A Computer Vision Pipeline to Match Lost and Found Dogs Together

Aidan Vickars, Anant Sunilam Awasthy, Karthik Srinatha, and Rishabh Kaushal

June 29, 2022

Contents

1 Motivation and Background

As the most common house pet, lost dogs are a frequent problem around the world. With this is mind, it stands to reason there is no shortage of interest in new techniques for finding a lost dog. Of course, there are a variety of classical methods that include posting flyers on telephone poles, posting adds on Craiglist and social media sites as well as leveraging purpose built applications like the BC SPCA's "Pet Search" [1]. However, all of these methods require creating some form of an eye catching poster or description that has been used in so many different forms that they have lost their intended affect. As a result, a new method is needed. Thus, in this paper we will present an Android application that leverages three separate convolutional neural networks to match lost and found dogs together. We do this by computing the similarity between lost and found dogs and subsequently returning the most similar matches to the user.

2 Related Work

While dog-identification is a sub-class of the heavily researched facial recognition area, it remains extremely undeveloped. However, there are two related works that we discuss here. The first is "A Deep Learning Approach for Dog Face Verification and Recognition" [2] by Guillaume Mougeit, Dewei Li and Shuai Jia. In this paper, Mougeit, Li and Jia present "VGG-like and "ResNet-like" models that encode the image of a dog and compute the Euclidean distance between the encodings to measure the similarity between two dogs. This value is used to perform face verification to determine if two dogs are the same. To quantify the accuracy of their models they generated 2500 positive pairs and 2500 negative pairs of images and applied their model on each pair. Note, positive indicates the images represent the same dog and negative indicates the images represent different dogs. Their models made the correct classification 92% and 91% of the time respectively.

The second work is "Dog Identification using Soft Biometrics and Neural Networks" [3] by Kenneth Lai, Xinyuan Tu and Svetlana Yanushkevich. In this paper, Lai, Tu and Yanushkevich present an approach to increase the accuracy in dog-identification that is also the inspiration behind the work presented here, and all credit is given with respect to the similarities between our works. Lai, Tu and Yanushkevich developed a dog detection model, a breed classification model, and a dog-identification model that work together in the order specified. The dog detection model determines the bounding box of the face of a dog and is used to crop the image accordingly. The breed classification model like other models of this type, simply determines the most likely breed(s) of the dog. Finally, the dog-identification model functions in the same way as the models created by Mougeit, Li and Jia. By first cropping the image, and filtering by the breed of the dog, the face verification accuracy is improved. However, we do not state the accuracy of the models as the dog-identification model is trained on the "Flickr-dog" [3] data-set that contains only 374 images made up of just two breeds. As a result, our findings are incomparable.

3 Problem Statement

It should be noted that in both papers discussed above, the dog-identification models are trained and tested on highly curated images that contain only the face of a dog. To be succinct, the entire body of the dog is cropped out and the remaining face is normalized by horizontally aligning the image according to the dog's eyes. Images that do not contain the dogs face are removed. This presents a deficiency in two ways. The first is that by training the model on such a highly curated data-set, the assumption is made that all applicable images contain the dog's face in a relatively front facing fashion. This is obviously not the case. Any dog owner knows convincing your dog to look at the camera is a non-trivial task and that the majority of their photos are of the dog in an appealing position with their face obscured. An example of this is shown in Figure 1 below.

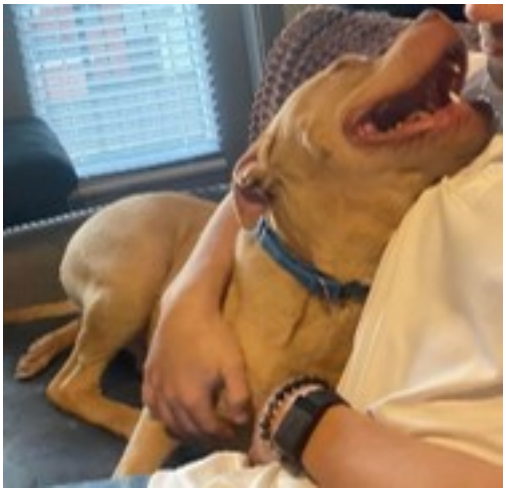

Figure 1: An Example of a Dog with their Face Obscured

The second deficiency is that by cropping the image to only the dogs face we theorize that valuable information is lost. In the case of face verification in humans, only examining the face is desirable because humans change clothes. However, dogs do not. We certainly note that there are cases where dogs do wear clothes but these cases are infrequent. Thus, we theorize that by leveraging a dogs entire body we may see an improvement in the accuracy of the dog-identification model relative to work done by Lai, Tu and Yanushkevich. This is because the model can leverage additional characteristics such as the size and shape of the dog. This is illustrated in Figures 2 and 3. In Figure 2 , the two dogs have very similar faces; one could forgive a model for classifying these dogs as the same or at least determining that they are very similar. But in Figure 3, we can clearly see that are not. By leveraging the entire body of dog, this miss-classification should be eliminated.

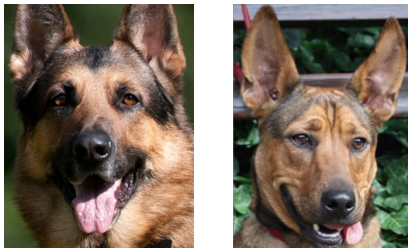

Figure 2: Two Dogs with Similar Faces

Figure 3: Two Dogs with Similar Faces but Different Bodies

Thus, we can now present the key problems this project aims to answer:

1. By leveraging the entire body of a dog, can we construct a dog-identification model that can accurately

determine if two dogs are the same or not? Furthermore, can we achieve a better accuracy than that achieved by Mougeit, Li and Jia?

2. By removing the restriction of curated front facing dogs, can we construct a pipeline that can accurately match lost and found dogs together?

4 Data Product

To answer the questions stated above, we use the work done by Mougeit, Li and Jia and use a similar VGG model to compare dogs. We also incorporate the work done by Lai, Tu and Yanushkevich and create a dog localization model and breed classification model to improve accuracy. We extend this by using the entire body of a dog instead of the face to remove possible miss classifications as explained above. In future references we refer to the these models as the Dog Comparator, the Dog Extractor and the Dog Classifier respectively. However, before elaborating on each individual model we will present the Data Product to give the reader context as to how the models work together.

To utilize this work in a production environment, we have created an Android application to act as a user interface to allow for easy image upload, and have packaged the models inside a Flask API to match lost and found dogs together. We also use an AWS S3 bucket and a Relational Database System to store images and image metadata respectively. This system is visualized below in Figure 4.

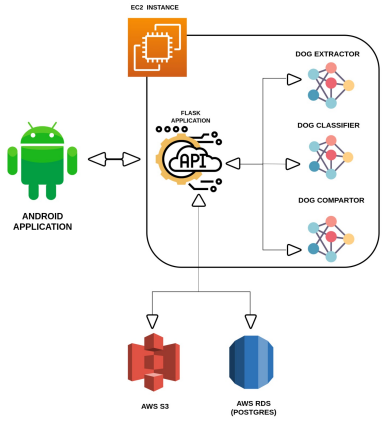

Figure 4: Dog Finder System

At a lower level if the user has lost a dog, they will submit a photo of their dog, their contact information and their location to the application. The application will then submit this information to the API. Once a submission has been made to the API, the application pipeline that is visualized below in Figure 5 is triggered. The following steps outline this pipeline:

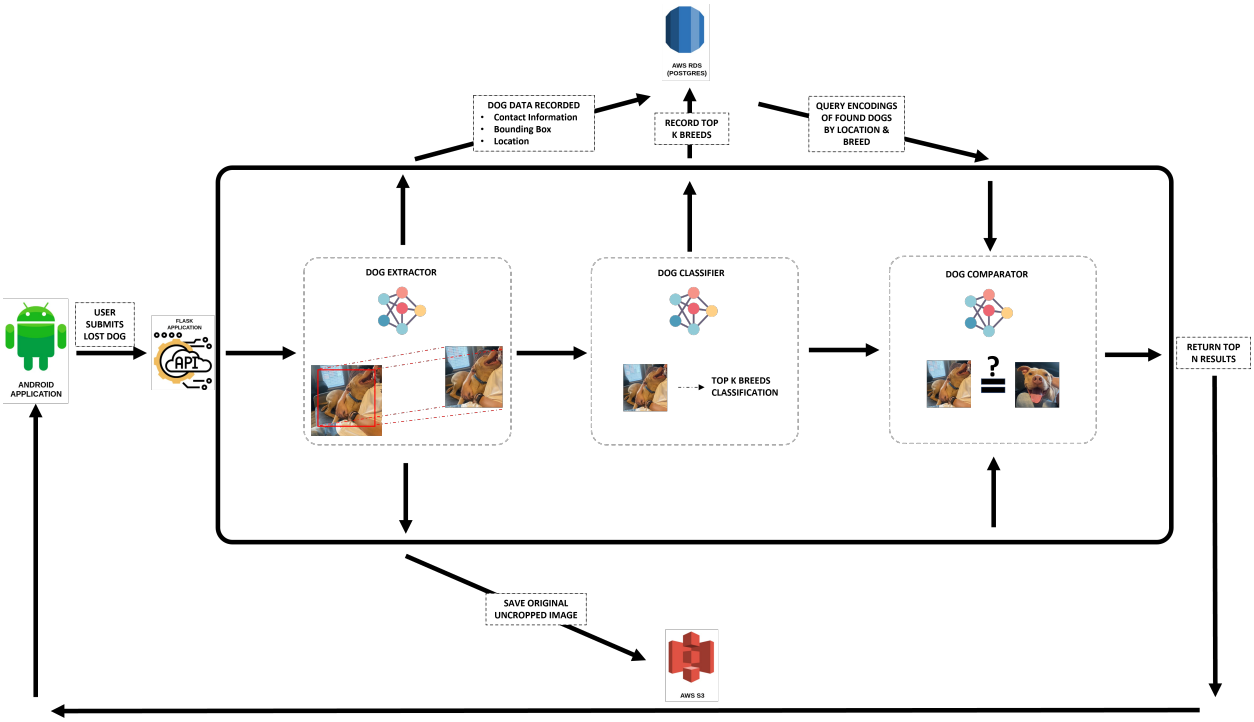

Figure 5: Application Pipeline

1. Once a lost dog has been submitted, the image is passed to the Dog Extractor model that computes the coordinates of the bounding box of the dog. This model also acts as a quality control by validating the image to ensure that the it contains a dog, and only one dog. If these conditions are not met, an error is returned to the user.

2. After validation, the original image is saved into an S3 bucket, and the related information such as the users contact information, location and the coordinates of the bounding box are inserted into a PostGre relational database system (RDS).

3. The original image is then passed into the Dog Classifier model that determines the most likely breed(s). This information is inserted into the RDS.

4. The original image is cropped using the computed bounding box coordinates, and passed into the Dog Comparator model that creates a five dimensional encoding of the image that is inserted into the RDS.

5. We then query the encodings of the dogs marked as found according to:

   (a) Dogs that have a non empty intersection between their b most likely breeds respectively, and the m most likely breeds of the lost dog.

   (b) Dogs that are within x distance from the lost dog.

   These encodings are then compared against the encoding of the lost dog by computing the euclidean distance between them. The sigmoid function is applied to the distance value to constrain it to $[0, 1]$. The n most similar dogs are returned to the user.

6. If a match is confirmed by the user, the corresponding lost and found dogs are removed from the RDS and S3 bucket. Otherwise, the lost dog is left in the system for future comparisons.

An attentive reader will notice that we discuss only the submission of a lost dog. This is done to minimize confusion. If the user submits a found dog the pipeline is identical except that the dog is instead marked as found and is compared against lost dogs. This completes the pipeline contained within the application.

From the perspective of the user, the application contains several features to augment the process of matching lost and found dogs together. For instance, in Figure: 6 the matches found for a sample lost dog submitted to the application are shown. We can see that the application does extremely well in identifying similar dogs to the point that it is difficult to identify which found dog is the lost dog (it is the right most dog).

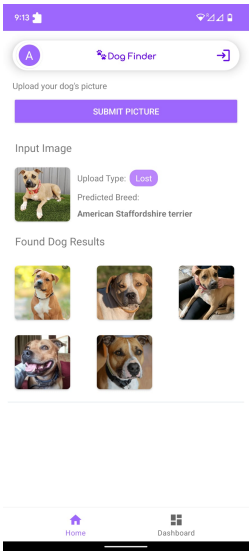

Figure 6: Results for Sample Lost Dog

A second feature in the application is a visualization that displays the location with additional information such as the distance, estimated driving time, breed and contact information of the users matches. This is shown in Figure 7.

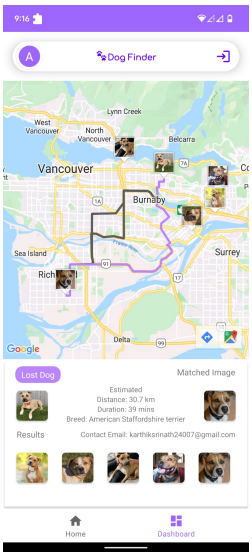

Figure 7: Additional Information contained in the Results for a Lost Dog

For a comprehensive demo of the application and all of its features, the reader is directed to the following video.

5 Data Science Pipeline

5.1 Open Images Data-set

To train each model, we leveraged three different data sets for each model respectively. For the Dog Extractor model, we utilized the "Open Images" [4] data-set that contains thousands of images of dogs with corresponding bounding boxes. While this data is already relatively clean, we discarded all grey-scale images, and converted all images to RGB format. The decision was made to discard grey-scale images because our expectation is that in the production environment of the app, the vast majority of images will be colour images. After cleaning this left 19 995 training images, 1568 validation images , and 4791 test images.

5.2 Stanford Dogs Data-set

For the Dog Classifier Model, we used the "Stanford Dogs" data-set [5] that contains 20 580 images with 120 different breeds from around the world. This data-set was built using images and annotations from "ImageNet" [6] for the task of fine-grained image categorization. We split this data-set into training, validation and test splits of approximately 80%, 10%, and 10% respectively.

5.3 Petfinder Data-set

Finally, to train the Dog Comparator model we required multiple pictures of many individual dogs where each picture contained the entire body of the dog. However, we found that there was no data-set that met these requirements. To solve this we scraped the trove of images on Petfinder.com that at the time of writing lists over 100 000 dogs for adoption across the world where almost every dog has multiple images. However, scraping the images presented a challenge because the links to every dog are dynamically generated. This meant scraping the HTML of the web page containing the grid of available dogs using Python's request package was insufficient because during download the URLs pointing to each available dog would not be included due to their dynamic creation. To solve this, we split the scraping into two parts. We first created a program in Python that uses Selenium to scrape the URLs pointing to each individual dog. Then using these URLs, we scraped and downloaded the images for every dog and also recorded additional information such as name, breed, age, size etc. This resulted in images for 9729 dogs with 0 - 6 images for every dog. Once the data was downloaded, we applied the following cleaning process on the images of every dog:

1. If the dog had 1 or less images, we discarded the dog and its image.
2. Confirmed every image was in RGB format or converted it to RGB format. Otherwise the image was discarded.
3. Passed every image into the Dog Extractor Model:
   - Verified the image contained a dog.
   - Verified the image contained only one dog.
   - Recorded the bounding box coordinates of the dog.

   If either of the conditions in the first two bullets were not met, the image was discarded.
4. If after the previous step, the Dog had 1 or less images we discarded the dog and its images.

After cleaning we were left with 8349 dogs with 2 - 6 images for every dog. The data set was then divided into a Train, Validation and Test split of approximately 80%, 5%, and 15% respectively. This gave a total of 6679 training dogs, 501 validation dogs and 1169 testing dogs respectively. We note that no additional cleaning was done according to the number of images each dog contained. This is because during the training and testing we only required that each dog had at least two images.

After applying the above cleaning steps, we further processed the data by parsing the breed of each dog. We found that while the breed was available for most dogs in the data-set, it was not standardized. For instance, the breed could be in many forms such as "German shepherd dog" and "terrier" where capitalization was not consistent and also contained inconsequential words like "dog". Furthermore, the data-set contained

mixed breeds that were in the form of "German shepherd & terrier". This meant that the breeds needed to be standardized. We did this by applying the following steps:

1. Converted all strings to lowercase.
2. Removed inconsequential words like "mixed", "breed", and "dog".
3. Split the breed into two strings using '&' as the separator to account for mixed breeds.
4. Compared the list of breeds against the standardized list of 120 breeds contained in the "Stanford Dogs Data-set" [5] by:
   (a) Computing the cross product between the two lists.
   (b) Computing the Jaccard similarity between every pair.
   (c) Removing any pairs with a similarity less than 0.75.

After completing this, we successfully standardized the breeds contained in the data-set. The reader may notice that dogs without a breed were not removed. This is because during the initial training of the Dog Comparator model as discussed below, we did not require the breed. During testing, dogs without a breed were removed when required.

6 Methodology

Now that the data-sets used have been outlined, the approaches used in each model can be discussed. For all three models, we present the architecture used and their respective results. We also perform a analysis into the strength and weaknesses of each model respectively.

6.1 Dog Extractor

To develop the Dog Extractor model, we investigated multiple avenues that included developing an original implementation of a transfer learning approach to Yolo V2 [7]. However, we found a significant limiting factor to be a lack of GPU memory. To be succinct, we trained the Dog Extractor model on an RTX 3070 with only 8 GB of memory. Because we wanted to use larger and more complex models to achieve high degrees of accuracy, our models would train very slowly due to the requirement of having a very small batch size during training due to memory limitations. This necessitated the requirement to utilize a largely pre-trained model via transfer learning and only make small adjustments with minimal amounts of additional training. To achieve this we employed transfer learning using a pre-trained Faster RCNN [8] model using PyTorch with a feature extractor trained on the COCO data-set [9] to act as the back bone of the network. We adjusted the output of the model from predicting a multitude of classes to only two and in doing so converted the model to a dog localization model. In short, the model was adjusted to predict only the background class and the dog class. In case the reader is unfamiliar with object localization models, we give a brief description of the model input and output here. The dog localization model accepts an image as input, and outputs a list of bounding box proposals and object score pairs. Each pair gives the coordinates of a bounding box surrounding a dog and the confidence that the box contains a dog respectively. This list is subsequently passed into the Non Max Suppression algorithm [10] that filters out overlapping proposals and proposals with low degrees of confidence. If the reader is unfamiliar with NMS, we present the algorithm here:

Non Max Suppression Algorithm

Input: List of bounding box proposals and object score pairs.

Output: Filtered list of bounding box proposals and object score pairs.

Algorithm:

1. Initialize empty list, lets call it list B.
2. Order the bounding boxes and object score pairs according to the object score in descending order.
3. Discard any pairs whose object score is less than the pre-defined object threshold. Bounding boxes with an object score lower than this threshold likely do not bound anything or do so poorly.
4. While there are still bounding boxes and object score pairs on the list:
    - Pop the first pair off the list and record it in list B.
    - Compute the intersection over union (IOU) between the pair just popped off the list, and all remaining pairs on the list.
    - If the resulting IOUs are greater than the predefined IOU threshold, discard the corresponding pairs. The bounding boxes of these pairs likely bound the same dog.

The model was trained on the cleaned "Open Images" data-set for 10 epochs with an initial learning rate of 0.005 and a decay of 0.1 every 3 epochs, as well as a momentum value of 0.9. Due to memory limitations, our batch size was set to one. We note that PyTorch's tutorial on object detection [11] was very helpful here and we give all credit accordingly. During training we concerned ourselves only with the validation Mean Average Precision (MAP) because the network was largely already pre-trained and only required fine-tuning. We coded and computed the MAP over an IOU threshold range from 0.5 to 0.95 in increments of 0.05. We denote this value as MAP 0.5:0.95. If the reader is unfamiliar with MAP, we outline the algorithm here:

Average Precision Algorithm

Input:

- List of bounding box proposals and object score pairs for each image in the data-set.
- List of true bounding boxes for each image in the data-set.
- IOU Threshold.

Output: Average Precision

Algorithm:

1. Order the list of bounding box and object score pairs by object score in descending order.
2. Denote every bounding box and confidence score pair as either a true positive or a false positive.

   A bounding box proposal and object score pair is a true positive with respect to the corresponding image if there exists a true bounding box such that the IOU between them is greater than the IOU threshold and the true bounding box has not already been detected by another proposed bounding box with a higher object score. Otherwise a bounding box proposal and object score pair is denoted as a false positive.
3. Compute the running precision value ($True\ Positives/(True\ Positives + False\ Positives)$) over the ordered list of bounding box proposals and object score pairs.
4. Compute the running recall value ($True\ Positives/(Total\ Number\ of\ True\ Bounding\ Boxes)$) over the ordered list of bounding box proposals and object score pairs.
5. Compute the area under the precision-recall curve. This is the average precision.

   The average precision is computed repeatedly over an IOU threshold range from 0.5 to 0.95 in increments of 0.05. The mean of the resulting values is taken to compute the MAP 0.5:0.95 value.

During training we saved the model weights only when the MAP 0.5:0.95 increased and achieved a best validation MAP 0.5:0.95 of 0.73 during training using an object score threshold of 0.6 and an IOU threshold of 0.5 in NMS. The MAP 0.5:0.95 is plotted below in Figure 8 over 10 epochs.

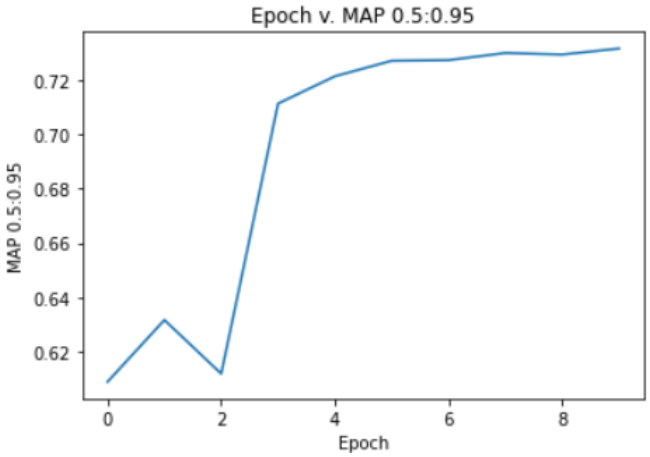

Figure 8: Epoch vs. MAP 0.5:0.95

To improve the accuracy of the model we tuned the object score and IOU thresholds used in NMS. To do this we used a brute force approach by applying NMS on the model output on the validation data over a grid of object score and IOU thresholds and then computed the MAP 0.5:0.95 on the results. We used a large grid that ranged from 0.05 to 0.95 for both thresholds in increments of 0.05. The results are visualized below in Figure 9.

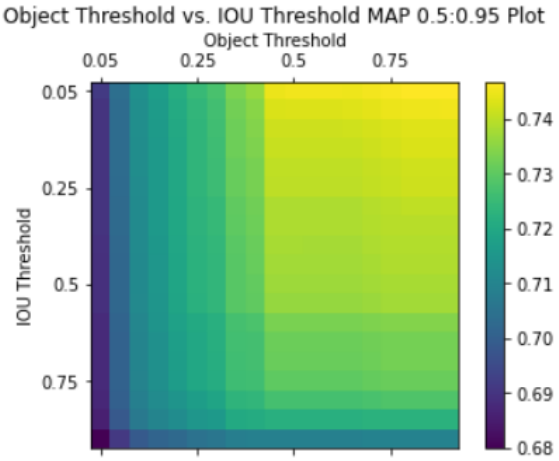

Figure 9: Object vs. IOU Threshold MAP 0.5:0.95

We achieved the maximum MAP 0.5:0.95 of 0.75 in the top right corner of the plot using an object threshold of 0.05, and an IOU threshold of 0.9. However, we considered the consequences of using such extreme thresholds. By using such a low object threshold, many more bounding box proposals would be returned by NMS regardless of how uncertain the model is that there is a dog in the bounding box. This would have increased the number of false positives the model produced. In a similar fashion, by using a strong IOU threshold of 0.9 any pair of bounding boxes must achieve an IOU greater than 0.9 to be classified as bounding the same dog. This would again increase the false positive rate. In the context of the application, both cases would significantly increase the number of errors users would receive when submitting an image. This is because every image submitted is validated to ensure it contains only one dog. To combat this, we deviated from the optimum thresholds and increased the object threshold to 0.5 and decreased the IOU threshold to 0.75. This combination achieved an MAP 0.5:0.95 of 0.74. A very small decrease of just 0.01. Finally, using our chosen thresholds, the model achieved an MAP 0.5:0.95 of 0.74 on the test data indicating the model performs very well.

Now that the parameters in NMS have been optimized we further assessed the performance of the mode

with respect to different sized bounding boxes contained in the test data. To do this we first normalized the height and width of each bounding box by dividing by the height and width of the image respectively. Then the k-means algorithm was applied using three clusters to divide the bounding boxes into small, medium and large groups. The results of the clustering is shown in Figure 10 below where we can see that the algorithm accurately divided the bounding boxes into small medium and large groups.

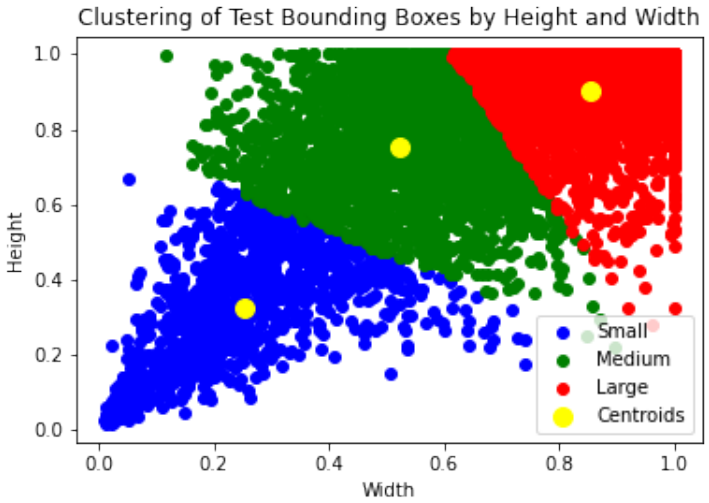

Figure 10: Results of Clustering Test Bounding Boxes by Height and Width

The Dog Extractor was then applied on the images corresponding to each group to compute the bounding box proposals. The MAP 0.05:0.95 of each group was then computed, and is shown in Figure 11 below. We can see that the model performs extremely well on medium and large dogs relative to the image, however there is a steep drop in performance on smaller dogs. This drop in performance is not surprising because object localization models typically perform significantly worse on smaller objects. Furthermore, it should be noted that the model's performance over smaller dogs is still very strong. The MAP 0.5:0.95 of 0.61 indicates the model still performs very well.

Group	**MAP 0.5:0.95**
Small	0.61
Medium	0.77
Large	0.78

Figure 11: Test MAP 0.5:0.95 by Bounding Box Size

6.2 Dog Classifier

To train a convolutional neural network for breed classification, we used a transfer learning approach by using a pre-trained model from PyTorch. To do this, we first loaded a pre-trained model and adjusted the dimension of the output layer to 120. The dimension was set to 120 to correspond with the number of breeds contained in the "Stanford Dogs" data-set [5]. To devise the optimum model, several models were trained and chosen through either random selection or according to the best ImageNet error rates for top one and top five accuracy. The pre-trained models from PyTorch [12] listed in Figure 12 with their corresponding training attributes were used as potential models for dog breed classification. As the reader can see, a variety of breed classification models were trained where the training was done on Simon Fraser University's CSIL BLU9402 workstation using an Nvidia Quadro RTX 4000 GPU card [13]. In the interest of maintaining reproducible results, we state the optimizer and number of Epochs used. Furthermore, we found that there

was a direct correlation between the dimension of the output layers of the feature extractors and the training time. This is because the dimension of the output layers strongly correlative with the size of each model.

Model	Feature Extractor Input Layer Dimension	Trained Feature Extractor	Optimizer	Max Number of Epochs	Training Time
Densenet-121	1024	No	SGD	30	31m 53s
EfficientNet-B0	1280	No	SGD	30	48m 51s
GoogLeNet	1024	No	SGD	30	18m 7s
Inception v3	2048	No	SGD	30	35m 55s
VGG 16	4096	No	SGD	30	37m 59s
VGG 19	4096	No	SGD	60	1545m 13s
ConvNeXt Large	1536	No	SGD	2	12m 11s
Regnet Y 32GF	1512	No	SGD	30	31m 5s
EfficientNet-B7	2560	Yes	SGD	30	267m 15s
Vit_b_1	768	Yes	SGD	30	172m 57s

Figure 12: Dog Classifier Model: Training Data

While a number of models were trained, to reduce repetition we will elaborate on training process of the two best models: the "ConvNeXt Large" model and the "VGG 19" model. In our initial training of the "ConvNeXt Large", we trained the model for 30 epochs. However, we found that this resulted in an extremely poor classification accuracy of 20% on the test data. The model was clearly over fitting to the training data. To remedy this, we added dropout to the last CNN layer with a probability of 0.5 in the feature extractor during training and reduced the number of epochs to just two. This resulted in a huge improvement in the test accuracy. To be succinct, the test accuracy increased from 20% to 96%; an increase of 76%. In a different facet the "VGG 19" the model was trained for a maximum of 60 epochs but the optimum weights with respect to the validation data were achieved during the sixteenth epoch. It should be noted that during the training of each model, the model weights were only saved if the validation accuracy increased. Otherwise, weights were not saved. This was done to minimize possible over-fitting.

The validation results of the models are shown in Figure 13 and have been further visualized in Figure 14. Note: the size of each bubble point in Figures 14 and 16 corresponds to the dimension of the last layer in the feature extractor of each model. The "ConvNext" model had the best validation accuracy, followed by the "VGG 19" and "inception_v3" models. Overall, the validation scores were promising but they alone cannot be trusted for model evaluation and comparisons. As a result, the models were tested on the test data set.

Model	Validation Loss	Validation Accuracy
ConvNeXt Large	0.125	0.96637
VGG 19	0.4323	0.8767
inception v3	0.3787	0.8762
Regnet Y 32GF	0.4616	0.864
VGG 16	0.5148	0.8635
Vit b 1	0.5445	0.8372
EfficientNet-B7	0.6235	0.8358
Densenet-121	0.5495	0.8289
GoogLeNet	0.6956	0.7904
EfficientNet-B0	0.92	0.7675

Figure 13: Validation Top One Accuracy Scores and Cross-Entropy log Loss

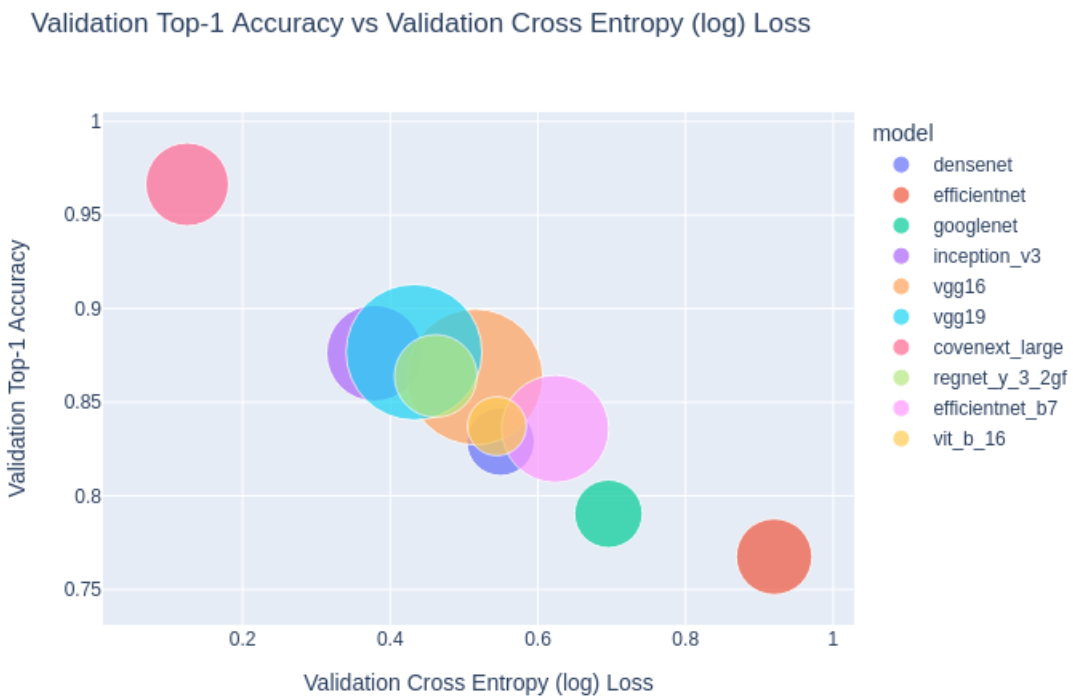

Figure 14: Dog Classifier Model - Validation Top-1 Accuracy vs. Validation Cross Entropy Loss

The test results of the models are shown in Figure 15. As we can see the "ConvNext" model easily outperformed all the other models. This is particular evident in the top one accuracy of the model where the difference in accuracy relative to the next best model is approximately 11%. This is further visualized in Figure 16 where the "ConvNext" model exists in the top left corner far away from the other models. However, because the accuracy of the "ConvNext" model was high relative to the other models we were concerned that the model may have over-fitted to the "Stanford Dogs" data-set [5] in general. To account for this, we picked the top two models: the "ConvNext" model and "VGG 19" model as the final candidates for the application.

	Testing		Macro			Weighted		
Model	**Loss**	**Accuracy**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
ConvNeXt Large	0.10765	0.96	0.97	0.967	0.967	0.97	0.969	0.968
VGG 19	0.45274	0.85	0.86	0.852	0.851	0.861	0.856	0.854
Regnet Y 32GF	0.459066	0.85	0.863	0.849	0.845	0.862	0.852	0.846
VGG 16	0.489324	0.84	0.846	0.837	0.837	0.847	0.841	0.839
Densenet-121	0.564295	0.82	0.835	0.82	0.817	0.836	0.825	0.82
EfficientNet-B7	0.652045	0.82	0.835	0.822	0.819	0.835	0.826	0.822
Vit_b_1	0.555616	0.82	0.832	0.824	0.824	0.834	0.827	0.8247
inception_v3	0.900824	0.8	0.814	0.803	0.801	0.815	0.809	0.803
GoogLeNet	0.718252	0.77	0.789	0.772	0.768	0.789	0.778	0.771
EfficientNet-B0	0.876047	0.75	0.766	0.753	0.748	0.768	0.759	0.753

Figure 15: Dog Classifier Model Testing Data Results

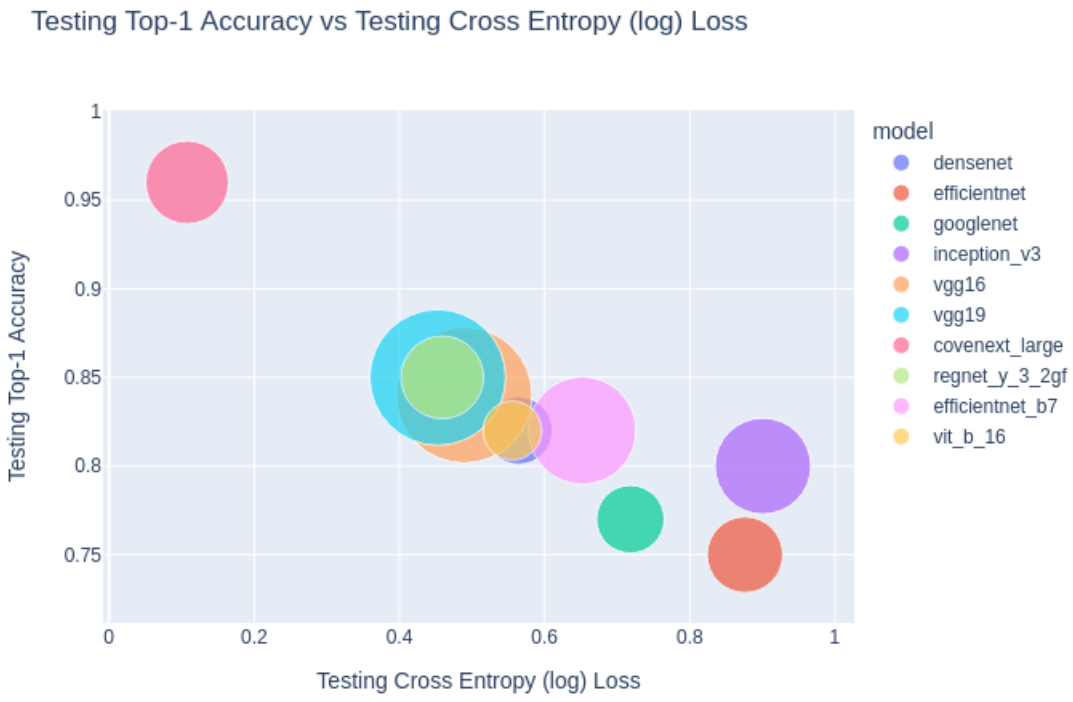

Figure 16: Dog Classifier Model - Testing Top-1 Accuracy vs. Testing Cross Entropy Loss

6.3 Dog Comparator

To create the dog comparator model that assesses the similarity between two dogs, we leveraged the pre-trained feature extractor from the VGG 19 classifier [14] and added three additional layers. This is visualized in Figure 17 below.

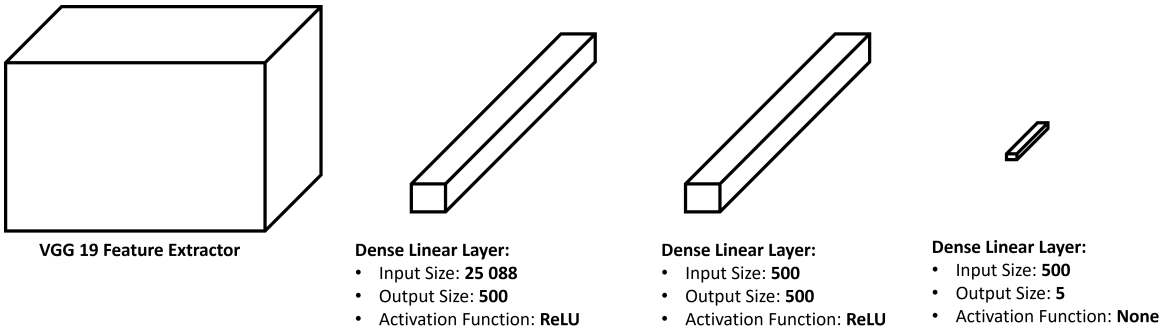

Figure 17: Dog Comparator Model Architecture

When an image is passed through the model, a five dimensional encoding of the image is produced. To compute the similarity between two dogs, the euclidean distance between their encodings is computed. The sigmoid function is then applied to constrain the value to $[0, 1]$. We call this the similarity value. If two dogs are very similar, their encodings should be relatively close together in five dimensional space. As a result, the similarity value will be close to zero. In contrast, if two dogs are very dissimilar the distance between their encodings should be large and as a result the similarity value will be close to one.

To train the model we initially employed a triplet loss function that required a batch element that contained three images. An anchor and positive image that contained different pictures of the same dog as well as a negative image that contained a different dog. To generate a single batch element we used the following process:

Single Batch Element Generation

Input: Index of a dog in the data-set

Output: Positive Image, Negative Image, and Anchor Image

Algorithm:

1. Randomly choose two different images of the indexed dog. Assign one image as the positive image and anchor image respectively.
2. Randomly select a different dog, and randomly select an image of the dog. Assign this image as the negative image.
3. Return the positive, anchor and negative images.

Each image was then passed through the model and the loss was computed. In case the reader is unfamiliar with the triplet loss function, it is defined as $L(\vec{P}, \vec{A}, \vec{N}) = ReLU(\sigma(||\vec{P} - \vec{A}||_2) - \sigma(||\vec{A} - \vec{N}||_2) + M)$ where $\vec{P}, \vec{A}$ and $\vec{N}$ are the encodings of the positive, anchor and negative images respectively, $\sigma()$ is the sigmoid function and M is the margin that was set to 0.9. Note, an attentive reader will notice the large decrease in dimension in the first dense layer of the model. This was done to accommodate GPU memory limitations so that the batch size could be increased during training. The model was trained for 41 epochs with a learning rate of 0.01 that decayed by 0.1 every four epochs and a batch size of 30. The training and validation losses are vizualized below in Figure 18 .

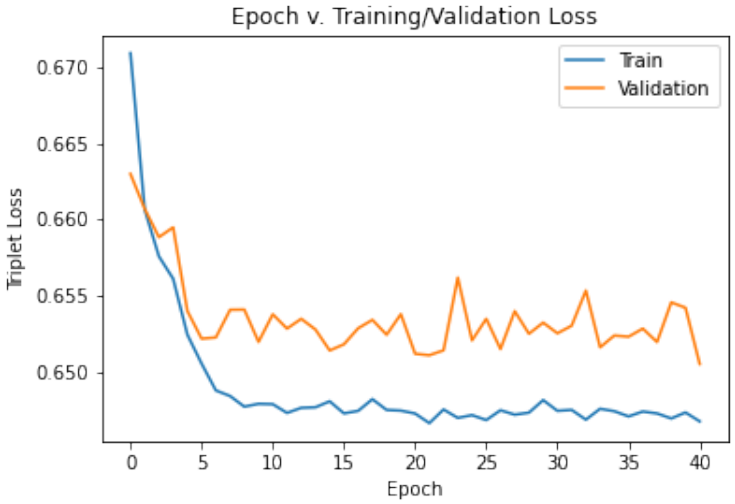

Figure 18: Dog Comparator Model Training Using a Triplet Loss Function

To assess how well the dog comparator model performed, we first determined the optimum classification threshold by computing the approximate *knee* of the ROC curve on the validation data to minimize the false positive rate and maximize the true positive rate. The ROC curve is vizualized below in Figure 19. We chose a classification threshold of 0.79.

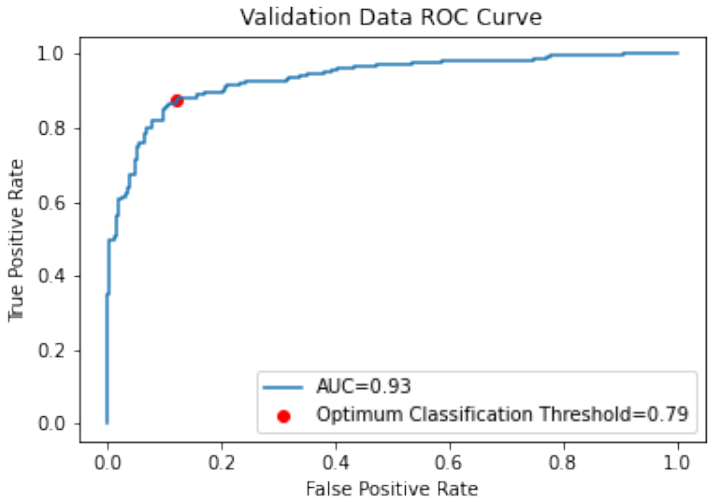

Figure 19: Validation ROC Curve Using a Triplet Loss Function

It should be noted that the reader may question why the optimum classification threshold is so high. This is due to the significant dissimilarity between all images even of the same dog in the training data. By placing no constraints on the type of images in the data-set set, all images had some degree of dissimilarity due to the variety of positions of each dog. As a result, we found that similar dogs tended to have a similarity value near or slightly above 0.5 while dissimilar dogs had a similarity value near one.

Returning to the assessment of the model, using a classification threshold of 0.79 resulted in a strong test accuracy and F1 score of 0.87 and 0.87 respectively. However, after examining the corresponding similarity values of the model we found it did not do well in grouping the comparisons between the same dogs and the comparisons between different dogs. This is visualized below in the line plot contained in Figure 20 where

we can see fairly significant overlap between both groups.

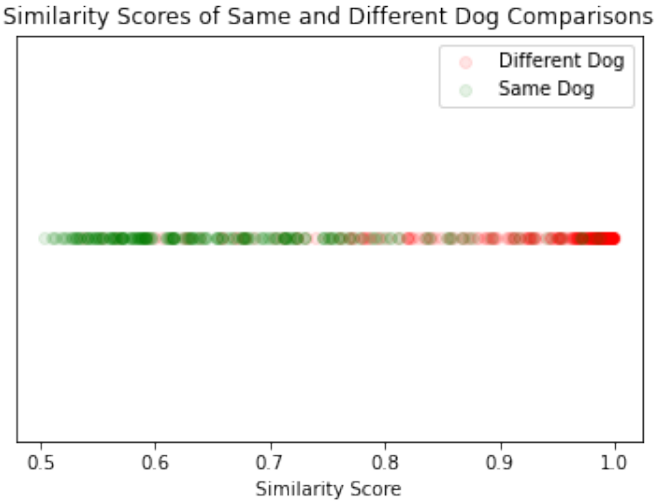

Figure 20: Line Plot of Similarity Scores Using a Triplet Loss Function

To remedy this, we retrained our model for 49 epochs with the same parameters as above using a cross entropy loss, and adjusted the individual batch elements to randomly select two images of the same dog, or two images of different dogs with equal probability. Using the same methodology, we determined the optimum classification threshold to be 0.67. This resulted in a modest increase in test accuracy and F1 score to 0.89 and 0.89 respectively. However, this significantly improved the groupings between the comparisons between the same dogs and the comparisons between different dogs. This is shown in Figure 21.

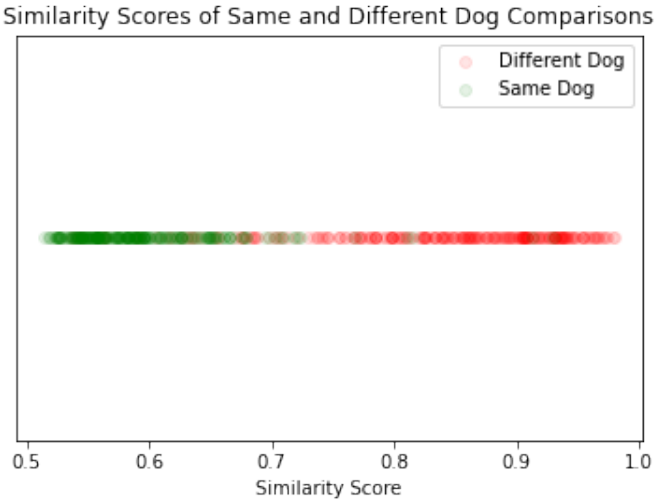

Figure 21: Line Plot of Similarity Scores Using a Cross Entropy Loss Function

We also note that the accuracy of 89% is highly comparable to the models created by Mougeit, Li and Jia that had an accuracy 91% and 92% [2]. Furthermore, while we concede that our model has a $2-3\%$ decrease in accuracy, we allow for a far more diverse set of images by not applying our model solely on front facing images of dog faces.

However, an observant read will note that by using images from two randomly selected dogs during both the training and testing processes, differing between dogs is an easy task for the model. This is because dogs

of different breeds tend to be very dissimilar and as a result telling them apart is easy. To account for this, we performed additional testing of our model by testing the performance of the model over different dogs that are known to be similar. To do this we adjusted the generation of a single batch such that when two images of different dogs are produced, the dogs are from the same breed. Thus, producing two similar dogs. The model then achieved a significantly decreased but still respectable classification accuracy and F1 score of 0.79 and 0.77 respectively on the test data. To attempt to improve this we performed additional training of the model with the added constraint that different dogs be from the same breed. However, this did not improve the model. Upon further inspection, we found a crucial flaw in the data. The data is biased towards the most popular dog breeds where the top three most populous dog breeds account for approximately 50 % of the dogs in the data-set. The frequency of the top three least and most populous breed as percentages are vizualized below in Figure 22.

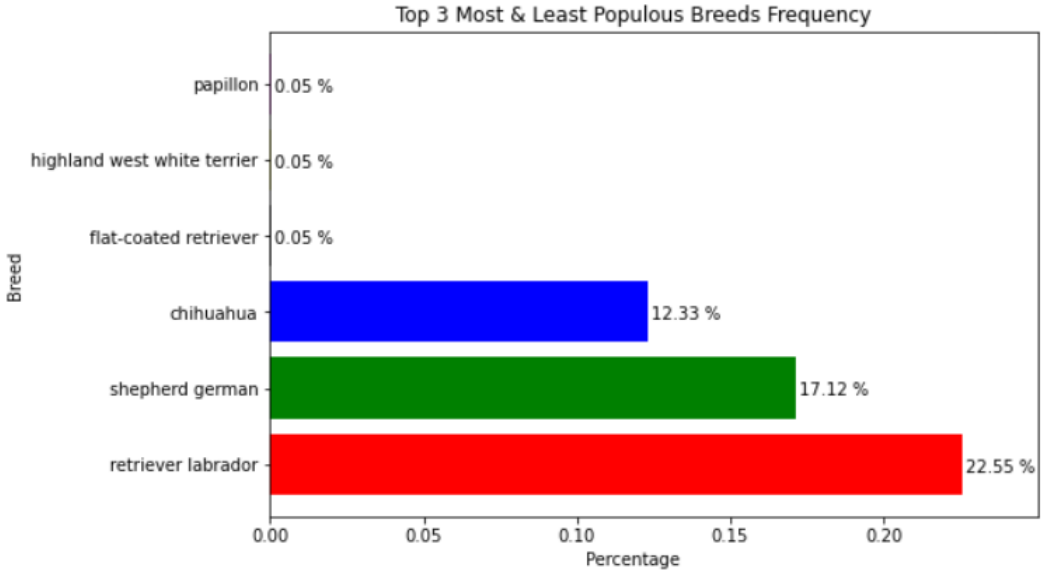

Figure 22: Top & Bottom 3 Most Populous Breed Percentage

We theorize that by performing additional training by breed we only further biased the model towards these dogs. Unfortunately, we are not able to account for this flaw in the data. This is because in other data-sets, one could augment the data by adding random samples from the underrepresented groups with an additional degree of random variation as well. However, in this case this we are unable to do this without creating two significant issues. The first, is that by adding random samples of the underrepresented breeds we risk over-fitting the model to specific dogs. We suspect this would occur because some of the underrepresented breeds have only a few instances and thus adding random samples of these breeds would require randomly sampling the same dogs many times. The second, is by adding some degree of randomness to the images of the randomly sampled dogs the images would likely be distorted significantly. It should be noted that there is a possible solution to this. We theorize that unique images of underrepresented breeds could be generated using a generative adversarial network. However, this is outside the scope of this project.

To investigate the bias of the model towards the most populous dogs, we divided the test data into two groups. The first contained dogs from the top three most populous breeds and the second contained dogs from all other breeds. We then applied the model on each group and recorded the results below in Figure 23

	F1 Score	**Classification Accuracy**
Top 3 Breeds	0.85	0.86
All Other Breeds	0.91	0.90

Figure 23: Model Accuracy & F1 Score by Breed Group

Surprisingly, the model performs better on the second group. We theorize this is because the second group contains dogs from a multitude of different breeds. As a result the similarity between every pair of different dogs is greater and thus the model can better differentiate between them. To confirm this, we performed the same experiment except we added the requirement such that when selecting a pair of different dogs, the dogs must be from the same breed. The results are shown in Figure 24.

	F1 Score	**Classification Accuracy**
Top 3 Breeds	0.83	0.84
All Other Breeds	0.74	0.78

Figure 24: Model Accuracy & F1 Score by Breed Group Selection within Breeds

There is a clear decrease in model performance between the top three most populous breeds and all others. Thus, the model does exhibit some bias towards the most populous breeds. However, the difference is reasonably small at approximately 9 % with respect to the $F1$ score.

7 Evaluation

To evaluate the accuracy of the data product, we devised an experiment to assess how well the three models worked together to match lost and found dogs. To do this, we first took the entire validation split from the Petfinder data-set and designated every dog as lost. At the same time, we randomly designated 10 % of these dogs as found. For every lost dog we randomly selected an image to use as input and for every found dog we randomly selected a different image to use as input as well. All three models were then applied over a grid of parameters to determine the optimum parameter combination to maximize the success rate of matching lost and found dogs. To be succinct, we varied the number of the most likely breeds recorded for lost and found dogs that are used to reduce the number of comparisons over a range from one to ten respectively. In other words, we varied the number of breed combinations that should be searched to find the lost dog. We also varied the number of results returned to the user from one to fifteen. For every combination we determined the success rate of matching lost and found dogs. A success was defined as the found dog being among the lost dogs returned to the user. It is assumed that if the found dog was among the lost dogs returned to the user, the user would be able to identify the correct dog.

Note, in the following experiment the second model in the pipeline the Dog Classifier uses the "VGG 19" model as discussed in the Dog Classifier section. We found that by searching in only the top one breeds for both lost and found dogs, as well as returning the top 15 matches to the users resulted in the highest success rate of 98% on the validation data. Unsurprisingly, the success rate improved linearly with the number of matches returned with the user. Surprisingly however, the highest success rates were achieved by searching in the fewest amount of breed combinations between lost and found dogs. This clearly indicates the positive impact the Dog Classifier has acting as a filter for the Dog Comparator. This is vizualized below in Figure 25 where the the number of breeds used to find matches for both lost and found dogs have been multiplied together into a single number to give the number of combinations searched. This is shown along the x-axis. Along the y-axis, the average accuracy is shown. We can clearly see that as the number of combinations searched increases, the accuracy decreases.

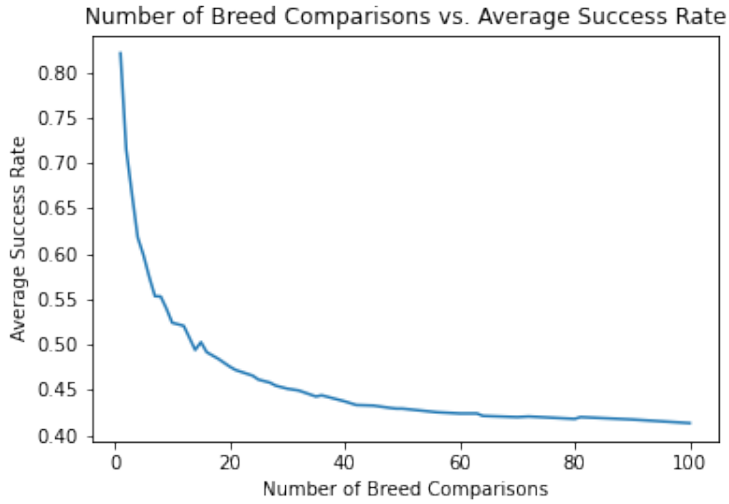

Figure 25: Number of Breed Comparisons vs. Average Success Rate

Using the optimum parameter combination as determined over the validation split of using only the most likely breed for both lost and found dogs respectively and returning the top 15 matches to the user, we achieved a success rate of 89% on the test split. We do note the decreased success rate over the test split compared to the validation split is due to the significantly increased size of the test split. The test split is approximately two times the size of the validation split and as a result the difficulty of finding the found dog among the lost dogs is significantly increased. However, after achieving a final success rate of 89% we suspect that the true success rate is actually significantly higher in the real world. This is because of two reasons. The first is that in this experiment, we are applying this experiment using a single image for both lost and found dogs. In reality, we expect users would submit multiple images in either case and thus the success rate would be increased significantly. Unfortunately, we are unable to account for this because many dogs in the data-set have only two images. The second is that in this experiment we are not filtering by location. By filtering by location before applying the Dog Comparator, the number of false matches would be significantly reduced. Thus, again significantly increasing the success rate.

The above experiment was performed again using "ConvNeXt Large" model as the Dog Classifier as discussed in the Dog Classifier section. We found the experiment gave identical results with the notable exception that the validation and test splits gave success rates of 87% and 92% respectively using the same optimum parameter combination as above. We suspect the reason for this decrease is the likely over-fitting of the "ConvNeXt Large" model to the "Stanford Dogs" data-set stanforddogs. As a result, the "ConvNeXt Large" model does not generalize to the PetFinder data-set as well as the "VGG 19" model.

8 Lessons Learnt

From this project we learned several new skills that can be broken down into a team dynamics category as well as a technical skills category. With respect to team dynamics, we discovered the benefits of splitting the work into independent parts. Where the parts were distributed among the team members such that each member owned, and was responsible for their assigned parts. This benefited the group in two ways. The first was that it ensured every member had a well defined definition of what their role in the project was and what their definition of done was. The second was that by splitting the work into independent parts, the development process was significantly enhanced. Because every part was relatively independent from the others, each member was able to work according to their own schedule and pace without having to wait or speed up for other team members.

With respect learnt technical skills, we acquired several and discuss three here. For instance, the most notable example of a skill we acquired was the deployment of machine learning models in a cloud environment.

After some experimentation we found that packaging them into a Flask API was the most convenient as it allowed us to maintain a smaller number of resources in the cloud. In an environment with greater load, we certainly realize we would likely deploy the models to their own endpoints to reduce latency. A second example of a technical skill we learned was how to accommodate the processing time models can require. In our application we take advantage of users not requiring immediate matches from the app by using larger and more accurate models. However, this resulted in the slower processing times when users submitted lost or found dogs. To accommodate this we implemented Celery that allowed the application to continue running seamlessly from the perspective of the user rather than waiting for API. Celery continues to wait for a response from the API in the background and sends the notification to the user when the task is completed. Finally, a third example of a skill we learnt in this project was less a new skill but rather a refinement. To be precise, this project allowed us to gain additional experience in model experimentation by assessing the strengths and weaknesses of models. We did this by not only trying a multitude of models to determine which performed the best, but also by delving deeper into the models performance on subsets of the data. For example, in the Dog Comparator we assessed the model both at a high level with respect to accuracy but we also delved deeper and assessed the accuracy of the model over similar dogs.

9 Summary

In this project we developed an Android application where users will submit an image of their lost dog and the most similar lost dogs that have been found will be returned. Similarly, users that find a lost dog will submit an image and the most similar dogs that have been lost will be returned. For a comprehensive demo of the application and all of its features, the reader is directed to the following video. To accurately match lost and found dogs, the app uses three convolutional neural networks that work together. The first model that is denoted as the Dog Extractor computes the bounding box coordinates of the dog contained in every image submitted to the application that are used to crop the images accordingly. The second model that is denoted as the Dog Classifier computes the most likely breed of every dog submitted to the application. This is done to reduce the number of comparisons made using the third model by ensuring only dogs from the same breed are compared. Finally, the third model that is denoted as the Dog Comparator is used to create a similarity score between two dogs. This is done by passing the cropped images of each dog into the Dog Comparator model that creates a five dimensional encoding of each image. The euclidean distance between the encodings of each image is computed and the sigmoid function is applied to constrain the values between zero and one. This creates the similarity value where values near one indicate the dogs are very dissimilar and values near zero indicate the dogs are very similar.

To assess how well the application matches lost and found dogs, we assessed each model individually as well as together. The Dog Extractor achieved a mean average precision score (MAP) of 0.74 when evaluated over intersection over union thresholds of 0.5 to 0.95 in increments of 0.05. The Dog Classifier achieved a top one classification accuracy of 85%. While the Dog Comparator achieved a classification accuracy of 89% when determining if two dogs are the same or different. To test how well all three models worked together, we designated the entire test data set (approximately 1000 dogs) as lost. At the same time, we randomly designated 10 % of these dogs as found. For every lost dog we randomly selected an image to use as input and for every found dog we randomly selected a different image to use as input as well. A success was defined as the found dog being among the top 15 lost dogs returned to the user. We found that the found dog was among the top 15 lost dogs 89% of the time. This indicated that together the models accurately matched lost and found dogs together. Furthermore, built into the application is the ability to filter comparisons by location that only improves accuracy.

References

[1] BCSPCA pet search. http://www.bcpetsearch.com/.

[2] Guillaume Mougeot, Dewei Li, and Shuai Jia. A deep learning approach for dog face verification and recognition. In *PRICAI 2019: Trends in Artificial Intelligence*, volume 11672 of *Lecture Notes in Computer Science*, pages 418–430, Cham, 2019. Springer International Publishing.

[3] Kenneth Lai, Xinyuan Tu, and Svetlana Yanushkevich. Dog identification using soft biometrics and neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[4] Open images dataset v6. https://storage.googleapis.com/openimages/web/index.html.

[5] Stanford dogs data-set. http://vision.stanford.edu/aditya86/ImageNetDogs/.

[6] Imagenet. https://image-net.org/.

[7] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. 2016.

[8] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[9] Coco data-set. https://cocodataset.org/#home.

[10] Non max suppression. https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.

[11] Torchvision object detection fine-tuning tutorial. https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html.

[12] Models and pre-trained weights. https://pytorch.org/vision/stable/models.html#id59.

[13] Sfu blu9402 workstation. https://www.sfu.ca/computing/about/support/csil/change-log.html.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.