Capstone Project Report

Dog Breed Classifier
Machine Learning Engineer Nanodegree

Muhammad Ovais **3rd Dec, 2020**

Problem Background

Classifying dog breeds from images is a fairly challenging classification problem due to the significant overlap of features found in different dog breeds. As a result, even experienced humans sometimes fail to differentiate between certain breeds, e.g. a Brittany and a Welsh springer spaniel.

Since the start of this decade, researchers have been using this particular problem as the benchmark for the evaluation of their novel multi-classification approaches. (Liu, et al. 2012) succeeded to achieve 67% recognition rate on this dataset using their Part Localization Technique. Another faster approach on a smaller dataset was proposed in (Prasong and Chamnongthai 2012) using the size and position of key features in images and applying PCA. Our approach, however, will be different as we will be taking advantage of deep neural networks to achieve a more accurate, yet computationally intensive solution.

A convolutional neural network, or CNN, is a deep learning neural network designed for processing structured arrays of data such as images. Convolutional neural networks are widely used in computer vision and have become the state of the art for many visual applications such as image classification. [1]

Problem Statement

The goal is to make such a CNN-based classifier that performs two main tasks:

- ***Detect Dog breeds:*** If a dog's image is provided, it should be able identify its breed.
- ***Detect resemblance between human and dog faces:*** In case a human's face is detected, it should output the dog's breed closest to the human's face in resemblance.

Dataset Exploration

The human dataset can be obtained from this link: Human dataset.

The dog breed dataset is a subset of ImageNet dataset and can be obtained from this link: Dog dataset.

The human dataset contains 13233 total images divided into 5750 folders each named after the name of corresponding person. All the images are of size 250*250 pixels.

[1] Convolutional Neural Network (https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network)

The dog dataset consists of 8351 total images divided into 3 folders – train (6680), test (836) and validation (835) images. Each of these folders is further divided into 133 subfolders named after respective dog breeds. The size of the images is not uniform.

An important insight to remember is that none of the above datasets are balanced—containing varying number of images for different dog breeds and human classes.

Sample images from the dataset

Project Workflow:

The project has been implemented in the following steps:

- Step 0: Import Datasets – datasets can be downloaded from the above links.
- Step 1: Detecting human faces
- Step 2: Detecting dog faces
- Step 3: Creating a dog breed classifier by building a CNN from scratch.
- Step 4: Creating a dog breed classifier using transfer learning.
- Step 5: Writing an algorithm for the final app.
- Step 6: Testing the algorithm on new unseen images.

Step 1: Detecting human faces

Algorithm Specifications:

For human face detection, we will be using OpenCV's pretrained **Haar cascade classifier**. Haar Cascade is a machine learning object detection algorithm used to identify objects in an image or video and based

on the concept of features proposed by (Viola and Jones 2001). A very detailed explanation of this algorithm can be found on here. [2]

Performance Evaluation:

After testing on a short testing subset of 100 human and dog images each, the model's performance summary is as follows:

Human faces detected: 98%
Dog faces detected: 17%

Step 2: Detecting dog faces

Data Preprocessing:

For this step, the images are resized and cropped to size (224,224), converted into tensor and normalized with parameters given in the PyTorch documentation.[3] which is the default input tensor parameters for the CNN model we will be using.

Algorithm and Technique:

For the detection of a dog face in an image, we used a pretrained Convolutional Neural Network (**VGG-16**). VGG16 is a convolutional neural network model proposed by (K. Simonayan 2014) and has a very high accuracy on the ImageNet dataset.

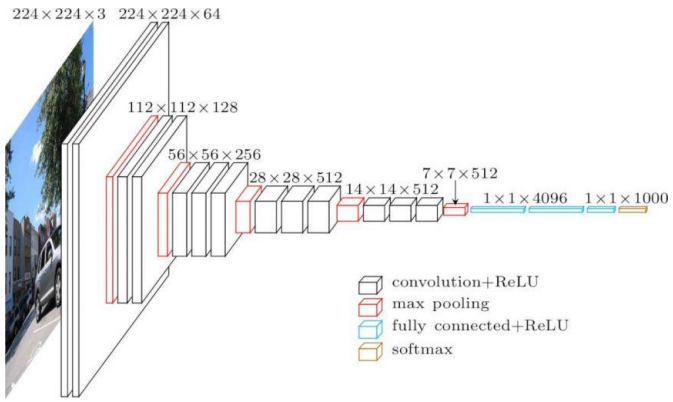


Figure 1 VGG16 Architecture

Performance Evaluation:

The pre-trained model for dog face detection performed with absolute accuracy.

human_files_short: 0%
dog_files_short: 100%

[2] Deep Learning Haar Cascade Explained (http://www.willberger.org/cascade-haar-explained)
[3] Torchvision.Models (https://pytorch.org/docs/stable/torchvision/models.html)

Step 3: Creating a dog breed classifier by building a CNN from scratch

Data Preprocessing & Loading:

Again, the data is preprocessed like before but with certain modifications to assist training:

- **Data Resizing:** All the images are resized to (256,256) and then cropped to (224,224) because this is the default size of input tensors that the initial VGG16 model was originally trained on. The tensors are also normalized with mean and std values prescribed by the PyTorch documentation.
- **Data Augmentation:** The train dataset is also augmented through RandomHorizontalFlip with p=0.2 and RandomRotations of 20 degree and RandomCrop to introduce some complexity in the training data to avoid overfitting.

After the above preprocessing, loaders for the training, validation and test datasets are defined and saved to be used by the model.

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%.

Evaluation Metrics:

- **Loss Function:** Cross-Entropy loss due to its ability to learn efficiently on multidimensional classification datasets with imbalance.
- **Optimizer:** SGD or Adam, both optimizers are suitable for our problem so we'll experiment and see which one works better on our model.

Model Development:

The model development is a very tricky and time consuming step. But can be simplified by starting with a small CNN architecture with only one hidden layer and then optimizing the same architecture or adding one or more layers to model.

At first I tried a 3-layer CNN (3->32->64->128) with each conv2d layer of kernel_size=(3,3) and stride = (1,1) and 2D-maxpooling with stride=(2,2). 2 FC layers at the end, ReLU as activation function, and Adam as Optimizer Function with lr=0.0003.

This model gave me only 7% accuracy after 20 epochs.

Then I experimented with adding an additional convolution layer to see if that improved accuracy. To my dismay, the model's validation loss kept at nearly 4.8 after 3 epochs and I gave up on the training.

Further change to this architecture did not give the desired results and I went back to previous architecture with some modifications.

Final Model:

The final model is a modification of the first model that I tried but with some modifications.

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  F.relu()
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  F.relu()
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  F.relu()
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

  (drop): Dropout(p=0.3)
  (fc1): Linear(in_features=6272, out_features=512, bias=True)
  F.relu()

  (drop): Dropout(p=0.3)
  (fc2): Linear(in_features=512, out_features=133, bias=True)
  F.relu()
)
```

Figure 2 Model from scratch

Explanation of the architecture

The first and second convolutional layers have kernel size = 3 and stride = 2. This downsizes the input tensor (224*224*3) by half in each con2d layer. Both followed by a maxpooling with stride = 2, which further downsizes the tensor. The 3rd conv2d layer has stride=1 so this does not reduce the h and w of tensor. Then with the last maxpooling with stride =2, resulting in the total downsize of image by a factor of 32.

The tensor is then passed to FC layers: (7*7*128->512) and (512->133) with droput = 0.3 to avoid overfitting and relu as activation function. The function of FC (or fully-connected) layers can be thought of as a nonlinear PCA, it rectifies the "good" features and diminishes the others via learning the full set of weights. For further understanding of FC layer function, see this. [4]

Performance Evaluation

```
Test Loss: 3.789777
Test Accuracy: 13% (114/836)
```

This accuracy greater than the benchmark accuracy of 10%, so the model is accepted.

[4] https://stats.stackexchange.com/questions/182102/what-do-the-fully-connected-layers-do-in-cnns)

Step 4: Creating a dog breed classifier using transfer learning.

In this step, we make use of pre-trained CNN architectures on the ImageNet dataset and further train the model for our specific subset of 133 classes.

Data Preprocessing & Loading:

As before, the images are first resized to (224,224) for architectures like VGG16, Resnet50, etc. and, if needed, to (299,299) for Inceptionv3 architecture. The data augmentation is the same as previous model. The data loaders are then defined and saved in a dictionary for use by our model as before.

Benchmark:

The result of our final model built using transfer learning must have at least 60% accuracy.

Model Development:

Choosing a better pre-trained model can be tricky all of the above mentioned CNNs have been trained on the same ImageNet dataset. So, the best way to decide is to see literature on the comparison of these architectures. This article gives a very insightful comparison of 10 best CNNs. [5] The following table has been obtained from there:

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
InceptionV3	92 MB	0.779	0.937	23,851,784	159
ResNet50	98 MB	0.749	0.921	25,636,712	-
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
ResNeXt50	96 MB	0.777	0.938	25,097,128	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

Figure 3 Source: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

As evident from the table, the InceptionV3 model gives very good values of Top-1 and Top-5 Accuracy while consuming less memory than VGG16. ResNet50 and others also seem impressive too.

Choosing InceptionV3:

We selected the InceptionV3 model based on previous observations. Let's look at its architecture:

[5] https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

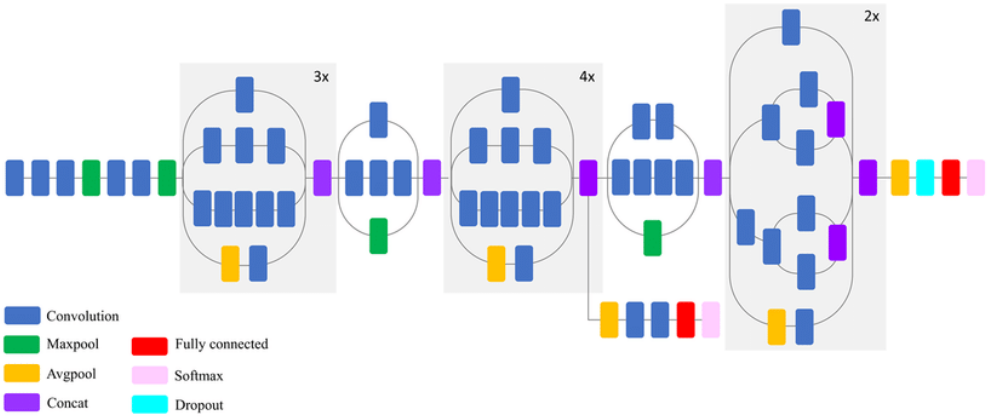


Figure 4 InceptionV3 Architecture

Printing the model's architecture we see that the final FC layer gives out 1000 outputs. We'll redefine the layer as torch.nn.Linear(2048,133,Bias=True) for our case of 133 classes.

Training this model on our training data gives promising loss values in the vicinity of 0.3-0.4.

Performance Evaluation:

```
Test Loss: 0.457515
Test Accuracy: 85% (718/836)
```

This accuracy is way above the benchmark 60% accuracy, so the model is accepted.

Challenges in Model Development:

My first approach, therefore, was to try out InceptionV3. However, I realized that the implementation of InceptionV3 varied significantly from other CNNs like VGG and ResNet, and fell in a loop of errors. One particular error was "TypeError: log_softmax(): argument 'input' (position 1) must be tensor, not tuple" which took me a long time to resolve.

Meanwhile, I resorted to using ResNet50, while using previous data loaders from model_scratch part. But for some reason, the valid_loss seemed to be stuck at >1 after 5 epochs and didn't seem to improve. I could have optimized this model to get the desired result but I had found the solution to the Inceptionv3 error in the meanwhile!

After reading this discussion I found out that the InceptionV3's output by default by default comes as a tuple containing an additional "aux_outputs" from the auxiliary branch (hence the TypeError). Assigning aux_logits=False solves the issue and the final InceptionV3 model works like a charm!

Justification for InceptionV3:

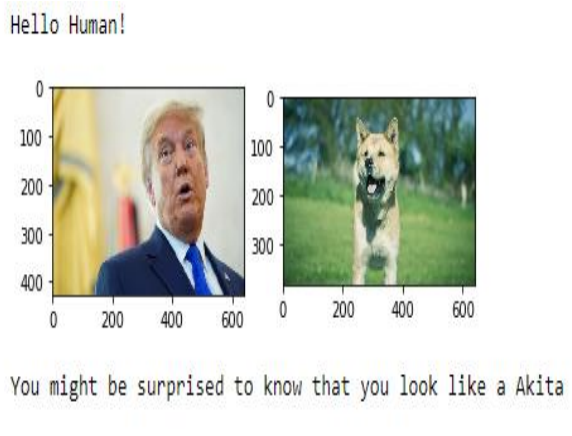
Inceptionv3 seems to be more suitable for this problem and other subsets of ImageNet dataset than other architectures especially VGG (which takes up about 5 times as much memory as Inceptionv3). It introduces an auxiliary approach rather than simply a sequential approach which helps optimize the learning process faster. It is also lighter on memory and has a high top-1 and top-5 accuracy.

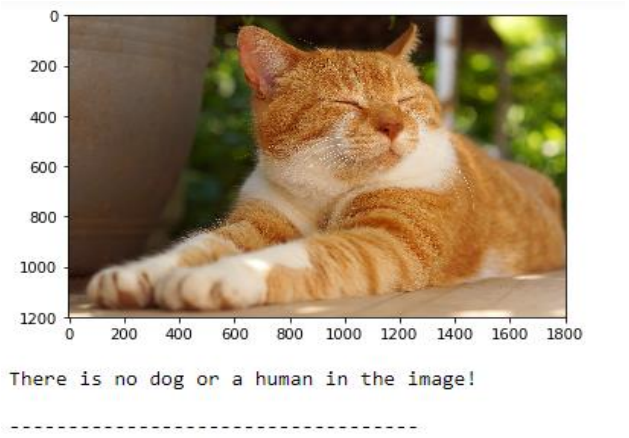
Step 5: Writing an Algorithm for the Final App

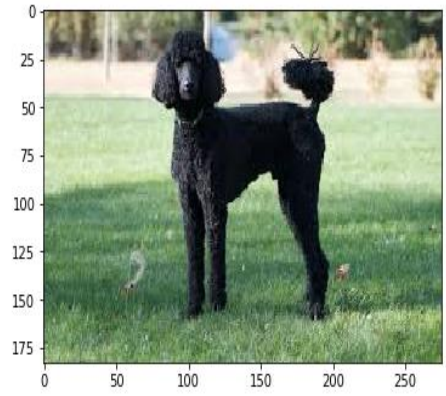
The algorithm for the app is quite simple:

- Take an image path as input.
- Check if the image is of a human. If true, print the resembling dog breed's name while displaying both the human's and the resembling dog's image.
- If false, check if the image is of a dog. If true, print the predicted dog's breed.
- If false, print that the object is neither a human nor a dog.

Step 6: Testing the model on unseen images

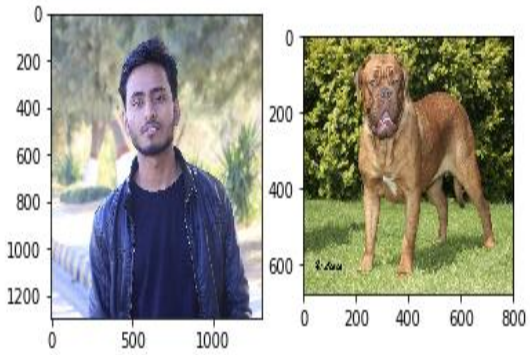






This dog seems to be a Poodle

Hello Human!



You might be surprised to know that you look like a Dogue de bordeaux

Further Improvements

Although the model performs quite well on the given problem set, further improvement can be made to make it more generic and robust. At one point it detected a cat's image as a human face. Following improvements might help:

- There is a room for improvement in the human face detection part, since it is prone to false positives (detected 17% of dog images as humans and a cat image as human during testing. We should try data augmentation on human dataset too.
- Applying a pretrained deep learning architecture on face detection could make our model more robust and overcome the false positive situation.
- A more thorough hyperparameter tuning (possibly with AWS Sagemaker's hyperparameter tuners) could further improve our model's performance on confusing images.
- The algorithm currently works on pure breeds (breeds that have not been mixed with other breeds). An improvement could be if the model could detect a mix-breed and return the two highest percentage scores of dog's corresponding breeds.

References

K. Simonayan, A Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." *ILSRVC*, 2014.

Liu, J, A Kanazawa, D Jacobs, and P Belhumeur. "Dog Breed Classification Using Part Localization." *Computer Vision – ECCV 2012.* Florence, Italy: Springer-Verlag Berlin Heidelberg, 2012.

Prasong, P, and K Chamnongthai. "Face-recognition-based dog-breed classification using size and position of each local part, and PCA." *9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology.* Phetchaburi, 2012. 1-5.

Viola, Paul, and Michael Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features." *IEEE*, 2001.