# Capstone Project

Machine Learning Engineer Nanodegree

Akash Kumar

September 17, 2020

---

# Definition

## Project Overview

As of today, images are a very important aspect of everyday human life. With smartphones and digital cameras, humans capture millions of pictures each day. It is predicted that 1.4 trillion images will be captured in the year 2020[1], and this number is only expected to increase. This is an enormous amount of data being produced through the electronic medium. A lot of information is conveyed using images, from a line chart of stock portfolio investments to paintings.

It is easy for humans to interpret the meaning from a picture, its content, and the information it conveys. But to a computer, images are just big matrices of pixels, without any idea of what meaning it holds. In order to make artificially intelligent systems, it is important that computers be able to extract information from images. A lot of classical image processing algorithms exist, which leverage edge detection techniques to segment images, but it's really with the rise of deep learning and the 2012 ImageNet challenge, that it has bloomed.

In this project, we demonstrate the use of Convolutional Neural Networks, using one of the ImageNet competition-winning deep learning models to demonstrate an example image classification task. We take the following definition of image classification from the book, Deep Learning for Computer Vision with Python by Dr. Adrian Rosebrock,

[1] "How Many Photos Will Be Taken in 2020? - Life In Focus." 10 Jan. 2020, https://focus.mylio.com/tech-today/how-many-photos-will-be-taken-in-2020. Accessed 13 Sep. 2020.

"Image classification is the task of using computer vision and machine learning algorithms to extract meaning from an image."[2]

In our project, we solve the problem of "classifying dog breed", where we use pre-trained weights from a ResNet-152 model, which was trained on a very huge dataset, and train our own fully connected network to be able to classify input images into their correct dog breeds. The data to be used for training our model is provided by Udacity, and is explained in more detail in the section "Data Exploration". The next section describes the problem statement in more detail.

# Problem Statement

In this project, we build a "Dog Breed Classifier". As the name suggests, in the completed project, a user will be able to supply an input image, and our model will suggest a dog-breed for the image. When taking the image input, the following cases will be dealt with:

Case I: The input image is of a dog

This is the positive scenario, where the user will input an image of a dog. In this case, our model will predict the breed of the dog and display it to the user.

Case II: The input image is of a human

If the input image is of a human instead of a dog, we still feed the image into our model and get the predicted dog-breed returned by the model. Then, we display the predicted breed as the closest matching breed of the input human face.

Case III: The input image is neither of a dog or a human

If the image is neither of a dog or a human, then we simply convey the same as our output.

The following approach has been used to achieve the above-stated goal:

1. The data provided by Udacity was downloaded and imported to the project environment. It was analyzed to some extent, to understand its size and distribution. It was confirmed that the dataset has enough data for the given task.
2. Next, we built a human face detector to be able to detect if the input image contains a human face.

[2] "Deep Learning for Computer Vision with Python: Master Deep ...." https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/. Accessed 13 Sep. 2020.

3. We also built a dog image detector to be able to detect if there are any dogs in the input image. When given an image, it would simply tell us if there is a dog in the image.
4. Once the basic detectors were made, the main dog classification algorithm was built using transfer learning from a pre-trained model. It was made sure that the algorithm had an acceptable amount of accuracy for being used in the final app. Given the complexity of the task, the benchmark accuracy was set to 60%.
5. Finally, the dog detector, human detector, and the dog breed classifier were used together to create our final app. The flow of our final app is discussed below.

The algorithmic details of the human detector, dog detector, and dog breed classifier have been discussed later in the section "Algorithms and Techniques". Below, we discuss the flow of our final app:

1. The user will input an image of their choice.
2. The image will be preprocessed, by cropping to 224×224 pixels and normalizing.
3. The image will then be fed to our dog-detector. If a dog is detected in the image, we'll output its predicted breed from the dog breed classifier model.
4. If a dog is not detected in the image, then the image will be tested for the presence of a human face using our human face detector. If a human face is detected, the predicted dog breed will be output as the closest matching breed of the input human face.
5. If the image neither contains a dog or a human, then we output a message that suggests the presence of aliens in the image, to promote humor and make the app more interesting.

# Metrics

The metric that we use to measure how well our model is performing in classifying the dog breed of images is "accuracy". Accuracy is the percentage of total items classified correctly by our model. It can be given as

$$accuracy\ \% = \frac{total\ number\ of\ correct\ predictions}{total\ number\ of\ images}$$

The accuracy would be an appropriate metric for judging our model, as it provides a solid measure of how well our model performs and produces a number bound between 0 and 100.

The greater the accuracy, the better would be our model in classifying dog images. Improving the accuracy can also be thought of as minimizing our loss metric, which is log loss. The log loss for multiclass classification[3] is given as

$$-\sum_{c=1}^{M} y_{o,c}\ log(p_{o,c})$$

where M = number of classes (133 in our case),
log = the natural log,
y = binary indicator (0 or 1) if class label c is the correct classification for observation o,
p = predicted probability observation o is of class c

The log loss error takes into account how different the predicted probability, say 0.6, is from the actual probability, which is 0 or 1. This is directly usable in our problem.

# Analysis

## Data Exploration and Visualization

In this section, we analyze the data that we use for the problem. As mentioned before, the two datasets, of human images and dog images, have been provided by Udacity. Both of the datasets have been discussed below.

Fig 1. Sample images from the human dataset and the dog dataset.

[3] "Loss Functions - ML cheatsheet - Read the Docs." https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. Accessed 13 Sep. 2020.

Figure 2. The directory structure of  train, test, and valid folders in the dog images dataset

|-test
|---001.Affenpinscher
|---002.Afghan_hound
|---003.Airedale_terrier
|---004.Akita
|---005.Alaskan_malamute
|---006.American_eskimo_dog
|---007.American_foxhound
|---008.American_staffordshire_terrier
|---009.American_water_spaniel
|---010.Anatolian_shepherd_dog
|---011.Australian_cattle_dog
|---012.Australian_shepherd
|---013.Australian_terrier
|---014.Basenji
|---015.Basset_hound
|---016.Beagle
|---017.Bearded_collie
|---018.Beauceron
|---019.Bedlington_terrier
|---020.Belgian_malinois
|---021.Belgian_sheepdog
|---022.Belgian_tervuren
|---023.Bernese_mountain_dog
|---024.Bichon_frise
|---025.Black_and_tan_coonhound
|---026.Black_russian_terrier
|---027.Bloodhound
|---028.Bluetick_coonhound
|---029.Border_collie
|---030.Border_terrier
|---031.Borzoi
|---032.Boston_terrier
|---033.Bouvier_des_flandres
|---034.Boxer
|---035.Boykin_spaniel
|---036.Briard
|---037.Brittany
|---038.Brussels_griffon
|---039.Bull_terrier
|---040.Bulldog
|---041.Bullmastiff
|---042.Cairn_terrier
|---043.Canaan_dog
|---044.Cane_corso
|---045.Cardigan_welsh_corgi
|---046.Cavalier_king_charles_spaniel
|---047.Chesapeake_bay_retriever
|---048.Chihuahua
|---049.Chinese_crested
|---050.Chinese_shar-pei
|---051.Chow_chow
|---052.Clumber_spaniel
|---053.Cocker_spaniel
|---054.Collie
|---055.Curly-coated_retriever
|---056.Dachshund
|---057.Dalmatian
|---058.Dandie_dinmont_terrier
|---059.Doberman_pinscher
|---060.Dogue_de_bordeaux
|---061.English_cocker_spaniel
|---062.English_setter
|---063.English_springer_spaniel
|---064.English_toy_spaniel
|---065.Entlebucher_mountain_dog
|---066.Field_spaniel
|---067.Finnish_spitz
|---068.Flat-coated_retriever
|---069.French_bulldog
|---070.German_pinscher
|---071.German_shepherd_dog
|---072.German_shorthaired_pointer
|---073.German_wirehaired_pointer
|---074.Giant_schnauzer
|---075.Glen_of_imaal_terrier
|---076.Golden_retriever
|---077.Gordon_setter
|---078.Great_dane
|---079.Great_pyrenees
|---080.Greater_swiss_mountain_dog
|---081.Greyhound
|---082.Havanese
|---083.Ibizan_hound
|---084.Icelandic_sheepdog
|---085.Irish_red_and_white_setter
|---086.Irish_setter
|---087.Irish_terrier
|---088.Irish_water_spaniel
|---089.Irish_wolfhound
|---090.Italian_greyhound
|---091.Japanese_chin
|---092.Keeshond
|---093.Kerry_blue_terrier
|---094.Komondor
|---095.Kuvasz
|---096.Labrador_retriever
|---097.Lakeland_terrier
|---098.Leonberger
|---099.Lhasa_apso
|---100.Lowchen
|---101.Maltese
|---102.Manchester_terrier
|---103.Mastiff
|---104.Miniature_schnauzer
|---105.Neapolitan_mastiff
|---106.Newfoundland
|---107.Norfolk_terrier
|---108.Norwegian_buhund
|---109.Norwegian_elkhound
|---110.Norwegian_lundehund
|---111.Norwich_terrier
|---112.Nova_scotia_duck_tolling_retriever
|---113.Old_english_sheepdog
|---114.Otterhound
|---115.Papillon
|---116.Parson_russell_terrier
|---117.Pekingese
|---118.Pembroke_welsh_corgi
|---119.Petit_basset_griffon_vendeen
|---120.Pharaoh_hound
|---121.Plott
|---122.Pointer
|---123.Pomeranian
|---124.Poodle
|---125.Portuguese_water_dog
|---126.Saint_bernard
|---127.Silky_terrier
|---128.Smooth_fox_terrier
|---129.Tibetan_mastiff
|---130.Welsh_springer_spaniel
|---131.Wirehaired_pointing_griffon
|---132.Xoloitzcuintli
|---133.Yorkshire_terrier

## The Dog Images Dataset

The dog images dataset contains a total of 8351 images of dogs. These images have been split into training, testing, and validation images with a split of 80% training data, 10% testing data, and 10% validation data. Thus, there are 6680 training images, 836 testing images, and 835 validation images.

The train, test, and validation images have been stored in separate folders, namely, “train”, “test”, and “valid”. Figure 2 shows the directory structure of the followed by these folders. As can be seen, there are 133 folders, corresponding to our 133 output dog-breed classes, and each of these folders contain images of the specific breed. On analysis of the data, it was found that the number of images in each folder is different, thus making the training set highly imbalanced. A histogram of the counts has been shown in Figure 3.

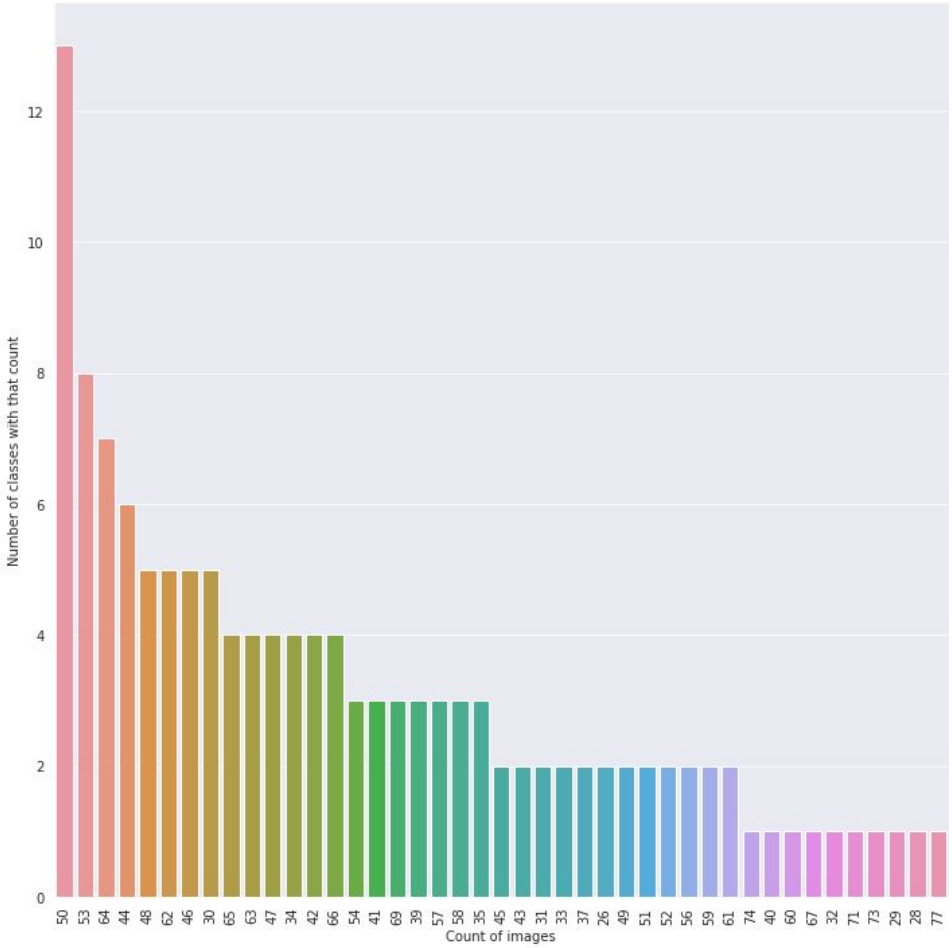


Fig 3. A plot showing a histogram of the counts of images in different classes in the training dataset

From figure 2 it is clear that there is a varying amount of images for different classes, from 26 to 77 classes. On average, most of the classes have 50 images each. Such an imbalance exists for the train, test as well as the valid datasets. It should also be noted that the resolution of each image is also not the same.

## The Human Images Dataset

The structure of the dataset containing human images is different from that of the dog images. The human dataset contains images of 5749 people. These have been organized into 5749 folders, where each folder contains a varying amount of pictures. There is no splitting into training or testing sets.

A look into the number of images in each folder reveals that the number the dataset is highly imbalanced, and the count of images varies highly from person to person.

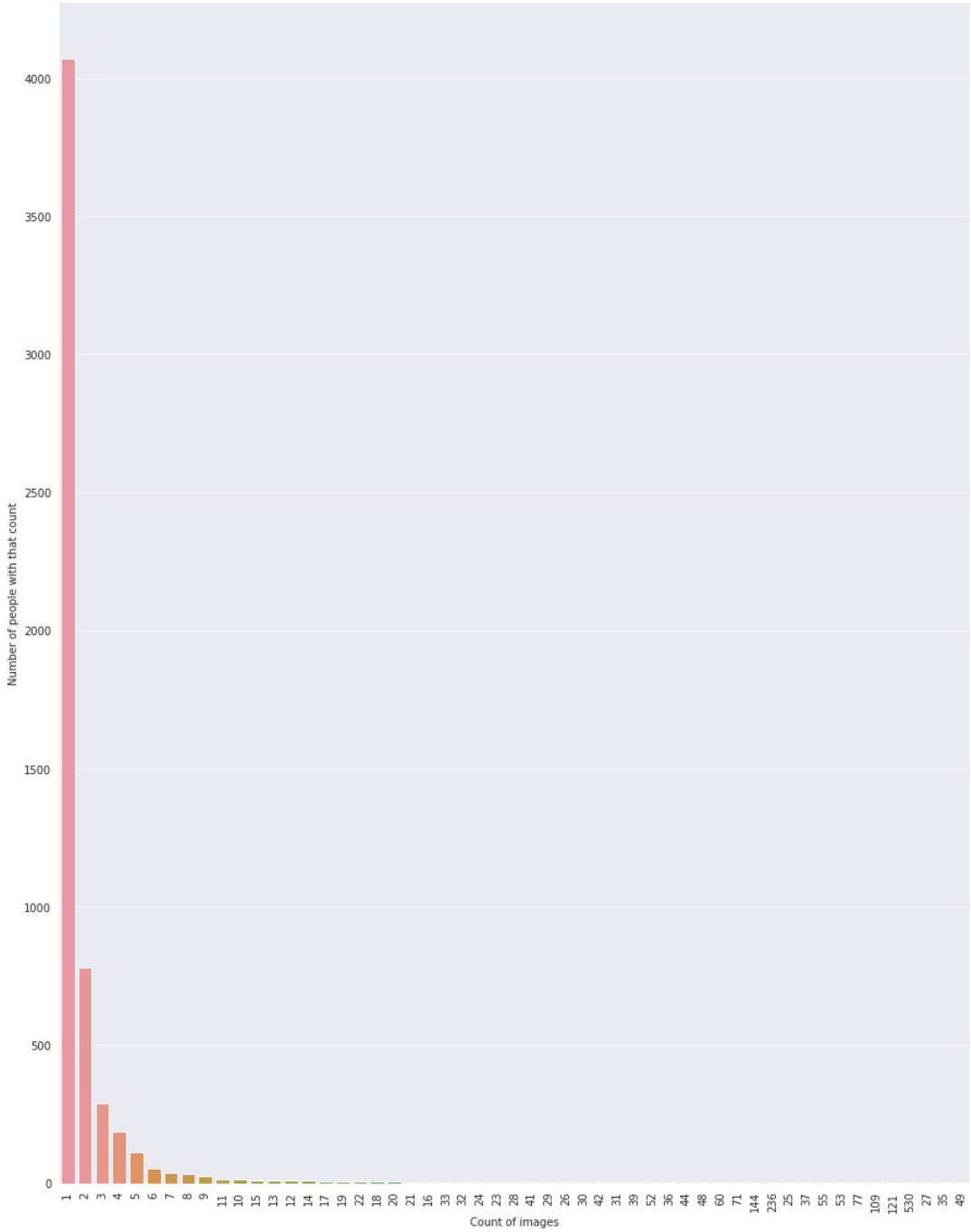


Fig 4.
A plot showing a histogram of the counts of images for different humans in the human dataset.

From the plot, we can observe that the majority of people have only one image. To better understand the distribution of the later classes, which are not visible in figure 4, have been plotted below in figure 5.

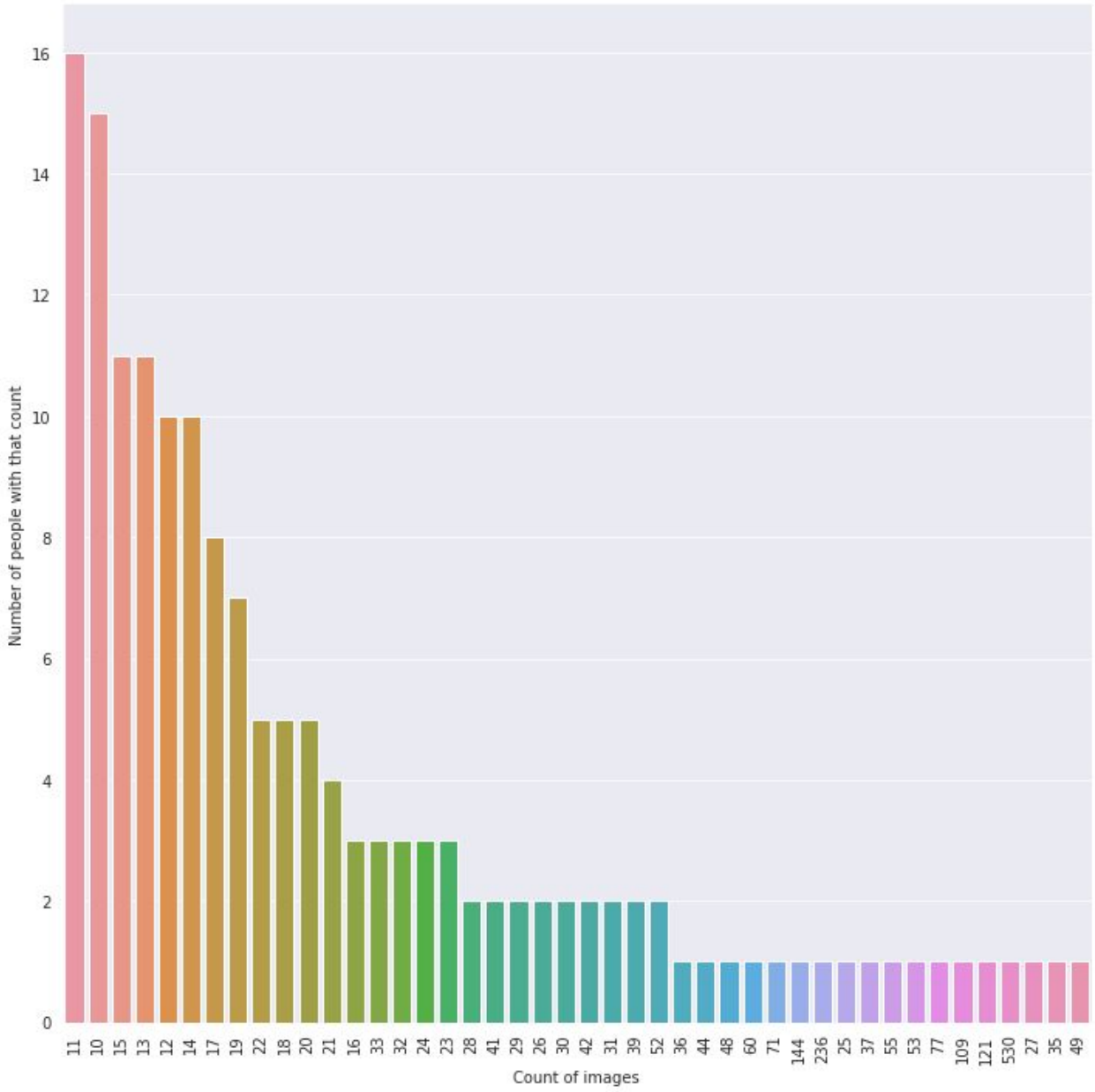


Fig 5. A plot showing a histogram of the counts of images for different humans in the human dataset excluding the counts 1 to 9

From figure 5, we observe that many people have a very large number of images. The largest of such examples is 530 images, which are of George W Bush. Thus, it is clear that this is a highly imbalanced dataset.

# Algorithms and Techniques

This section discusses the algorithms and techniques used to achieve the goals of our project. Once again, we divide it into three sections, for detecting humans, detecting dogs, and detecting dog breeds.

1. For detecting humans, we use OpenCV's implementation of Haar feature-based cascade classifiers[4]. OpenCV provides many pre-trained face detectors, stored as XML files on GitHub. We use the frontface_default XML. The only input parameter to the Haar classifier is the XML file containing the pre-trained face detector.
2. For detecting dogs, we use a pre-trained model, the VGG 16[5] model, which has been trained on the ImageNet dataset, a very large, very popular dataset used for image classification and other vision tasks. We feed our input image into the pre-trained VGG-16 model and get numeric output, which corresponds to the class of the predicted output. If we see the classes corresponding to these indices[6], we will find that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from '*Chihuahua*' to '*Mexican hairless*'. Thus, if the output of the VGG-16 model lies within this range, we can assume that the image has a dog in it. The input to our VGG-16 model is a 224×224 normalized image tensor. Since we are using the pre-trained weights of the model, there is no training step involved.
3. Our main goal is to detect the breed of the dog. This is not a trivial task, as even humans would err in distinguishing between some breed of dogs, which is described in figure 6. The pre-trained models used for the ImageNet challenge were trained on a very large dataset, and hence their convolutional layers have significant information about the image, such as the different edges, orientational variations, etc. This information that the model has learned can be applied to our problem too. Thus, we make use of the pre-trained weights from the convolutional layer of the model and retrain our own fully connected layer added to it, which will be used to produce a probabilistic output of the

[4] "Cascade Classifier - OpenCV." https://docs.opencv.org/4.3.0/db/d28/tutorial_cascade_classifier.html. Accessed 15 Sep. 2020.

[5] "Very Deep Convolutional Networks for Large-Scale Image ...." 4 Sep. 2014, https://arxiv.org/abs/1409.1556. Accessed 15 Sep. 2020.

[6] "text: imagenet 1000 class idx to human readable labels (Fox ...." https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a. Accessed 15 Sep. 2020.

133 output classes as per our use case scenario. The model we use here is the ResNet-152[7] model, because of its high accuracy. The parameters for this model are

a. The training parameters, which include
    i. The number of epochs, which is the number of iterations that our model trains over
    ii. Batch size, which is the number of images processed by the model in one iteration of the training step
    iii. The learning rate, which is the factor by which the size of each step is determined while searching for the minimum loss.
    iv. Dropout, which is the probability of the number of nodes which are randomly ignored to prevent overfitting
b. Network parameters
    i. The total number of layers in the fully connected part of the network
    ii. The total number of nodes in each hidden layer
    iii. Since the convolutional layers are already trained, there are no parameters that we need to configure for the convolutional layers as such.

To train the model, we make use of the Stochastic Gradient Descent[8] algorithm.


Fig 6. An example of two different dog breeds which look very similar. The left dog is a Brittany, which the dog on right is a Welsh Springer Spaniel. Even humans would have trouble distinguishing between these two breeds.

[7] "Deep Residual Learning for Image Recognition." 10 Dec. 2015, https://arxiv.org/abs/1512.03385. Accessed 15 Sep. 2020.
[8] "Reducing Loss: Stochastic Gradient Descent." 10 Feb. 2020, https://developers.google.com/machine-learning/crash-course/reducing-loss/stochastic-gradient-descent. Accessed 15 Sep. 2020.

## Benchmark

The following conditions served as benchmarks for our model:

- The Convolutional Neural Network made from scratch needed to have at least an accuracy of 10%. This was chosen because, as shown in figure 6, the task of differentiating between some dog breeds is difficult even for humans themselves, and thus achieving a high accuracy using bare-CNN is not a trivial task.
- The pre-trained model was required to have an accuracy of at least 60% so that it could be successfully used in the dog breed classifier app. This comes from the intuition that at least more than half of the dogs should be correctly classified for being used in our final app.

# Methodology

## Data Preprocessing

The input to our models is images. There are two types of images in our dataset — human images and dog images, and we have separate types of classifiers for them. Below, we discuss the preprocessing steps related to each:

1. As mentioned before, we use a pre-trained classifier for our human face detector. Thus, the human dataset images are not required for training any model. However, we use the images for testing the performance of our human face detector. To do the same, the only preprocessing step required is converting the image to grayscale.
2. For the dog detector and the dog breed classifier, we use the images for training the dataset. Before feeding the model with the images, we do some pre-processing to them. They have been listed below:
    a. The images are resized to 224×224. This is inspired by the original VGG 16 paper.
    b. The images are cropped to the center.
    c. The images are normalized.

The images have 3 channels, red, green, and blue. They are normalized using a mean of 0.485, 0.456, and 0.406 respectively, and a standard deviation of 0.229, 0.224, and 0.225 respectively. These values have been taken from the PyTorch documentation.[9] What this essentially does is it changes each value in the image matrix as

$$image = \frac{image - mean}{std}$$

Normalization helps get data within a range and reduces the skewness which helps learn faster and better.[10]

d. For the dog breed classifier, the images are randomly flipped horizontally.
e. The images are also randomly rotated.

These pre-processing steps provide us with a greater variety of data and also prevent our model from overfitting.

# Implementation

This section describes in detail how each metric, algorithm, and technique was implemented. We go over each implementation one-by-one over the next few paragraphs.

### 1. The Human Face Detector

As mentioned in the section, "Algorithms and Techniques", we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. As said in the "Data Preprocessing" section, the image was converted into grayscale, using the OpenCV method, "cvtColor". The XML containing the pre-trained classifier was loaded using OpenCV's "CascadeClassifier". To find the number of faces in the image, the method detectMultiScale() was used. This returns the number of human faces that the classifier has detected. A function called face_detector() was written that would take in an image path a parameter, load it, convert it to grayscale, and use detectMultiScale() to get the number of human faces

[9] "torchvision.models — PyTorch 1.6.0 documentation." https://pytorch.org/docs/stable/torchvision/models.html. Accessed 15 Sep. 2020.

[10] "Understanding transform.Normalize( ) - vision - PyTorch Forums." https://discuss.pytorch.org/t/understanding-transform-normalize/21730. Accessed 15 Sep. 2020.

in the image. The return value is a boolean, which returns true if the number of humans is greater than 0, and returns false otherwise.

2. **The Dog Detector**

The dog detector is made using a VGG16 pre-trained model. PyTorch already provides many pre-trained models, readily available for use. The VGG 16 model was obtained, and loaded with its pre-trained weights, and the model was moved to the GPU. In the predict function, the transformations described in the "Data Preprocessing" step are implemented, and the image is fed into the model. If the output lies in 151 and 268 (inclusive), we claim that the image contains a dog.

3. **The Dog Breed Classifier**

At the heart of our project is our dog breed classifier. The dog classifier makes use of a ResNet-152 model. Data processing is similar to that done in the dog detector. The ResNet-152 model's pre-trained weights from the convolutional layer have been taken and used for the dog breed classifier model. We train our own fully connected layer, which takes 2048 input features, has 512 nodes in the hidden layer, and 133 output classes in the final layer. The model is trained using Stochastic Gradient Descent, with a learning rate of 0.001. The model was trained for 20 epochs, and its performance was measured using the log loss error, as mentioned in the section "Metrics".

4. **The Final Dog Breed Classifier app**

The final dog breed classifier app integrates all of the above implementations, to produce one final demo app. We check if the image is that of a human or dog using our detectors from steps 1 and 2. If it is not a dog or human image, we output the same to our user. Else, much similar to the earlier workflow, we use our dog breed classifier app, which takes in the image path, loads it, and obtains its predicted breed using the predict_breed_transfer() function. The predict_breed_transfer() function resizes the image to the size expected by the model, which is 224×224, and normalizes it. Then it obtains a prediction for the image using our dog breed classifier model. The prediction is then produced to the user as the predicted dog breed or the closest match to the human in the image.

# Refinement

The initial model consisted of Stochastic Gradient Descent with a learning rate of 0.001, up to 20 epochs, which achieved an accuracy of 80%. By using a learning rate

of 0.05, we were able to increase it to 84% which meets our benchmark expectations. The model was allowed to be trained for 30 epochs to observe the decrease of loss with epochs. It was observed that the model loss does not change much after around 19 epochs. Thus, we arrive at our final model consisting of 20 epochs and Stochastic gradient descent with a learning rate of 0.05.

# Results

## Model Evaluation and Validation

The model was evaluated using a validation set during training, and then on the test set. The best performing hyperparameters were chosen from all the combinations.

The final model uses Stochastic Gradient Descent with a learning rate of 0.05 and has a test accuracy of 84%, by predicting 704 out of our 836 testing images correctly. This sufficiently meets our expectations and can be used in our final app.

The final dog breed classifier model architecture consists of

- Convolutional layers from ResNet-152
- The first fully connected layer, with 512 hidden nodes, followed by a ReLU activation function
- A dropout with a probability of 0.3
- Our final fully connected layer, with 512 inputs and 133 outputs
- The training runs for 20 epochs

To verify the robustness of the model, the model was trained with 834 images, comprising all the 133 dog breeds. The learning rate of 0.05 worked fairly well in this case, and 20 epochs were enough to obtain a good accuracy result.

## Justification

Our final model has an accuracy of 84% on the test dataset, which is a better performance than our benchmark of 60%. With the discussions made before on the complexity of classifying dog images, this is a reasonable accuracy that we can use in our dog breed classifier app.

# Conclusion

## Free-Form Visualization

In this section, we provide some screenshots of the output produced by our model.

Hello hooman. Your doggo version would be Chinese crested.

Hi Bullmastiff.

Hi Anatolian shepherd dog.

# Reflection

The project, as described in this document, is a result of several days of research and development. This required extensive analysis of the data, research on the best available solutions, and a lot of experimentation with fine-tuning the hyperparameters. The following is a brief overview of the process used for this project:

1. The dataset was obtained and studied
2. The data was pre-processed, after relevant research on the general steps required for pre-processing
3. A basic plan to achieve our goal was thought out
4. A reliable technique for detecting humans in an image was found and implemented
5. A dog detector was created, after finalizing on the VGG 16 model
6. A benchmark was chosen for our models
7. A dog breed classifier was made from scratch to understand the complexity of the problem
8. A final dog breed classifier was made using transfer learning for use in our final app
9. All parts of the project were integrated

The most difficult aspect of the project was to be able to tune our final dog breed classifier for the best results.

# Improvement

Some areas of improvement in this project have been listed below:

- The classifier app currently has no provision of detecting multiple faces of dogs or humans in an image. In the future, the project could be extended to detect breeds for multiple dogs in the same image. This is not a trivial task and has a lot of corner cases, however, this could be achieved to some extent by limiting the number of dogs in an image. This might have required an additional object detection and separation technique implemented.
- The number of breeds classified by the dog could be increased from 133 to a greater number.