Machine Learning Engineer Nanodegree

Capstone Project

Ikechukwu Nigel Ogbuchi
February, 2020

I. Definition

Project Overview

Lots of strides has been done in the field of image recognition over the last few years. We are looking to build a model that easily identifies the breed a dog belongs, based only on a given image.

Computer vision is the field of Artificial intelligence that this project comes from. Computer vision enables computer gain a high level understanding of images and videos.

A convolution neural network is a special deep learning structure that allow create new features, like edges. This bring a lot of the abstraction and allow to the model adjust the parameters that are needed to create a good model. However, that requires a lot of computational power that means hours or may be days of training. GPU's made it easy to enhance the training of the CNNs. The develop of this field has allowed to improve different systems, like systems that can find images related to a search, several image recognition systems that are used to identify people, systems that can identify disease in x rays images, etc. In particular, in this project we built an algorithm that first determines if a photo is human or dog and then goes ahead to predict what model the dog is(in the case when a dog recognized).

There are millions of dog owners throughout the world. The European Union has an estimated dog population of 85 million. With over 300 breeds of dogs, it is difficult for dog owners to be aware of the different breeds in existence or the breed of their own dog. The importance of breed awareness could arise when considering a buying a dog, toys, food or medical visits to the local veterinary physician. Therefore, each owner should at least know the breed(s) of their dog(s).

Problem Statement

This is the problem: Given an image of a dog, our algorithm should be able to l identify the dog's breed. If supplied an image of a human, the code will identify the resembling dog breed."

The aim of the project is to build a pipeline to process real-world, user-supplied images. The algorithm will identify an estimate of the dog's breed given an image. If the image is of a human, the algorithm will choose an estimate of a dog breed that resembles the human. If neither a dog nor a human is detected, then an error message is output. Therefore, the models in place should be capable of detecting a dog or human in an image, classify the dog to its breed and classify a dog breed that the human resembles.

There are several factors that make the problem of dogs categorization challenging. Firstly, many different breeds of dogs exist, and all breeds on a high level look alike, i.e. there can be only subtle differences in appearance between some breeds. In fact, a dog's breed might be not obvious right away even for people who have an expertise in the domain. Therefore, determination of dog breeds provides an excellent domain for fine grained visual categorization experiments. Secondly, dog images are very rich in their variety, showing dogs of all shapes, sizes, and colors, under differing illumination, in innumerable poses, and in just about any location. The photos have different resolutions, backgrounds, and scales. On some images the dogs are partially covered by other objects or wear clothes such as hats, scarfs, glasses. Thus, there is a lot of noise on many of the photos that makes the problem more challenging. Thirdly, our data set is small in terms of the number of photos per breed. If humans want to distinguish a dog from a cat, or from some other object, then they need to look at things like ears, whiskers, tails, tongues, fur textures, and so forth. But none of these features are available to a computer, which receive only a matrix of independent integers as an input. Models such as Convolutional Neural Networks (CNN) allow computers to automatically extract hierarchies of features from raw pixels. These techniques proved to be successful in a variety of visual analysis tasks. Using these techniques, we can create a dog-breed classifier and allay that to solve our problem.

Metrics

For this project, we will measure all models with an accuracy score metric. The breed model is required to have almost 60% of accuracy

Accuracy will be the main metric used to test both the benchmark model and the solution model. This is so because the problem at hand is a classification task, where the model should classify the images accurately.

The goal of what we’re looking to do here is pretty simple: we want to see how well we can do at classifying breeds of dogs. Accuracy will be able to tell us in a simple and easy-to-understand way how well our deep learning classification model is performing in this regard.

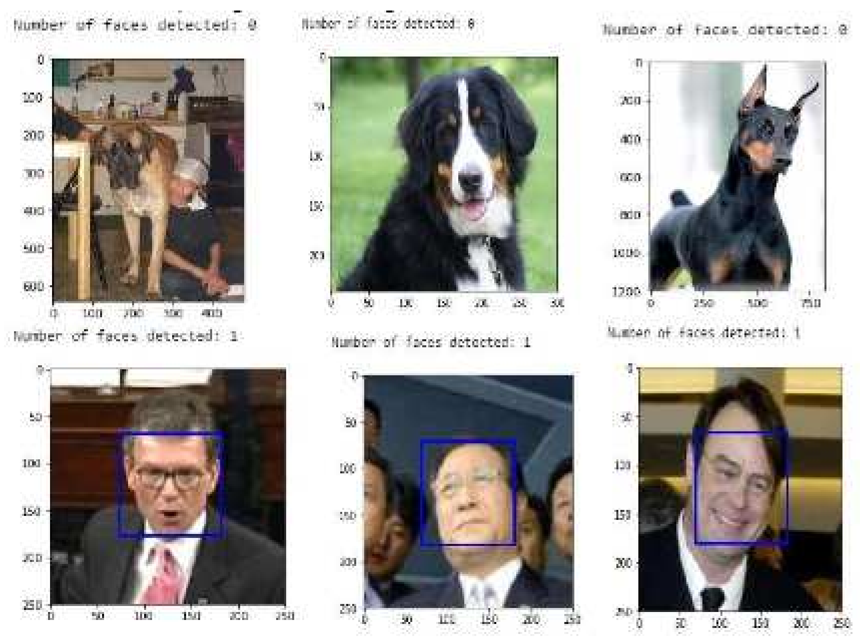
II. Analysis

The human dataset contains 13233 images of humans, first names and last names. The dog dataset contains 8351images of dogs, 133 breeds and each has a representation of 8 images. The images in both datasets have a size of approximately 400 by 400 but will be resized when classifying dog breeds from scratch.

Data

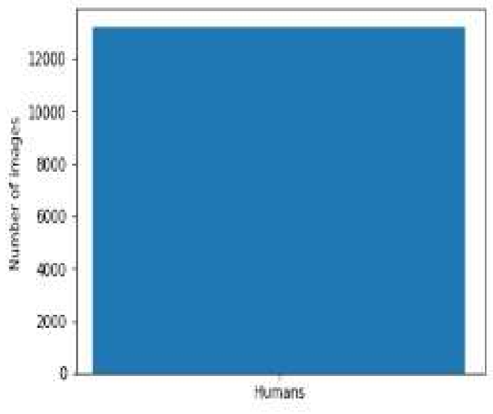
- The dataset includes 13233 images of humans and 8351 images of dogs.
- Every image of humans is of size (250,250,3) while the images of dogs do not have a constant size.
- 6680 images belong to the train data for dogs.
- Found 835 images belong to the validation data for dogs
- Found 836 belong to the test data for dogs.

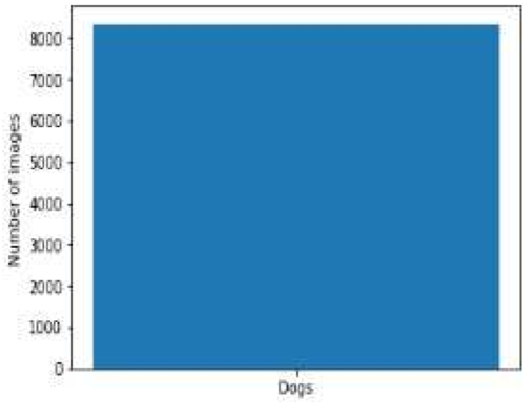
Below are sample images from the distribution:


Number of faces detected: 0
Number of faces detected: 0
Number of faces detected: 0
Number of faces detected: 1
Number of faces detected: 1
Number of faces detected: 1

Exploratory Visualization

The given dataset contains 13233 total human images and 8351 total dog images





The bar charts show the number of humans compared to dogs in the distribution

Algorithms and Techniques

Convolutional Neural Networks (CNNs) are a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing images and videos. The layers of a CNN have neurons arranged in 3 dimensions known as called convolution layers. In the convolution layer, the filters are passed across the input, per row, and they activate when they see some type of visual feature.

The algorithm used for my project has three models:

1. Dog Detector: It is a resnet50 model using PyTorch. If the model generate a category between 151 and 268, it will be classified as Dog.

2. Human Detector: If the resnet50 doesn't predict as a Dog the input will go through the Haarfeature-based OpenCV model. This model is for facial detection, so if it detects a face, it will classify as a Human.(I didn't have to train the model)

3. If the algorithm detects a human or dog, the input goes through the Breed Classifier Model (a resnet50 with two more convolutional layers). It was retrained with all the layers frozen, except the two additional convolutional layers. It then predicts what breed is the most likely for the input image.

Benchmark

- The benchmark gave an accuracy score of 52.55% for a similar problem, but the retrained model which combined transfer learning and CNN architecture gave an accuracy score of 71%.

III. Methodology

Open CV's implementation of Haar feature-based-cascade-classifiers is used todetect human faces in the user-supplied images. There are many pre-trained de-tectors given by Open CV, for this project they are stored in the 'haarcascades' directory. The images are converted to greyscale before being passed to a face detector. The face detector is tested with 100 images each from the human and dog datasets. The face detector finds 98% of the human images have a human face and 17% of the dog images have a human face.

Next, a pre-trained VGG-16 model is used to detect dogs in images. It is used with weights trained on ImageNet, it has more than 10 million URLs where each URL links to an image containing an object from one of 1000 categories. Dog breeds occur consecutively on a dictionary from ImageNet from keys 151 to 268 inclusively. Therefore, the VGG-16 model is expected to return an index between 151 to 268 (inclusive).

Data Preprocessing

The images are resized to 224 by 224 in order to be able to reuse the same data loaders for the transfer learning step. The images are resized by doing a crop of random size of the original size and a random aspect ratio of the original aspect ratio

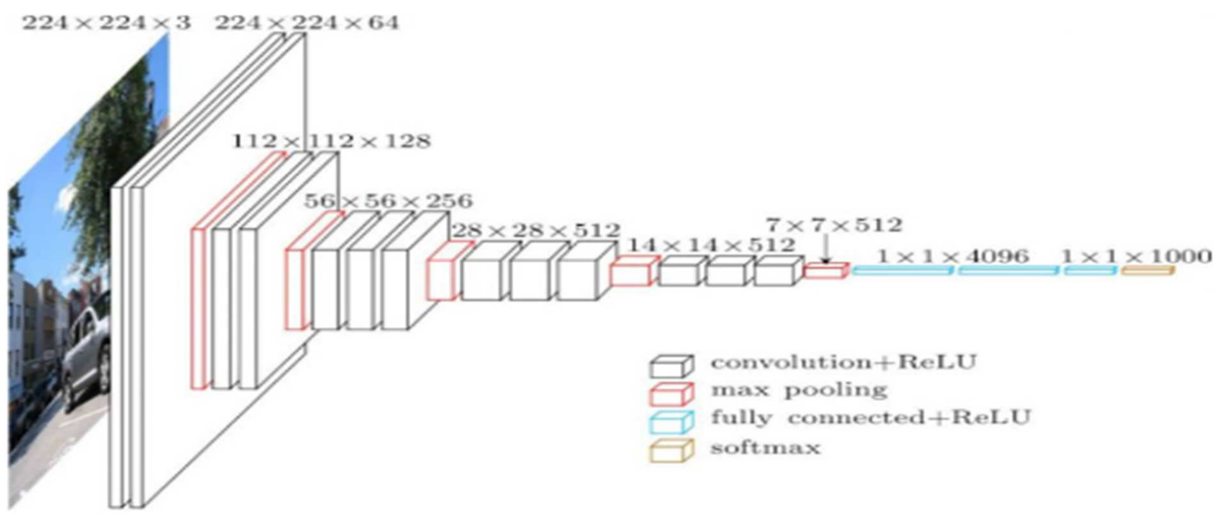
Data augmentation is done through flips, rotations and crops. The dataset is then split into train data (for training the model), validation data (for choosing the most accurate model), test data (for testing the model). CNN layers are utilized for feature extraction and image generalization. The dropout layer ensures that overfitting is dealt with and the linear classifier converts the extracted features to a classified type.

Implementation

Architecture

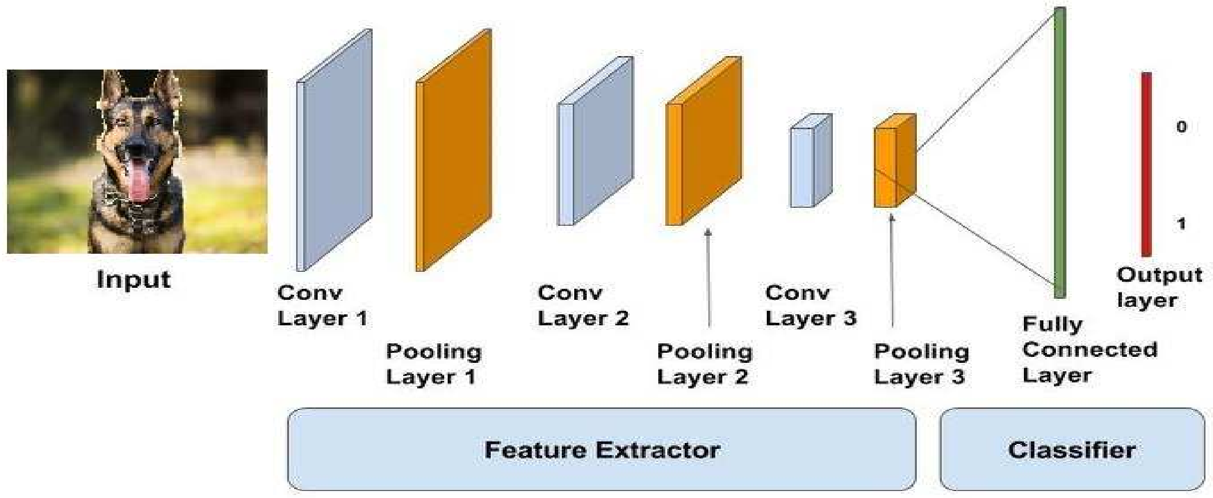
Finally, as we have been able to detect humans and dogs in images in the last sections using OpenCV and a VGG-16 model respectively. Now, a convolutional neural network (CNN) will be built from scratch with the aim of classifying dog breeds accordingly with the problem statement. However, the images in the datasets have to be preprocessed before passing them as into the CNN model.

The CNN model built from scratch can be improved significantly from its 11%test accuracy. This can be done through transfer learning. The CNN model will now use ResNet50 with the preprocessed data for the previous CNN model.



VGG-16 Model Architecture

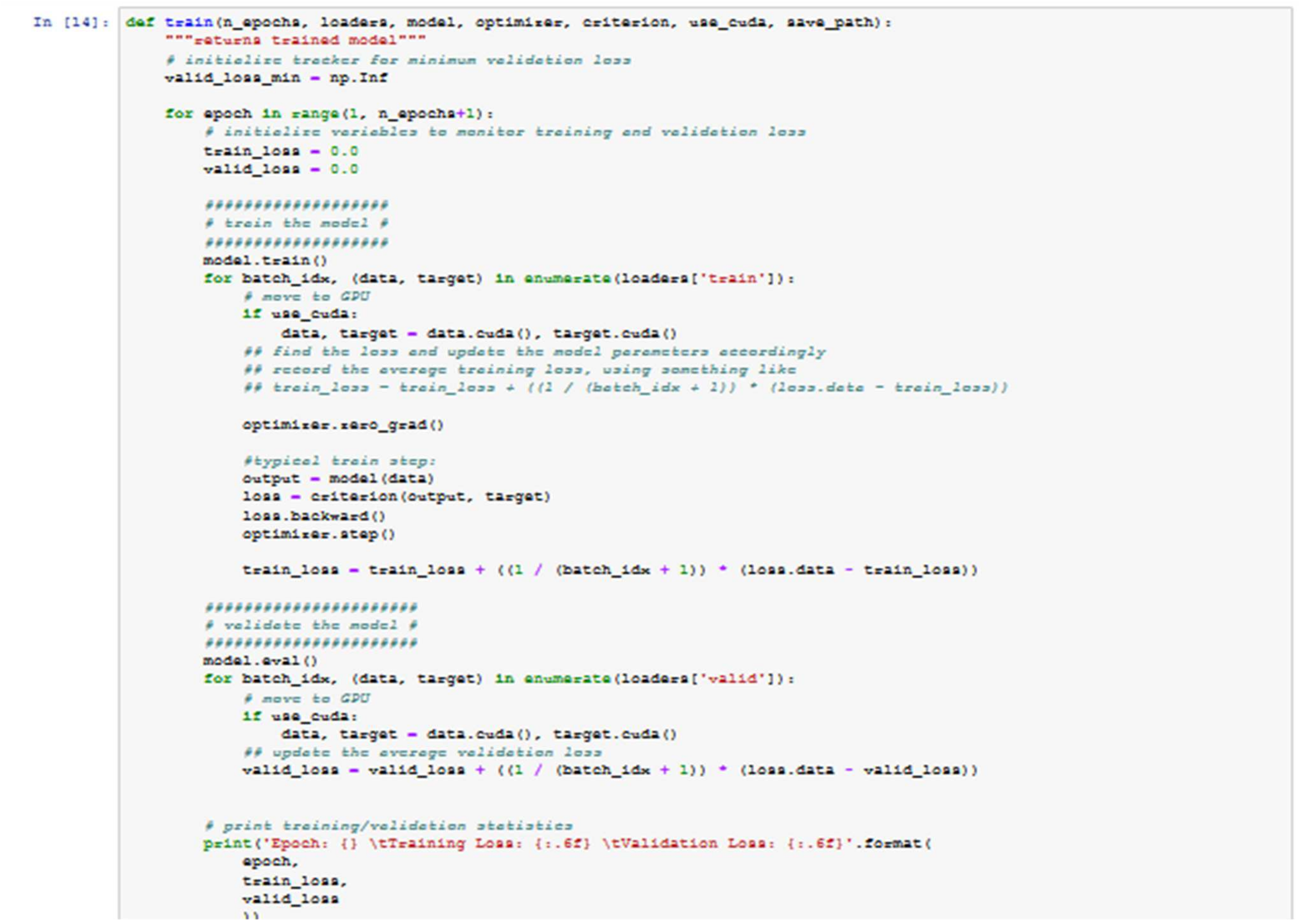
The architecture is composed from a feature extractor and a classifier. The feature extractor is has three CNN layers in to extract features. Each CNN layer has a ReLU activation and a 2D max pooling layer added to reduce the amount of parameters and computation in the network. After the CNN layers we have a dropout layer with a probability of 0.5 in to prevent overfitting and an average pooling layer to calculate the average for each patch of the feature map. The classifier is a fully connected layer with an input shape of 64 x 14 x 14 (which matches the output from the average pooling layer) and 133 nodes, one for each class (there are 133 dog breeds). We add a softmax activation to get the probabilities for each class.



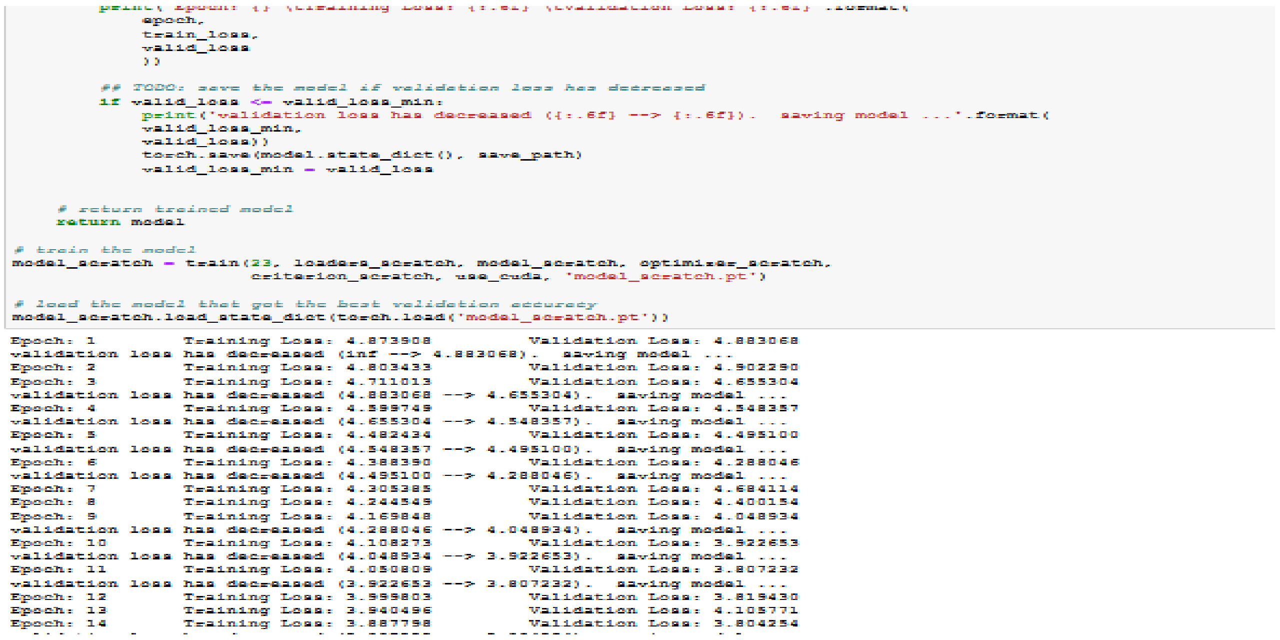
CNN Model Architecture

Training and Validation

For training and validating our model, we execute the following code snippet below:



```
In [14]: def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path):
             """returns trained model"""
             # initialize tracker for minimum validation loss
             valid_loss_min = np.Inf

             for epoch in range(1, n_epochs+1):
                 # initialize variables to monitor training and validation loss
                 train_loss = 0.0
                 valid_loss = 0.0

                 ###################
                 # train the model #
                 ###################
                 model.train()
                 for batch_idx, (data, target) in enumerate(loaders['train']):
                     # move to GPU
                     if use_cuda:
                         data, target = data.cuda(), target.cuda()
                     ## find the loss and update the model parameters accordingly
                     ## record the average training loss, using something like
                     ## train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))

                     optimizer.zero_grad()

                     #typical train step:
                     output = model(data)
                     loss = criterion(output, target)
                     loss.backward()
                     optimizer.step()

                     train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))

                 ######################
                 # validate the model #
                 ######################
                 model.eval()
                 for batch_idx, (data, target) in enumerate(loaders['valid']):
                     # move to GPU
                     if use_cuda:
                         data, target = data.cuda(), target.cuda()
                     ## update the average validation loss
                     valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss))


                 # print training/validation statistics
                 print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
                     epoch,
                     train_loss,
                     valid_loss
                     ))
```



```
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
            epoch,
            train_loss,
            valid_loss
            ))
```

```

        ## TODO: save the model if validation loss has decreased
        if valid_loss <= valid_loss_min:
            print('validation loss has decreased ({:.6f} --> {:.6f}).  saving model ...'.format(
            valid_loss_min,
            valid_loss))
            torch.save(model.state_dict(), save_path)
            valid_loss_min = valid_loss


    # return trained model
    return model

# train the model
model_scratch = train(23, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')

# load the model that got the best validation accuracy
model_scratch.load_state_dict(torch.load('model_scratch.pt'))
```

```
Epoch: 1        Training Loss: 4.873908         Validation Loss: 4.883068
validation loss has decreased (inf --> 4.883068).  saving model ...
Epoch: 2        Training Loss: 4.803433         Validation Loss: 4.902290
Epoch: 3        Training Loss: 4.711013         Validation Loss: 4.655304
validation loss has decreased (4.883068 --> 4.655304).  saving model ...
Epoch: 4        Training Loss: 4.599749         Validation Loss: 4.548357
validation loss has decreased (4.655304 --> 4.548357).  saving model ...
Epoch: 5        Training Loss: 4.482434         Validation Loss: 4.495100
validation loss has decreased (4.548357 --> 4.495100).  saving model ...
Epoch: 6        Training Loss: 4.388390         Validation Loss: 4.288046
validation loss has decreased (4.495100 --> 4.288046).  saving model ...
Epoch: 7        Training Loss: 4.305385         Validation Loss: 4.684114
Epoch: 8        Training Loss: 4.244549         Validation Loss: 4.400154
Epoch: 9        Training Loss: 4.169848         Validation Loss: 4.048934
validation loss has decreased (4.288046 --> 4.048934).  saving model ...
Epoch: 10       Training Loss: 4.108273         Validation Loss: 3.922653
validation loss has decreased (4.048934 --> 3.922653).  saving model ...
Epoch: 11       Training Loss: 4.050809         Validation Loss: 3.807232
validation loss has decreased (3.922653 --> 3.807232).  saving model ...
Epoch: 12       Training Loss: 3.999803         Validation Loss: 3.819430
Epoch: 13       Training Loss: 3.940496         Validation Loss: 4.105771
Epoch: 14       Training Loss: 3.887798         Validation Loss: 3.804254
```

Above code(run for 23 epochs) provides the following output which indicates a loss of 3.65 and an accuracy of 15%which is very less and hence a refinement is needed.

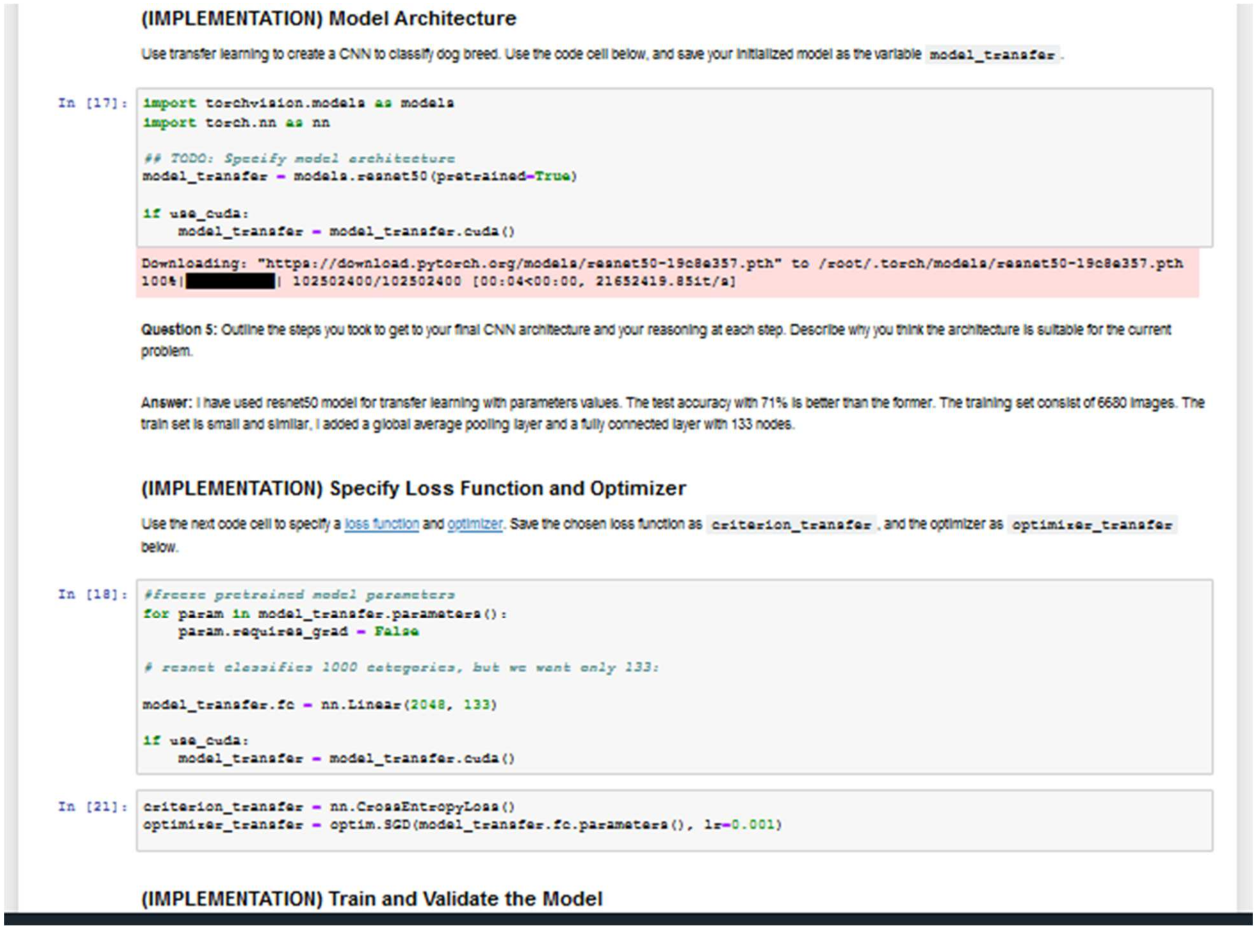
Test Loss: 3.76

Test Accuracy: 13% (116/836)

IV. Model Refinement and Results

Since our first CNN model scratch didn't do too great, we'll leverage on one of the pre-trained architectures packaged up nicely for us from the options between VGG-16, ResNet-50, InceptionV3. We use the ResNet-50 network that was pre-trained on the ImageNet dataset. We then train the model but we freeze the parameters for the feature extractor so only the classifier parameters get back propagated.

Now we again try out our model on the test dataset of dog images. Use the code snippets below to architect, train, calculate the test loss and accuracy :



(IMPLEMENTATION) Model Architecture

Use transfer learning to create a CNN to classify dog breed. Use the code cell below, and save your initialized model as the variable `model_transfer`.

```
In [17]: import torchvision.models as models
         import torch.nn as nn

         ## TODO: Specify model architecture
         model_transfer = models.resnet50(pretrained=True)

         if use_cuda:
             model_transfer = model_transfer.cuda()

Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.torch/models/resnet50-19c8e357.pth
100%|██████████| 102502400/102502400 [00:04<00:00, 21652419.85it/s]
```

Question 5: Outline the steps you took to get to your final CNN architecture and your reasoning at each step. Describe why you think the architecture is suitable for the current problem.

Answer: I have used resnet50 model for transfer learning with parameters values. The test accuracy with 71% is better than the former. The training set consist of 6680 images. The train set is small and similar, I added a global average pooling layer and a fully connected layer with 133 nodes.

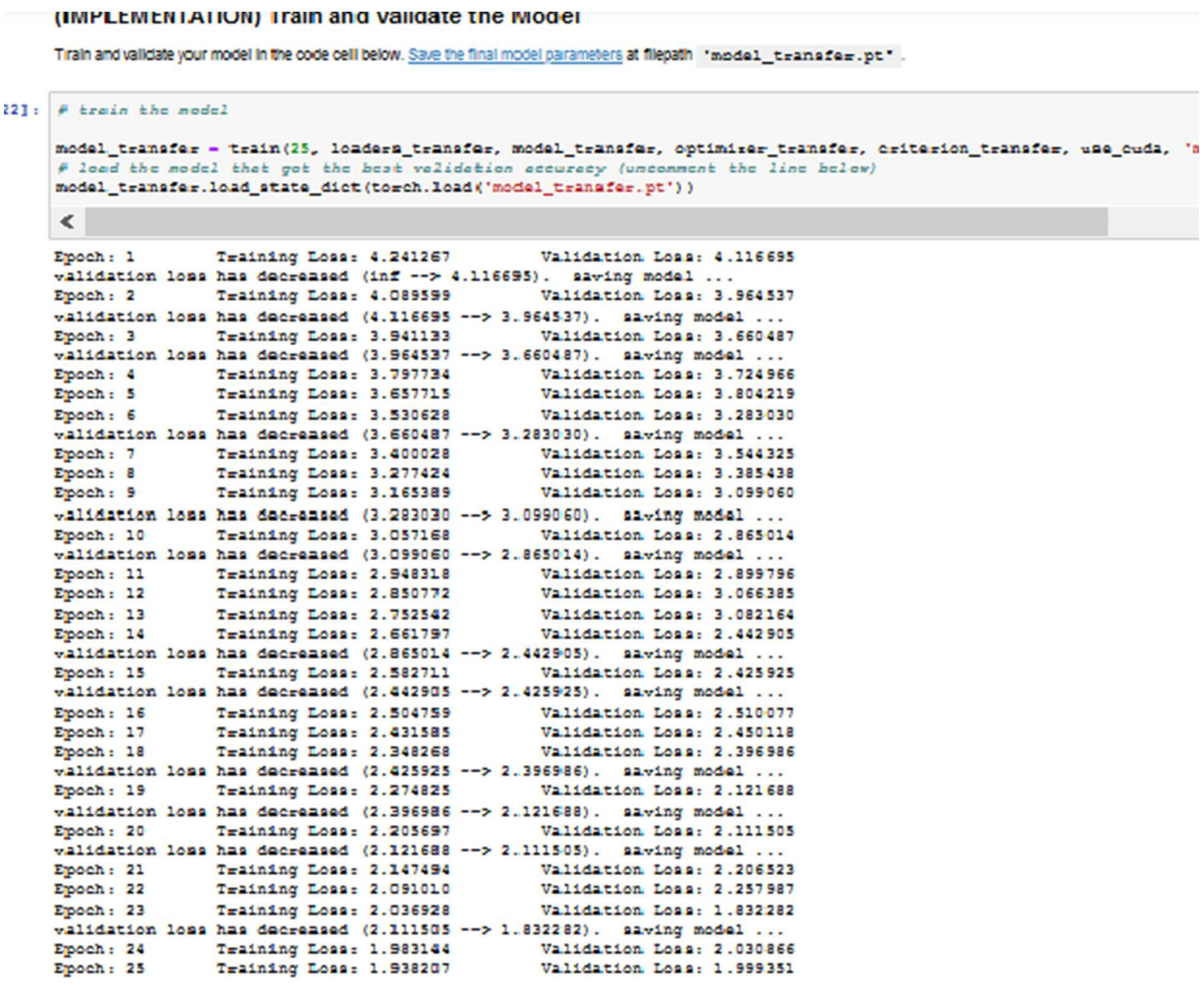
(IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_transfer`, and the optimizer as `optimizer_transfer` below.

```
In [18]: #freeze pretrained model parameters
         for param in model_transfer.parameters():
             param.requires_grad = False

         # resnet classifies 1000 categories, but we want only 133:

         model_transfer.fc = nn.Linear(2048, 133)

         if use_cuda:
             model_transfer = model_transfer.cuda()

In [21]: criterion_transfer = nn.CrossEntropyLoss()
         optimizer_transfer = optim.SGD(model_transfer.fc.parameters(), lr=0.001)
```

(IMPLEMENTATION) Train and Validate the Model

Architecture



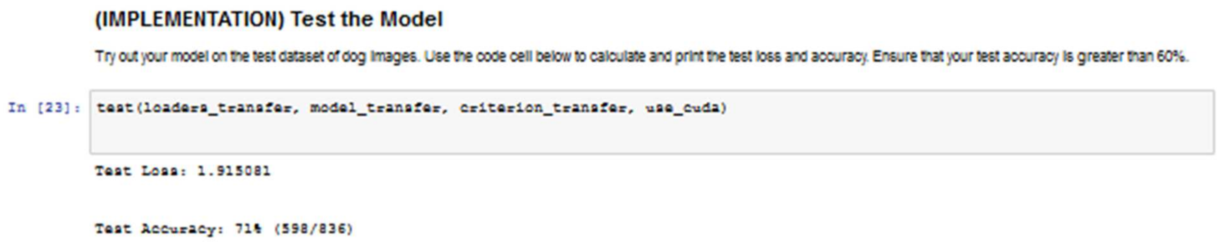
(IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. Save the final model parameters at filepath `'model_transfer.pt'`.

```
[22]: # train the model

model_transfer = train(25, loaders_transfer, model_transfer, optimizer_transfer, criterion_transfer, use_cuda, 'm
# 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: 4.241267         Validation Loss: 4.116695
validation loss has decreased (inf --> 4.116695).  saving model ...
Epoch: 2        Training Loss: 4.089599         Validation Loss: 3.964537
validation loss has decreased (4.116695 --> 3.964537).  saving model ...
Epoch: 3        Training Loss: 3.941133         Validation Loss: 3.660487
validation loss has decreased (3.964537 --> 3.660487).  saving model ...
Epoch: 4        Training Loss: 3.797734         Validation Loss: 3.724966
Epoch: 5        Training Loss: 3.657715         Validation Loss: 3.804219
Epoch: 6        Training Loss: 3.530628         Validation Loss: 3.283030
validation loss has decreased (3.660487 --> 3.283030).  saving model ...
Epoch: 7        Training Loss: 3.400028         Validation Loss: 3.544325
Epoch: 8        Training Loss: 3.277424         Validation Loss: 3.385438
Epoch: 9        Training Loss: 3.165389         Validation Loss: 3.099060
validation loss has decreased (3.283030 --> 3.099060).  saving model ...
Epoch: 10       Training Loss: 3.057168         Validation Loss: 2.865014
validation loss has decreased (3.099060 --> 2.865014).  saving model ...
Epoch: 11       Training Loss: 2.948318         Validation Loss: 2.899796
Epoch: 12       Training Loss: 2.850772         Validation Loss: 3.066385
Epoch: 13       Training Loss: 2.752542         Validation Loss: 3.082164
Epoch: 14       Training Loss: 2.661797         Validation Loss: 2.442905
validation loss has decreased (2.865014 --> 2.442905).  saving model ...
Epoch: 15       Training Loss: 2.582711         Validation Loss: 2.425925
validation loss has decreased (2.442905 --> 2.425925).  saving model ...
Epoch: 16       Training Loss: 2.504759         Validation Loss: 2.510077
Epoch: 17       Training Loss: 2.431585         Validation Loss: 2.450118
Epoch: 18       Training Loss: 2.348268         Validation Loss: 2.396986
validation loss has decreased (2.425925 --> 2.396986).  saving model ...
Epoch: 19       Training Loss: 2.274825         Validation Loss: 2.121688
validation loss has decreased (2.396986 --> 2.121688).  saving model ...
Epoch: 20       Training Loss: 2.205697         Validation Loss: 2.111505
validation loss has decreased (2.121688 --> 2.111505).  saving model ...
Epoch: 21       Training Loss: 2.147494         Validation Loss: 2.206523
Epoch: 22       Training Loss: 2.091010         Validation Loss: 2.257987
Epoch: 23       Training Loss: 2.036928         Validation Loss: 1.832282
validation loss has decreased (2.111505 --> 1.832282).  saving model ...
Epoch: 24       Training Loss: 1.983144         Validation Loss: 2.030866
Epoch: 25       Training Loss: 1.938207         Validation Loss: 1.999351
```

Training for 25 epochs



(IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 60%.

```
In [23]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 1.915081


Test Accuracy: 71% (598/836)
```

Test results

Above code provides the following output which indicates a loss of 1.91 and an accuracy of 71% which is much better than our previous approach.

Result

The final CNN model created with ResNet50 had a test loss of 1.91 and a test accuracy of 71% (598/836). This surpasses the benchmark threshold of 60% by 11%. I looked at various samples from both datasets and their results from the model.

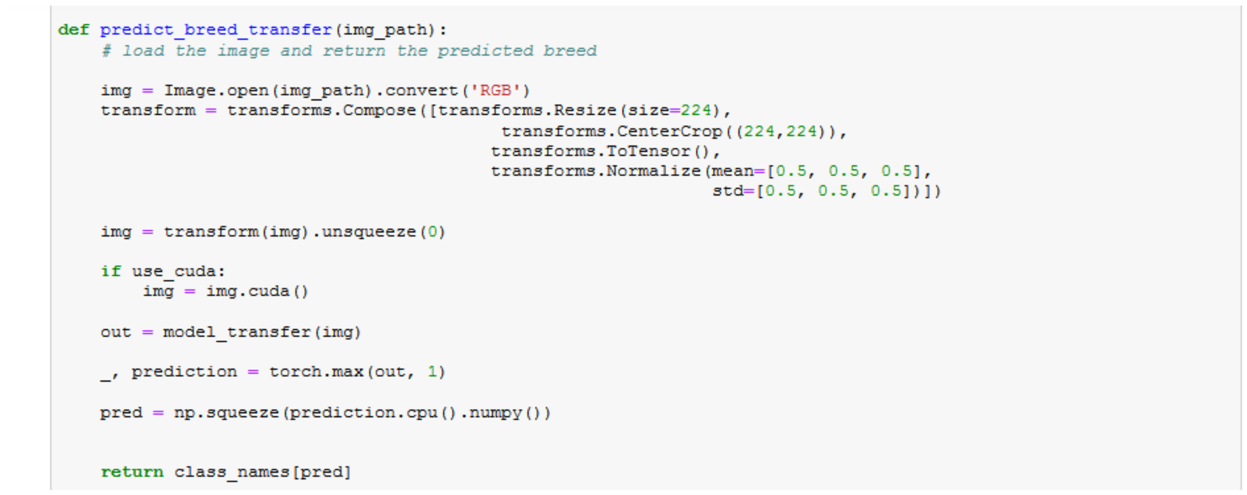
Model Evaluation and Validation

As we have successfully created a model architecture using transfer learning, I'll test out my model on a combination of some dog images as well as other random images to see how well it will perform. For this purpose, I have created an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither.

It works thus:

- If a dog is detected in the image, return the predicted breed.
- If a human is detected in the image, return the resembling dog breed.
- If neither is detected in the image, provide output that indicates an error.

The code snippet for the above mentioned process is shown below:

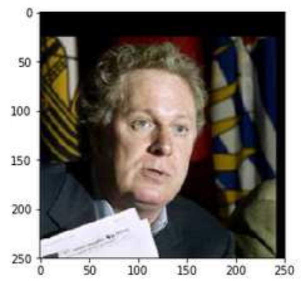


```python
def predict_breed_transfer(img_path):
    # load the image and return the predicted breed

    img = Image.open(img_path).convert('RGB')
    transform = transforms.Compose([transforms.Resize(size=224),
                                     transforms.CenterCrop((224,224)),
                                    transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.5, 0.5, 0.5],
                                                         std=[0.5, 0.5, 0.5])])

    img = transform(img).unsqueeze(0)

    if use_cuda:
        img = img.cuda()

    out = model_transfer(img)

    _, prediction = torch.max(out, 1)

    pred = np.squeeze(prediction.cpu().numpy())


    return class_names[pred]
```

```
def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    if face_detector(img_path):
        print("Human Being detected!")
        predicted_breed = predict_breed_transfer(img_path)
        image = Image.open(img_path)
        plt.imshow(image)
        plt.show()
        print("This human looks like the ", predicted_breed)
        print()

    elif dog_detector(img_path):
        print("Dog detected!")
        predicted_breed = predict_breed_transfer(img_path)
        image = Image.open(img_path)
        plt.imshow(image)
        plt.show()
        print("This dog is ", predicted_breed)
        print()

    else:
        print("No dog or human detected. Try another image!")
        image = Image.open(img_path)
        plt.imshow(image)
        plt.show()
        print('\n')
```

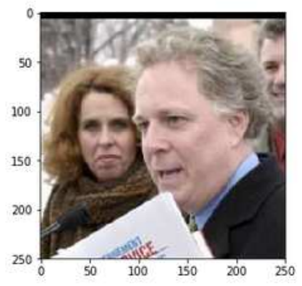
In [44]:
```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.

## suggested code, below
for file in np.hstack((human_files[133:136], dog_files[200:203])):
    run_app(file)
```

```
Human Being detected!
```



```
This human looks like the  Chinese crested
```

```
Human Being detected!
```



```
This human looks like the  Irish water spaniel
```

```
Human Being detected!
```

```
This human looks like the  French bulldog
```

```
Dog detected!
```

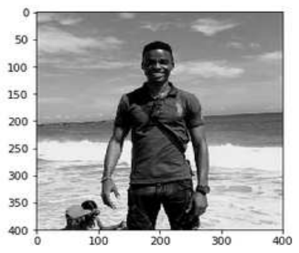
```
This dog is  Borzoi
```

```
Dog detected!
```

```
This dog is  Borzoi
```

```
In [45]: run_app('images/nige.jpg')
```

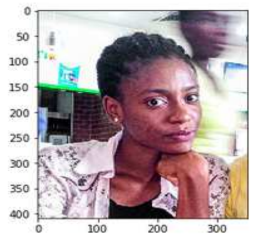
```
Human Being detected!
```



```
This human looks like the  Cane corso
```

```
In [46]: run_app('images/lady1.jpg')
```

```
Human Being detected!
```



```
This human looks like the  Chinese crested
```

V. Conclusion

In this project we tried to tackle the dog breed identification problem using the given data from Udacity. First, a small convolutional neural network was built and trained on the dataset from scratch, and the accuracy of 13% was achieved on the testing dataset. Then this result was improved by applying ResNet50 using transfer learning. The testing accuracy then improved to 71%. This result is quite good considering the complexity of the problem, and the limited amount of data given.

Reflection

Although the dog breed identification seems to be a very challenging problem, it was shown that a powerful and highly accurate CNN-based image classification model can be built with help of transfer learning. Transfer learning helped our model to perform much better, and it became more efficient computationally. Computational efficiency is very important when it comes to improving the performance of deep learning models. It is also worthy to note that the time and processing power it takes to run these deep learning models is high. If we were to all create deep learning models from scratch, we would compound this problem. Transfer learning goes a long way to address this.

Improvement

Here are following are ways I think the model's performance could be improved:

- More training data
- Altering the CNN's architecture
- Adjusting the transfer learning architecture
- Increase the breeds and train more images
- Increase number of epochs
- Augmentation of training data
- Tune some of the model parameters