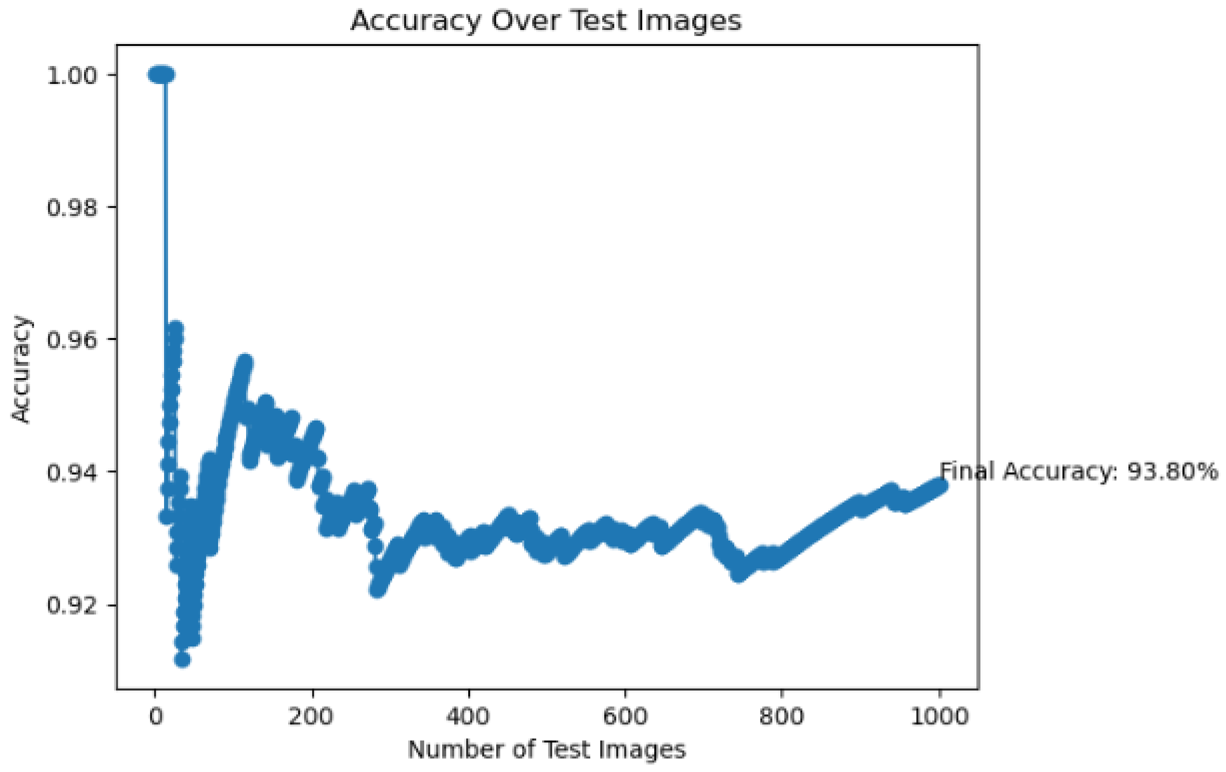
EfficientNet Attempt

Attempting a new model we faced a different type of issue. When attempting to work with EffiicentNet in a similar way as InceptionV3. We could not figure out the correct parameters or configuration to get the model working effectively. The model ended up having a 1% accuracy rate which we decided to scrap at the end to not pursue down the same rabbit hole any longer.

6 Analysis

6.1 Testing

To test our model we created a testing file. This file would run the model against a folder of 1,000 dog images. The images also include labels that identify what their “true labels” are. With this, we were able to test the accuracy of our model and identified that after 1,000 images were checked there was a 93% accuracy rate in identifying the correct breed. The file takes the total number of correct guesses/ by the total number of images, to produce what is below. It also shows how accurate our model is over time. If we had more testing data we could get a better measure of accuracy.



While accuracy isn’t always the best way to test the performance of models we were limited on the feedback we could collect from the model. Since the model doesn’t produce false negatives or positives due to it always going with its best guess. We are only limited to accuracy testing.

7 App Implementation

Dogpedia was built using React.js, specifically using the React framework NextJS. NextJS was chosen for certain features such as its compatibility with Vercel, which we used for deployment, page and layout routing, and its built-in optimization for images and text. We utilized the CSS library Tailwind CSS to speed up development as it allows us to declare CSS styling within the JSX element's class names. Deployment was done with Vercel's Frontend Cloud, as Vercel is the creator of NextJS which allows for a seamless, fast, and reliable deployment.

8 Summary/ Conclusion

Final Implementation

Ultimately, our group effort led to the development of an application that proficiently identifies dog breeds from user-uploaded images with a success rate of 93%. Explore our demo at https://dogpedia-nine.vercel.app/ for a firsthand experience.

What We Learned

Each of us focused on different parts of the project but one thing we all agreed on was the difficulty with training a model. There are a lot of opportunities for error with incorrectly using parameters or an oversight on handling the architecture. This can lead to a few mistakes and valuable time spent on training models that might not work effectively. They do however provide an insight on how to approach future renditions.

Limitations

Something we want to note are the limitations of our project. There were issues with the performance analysis. To analyze the performance of our model we decided to go with accuracy. Which is measured by the number of correct guesses over the total number of guesses. However, we do understand that this is not the best way to measure performance. Several other metrics could be used such as confusion matrix, precision, recall, and F1 score. All these measurements should be used in conjunction to test how well a model performs. However, due to the characteristics of our model, it will not produce a false negative, meaning that if it sees a dog breed it will not identify “no breed”. Without the use of false negatives, our analysis scope is limited to just accuracy.

Next Steps

The process of optimizing our model with specific parameters seemed to work with our current approach of 10 epochs, however, given the ability it would be plausible to increase the number of epochs to 30 before our models progress flatlined. This along with experimenting with the amount of layers we had would provide a healthy increase in our model’s efficacy.

Something that we considered that could be a great addition to our application would be having a way to include user feedback in our model. Our idea was if the model incorrectly identifies a breed the user can label the image correctly and send it back for feedback. The app would then add that image to the model to further improve it.

Rethinking The Model

We could also rethink our model. Currently, our model will always predict a breed based on the image. However, what if the image isn’t a dog? It would then make its best prediction on that image. To improve upon this we could have the model indicate whether it has identified there is a dog. This would open up a whole new avenue of testing as now we could check for False Positives and False Negatives. This would allow us to measure our model with more than just accuracy we could then draw further insights with a confusion matrix, F1 score, Recall, and precision. This would also give us the ability to see what breeds are consistently misidentified and make corrections/additions as needed.

9 Contributions

Johan Delao

- Developed the web application using NextJS, React, and TailwindCSS.
- Connected the APIs (both conventionally named Dog API) Dog API and Dog API as well as the model response API.
- Collaborated on Report

Mansij Mishra

- Pre-processed and trained the Inception-v3 model
- Developed an API for sending model responses from image input
- Hosted the model API.
- Collaborated on Report

Henry Suarez

- Attempted EfficentNet Model Approach
- Attempted VGG16 Model Approach
- Attempted Data Augmentation on Single Model
- Created Accuracy Test Script
- Collaborated on Report

10 References

Keras API Documentation: https://keras.io/api/

Dog Ceo API Documentation: https://dog.ceo/dog-api/documentation/

Dog API Documentation: https://dog.ceo/dog-api/documentation/

Inception V3 Documentation: https://cloud.google.com/tpu/docs/inception-v3-advanced

TensorFlow Documentation: https://www.tensorflow.org/api_docs

NumPy Documentation: https://numpy.org/doc/

Pickle Documentation: https://docs.python.org/3/library/pickle.html

Next.js Documentation: https://nextjs.org/docs

React.js Documentation: https://react.dev/

Juypter Notebook Documentation: https://docs.jupyter.org/en/latest/

Cv2 Documentation: https://docs.opencv.org/4.x/

Dataset: http://vision.stanford.edu/aditya86/ImageNetDogs/

Tailwind CSS Documentation: https://v2.tailwindcss.com/docs

Vercel Documentation: https://vercel.com/docs