Capstone Project

Pornrapee Kajorndech

Machine Learning Engineer Nanodegree **Aug 27th, 2020**

Definition

Project Overview

The dog breed classifier is a problem that can be solved with the ML model. The problem is identifying a breed of a dog with a dog image that is given as an input. This is a multi-class classification problem where we can use a supervised ML model to solve. Beside the classifier model, I build a web application and Rest-API for users to upload an image and get the predicted result. This project is a good challenge for me at the point of building an end-to-end ML project, so I have chosen this project as my capstone project.

Problem Statement

The end goal of the project is to build a ML model to classify dogs among 133 breeds and also implement a web application that connects to Restful API service for the final model to mimic real world use cases. Users can upload a dog image and the model will return the dog breed.

Metrics

The goal here is to compare the performance of my model with that of the benchmark model. Therefore, I would use accuracy as an evaluation metric. Accuracy is the ratio of correct predictions to the total size of data. In addition, because the benchmark model, that will be explained later in this report, only specifies the accuracy.

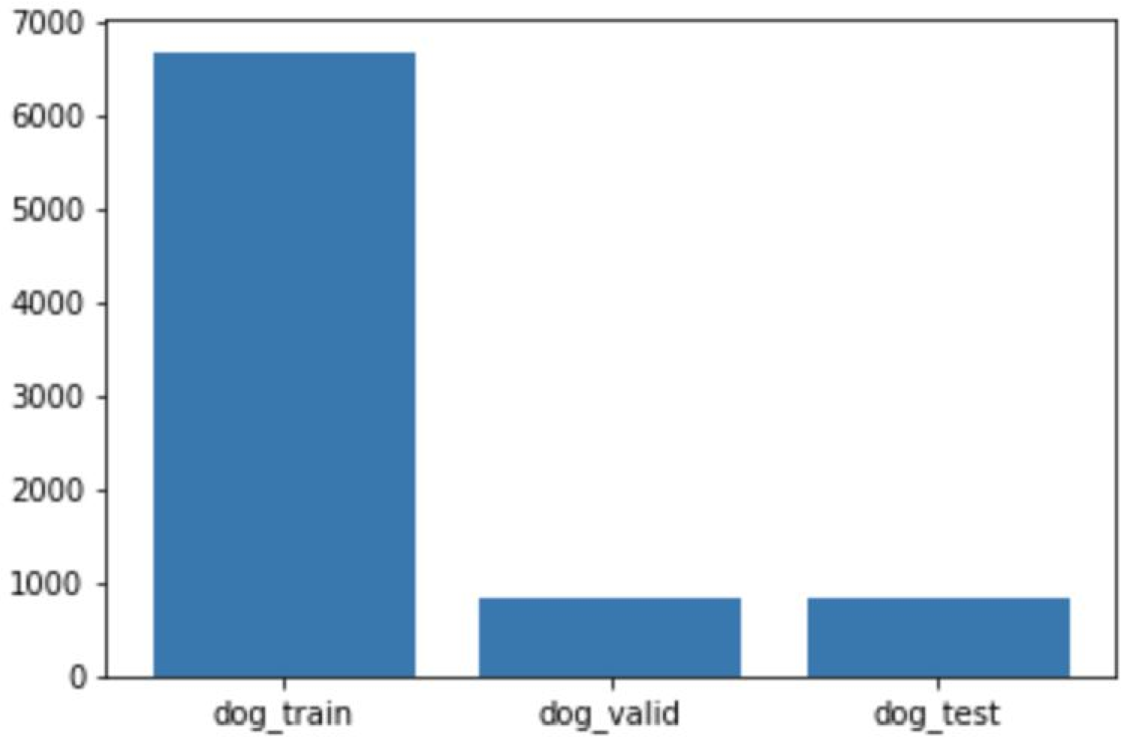
Analysis

Data Exploration

In the Dog Breed Classification, the dataset contains the images of Dogs. There are a total of 133 breeds, 8351 images of dogs. Using these images as data, I do the data preprocessing to improve robustness of the model. On making the analysis on the data, I see that the resolution of the images are not the same for all images of dogs in their respective breeds. The images have a varied resolution and they need to be resampled based on the requirement of our model. Here is few examples of the images discussed in terms of resolution

Example of dog image in the dataset

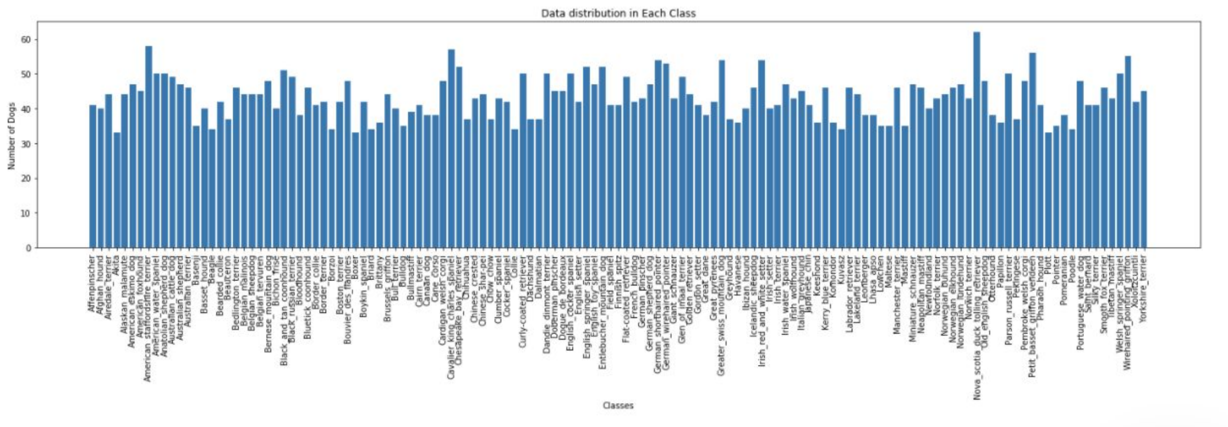
In our case, I observe that the split of train and test data is 90%-10%, i.e. 90% for training and 10% for testing purposes. In the training data, I have reserved another 10% for validation. The resultant split of data can be observed in the below graph. From the below plot, I can observe that a total of 6680 images will be used to train with our machine, to further fine-tune the parameters I use another 835 images for validating it. And, lastly, I will be using 836 images to test our model's performance for the evaluation of metric.



Plot of training, validation and testing image ratio

Exploratory Visualization

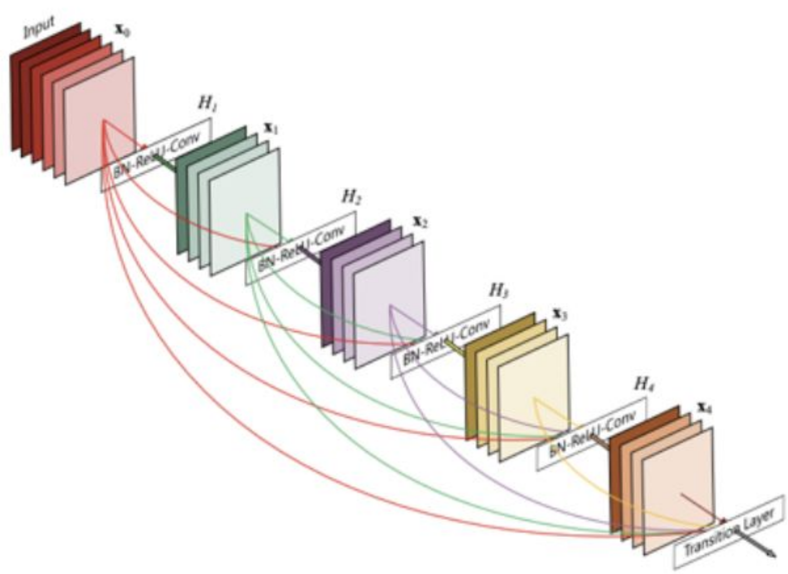
The study on the distribution of data gives us the information on balance or imbalance in the data. If there is an imbalance in the data beyond a certain threshold, we must see that the data is balanced by adding relevant images. If the balance in the data is comparatively near to the threshold, it is good to carry forward with the operation. Let us see how it works with our data in the below figure. The plot shows a clear description of breed class with the number of dogs.



Dog breed Distribution

Algorithms and Techniques

Dog Breeds Classification: i have use Densenet161 pretrained model to build the network



Densenet161 Architecture

Restful API service: I have used Flask (Python framework) to build API.

Web Application: I have used ReactJS (Javascript framework) to build a web application.

Benchmark

The VGG-16 bottleneck features from https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification and add some layer to this pretrained model, described by image below.

```
In [43]: VGG16_model = Sequential()
         VGG16_model.add(GlobalAveragePooling2D(input_shape=train_VGG16.shape[1:]))
         VGG16_model.add(Dense(133, activation='softmax'))

         VGG16_model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 133)               68229
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
_________________________________________________________________
```

And evaluate performance of benchmark model.

```
Test accuracy: 73.0861%
```

This model got 73% accuracy.

Methodology

Data Preprocessing

The image I give as input to the model processes the image before passing it to the network. In this step, the image is normalized, resized, flipped and cropped to improve robustness of the model in the training phase. Then the image is converted into tensors.

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

train_dataset = datasets.ImageFolder(train_path,  transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(15),
            transforms.ToTensor(),
            normalize,
        ]))

val_dataset = datasets.ImageFolder(val_path,  transforms.Compose([
            transforms.Resize(size=(224,224)),
            transforms.ToTensor(),
            normalize,
        ]))

test_dataset = datasets.ImageFolder(test_path,  transforms.Compose([
             transforms.Resize(size=(224,224)),
            transforms.ToTensor(),
            normalize,
        ]))
```

Implementation

In this step, I used the Densenet161 pretrained model. And added some layers on a pretrained model.

```
      )
      (norm5): BatchNorm2d(2208, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (classifier): Linear(in_features=2208, out_features=133, bias=True)
  )
```

I specified Cross-Entropy-Loss to loss function and Adagrad to optimizer

```
import torch.optim as optim
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.Adagrad(model_transfer.classifier.parameters(), lr=0.01)
```

I trained model with 15 epochs

```
Epoch: 1        Training Loss: 2.545464        Validation Loss: 1.081826
Validation loss decreased (inf ⟶ 1.081826).  Saving model ...
Epoch: 2        Training Loss: 1.316941        Validation Loss: 0.777360
Validation loss decreased (1.081826 ⟶ 0.777360).  Saving model ...
Epoch: 3        Training Loss: 1.115513        Validation Loss: 0.653974
Validation loss decreased (0.777360 ⟶ 0.653974).  Saving model ...
Epoch: 4        Training Loss: 1.007111        Validation Loss: 0.595116
Validation loss decreased (0.653974 ⟶ 0.595116).  Saving model ...
Epoch: 5        Training Loss: 0.964999        Validation Loss: 0.545928
Validation loss decreased (0.595116 ⟶ 0.545928).  Saving model ...
Epoch: 6        Training Loss: 0.909896        Validation Loss: 0.505973
Validation loss decreased (0.545928 ⟶ 0.505973).  Saving model ...
Epoch: 7        Training Loss: 0.843353        Validation Loss: 0.495662
Validation loss decreased (0.505973 ⟶ 0.495662).  Saving model ...
Epoch: 8        Training Loss: 0.844922        Validation Loss: 0.489306
Validation loss decreased (0.495662 ⟶ 0.489306).  Saving model ...
Epoch: 9        Training Loss: 0.813664        Validation Loss: 0.480000
Validation loss decreased (0.489306 ⟶ 0.480000).  Saving model ...
Epoch: 10       Training Loss: 0.782049        Validation Loss: 0.471942
Validation loss decreased (0.480000 ⟶ 0.471942).  Saving model ...
Epoch: 11       Training Loss: 0.758415        Validation Loss: 0.457790
Validation loss decreased (0.471942 ⟶ 0.457790).  Saving model ...
Epoch: 12       Training Loss: 0.757047        Validation Loss: 0.454373
Validation loss decreased (0.457790 ⟶ 0.454373).  Saving model ...
Epoch: 13       Training Loss: 0.735301        Validation Loss: 0.436120
Validation loss decreased (0.454373 ⟶ 0.436120).  Saving model ...
Epoch: 14       Training Loss: 0.744315        Validation Loss: 0.431123
Validation loss decreased (0.436120 ⟶ 0.431123).  Saving model ...
```

Refinement

In the Dog Breed Classifier with Transfer Learning stage, I’ve explored state-of-the-art models and made important design decisions about the performance of the model. I’ve used the ResNet-50 and got the accuracy 83%. By the way, after trying another network, I got better performance from Densenet161 as reported in the next section.

Results

Model Evaluation and Validation

In this section, I would like to observe the performance of the model on a test dataset and show some outputs from a web application. Ensure that the test accuracy is greater than benchmark.

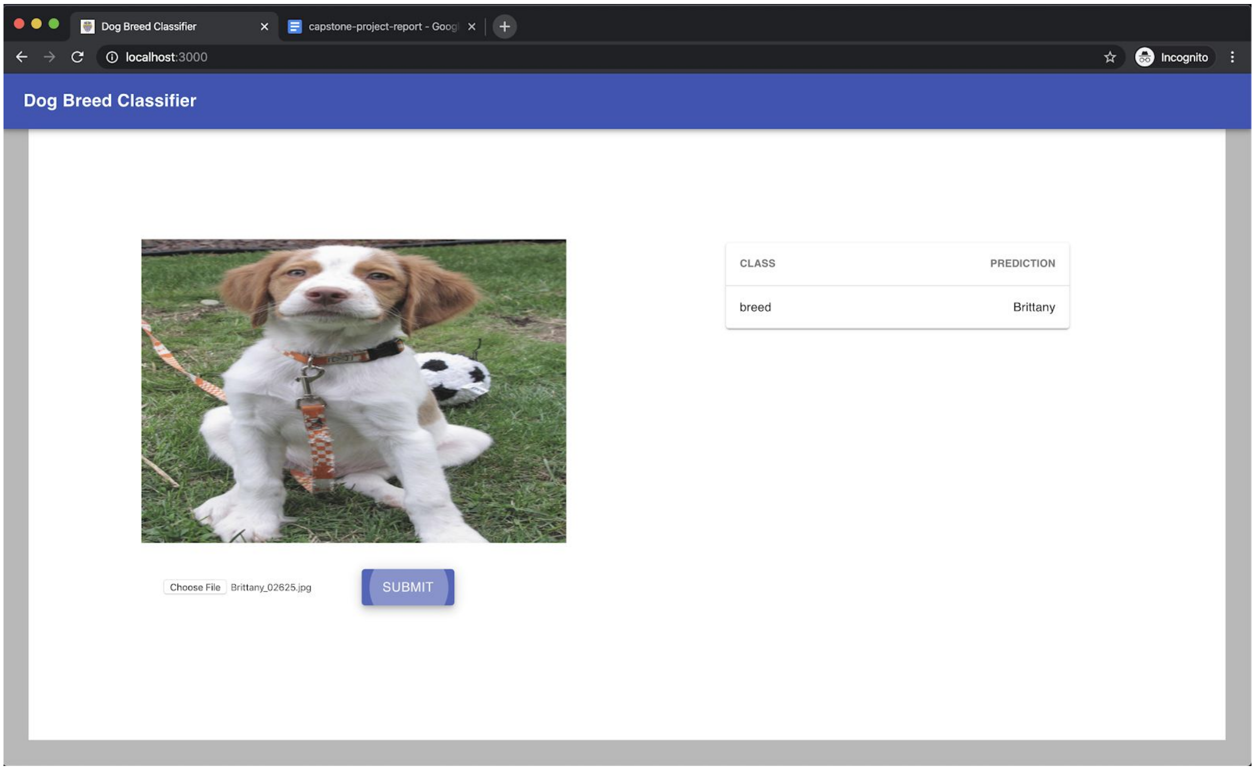
```
Test Loss: 0.471578


Test Accuracy: 87% (735/836)
```

Justification

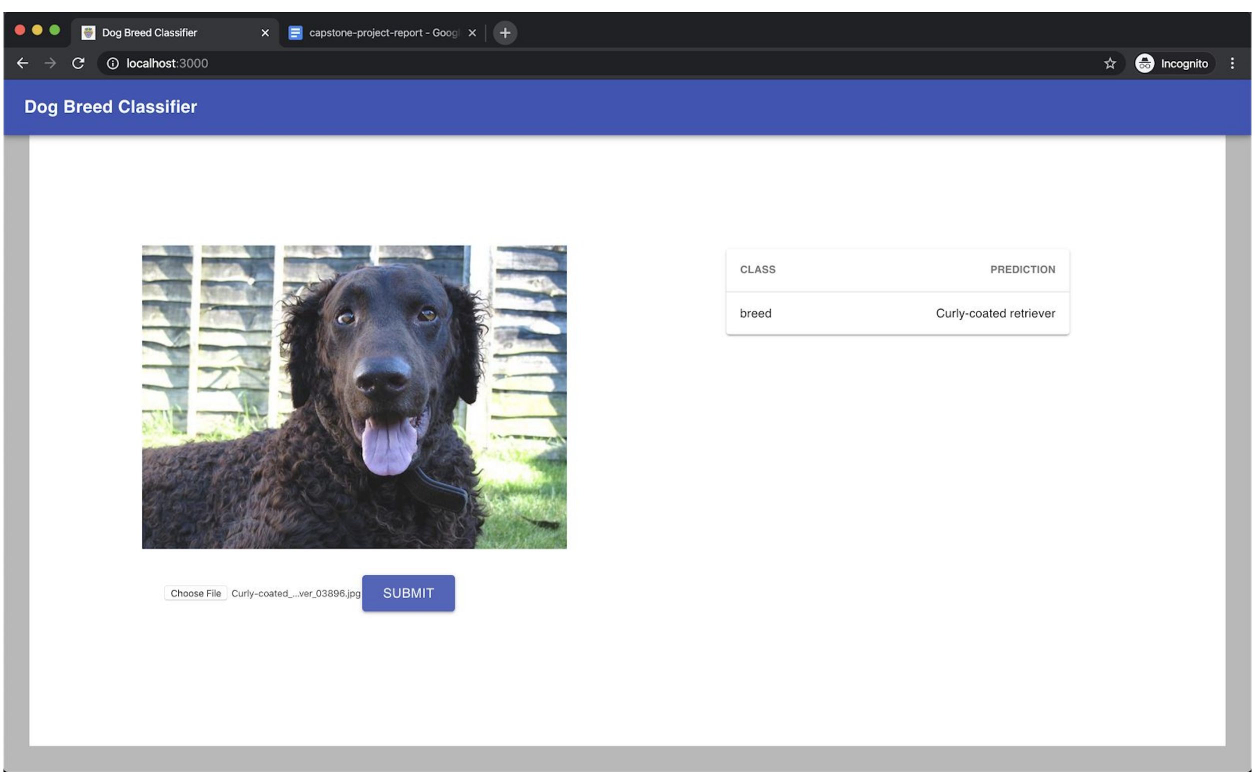
With this model the results are also better than expected. I got 87% accuracy that more than the benchmark (73% accuracy)

At last, I used other images to test the model by web application and Restful API service and found that the model correctly classified the images and the web application worked well for facilitating access for users.

1) **Brittany** breed



2) **Curly-coated retriever** breed



Conclusion

In this project, I aim to implement an end-to-end ML project including a classifier model and a web application for the final model of the dog-breed classification problem. I've implemented the dog-breed classification from the data provided by Udacity that consist of more than 8000 images with exactly 133 dog breeds. Firstly, I study and make an analysis of the split of data, distribution of data, and also the visualization of how the data is distributed. I also preprocess the data by normalized, resized, flipped and cropped to improve robustness of the model in the training phase.

In the implementation phase, I applied the transfer learning technique by using the pre-trained model, modified according to the intended use and also improvising on fine-tuning the hyperparameters such as epochs, dropout value, and learning rate. Then train with a prepared dataset that is split 90% for training, 10% for testing and 10% for validation. From an exploration of many state-of-the-art models, the best result received from Densenet161 pretrained model with 87% accuracy. On making the comparative study with the benchmark model, our model has outperformed the benchmark that got 73% accuracy.

In the last phase of the project, I have developed the web application and Restful API service using ReactJS and Flask. For the purpose of facilitating access for end-users by letting them upload the dog image and get the predicted dog-breed result.