UDACITY

Dog Breed Classifier

Ioannis Polymenis

Machine Learning Engineer Nanodegree
Capstone Project Report

Udacity

05 January 2021

The school of:
Artificial Intelligence

Contents

1 Chapter 1: Definition

1.1 Project Overview

The concept of Machine Learning originated in the 1950s when the Computer Science pioneer Alan Turing proposed the question "Can machines think?" in the paper "Computing Machinery and Intelligence" (Turing, 2012) and introduced the famous "Turing test" which explains how an intelligent machine should perform and other concepts that give life to modern Artificial Intelligence (AI) and Machine Learning (ML). The general concept that Turing proposes is how a general purpose computer can learn to perform a specific task on its own. In classical computer programming, the human gives instructions (the program) and data to be processed to the computer and the output is the answer to the problem. In contrast, with machine learning humans give the data and the answers to the computer and the outputs are the rules that the computer developed during the process, and these rules can be further applied to new data to generate novel answers.

The developments in machine learning as well as in deep learning is contineusly evolving espetially the during the last decade in areas of image recognition, face recognition, speech recognition, natural language processing, real-time translation and even the first intelligent system, with Google's Brain project of AlphaGo, which beat a human professional player in the game of Go without human intervention, in 2017 (Silver et al., 2017). Furthermore, in the image classification competition organised by ImageNet, the ILSVRC (ImageNet Large Scale Visual Recognition Challenge), the top five image classification error has been decreasing since its inception in 2009 (Jia Deng et al., 2009) and reached an astonishing 2.3% which is well below the human performance of around 5% error rate (Hu et al., 2018).

For a human image classification and object classification is part of our nature, we can just classify and recognize objects with ease. For computers on the other hand is not so straight forward this process. Thus, machine learning models, and particularly deep learning models, can be used to solve the problem of how a computer can see.Image classification is one of the main domains of deep learning where the deep neural networks play the most noticeable role. In a broad sense, image classification is the ability of a computer to analyze a given image and identify the objects of interest which is referred as a **"class"**. Therefore, a class is simple referred to the object's label which for

instance can be a “car”, “animal”, a “person”, and so on. In the current project, a “Dog Breed Classifier” will be developed since is an excellent oportunity to create classifier which will solve a such challenging task, because even for humans it is difficult to distinct one breed from another.

The complex task of dog breed classification will be accomplished using Convolutional Neural Networks (CNNs), which have been used since the 1990s in applications such as character recognition and are inspired by the brain’s visual system (LeCun et al., 1998; Voulodimos et al., 2018), but their real advantages and benefits for computer vision started with the AlexNet in 2012 in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

1.2 Problem Statement

The aim of the current project is to create a pipeline image classification for a variety of dog breeds. A deep learning model will be used to distinguish the different breeds when a picture of a dog is given by the user. The first step will be to identify whether the given image is a dog or not, and at the next step should identify the correct dog breed. Furthermore, the model should be able to exhibit a reasonable performance on predicting mixed dog breeds. The final step in the model will be to detect whether a dog or human is in the given image, classify the dog breed and classify the dog breed that the human resembles. This is a supervised learning problem, and the learning method is based on labeled training examples and makes predictions on unseen data. Specifically, the training dataset has been appropriately labeled, for instance, in classification between different dog breeds and humans, and the algorithm is training on that dataset.

1.3 Metrics

The current project has as an objective to compere the performance of the model created from scratch with the one used for benchmarking. Thus, the accuracy of the CNN model from scratch can be compared with the pre-trained VGG-16 model for the evaluation metrics. Accuracy can be described as the fraction of predictions that the model got correct and can be shown in the following equation:

$$Accuracy = \frac{Number - of - correct - predictions}{Total - number - of - predictions} \tag{1}$$

Using the the accuracy as metric it is easier to quantify as well as to compare the initial benchmark model with the final model. The accuracy of the model could give us indications of how much the model need to be optimized, for example the customer that will potentially use the app will expect to get correct results on the breed classification. So, if the model get a performance of 50% then need to optimize it to achieve as much as close to optimal. On the other hand, accuracy close to 100% is also not desirable because it is an indication that the network is overfitting.

2 Chapter 2: Analysis

2.1 Data Exploration

The datasets that was used in this project provided by Udacity through the GitHub repository. Firstly, the dog dataset contains images of 133 different breeds and is splitted to training, test, and validation sets totaling of 8351 dog images. The training set constitutes the largest portion of the entire dataset which is 6680 images, the test set has 836 images and the rest 835 images are in the validation set.

Specifically:

1. Train dataset: The train dataset contains 133 folders with the representative dog breeds, and each folder contains 40 to 70 dog images totaling to 6680 images.
2. Validation dataset: The validation dataset contains 133 folders with the representative dog breeds, and each folder contains 4 to 10 dog images totaling to 836 images.
3. Test dataset: The test dataset contains 133 folders with the representative dog breeds, and each folder contains 4 to 10 dog images totaling to 835 images.

```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from glob import glob

# define function to load train, test, and validation datasets
def load_dataset(path):
    data = load_files(path)
    dog_files = np.array(data['filenames'])
    dog_targets = np_utils.to_categorical(np.array(data['target']), 133)
    return dog_files, dog_targets

# load train, test, and validation datasets
train_files, train_targets = load_dataset('datasets/dogImages/train/')
valid_files, valid_targets = load_dataset('datasets/dogImages/valid/')
test_files, test_targets = load_dataset('datasets/dogImages/test/')

# load list of dog names
dog_names = [item[20:-1] for item in sorted(glob("datasets/dogImages/train/*/"))]

# print statistics about the dataset
print('There are %d total dog categories.' % len(dog_names))
print('There are %s total dog images.\n' % len(np.hstack([train_files, valid_files, test_files])))
print('There are %d training dog images.' % len(train_files))
print('There are %d validation dog images.' % len(valid_files))
print('There are %d test dog images.'% len(test_files))
```

```
There are 133 total dog categories.
There are 8351 total dog images.

There are 6680 training dog images.
There are 835 validation dog images.
There are 836 test dog images.
```

Figure 2.1: Dataset Structure.

Furthermore, there is a human dataset which contains images of a variety of people and the total size of the dataset is 13233 images, and these images are used to the performance of the Human Face Detector. Figure 2.1 shows the code that is responsible to load the datasets from the dataset directory, and prints the number of each folder. Also, prints the total dog categories and dog images.

Additionally, the snippet code below and the figure 2.2 shows the actual results of complete dog breeds. The list below is only a small part because the complete one is too large and is not worth to give here the entire printed list.

```
dog_names_array = np.array(dog_names)
for count, dog_names_array in enumerate(dog_names_array):
    print(count, dog_names_array)
```

```
0 Affenpinscher
1 Afghan_hound
2 Airedale_terrier
3 Akita
4 Alaskan_malamute
5 American_eskimo_dog
6 American_foxhound
7 American_staffordshire_terrier
8 American_water_spaniel
9 Anatolian_shepherd_dog
10 Australian_cattle_dog
11 Australian_shepherd
12 Australian_terrier
13 Basenji
14 Basset_hound
15 Beagle
16 Bearded_collie
17 Beauceron
18 Bedlington_terrier
19 Belgian_malinois
```

Figure 2.2: Dog Breeds list.

Lastly, it is worth to know how the dataset is structured and how many images each folder contains. The bellow code does exactly this.

```
from glob import glob


def img_number(train_dog, valid_dog, test_dog):
    list_train = []
    list_valid = []
    list_test = []
    for i in train_dog:
        list_train.append(len(glob(i+"/*")))
```

```
        for j in valid_dog:
            list_valid.append(len(glob(j+"/*")))
            for k in test_dog:
                list_test.append(len(glob(k+"/*")))
    train_max = min(list_train)
    train_min = max(list_train)
    valid_max = min(list_valid)
    valid_min = max(list_valid)
    test_max = min(list_test)
    test_min = max(list_test)

    return train_max, train_min, valid_max, valid_min, test_max, test_min

train_dog = glob('./datasets/dogImages/train/*')
valid_dog = glob('./datasets/dogImages/valid/*')
test_dog = glob('./datasets/dogImages/test/*')

train_max, train_min, valid_max, valid_min, test_max, test_min = \
img_number(train_dog, valid_dog, test_dog)

print("Training set have {} to {} images".format(train_max, train_min))
print("Validation set have {} to {} images".format(valid_max, valid_min))
print("Test set have {} to {} images".format(test_max, test_min))

# Results
Training set have 26 to 77 images
Validation set have 4 to 9 images
Test set have 3 to 10 images
```

2.2 Exploratory Visualization

The images in both the dog dataset and the human dataset, contain colour images in various dimensions, particularly the dog dataset. Thus, it is essential to transform the photos to have a particular size and not be in random dimensions. This transformation is needed because the neural network requires a consistent feed of images with the same dimensions. The code snipet helps us to visualize any image that we want from the dataset.

```
# visualize the dog dataset
import cv2
from glob import glob
import matplotlib.pyplot as plt
%matplotlib inline
def visualise(path, num=10):
    """
    The visualise function allows to visualize any image in the disered
    directory.
    Arguments:
        path: the path in the form of './path/to/directory/'
        num: the number of a random image in the directory besed on the results
             from the load_dataset funtion (e.g. for the training dataset
             between 0 and 6679), defauly value is 10.
    """
    for image in [glob(path+'/*/*')[num]]:
        img = cv2.imread(image)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img)
        image_show = plt.show()
    return image_show
path = './datasets/dogImages/train/'
visualise(path)
```

```
path = './datasets/dogImages/train/'
visualise(path, 20)
path = './datasets/dogImages/train/'
visualise(path, 6679)
```

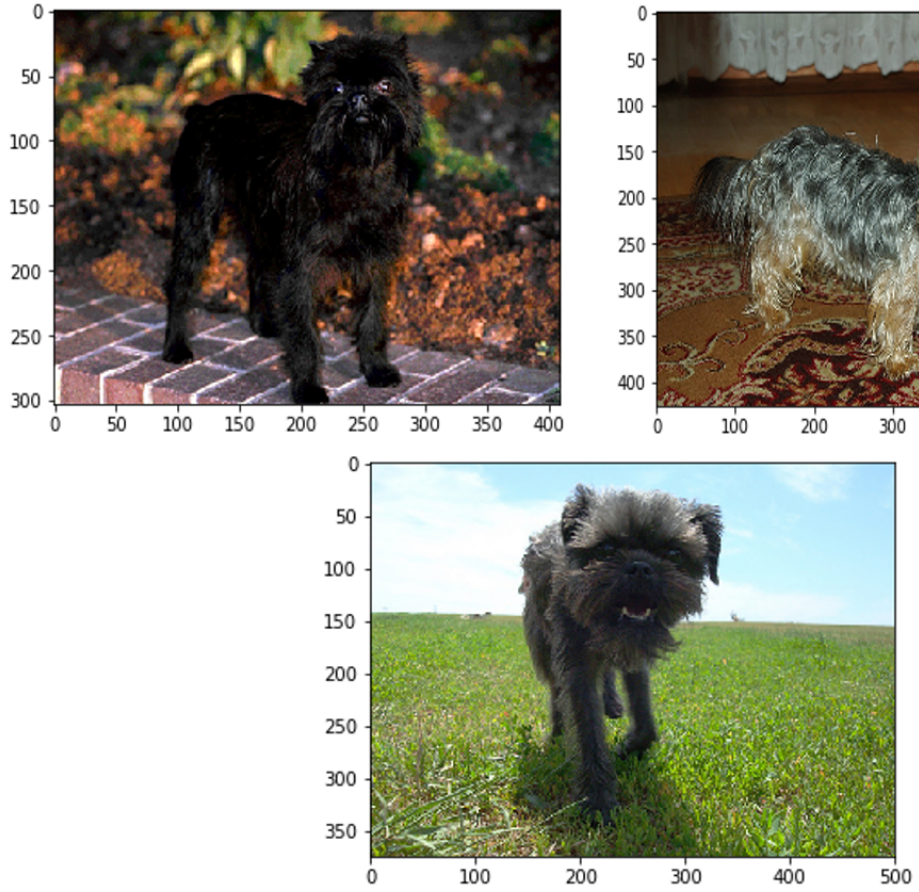


Figure 2.3: Visualize dog breeds from the dataset .

2.3 Algorithms and Techniques

The current project uses a Convolutional Nural Network (CNN) for dog breed detection. The development of deep learning models for image classification has as a main component of the convolutional neural networks. Convolutional Neural Networks (CNNs) have been used since the 1990s in applications such as character recognition and inspired by the brain's visual system (LeCun et al., 1998; Voulodimos et al., 2018), but their real advantages and benefits on computer vision

started with the AlexNet in 2012 as mentioned above.

A CNN architecture includes the following components, the convolutional layers, the pooling layers, and lastly the fully connected layers and every element have a different role to play. To better understand how a CNN network works need to visualise it as figure 2.4 shows, and that network is the underlying architecture of the most computer vision applications that using deep learning. Each convolution transforms the input image and extracts different features from it with the first layers to extract only low-level features (edges and lines). At the same time, as the network deepens the feature extraction have a higher level of information and resulting to accurately mapping the input data, and thus to recognise or to classify the desired objects in the image.

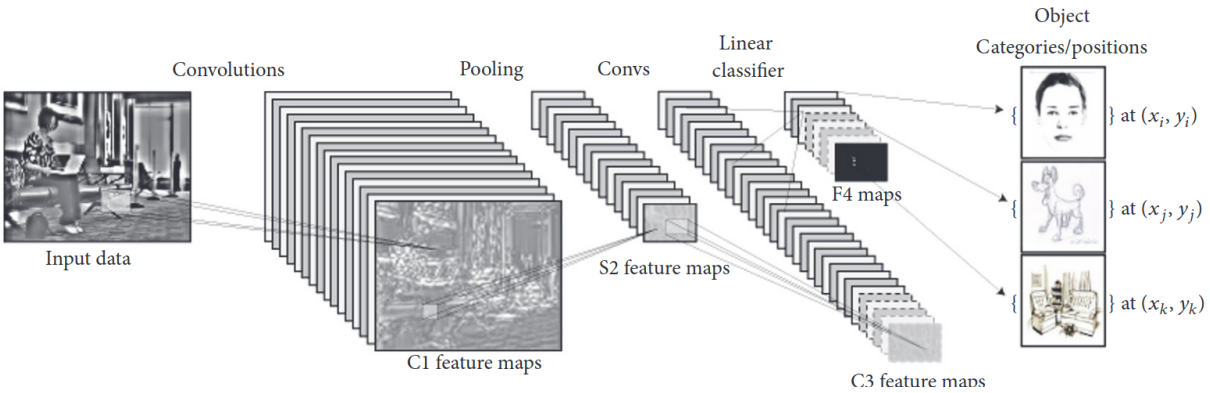


Figure 2.4: Convolutional Neural Network of LeNet (Voulodimos et al., 2018)

CNN are using various filters or kernels to convolve the input image and generating a variety of feature maps which leads to faster learning times comparing to the traditional fully connected layers. This, parallel feature extraction that CNN's offering has as a trade-off in computational power, since hundreds or even thousands of computations need to be done at the same time. Thus, Deep Learning and particularly the CNNs in computer vision applications need the use of Graphics cards (GPUs) to compute a given image dataset faster and was able to achieve those performances only in the last decade with the developments in GPU hardware.

A CNN architecture also using features which are called **hyperparameters** and can summurized in the following:

- The number of hidden layers. The more layers, higher the computational cost and less layers can lead to underfitting.

- Number of epochs, which is the number of times that the complete dataset will pass trought the network
- Batch size, which is how many images will pass at once during a single training step.
- The activation function, which gives the non linear behaviour to the network and can be simgoid, ReLu, softmax etc.
- The learning rate, which will determine how fast or slow the network learns.

In this project, the network uses the ReLu activation function in the hidden convolutional layers and softmax activation function at the last Dense layer. The classification is needed to recognize the dog breeds. Additionally, a Dropout layer introduced to minimize the network's overfitting. This architecture gives exceptional results in image classifications tasks. Initially, a CNN model is created from scratch for the dog breed classification. Figure 2.5 indicates the summary of the architecture.

The next step was to create a CNN model using Transfer Learning. This was done to improve the model's accuracy which should attain at least 60% of accuracy on the test set. The following architectures were available:

- VGG-19
- ResNet-50
- Inception
- Xception

Different architectures were tried and final the Xception model was chosen to contain an addition convolutional layer with 64 filters and one MaxPooling layer. Comparing the Xception model with only the GlobalAveragePooling and the Xception architecture with the Convolution-MaxPooling layers, it can be seen that the training parameters almost doubled from 272,517 to 532,997. The network achieved a better overall performance in the second case was significantly higher at 85.6% whereas in the architecture without the additional Convolution-MaxPooling layers was about 80%. Additionally, the architecture tested with 124 filters, and although the parameters doubled to almost 1.2 million, the gain in performance was not so significant. Also, with the use of 124 filters, the training time was increased, which was not worthing to use such a large number of filters. Thus, the final model was trained in the Xception architecture with the additional Convolution-MaxPooling

layers and additionally, since the scope of the present project is to classify dog breeds, the softmax activation function at the last Dense layer was used.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               25088500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               66633
=================================================================
Total params: 25,165,677.0
Trainable params: 25,165,677.0
Non-trainable params: 0.0
_________________________________________________________________
```

Figure 2.5: Convolutional Neural Network model summary

2.4 Benchmark

As a benchmark for the model the beating of the predefined values was used. The values in the notebook were 1%for the CNN from scratch model, and 60% for the model using Transfer Learning. For the model from scratch the test accuracy was 7.4% while for the Transfer Learning model was 85.6%

3 Chapter 3: Methodology

3.1 Data Preprocessing

The data pre-processing was done according to Keras specifications which requires a 4D array (aka a 4D tensor) as an input shape. To obtain the required tensor, the input images need to have the same size which 224x224 pixels was chosen. Then, the image is converted to an array and finally transformed to a 4D tensor. The following code snippet shows the above process.

```
from keras.preprocessing import image
from tqdm import tqdm
def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)
def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

3.2 Implementation

The first step is to write the Human detector using OpenCV's implementation of Haar feature-based cascade classifier to detect human faces. For face detection it is a standard procedure to convert the images to gray scale, and this can be done as follows:

```
# load color (BGR) image
img = cv2.imread(human_files[5])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

The following function detects if there is a human face in the given image and returns True when detects a face and False when there is no human face.

```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

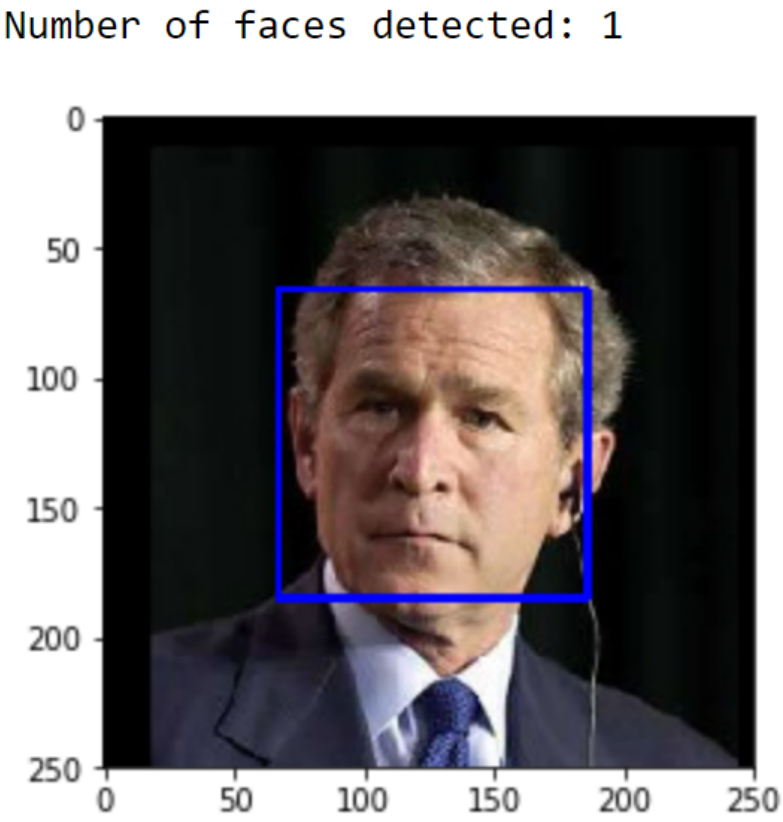


Figure 3.1: Face detection using the face_detector function

Next is the implementation of the dog detector. If no human face is detected from the face detector, the image will be processed by the dog detector to check if dogs are present. The dog detector uses a pre-trained ResNet-50 model based on the ImageNet weights. Given an image, the ResNet-50 model should return a prediction, found in those ImageNet categories, for the object in any given image.

The following function detects if there is a dogin the given image and returns True when detects a dog and False when there is no dog.

```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    prediction = ResNet50_predict_labels(img_path)
    return ((prediction <= 268) & (prediction >= 151))
```

When the dog_detector function finds a dog, the algorithm will continue and the image will used as an input to the dog breed classifier. For this model the given from the notebook architecture used mainly because it is a well known an classic starting CNN architecture which includes convulution layers, maxpooling layers, dropout layers as well as using ReLu as an activation function for all layers except the last one which uses a softmax activation since a classification is what need to achieved by the model. The steps followed are as follows:

1. Import the necessary libraries and define a sequential model
2. Add a two dimensional Convolutional layer followed by a maxpooling layer and in each layer the parameters such as the activation function and the filters were defined as well. The first convolutional layer includes the input shape of the images, and in this architecture is 224 by 224 pixels in 3 RGB scale (depth of the image).
3. Additionaly, the depth of the Conv-MaxPoll layers is repeated 3 times in total with different filter size begining from 16, 32 and finally 64.
4. At the end of the last Conv-MaxPoll layer a Flatten layer was used to connect the outputs to the Dense layer.
5. A Dropout of 30% was used to avoid overfitting which again is connected to a Dense layer.
6. The last Dense layer has a size of 133 because this is the number of dog breeds that examined in this project and a softmax activation function is used for the classification task.
7. Finally, a summary of the model is printed to obtain a visualisation of it.

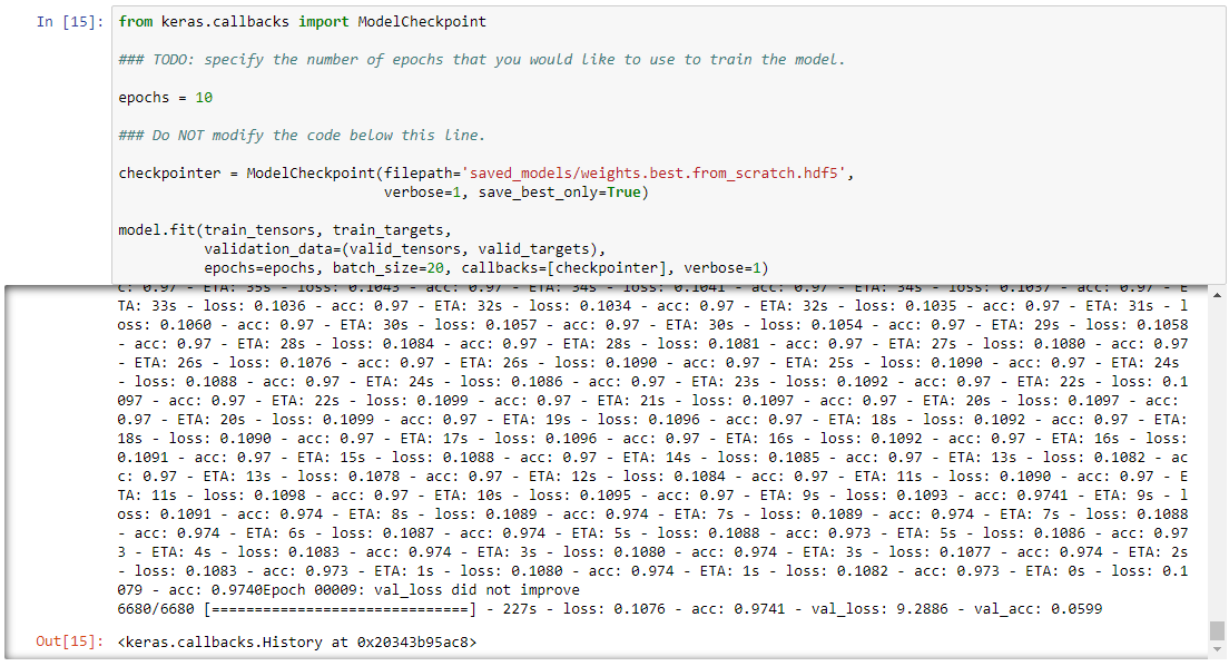
3.3 Refinement

Initially, the test accuracy was 7.4%, which is above the required 1%, but continued to be poor performance. Fortunately, there is the option to use a pre-trained model choosing between different architectures, as mentioned in section 2.3. During the refinement process, different architectures were used with a different number of layers. It found that the Inception and the Xception models give the best results, such the Xception model used as a final detection algorithm. Additionally, a Conv2D layer with 64 filters and a MaxPooing2D layers was added to the Xception model to increase the test accuracy, which was initially about 80% to 85.6%. Further increase of convolutional layer and filters resulted in more computational intense models and increased training time. The Xception model has also been tuned to mach the 133 output dog breed classes, and training a more depth network with more filters in a GPU could result in more accurate classifier. For this project a local machine was used.

4 Chapter 4: Results

4.1 Model Evaluation and Validation

The first CNN model tested as a dog breed classifier was the CNN model from scratch has a training loss of 0.1076 and comparing it with the validation loss of 9.2886 we can see that the model is overfitting since training_loss is much smaller than the validation_loss. The loss values are getting better when using the pre-trained VGG-16 model with training loss of 7.0566 and validation loss of 7.6807. The loss values of VGG-16 model are relatively close to one another, that is, the model neither overfits nor underfits. The model chosen for the final step of the dog breed classification application was the Xception model with the addition of the Convolution-MaxPooling layers. This model has been selected because it offers the best test accuracy, from the models that test in this project, without spending many resources and time to train the model. Furthermore, as mentioned in the section 3.3, the model after 20 epochs of training, the validation loss is 0.8773, and the validation accuracy is 0.8527 which is exceptionally better if will be compared with the CNN model from scratch that has validation loss of 9.2886 and validation accuracy of 0.0599. Figures 4.1, 4.2, and 4.3 present the actual relustus as obtained in the notebook.



```
In [15]: from keras.callbacks import ModelCheckpoint

         ### TODO: specify the number of epochs that you would like to use to train the model.

         epochs = 10

         ### Do NOT modify the code below this line.

         checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.from_scratch.hdf5',
                                        verbose=1, save_best_only=True)

         model.fit(train_tensors, train_targets,
                   validation_data=(valid_tensors, valid_targets),
                   epochs=epochs, batch_size=20, callbacks=[checkpointer], verbose=1)

         TA: 33s - loss: 0.1036 - acc: 0.97 - ETA: 32s - loss: 0.1034 - acc: 0.97 - ETA: 32s - loss: 0.1035 - acc: 0.97 - ETA: 31s - l
         oss: 0.1060 - acc: 0.97 - ETA: 30s - loss: 0.1057 - acc: 0.97 - ETA: 30s - loss: 0.1054 - acc: 0.97 - ETA: 29s - loss: 0.1058
         - acc: 0.97 - ETA: 28s - loss: 0.1084 - acc: 0.97 - ETA: 28s - loss: 0.1081 - acc: 0.97 - ETA: 27s - loss: 0.1080 - acc: 0.97
         - ETA: 26s - loss: 0.1076 - acc: 0.97 - ETA: 26s - loss: 0.1090 - acc: 0.97 - ETA: 25s - loss: 0.1090 - acc: 0.97 - ETA: 24s
         - loss: 0.1088 - acc: 0.97 - ETA: 24s - loss: 0.1086 - acc: 0.97 - ETA: 23s - loss: 0.1092 - acc: 0.97 - ETA: 22s - loss: 0.1
         097 - acc: 0.97 - ETA: 22s - loss: 0.1099 - acc: 0.97 - ETA: 21s - loss: 0.1097 - acc: 0.97 - ETA: 20s - loss: 0.1097 - acc:
         0.97 - ETA: 20s - loss: 0.1099 - acc: 0.97 - ETA: 19s - loss: 0.1096 - acc: 0.97 - ETA: 18s - loss: 0.1092 - acc: 0.97 - ETA:
         18s - loss: 0.1090 - acc: 0.97 - ETA: 17s - loss: 0.1096 - acc: 0.97 - ETA: 16s - loss: 0.1092 - acc: 0.97 - ETA: 16s - loss:
         0.1091 - acc: 0.97 - ETA: 15s - loss: 0.1088 - acc: 0.97 - ETA: 14s - loss: 0.1085 - acc: 0.97 - ETA: 13s - loss: 0.1082 - ac
         c: 0.97 - ETA: 13s - loss: 0.1078 - acc: 0.97 - ETA: 12s - loss: 0.1084 - acc: 0.97 - ETA: 11s - loss: 0.1090 - acc: 0.97 - E
         TA: 11s - loss: 0.1098 - acc: 0.97 - ETA: 10s - loss: 0.1095 - acc: 0.97 - ETA: 9s - loss: 0.1093 - acc: 0.9741 - ETA: 9s - l
         oss: 0.1091 - acc: 0.974 - ETA: 8s - loss: 0.1089 - acc: 0.974 - ETA: 7s - loss: 0.1089 - acc: 0.974 - ETA: 7s - loss: 0.1088
         - acc: 0.974 - ETA: 6s - loss: 0.1087 - acc: 0.974 - ETA: 5s - loss: 0.1088 - acc: 0.973 - ETA: 5s - loss: 0.1086 - acc: 0.97
         3 - ETA: 4s - loss: 0.1083 - acc: 0.974 - ETA: 3s - loss: 0.1080 - acc: 0.974 - ETA: 3s - loss: 0.1077 - acc: 0.974 - ETA: 2s
         - loss: 0.1083 - acc: 0.973 - ETA: 1s - loss: 0.1080 - acc: 0.974 - ETA: 1s - loss: 0.1082 - acc: 0.973 - ETA: 0s - loss: 0.1
         079 - acc: 0.9740Epoch 00009: val_loss did not improve
         6680/6680 [==============================] - 227s - loss: 0.1076 - acc: 0.9741 - val_loss: 9.2886 - val_acc: 0.0599

Out[15]: <keras.callbacks.History at 0x20343b95ac8>
```

Figure 4.1: CNN model from scratch training results

```
In [21]: checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.VGG16.hdf5',
                                        verbose=1, save_best_only=True)

         VGG16_model.fit(train_VGG16, train_targets,
                  validation_data=(valid_VGG16, valid_targets),
                  epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)
```

```
209 - acc: 0.553 - ETA: 0s - loss: 7.1267 - acc: 0.552 - ETA: 0s - loss: 7.1098 - acc: 0.5540Epoch 00018: val_loss improved f
rom 7.76776 to 7.72549, saving model to saved_models/weights.best.VGG16.hdf5
6680/6680 [==============================] - 1s - loss: 7.1395 - acc: 0.5522 - val_loss: 7.7255 - val_acc: 0.4659
Epoch 20/20
6500/6680 [============================>.] - ETA: 1s - loss: 7.2547 - acc: 0.550 - ETA: 1s - loss: 7.3031 - acc: 0.545 - ETA:
1s - loss: 7.3991 - acc: 0.538 - ETA: 1s - loss: 7.3649 - acc: 0.535 - ETA: 1s - loss: 7.1596 - acc: 0.548 - ETA: 1s - loss:
7.1512 - acc: 0.549 - ETA: 1s - loss: 7.1171 - acc: 0.551 - ETA: 1s - loss: 7.0764 - acc: 0.552 - ETA: 1s - loss: 7.0685 - ac
c: 0.553 - ETA: 1s - loss: 7.0412 - acc: 0.553 - ETA: 1s - loss: 6.9889 - acc: 0.557 - ETA: 1s - loss: 6.9886 - acc: 0.557 -
ETA: 1s - loss: 7.0022 - acc: 0.556 - ETA: 0s - loss: 7.0068 - acc: 0.556 - ETA: 0s - loss: 6.9445 - acc: 0.561 - ETA: 0s - l
oss: 6.9791 - acc: 0.559 - ETA: 0s - loss: 7.0033 - acc: 0.557 - ETA: 0s - loss: 7.0708 - acc: 0.553 - ETA: 0s - loss: 7.0814
- acc: 0.552 - ETA: 0s - loss: 7.1133 - acc: 0.550 - ETA: 0s - loss: 7.1175 - acc: 0.550 - ETA: 0s - loss: 7.0935 - acc: 0.55
1 - ETA: 0s - loss: 7.0963 - acc: 0.551 - ETA: 0s - loss: 7.0790 - acc: 0.552 - ETA: 0s - loss: 7.0595 - acc: 0.553 - ETA: 0s
- loss: 7.0606 - acc: 0.553 - ETA: 0s - loss: 7.0345 - acc: 0.555 - ETA: 0s - loss: 7.0160 - acc: 0.556 - ETA: 0s - loss: 7.0
476 - acc: 0.554 - ETA: 0s - loss: 7.0532 - acc: 0.554 - ETA: 0s - loss: 7.0213 - acc: 0.556 - ETA: 0s - loss: 7.0214 - acc:
0.556 - ETA: 0s - loss: 7.0287 - acc: 0.5557Epoch 00019: val_loss improved from 7.72549 to 7.68073, saving model to saved_mod
els/weights.best.VGG16.hdf5
6680/6680 [==============================] - 1s - loss: 7.0566 - acc: 0.5542 - val_loss: 7.6807 - val_acc: 0.4683
```

```
Out[21]: <keras.callbacks.History at 0x20345bc7a90>
```

Figure 4.2: VGG-16 model training results

```
In [71]: ### TODO: Train the model.
         checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.xception2.hdf5',
                                        verbose=1, save_best_only=True)

         history = model_xception_cnn.fit(train_xception, train_targets,
                                  validation_data=(valid_xception, valid_targets),
                                  epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)
```

```
- ETA: 3s - loss: 0.0326 - acc: 0.991 - ETA: 3s - loss: 0.0325 - acc: 0.991 - ETA: 3s - loss: 0.0323 - acc: 0.991 - ETA: 3s -
loss: 0.0322 - acc: 0.991 - ETA: 3s - loss: 0.0320 - acc: 0.991 - ETA: 3s - loss: 0.0317 - acc: 0.992 - ETA: 3s - loss: 0.031
4 - acc: 0.992 - ETA: 3s - loss: 0.0313 - acc: 0.992 - ETA: 3s - loss: 0.0312 - acc: 0.992 - ETA: 3s - loss: 0.0315 - acc: 0.
991 - ETA: 3s - loss: 0.0314 - acc: 0.991 - ETA: 3s - loss: 0.0316 - acc: 0.991 - ETA: 3s - loss: 0.0318 - acc: 0.991 - ETA:
2s - loss: 0.0315 - acc: 0.991 - ETA: 2s - loss: 0.0313 - acc: 0.991 - ETA: 2s - loss: 0.0314 - acc: 0.991 - ETA: 2s - loss:
0.0312 - acc: 0.991 - ETA: 2s - loss: 0.0310 - acc: 0.991 - ETA: 2s - loss: 0.0308 - acc: 0.991 - ETA: 2s - loss: 0.0306 - ac
c: 0.991 - ETA: 2s - loss: 0.0302 - acc: 0.991 - ETA: 2s - loss: 0.0300 - acc: 0.991 - ETA: 2s - loss: 0.0299 - acc: 0.991 -
ETA: 2s - loss: 0.0296 - acc: 0.992 - ETA: 2s - loss: 0.0294 - acc: 0.992 - ETA: 2s - loss: 0.0292 - acc: 0.992 - ETA: 2s - l
oss: 0.0292 - acc: 0.992 - ETA: 1s - loss: 0.0319 - acc: 0.991 - ETA: 1s - loss: 0.0317 - acc: 0.991 - ETA: 1s - loss: 0.0316
- acc: 0.992 - ETA: 1s - loss: 0.0316 - acc: 0.991 - ETA: 1s - loss: 0.0315 - acc: 0.991 - ETA: 1s - loss: 0.0316 - acc: 0.99
1 - ETA: 1s - loss: 0.0314 - acc: 0.991 - ETA: 1s - loss: 0.0312 - acc: 0.991 - ETA: 1s - loss: 0.0314 - acc: 0.991 - ETA: 1s
- loss: 0.0312 - acc: 0.991 - ETA: 1s - loss: 0.0311 - acc: 0.991 - ETA: 1s - loss: 0.0308 - acc: 0.991 - ETA: 1s - loss: 0.0
312 - acc: 0.991 - ETA: 1s - loss: 0.0310 - acc: 0.991 - ETA: 1s - loss: 0.0309 - acc: 0.991 - ETA: 0s - loss: 0.0307 - acc:
0.991 - ETA: 0s - loss: 0.0307 - acc: 0.991 - ETA: 0s - loss: 0.0305 - acc: 0.991 - ETA: 0s - loss: 0.0302 - acc: 0.991 - ET
A: 0s - loss: 0.0301 - acc: 0.991 - ETA: 0s - loss: 0.0299 - acc: 0.991 - ETA: 0s - loss: 0.0298 - acc: 0.992 - ETA: 0s - los
s: 0.0297 - acc: 0.992 - ETA: 0s - loss: 0.0296 - acc: 0.992 - ETA: 0s - loss: 0.0295 - acc: 0.992 - ETA: 0s - loss: 0.0293 -
acc: 0.992 - ETA: 0s - loss: 0.0292 - acc: 0.992 - ETA: 0s - loss: 0.0291 - acc: 0.992 - ETA: 0s - loss: 0.0289 - acc: 0.992
- ETA: 0s - loss: 0.0288 - acc: 0.992 - ETA: 0s - loss: 0.0290 - acc: 0.9920Epoch 00019: val_loss did not improve
6680/6680 [==============================] - 11s - loss: 0.0290 - acc: 0.9921 - val_loss: 0.8773 - val_acc: 0.8527
```

Figure 4.3: Xception model training results

4.2 Justification

Using three differnt models, the CNN model from scratch, the pre-trained VGG-16 and the Xception model with the additional Convolution-MaxPooling layers the best prediction accuracy was achieved buy the Xception architecture, obtaining test accuracy of 85.5%. The results showing that a more complex and sophisticated models can noticeable improve the performance, especially when carefully adjust the hyperparameters, also the models training time can dramatically improved if a GPU is available.

5 Chapter 5: Conclusion

5.1 Free-Form Visualization

It is important to visualize the output results of the dog breed classification algorithm, that is, to see how it performs when a variety of images are presented as an input, which should recognize a human face if the image contains a human and then resemble the predicted dog breed of what the human looks like. If the image contains as dog, then the classifier should identify its breed. Finally, if the image does not have neither a human face nor a dog should return a statement that will states that. For the test of the algorithm the folder **my_images** contains 11 different images of dogs, humans and of an airplane. The results are presented in the following images.

Figure 5.1: Dog breed classification algorithm output

As it can be seen, the algorithm's output is what expected, and the overall performance is excellent, given the simple architecture. Given that the Xception model has been trained for only 20 epochs and have an additional Convolution-MaxPooling layer, 85% of performance is quite

reasonable. Though, when the algorithm tested using custom images cannot recognize all the given breeds correctly. This misclassification is because the dataset with the 133 dog breeds does not contain the breeds that have been used. For example, given the images of a Bluenose Pitbull and a Pug, the algorithm classifies them as American_staffordshire_terrier and Chinese shar-pei. The misclassification is because the tested breeds do not exist in the dataset, but the algorithm performs exceptionally well since the classifying outputs are close to the real breeds. To overcome the missclassification problem few key points could be improved:

1. Increase the dog breed dataset and include more breeds.
2. Improve the model accuracy using different techniques such as data augmentation can be used in the training set.
3. Using a pre-trained model which would have been trained in a very large dataset (e.g. ImageNet dataset) and fine-tune the dog-breed classifier in that model.

5.2 Reflection

The primary learning aspect of this project as well as from the entire Machine Learning NonoDegree is that data mainly drive machine Learning. The data that will be passed through the network, any neural network, need to be processed, to clean, and generally to prepared and accordingly formatted for the given task; otherwise, the results could be wrong or misleading. It was also clear that the best practice is to use an already pre-trained model, such as the Xception or Inception architecture, to achieve reliable performance. This is because these networks have been designed by the field experts and have been trained in very large datasets, such as the ImageNet dataset, and give the best output. The pre-trained networks can be used and reconfigured to execute the needed task, with the only effort to fine-tune the pre-trained network.

Additionally, the CNN network design from scratch was valuable because it required basic architecture development, using the Keras framework. This design required a depth understanding of a neural network, the importance of hyperparameters, and the basic principles of a Convolutional layer, a MaxPooling layer, the Dense layer, and the Dropout layers functioning. It was also made clear how and why to use specific activation functions such as the **softmax** and the **relu** activation functions.

Finally, it was made clear that for complex and sophisticated architectures a GPU is needed for quicker training times.

5.3 Improvement

Although the project achieved a quite good test accuracy some of the results were not correct such as the missclassification of the Bluenose Pitbull and the Pug as American_staffordshire_terrier and Chinese shar-pei. Thus, a dataset with a larger variety of dog breeds is needed. Also, using more capable hardware can improve not only the training time, but to allow to use more complex networks such as GANs. Also, the use of more capable hardware, will allow to tweak the hyperparameters such as the training epochs, the batch size and the filters in each layer.

6 References

Hu, J., Shen, L. & Sun, G. (2018) 'Squeeze-and-Excitation Networks', in *Proceedings of the ieee computer society conference on computer vision and pattern recognition.* [Online]. 2018 pp. 7132–7141.

Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li & Li Fei-Fei (2009) '*ImageNet: A large-scale hierarchical image database*', in [Online]. 2009 pp. 248–255.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE.* 86 (11), 2278–2323.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T. & Hassabis, D. (2017) Mastering the game of Go without human knowledge. *Nature.* 550 (7676), 354–359.

Turing, A.M. (2012) 'Computing machinery and intelligence', in *Machine intelligence: Perspectives on the computational model.* [Online]. Taylor; Francis. pp. 1–28.

Voulodimos, A., Doulamis, N., Doulamis, A. & Protopapadakis, E. (2018) Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience.* 2018.