Capstone Project

Mohamed Gamal

Machine Learning Engineer Nanodegree

Aug 31th, 2020

Definition

Project Overview

If dog owners are unsure about their dog's breed, a dog-breed classifier can help them identify their dog's breed. This would be especially helpful for inexperienced dog owners. If a dog owner have doubts about the breed before buying a dog, i.e. there is a discrepancy between what the seller says and what you remember about the breed, you can quickly confirm your doubts with a dog breed classifier . This is possible via the image classification in the area of deep learning, which is part of machine learning. Image classification refers to a process in computer vision that can classify an image according to its visual content. A convolutional neural network, CNN for short, is preferably used here. CNN are a specialized type of neural network for processing data that have the structure of a grid. Image classification with CNN saw the light of day in 1994 with Yann LeCun's LeNet5 and received a huge boost in 2012 with AlexNet by Alex Krizhevsky who won the ImageNet competition.1 In this project I created a CNN model that can classify dog breeds. A dataset from Udacity is available for training the model, which includes more than 8000 images which is not the largest data set but it is ok for our purposes.

Problem Statement

In this project it is important to build a data processing pipeline to classify real-world, user-supplied images. On the basis of a image of a dog, an algorithm is supposed to give an estimate of the breed of the dog. If the image of a person is given, the algorithm should reproduce the most similar dog breed. In addition, an accuracy of 60 percent or greater should be achieved. To achieve this goal transfer learning is used . In transfer learning approach feature part of the network is freezed and only classifier is trained. Also a web application is being developed to facilitate access for end users. The following therefore applies: It must be recognized on an image whether a person or a dog is present. If neither of the two is the case, an error message is issued The breed of the dog must be estimated An accuracy of 60 percent or greater should be achieved Transfer learning will be used to reach this goal.

Metrics

The data is split into train, test and valid dataset. The model is trained using the train dataset. We use the testing data to predict the performance of the model on unseen data. We will use accuracy as a metric to evaluate our model on test data. Accuracy=Number of items correctly classified/ All classified items Also, during model training, we compare the test data prediction with validation dataset and calculate Multi class log loss to find the best performing model. Log loss takes into the account of uncertainty of prediction based on how much it varies from actual label and this will help in evaluating the model.

Analysis

Data Exploration

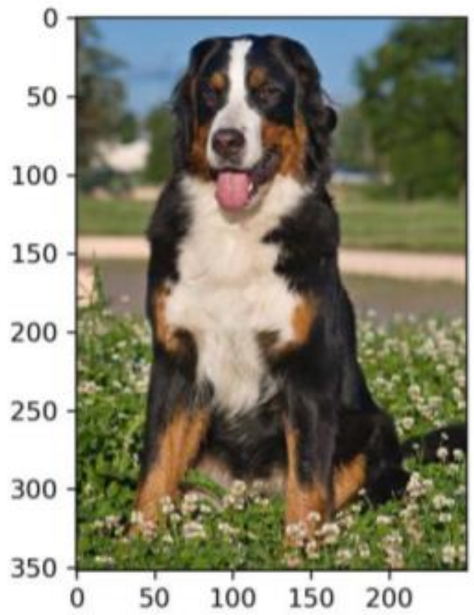
The data set provided by Udacity has the following:
13233 Human images
8351 Dog images

2.1.1 The dog dataset:

The dataset contains 8,447 dog pictures, which are subdivided into training, test and validation data. So we have 6,756 images to train the model, 846 images for testing and 845 images for validation. The entire dataset contains 134 different breeds from the Afghan dog to the Yorkshire Terrier. Here are two examples of very nice dogs from the dog dataset: Chihuahua Bernese Mountain Dog All pictures come in a .jpg format. The images have a mean of 565.85 pixel in width and a mean of 527.64 pixel in height.

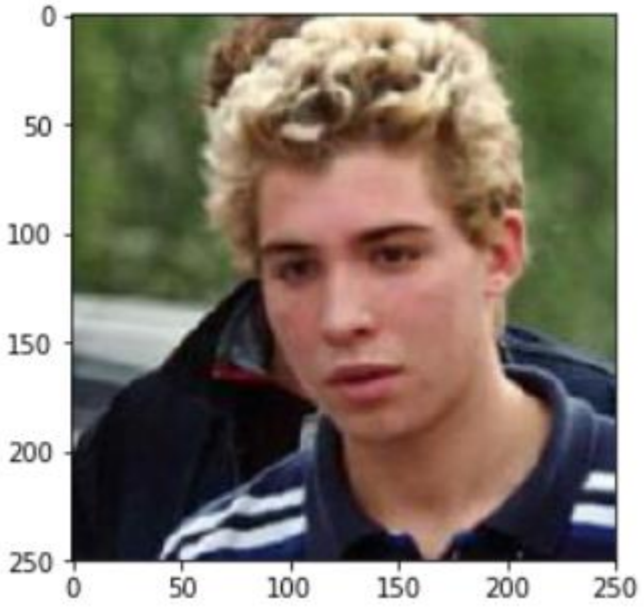
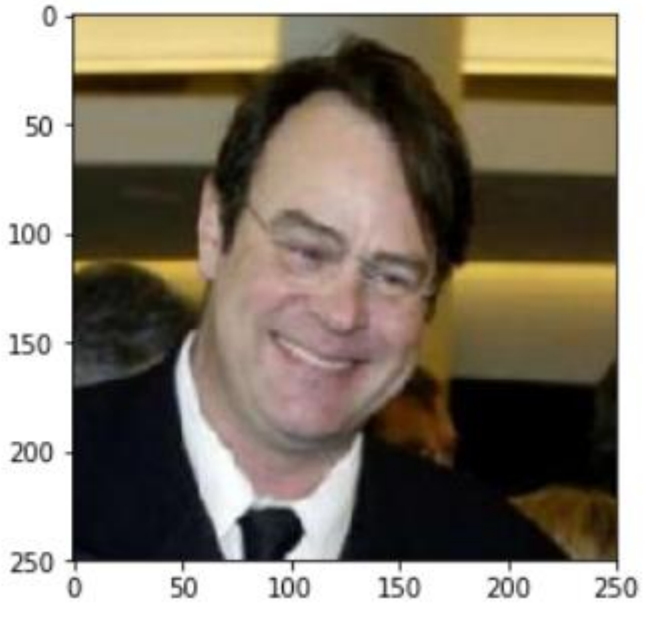
Here are two examples of very nice dogs from the dog dataset:

Chihuahua


Bernese Mountain Dog

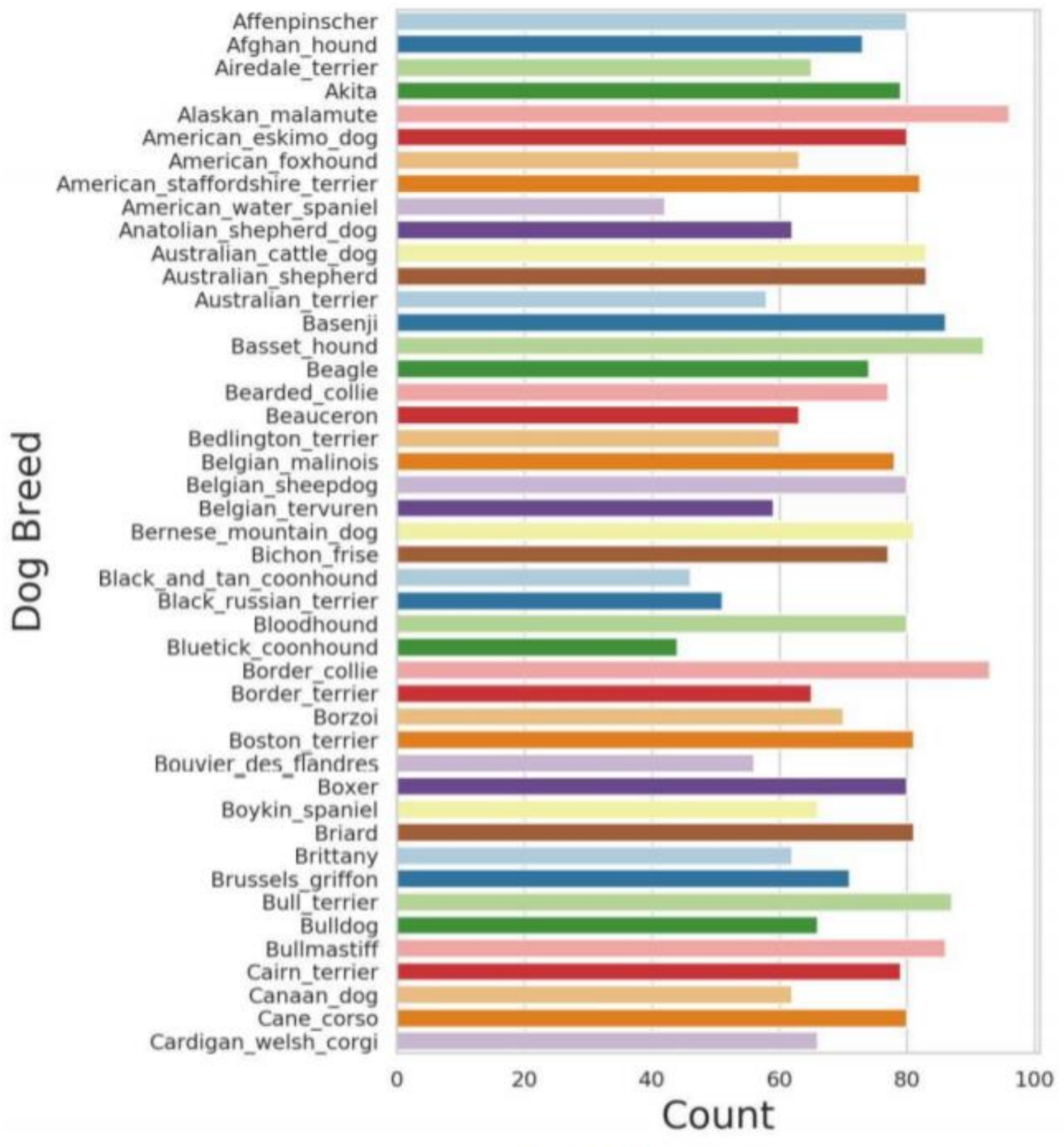
2.1.2 The human dataset:

This dataset contains 13233 ordinary images of more or less ordinary people. These images are used to test the performance of the Human Face Detector. This will be the only use of these pictures in my project. Here are some sample images from the human dataset:

Exploratory Visualization

In this section we will explore the data and I will also provide plot graphics to get a feel for what is contained in the dataset and how. So that you can get an idea of what the dataset contains, I will show you a part of the distribution of the dog breeds over the complete dataset:



Dog Breed Distribution

Algorithms and Techniques

For performing this multiclass classification, we can use Convolutional Neural Network to solve the problem. A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The solution involves three steps. First, to detect human images, we can use existing algorithm like OpenCV's implementation of Haar feature based cascade classifiers. Second, to detect dog-images we will use a pretrained VGG16 model. Finally, after the image is identified as dog/human, we can pass this image to an CNN model which will process the image and predict the breed that matches the best out of 133 breeds. Benchmark The CNN model created from scratch must have accuracy of at least 10%. This can confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Benchmark

The CNN model created from scratch must have accuracy of at least 10%. This can confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Methodology

Data Preprocessing

First my data loaders apply chained transformations to the loaded images. I decided to create two transforms:

- one for validation and test with image resized and croped to 224x224, and then normalized;
- one for training with some **data augmentation: random image rotation, random resized crop and random horizontal flip**

The size of 224x224 was initially chosen to have my data loaders compatible with the transfer learning task (latter in this notebook) as I initially planned to reuse VGG-16 as a starting point. Finally I found a better model and I could have changed the size to 256 or another. I have also tried with smaller size of 32 and 64, but I can achieve better results with larger image without performance impact (at least with a GPU, because without a GPU and a size of 224, even the first epoch had reached my patience limit).

Then I initially started with 3 convolutional layers and **2 fully connected layers**. My kernel size is fixed at 3x3 and padding at 1, because it's a common configuration and it has the benefits of keeping the same width and height. Each hidden layer has a **ReLU activation** function. It has proven its benefits over sigmoid on deep networks in hidden layers. After each convolutional layer, I reduce the dimensions by 2 (height and width) with a **max pooling layer**.
Before the fully connected layers, we need to flatten our tensor from 3D to 2D. There's a **dropout layer** applied to this result to help avoid overfitting to the training data.

Then comes the first fully connected layer with 1024 neurons, with ReLU activation and dropout. 1024 was intermediate size between the output of my flatten tensor and my output layer that must have 133 neurons because we are classifying 133 dog breeds.

During the first training, the model was learning, but slowly and reached 11% of accuracy after 30-40 epochs.
I then tried to add a fourth convolutional layer, and reached higher score after 6-10 epochs. And then my final architecture has **5 convolutional layers** because I still got better results with a fifth conv layer.

I also tried to add a third, or even keep a single fully connected layer but it did not train very well.
I finally found that adding **batch normalization** after each layer speed up training and have better results with less epochs! The final score gives me a **27% accuracy on the test set, after 15 epochs**. I could have trained it longer and reached better results as the validation loss is still decreasing after 15 epochs.

Using transfer learning:

I first tried to reuse **VGG-16** because it's a **pretrained model** that achieved great results in ImageNet, without having too many layers.
The model is already trained to detect dogs, and **already classify a lot of dog breeds**. So it can detect the features we need for our problem.

We just need two modifications:

- change the model so that it does not train these features anymore: requires_grad=False
- and then **replace the last fully connected layer (the classifier itself) so that it outputs 133 classes** instead of 1000.

I achieved great results from the very first epoch and quickly reached 75-80% of accuracy after 10 epochs.
I then decided to challenge myself to even go further. I selected **DenseNet-161** as a good model to try because it has **improved performance over VGG-16 on ImageNet** and it introduces an interesting architecture with **each layer taking all preceding feature-maps as input**. Moreover despite it has more layers than VGG-16, the time to train a single epoch is very similar.
And I indeed got higher scores and faster. It reaches 79% of accuracy after the first epoch, around 88% after five epochs and finally reached **88% of accuracy** after 15 epochs.

I consider the final result to be very good. As a human I couldn't reach 88% of accuracy while classifying dog breeds among 133 classes.

Implementation

I have built a CNN model from scratch to solve the problem. The model has 3 convolutional layers. All convolutional layers have kernel size of 3 and stride 1. The first conv layer (conv1) takes the 224*224 input image and the final conv layer (conv3) produces an output size of 128. ReLU activation function is used here. The pooling layer of (2,2) is used which will reduce the input size by 2. We have two fully connected layers that finally produces 133-dimensional output. A dropout of 0.25 is added to avoid over overfitting.

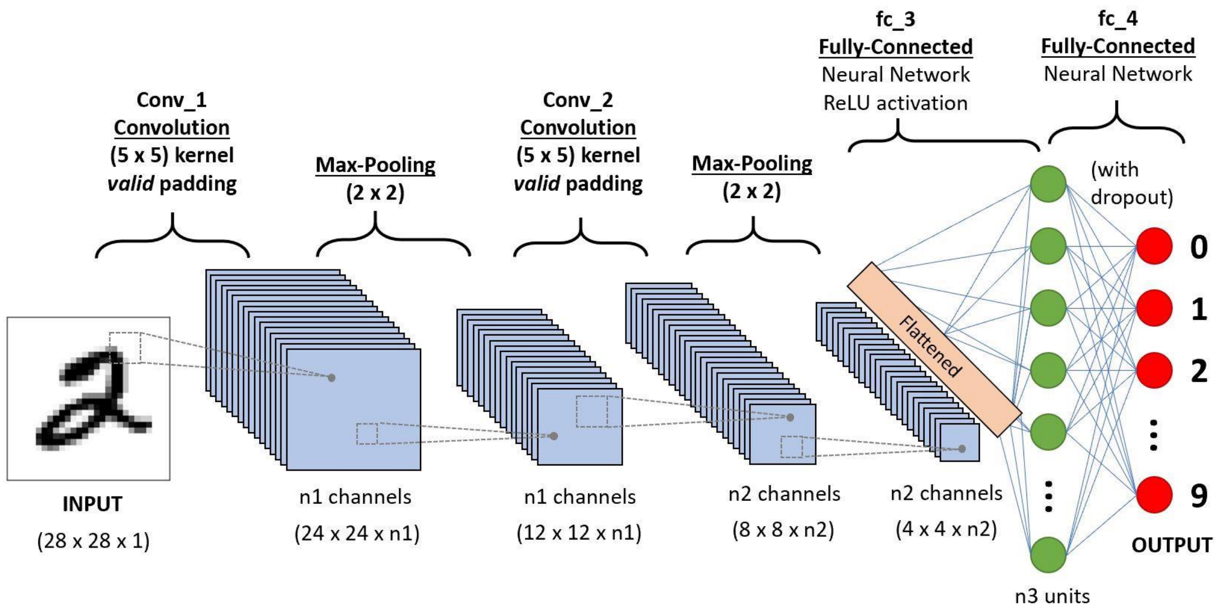


Figure 1An illustrative figure for how a CNN works

Refinement

The CNN created from scratch have accuracy of 13%, Though it meets the benchmarking, the model can be significantly improved by using transfer learning.To create CNN with transfer learning, I have selected the **DenseNet-161** architecturewhich is pre-trained on ImageNet dataset, the architecture is 101 layers deep. Thelast convolutional output of **DenseNet-161** is fed as input to our model. We only need to add a fully connected layer to produce 133-dimensional output (one for each dog category). The model performed extremely well when compared to CNN from scratch. With just 15 epochs, the model got 88% accuracy.

Results

Model Evaluation and Validation

Human Face detector: The human face detector function was created using OpenCV's implementation of Haar feature based cascade classifiers. 98% of human faces were detected in first 100 images of human face dataset and 17% of human faces detected in first 100 images of dog dataset.

Dog Face detector: The dog detector function was created using pre-trained VGG16 model. 100% of dog faces were detected in first 100 images of dog dataset and 0% of dog faces detected in first 100 images of human dataset.

CNN using transfer learning: The CNN model created using transfer learning with DESNet101 architecture was trained for 15 epochs, and the final model produced an accuracy of 88% on test data. The model correctly predicted breeds for 680 images out of 836 total images. Accuracy on test data: 88% (736/836)

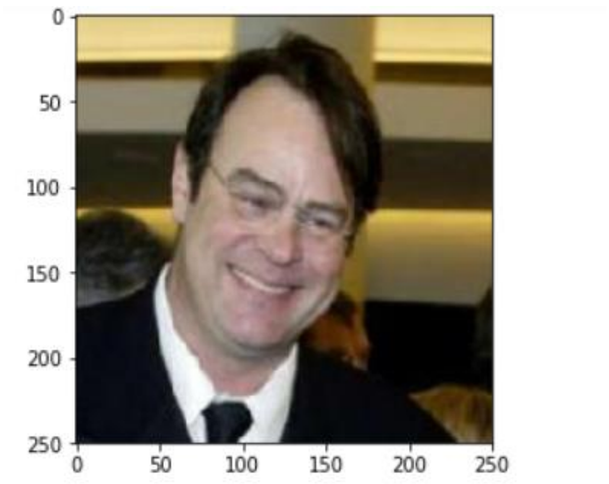
Justification

I think the model performance is better than expected. The model created using transfer learning have an accuracy of 88% compared to the CNN model created from scratch which had only 29% accuracy.

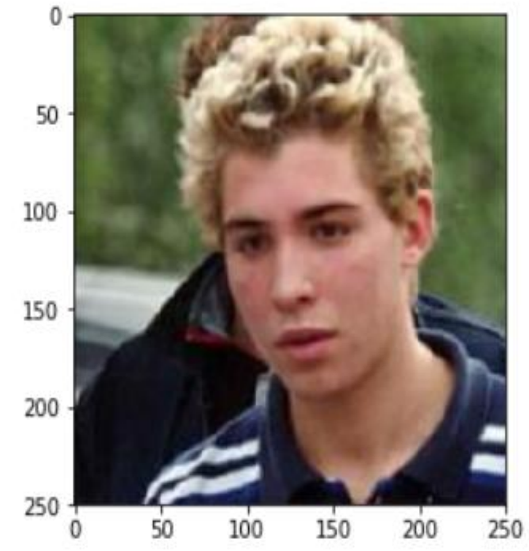
Conclusion

Free-Form Visualization

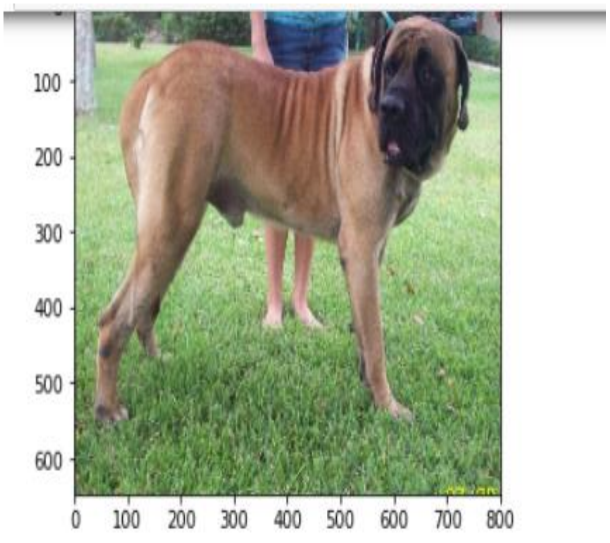
On this occasion I would like to show some outputs of the developed function to predict the breed:



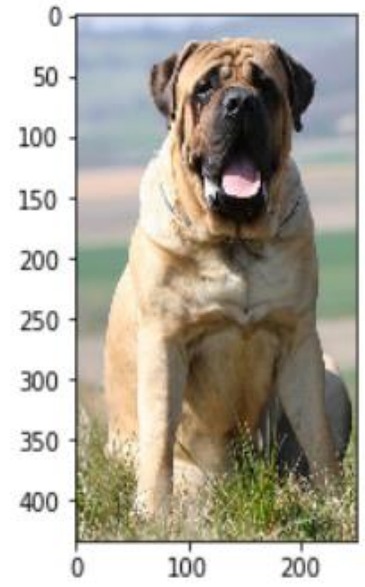
```
Hello human
You look like a Kerry blue terrier
```



```
Hello human
You look like a Dachshund
```



```
Hello dog
I predict you are a Mastiff
```



```
Hello dog
I predict you are a Mastiff
```

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant public dataset of images were found
2. The images was downloaded
3. A Human Face Detector was developed
4. A dog detector was developed
5. The images were preprocessed and data augmentation was applied
6. A Model from Scratch was developed, trained and tested
7. A Model with transfer learning was developed, trained and tested
8. An application was designed and developed to show the user the top 3 predictions for a given image
9. A website was designed and developed to show the user the top 3 predictions for a given image
10. Every function and every part of the project has been clearly documented.

Improvement

Here is a list of possible improvement that could be tried to reach better results:

- more data augmentations techniques: randomly shift image, randomly change the brightness, contrast and saturation, randomly convert image to grayscale...
- find more images: 8351 dog images for a 133 classes problem, it's not so many image to train.
- add yet another 6th convolutional layer as I got better results with 4th and 5th layer
- give a try to other transfer learning models : resnet, Inception v3...
- fine tune learning rate and other optimizer hyper parameters to speed up gradient descent
- give a try to other optimizers: Adam, Adamax, Adadelta...
- train for more epochs as validation loss is still decreasing
- try different batch_size to see how it impacts training
- display images where the model fails to understand why it fails and maybe improve algorithm, data augmentation (rework image and do not crop image randomly but where the dog lies)