Machine Learning Engineer Nanodegree

Capstone Project: Dogs Breed Classifier

Victor Nascimento
May 28st, 2020

1. Definition

1.1 Project Overview

One of the main topics in computer vision field is the problem of image recognition which consist in detect a assign a label to a given image. With the advance of Deep Learning Computer Vision Engineers can achieve greater results with less efforts in many Image Recognition Tasks[2].

In traditional Computer Vision workflow a specialist in the problem should manually extract and select features from images and then this features were passed to a shallow classifier[2].

Nowadays, using convolutional neuron networks people can achieve great results in multi-label Image classification tasks with less efforts and in largers datasets such as ImageNet[3].

In this project, it is purposed to use convolutional neuron networks to solve a multi-label image classification task, hard to solve, even for humans. The task is to use CNN's to built a Dog Breed Classificator. This is originaly one of the proposed projects in Udacity's AI Nanodegree and more information about the project can be found at Udacity's github repository.[1]

Two Datasets are used int his project, one containing images of human faces and other images of dogs. Both Datasets are available to download in Udacity's github repository.[1] More details about them can be found at section **2** of this report.

1.2 Problem Statement

The objective of this project is to built a dog breed classifier using convolutional neuron networks (CNN). The purpose is to use this classifier in a web app that accepts images of dogs or people from users as inputs and answers which dog breed more resembles to the given input. To achieve this task is planned to use transfer learning to train the CNN.

1https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification

At the end of this project is expected to have a trained CNN capable to classify dog breeds with a minimum of 70% of accuracy.

1.3 Metrics

Since the problem is multi-class classification problem the class **nn.CrossEntropyLoss** is going to be used to train the models. And the function is defined as below[2]:

$$Loss(x, class) = -x[class] + \log\left(\sum_j \exp(x[j])\right) \qquad (1)$$

The metric that is going to be use to quantify the performance of both the benchmark and final models is Accuracy and it is defined as the equation bellow:

$$Accuracy = \frac{True\ Positives + False\ Negatives}{DatasetSize} \qquad (2)$$

---

2. Analysis

2.1 Data Exploration and Exploratory Visualization

In this project, two datasets were used: a dog dataset and a Human dataset. Both datasets are available to download in the Readme file at Udacity's git hub repository.[1]

Since the purpose of this project is to use a dog breed classifier to be use in a web app that will also accept images of persons as input, it is necessary to have a model that identify human faces in pictures. In this regard, the dataset of human faces was used to quantify the performance of the human faces detector.

The Human dataset has in total 13223 images from 5749 people. The resolution of all images from this dataset is (250x250x3).Two image samples from this Dataset can be seen in Figure 1.

The Dogs Dataset is used for two purposes. First, to use to build a dog detector in Images and to serve as input to train the CNN model that is going to be used to detect the dog breeds.

2 https://pytorch.org/docs/stable/nn.html#crossentropyloss

Figure 1: Two images Samples from Human dataset.

In Dogs Dataset a there are images in total 8351 images from 133 different dog breeds. One characteristic of this data set that is worth to mention is that the resolution of the images may vary depending on the image. Therefore a preprocess step was required to reshape the images so all images can have the same resolution. This is a required because the final model will only accept inputs with a given shape which will be discussed in the next sections. More information about that can be seen in section 3. Two images samples of this dataset can be seen in Figure 2.

Figure 2: Two images Samples from Dogs dataset. A Beagle in the left and a Border Collie in the right.

The Dogs Dataset was divided in Three subsets. One for training the CNN which contains 6680 images. One for validation which contains 835 images. One for testing containing 836 images. The dataset was splitted in a way that which subset has at least one image for each dog breed.

2.2 Algorithms and Techniques

Since the problem is a Multi-label classification problem in images with thousand of pixels, try to solve this problem building a neuron network with just Dense

Layers would not be feasible due the quantity of neurons that it would required to represent due the quantity parameters that would need to be trained. Therefore, to build a model capable to predict dog labels from images it was used Convolutional Neuron Networks (CNN's). CNN's are well known to perform well in images datasets by applying kernels in images and maintaining the most relevant features which makes easier to process[3].

In this project it was built two Convolutional Layers. The first one was built to solve the problem, training the weights from scratch. The purpose of this was to use the model to serve as benchmark to compare with the next model. The second model was training using Transfer Learning using the pre-trained weights from a pre-trained model. Two pre-trained models were tested while building the second model, the VGG16 and the Resnet50. Both pre-trained models required that images were pre-processed in the same way. All images should have the minimum resolution of (224x224x3) , should have to be loaded in range [0,1] and then normalized using the same parameters. All information can be found in the Pytorch documentation[4].

To avoid overfitting during training and with the intention to help the model generalize better, some techniques of Data augmentation were used such as flips, resizes, rotations and images crops. More details about how this techniques were applied are described in section 3.

2.3 Benchmark

The Dataset that was used in this project has a total of 133 Breed of dogs, therefore we have 133 labels. That said, if an algorithm that makes random guesses try to predict the breed of a dog based on an image the accuracy would be less than 1%.

In this project it was tried to compare the advantages to use transfer learning when building a CNN over training it from scratch. In the requirements of the project in Udacity's proposal, it is proposed to create a CNN from scratch with a minimum accuracy of 10%. This CNN was created and it was used as a benchmark to compare the performance over the final model that was training using Transfer Learning. Besides that, the minimum accuracy suggested in Udacity's proposal for the final model is 60% but here the objective is try to achieve at least 70% of accuracy.

3 https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

4 https://pytorch.org/docs/stable/torchvision/models.html

3. Methodology

3.1 Data Preprocessing and Implementation

The first step of the project was to build a face detector capable to detect human faces in images. In that step it was used a pre-trained face detector which is an OpenCv's implementation of Haar Feature-based cascade classifier[5] and it's available at OpenCv' github repository[6].

To measure the performance of the face detector, it was take the first 100 images from Human Dataset and 100 images from dogs datasets and it was measured the percentage that was detected an human face in those images. The results are shown at the Table 1.

Table 1: Percentage of images that were detected an human face by the face detector.

	Human Dataset	Dogs Dataset
Percentage	98%	17%

The Face Detector is very efficient in detect human faces in images when there is a person in the image. Although it seems that the face detector is detecting human faces in some images of dogs, the face detector it will only be called when a dog is not recognized in an image. That's why the face detector gives a acceptable performance for the purposes of this project.

The second step was to build a model to detect dogs in images. For this task it was used the pretrained VGG-16 model. This model was trained on Imagenet[7] dataset, a popular dataset with 1000 classes used for image classification. The idea was if, given an image, the output of the model returns a class of a dog (indexes 151 to 258, according with Imagenet dictionary), then a dog would be detected in that image.

To use the pretrained VGG-16 model it is necessary to preprocess the images in a specific way so the model can accept the input. For that, all images were resized to the shape (224x224x3) and all images were normalized using the following parameters: mean=[0.485,0.456,0.406], std = [0.229, 0.224, 0.225]. All this preprocess step was made following the Pytorch documentation[4].

To measure the performance of the dog detector the same procedure of the human face detector was done. The results are shown at the Table 2.

5 https://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html
6 https://github.com/opencv/opencv/tree/master/data/haarcascades
7 http://www.image-net.org/

As shown in the Table 2, the dog detector gives a great performance for the purposes of this project, being able to detect dogs 100% of the images from Dog Dataset.

The third step was to build The Convolutional Neuron Network from Scratch. This is the model that was used as benchmark. First it was necessary to decide the shape of the input tensors. An input size too large would make the CNN bigger and that would require more time and more memory to train it. Likewise, An input size too small would lose some information about the images. Since, the input tensor size of the VGG16 is (224,224,3), it was decided to pick an input size similar to that but with more information. So it was choose an input size of (256,256,3).

Table 2: Percentage of images that were detected Dogs by the dog detector.

	Human Dataset	Dogs Dataset
Percentage	1%	100%

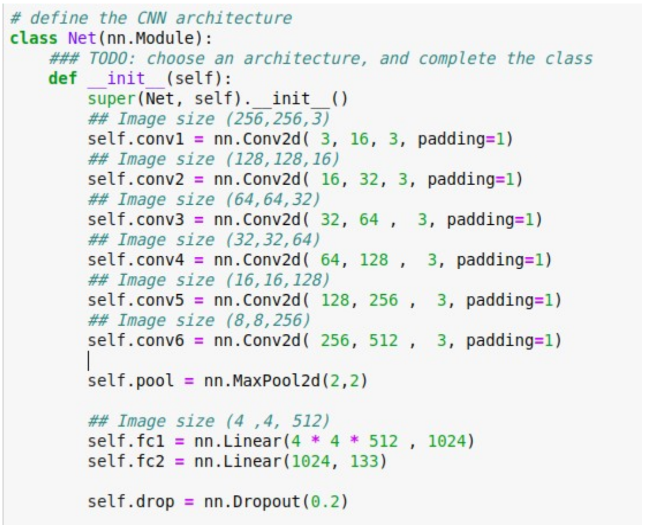
Next, to help the model generalize better, some techniques of Data argumentation were applied. For each input image it was applied a Random Resized Crop, a Random Rotation of 30 degrees , a random horizontal flip and a normalization with the following parameters mean=[0.5,0.5,0.5], std = [0.5,0.5,0.5].

The next step to build the CNN from scratch was decide its architecture. The first idea was that in each pair of convolutional/pooling layer, the output tensor size should be the half of the input tensor size. And the number of the features maps should double. So, it was decided that the covolutional layers would keep the size of the input tensor. And the pooling layers would shrink the size by half. For that, all convolutional layers would have padding equal to one and kernel size of (3x3), that way keeping the size of the tensor, and all pooling layers would have kernel size of (2x2), that way the pooling layer is going to shrink the tensor size by half. The output of the convolutional layers is the passed for two Dense Layers which finally gives the logits for each dog breed. A dropout of 20% was also added at the two last dense layers. The final architecture can be seen in code in the Figure 3.

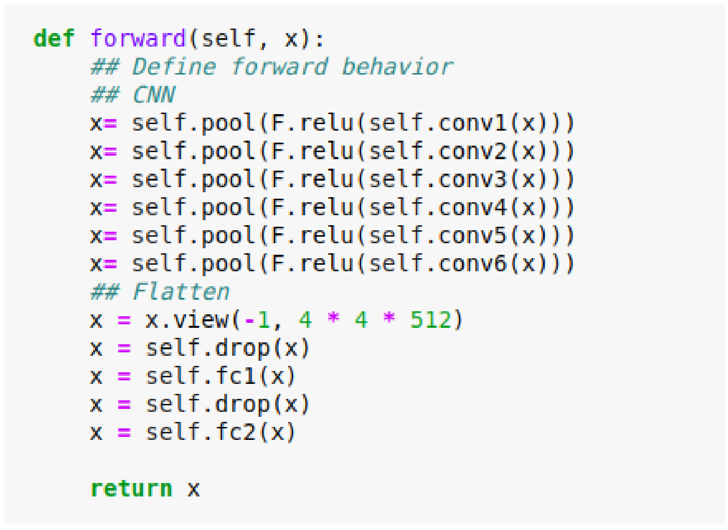
Next, it was defined as loss function the Cross Entropy Loss (described in Section 1) and the Stochastic gradient descent (SGD) as optimizer with a learning rate of 0.01. The model was trained during 100 epochs and, for each epoch that the validation loss decreased, the model parameters were saved.

The CNN trained from scratch obtained an accuracy of 20% when tested in the test set of dog dataset, being able to correctly predict the class of 173 images.

Since the proposed objective was to have at least 10% of accuracy in this step, no further improvements in this model was done.



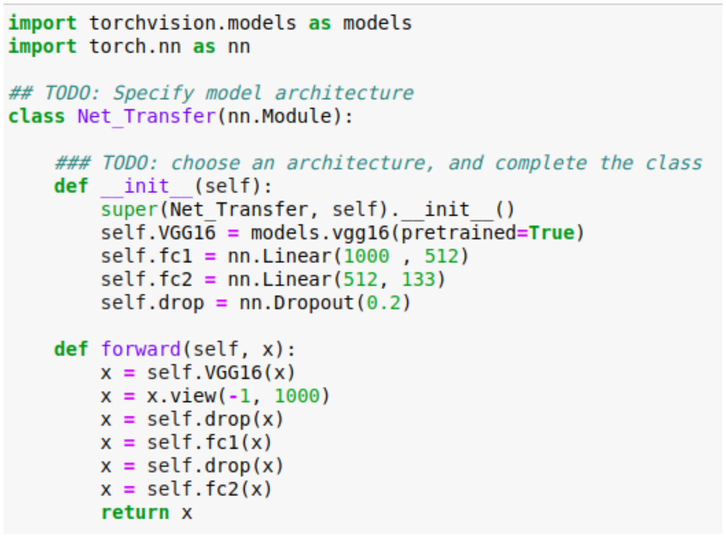
```
# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Image size (256,256,3)
        self.conv1 = nn.Conv2d( 3, 16, 3, padding=1)
        ## Image size (128,128,16)
        self.conv2 = nn.Conv2d( 16, 32, 3, padding=1)
        ## Image size (64,64,32)
        self.conv3 = nn.Conv2d( 32, 64 ,  3, padding=1)
        ## Image size (32,32,64)
        self.conv4 = nn.Conv2d( 64, 128 ,  3, padding=1)
        ## Image size (16,16,128)
        self.conv5 = nn.Conv2d( 128, 256 ,  3, padding=1)
        ## Image size (8,8,256)
        self.conv6 = nn.Conv2d( 256, 512 ,  3, padding=1)

        self.pool = nn.MaxPool2d(2,2)

        ## Image size (4 ,4, 512)
        self.fc1 = nn.Linear(4 * 4 * 512 , 1024)
        self.fc2 = nn.Linear(1024, 133)

        self.drop = nn.Dropout(0.2)
```



```
def forward(self, x):
    ## Define forward behavior
    ## CNN
    x= self.pool(F.relu(self.conv1(x)))
    x= self.pool(F.relu(self.conv2(x)))
    x= self.pool(F.relu(self.conv3(x)))
    x= self.pool(F.relu(self.conv4(x)))
    x= self.pool(F.relu(self.conv5(x)))
    x= self.pool(F.relu(self.conv6(x)))
    ## Flatten
    x = x.view(-1, 4 * 4 * 512)
    x = self.drop(x)
    x = self.fc1(x)
    x = self.drop(x)
    x = self.fc2(x)

    return x
```

Figure 3: Architecture of the CNN trained from scratch

The Forth step of the project was to build the CNN using transfer Learning. The first try was to use the VGG-16 model that was used to built the dog detector and add two Dense Layers with dropout of 20% after its output. The first one with 512 nodes and the last one with 133 nodes which is the number of dog breeds. Since the pretrained model was used, all images were resized to the shape (224x224x3) and all images were normalized using the parameters described in the Pytorch documentation[4]. The Data Augmentation techniques such as Random resized crop, random rotation of 30 degrees , a random horizontal flip were maintained. The architecture of the CNN model using VGG-16 can be seen in code in the Figure 4.



```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
class Net_Transfer(nn.Module):

    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net_Transfer, self).__init__()
        self.VGG16 = models.vgg16(pretrained=True)
        self.fc1 = nn.Linear(1000 , 512)
        self.fc2 = nn.Linear(512, 133)
        self.drop = nn.Dropout(0.2)

    def forward(self, x):
        x = self.VGG16(x)
        x = x.view(-1, 1000)
        x = self.drop(x)
        x = self.fc1(x)
        x = self.drop(x)
        x = self.fc2(x)
        return x
```

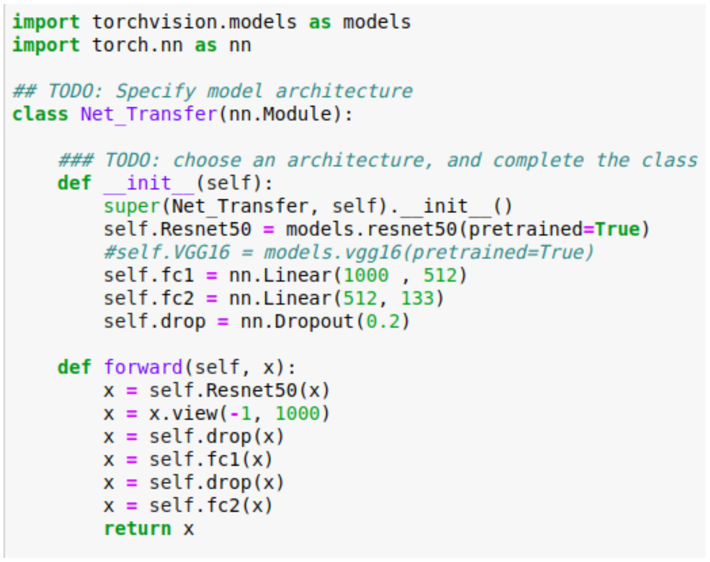
Figure 4: Architecture of the CNN model using VGG-16.

The loss function used was the Cross Entropy Loss and the Stochastic gradient descent (SGD) was the optimizer with a learning rate of 0.01. The model was trained during 30 epochs and, for each epoch that the validation loss decreased, the model parameters were saved.

The CNN model using VGG-16 obtained an accuracy of 69% when tested in the test set of dog dataset, being able to correctly predict the class of 578 images. In Udacity's project description it was suggested that the model trained with transfer learning should have a minimum accuracy of 60% in the test set. Although this model already accomplish this requirement, the desired accuracy is 70%. That said, some refinement were necessary to achieve this goal.

3.2 Refinement

The CNN model trained using VGG-16 didn't achieved the minimum accuracy established (70%). So, some refinement were made in the architecture and in the hyper-parameters.



```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
class Net_Transfer(nn.Module):

    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net_Transfer, self).__init__()
        self.Resnet50 = models.resnet50(pretrained=True)
        #self.VGG16 = models.vgg16(pretrained=True)
        self.fc1 = nn.Linear(1000 , 512)
        self.fc2 = nn.Linear(512, 133)
        self.drop = nn.Dropout(0.2)

    def forward(self, x):
        x = self.Resnet50(x)
        x = x.view(-1, 1000)
        x = self.drop(x)
        x = self.fc1(x)
        x = self.drop(x)
        x = self.fc2(x)
        return x
```

Figure 4: Architecture of the CNN model using Resnet-50.

The main differences were to use Resnet50 as the pretrained model in the place of VGG-16, the learning rate was increased from 0.01 to 0.02 and the model was trained for 20 epochs. The Data augmentation techniques, the optimizer an the architecture of the two dense layers remained the same. The final architecture of the model can be seen in the Figure 5.

The CNN model using Resnet50 obtained an accuracy of 78% when tested in the test set of dog dataset, being able to correctly predict the class of 657 images.

4. Results

4.1 Model Evaluation and Validation

The final step was to build the final code to the web app. For that, it was necessary to choose which model to use and how the would be the pipeline of the algorithm.

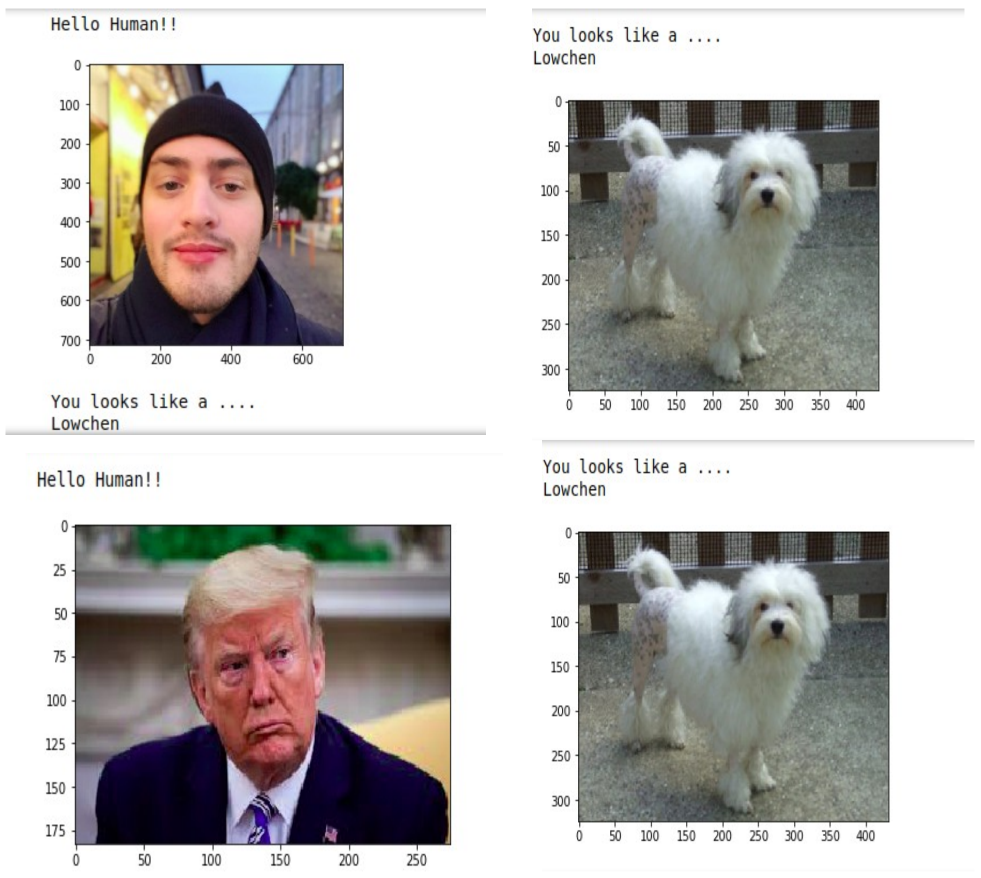
The Table 3 shows the Cross Entropy Loss (defined in section 1) in the validation set in the best epoch for each model, the test loss and the final accuracy in the test set. As it an bee seen in the table, the CNN trained from scratch had the worst performance in the test set although it was the model that take more time to train (total of 100 epochs). The CNN trained with Resnet-50 had the best performance being able to achieve the goal to achieve the minimum accuracy of 70%. Based on these metrics, the CNN is safe to say that the Last model will fit for the purposes of this project.

Table 3: Metrics of the CNN trained models

	Validation Loss	Test Loss	Test Accuracy
CNN from scratch	3.4701	3.5316	20%
CNN with VGG-16	0.9471	1.1024	69%
CNN with RESNET50	0.6609	0.7821	78%

After decide which model to use, the final algorithm for the web app was write. The final pipeline is the following: First, the user send an image, then the dog detector is called. If a dog is detected in the image then the dog breed classifier is called and then the breed of dog is returned. If a dog is not detected then the human face detector is called. If a human face is detected in the image then the dog breed classifier is called and a message if the most resembled dog breed is returned. In case that neither a dog or a human face is detected a message of error is returned. It is worth to mention that before the image is send to the breed dog classifier the image is preprocessed following the same rules described in Section 3, without the Data Augmentation techniques such as Random Crop, Random horizontal flip and etc. This is done to eliminate the possibility of the same image to get an different output if passed over the network twice.

Some additional test with new images were also done to confirm the quality of the model. Two images of humans, two Images of Dogs, a image with more than one dog in it and an image of a cat were passed to the final algorithm. The results can be seen in the images below.



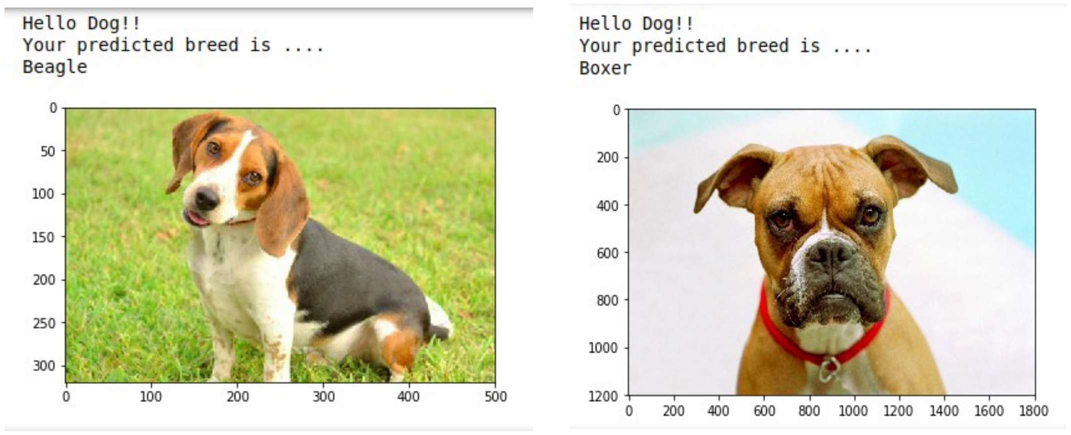
```
Hello Human!!

You looks like a ....
Lowchen
```

```
You looks like a ....
Lowchen
```

```
Hello Human!!
```

```
You looks like a ....
Lowchen
```

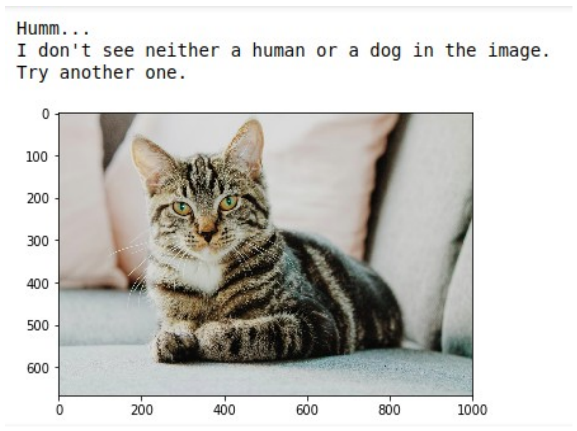
Figure 5: Output of the Human Images.



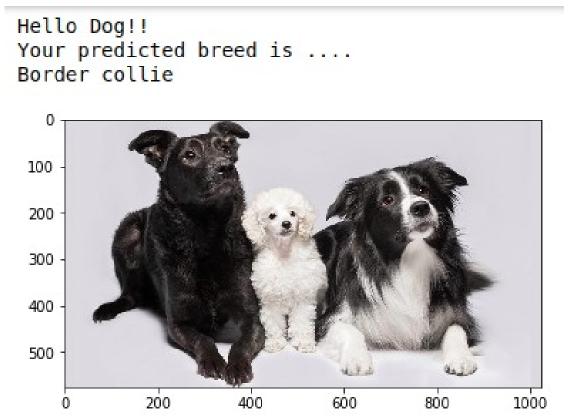
```
Hello Dog!!
Your predicted breed is ....
Beagle
```

```
Hello Dog!!
Your predicted breed is ....
Boxer
```

Figure 6: Output of the Dog Images.



```
Humm...
I don't see neither a human or a dog in the image.
Try another one.
```



```
Hello Dog!!
Your predicted breed is ....
Border collie
```

Figure 7: Output of the image of a cat in the left ans the output of the image of three dogs in the right.

Since there is no way to know for sure which breed of dog a person should be, there is no correct or wrong answer when predict the most resembled breed to a person. Analyzing the output of the dog images, the model seems to do a great job classifying dog breeds. In respect of the two last outputs, the models seems recognize images that neither a dog or a human are present. And when more than one dog are present in the image the model only answers a breed of one of these dogs. Try to predict breeds for more than one dog per image can be a future improvement in the algorithm.

4.2 Justification

Two dog breeds classifiers were trained in this project. The first was a Convolutional Neuron Network trained from scratch which was used as benchmark. The Udacity’s proposal in this part was to achieve a minimum accuracy of 10%. After trained for 100 epochs, this model achieved an accuracy of 20% in the test set.

The Second was a CNN with transfer learning using Resnet50 as pre-trained model. The Udacity’s proposal in this part was to achieve a minimum accuracy of 60%, the personal goal was to achieve at least 70%. After trained for only 20 epochs the model a achieved an accuracy of 78% in the test set.

The final model it’s not perfect but given the purpose of the project it does a good job predicting dog breeds ant it will create a fun experience for the final user. Besides that, comparing the models it is clear the advantages of using transfer learning over training a model from scratch in image classification problems. Besides the fact the the model using transfer learning achieved a better result it also required less time to train it.

5. Conclusion

In this project a Dog Breed Classifier capable to identify 133 dog breeds from images was created using Covolutional Neuron Networks with transfer Learning. The final model achieved an accuracy of 78% in the test set. The model was created with the purpose to be use in a web app.

During the implementation of this project, it was clear the advantages of using deep learning to solve a image recognition problem over the traditional computer vision workflow. Using Deep Learning it was possible to create a dog breed classifier without an specialist in dogs to extract features. By the way, the final model can detect dog breeds better than its creator. It was also clear that training a Convolutional neuron Network requires a lot of resources, being require to use GPU's to speed up the process or using a environment with GPU's available like AWS.

Some future improvements can be done to make a better user experience. First, it is possible to test the model using other pre-trained models to see if a higher accuracy can be achieved. Likewise, some changes in the architecture can also be done such as increasing the number of Dense Layers. Other changes such apply other data augmentation techniques or do a grid search to test different learning rates and optimizers can also be implemented.

Another future improvement can be to modify the model so it can detect more than one dog per image and be able to classify each dog in that image.

References

1. Huang, T. (1996-11-19). Vandoni, Carlo, E (ed.). *Computer Vision : Evolution And Promise* (PDF). 19th CERN School of Computing.

2. Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

3. O'Mahony, Niall, et al. "Deep learning vs. traditional computer vision." *Science and Information Conference*. Springer, Cham, 2019.