Project: Dog Breed Classifier with Pytorch

Project Report

1. Project Overview and Problem Statement

1.1. Domain Background

Artificial Neural Networks have rapidly growing applications in the area of image-based classification problems. Moreover, recent advances in the ANN technology allows for the use/reuse of pre-trained Deep Artificial Neural Networks (e.g. AlexNet, Resnet, VGG, and others) that were trained to recognize thousands of different categories by sifting through millions of high resolution images. This re-utilization of pretrained Deep Neural Networks belongs to a field of Machine Learning called "Transfer Learning" where these pre-built models can be repurposed to be utilized in numerous applications of image recognition and classification.

1.2. Problem Statement

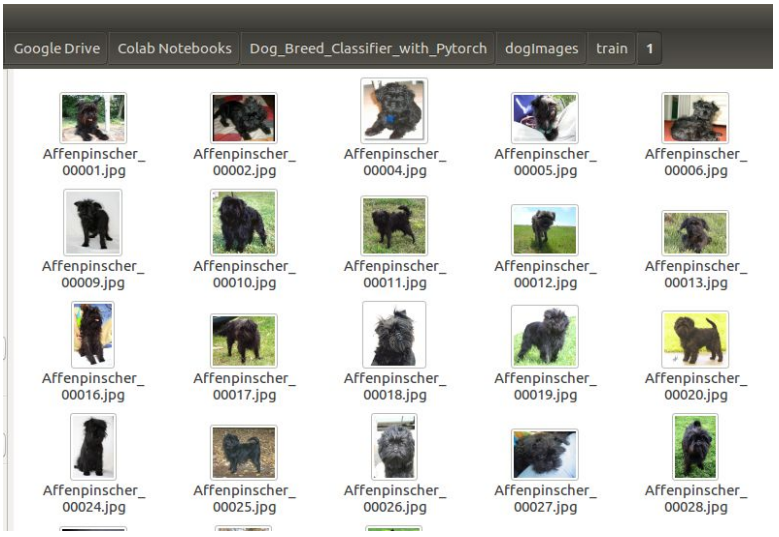
Recognizing the breeds of dogs is certainly a challenging task. The estimated number of known dog breeds ranges between 200 and 350 different breeds, making the task of telling "which breed a dog belongs to" a very challenging task. This task gets more difficult when the breeds are visually similar and cannot be easily distinguished.

1.3. Datasets and Inputs

The dataset that will be utilized for this project is a collection of 8,753 dog images across 133 different breeds. The dataset is offered by Udacity and can be downloaded through the following link:

https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

The data comes in the form of images contained in numbered folders where each numbered folder is for a specific dog-breed. It's important to note that the photos are not of the same size; hence, randomized cropping needs to be implemented as part of the data augmentation.



1.4. Metrics

In terms of the optimization of the models, the CrossEntropyLoss will be used to determine if a model is more suitable than another model.

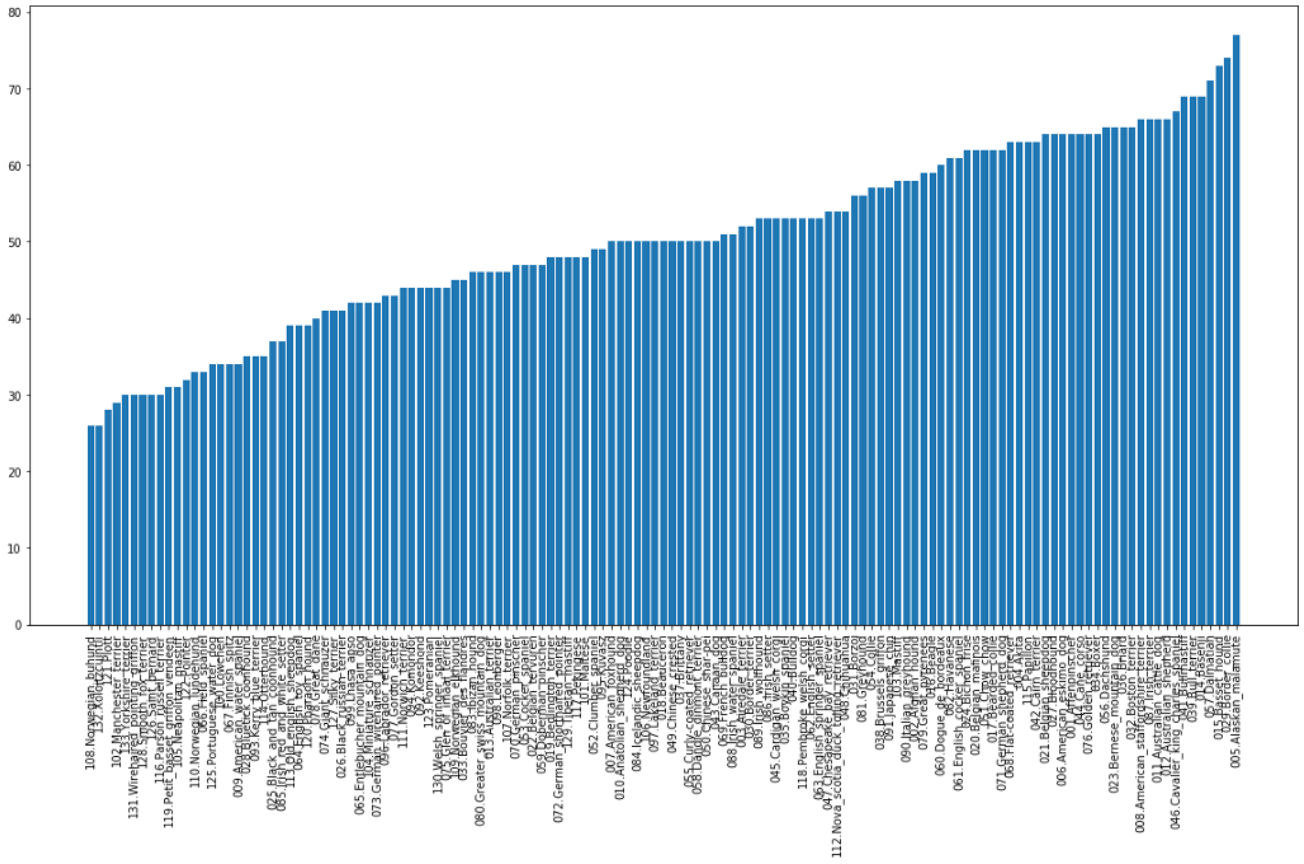
Eventually, to compare the final trained model to the benchmark model, the simple metric of accuracy along with the CrossEntropyLoss are going to be used to compare the models.

Although accuracy is not necessarily the best metric to evaluate the performance of a model, it relatively sufficient here as the distribution of training/validation/testing data across the different classes (133 breeds) is not extreme. Moreover, the Cross Entropy should also serve as a secondary comparison metric between the trained model and the benchmark model.

2. Data Analysis

2.1. Data Exploration

In terms of the balance of training data across the 133 classes (dog breeds), we can see that we have a considerable variance across the classes. However, the bulk of the classes remain in the range of 40-60 images which is close to an average of around 50.



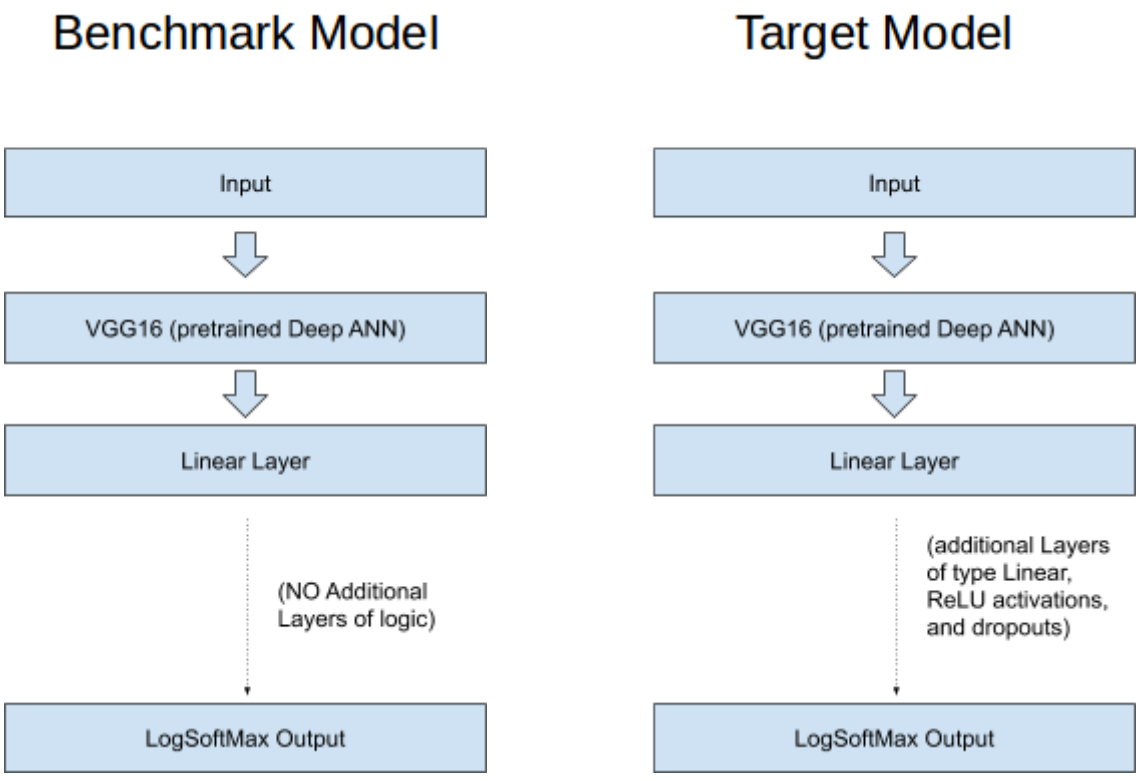
2.2. Algorithms and Techniques

In order to build a robust classifier, we need to rely on the following specifications when we build the model:

- A Stochastic Gradient Descent Optimizer
- A Cross Entropy Loss function that will be used to optimize the model
- A pre-trained VGG16 Deep Neural Network for the initial feature recognition from the input images. This VGG16 network will not be trained with the rest of the model.
- Additional layers on top of the pretrained VGG16 network to be trained of the recognition of the 133 dog breeds at hand.
- The use of ReLU activation functions in order to combat the problem of vanishing gradients.
- Dropout layers in order to reduce the risk of overfitting.

2.3. Benchmark Model

The target trained model will be benchmarked against a similar, yet simplified, model. As our target model will involve additional layers on top of the VGG16 pretrained Network that will include multiple layers of Linear, ReLU, and, Dropout functions, the benchmark model will include a single Linear layer followed by a logsoftmax output as shown below. It's important to keep in mind that the benchmark model will be trained too till it cannot improved any further.



3. Implementation

3.1. Data Loading and Preprocessing

As a prerequisite to training the model, the data need to be loaded, normalized, and augmented. Accordingly, the data will be augmented first by introducing random rotations, random resizing and cropping to the desired input size, random flipped

across the horizontal axis. After that all pixel RGB values will be Normalized to a pre-specified mean and standard deviation.

It's important to note that the augmentations introduced here are only necessary for the training dataset. The validation and testing datasets don't need anything more than resizing, cropping, and normalization.

Due to the big size of our dataset, we will not be able to fit it in the RAM. Accordingly, batch loader need to be defined in order to load the data in steps. The batch size to be used here is 64 images at a time.

```
# TODO: Define your transforms for the training, validation, and testing sets
train_transforms = transforms.Compose([
                                       transforms.RandomRotation(30),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
                                      ])
test_transforms = transforms.Compose([
                                       transforms.Resize(266),
                                       transforms.CenterCrop(224),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
                                      ])

# TODO: Load the datasets with ImageFolder
train_data = datasets.ImageFolder(train_dir,transform=train_transforms)
valid_data = datasets.ImageFolder(valid_dir,transform=test_transforms)
test_data = datasets.ImageFolder(test_dir,transform=test_transforms)

# TODO: Using the image datasets and the trainforms, define the dataloaders
train_loader = torch.utils.data.DataLoader(train_data,batch_size=64,shuffle=True)
valid_loader = torch.utils.data.DataLoader(valid_data,batch_size=64)
test_loader = torch.utils.data.DataLoader(test_data,batch_size=64)
```

3.2. Building the Benchmark and Target Models

3.2.1. The Benchmark Model:

The design of the benchmark model is a simple Linear layer on top of the VGG16 pretrained network, followed by a LogSoftmax output to the 133 dog breeds.

As shown on the screenshot below, the gradient descent of the VGG16 pretrained network is switched off in order to freeze it such that it does not get trained. This will restrict training only to the additional Linear and LogSoftmax layers setting on top of it.

```
def set_benchmark_model():
    # create a dense net model
    model = models.vgg16(pretrained = True)

    # switch off the gradients of the convolutional feature recognition layer
    for param in model.parameters():
        param.requires_grad = False # this will stop training for the vgg16 pre-trained network

    model.classifier = nn.Sequential(
        nn.Linear(25088,133),
        nn.LogSoftmax(dim=1)
    )

    return model
```

3.2.2. The Target Model:

The design of the target model is slightly more complex than the Benchmark model. Just like the benchmark model, it uses a pretrained VGG16 pretrained network with gradient descent switched off. However, the additional layers sitting on top of the pretrained network are more complex. There are three layers of linear logic with decreasing sizes all the way from 25088 perceptrons to 133 perceptrons (corresponding to dog breeds). In between these linear layers, there are ReLU activation functions (these are more suitable to reduce the effect of the vanishing gradient problem). There are also Dropout layers that will force the training to be focused on subsets of the perceptron at each layer resulting in less risk of overfitting. Finally, the output is LogSoftmax output just like the benchmark model.

```
def set_target_model():
    # create a dense net model
    model = models.vgg16(pretrained = True)

    # switch off the gradients of the convolutional feature recognition layer
    for param in model.parameters():
        param.requires_grad = False # this will stop training for the vgg16 pre-trained network


    # replace the classifier layer
    # parameters here require gradient by default
    model.classifier = nn.Sequential(
        nn.Linear(25088,750),
        nn.ReLU(inplace=True),
        nn.Dropout(0.2),
        nn.Linear(750,300),
        nn.ReLU(inplace=True),
        nn.Dropout(0.2),
        nn.Linear(300,133),
        nn.LogSoftmax(dim=1)
    )
    return model
```

3.3. Training the Models

The number of training epochs is set to 40 for the benchmark model and 80 for the target model. The numbers were chosen as such after multiple trials of training where the models could not be usefully trained any further .

It's also important to note that the training routines are built such that the model with the lowest loss on the validation dataset gets saved as the most optimum model.

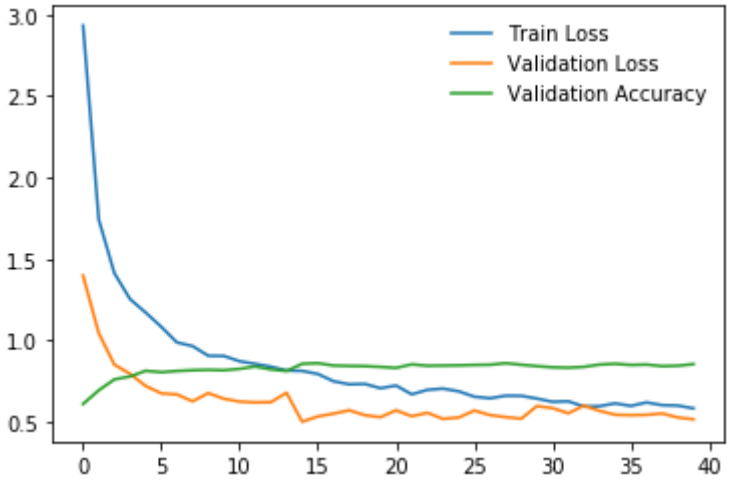
```
benchmark_model, bench_train_losses, bench_validation_losses, bench_validation_accuracies = train_model(
    benchmark_model,
    mode_of_operation='heavy',
    epochs=40,
    save_path='gdrive/My Drive/Colab Notebooks/Dog_Breed_Classifier_with_Pytorch/benchmark_model.pt')
```

```
target_model, target_train_losses, target_validation_losses, target_validation_accuracies = train_model(
    target_model,mode_of_operation='heavy',
    epochs=80,
    save_path='gdrive/My Drive/Colab Notebooks/Dog_Breed_Classifier_with_Pytorch/target_model.pt')
```

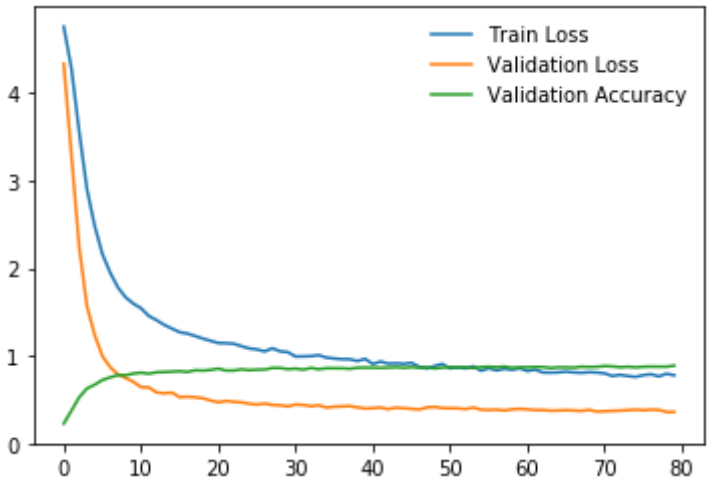
4. Result Analysis

4.1. Compare the Benchmark and Target Models

Benchmark Model Scree plot



Target Model Scree plot



As per the scree plot above, the validation loss for the target model is way more stable in it's decent. It's notable that both models can potentially be improved if trained for more epochs. Moreover, we could also try adding more layers to our target model, or expand the layer sizes in order to try and improve its performance.

4.2. Testing the Optimized Models

```
run_testing(
                model=benchmark_model,
                criterion=test_criterion,
                device=test_device,
                validloader=test_loader,
                validation_losses = None,
                validation_accuracies = None,
                test_type = "Test"
            )
```

```
Test Loss: 0.626
Test Accuracy: 0.831
```

```
run_testing(
                model=target_model,
                criterion=test_criterion,
                device=test_device,
                validloader=test_loader,
                validation_losses = None,
                validation_accuracies = None,
                test_type = "Test"
            )
```

```
Test Loss: 0.436
Test Accuracy: 0.866
```

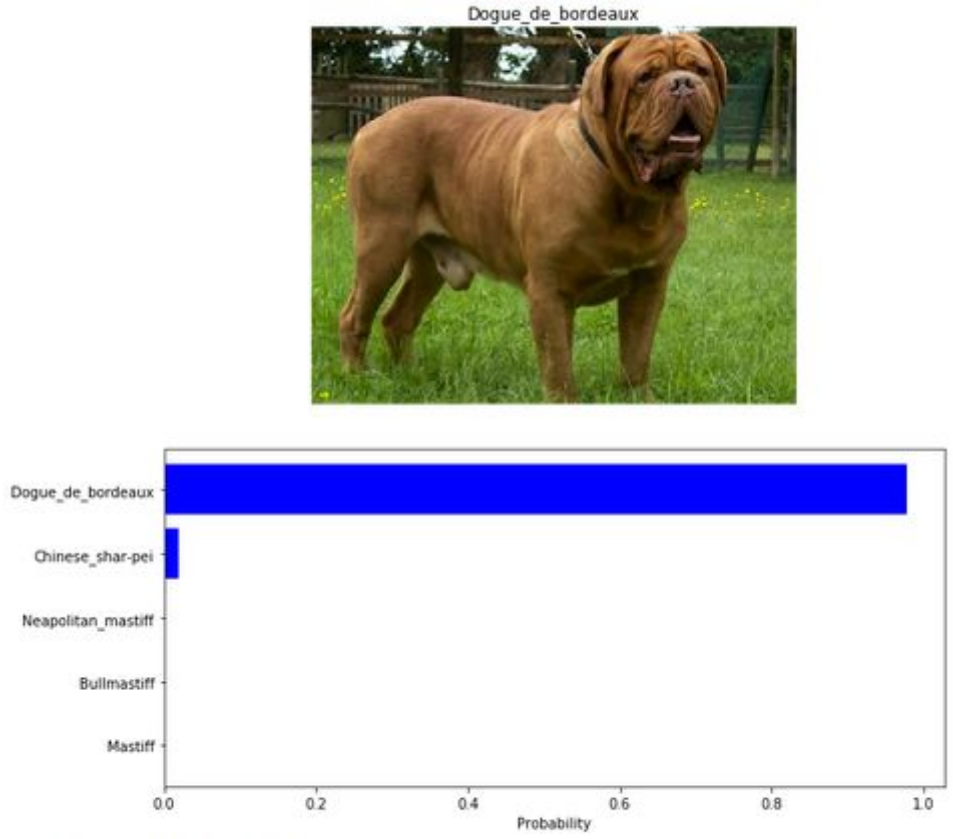
We can see a considerable improvement in the target model over the benchmark model. Despite the fact the accuracy of the target model did not significantly improve over the benchmark model, the Cross Entropy loss, however, did drop significantly. We could also resort to more metrics (such as precision and recall) in order to properly further assess the improvement over the benchmark model.

Model	Cross Entropy Loss	Accuracy	CrossEntropyLoss difference (compared to benchmark)	Accuracy difference (compared to benchmark)
Benchmark Model	0.626	83.1%	0%	0%
Target Model	0.436	86.6%	-30.3%	+4.2%

4.3. Test Demo

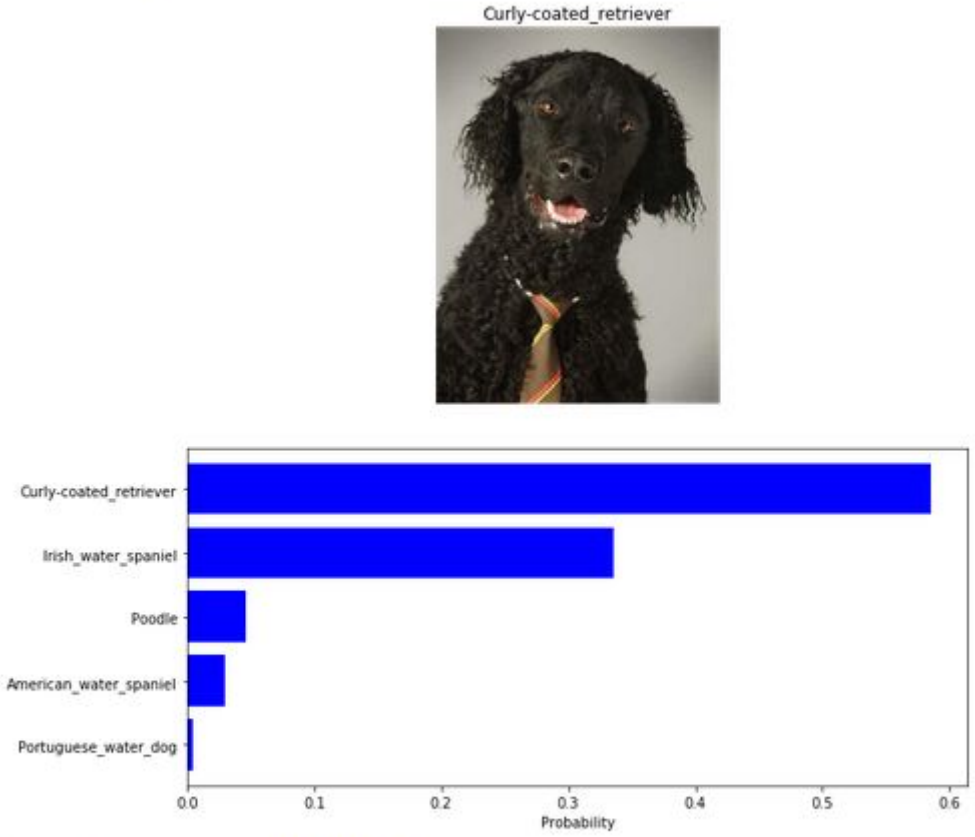
Below are 5 randomized tests on the target model to demo its capability to

```
run_a_random_test()
```



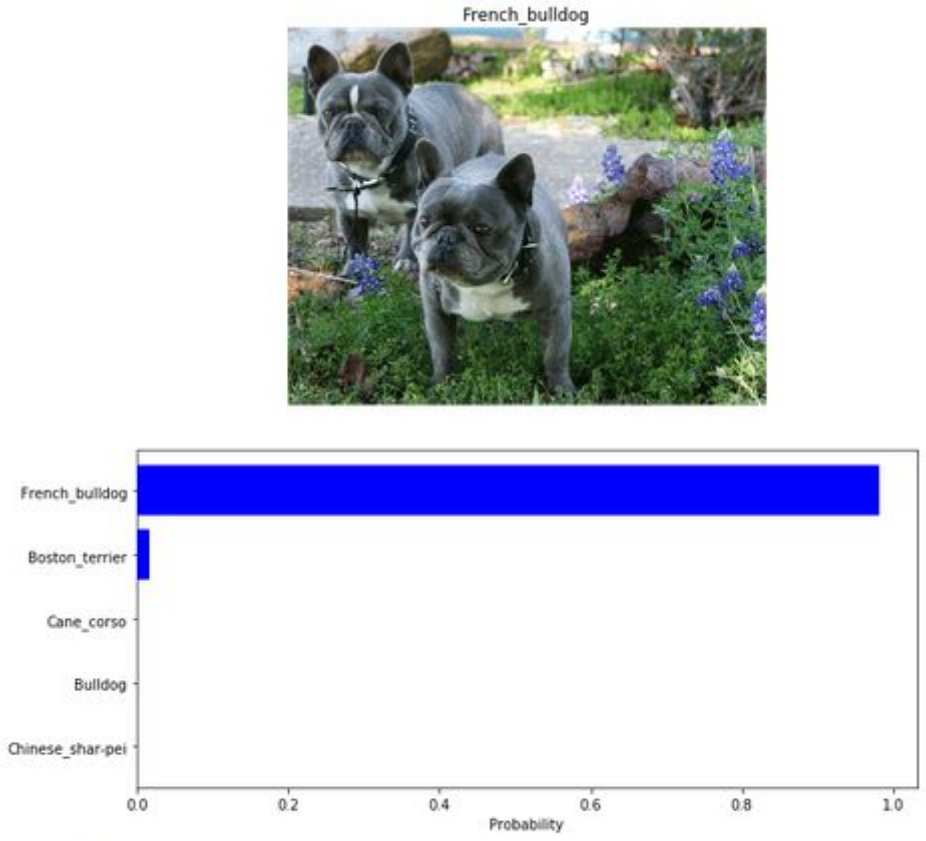

```
Actual Class: Dogue_de_bordeaux
Prediction Class: Dogue_de_bordeaux
Is Prediction Correct?: True
```

```
run_a_random_test()
```



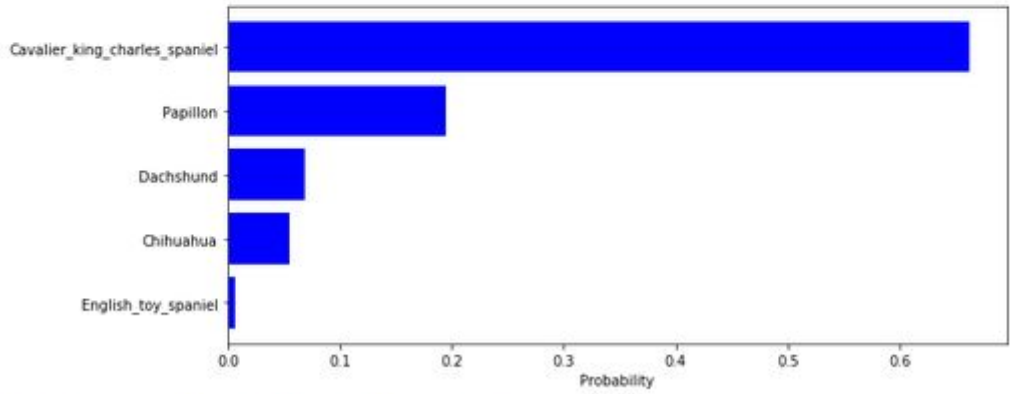

```
Actual Class: Curly-coated_retriever
Prediction Class: Curly-coated_retriever
Is Prediction Correct?: True
```

```
run_a_random_test()
```



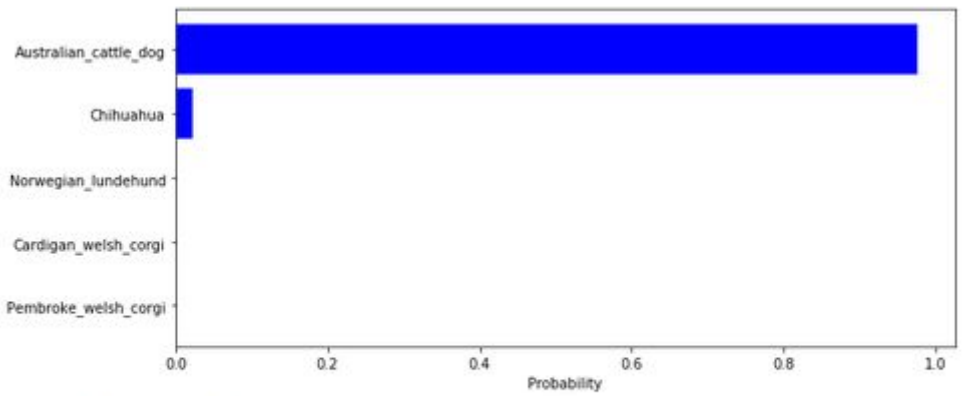

```
Actual Class: French_bulldog
Prediction Class: French_bulldog
Is Prediction Correct?: True
```

```
run_a_random_test()
```




```
Actual Class: Cavalier_king_charles_spaniel
Prediction Class: Cavalier_king_charles_spaniel
Is Prediction Correct?: True
```

```
run_a_random_test()
```




```
Actual Class: Australian_cattle_dog
Prediction Class: Australian_cattle_dog
Is Prediction Correct?: True
```

5. Conclusions

- We have successfully built and trained a Deep Neural Net model based on the VGG16 pretrained network to predict the breed of a dog by image input for 133 dog breeds.
- The model can predict the breed of 133 dogs with 86.6% accuracy.
- The trained model has achieved an increase in prediction accuracy of 4.2% over the benchmark model on the testing dataset.
- The trained model has achieved a reduction of Cross Entropy Loss of 30.3% below the benchmark model on the testing dataset.