Machine Learning Engineer Nanodegree

Capstone Project

Arnesh Sahay
July 2020

I. Definition

Project Overview

Over the course of the last 30,000 years, humans have domesticated dogs and turned them into truly wonderful companions. Dogs are very diverse animals, showing considerable variation between different breeds. On an evening walk through a given neighborhood, one may encounter several dogs that bear no semblance to each other. Other breeds of dogs are so similar that it is difficult for the untrained eye to tell them apart. Included below are a few different images, see if it would be possible for the average person to determine the breed of these dogs.

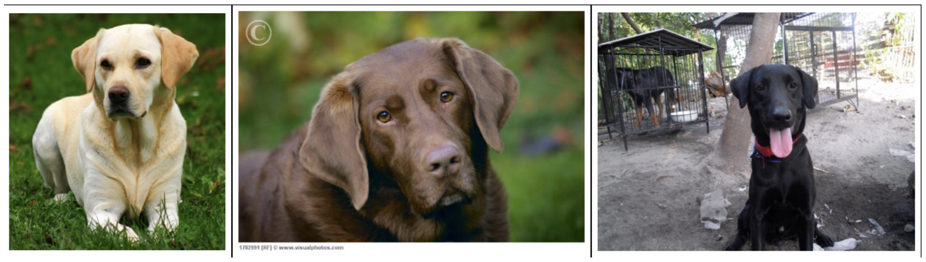

Figure 1: Labrador Retrievers are recognized as having three possible coat colors: yellow, chocolate and black

> LTHQ et al. Silver Labrador Retriever Facts And Controversy. In *Labrador Training HQ*, 2020.

It may take many weeks or months for a person to learn enough about the physical attributes and unique features of different dog breeds to effectively identify them with a high degree of confidence. It would be interesting to see if a machine learning model can be trained to accomplish the same task in a matter of a few hours. The goal of this project is to use a Convolutional Neural Network (CNN) to train a dog breed classifier across 133 breeds of dogs, using the `dogImages` dataset, which consists of 8,351 total images split into 133 different categories by dog breed.

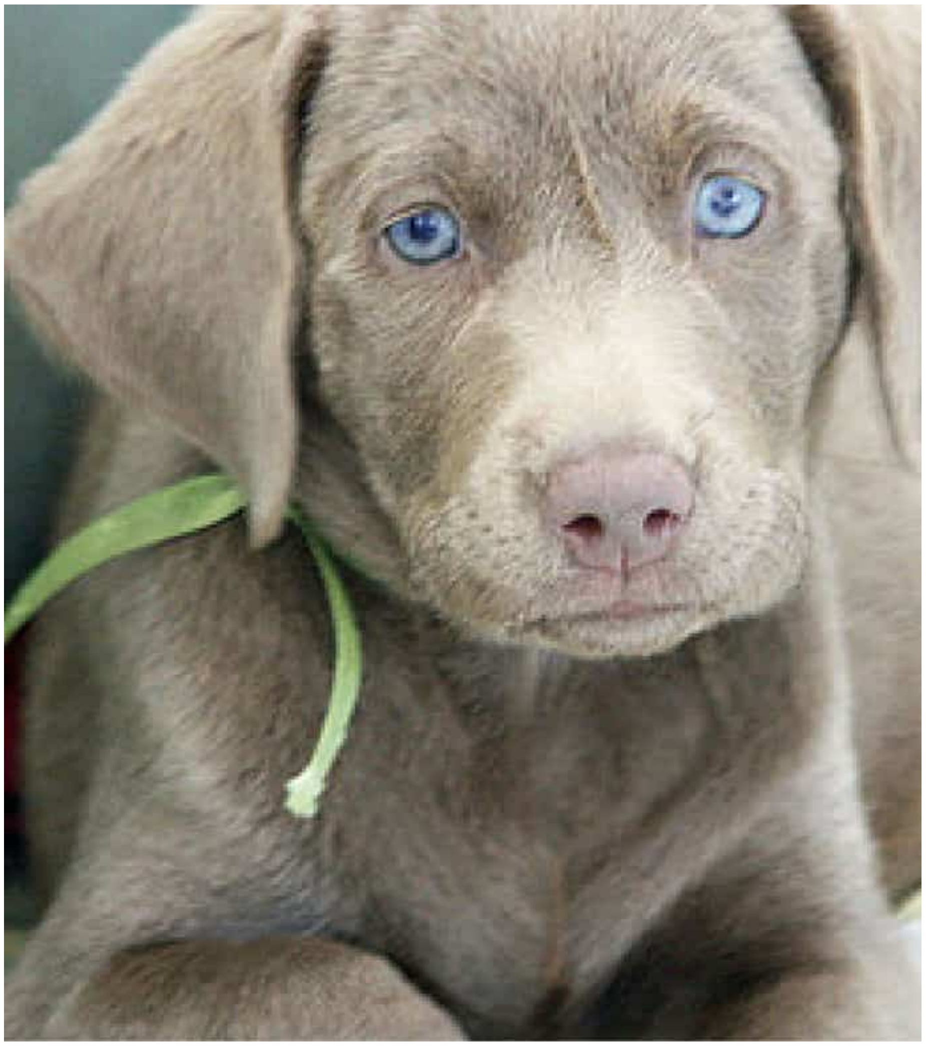

Figure 2: The legitimacy of a fourth color, silver, is widely contested amongst breeders

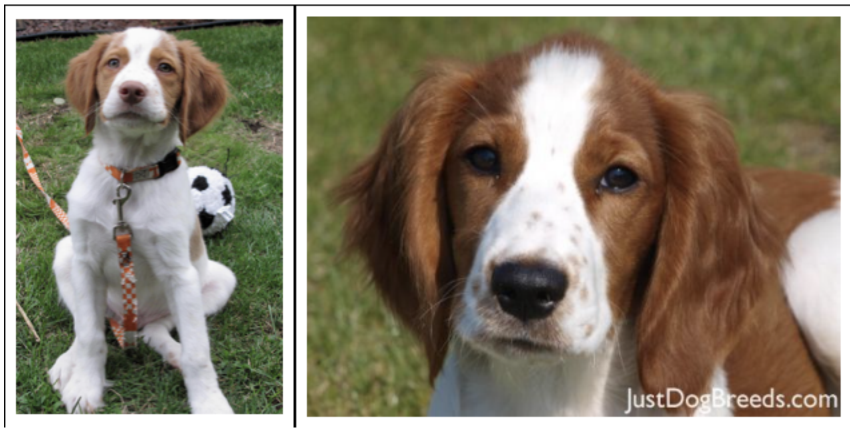

Figure 3: It is not easy to distinguish between the Brittany (left) and the Welsh Springer Spaniel (right) due to similarities in the patterned fur around their eyes

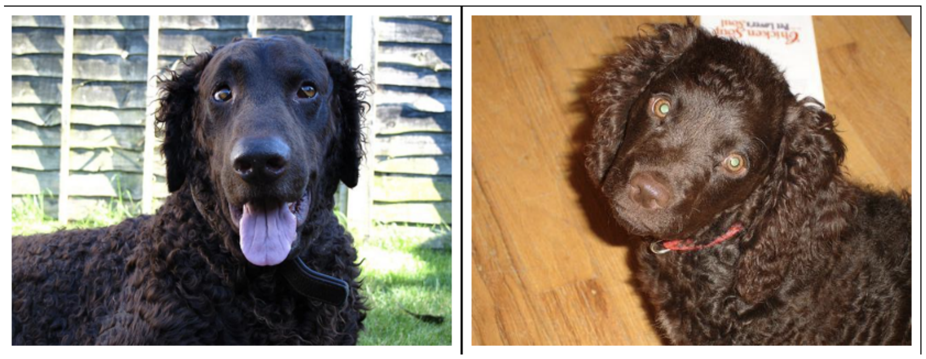

Figure 4: Another pair of dogs, the Curly-coated Retriever (left) and the American Water Spaniel (right), that are difficult to tell apart from the texture of their coats

Problem Statement

In order to identify the breed of a dog, a model would first need to identify if the image even contains a dog. Once the presence of a dog in the image is confirmed, a multi-class classifier can then begin to determine which breed of dog the image contains. For the scope of this project, 133 breeds were included in the model.

In addition to classifying images of dogs by breed, it would be interesting if the model passed human images as valid input to the classifier as well and found the closest matching breed depending on the human's face. For that purpose, another model can be used to identify if the image contains a human. Once a human has been identified as being present in the image, it can be fed into the multi-class classifier to determine which of the 133 breeds is the closest match.

All in all, the solution should have three models: two binary classifiers to determine if the image contains a human or a dog, respectively, and a third multi-class classifier to identify the dog breed of the human or dog in the image.

Metrics

Using a test set from the `dogImages` dataset, it is possible to measure accuracy by counting `true positives` identified by the model and weighing them against the total number of predictions made. This calculation is outlined below.

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Figure 5: Accuracy

Accuracy is a decent measure for evaluation of machine learning models, but a far better approach is to evaluate the precision and recall of a model. Accuracy, alone, is not a very helpful metric for understanding what changed between iterations of models. For instance, if a model has very high precision but low recall, then it might be necessary to retrain the model on more data; on the other hand, if a model has good recall but lower precision, then it might be necessary to tune the features being used. Accuracy might be identical across both of the previous scenarios and does not lend very much information for improving model performance. A graphic below explains the calculation for both precision and recall.

A further calculation can be made using precision and recall, yielding the F1 score, a metric which can be used to effectively compare performance between

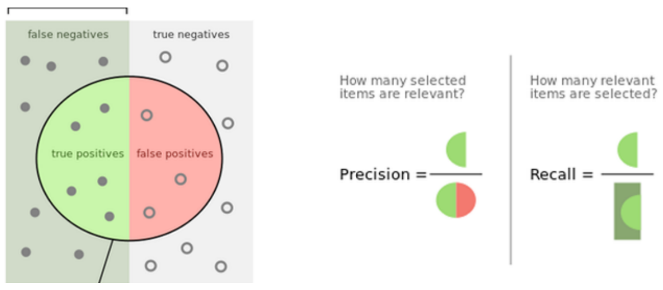

Figure 6: Precision and Recall

different iterations of a model.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Figure 7: F1 Score

For the scope of this project, it will be sufficient to calculate the precision, recall, and F1 score for the CNN against a test set from the `dogImages` dataset alone. The binary classifiers used to detect dogs and humans will not be evaluated since they are both out-of-the-box models only used for binary classification.

II. Analysis

Data Exploration

There are two datasets for this project to be used for this project - dogImages, which contains images of dogs categorized by breed, and lfw, which contains images of humans separated into folders by the individual's name. The dog images dataset has 8,351 total images split into 133 breeds, or categories, of dogs. The data is further broken down into 6,680 images for training, 835 images for validation, and 836 images for testing. The human images dataset has 13,233 total images across 5,749 people. Below are some examples of images from both datasets.

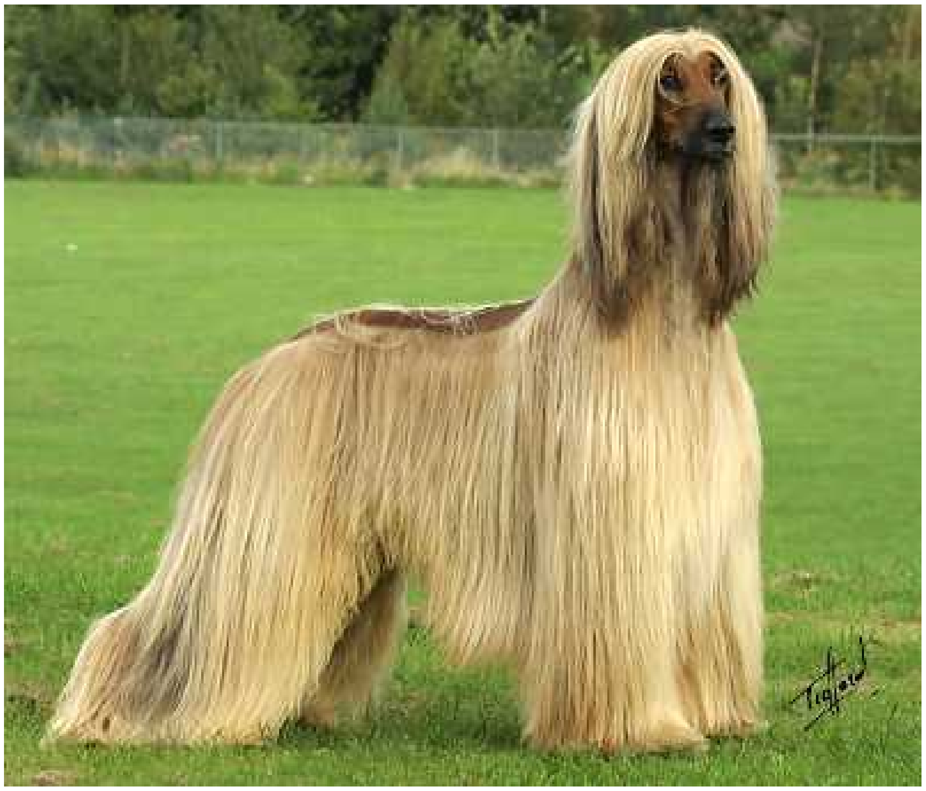

Figure 8: An image of the Afghan Hound breed from the `dogImages` dataset

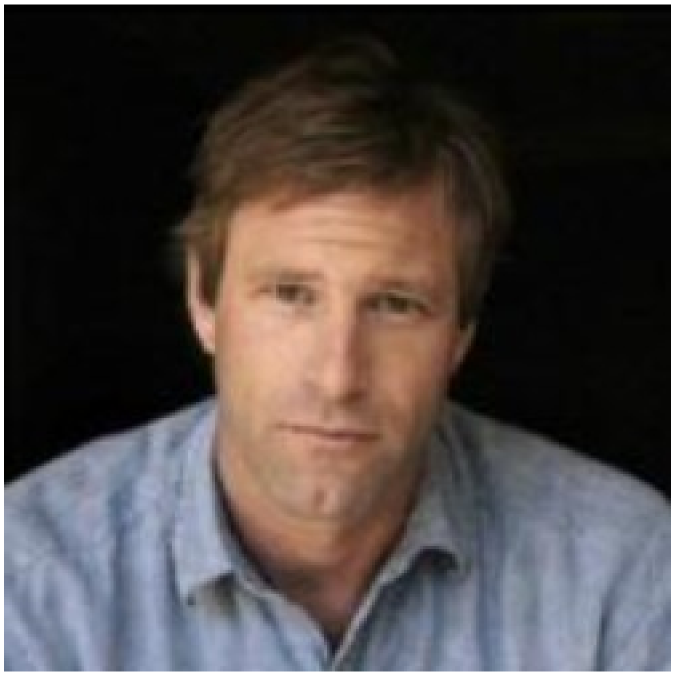

Figure 9: An image of Aaron Eckhart, who also goes by Harvey Dent in some circles, from the `lfw` dataset

As far as the image resolutions go, the data consists of varying sizes and dimensions. Since the CNN will need to be fed in standardized input, the data will need to be pre-processed prior to using it in our model. The latter sections will cover the specifics of the pre-processing techniques to be used.

Exploratory Visualization

When picking a model to use for transfer learning to build the CNN, it would be helpful if there was a measure to gauge the effectiveness of the model before going through training. One possible way to do this would to evaluate the models on their capability to distinguish between images of humans and dogs. In the plots below, four different models have been plotted against the training set data so the visual representation of the features used to detect dogs and humans can be understood.

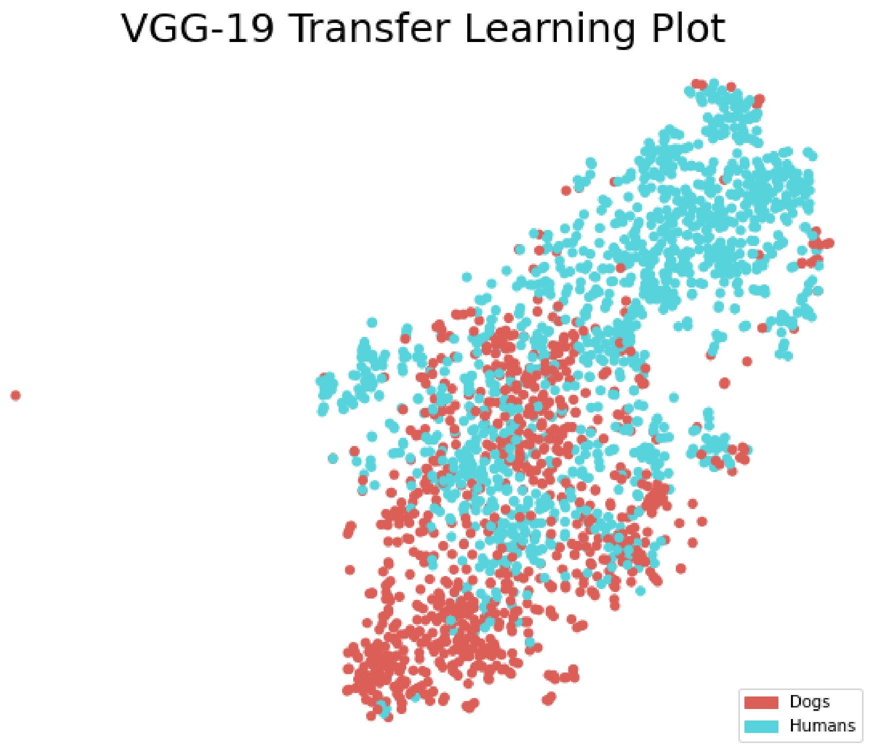

Figure 10: VGG-19 Model

From looking at the graphs, the ResNet-50 model seems to have the most distinct separation between dog and human data points. Inception-V3 has a close but rigid line of separation, whereas Xception and VGG-19 seem to have more

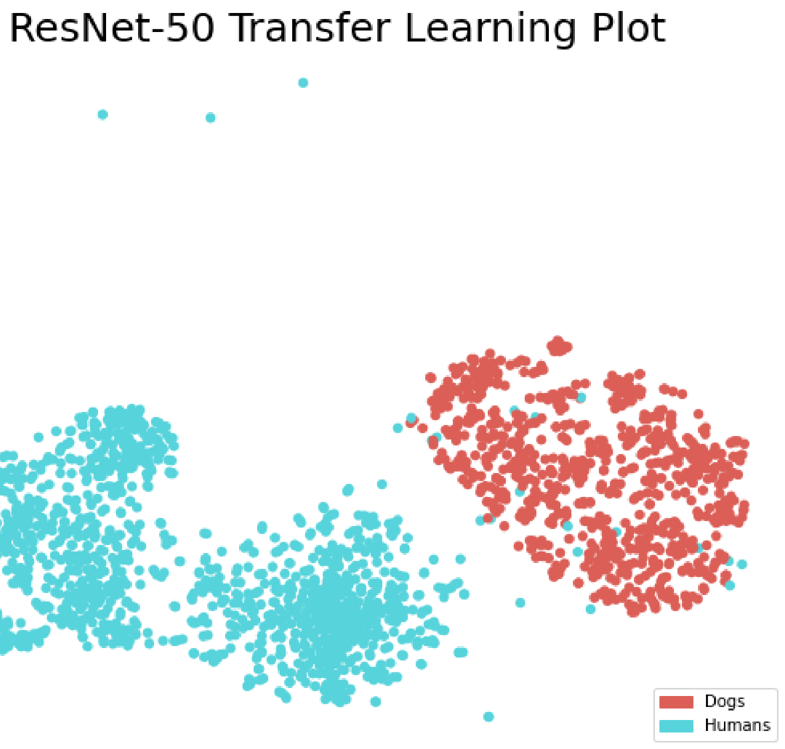

Figure 11: ResNet-50 Model

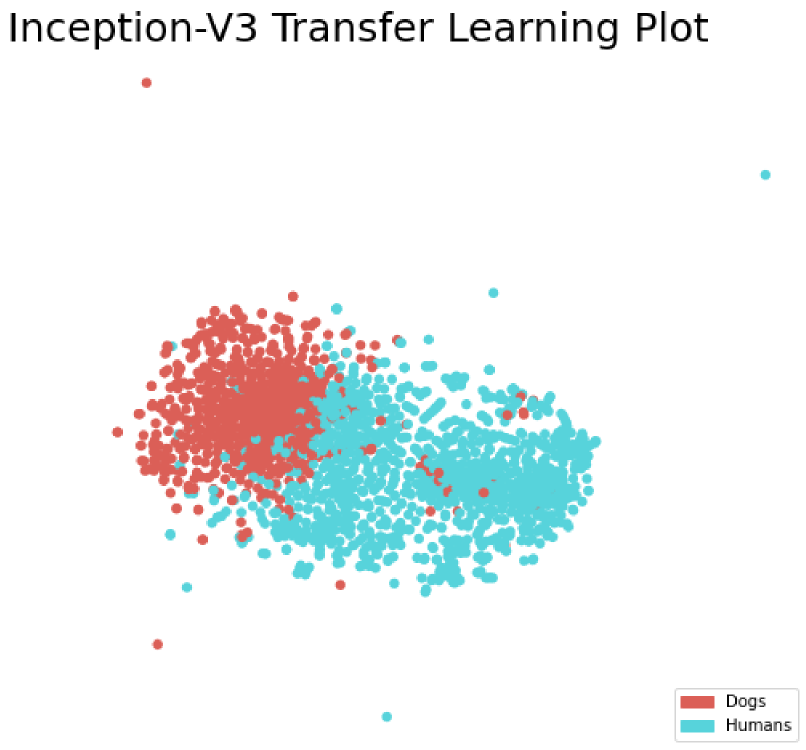

Figure 12: Inception-V3 Model

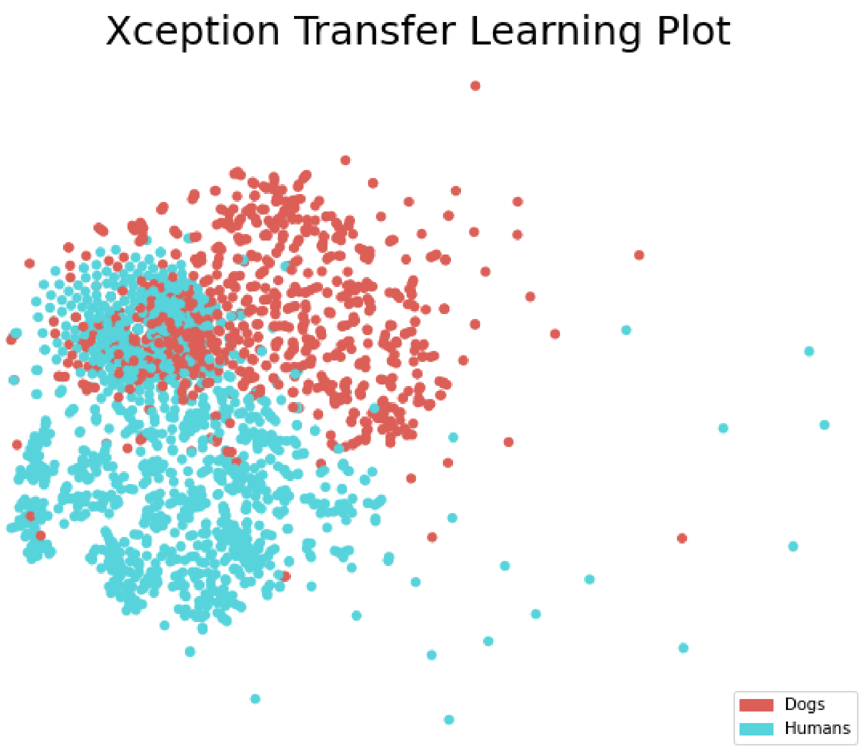

Figure 13: Xception Model

interspersed data points. It is worth investigating their underlying architectures and reading any relevant research papers to determine which model to go with, but this is an interesting exercise to get another perspective.

Algorithms and Techniques

There will be a total of three models, as stated earlier. The first model, `face_detector`, will use OpenCV's implementation of Haar feature-based cascade classifiers to determine if a human face is present, returning `true` if a face is present, or `false` if a face is not present in the image. The second model, `dog_detector`, will leverage a pre-trained VGG-16 model to identify if the images contain a dog, returning `true` if a dog is present, or `false` if a dog is not present. The third model will be a CNN trained using transfer learning from a pre-trained VGG-19 model.

As the focus of this project is on the CNN multi-class classifier, it would be fitting to discuss its architecture in more depth. To give a brief overview on the structure of neural networks, they typically consist of connected nodes organized by layers, with an input layer, an output layer and some number of hidden layers in between. At a very high level, the input layer is responsible for taking in data, the hidden layers apply some changes to that data, and the output layer yields the result. There will actually be two CNNs built for this project, one from scratch and another using transfer learning. The goal is to understand the architecture of a CNN better by building one from scratch and then train one using transfer learning to achieve better results. Both CNNs will contain an input layer that takes in a pre-processed 3 x 224 x 224 image tensor. That tensor will then be mapped across five convolutional layers and two fully connected layers to generate a final prediction out of 133 categories.

In the CNN that will be built from scratch, the five convolutional layers will all be normalized and max-pooled, and the two fully connected layers will be configured with 50% probability of dropout to prevent overfitting. This architecture is inspired by the AlexNet model. All layers will use Rectified Linear Units (ReLUs) for the reduction in training times as documented by Nair and Hinton.

> Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Neural Network Based Image Classification Using Small Training Sample Size. In *Proceedings of ICLR*, 2015.

> Vinod Nair and Geoffrey Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of ICML*, 2010.

In the CNN built using transfer learning, a pre-trained VGG-19 model will be used to get much better results than the CNN trained from scratch. Transfer learning involves taking a pre-trained model and using its learnings to get better outcomes for a closely related problem. For instance, a model used to identify images of stop signs could benefit from the learnings of a model trained to identify other street signs. After reading the research paper and looking at the results of the VGG-19 implementation, it seems like a good fit for this project.

Benchmark

The pre-trained VGG-16 model used as the `dog_detector` model can be used as a benchmark model for breed classification since it can also be used to identify dog breeds in images provided as input. The model performs with an accuracy of roughly 40% on a test set of 100 randomly selected images. While this performance is not stellar, it can be used as a benchmark to compare against the CNN. The goal of this project is to create a CNN that out-performs the VGG-16 model and delivers an accuracy rate of greater than 60%.

III. Methodology

Data Pre-processing

In addition to separating the data into training, test, and validation sets, the data also needs to be pre-processed. Specifically, all images should be resized to 256 x 256 pixels and then center cropped to 224 x 224 to ensure that they are all uniform in dimensions for the tensor. Also, the red, green, and blue (RGB) color channels should be normalized so that gradient calculations performed by the neural network during training can done more consistently and efficiently. Apart from these items, the images are part of a prepared dataset so there are no abnormalities or inconsistencies that need to be addressed in data pre-processing.

Implementation

The end-to-end functionality of this project will employ a multitude of separate components. Any input would first be run through the `face_detector` and `dog_detector` functions to determine if the image is of a person or dog. Afterwards, the image would be fed into a CNN to determine the appropriate breed to which the image should belong.

The machine learning pipeline will include:

1. Importing the datasets
2. Binary classification
    - a. Detecting humans
    - b. Detecting dogs

3. Convolutional Neural Network (CNN)
    - a. Classifying dog breeds from scratch
    - b. Classifying dog breeds using transfer learning
4. Algorithm implementation for images provided as input
5. Model performance testing

The `face_detector` function will leverage OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. The `dog_detector` function will utilize a pre-trained VGG-16 model to detect dogs in images. The CNN will use transfer learning and extract bottleneck features froma VGG-19 model.

To better describe the function of the CNN in this project, the illustration below shows an example of how CNNs handle image classification. The convolutional layers set up with normalization and max-pooling layers, extract features from a provided input image. Those features are then used to perform non-linear transformations in the fully connected layer and produce a classification result.

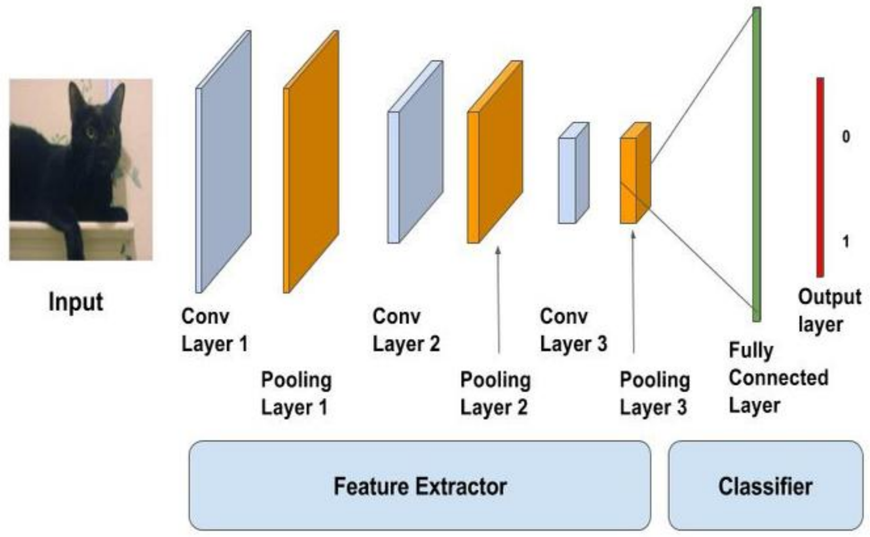

Figure 14: CNN Schema

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of NIPS*, 2012.

As mentioned in earlier sections, there were two CNNs built for this project. The one that led to issues in particular was the one built from scratch. This CNN contains five convolutional layers, all normalized and max-pooled, and two fully connected layers with dropout configured at 50% probability. This architecture was modeled after AlexNet. All layers use Rectified Linear Units (ReLUs) for their documented reduction in training times. Even after all of this, the trained model performed quite poorly with the validation loss function severely increasing as the training loss function decreased, a classic sign of bad overfitting. Ultimately, the model managed to identify less than 10% of dog breeds correctly from the test set. Despite several hours' worth of work to combat the overfitting problem, such as trying many iterations of simpler architectures containing only three convolutional layers and increasing the dropout probability of the fully connected layers, the model's performance did not improve. Ultimately, this effort was abandoned in favor of the transfer learning method to create a more proficient model in a lesser amount of time.

Refinement

There were several iterations on the CNN in this project, specifically in terms of modifying the number of convolutional layers, configuring dropout, adding max-pooling, and implementing batch normalization. All of these adjustments were made in an effort to lower the degree of overfitting exhibited by the model. Overfitting would imply that the model performs well on the training set but displays noticeable degradation in performance when evaluated against a validation set. This could be observed by noting the divergence of the validation loss function from the training loss function during training; simply put, the validation loss function would begin to increase while the training loss function would continue to decrease, a blatant sign that the model was overfitting. This occurred quite a number of times when training a CNN from scratch, but this behavior was not observed when training the CNN using transfer learning.

IV. Results

Model Evaluation and Validation

The final CNN trained using transfer learning performed at a satisfactory level, correctly classifying 60% of the 836 unseen images in the test dataset. This corresponds to a precision of 60% and a recall of 60%. Since this is a multi-class classification problem, every incorrect classification performed by the model is categorized as both a `False Positive (FP)` and a `False Negative (FN)`. The calculations for precision and recall are included below to illustrate why they are equivalent in this scenario.

While this performance of the model is not particularly extraordinary, it is sufficient for the purposes of this project. While the model is adequate for its

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Figure 15: Precision and Recall

purposes, it can be improved by spending more time to perform some additional tuning that is documented earlier in the report. The results from this evaluation can be trusted as the data used for evaluation is data that was unseen by the model during testing. The final model was also thoroughly evaluated on how well the model generalizes to unseen data during training, since a validation set was being used at every training epoch to determine how well the model was performing.

Justification

The model's final performance of 60% accuracy on the training set is definitively stronger than the benchmark of 40% that was set earlier in the project. The final solution is sufficient to solve the problem posed in the project but at the very bare minimum. Ideally, the final solution would have outperformed the minimum threshold by a significant margin. Overall, the final solution solves the problem at a passable mark.

V. Conclusion

Free-form Visualization

Above is one example of an architecture of a CNN. There were several images like this that were encountered while reading several research papers, trying to learn more about the functions of different layers and how they can be adjusted to improve the results of the model. This CNN architecture, in particular, is of the VGG-19 model. The five convolutional layers have max-pooling applied

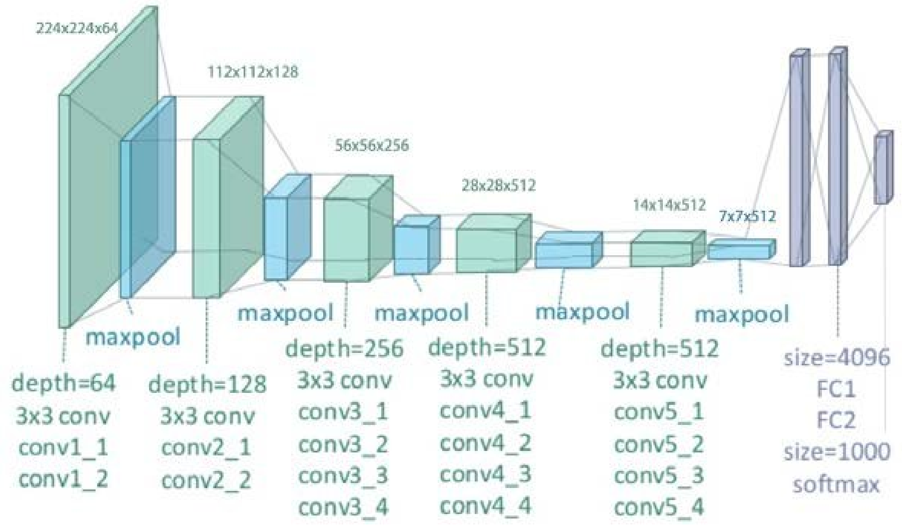

Figure 16: VGG-19 CNN Architecture

at every layer, compared to the model described in the AlexNet paper, which applies max-pooling for just the first two of five convolutional layers (as well as the fifth convolutional layer). Also, the VGG-19 architecture shows three fully connected layers, whereas the previous CNN built from scratch contained only two. This was another interesting deviation from the AlexNet model as well.

Reflection

The end-to-end solution was challenging to complete, specifically difficult was the task to build a CNN architecture from scratch. This was also the most interesting portion of the project and required a firm understanding of CNNs to complete successfully. This part of the project was, in a sense, designed to fail as the bar to pass was set very low at 10%. Employing multiple methods to reduce overfitting was necessary, and even then the results were not promising. In general, using a less complex CNN, with fewer layers, is less likely to overfit, and dropout layers may increase training time but will lead to the model being less reliant on the same features to produce a result and will decrease overfitting. The final model's performance was quite underwhelming, the expectations were for the model to perform much better than it did, but it was a great introduction to CNNs, and they are definitely wonderful tools to use for image classification problems. Ultimately, the most valuable lesson this project imparted was on how to find relevant research papers, explore the architectures of the solutions being used to solve those problems, and understand how those solutions are applicable to the task on hand.

Improvement

There were potential changes that could have been made to improve the performance of the solution in this project: augmenting the training dataset by adding flipped/rotated images would yield a much larger training set and ultimately give better results, experimenting with even more CNN architectures could potentially lead to uncovering a more effective architecture with less overfitting, and utilizing more training epochs, given more time, would both grant the training algorithms more time to converge at the local minimum and help discover patterns in training that could aid in identifying points of improvement. Out of those three, the one that was definitely the easiest to implement, and should have been completed to begin with, was performing data augmentation to the training set with flipped and rotated images. This task alone, would have provided the model with a sizably larger training set to train upon. Using the final solution of this project as a benchmark, it is definitely possible to iterate on it and outperform 60% on dog breed classification.