Dog Breed Classification

Final project report for the Udacity Machine Learning Engineer Nanodegree

Author: Gavin Ayres **Date:** April 2021

Domain Background

This project is primarily concerned with the field of computer vision and specifically in an applied setting. The task at hand, classifying dog breeds, is a challenging one, even for humans. However, it is precisely this challenge which makes it an important illustration of the power of deep learning approaches to computer vision. It is a particular aim of this project to achieve significant accuracy on the test set (>60%), with our dog breed classifier, as a means of illustrating the speed to development of deep learning approaches to certain problem representations. The motivation for this project choice is primarily because the author has limited previous experience in computer vision tasks and would like to further develop some proficiency.

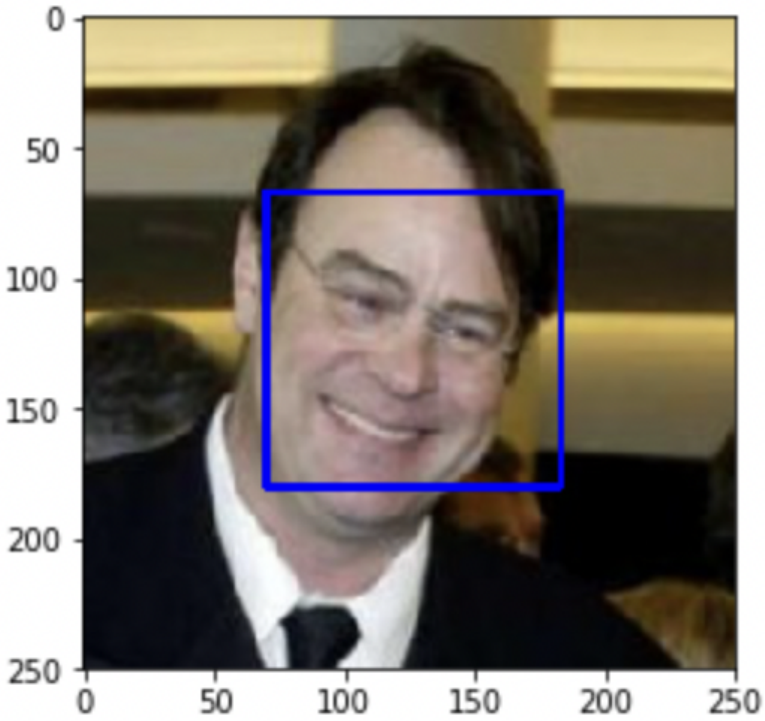
Problem Statement

The problem to which this project will provide a solution is that of dog breed classification. More precisely, given an input image of a dog we aim to assign it to a specific breed. If we are shown an image of a human we will assign it the dog breed of highest association. The solution will therefore be required to implement two high level features, filtering input images to ones we can appropriately classify and then assigning *dogs* and *humans* to breeds.

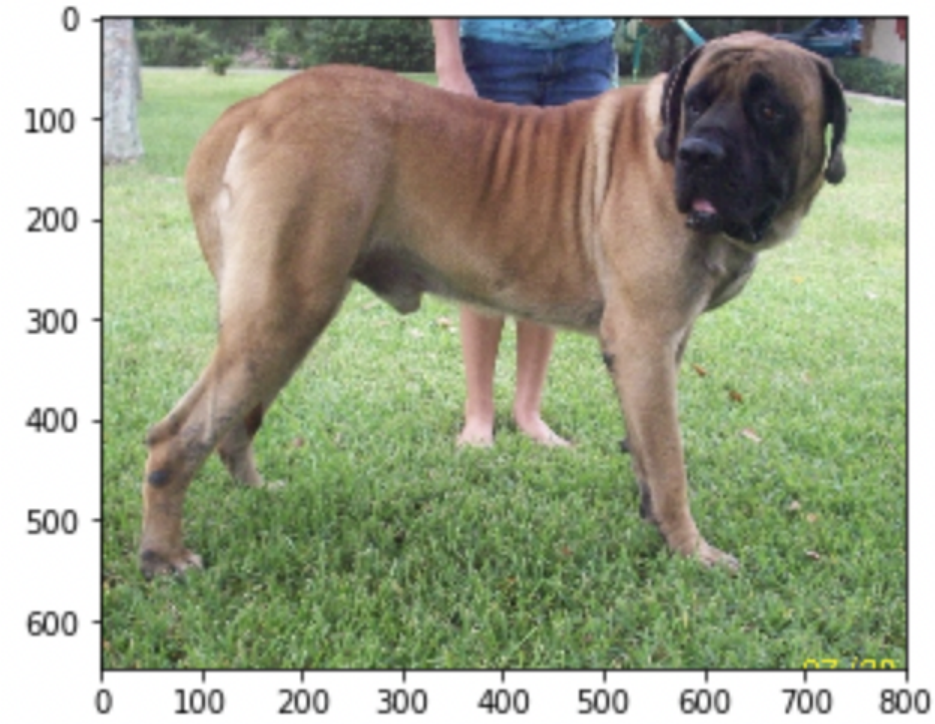
Datasets and Inputs

The datasets required for this project consist of, as one would expect, images of dogs and humans(1). Necessarily, these datasets are separated by species. The *dogs* dataset is broken up into train, test and validation folders with subfolders consisting of breeds and their images. The amount of images that make up each breed directory are imbalanced and consist of different size and quality. The *humans* dataset are subsetted by name and again vary in

Number of faces detected: 1



size and quality.



Nice dog, I think you're a...
Mastiff! That's the one!

Solution Statement

We outlined previously that our solution would require two high-level features. We will now go into further detail. Starting from the end of our pipeline, the breed classifier we will train will be a convolutional neural network (CNN), the exact architecture will be decided upon via experimentation on the test data set. The component to identify dogs will be a pre-trained VGG-16 model(2). We will use Haar feature-based cascade classifiers(3) to identify humans.

Evaluation Metrics

We will use categorical cross entropy as our loss function for the multi-class classification problem of assigning images to dog breeds. We will also consider typical classification metrics, such as accuracy, for each class.

Benchmark Model

We have an implicit benchmark of ~10% accuracy on the test set (outlined in the notebook template). Random chance (1 in 133) will set a baseline of <1%. The fine-tuned CNN must have at least 60% accuracy on the test set. We also have a baseline CNN architecture which achieved 23% accuracy on the test set.

Algorithms and Techniques

The discussion of pertinent algortihms will remain at a high level here and we will go into further detail regarding their implementation in later sections. We detect detect dogs using a pretrained VGG-16(2). VGG16 is a convolutional neural network model developed by researchers from the University of Oxford. The model achieves 92.7% top-5 test accuracy on ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. Now in order to narrow our task from general image classification to *dog* classification we use the following process;

- input image to VGG16 model,
- obtain vector of probabilities across all classes,
- take the index of the highest probability class,
- check if this index aligns with the class indices of ImageNet which relate to dogs,
- return True if this condition satisifed, False if not.

For classifying human faces we use a slightly different approach. We use a pre-trained Haar feature-based cascades classifier(3) downloaded from OpenCV(4). At a very high level, the classifier works by first computing Haar features, calculated by summing pixel intensities over particular regions and calculating the differences between regions. We then learn a classifier to learn a threshold which can separate features indicative of an object from those which aren't. The cascade comes into play because we essentially build up our classifier from a series of classifiers each applied on features computed over a different sized window.

The next step is now to construct our dog breed classifier. The final classifier chosen was a pre-trained DenseNet(5) where we fine tune a fully connected layer appended to the end of the pretrained layers. The motivation for this choice was due to the computational efficiency of DenseNet's and their state of the art performance.

Data preprocessing

The image transforms chosen followed conventions common in online examples as well as in the paper(6), note that they differ between training set and testing and validation sets, this is because we want reproducible behaviour at test time and thus want to remove any random elements we can. *Train*:

- Firstly we randomly resize the image and then crop a 224 by 224 patch.
- Next we produce a random horizontal flip with a 50% probability.
- We then randomly rotate the image, by performing a uniformly sampled rotation of between -90 and 90 degrees.
- Convert to tensor.
- Normalize.

Validation:

- Resize the image to a 256x256 patch.
- Center crop down to 224.
- Convert to tensor.
- Normalize.

Test:

- Same as validation.

Implementation

The implementation of the dog detector and the human detector adhered to the standard out-of-the-box implementation offered by PyTorch(7) and OpenCV respectively. However for the human detector we experimented with a deep learning based approach, specifically, a pre-trained Multi-Task Convolutional Neural Network (MTCNN)(8). We tried this architecture for it's purportedly state of the art results. However, we were not able to find suitable hyperparemeters for it's implementation which could lead to any significant improvement over the Haar feature-based classifier.

In implementing our baseline CNN model it took a few iterations to find a suitable architecture. Initially, it was decided to implement AlexNet from scratch. The motivation for this was the status of AlexNet as a seminal architecture in the field and the author wanted to take advantage of the GPU instance to develop an intuition around the training process for a successful architecture on a slightly different task (just dogs vs all of ImageNet). Of course, it transpired that the training process would be lengthy and hence, the iteration speed slow so it was then decided to pare back a few layers from the initial AlexNet architecture. We then tried a few experiments training these architectures with Stochastic Gradient Descent as the optimizer however, accuracy was never much greater than around 8% even after approximately 50 epochs. Upon closer examination (training with a slightly higher learning rate lead to no decrease in the loss function at all), it seemed as though the model was suffering from vanishing gradients due to the ReLU activation functions. Therefore, it was decided to add two batch norm layers after each linear layer and initialise the weights of the convolutional and linear layers with Kaiming normal(9) and Xavier normal(10) respectively. The final tweak made was to change the optimizer to Adam(10). This lead to significantly greater accuracy in much fewer epochs than the initial experiments.

The fine-tuning of the pre-trained DenseNet was more straightforward. The steps when preparing our network for transfer learning were as follows:

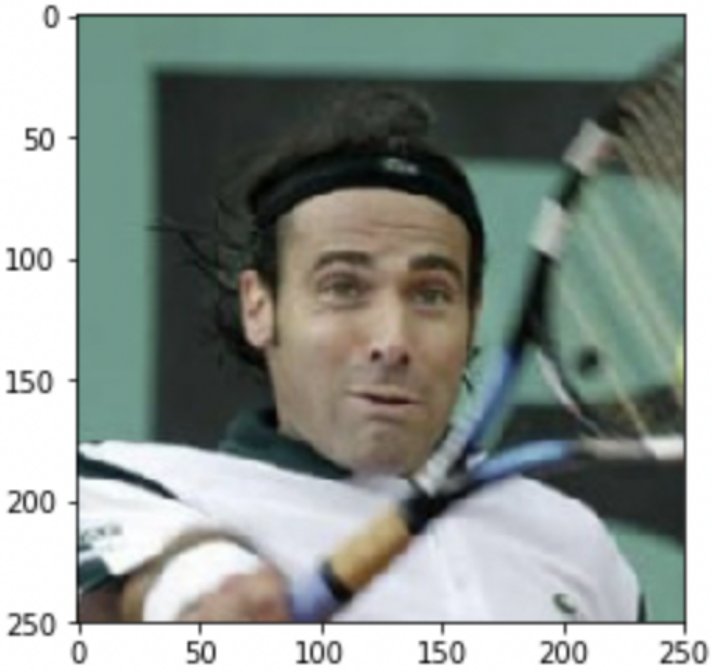
1. Download pre-trained model.
2. Disable gradient calculations for pre-trained layers (we don't need to backpropagate gradients throughout the entire network, only through our fine-tuning layers).
3. Append fine tuning layers (one fully connected layer with output size equal to the number of classes).
4. Create a list of the trainable parameters to pass to our optimizer.

Evaluation

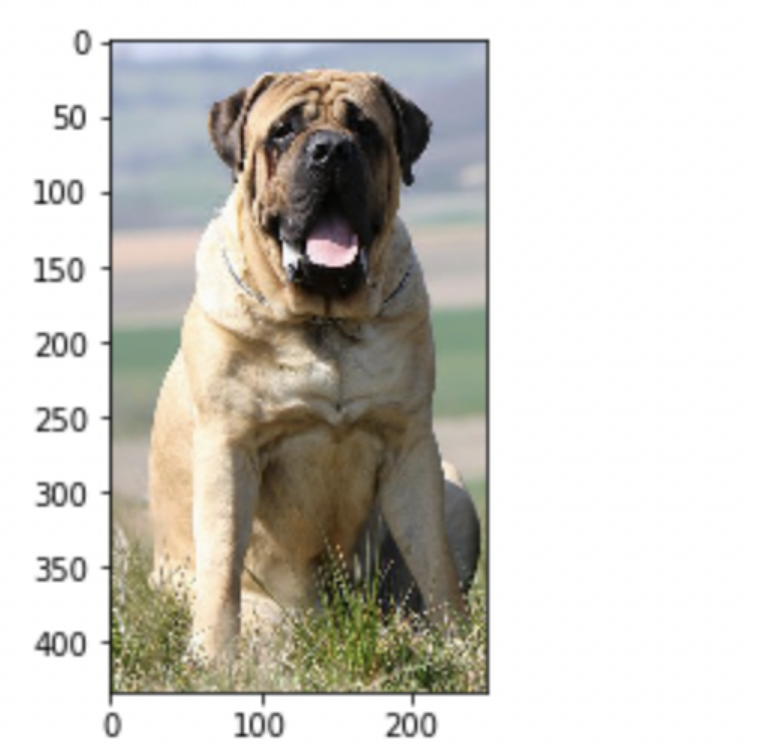
The fine-tuned DenseNet was able to achieve 82% accuracy after only 10 epochs. For our purposes this was deemed to be more than sufficient. A qualitative evaluation of our application would suggest a few potential improvements;

- Improve the accuracy of the dog breed classifier, this could be done by training for more epochs, fine tuning more layers or potentially using a different loss function weighted by class.
- The human face detector could be improved by using a more advanced deep learning based approach.

- The algorithm itself could have improvements made to the user experience. One idea might be to augment the current algorithm with a generative model that converts the input human image to its most likely dog breed(12).



This human reminds me of a certain dog breed....
Dachshund! That's the one!



Nice dog, I think you're a...
Mastiff! That's the one!

References

1. Udacity GitHub repository with notebook template; https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification.
2. Simonyan, K. and Zisserman, A. Very Deep Convolutional Neural Networks for Large Scale Image Recognition. (ICLR 2015) https://arxiv.org/pdf/1409.1556.pdf.
3. Viola, P. and Jones, M. Robust real-time face detection. (International Journal of Computer Vision 2004) https://www.face-rec.org/algorithms/boosting-ensemble/16981346.pdf.
4. OpenCV; https://docs.opencv.org/3.4/index.html.
5. Huang, G. Liu, Z. van der Maaten, L. and Weinberger, K. Densely Connected Convolutional Neural Networks; https://arxiv.org/pdf/1608.06993.pdf.
6. Ayanzadeh, A. Vahidnia, S. Modified Deep Neural Networks for Dog Breeds Identification; https://www.researchgate.net/publication/325384896_Modified_Deep_Neural_Networks_for_Dog_Breeds_Identification.
7. PyTorch; https://pytorch.org.
8. Zhang, K. Zhang, Z. Li, Z. and Qiao, Y. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks; https://arxiv.org/abs/1604.02878.
9. He et. al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification (ICCV2015); https://openaccess.thecvf.com/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf.
10. Glorot, X. and Bengion, Y. Understanding the difficulty of training deep feedforward neural networks; http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.
11. Kingma, D. Ba, J. Adam: A Method for Stochastic Optimization; https://arxiv.org/abs/1412.6980
12. https://www.vice.com/en/article/884wek/ai-algorithm-turns-humans-into-animals