Machine Learning Engineer Nanodegree

Capstone Project
Dog Breed Classifier

Julien Weiss
July 25th, 2020

I. Definition

PROJECT OVERVIEW

Classifying elements on images is a study field that has improved a lot in recent years and has become critical for many autonomous systems. One classical classification problem is the classification of dogs according to their apparent breed. As some breed look very similar to another and though some very different, it is a challenging problem for a lot of state of the art algorithms. Machine Learning, more specifically Deep Learning and Convolutional Neural Networks (CNNs) have proved to be very good at such multi-class image classification problems.
In this project, I built an app capable to predict the breed of a dog, given a dog image. If a picture of a human was provided to the app, it will say which dog breed is the most resembling. If no dog and no human were detected, it will output a corresponding message. For this project I relied on CNNs, especially a simplified version of a VGG11, made from scratch as benchmark and for the final app, a pre-trained VGG19, which, using transfer learning, was adapted to our problem.
The project was suggested and a template was given by Udacity.

PROBLEM STATEMENT

Our goal is to build a Dog Breed Classifier app, that, provided an image, will respectively predict the breed of the dog, if a dog is detected in the image, or the most resembling dog breed if a human face is detected in the image, or output an error message if neither a dog nor a human face are detected in the image.
This multi-class classification problem will involve detecting dogs and human faces in images, training and deploying a CNN model with PyTorch to predict the breed of a dog detected in a picture and building a small app which allows a user to input an image and get a prediction of the dog breed our model thinks it sees. Our problem will be evaluated against its accuracy for the dog and human face detection, and the dog breed classification and we will use a CNN model made from scratch as a benchmark. For training the model we will use CrossEntropyLoss.

For our approach, we will follow a 6 steps workflow.
• Step 1: Import datasets.
• Step 2: Detect humans using a pre-trained model.
• Step 3: Detect dogs using a pre-trained VGG-16 model.
• Step 4: Create a CNN from scratch to classify dog breeds.
• Step 5: Create a CNN using Transfer Learning to classify dog breeds.
• Step 6: Write and test an algorithm / app to classify dog images into the detected dog breed or human faces to the most resembling dog breed. The app handles images in which neither a dog nor a human are detected and return an error message to the user.

METRICS

For our classification problem with multiple classes and a large, rather balanced dataset, we relyied on accuracy as a metric. As we first wanted to detect dogs and human faces on the input pictures, we used accuracy as an evaluation metric for the dog and human face detection. We then evaluated our CNN model performance against its accuracy in predicting the correct dog breed given a dog picture.
Accuracy is defined as follows:

$$Accuracy = \frac{True_P + True_N}{Size_D}$$

Where $True_P$ and $True_N$ are respectively the number of true positive results and of true negative results. $Size_D$ is the overall size of the dataset we tested on.

II. Analysis

DATA EXPLORATION

As input data for our project we required pictures, as we aimed to classify images. We relied on the dog pictures and human pictures datasets provided by Udacity: “dog dataset” and “human dataset”.
The dog dataset has a size of 8351 pictures in total which are split into purpose directories for training (80%), validation (10%) and testing (10%).
Within each purpose directory, each dog breed for our multi-class classification problem has its directory of pictures. There are 133 possible dog breeds and thus directories, each containing different number of pictures of different sizes.
The human dataset has a size of 13233 pictures organized in 5750 subdirectories corresponding to a person’s name, containing different number of images. All images have same size (250x250). This dataset will only be used to test a pre-trained human face detector.

Fig. 1: Images from the dog dataset

ALGORITHMS AND TECHNIQUES

As mentioned in the project overview and problem statement, we made use and implemented four different algorithms, i.e. in our case Machine Learning models, for the specific purposes in our Dog Breed Classifier.

- In order to detect the presence of a human face in an image that we would be input to our app, we used a pre-trained human face detector. For this, OpenCV provides a Haar feature-based cascade classifier that detects human faces on black and white images, and returns the coordinates of the square(s) in which a face was detected. We used this model as a face detector, returning True, when the number of human faces in an image is larger that 0.

- In order to detect the presence of a dog in an image, we used a VGG16 CNN pre-trained on ImageNet. ImageNet is a dataset of millions of urls, that link to images of objects among 1000 classes. We used this pre-trained model to infer categories of dogs in an image and whenever the model predicts the presence of a class within the range of possible dog classes, our dog detector returns True.

- For the actual breed classification of a given dog image, we first implemented a CNN from scratch. As CNN are slow to train, we decided to base our model on a simplified architecture of the ConvNet configuration A (VGG11) described in "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION" by Karen Simonyan and Andrew Zisserman. We adapted the fully connected layers to fit the problem of classifying among 133 dog breeds. We used a LogSoftMax function in the end, as we trained the network using Negative Log Loss as criterion. The gradient descent optimization algorithm we will use is adam.

- In order to improve performance and to save time, using transfer learning for solving deep learning problems which would require a lot of heavy training and computing is a good idea. For our dog breed classification problem, using most of the performance of a pre-trained torchvision model seemed adapted, as they are trained in ImageNet and are among the best in class for image classification tasks. We chose a VGG19 network with batch normalization, pre-trained on ImageNet.

BENCHMARK

Our CNNs models will be compared against two indicators. The CNN model made from scratch will be compared to a random model: A random guess model would roughly have 1 in 133 chance to predict the correct dog breed (less than 1% accuracy), provided a random dog image. Thus aiming for 10% accuracy for our CNN model from scratch is a good indicator that our model is going in the right direction.
For the CNN model that uses Transfer Learning, we will aim for at least 60% accuracy.

III. Methodology

DATA PREPROCESSING

All images used as input for our CNN models (from scratch and with Transfer Learning) require some pre-processing. Further, we distinguish between the pre-processing applied to training data and to validation and test data.

- For the training set, we will first rotate the image randomly within a range of 30 degrees. Then the image is sized to 255 pixels, randomly cropped between 8% and 100% of its size with a random ratio and resized to 224 pixels. It is then randomly flipped horizontally. The image is then convert to a tensor and and normalized the input image. This augments our dataset as it allows for less overfitting.

- For validation and testing, we resize the image to 255 pixels, center crop the image to 224 pixels. The image is then converted to a tensor and normalized.

IMPLEMENTATION

For the implementation or our CNN made from scratch, an overview of the architecture is provided in Table 1.

Operations	**Layers**	**Parameters**	**(Output) Tensor Dimension**
Input	**RGB Image Tensor**	-	224x224x3
Convolutional Layer and Rectifed Linear Unit	**Conv1 + ReLu**	kernel_size=3, padding=1	224x224x64
Pooling Layer	**MaxPool**	2,2	112x112x64
Convolutional Layer and Rectifed Linear Unit	**Conv2 + ReLu**	kernel_size=3, padding=1	112x112x128
Pooling Layer	**MaxPool**	2,2	56x56x128
Convolutional Layer and Rectifed Linear Unit	**Conv3 + ReLu**	kernel_size=3, padding=1	56x56x256
Pooling Layer	**MaxPool**	2,2	28x28x256
Convolutional Layer and Rectifed Linear Unit	**Conv4 + ReLu**	kernel_size=3, padding=1	28x28x512
Pooling Layer	**MaxPool**	2,2	14x14x512
Convolutional Layer and Rectifed Linear Unit	**Conv5 + ReLu**	kernel_size=3, padding=1	14x14x512
Pooling Layer	**MaxPool**	2,2	7x7x512
Fully Connected Layer and Rectified Linear Unit and Dropout	**Linear 1+ ReLu + Dropout**	Dropout=0.2	512
Fully Connected Layer and Rectified Linear Unit and Dropout	**Linear2 + ReLu + Dropout**	Dropout=0.2	256
Fully Connected Layer and LogSoftMax function.	**Linear3 + LogSoftMax**		133

Table 1: CNN from scratch architecture.

Our input tensor comes in the format 224 x 224 x 3 and we apply a series of 2d convolution layers with a 3x3 kernel and padding of 1, and pooling layers to it. Some convolution layers increase the number of feature maps, the pooling layers halve the size of the feature maps. We keep track our our tensor dimension and after the convolution and pooling layer, we get a 7 x 7 x 512 tensor, which we reshape into a 1D tensor. Eventually, we pass through 3 fully-connected layers, apply dropout between them, LogSoftMax and get the output of dimension 133, which corresponds to the log probabilities for the possible dog breeds.

We train this network over 25 epochs, using Adam as gradient descent optimization alogorithm, with a learning rate of 0.001 and obtain an accuracy of 20%, which is good, given that we aimed for a 10% accuracy for this CNN made from scratch.

REFINEMENT

As we aim for higher accuracy, we wish to improve our CNN model significantly, but given the heavy computing induced by training, we decided to use Transfer Learning for our refined model. As stated before, we chose to apply Transfer Learning with a pre-trained VGG19 model using Batch Normalization.

In order to adapt this pre-trained network to our Dog Breed Classification problem, we had to adapt the output layer (linear / fully connected layer) to the classes (133 possible breeds of dogs) of our classification problem. For this we froze the gradients of the other hidden linear layers, changed the last linear layer and added a LogSoftMax function, as we wished to use NegativeLogLoss as criterion for our model. During the training of the customized network, only the gradients of the last linear layer were changed. The overview of this process is provided in Table 2.
We train this model over 5 epochs, using adam as gradient descent optimization algorithm and a learning rate of 0.001.

Operations	**Layers**	**Gradients**	**(Output) Tensor Dimension**
pre-Trained VGG19_BN (last layer excluded)	**VGG19_BN**	**Frozen**	**4096**
Last Layer of pre-trained VGG19_BN (Replaced)	Linear + Softmax (Replaced)	-	1000
Custom Last Layer	**Linear + LogSoftMax**	**To be trained**	**133**

Table 2: Architecture of customized VGG19_bn (Transfer Learning).

IV. Results

MODEL EVALUATION AND VALIDATION

We evaluated our human face detector and our dog detector on a small dataset (100 samples) extracted from the human dataset and dog dataset.

- Human face detector: Respectively 98% and 17% of the images in the extracted human and dog files have a detected human faces.
- Dog detector: Respectively 0% and 100% of the images in the extracted human and dog files have a dog detected.
- CNN made from scratch: The model achieved 20% accuracy on the test dataset. Hence, this proved that a CNN model was a good approach to our problem, as this accuracy is much greater than a random guess among the possible breeds (Less than 1% accuracy). We have a better result that our goal value of 10% accuracy.

- CNN with Transfer Learning (VGG19_BN pre-trained in ImageNet): The customized model implemented with Transfer Learning achieved an accuracy of 82% on the test dataset.

JUSTIFICATION

The final result of our refined model (custom VGG19_BN pre-trained in ImageNet) shows that it is much more accurate than the from scratch model (81% accuracy vs 20% accuracy). It achieves a much better accuracy than our goal of 60%. We believe that the achieved performance is enough to provide for an interesting user experience, when running the app on personal images.

V. Conclusion

FREE-FORM VISUALIZATION

To conclude our work, we put together the human face detector, dog detector and custom pre-trained VGG19-BN model and ran those on images that the user provided as input.

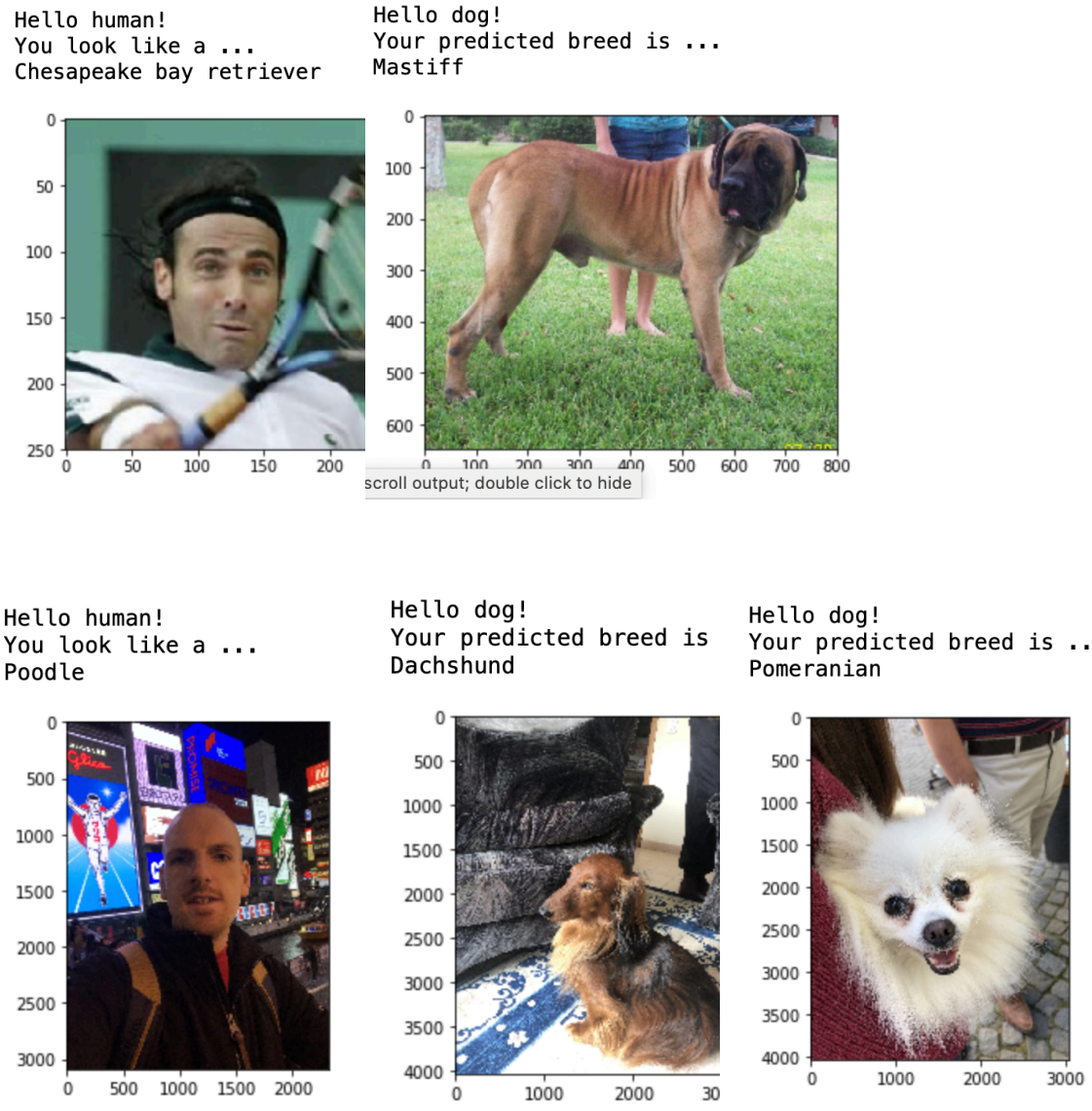

Fig. 2: Dod Breed Classification examples

The algorithm for this is straightforward:
- For a given image, we apply the dog detector an on it. If there is a dog on the image, we infer the dog breed with our CNN model. If no dog is detected, we look for human faces.
- If a human face is detected on the image, we apply our CNN model to the image and print a rather funny message with the inferred dog breed output as the dog that most resembles the human.
- In other cases, an error message alerts the user that no dog or human were detected.

REFLECTION

We saw that applying Deep Learning Techniques, especially Convolutional Neural Network allows for good performance in image recognition and in our case dog breed classification. Some pre-trained models allow easy and quick detection of human faces or dog in images. Making a CNN from scratch is a valid approach to understand and tweak some learning parameters but this is time consuming as such a network requires heavy computing for training and training has to be repeated with different hyper parameters (learning rate, epochs) to achieve a promising result. Thats why this project highlights the benefits of Transfer Learning which allows to use an already trained network, train only its last layer(s) for our problem and achieve a good accuracy and a faster training time as we needed only 5 epochs.

IMPROVEMENT

We could improve our algorithm by tweaking its training data, its hyperparameters used for training and its architecture. For examples, we could:
- Train over a larger number of epochs.
- Use more custom layers after the pre-trained model.
- Use more training data and augment it even more by tweaking our training pre-processing pipeline.

REFERENCES

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

2. Image classification with CNNs:
https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8

ImageNet Classification with CNNs:
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

3. PyTorch documentation:
https://pytorch.org/docs/stable/nn.html#loss-functions

4. Deep Learning Review:
https://s3.us-east-2.amazonaws.com/hkg-website-assets/static/pages/files/DeepLearning.pdf