Image Classification of Mixed Breed Dogs

Allison Aprile
Advisor: Dr. Richard Souvenir
B.S. Data Science (Computational Analytics)
Temple University, Philadelphia PA

Abstract

Mixed breed dogs populate shelters more than any other animal. Due to the lack of background knowledge and resources, shelters must rely on the dog's visual attributes to identify its parent breeds. There exist no public datasets nor few deep learning projects concerning mixed dog breed image classification. Thus, this project focused on creating a mixed breed dataset using Petfinder data and using the purebred Stanford Dogs Dataset [1] to develop a generalized dog breed classification model capable of predicting one or more parent breeds, given an image. Throughout the semester, I created a web scraper, compiled a mixed breed dataset, was familiarized with HPC [5], and built a variety of models examined with the mixed breed dataset. This project is an application in the domain of computer vision, particularly multi-label fine-tuned image classification.

1. Introduction

Animals are entering shelters at increasing rates, causing overcrowding and a need for pet adoption. Several companies have eased the adoption process for consumers by using technology to advertise available pets. Petfinder is one of the original and largest online adoption services, hosting a database of over 255,00 animals. Like most shelters, Petfinder's listings consist of primarily mixed breed dogs: dogs with parents of two or more different breeds. Take, for example, the popular Labradoodle (**Figure 1**), having one parent as a purebred Labrador Retriever, and the other as a purebred Poodle. Although the Labradoodle is a success story of breed mixing, most 'failed' attempts end up in the shelters. The overpopulation of mixed breed dogs in shelters is an unfortunate consequence of the modern desire to invent the next 'designer' breed.

Figure 1. Labradoodle, resulting from mix of purebred Labrador Retriever and Standard Poodle

Even more unfortunate, shelters lack the resources to properly identify the mixed breed dogs, such as with a DNA test or breeding certificate. With so many dogs coming to shelters with unknown parents and backgrounds, visual classification is the only viable way to hypothesize the breeds. This hypothesis is not generally an issue but can play an important role of the adoption

retention, given owners with certain restrictions or even fears with certain breeds (e.g. many apartments do not allow Siberian Huskies, and Pit Bulls have an aggressive reputation).

The challenge of mixed breed classification for shelters, as well as for curious dog owners, can possibly be lessened through machine learning. Finalized machine learning models are typically quick, consistent, and automated, which would allow busy shelters to expedite and perhaps improve their initial breed predictions. Therefore, the goal of this project was to use deep learning, specifically Convolutional Neural Networks, to build a mixed dog breed image classification model. Although behavior and other factors could play a huge role in determining parents, visual attributes are often a large indicator of a dog's genetic history.

Convolutional Neural Networks are similar to standard neural networks, having the regular input transformations and connected layers built from neurons. However, they differ in their architecture, specifically with the layers organized in three dimensions (for width, height, and color channel/depth), which are not necessarily fully connected. Thus, Convolutional Neural Networks are designed for image classification and the appropriate deep learning class for this problem.

Dog breed identification complicates the standard image classification problem. While cat-versus-dog classification relies on the large inter-class variation, dog versus dog classification must tackle the large intra-class, as well as background, variation [2]. That being said, mixed dog breed classification is considered fine-grained. Fine-grained image classification typically involves feature extraction and part localization, the first of which is explored in this project through the use of bottleneck features in the model training. Bottleneck features, simply the last weight output before the fully connected layers, are key to eliminating noise in larger datasets and reducing overfitting.

There is the additional challenge of training data availability, as most of the available dog image datasets consist solely of purebreds. Creating a mixed breed image dataset for testing is troublesome when considering the unlimited combinations of dogs; it is not at all scalable. Because the model should also be more generalized, a more reasonable approach is training the model on a collection of purebred images and modifying it to output multiple breed predictions, given a mixed breed input. For this project, the Stanford Dogs Dataset [1] was used for training, testing and validation, while a mixed-breed experimentation dataset was created using the Petfinder API.

This project was initiated in April 2019 for a Data Science course, with the goal of testing whether Petfinder's primary and secondary breed classifications were predominately based on visual attributes. This was broadened to the more applicable goal of creating a mixed breed image classification model, capable of classifying two or more parent breeds, using the mixed breed dataset for result examination only.

Throughout the semester, I worked with the Petfinder API to create a mixed breed dataset, familiarized myself with Temple's High Performing Cluster (HPC) [5], and built a variety of classification models. Despite not completely finishing the project, I did create a moderately accurate mixed breed classifier that yielded interesting results with my Petfinder dataset.

2. Related Work

2.1 Stanford Dogs Dataset

The Stanford Dogs Dataset [1] is an image dataset of purebred dogs, built and published by Stanford University in 2011. It consists of 20,580 dog images, spanning 120 breed classes, sourced and annotated by ImageNet. It is intended for use in Fine-Grained Image Classification problems, as popular on Kaggle. As of 2011, the baseline results included a mean accuracy of 22%, with 100 training examples per class. It is noted that the testing images overlap with the training (in ImageNet). Therefore, transfer learning with models pre-trained on ImageNet cannot represent the true performance [2].

2.2 Mixed Breed Dogs Classification

Arava, Yang and Yuan [2] authored the only current work on the topic of mixed dog breed classification. They differentiate the standard image classification with this fine-grained classification problem, where there are small inter-class but large intraclass variations in the dog dataset. Further, they discuss how visually identifying the purebred dogs is complicated by the visual inconsistency in their mixed breed children; dogs having the same parent breeds can have drastically different features, including fur texture, coloring and dominant facial structures. Therefore, they took various approaches to the problem, particularly focusing on enhancing the input data.

The initial goal was to build a purebred classifier and run the mixed breed images through such to generate a prediction. Their base models were pretrained, then fine-tuned with the Stanford Dogs Dataset. However, as discussed in Section 2.1, transfer learning was not a viable approach due to overlap between the testing and ImageNet training data; they considered the testing set to be ‘contaminated’. To bypass this, they trained various models (Inception-v3, Inception-v4, ResNet18 and ResNet50) built from scratch. All yielded large computational complexity and testing accuracy less than 10%, results similar to the trial runs of this project. Because pre-trained models were necessary, they adjusted their goal to build a mixed breed classifier. Then, they could use their mixed breed images as the strict testing dataset.

They built four mixed breed classification models: Fine-tuned Inception, Saliency Augmented Inception, Data Augmented Inception, and Attention. All models were trained on the Stanford Dogs Dataset and tested on their mixed breed dataset, images collected from the Internet independent of ImageNet. They ensured that the mixed breed dataset consisted of dogs with exactly two different purebred parents, whose classes were a subset of the Stanford Dogs Dataset. For the first model, they fine-tuned an Inception-v3 model (pretrained on ImageNet) by adding one more fully connected layer on the top to give the scores for 120 classes. To boost its performance, they incorporated saliency maps into their second model. The saliency maps differentiate the pixels that are more important for classification. Generated with a pretrained ResNet50 architecture, they were used with an

intensity parameter to generate augmented part images (predominately face characteristics), decreasing overfitting and preserving labels despite the transformations. For Data Augmented Inception, they implemented feature-wise and sample-wise standardization, as well as typical data augmentation (e.g. flipping and cropping) to the input data. This increased the training data size and improved the validation accuracy. Finally, the Attention model was trained on the original Stanford Dog images first, and then run on a variety of tests based on another project's guidelines. Attention models are designed for part localization and are advantageous as they do not require annotations. Nonetheless, the Attention model provided for only a small accuracy improvement despite its high computational cost.

The model performance was based on the accuracy of predicting the parent breeds independently (i.e. only attendance in top K predictions, not order, considered). For identifying the top breed, the Saliency Augmented Inception was superior, while the data augmented performed the best for the top K greater than one breeds.

Overall, they determined that incorporating saliency maps and data augmentation into pretrained Inception models can improve the performance in the fine-tuned mixed breed classification problem. Although the overall improvement is not tremendous, these methods are suitable for increasing the performance for the misclassifications while maintaining the other accuracies.

3. Data

3.1 Purebred: Stanford Dogs Dataset

The Stanford Dogs Dataset [1] includes 20,580 purebred dog images, spanning 120 breeds. The .tar file was downloaded directly from the website and unpacked using the command line. The images were located in one directory, organized by subdirectories corresponding to each class. To make this data workable for model training, I wrote a quick Python program that would Train/Test/Validation split the directory, update the breeds with the correct mapping (see Section 3.2), and create a CSV containing the image ID and the class label.

3.2 Mixed Breed: Petfinder

Because a mixed breed dog image dataset is not easily accessible, it was necessary to create one. The dataset was originally intended to be used for experimentation, rather than testing, so the images and labels were acquired using the Petfinder API. Importantly, the Petfinder labels do not necessarily represent the ground truth parent breed labels. It can be assumed that majority of the dog breeds were labelled based on visual appearance; due to the high cost and time associated with DNA testing, it is not a sustainable method for shelters with many incoming animals.

I wrote a Python program that handled the API requests and responses. After connecting with an API Key and Secret, I implemented a while loop with a try-except block that would handle the time constraints, limits, and invalid responses. Because the Petfinder Database is constantly updated, it was important to set the pages to static and scrape from the oldest to newest records. This detail allows the scraper to be more flexible and reproducible, and further, the mixed breed dataset to be easily increased without repeating records. The scraper returned all dog records up to the most recent posting of that date.

The organization of the dataset was intended to mimic that of the Stanford Dogs Dataset for easily translation between the purebred and mixed breed classifiers. This included a folder (**Figure 2**) containing images named with a unique ID, which would be matched with a breed classification on a CSV. Because the API responses were best accessible in JSON format, there was a significant amount of postprocessing. To eliminate some, I incorporated several filters to accompany each response before calling the image download and CSV write commands.

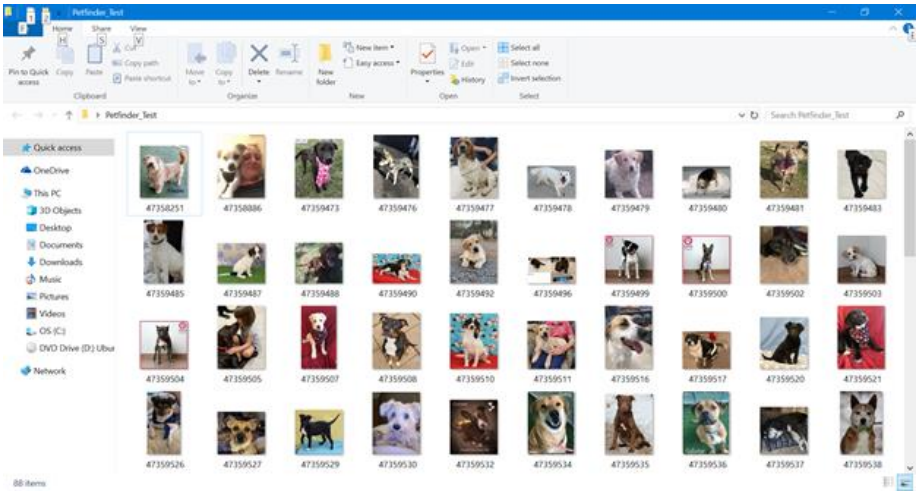

Figure 2. Results of Petfinder scraper

First, as the Stanford Dogs Dataset labelled the breeds differently from Petfinder, I created a breed mapping (**Figure 3**). The Stanford Dogs Dataset could not be easily changed or appended, so the Petfinder list was reduced and mapped to match the 120 pure breeds. The mapping required obtaining a list of all the Petfinder breeds and manually comparing it to the Stanford Dogs breeds. I used Excel and Google Search to confirm the breed names, as well as research any alternative names (e.g. ‘Corgi’ versus ‘Cardigan’) and delete any unrepresented breeds (e.g. Australian Shepherd). Clearly, the Petfinder breed list was more robust than that of Stanford Dogs, as the latter is outdated; it does not include any new breeds or officially recognized mixes, such as the ‘Labradoodle’.

Australian	australian_terrier
Basenji	basenji
Basset Ho	basset
Beagle	beagle
Bedlingtor	bedlington_terrier
Belgian Sh	malinois
Belgian Sh	groenendael
Bernese M	bernese_mountain_dog
Black Labr	labrador_retriever

Figure 3. Subset of breed mappings

To incorporate the mapping into the scraper, I created a dictionary having the keys as the Petfinder labels and the values as the Stanford Dogs labels. This determined if the current response's breeds were represented by the Stanford Dogs Dataset. If so, the image would be downloaded and saved as the ID, and the ID would be written to a CSV alongside the hypothesized breed classifications (primary and secondary). If not, that response would not be included in the dataset.

The JSON structure of the responses complicated the scraping process. The Petfinder records include many nested attributes, which translates to nested lists of dictionaries when JSON normalized with Pandas. Certain features, specifically the images, were difficult to unpack and required complex indexing to access. Nonetheless, over 50,537 mixed breed images, spanning 114 of the 120 breeds were collected.

4. Methodology

4.1 HPC

The volume of data and proposed model architectures would result in long training times if completed on my local machine. Therefore, I predominately ran scripts on Temple's High-Performance Computer cluster (HPC), specifically the Owl's Nest Linux cluster [5]. Hosting over 6,464 CUP cores and access to various size nodes, it was a vital resource for expediting model experimentation. I had access to three different storage paths, including home (for scripts, source code and results), work (for data), and scratch (for sharing temporary data). I only work on home and work, using the former for my scripts and the latter for my images and weights, as intended.

To use HPC, jobs must be submitted to the Portable Batch System (PBS) with a shell script. The scripts are plain text files including resources, such as wall-time and nodes, and executable statements. The PBS has machine learning workflows with virtual environments or containers (a portable way of defining environment), which made it compatible with my project. Familiarizing myself with Singularity image and containers, as well as HPC in general, was a great proportion of my project. Throughout the

semester, I served as an unofficial tester of my teammate's, HPC documentation (see Acknowledgements). I worked with him to get more comfortable with Singularity and provide feedback as a new user. Among the other trial runs, he helped me pull a TensorFlow container, build a writable directory (Sandbox) with it, and then download images using the environment (**Figure 4**).

```
module load singularity
singularity
singularity pull docker://tensorflow/tensorflow:latest-gpu-py3
singularity run
singularity run tensorflow_latest-gpu-py3.sif
singularity build --sandbox sandboximg tensorflow_latest-gpu-py3.sif
```

Figure 4. History of Singularity Linux commands

Before running my models, I submitted some testing scripts using the Singularity image 'tensorflow_latest_gpu_py3.sif'. This provided Python3 and TensorFlow capabilities. However, the scripts all failed on the imports because my project involves many Keras packages. Because the Sandbox approach is rather volatile, it is recommended to work with virtual environments. I attempted this but had many issues, particularly with the package installation. I also tried to debug locally by first trying to pull the latest Docker image, then pulling development images and simple TensorFlow containers, and finally by secure copying other images to/from HPC – all of which failed. During one of the research discussions, it was brough to my attention that I had been overcomplicating the process; I just need to work with a Singularity image constructed with Keras (I selected 'tensorflow-keras-18.06-py3.simg') and pip install the necessary packages once running the image. Using this, I successfully submitted and completed several test jobs. HPC proved to be a valuable tool for the remainder of the semester, both allowing me to complete model training quickly and learn more about Linux and high-performance computing.

4.2 Purebred Classification Models

My approach to building a mixed dog breed image classifier consisted of using the Stanford Dogs Dataset to fine-tune a Convolutional Neural Network (CNN) pretrained on ImageNet. I initially aimed to build a purebred classifier with at least 75% accuracy, which then could be slightly adjusted to become a mixed breed classifier. Ultimately, the goal was that model would learn to recognize the dominant visual features of purebred parents apparent in the mixed breed child dogs.

4.2.1 Experimentation

I first explored a variety of pretrained models, including ResNet50, VGG-19, InceptionResNet-v3, and MobileNet, using a sample of 938 images from the Stanford Dogs Dataset. I decided to work with a subset data in the trial runs for a variety of reasons: first, at the time I was still getting familiarized with HPC; second, the resources were sufficient that model could be built locally and the coding process is more interactive; and finally, debugging the model would be more controllable with five, versus 120, breed classes. Without modification, the pretrained models are sufficient for higher-level image classification (e.g. cat versus dog), while fine-grained image classification requires the fine-tuning of these models [2]. This can be accomplished with layer revision, data augmentation, and feature extraction, all of which were attempted.

While all of the layers can be modified in some way, the main focus is the final Dense layer. The Dense layer is a fully connected layer, meaning that every input is connected to every output by a weight. It is generally followed by a non-linear activation function [3]. In the purebred instance, this activation is ‘softmax’, and in the mixed breed, ‘sigmoid’. The final Dense layer in pretrained network defaults 1,000 neurons, so this number must be changed to the number of classes, being 120 for this project. This is achieved by simply removing and redefining the top layer.

ImageDataGenerator was the simplest method for collecting input and data augmentation. ImageDataGenerator is offered in the Keras Preprocessing package. In short, it sends the data for training in batches, applies a series of random transformations, and then uses the new images to train the CNN. This increases the generalizability of the model input [4].

Before experimenting with feature extraction, I ran my first base model with the MobileNet architecture, which simply included the ImageDataGenerator and the subset of training images. Although it trained very fast, the accuracy was below 0.04. After a research discussion, I learned that MobileNet is not designed for this volume of data and classification, and was recommended to try ResNet50, VGG-19 and Inception-v3. My simple ResNet50 model achieved 0.04 training accuracy and replacing the base architecture with VGG-19 and Inception-v3 made no notable difference.

I researched other purebred dog classification models (strictly those using the Stanford Dogs Dataset) to understand the mistakes of my approach. My experience was not uncommon; majority of other basic attempts with transfer learning had recorded validation accuracies below .10. However, the use of bottleneck feature extraction was responsible for significant improvements in performance.

Formally, bottleneck features are the last activation maps (i.e. the final output of the convolutional layers) of the input data before the fully connected layers in a pretrained model. Essentially, the pretrained model, trained on a large dataset like ImageNet, is used as a feature extractor for the smaller dataset. These features can be then fed into a smaller fully connected network to output class predictions. The procedure involves extracting the features of a dataset by predicting with a base, or simple, pretrained model. The

output are weights, which can be flattened and fed into a fully connected deep neural network classifier to yield the class predictions. Once training is complete, testing and other prediction involves the similar method of feature extraction on the input image and predicting using those features [4].

I attempted bottleneck feature extraction first with a VGG-19 base model, and then ResNet50. Both performed with 0.98 training and 0.82 validation accuracy on the five-breed sample. Clearly, this performance was not representative of the entire 120 class dataset, although the general infrastructure and feature extraction was promising. Therefore, I used it in my following attempts with the complete purebred classifier, as well as with the mixed breed classifier.

4.2.2 Model 1: VGG-19

My first attempt consisted of two models: a VGG-19 network pretrained on ImageNet for feature extraction, followed by a simple Sequential model for prediction. To build this model, I referenced the same tutorial as in the experimentation [4]. First, the Stanford Dogs training set was read and augmented with the ImageDataGenerator. Then, they were fed into the VGG-19 and the output weights were saved. Next, these bottleneck features were input to the Sequential model, which consisted of a final Dense Layer with 120 neurons and the ‘softmax’ activation function. The model was compiled using ‘rmsprop’ optimization and categorical cross entropy loss. It was trained for 100 epochs with a batch size of 32, and later for varying combinations of epochs and batch sizes with minimal performance difference.

With the introduction of 115 new classes, I was not expecting a great performance, although I did expect decent. The result was not at all viable: less than 0.011 validation accuracy and greater than 4.0 validation loss (**Figure 5**). To better understand the poor performance, I wrote a short Python program that would test a few images and highlight the misclassifications. Immediately, it was evident that all of the images were being mapped to Class 0 (i.e. the first alphabetical breed, Affenpinscher). This implied issues with input dimension, which could be traced back to a bug in the bottleneck feature extraction.

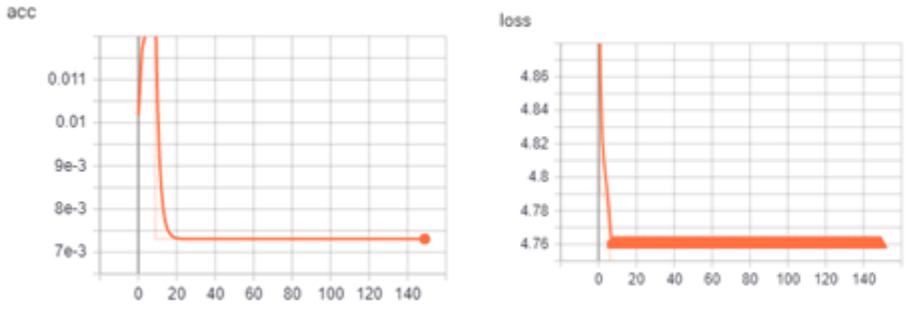

Figure 5. Validation accuracy (left) and loss (right) against epochs, Model 1

4.2.3 Model 2: ResNet50

After reading more into both the model architectures and bottleneck feature extraction more, I decided to try the ResNet50 architecture. As I understood, the ResNet50 architecture has a different sensitivity to input dimension, particularly the color channel. I was curious as to whether I had been accidentally changing the color channel to one (for grayscale) from three (for RGB images) in the previous model, resulting in an incorrect element in the fourth place of the image tensor (width=224, height=224, color channel=1 versus 3). Although I could not make a direct correlation to the single class predictions experienced in Model 1, it was still a great learning experience.

Again, this attempt included two models: a ResNet50 architecture pretrained on ImageNet for feature extraction, followed by a simple Sequential model for prediction. I approached the bottleneck feature extraction and Dense layer update the same way as for Model 1, but this time carefully paying attention to the ImageDataGenerator parameters and including sanity check print statements of the dimensions after each step. In a similar approach [6], Fredrickson included a GlobalAveragePooling2D layer, where the input shape in the Sequential top model disregarded the first dimension of the input tensors. Because the first dimension of the entire array of bottleneck features corresponds to the image index, it is sensible to exclude such. This made me curious as to whether this was intended for less pre-processing for prediction (i.e. reshaping the single input tensors from 3D to 4D), or to avoid results such as those experienced in Model 1.

The model was trained for 100 epochs with a batch size of 32, and then varying other combinations of such following. Unfortunately, this model performed just as poorly as Model 1, resulting in less than 0.01 validation accuracy and greater than 4.6 validation loss in all tests (**Figure 6**). Because the performance was not at all ideal, I continued to read into the bottleneck feature extraction process. I was able to pinpoint my issue as the ImageDataGenerator, which is notorious for its poor integration with other Keras applications and difficulty with large datasets for feature extraction [8]. Although a useful tool for data augmentation, the ImageDataGenerator was not workable for my project. I eventually opted to manually extract the features instead, but first looked into a model where the author limited, opposed to deleting, the dependency on the ImageDataGenerator.

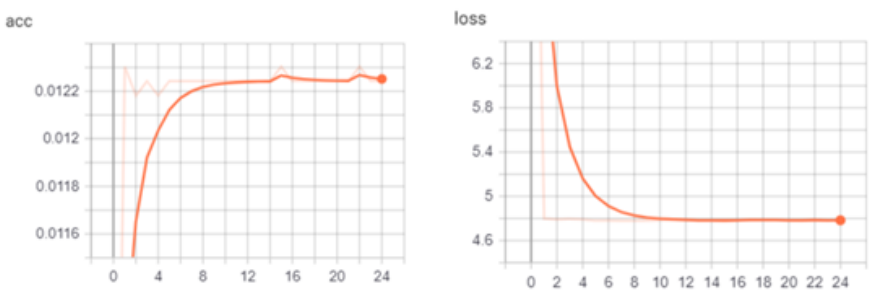

Figure 6. Validation accuracy (left) and loss (right) against epochs, Model 2

4.2.4 Model 3: Batch Normalization VGG-19

In his attempt at a purebred dog classifier [7], Vikram used a VGG-16BN model for both feature extraction and final prediction. Interestingly, he still incorporated the ImageDataGenerator, but manipulated the parameters so that there would be no augmentation or generator-involved prediction (i.e. calling 'model.predict' opposed to 'model.predict_generator' as in the previous model attempts).

I worked to update his model to a VGG-19 architecture. Unfortunately, the code and syntax were outdated, so I faced many issues trying to debug and modernize his code. Because of my time constraints, I decided to abandon this approach and instead try manual bottleneck feature extraction.

4.2.5 Model 4: Inception-v3

The last and most successful purebred model attempt consisted of two models: an Inception-v3 base network pretrained on ImageNet, used for feature extraction, followed by a simple Sequential model for prediction. Likewise to the other attempts, this model incorporated bottleneck feature extraction, but this time manually. To extract the features for the training, validation, and testing Stanford Dogs images, I wrote a simple Python script. Given the directory, the program would read and process the images individually, first loading and converting the image to a 3D tensor, having shape (width=224, height=224, color channel = 3). Then, it was converted to a 4D tensor with shape (index = 1, 224, 224, 3). To get the complete feature array, all of the individual image tensors were vertically stacked. It is important to note that the targets were independent of these tensors, saved into their own NumPy files after generated.

For help extracting the bottleneck features, I consulted the work presented by Mason [8]. He found that the Inception-v3 architecture performed the best with the manual features, so I followed that suggestion. First, I loaded the Inception-v3 model pretrained on ImageNet weights, specifying average pooling. I then fed the stacked tensor arrays into the model individually, saving the resulting bottleneck features as NumPy array files.

For prediction, I defined a new Sequential model, first featuring a Dense layer with 256 neurons and the 'relu' activation function, followed by a Dropout layer (0.5), and finally the standard Dense layer with 120 neurons and the 'softmax' activation function [9]. Then, I fed the training and validation bottleneck features into the model, compiling with the 'rmsprop' optimizer and categorical cross entropy loss. I trained the model with a variety of batch sizes and epochs, with the performance falling within the same range. Ultimately, the best attempt was for 75 epochs with a batch size of 32, resulting in 0.90 training accuracy, 0.80 validation accuracy, 0.82 testing accuracy, and 2.01 validation loss (**Figure 7**). All the loss was not preferable, this purebred classification model was sufficient to use as a base for the mixed breed classifier.

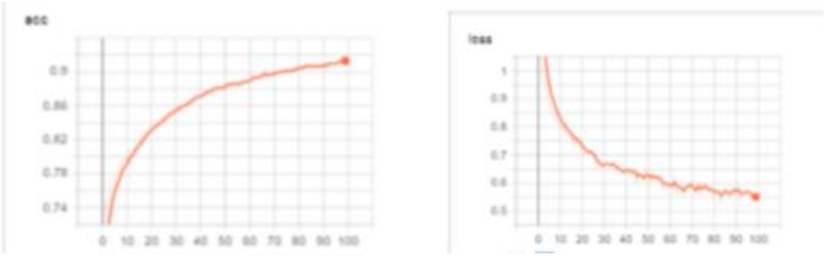

Figure 7. Training accuracy (left) and loss (right) against epochs, Model 4

4.3 Mixed Breed Classification Model

The mixed breed classification model had an identical architecture and process as in Model 4: first, a Inception-v3 network pretrained on ImageNet was used for bottleneck feature extraction, and then training and prediction commenced on a simple Sequential model. To make the model capable of predicting mixed breeds, it was necessary to convert the problem from multi-class to multi-label. This involved changing the final Dense layer's activation function to 'sigmoid', the compiling loss to binary cross entropy, and the optimizer to 'Adam'.

In the purebred classifier, the Dense layer included a 'softmax' activation function and was compiled with categorical cross entropy loss. The softmax activation function maximizes the probability of a single class for each image input. Therefore, as the probability of one class increases, the other decreases, making the class probability dependent. For the classifier to predict multiple labels, the class probabilities must be independent. The 'sigmoid' activation function accomplishes this by internally creating a model for each class (in this project, 120) and uses such to independently predict the class probabilities. By doing this, the overall problem is converted into n-binary classification, where each image will return a set of probabilities corresponding to its class participation. Binary cross entropy loss goes hand-in-hand with the 'sigmoid' activation function [10].

The performance of this model was suspiciously impressive; after observing training for 100 epochs with a batch size of 32, it was evident that the training and validation accuracy, as well as loss, hit its peak around epoch 10 (**Figure 8**). It also only less than two minutes to train on HPC, compared to the 20+ minutes of the other models. Overall, it achieved .9967 accuracy on the purebred testing set (since the set is contaminated, this metric should be discounted by approximately three percent [2]) and 0.01363 loss.

To examine the mixed breed classification results, I created a simple prediction script locally. First, Sequential model was first reconstructed and compiled with the saved training weights. Then, the list of unique breeds was used to create a mapping from the integer class labels to the common names. Then, a sample of the Petfinder image paths were loaded for testing. To be input into the model, the images must be represented by a 4D tensor, which was achieved by following the same manual bottleneck feature

extraction described for training. The output was a set of probabilities, which were indirectly sorted using NumPy's argsort to get the top two predictions. After viewing the results, I decided to add a component that would guess if the dog was mixed or purebred, determined by the threshold that the top prediction probability exceeds 0.9.

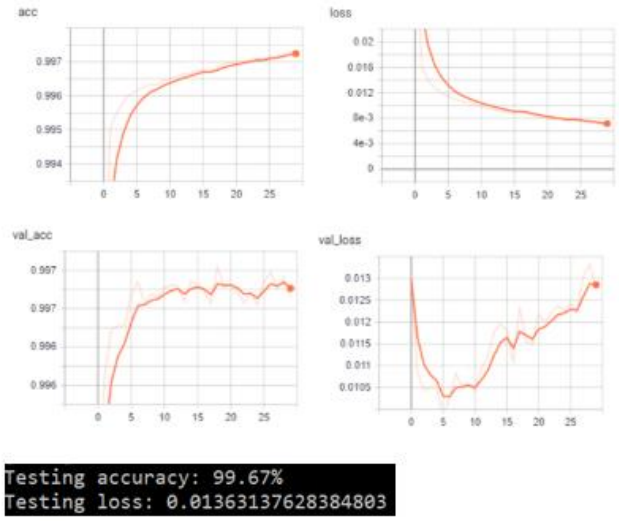

Figure 8. Collection of performance metrics, Mixed Breed Classification Model

5. Results

After building a series of models (detailed in Section 4), I decided on the Inception-v3 (Section 4.2.5/Section 4.3) for examining predictions firsthand. Due to time constraints, my results are not robust, yet still indicative of some possible issues with the model.

5.1 Purebred

I first ran several purebred images through the mixed breed model to view the results.

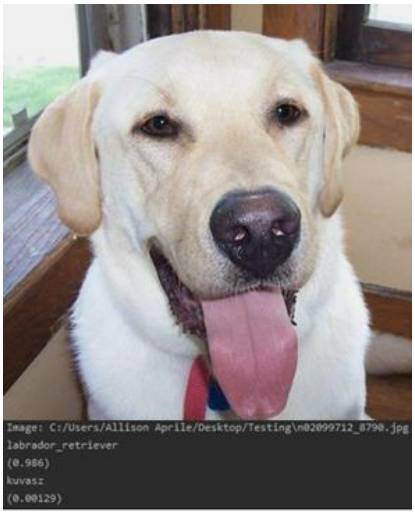

Figure 9. Labrador Retriever

Figure 10. Bernese Mountain Dog

Both images were procured from the Stanford Dogs Dataset [1]. **Figure 9** shows the result of a purebred Labrador Retriever run through the model without the purebred threshold (see Section 4.3). The model predicted the breed as Labrador Retriever with 0.986 probability, as well as Kuvasz with 0.00129 probability. Similarity, in **Figure 10** it is evident the that purebred Bernese Mountain Dog was correctly predicted with 0.949 probability, as well as Appenzeller with 0.00829 probability. This result includes the 'Likely purebred' determination of the purebred threshold. These results are sufficient but highlight a major issue with the model: purebred dogs will never be predicted with 1.0 probability.

I confirmed this by examining the results of the Stanford Dogs testing set. Although achieving .9967 testing accuracy, none of the predictions had a 0 probability any of the classes, unlike in the training data target vectors. The greater than .9 threshold was introduced in order to temporarily alleviate the issue but is not realistic. Ideally, the model should predict, not just approach, 0 probability for the breeds that do not play a dominant role in the predicted dog's visual attributes.

5.2 Mixed Breed

I then ran several mixed breed dogs through the mixed breed model to view the results.

Figure 11. Labrador Retriever/ Redbone Coonhound

Figure 12. Bloodhound/Rhodesian Ridgeback

Both images were procured from the Petfinder dataset. It is important to note that the Petfinder labels (i.e. primary and secondary breed) are not necessarily ground truth, unlike the purebred labels. They are hypothesized by the shelters. Additionally, all images scraped were flagged in the Petfinder database as mixed breed, so although some records only consist of a primary label, it implies that the shelter had difficulty pinpointing the secondary breed. For that reason, the mention of the model prediction versus the Petfinder label is strictly for comparison, especially for understanding the role of visual attributes. **Figure 11** shows a hypothesized Labrador Retriever and Redbone Coonhound mix, predicted as a Labrador Retriever and Rhodesian Ridgeback with 0.297 and 0.101 probabilities, respectively. **Figure 12** displays a Bloodhound and Rhodesian Ridgeback mix, predicted as a Rhodesian Ridgeback and Labrador Retriever mix with 0.136 and 0.0833 probabilities, respectively.

The class probabilities for the mixed breed dogs are relatively much lower; there is no dominant breed that sticks out with enormous probability in these examples and most of the other more visually mixed Petfinder samples.

5.3 Interesting Results

Figure 13. Great Pyrenees

Figure 14. Boxer

The above images were both procured from the Petfinder dataset. **Figure 13** shows one of the more commonly misclassified breeds, the Great Pyrenees. Although the model correctly predicted Great Pyrenees as a secondary breed, it had a far greater confidence in the Kuvasz, a breed very close in resemblance. Because the Kuvasz is typically slimmer with a more defined snout, the angle of this picture makes the misclassification reasonable. **Figure 14** shows a dog hypothesized by Petfinder as a Boxer, which does not appear at all in model predictions. However, the predictions of American Staffordshire Terrier and Staffordshire Bull Terrier are once again close in resemblance to the Boxer (primarily in terms of ear structure and fur texture). The misclassification is most likely due to the coloring, which is opposite of the Boxer's typical Tuxedo coat (brown body with white chest). These are instances where the Petfinder label is supported by additional background or behavioral observations.

5.4 Highlighted Issues

In the final research meeting, we discussed the above results, as well as the fast training time and high testing accuracy. My advisor suggested a variety of possible causes, including the use of different optimizers between the purebred and mixed breed models, and a mistake in the creation of the bottleneck features. He proposed examining the performance on the purebred testing dataset, looking specifically into the misclassified images. As discussed in Section 5.1, my model performance and generalizability would be improved if it could predict zero probability for the other classes, instead of having greater than 0.9 confidence in one breed with near-zero probabilities for the others.

Additionally, I should check if the model was reporting higher validation accuracy in the earlier epochs than the training accuracy. One my teammates also suggested exploring precision-recall charts and metrics.

The greatest area of concern was the use of bottleneck features. My advisor explained how in an extreme case, the bottleneck features could cause the inputs for each class to be identical. Then, the model has no way to separate classes. On another extreme, if separating the classes is very easy with a small network, it suggests that the original features were already separated. For that reason, bottleneck features are not typically used in a project such as this.

Due to time constraints, I was not able to explore all of these concerns in depth. However, I plan to look into them into the future and use the suggestions to improve my model.

6. Conclusion

6.1 Outcomes

Although I did not get to work with the Petfinder dataset as much as I would have hoped, I was very satisfied with the progress I made throughout the semester. Through this experience, I was able to learn more about web scraping and preview how my dataset could help me better tailor my model for the unique attributes of mixed breed dogs. Despite not yielding the best model, I did gain so much knowledge about neural network model architectures and areas of improvement, specifically image augmentation and feature extraction. I also had the opportunity to work extensively with Linux, allowing me to get more proficient at commands, job scripts, and general cloud operations.

When compared to the methodology and results from the April 2019 attempt, this project clearly demonstrated an improvement in my skills and understanding of the problem, and machine learning as a whole. I look forward to observing a similar, if not greater improvement, as I continue this project in the future.

6.2 Future Work

In August 2020, I will be starting a full-time position on a Data Science research team, as well as working towards a M.S. in Machine Learning. It is my hope that these experiences equip me with the skills to eventually finish this project, and perhaps propose it to Petfinder and other adoption sites for their feedback.

I will definitely consider all of the suggestions from my advisor and research team. I plan to keep the Inception Model architecture, but look more into bottleneck features and determine if they are appropriate for this problem. I may also try to supplement the model with the methods discussed by Fei-Fei and his team [2], or research new focuses of fine-

grained classification. As for the Petfinder dataset, I did extract more data than just the breed labels and plan to put it to important use. Features such as colorization, size and behavior (e.g. good with cats, children, etc.) could help better classify the animals beyond physical appearance. Ultimately, I wish to build a model that can not only achieve great performance but provide a great service to the shelters as well.

7. Acknowledgements

First, I thank Dr. Richard Souvenir for allowing me to join his research group and advising me throughout the semester. His feedback and direction were incredibly helpful, and I especially appreciate his thought-provoking discussions. I was able to approach the project independently but still felt comfortable asking for help. I also thank the other members in the research group for their advice and patience, particularly Mushin Fatih Yorulmaz for his extensive help with HPC and general Linux. Next, I thank Dr. Slobodan Vucetic for encouraging me to continue this project, as well as being supportive, following his course. Finally, I wish to acknowledge my mixed-breed dogs Owen and Suzie for inspiring this project.

References

[1] Fei-Fei, L., Jayadevaprakash, N., Khosla, A., & Yao, B. (2011). Stanford Dogs Dataset. Stanford University. http://vision.stanford.edu/aditya86/ImageNetDogs/.

[2] Arava, S.K., Yang, W., & Yuan, Y. (n.d.). Mixed Breed Dogs Classification. Academia. https://www.academia.edu/33721767/Mixed_Breed_Dogs_Classification.

[3] Marcelino, P. (2018). Transfer learning from pre-trained models. *Towards Data Science.* Medium. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

[4] Amaratunga, T. (2017). Using Bottleneck Features for Multi-Class Classification in Keras and Tensorflow. Codes of Interest. https://www.codesofinterest.com/2017/08/bottleneck-features-multi-class-classification-keras.html

[5] HPC Team. (2020). Owl's Nest. Temple University. https://www.hpc.temple.edu/owlsnest2/

[6] Fredrickson, J. (2019). Dog Breed prediction usings CNNs and transfer learning. *Towards Data Science.* Medium. https://towardsdatascience.com/dog-breed-prediction-using-cnns-and-transfer-learning-22d8ed0b16c5?gi=4e0fdd566bc6

[7] Vikram, Saksham. (2018). DOG-BREED-CLASSIFICATION-STANFORD-DOG-DATASET. GitHub repository. https://github.com/saksham789/DOG-BREED-CLASSIFICATION-STANFORD-DOG-DATASET/blob/master/vgg_model.ipynb

[8] Mason, C. (2018). Dog Breed Image Classification. Medium. https://medium.com/@claymason313/dog-breed-image-classification-1ef7dc1b1967

[9] Stancliffe, P. (2019). Udacity Dog Breed Classifier – Project Walkthrough. Medium. https://medium.com/@paul.stancliffe/udacity-dog-breed-classifier-project-walkthrough-e03c1baf5501

[10] Sharma, P. (2019). Build your First Multi-Label Image Classification Model in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/04/build-first-multi-label-image-classification-model-python/