



Capstone Project Report

Prepared for: Udacity's Machine Learning Engineer Nanodegree Program

Prepared by: Anantharam Vemuri

October 30, 2020

Report number: 001

I. Definition

Project Overview

The problem I have chosen to work on is to create a dog breed classifier. The aim is to detect a dog in an image and estimate the dog's breed using a CNN. If an image of a human is given, the algorithm will estimate the resembling dog breed.

A machine learning approach to image classification involves identifying and extracting key features from images and using them as input to a machine learning model. With this method, the computers are taught to recognize the visual elements within an image.

CNNs are especially sophisticated in the domain of computer vision. They are a class of deep neural networks and are inspired by the human brain. They are composed of 3 - dimensional layers.

The reason I have chosen this project is because I am interested in learning about the methods and algorithms used to solve problems involving image data. Also, solving the dog breed classification problem is not easy even for a human. There are similar looking dogs belonging to different breeds and differently looking dogs from the same breed.

The pipeline built here can be used within a web or mobile app, which could potentially take a user-supplied image as input and output the dog breed. This approach to image classification has many potential uses and can be used to solve many real life problems.

Problem Statement

Given an image of a dog, estimate and output the dog's breed. If a human's image is given, output the dog breed the human resembles. The problem can be broken down into the following steps:

1. Check whether the input image contains a dog. If it does, estimate the dog's breed.
2. If no dog is detected, check whether the image contains a human face. If it does, estimate the dog breed the human face most closely resembles.
3. If neither a dog nor a human have been detected in the image, output an error message.

Metrics

Accuracy is the fraction of predictions our model got right. In terms of positives and negatives it can be defined as:

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

where, TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives

II. Analysis

Data Exploration and Visualization

The datasets of dog images and human images provided by Udacity are used. There are 13233 total human images in the human dataset and there are 8351 total dog images in the dog dataset.



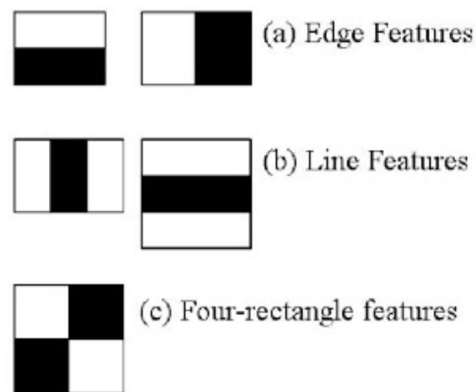
Sample dog images



Sample human images

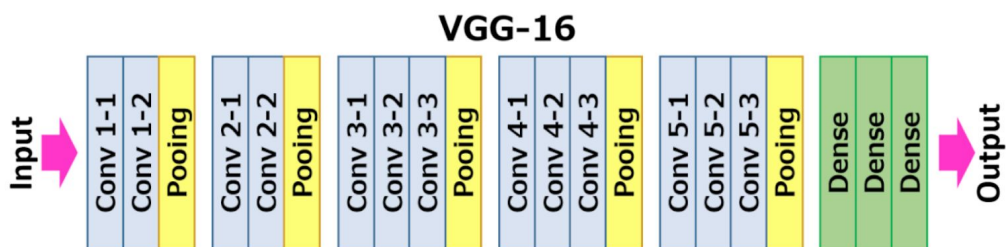
Algorithms and Techniques

1. For human detection, OpenCV's implementation of 'Haar feature - based cascade classifier' is used. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. Some example Haar features are given below:



All possible sizes and locations of each kernel are used to calculate lots of features.

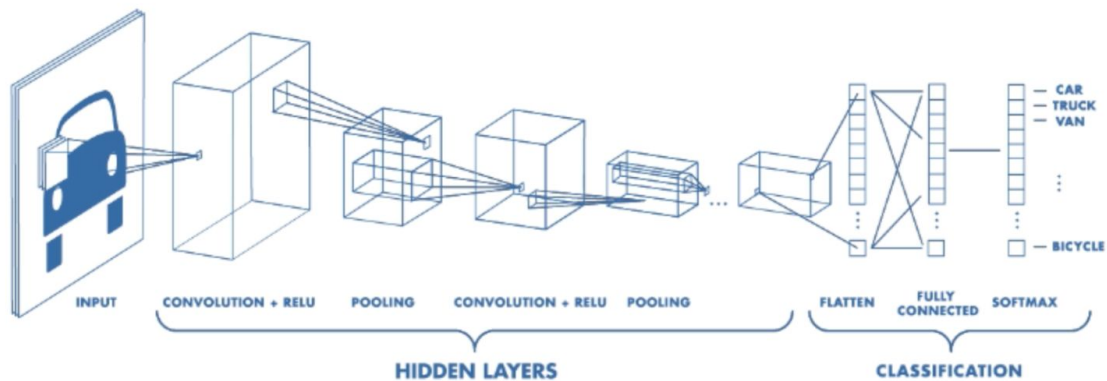
2. For dog detection, a pretrained VGG16 model is used. The VGG16 model made an improvement over AlexNet by replacing large kernel-sized filters with multiple 3x3 kernel-sized filters one after another. The architecture of VGG16 is given below:



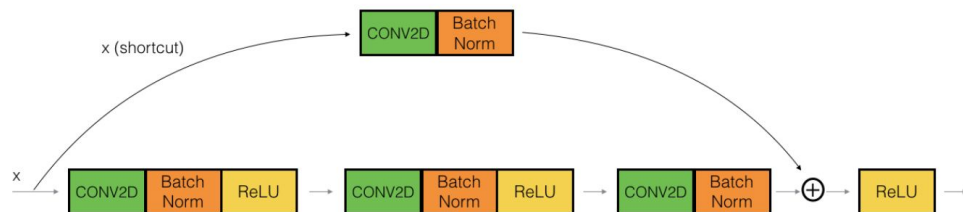
3. Finally, CNNs are used for dog breed classification. CNNs are a certain category of neural networks used for computer vision tasks. Like any neural network, CNNs consist of an input layer, an output layer and several hidden layers.

Since images are, at the core, large arrays of pixels, at a high-level, the CNN applies a series of filters to the arrays to achieve some objective such as feature extraction or object detection.

Some standard filters or operations are the convolution operation, pooling, flattening, dense layer, etc. An example CNN architecture is shown below:



ResNets are a type of CNN. ResNet stands for residual networks. A ResNet50 has 50 layers. Prior to ResNet, training very deep neural networks was difficult due to the problem of vanishing gradients. The strength of ResNets is the presence of a 'skip connection'.



A skip connection is basically a connection that does not connect directly to the next layer but instead to a layer several layers ahead. It essentially 'skips' layers. This allows an alternate shortcut path for the gradient to flow through, hence overcoming the vanishing gradients problem.

Benchmark

I initially constructed a CNN model from scratch with the goal of achieving at least 10% accuracy. This level of accuracy is enough to prove that the outputs aren't random. The random chance presents an exceptionally low bar. A random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

To improve the accuracy, transfer learning was then used. I chose the base model as ResNet50 (residual network with 50 layers) as it has a low error rate. It was first used in 2015 so it is a relatively recent one.

Here the goal was to get an accuracy of at least 60%.

III. Methodology

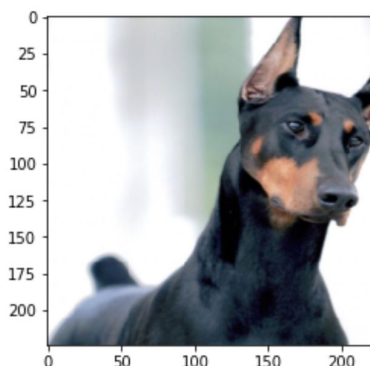
Data Preprocessing

The data was preprocessed for the 2 CNN models: CNN from scratch and ResNet50. The preprocessing steps performed are summarized below :

Images were resized to 224 x 224 as it is the default input size for the VGG16 model. Some data augmentation was also performed. Data augmentation is mainly used to increase data using the data we currently have. Here, images were randomly cropped and were randomly horizontally flipped. For validation and test data, only resizing has been done. Also, normalization is applied to all the three datasets. A sample original and transformed image pair is given below:



Original Image



Transformed Image

Implementation

The overall project can be split into 4 major parts:

1. Detecting Humans: Here, OpenCV's implementation of 'Haar feature-based cascade classifiers' is given as a .xml file. The input images are converted to gray-scale. These images are then input to a pre-trained instance of the above stated classifier.

The top-left coordinates and the height and width of the human face in the image, if exists, are returned and are stored in an array.

The algorithm detected human faces in 98 images out of the 100 human images. Human faces were erroneously detected for 17 images out of 100 dog images provided.

2. Detecting dogs: Here, a pre-trained VGG16 model is used to detect whether a dog is in a given image. VGG16 has been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

The image input is first converted to a tensor. This is given as the input to the instance of the pretrained VGG16 model. An index is returned which denotes a particular class of images. If the index returned is in between 151 and 268 (both inclusive), it means that the category recognised is a dog.

This algorithm detected dogs in 1 image out of the 100 human-face containing images. Dogs were detected in 77 images out of 100 dog images provided.

3. CNN (from scratch) to classify dog breeds: After algorithms and methods were specified for human and dog detection, a CNN was constructed to estimate the dog breed of the input image. This task is actually quite difficult even for a human to do as many dogs of different breeds look alike and many dogs of the same breed look different.

To see why, consider that even a human would have trouble distinguishing between a Brittany and a Welsh Springer Spaniel.



Brittany



Welsh Springer Spaniel


Since, the CNN is being constructed from scratch and then must be trained, data loaders are specified for the dog dataset. Namely: training, validation, and test datasets of the dog images (located at `dog_images/train`, `dog_images/valid`, and `dog_images/test`, respectively). Preprocessing is first performed for each part.

After specifying the various layers of the ConvNet, the optimizer and loss function are chosen. Cross entropy was chosen as a measure of loss and stochastic gradient descent (SGD) was chosen as the optimizer.

Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events. Information quantifies the number of bits required to encode and transmit an event. Lower probability events have more information, higher probability events have less information. Cross-entropy is widely used as a loss function when optimizing classification models. Cross-entropy is useful for optimizing a multi-class classification model.

Stochastic gradient descent(SGD) is an iterative method for optimizing an objective function with suitable smoothness properties. SGD has become an important optimization method in Machine Learning.

The model is then trained for 10 epochs and on testing, the accuracy obtained was about 11%. A screenshot of the results is shown below.



Epoch: 1	Training Loss: 3.435674	Validation Loss: 3.759032
Epoch: 2	Training Loss: 3.424115	Validation Loss: 4.009160
Epoch: 3	Training Loss: 3.387997	Validation Loss: 4.250486
Epoch: 4	Training Loss: 3.364592	Validation Loss: 3.655367
Epoch: 5	Training Loss: 3.342494	Validation Loss: 3.765188
Epoch: 6	Training Loss: 3.289076	Validation Loss: 4.179628
Epoch: 7	Training Loss: 3.221020	Validation Loss: 5.400033
Epoch: 8	Training Loss: 3.233429	Validation Loss: 3.931907
Epoch: 9	Training Loss: 3.140202	Validation Loss: 3.840931
Epoch: 10	Training Loss: 3.145440	Validation Loss: 3.748207

4. CNN (using transfer learning) to classify dog breeds: The same data loaders: training, validation, and test are used here as well. Since transfer learning is being used and the resulting CNN must be trained on the dogs dataset.

ResNet50 CNN architecture was chosen here as the base model to perform transfer learning. After creating an instance of ResNet50, an additional fully connected layer (also known as a dense layer) is added to the architecture with 133 neurons. The final fully connected layer is used to make the actual prediction and the neurons present in this layer indicate the 133 dog breeds.

Now, the loss function and optimizer are specified. Similar to the other CNN model, cross entropy was chosen as a measure of loss and stochastic gradient descent (SGD) was chosen as the optimizer.

The model is trained for 10 epochs, and on testing, the accuracy obtained was 81%. A screenshot of the results obtained is shown below.

```

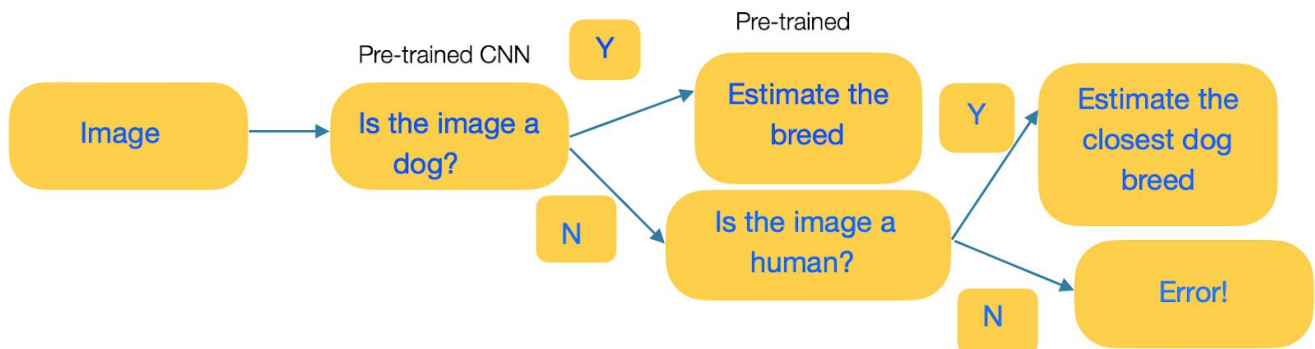
: # train the model
model_transfer = train(10, loaders_transfer, model_transfer, optimizer_transfer,
                       criterion_transfer, use_cuda, 'model_transfer.pt')

# load the model that got the best validation accuracy (uncomment the line below)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))

```

Epoch: 1	Training Loss: 3.318061	Validation Loss: 1.911950
Epoch: 2	Training Loss: 1.778175	Validation Loss: 1.084868
Epoch: 3	Training Loss: 1.362957	Validation Loss: 0.855016
Epoch: 4	Training Loss: 1.185670	Validation Loss: 0.773508
Epoch: 5	Training Loss: 1.066507	Validation Loss: 0.679991
Epoch: 6	Training Loss: 1.003799	Validation Loss: 0.628749
Epoch: 7	Training Loss: 0.963756	Validation Loss: 0.598442
Epoch: 8	Training Loss: 0.898581	Validation Loss: 0.605881
Epoch: 9	Training Loss: 0.862601	Validation Loss: 0.608073
Epoch: 10	Training Loss: 0.865810	Validation Loss: 0.603544

After all the above steps, all that remains is to develop the final algorithm which makes use of the above detectors and breed classifier. The decisions that the algorithm has to take are summarized in the below flowchart:



Finally the algorithm is tested on a few sample images.

Refinement

1. Initially, both the CNNs were trained with 5 epochs. There was a slight improvement in the accuracy when the number of epochs were increased to 10.
2. Adding a dropout layer to the CNN (built from scratch) changed the accuracy. Some hyperparameter tuning of the parameter specifying the probability at which the layers were being dropped out had to be performed.

IV. Results

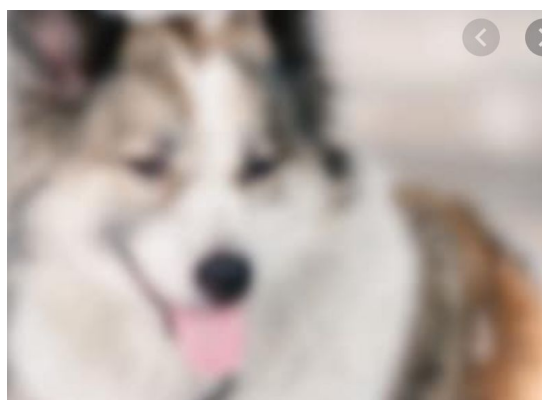
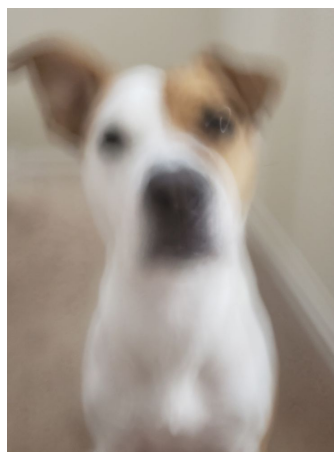
Model Evaluation and Validation

A validation dataset is a dataset of examples used to tune the hyperparameters of the model fit on the training dataset. On the other hand, the test dataset is predominately used to describe the evaluation of a final tuned model when comparing it to other final models.

Now, in addition to the images of dogs and humans present in the datasets, some additional testing was performed to evaluate the performance of the model.

1. Can a blurry image of a dog be detected?

When a blurry image of a dog is provided to the `dog_detector` function it returns 'false' response. That is, it isn't successfully detected. Some blurry images I tested the dog detector with:



2. What would happen when images with multiple dogs are input to the dog detector?

Here in some cases a dog was getting detected and in others it was not. So the output was unpredictable. Some test cases are:



Here, the method returned 'true' sometimes and 'false' sometimes for the same image.



Here, the detection was always 'true'.



Here the detection was consistently 'false'.

3. What would happen when images containing multiple dogs and humans are input to the dog detector?



Here, a dog was getting consistently detected.



Here, a dog was not getting detected.

Justification

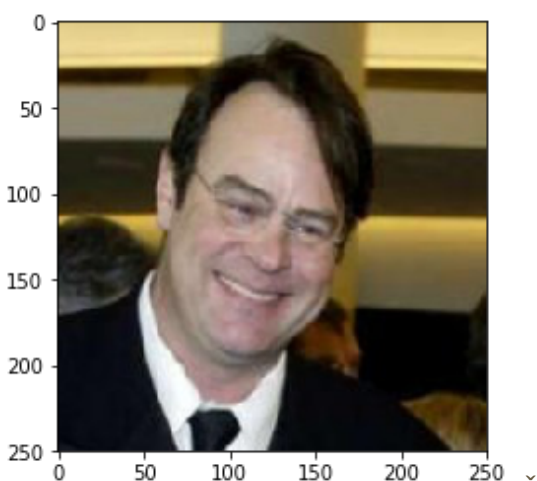
The performance of both of the ConvNets: CNN from scratch and ResNet50 are summarized below:

- The CNN I built from scratch got an accuracy of 11% and the ResNet50 CNN used for transfer learning got 81%. Both the models crossed the benchmark accuracy and hence have performed well.
- It can be seen that the detection does not work very well in case there are multiple objects in the image such as a distracting background. It also seems to be getting confused when multiple dogs are present in the same image.
- Overall, it works well for a limited scope of single object images with not too distracting backgrounds.

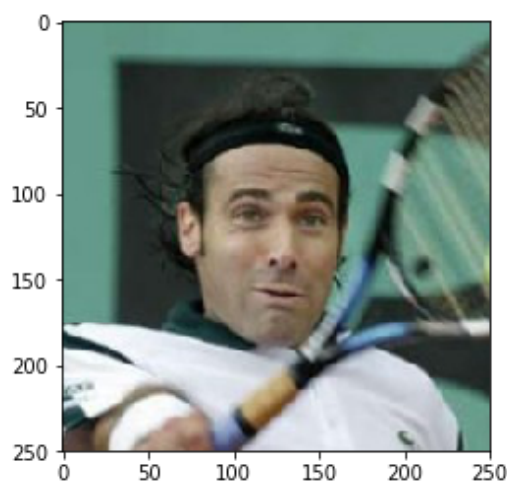
V. Conclusion

Free-Form Visualization

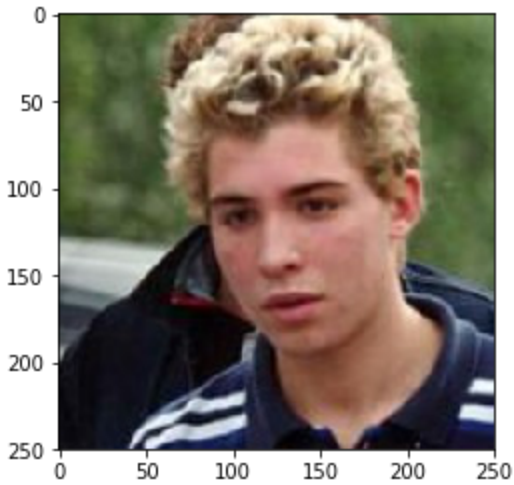
Examples of predicted breeds for 3 human images and 3 dog images are given below:



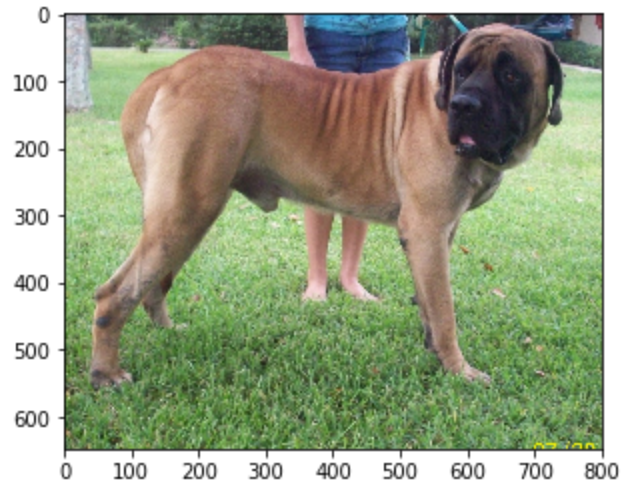
Human detected, you resemble a Newfoundland



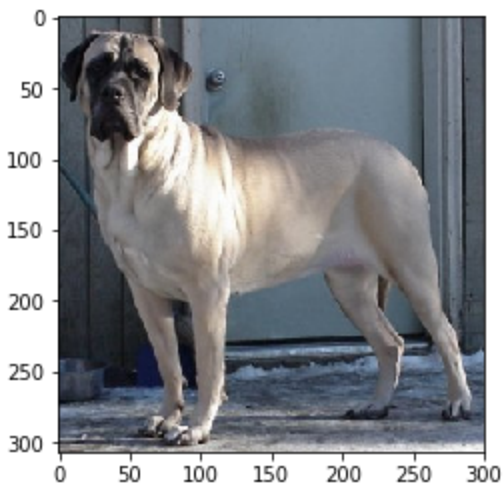
Human detected, you resemble a Briard



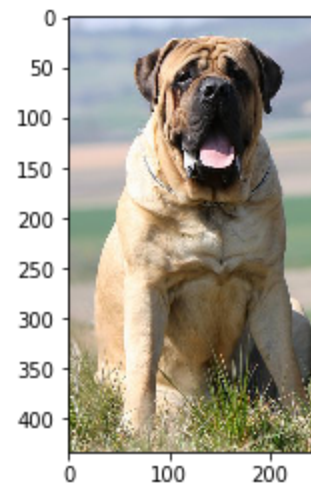
Human detected, you resemble a Briard



Dog Detected, it might be a Boston terrier



Dog Detected, it might be a Boston terrier



Dog Detected, it might be a Boston terrier

Reflection

At a high-level, the steps following for this project were:

1. Deciding the problem statement to work on.
2. Data was already provided by Udacity.
3. Creating benchmarks for the CNNs.
4. Building the dog and human detectors.
5. Building the CNN from scratch and then using transfer learning.

I think the part which took the longest was building the CNN from scratch, adjusting the parameters and fine-tuning the hyperparameters.

It was a fun learning experience giving different types of images to the model and seeing what it would predict.

I'm glad I chose this project as I now have a good basic understanding of how CNNs work.

Improvement

- Firstly the model's accuracy can be improved by increasing the data.
- Since the model works reasonably well for images containing primarily 1 object, multiple object detection can be introduced.
- Some more preprocessing can be done for the images to filter out all the unimportant stuff from the images.
- Finally, this model can be integrated into an android app and can be potentially used for real time dog breed classification.

References

1. 'Introduction to Machine Learning with Python', Andreas C. Muller & Sarah Guido
 2. <https://medium.com/nanonets/how-to-easily-build-a-dog-breed-image-classification-model-2fd214419cde>
 3. Architecture of a CNN. — Source:
<https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html>
-