I. Definition

Project Overview

This project is an image classifier, specifically one specializing in identifying dog breeds with a small twist added in. At its core it's a classic computer vision problem, how to take an image and enable a computer to extract data from said image in order to make complex and meaningful decisions.

Do take note most of the structure and goals of this project were dictated ahead of time. As this project specifically is the CNN Project: Dog Breed Classifier. This was one of the options for the capstone project and came with pre stated goals along with a jupyter notebook to complete (dog_app.ipynb). Ultimately the finished project is said jupyter notebook, which is able to take in an image. Identify if said image is a human dog or unknown. Then if a dog or human guess the dog breed, or specifically which one of 133 different breeds. Or in the case of a human guess which breed of dog the subject resembles. The images to be used for the data were also provided ahead of time.

Problem Statement

Due to the pre structured nature of this project as mentioned above one can view the problem I sought to solve in two ways. The first way is ultimately the base goal of the notebook itself. To by the end create a algorithm which flows through the following steps,

1. Take in a image
2. Decide if the Image if a Human, Dog or other.
3. Output to the user the result of running said image through the model
    a. If a dog inform the user that the image belongs to a dog and what breed.
    b. If the image is of a human identifying they are human and what breed the subject may resemble.
    c. If neither human or dog say it is unable to tell what the input image is of.

You can also look at this project in a second way. That is the base jupyter notebook itself being a set of problems to solve in order to dive into CNNs (convolutional neural network) and transfer learning. With the dog breed classification simply being an overall wrapper to tie the tasks together. The notebook itself breaks this down into 6 steps.

- **Import Datasets** - Simply download and load the provided image datasets

- **Detect Humans** - Create a function to detect human faces with OpenCV's implementation of Haar feature cascades(4).
- **Detect Dogs** - Create a function to detect dogs based on a VGG-16 model trained on ImageNet(5).
- **Create a CNN to Classify Dog Breeds (from Scratch)** - This task was to create, train & test a CNN built myself from scratch, aka no transfer learning, to identify dog breeds. The cited goal being to reach at least 10% accuracy
- **Create a CNN to Classify Dog Breeds (using Transfer Learning)** - This step is essentially the same as step 4 but in this case I had to use transfer learning. Such as using an existing model like vgg16 as a base. In this case the target was at least 60% accuracy.
- **Write your Algorithm** - Here the goal is basically to combine steps 2, 3 & 5. Achieving the initial problem outlined above to detect Human or Dog then guess which breed the image resembles.
- **Test Your Algorithm** - This is simple to test the Algorithm against a set of 6 images, three human and, three dog.

Metrics

As mentioned prior this project is measured against its accuracy in detecting what breed a dog belongs to. In the case of the CCN built from scratch it was required to reach at least 10%. Then in the case of the model built with transfer learning it needed to hit at least 60%.

II. Analysis

Data Exploration

As mentioned before the dataset was provided ahead of time. In addition already split between train, test and validation. As to the contents we have 8531 dog images spread across 133 different breeds. The image sets for each classification are fairly balanced, averaging 45ish images per breed classification. In addition we have 13234 images of humans looking to belong to 5794 unique individuals. I will also note the images themselves are not uniform in size or quality.

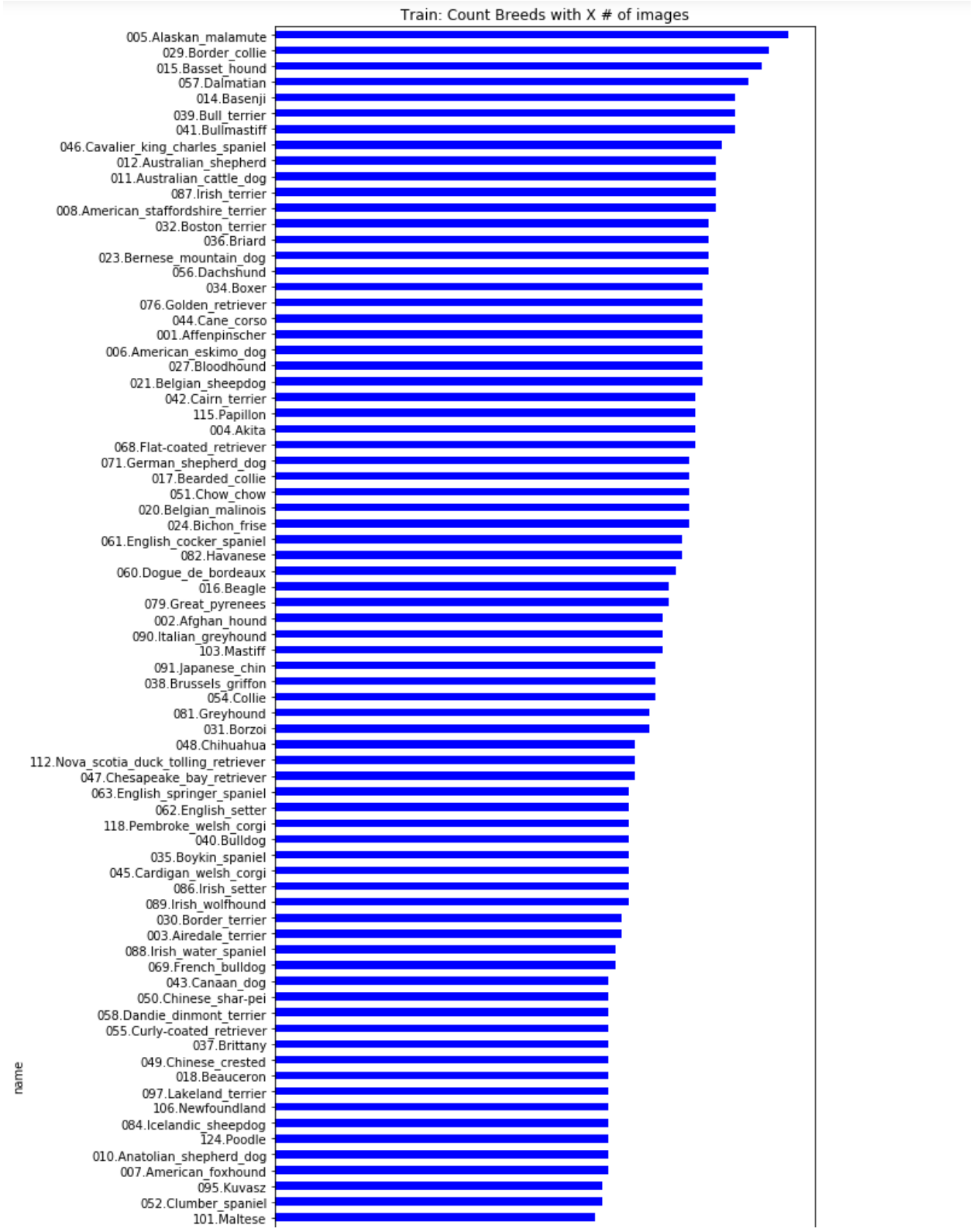
Data Set links:
Dogs - https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

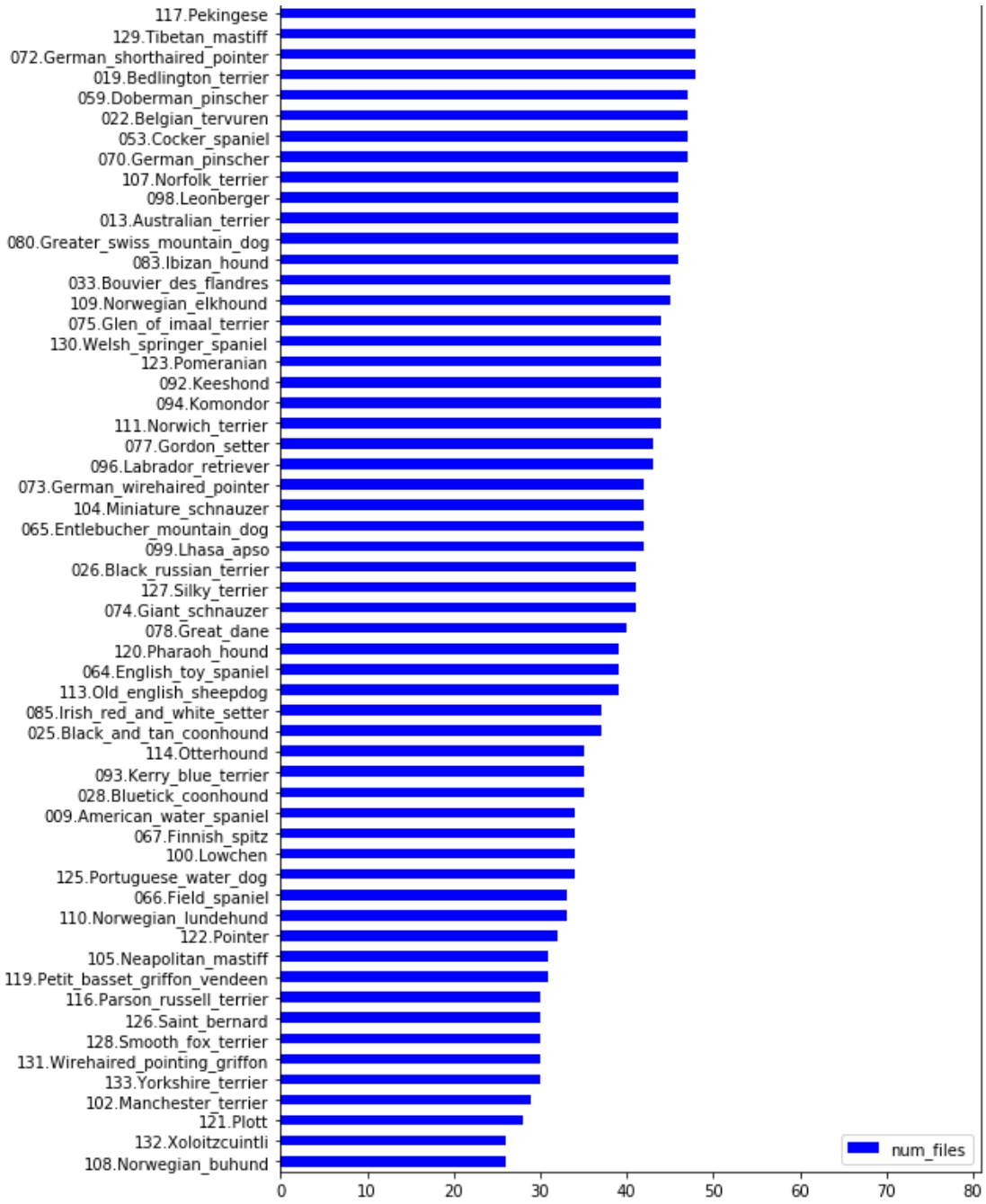
Humans - https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

See two examples from the dataset below,

Exploratory Visualization

In the below chart you can get an idea how many images the various dog breeds have. Along with the quantity of images. From this you can see the training data isn't exactly equal when it comes to the amount of data for each breed. The model gets over 70 images to learn what an Alaskan Malamute looks like but less than 30 to learn what a Norwegian Buhund looks like. This is something to bare in mind as the final model is likely to end up being better at recognizing some breeds over others.


Train: Count Breeds with X # of images
name
005.Alaskan_malamute
029.Border_collie
015.Basset_hound
057.Dalmatian
014.Basenji
039.Bull_terrier
041.Bullmastiff
046.Cavalier_king_charles_spaniel
012.Australian_shepherd
011.Australian_cattle_dog
087.Irish_terrier
008.American_staffordshire_terrier
032.Boston_terrier
036.Briard
023.Bernese_mountain_dog
056.Dachshund
034.Boxer
076.Golden_retriever
044.Cane_corso
001.Affenpinscher
006.American_eskimo_dog
027.Bloodhound
021.Belgian_sheepdog
042.Cairn_terrier
115.Papillon
004.Akita
068.Flat-coated_retriever
071.German_shepherd_dog
017.Bearded_collie
051.Chow_chow
020.Belgian_malinois
024.Bichon_frise
061.English_cocker_spaniel
082.Havanese
060.Dogue_de_bordeaux
016.Beagle
079.Great_pyrenees
002.Afghan_hound
090.Italian_greyhound
103.Mastiff
091.Japanese_chin
038.Brussels_griffon
054.Collie
081.Greyhound
031.Borzoi
048.Chihuahua
112.Nova_scotia_duck_tolling_retriever
047.Chesapeake_bay_retriever
063.English_springer_spaniel
062.English_setter
118.Pembroke_welsh_corgi
040.Bulldog
035.Boykin_spaniel
045.Cardigan_welsh_corgi
086.Irish_setter
089.Irish_wolfhound
030.Border_terrier
003.Airedale_terrier
088.Irish_water_spaniel
069.French_bulldog
043.Canaan_dog
050.Chinese_shar-pei
058.Dandie_dinmont_terrier
055.Curly-coated_retriever
037.Brittany
049.Chinese_crested
018.Beauceron
097.Lakeland_terrier
106.Newfoundland
084.Icelandic_sheepdog
124.Poodle
010.Anatolian_shepherd_dog
007.American_foxhound
095.Kuvasz
052.Clumber_spaniel
101.Maltese


117.Pekingese
129.Tibetan_mastiff
072.German_shorthaired_pointer
019.Bedlington_terrier
059.Doberman_pinscher
022.Belgian_tervuren
053.Cocker_spaniel
070.German_pinscher
107.Norfolk_terrier
098.Leonberger
013.Australian_terrier
080.Greater_swiss_mountain_dog
083.Ibizan_hound
033.Bouvier_des_flandres
109.Norwegian_elkhound
075.Glen_of_imaal_terrier
130.Welsh_springer_spaniel
123.Pomeranian
092.Keeshond
094.Komondor
111.Norwich_terrier
077.Gordon_setter
096.Labrador_retriever
073.German_wirehaired_pointer
104.Miniature_schnauzer
065.Entlebucher_mountain_dog
099.Lhasa_apso
026.Black_russian_terrier
127.Silky_terrier
074.Giant_schnauzer
078.Great_dane
120.Pharaoh_hound
064.English_toy_spaniel
113.Old_english_sheepdog
085.Irish_red_and_white_setter
025.Black_and_tan_coonhound
114.Otterhound
093.Kerry_blue_terrier
028.Bluetick_coonhound
009.American_water_spaniel
067.Finnish_spitz
100.Lowchen
125.Portuguese_water_dog
066.Field_spaniel
110.Norwegian_lundehund
122.Pointer
105.Neapolitan_mastiff
119.Petit_basset_griffon_vendeen
116.Parson_russell_terrier
126.Saint_bernard
128.Smooth_fox_terrier
131.Wirehaired_pointing_griffon
133.Yorkshire_terrier
102.Manchester_terrier
121.Plott
132.Xoloitzcuintli
108.Norwegian_buhund
num_files
0
10
20
30
40
50
60
70
80

Algorithms and Techniques

Three main items were used for this. A CNN, vgg-16 and OpenCV's implementation of Haar feature cascades. The simple but blunt reason for using these is, the project said to do so. In the case of all three they are used to look at a provided image outputting their best guess at the contents of the provided image. I will attempt to surmise why these were part of the project though.

First off why CNN? Ultimately CNNs tend to be the go to for image classification problems. This is for a few reasons. A major one being performance. Even one color image is a large amount of data. A color image that's even a smaller resolution like 224x224 still ends up as 150,528 (224x224x3) points of data. Multiply that by the thousands of images one would use to train the model. As well as looking at any real time applications and the performance speed becomes a major factor. Add on to that, how the data is handled. In a traditional neural network the data is flattened out. Basically like taking an image a lining up all the rows of pixels into one long row. Doing this loses a lot of context. CNN's are good at dealing with both of these issues. To avoid taking up to much time by diving into the why CNNs are good at handling these problems I will point you to an article on medium by Yash Upadhyay (see reference number 6 at the end of the report).

When it comes to actually implementing the CNN I do have several options to work with in order to tweak things for my needs.

- The training parameters
    - Epochs - how many time I train the model on the dataset
    - Batch size - How many of the images the model is trained on at a time
    - Learning rate - the speed at which it learns or how much emphasis it should place on the findings of each batch its trained against.
    - The optimiser and Loss function used.
- The Model or CNN itself
    - The types of layers and their structure (convolutional, fully-connected, pooling)
    - Dropout chance
    - Activation function on each layer

As to vgg-16 and OpenCV's implementation of Haar feature cascades there is not as much to say. I believe the project had me use Haar feature cascades as it was a quick and simple way to to perform face detection If you want more details see reference number 4 at the end of the report. For vgg-16 it is a pretrained model for image classification. It is easy to implement with pytorch. In addition it was trained against imageNet which already included dogs(5). Due to this and having had past experience working with vgg-16 I decided to use it for the base in the transfer learning section or step 5 as well.

Benchmark

Unfortunately due to the nature of the project there is not much to say about the target benchmarks. The project clearly stated the goal was a accuracy of 10% on step 4. Then an accuracy of 60% on step 5. There was no explanation as to why these numbers were chosen. I can only surmise this was to keep the problems relatively simply and avoid the need for any powerful hardware to train the model which may not be accessible to all students.

III. Methodology

Data Preprocessing

I put my image data through the following preprocessing. Note the validation and test images go through the same except for step 2 & 3.

1. Resize the image to 258
2. Apply a random horizontal flip
3. Apply a random rotation of 30
4. Center crop the image down to 224
5. Convert to a tensor
6. Lastly normalize with a mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]

This was chosen to stay in line with the standard input for pre trained pytouch models such as vgg16. Seeing as those projects have success with those inputs as well as they have worked well for me in prior projects I saw no need to change things. The resise to 258 before the crop enlarges the subject in the image and makes them a more dominant part, likely removing a layer of not essential data from the edges.

As to extra edits to the test set I added in a random horizontal flip & random rotation of 30. This was done in an effort to avoid overfitting. I chose these specific changes due to then working well for me in a different image classification model I made in the intro to AI with python nanodegree.

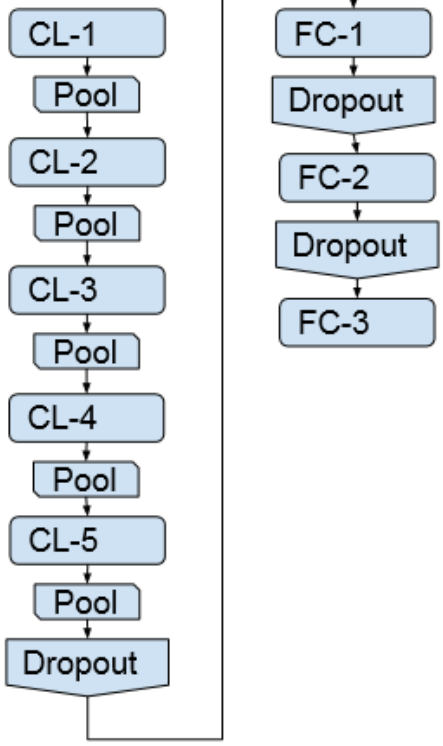
Implementation

In total this project has three main parts to discuss the implementation of. The CNN in step 4, the model with transfer learning in step 5, and the final algorithm from step 6.

To start off for step 4 I created a CNN or model to classify the dog breeds. As stated prior the goal of this step was to reach 10% accuracy. Laying the groundwork the model used cross-entropy for the loss function. Then Adam for the optimizer.

First the data was split into train, validation and test datasets in batch sizes of 100. Then the preprocessing transforms discussed prior are applied. From there they were loaded into three data loaders (train/valid/test). The test data begins its run through the model outlined below. The image tensor going into the model is 224x224x3.

Fig-1



See the Fig-1 for the basic structure of the model. CL are the convolutional layers, Pool are the maxpool layers, lastly DC are fully connected layers. Here are a few notes to better understand the figure.

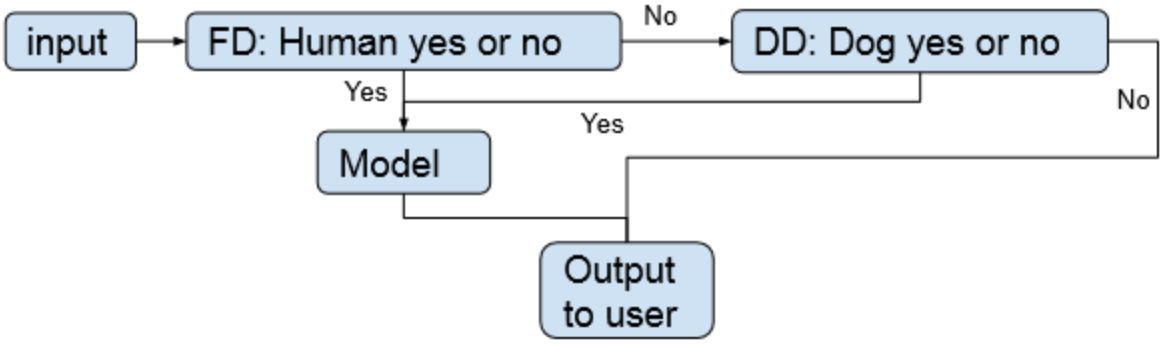
- The CL 1-5 increases the depth of the image tensor each time. Starting with the input size of 3 by the end of the 5th layer the depth is 1024
- The stride is set to 1 for each CL
- The padding on each CL is 1
- The filter size on each CL is 3x3
- The Pool are 2x2. Over the 5 CL this decreases the HxW of the image tensor to a 7x7 from 224x224
- Dropout rate is set to .5
- Each CL uses a relu function

After the image tensor passes out the first dropout layer it is 7x7x1024. FC-1 sends this to 512 hidden layers. FC-2 reduces it to 256 with FC-3 reaching the desired final output of 133 matching up with the total number of dog breeds.

For the transfer learning model in step 5 almost no changes were made to the original pretrained model. This was due to only a minor edit to change the output tensor to 133 being required. As opposed to 1000 it had originally from imagenet's categories. You can see details on VGG16 in the PDF found in reference link 7. I froze the feature layers of the pretrained model, then changed the third linear layer in the classifier from 1000 to 133. After training this was the only edit needed to reach the required accuracy.

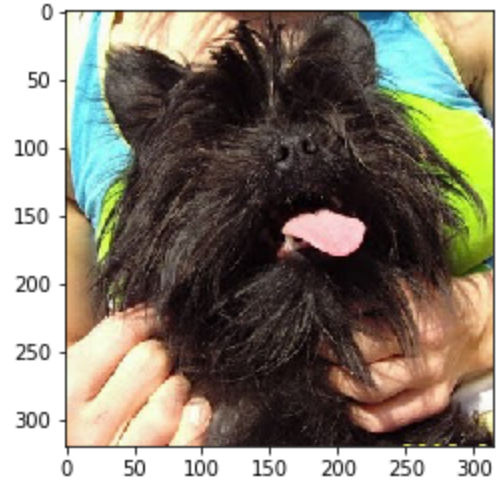
Lastly for the final algorithm created at the end of the project it is also fairly simple. See figure 2 for the overall structure. Basically the input of the image comes in. First it hits FD or a face detector function created in step 2 of the project. Using OpenCV's implementation of Haar feature cascades it determines if the image has a human face or not.

Fig-2



If yes it notes the image is human and proceeds to the transfer learning model created in step 5. If not, it goes to DD or the Dog image detector. This was created in step 3 using an unaltered version of VGG-16.

If at this point the subject of the image still can't be determined to be dog or human the image is printed back to the user stating the subject of the image could not be identified. If either FD or DD returned true the image is sent to the model created in step 5 to determine what dog breed the subject most resembles. After that it performs the output to the user. Printing the supplied image. Telling the user if the supplied image was a human or dog. Then stating what breed they most closely resembled. See image below for an example of the output.




```
Hello! I think you are a  dog!
You look like a Affenpinscher
```

Refinement

In total I think I tested around 20+ different models to get to the final version you see above. As a base I started with a model based on the CNN lesson of the Nano Degree. Only modifying the input parameters to fit. This unfortunately did not work well only yielding a 1% accuracy.

The model I reference in the nano degree which I used as a starting point similar structure to my final model. It was intended to take in 32x32 images so I adjusted accordingly. In this case it had three convolutional layers. Take the initial tensor depth of 3 and increase it to 64 by the end of the 3rd layer. It also used a 3x3 filter. Add to that it had only 2 fully connected layers. Also using a dropout of .25.

From there I started tweaking random parts of the model in order to see the results it had. Still no luck always getting 1% or worse. Then I started doing research online for inspiration and saw mention of creating a simplified version of vgg16 as a base. Still no luck, nothing I did could even hit 2%. So I started to look outside the model itself for the issue.

This approach worked, as I severely underestimated the effect of the ratio between batch size, learning rate and, epochs. After a little trial and error to find a batch size that didn't overload my GPU I settled on a batch size of 100 and LR of .001. I also found that switching my optimiser from SGD to Adam made a large improvement in the models performance. At this point I had hit the required 10%. Though seeing as I had really only touched the model itself prior to the batch/optimiser changes I wanted to go back and get a feel for the cause and effect of different edits before calling it good.

This gets into why I made some of the decisions on the model I did. I went with a maxpool of (2,2) and a filter of 3 as from what I saw online they look to be pretty common almost standard sizes. I went with 5 convolutional layers to get the final HxW of the tensor from 224x224 to 7x7 from the pooling layers in between each convolutional layer. Due to the size of my image I wanted to make sure I had a good deal of depth added to avoid losing too much from the maxpools. So I started at going from 3 to 64 depth doubling it each time ending on 1024 for the final convolutional layer. As to the FC layers I originally had 2 but adding in the third made a noticeable difference in the rate at which the loss decreased. I was noticing at this point the train_loss and valid_loss were getting badly out of sync and looked to be an issue of overfitting. To combat this I added upped the dropout to .5 then additionally added dropout after the convolutional layers instead of just the first two FC layers.

IV. Results

Model Evaluation and Validation

To reach the final models for both steps 4 & 5 a validation set was used during the training. Basically testing the model against a separate set of data each epoch. Then saving the model with the lowest validation loss during training. These saved models found to have the lowest validation loss were tested for accuracy. This was done against a separate set of test data not used during training or validation. It was these final accuracy results against the test data which helped me decide on the final model. Namely if they meet or exceed the target requirements.

In the case of the model created in step 4 (the CNN built from scratch) the target was 10% accuracy when run against the test data. My model as can be seen in the accompanying Jupyter notebook reached 37% test accuracy.

For the model created in step 5 (VGG-16 base with transfer learning) the target test accuracy was 60%. My final model achieved 83%.

In both cases I had created models with reached the target accuracy before reaching these final models. I decided to push them a bit further to add a little breathing room rather than submit a model that barely hit the targets.

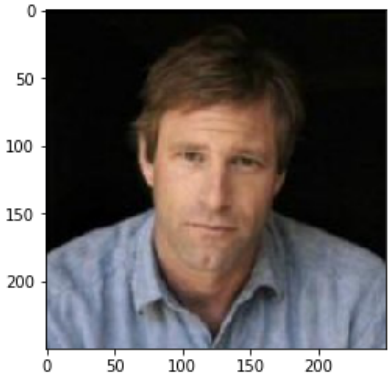
Justification

Unfortunately not much to say here as the required benchmarks were rather simple. In both cases the models significantly exceeded the target accuracy. The model from step 4 was only required to hit 10% where my model beat that goal by 27% reaching 37% accuracy. Then for the model from step 5 the target goal was 60% where my model hit 83%, 23% over the target benchmark.

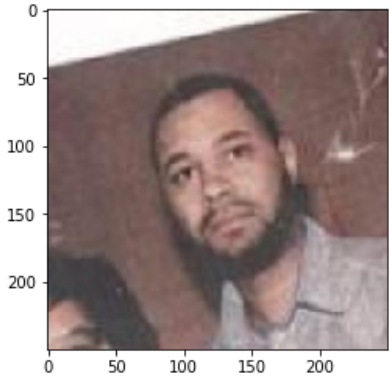
V. Conclusion

Free-Form Visualization

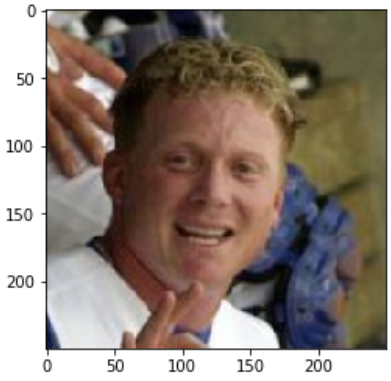
Below you can see results from some of the final tested run through the completed algorithm in step 6. With results from both dogs & Humans. You can see in the top right an example of when it failed to correctly identify the subject in the target image.



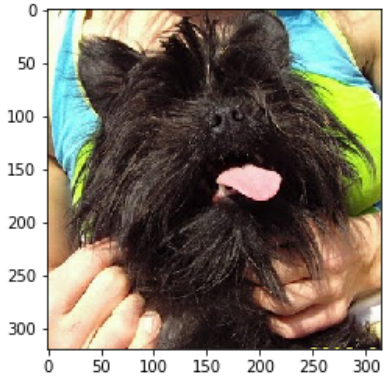
```
Hello! I think you are a  human!
You look like a Chinese crested
```



```
Hello! I think you are a well I am not sure what you are...
You look like a well again, nothing I am aware of
```



```
Hello! I think you are a  human!
You look like a Brittany
```



```
Hello! I think you are a  dog!
You look like a Affenpinscher
```

Reflection

I picked this as my capstone project due to finding image classification a very interesting topic. Seeing it as a way to learn more about it and CNN's above and beyond the standard curriculum by diving into the extra lessons and the project itself. This for better or worse dictated and structured much of the project for me. Such as the initial Jupyter notebook with instructions, I just had to add the code. As well as the preset dataset & benchmark targets. From there I followed through the steps outlined above. I will give some quick thoughts I had going through each.

- **Import Datasets** - Nothing of note, was convenient to have the data already provided and segmented.
- **Detect Humans** - This also was a simple task, side some basic image pre-processing it only ended up a few lines of code. I am curious to try writing my own version and see if I can hit similar levels of accuracy.
- **Detect Dogs** - This step was arguably simpler than the face detector. Vgg16 at its pretrained state can detect dogs so all I had to do was feed it pre-processed data. Luckily I had functions to process image data for vgg16 I could toss in from a prior project.
- **Create a CNN to Classify Dog Breeds (from Scratch)** - This is where the project started to give me some trouble. As mentioned in the refinement section. Getting a model that could even hit 10% was a major trial. I became overly focused on the model itself rather than looking at other factors. In the end increasing the batch size and trying new optimisers is what did the trick. At least getting me on the right course so I could properly test the effects of other changes.
- **Create a CNN to Classify Dog Breeds (using Transfer Learning)** - Not much to say for this part. Base pretrained vgg16 is already good at detecting dogs. It simply is setup to output for the image-net categories. After changing the final output to be 133 matching up with the breeds it easily hit the required 60% accuracy with only a few epochs. Just to be safe I let it run 20 epochs hitting 83%. In all likelihood I could have easily pushed it higher. My curiosity for future experiments will be to see how high I could get it before a more complicated model is required.
- **Write your Algorithm/test** - Honestly this part was very simple as well. As it was little more than tossing the functions built for steps 2&3 with the model from step 5 into if statement.

Improvement

I will say given more time or a round two of this project there are several things I would like to try to improve / make the final result more interesting. First off would be to get a larger and more diverse set of data to train the model with. For example I would be curious to see how the model would fair with images where a dog is in the image but possibly not the only subject. Or taking that a step further take the model to a point where it can tell the amount of subjects in an image and perform classification for all of them. For example if given an image with three dogs it could tell the user how many dogs and let you know the breeds of all three.

After that increasing the accuracy would likely be the most important improvement. For both of the final models i was able to significantly beat the target benchmarks. However for a real world application the accuracy is far too low. I would want to reach at least 95% if not 99% accuracy.

Lastly I would like to pull this out of the Jupyter notebook. As it stands the notebook is an easy way to test and present the model. It lacks any practical use though. So I would like to get it running as an app on a smartphone. Where you could take a picture of a dog and almost instantly be told the breed with some basic information and links to further reading.

References

1- https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3

2-https://www.smithsonianmag.com/smart-news/ai-helps-identify-1-143-new-nazca-lines-180973621/

3-https://healthtechmagazine.net/article/2019/01/computer-vision-healthcare-what-it-can-offer-providers-perfcon

4-https://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

5-http://www.image-net.org/

6- https://medium.com/alumnaiacademy/introduction-to-computer-vision-4fc2a2ba9dc

7-https://arxiv.org/abs/1409.1556