Convolutional Neural Networks (CNN) Project: Dog Breed Classifier

Artem Shramko

August 24, 2020

1 Definition

1.1 Project Overview

The project is devoted to building a dog breed classification app based on the Convolutional Neural Networks (CNN). The core of the project is to develop an end to end Machine Learning pipeline, which takes user input in form of a graphical image and returns a prediction of the dog breed.

For the entertainment purposes, the model is also intended to be able to accept pictures of humans as an input and return the name of the dog breed the submitted person on a picture resembles the most.

The project comprises of several problems that are to be solved via application of Machine Learning techniques. Among them are: object recognition (identify human and/or dog in a picture), ETL pipeline (preprocessing raw images as a model input), object classification with high interclass homogeneity.

1.2 Problem Statement

Given an image, identify whether a dog or a human is present on it. Based on the input, classify the dog's breed in the first case, or predict, which dog the person on the photo resembles in the letter. The predicted breed must be within 133 classes present in the dataset. If neither a dog nor human is present on the submitted picture, return an error. The algorithm must be able to accept images of different formats, sizes and resolutions and further transform them into the required by the model format.

The solution to the described problem is presented as an application that accepts image as an input and returns predicted dog breed (if applicable) as an output.

The proposed solution consists of the following steps:

1. Identify whether dog or human is present in the picture.
2. Predict resembling dog breed with CNN model.
3. Return predicted dog breed with a sample picture.

In the first step two separate models are applied. Both models return binary output: True if dog (human) is present in the picture, False otherwise. If both models return False, further steps are not performed and exception is raised. Otherwise, predicted class (dog or human) is passed further.

At step two the main model is applied. A Convolutional Neutal Network is trained solely on dog data and returns one of 133 dog breeds as an output. However, it can also accept human image as an input. In this case the neural network returns the dog breed that human facial features resemble the most.

Third step describes UX. In this particualar problem setting, a simple printed image with predicted class is foreseen as an output.

The described solution can further be embedded in a web and/or mobile application, which is, hovewer, out of scope of this project.

1.3 Metrics

For the first step (human/dog identification) a pre-trained OpenCV model is applied for human recognition. For the dog recognition a pretrained pytorch model VGG16 is suggested as a benchmark. Further models, such as SqueezeNet and ResNet151 are used. Models are compared based on classification accuracy. Models are tested on both human dataset (minimize False Positives) and dog dataset (minimize False Negatives). Metric applied:

$Accuracy : (TP + TN)/(TP + FP + TN + FN)$
Accuracy defines the fraction of predictions that the model has classified correctly.

For the 2nd step - dog breed cassification - a built from scratch CNN model is compared against a transfer learning based model. The selected metric for multiclass classification problem is a Cross-Etnropy Loss function.

$$CrossEntropyLoss = \frac{-1}{N} \sum y \times \log \hat{y}$$

The cross entropy metric comes from the Information Theory and calculates the dirrence between the true and estimated distribution. In out multiclass discrete problem y and $\hat{y}$ represent the true and predicted classes respectively. This is a classic metric for multiclass classification problems due to its ability to accept softmax function outputs as an input.

For example, our true values vector is $y = [0, 1, 0, 0]$ and predicted vector is $\hat{y} = [0.2, 0.9, 0.1, 0.3]$. Then the Cross-Entropy Loss can be calculated as $L = -(0 * log(0.2) + 1 * log(0.9) + ...)$.

It can be seen, that when only 1 true class is present, the Cross-Entropy Loss will only take into account how close the predicted value to True class is, since other 0 classes will be ignored.

2 Analysis

2.1 Data Exploration

The data used in this project has been provided by Udacity. It consists of two datasets: dog images and human images.

Human images dataset is not divided into train and test data and is based on the Labeled Faces in the Wild dataset, which is considered as a benchmark for face verification. Images are sorted into folders with several pictures of a particular person in each folder. Overall there are 13233 pictures of 5750 different humans in the dataset. Pictures are primarily face shots with face almost always being in focus of the picture.

The dog dataset has been provided by Udacity and comprises of 8351 dog images of 133 different dog breeds. It is related to the famous Stanford Dog Dataset. The data used for this project can be acquired by following this link.

The dataset is split into train (6680 images), validation (835) and test (836) image folders. So train data comprise 80% of the data, whereas test and validation form 10% each.

2.2 Exploratory Visualization

Human dataset

We further explore the datasets by visualizing some samples. Figure 1 provides randomly selected examples of human images. As we can see, images are centered around the face of a person, with the standardized image sizes of $250px \times 250px$. All images are colored, having three color channels.

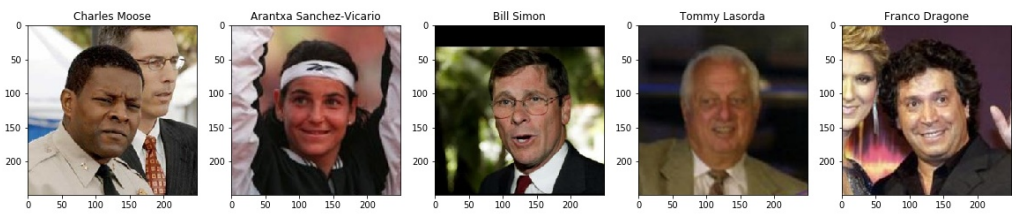

Figure 1: Example of human pictures

Analyzing class distributions. Figure 2 shows the amount of pictures per individual. It appears, that significant outliers are present, with having up to 530 pictures of one individual in the dataset. Nevertheless, since the human dataset will not be used for model training, this observation can be neglected. The median amount of pictures per each person is 1.

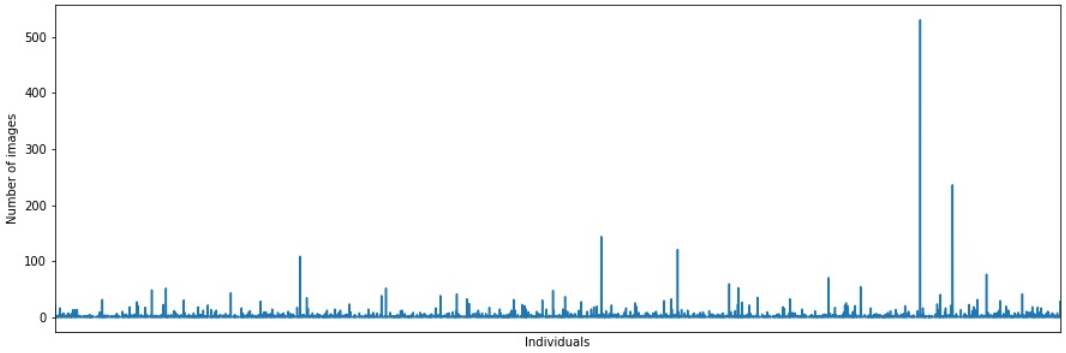

Figure 2: Number of pictures per individual

Dog dataset

Figure 3 shows randomly selected images of different dog breeds. As it can be seen, images vary in terms of their shapes, orientation ad resolution. Some pictures are of a medium size (approx. $600px \times 400px$), some are rather large with more than $2000px$ per dimension.

One of the shortcomings of the dataset that can be spotted right away is the content of pictures. In the examples below, second picture contains two dogs (of the same breed) in it. In the last picture, however, human is present. This might have an adverse effect on the model performance.

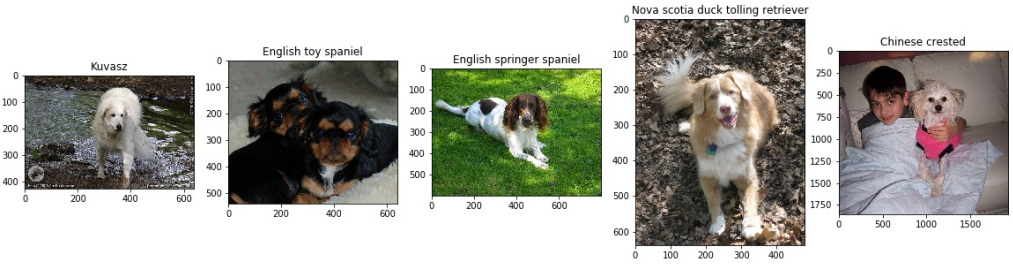

Figure 3: Example of dog pictures

If we look at the distribution of samples per class (Figure 4), the breed pictures are relatively evenly distributed with mean value of 50 and standard deviation of 11.8 pictures per class. Lowest amount if 26, max is 77.

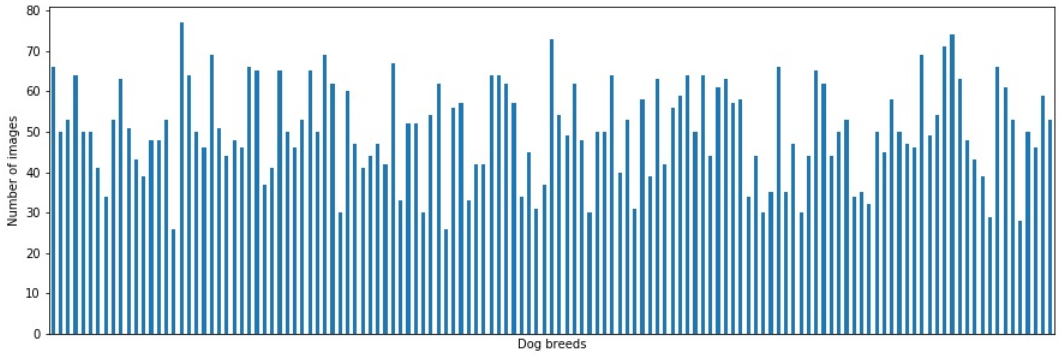

Figure 4: Number of pictures per dog breed

One of the specifics that describe this dataset is that it has a relatively low intraclass correlation and (for some breeds) significantly high interclass correlation. Thus, the model will be challange to be able to ignore some features like color for f.e. labrador retrievers (Figure 5), which come on golden, brown and black colours. And be able to differentiate between different kinds of spaniels, which share a lot of similarities, but are considered as separate classes in the dataset.

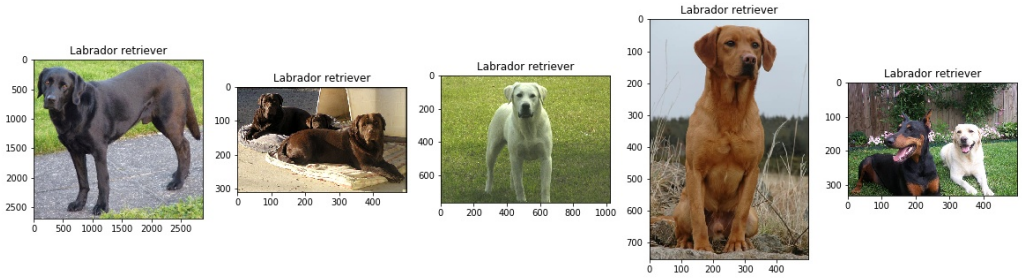

Figure 5: Example of labrador retriever pictures

2.3 Algorithms and Techniques

As described in the problem statement, the solution procedure entails two important steps:

1. Identify if a dog or a person is present in the image
2. Apply the classifier to predict the breed

Both tasks require an application of image classification techniques. The first step is conducted by using two separate models, one for identifying human faces, and the other one for identifying dogs in the image.

For human face identification, a Haar feature-based cascade classifier model (Viola and Jones, 2001) is used. The pretrained models implementation is provided by the one of the most popular computer vision libraries in Python - OpenCV.

For the main task of the project, namely dog breed recognition, a Convolutional Neural Network (CNN) has been used. It became a standard in the industry over the last decade since the famous AlexNet has first dominated the ImageNet competition (Krizhevsky et al, 2012).

CNNs are a class of deep neural network architectures that are most successful in visual processing tasks due to the convolutional layers. The convolutional layers can be viewed as filters that apply simple mathematical operations, like element vise as well as dot multiplications, pooling operations and classic for NN activation functions (like ReLu) on the input tensors. Unlike the other CV algorithms, convolutional layers learn the parameters for filers instead of using the hardcoded filters like in cascade model. Thus, CNNs have following hyperparameters:

Convolutional layers

- Convolutional kernels defined by a width and height
- Amount of input and output channels
- Stride
- Padding

Max pooling

Technique used to reduce dimensionality. Hyperparameters:

- Kernel width and height
- Stride
- Padding

Fully connected layers

Also known as linear layers. The model architect has to decide on such hyperparameters as number of input and output features. The Neural Network itself adjusts feature weights and biases during the error backpropagation procedure.

Further techniques such as dropouts, batch normalization etc. can be embedded in the model structure.

Besides, model architect can further optimize the model by deciding of following parameters:

- Epochs (number of passes of the entire dataset to the model)
- Batch size (number of images processed at each step)
- Optimizer (technique used to calculate gradients and backpropagate losses)
- Learning rate (gradient multiplier, defines how fast model adjusts the weights in networks given feedback)
- Momentum (can be viewed as "velocity" during gradient calculation. Takes into account previous changes in gradient. The technique is used to avoid falling into local minimum during optimisation)

2.4 Benchmark

For the first step, object identification, two models were suggested in the project setup. The cascades face detection model for human face recognition. The model achieves 100% accuracy on human dataset (True positives), but also classifies 12% of dog images as humans (False positives). Closer examination of miss-classified dog images has identified two reasons. On some images, humans are indeed present. On the others it was the False Positive case (primarily corgis, for some reason).

For the dog identification step, the VGG16 model has been suggested. It is a CNN model, that achieves 96% accuracy on the dog dataset (4% are False Negatives) and 99% accuracy on human dataset (False Positive).

For the second step, namely dog breed classification, a derived from scratch CNN is compared to the one built by using transfer learning technique. The imposed accuracy requirement for the first one is 10%, whereas for the letter its 60%.

3 Methodology

3.1 Data Preprocessing

The CNNs used for the project have been developed using the state of the art Python library for deep learning PyTorch. Pytorch CNN implementations requires input tensors to comply to certain requirements. In particular, tensor dimensions must represent: *[batch size, number of channels (colors), image width, image height]* .

Following preprocessing steps have been applied to the dog images:

- Standardize size of the image: resize to $224px \times 224px$ and crop at center
- Transform image to a tensor

- Normalize image by subtracting mean vector *[0.485, 0.456, 0.406]* and dividing by standard deviation vector *[0.229, 0.224, 0.225]*. Amount of elements in the vector represents the channels. This is done to normalize the image to *[-1, 1]* scale, which has become a common practice in the industry,

Besides, following data augmentation techniques have been applied in order to increase diversity in training data:

- Randomly flip images horizontally
- Randomly affine image up to 20%.

The preprocessed input batch looks as follows:

Figure 6: Example of preprocessed batch of images

Initially, other data augmentation techniques, such as random (not centered) crop and random image scaling have been applied. However, they appeared to have a negative effect on the training set, since dogs were often cut out of the images.

3.2 Implementation

The project comprises of three main steps:

1. Develop object identifier
2. Develop dog breed classifier
3. Build pipeline with user endpoint that accepts image as an input and returns predicted class

3.2.1 Step 1: Object Detection

This step can be further divided in two subsections: human detection and dog detection models.

Human detection

Given the suggested Haar cascades face detection model has proven to be very accurate when provided a human picture, and since no further manipulations with human faces dataset was foreseen, the Haar cascades model has been adapted as is in the final project workflow.

The human detection part is wrapped in a function *face_detector* which accepts path to the image and performs following actions:

1. Load pretrained OpenCV cascades model params from *haarcascade_frontalface_alt.xml*.
2. Read image at given path via OpenCV package
3. Convert image to grayscale
4. Pass preprocessed image to the model
5. Return $True$ if face is detected, else $False$

Dog detection

For the dog detection, a pretrained VGG16 CNN model was suggested as a benchmark. I have further comapred its performance on human and dog datasets against ResNet152 and SqueezeNet1_1 models. Following metric values were obtained:

Model	*Accuracy on Human dataset*	*Accuracy on Dog dataset*
VGG16	0.96	0.99
ResNet152	0.95	1.00
SqueezeNet	0.95	0.98

Table 1: Model comparison on dog detection

Given the alternative models did not provide significant improvement, the VGG16 model has been further kept.

The procedure is similar to the one described for human detection and is implemented in the *dog_detector* function. Similar to $face\_detector$, it accepts a path to the image, opens image, completes preprocessing steps, passes image to the model and return binary output. In this case, PyTorch implementation of VGG16 has been used. The preprocessing operations are also conducted via PyTorch utility function. The open image is converted to a tensor of shape $[1, 3, 224, 224]$.

The applied VGG16 model has been trained on the ImageNet dataset with 1000 classes, which containes a wide variety of dog breeds. If the predicted class lies within an Interval [151, 269], the image is classified as containing a dog.

3.2.2 Step 2: Dog Breed Classification

The problem to be solved in this step is assign the input image containing dog (or human) picture to one of the 133 dog breeds. Two CNN models are developed as potential solution: a built from scratch CNN model and a model derived by using a transfer learning technique.

The preprocessing steps applied to dog dataset are described in the Section 3.1.

Model from Scratch

Final architecture of the model was derived as a result of an iterative procedure and can be summarized as follows:

1. Conv Layer: 3 input channels, 6 output channels, kernel size: 3 by 3 pixels with 1px padding.
2. Conv Layer: 6 input channels, 16 output channels, kernel size: 3 by 3 pixels with 1px padding.
3. Conv Layer: 16 input channels, 32 output channels, kernel size: 3 by 3 pixels with 1px padding.
4. Conv Layer: 32 input channels, 64 output channels, kernel size: 3 by 3 pixels with 1px padding.
5. Conv Layer: 64 input channels, 128 output channels, kernel size: 3 by 3 pixels with 1px padding.
   Flatten output tensor of shape [1, 128, 7 ,7] into [1, 6272]. Dropout 20%.
6. Fully Connected Layer: 6272 input features, 256 output features. Dropout 20%. Activation: ReLu.
7. Fully Connected Layer: 256 input features, 512 output features. Activation: ReLu.
8. Fully Connected Layer: 512 input features, 133 output features. Activation: Softmax

Each convolutional layer is followed by Max Pooling: 2 by 2 pixels kernel with 2 pixels stride. Activation function: ReLu. Batch normalization of output channel.

Stochastic Gradient Descent with learning rate 0.001 and momentum 0.9 has been selected as an optimization technique. Cross Entropy Loss has been used as a loss function. The model was trained for 20 epochs with 10 images per batch.

Transfer learning model

The core idea behind transfer learning is to use a model, which has been pretrained on a different dataset, and apply it to new classification task. For the object recognition task models that have been pre trained on the ImageNet dataset are one of the most widely used.

Basically there are two options for transfer learning: fine-tune all layers of existing pretrained model given new data, or "freeze" params and add one Linear layer to predict classes in your data. Since we have a relatively small dataset, and the original ImageNet dataset contains a lot of different dog breeds already, picking the first technique could lead to overfitting. Which is why a pretrained model with fixed parameters has been selected, and a linear layer with 133 output channels has been added.

In terms of the model selection, I went with ResNet18 due to its lower training time given the limitations in computation power. Optimization procedure, loss function and training setup are similar to the setup in the model from scratch.

3.2.3 Step 3: Pipeline

The final application, that combines models from steps 1 and 2 into a pipeline, accepts path to an image as an input and has the following logic.

1. Open and preprocess the image at the given path.
2. Apply the *dog_detect* function. If True, go to step 4. Else, go to step 3.
3. Apply the $face\_detect$ function. If True, go to step 4. Else, terminate.
4. Feed preprocessed image to dog breed classifier.
5. Return predicted dog breed.

3.3 Refinement

Figure 7 presents the train and validation cross-entropy loss per epoch for the final scratch and transfer learning CNN models.

Scratch model

As already mentioned, the final architecture of the derived from scratch model was developed in the iterative process. Initial model consisted of 3 Convolutional layers and 2 Linear layers. The model was too shallow and validation loss was spiking after the 3rd epoch. Adding 1 hidden linear layer has improved models ability to learn, but the drop between train and validation loss was still very significant after 5-6th epoch. Further adding two more convolutional layers to increase granularity has allowed model to show better performance. The validation loss started to plateau as in Figure 7 instead of spiking up after the overfitting prevention techniques, such as batch normalization for convolutional layers and dropouts for fully connected layers were added.

Transfer learning model

There is less room for improvement available for the transfer learning model. First thing that can be noticed from the Figure 7 is that further increasing amount of epochs for learning

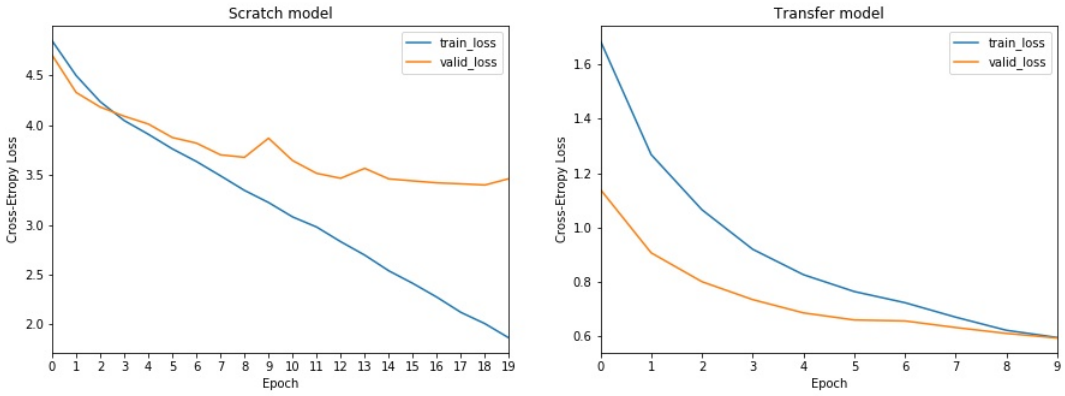

Figure 7: Train and validation Cross-Entropy losses per model

would improve the model, since there validation loss has just alligned with the training loss at the 10th epoch. Further performance could be achieved by improving the quality of input data.

4 Results

4.1 Model Evaluation and Validation

The performance of models selected for the Step 1: Object detection have been presented in section 3.2.1.

For the Step 2: Dog breed classification task, the scratch and transfer models performance has been evaluated on a dog breed test dataset, which consists of 836 test images. This is roughly 6 test images per class.

Following results have been obtained:

Model	*Train CSE*	*Valid CSE*	*Test CSE*	*Test Accuracy*
Scratch model	1.8643	3.4643	3.3742	20%
Transfer model	0.5953	0.5935	0.5961	81%

Table 2: Model metrics scores for dog breed classification

The architecture of the final Scratch model is presented in subsection 3.2.2. The structure of the transfer learning model based on ResNet18 can be referred to at (Kaiming He et al., 2015). In short, it consists of 5 convolutional layers and two fully connected layers, but the convolutional layers achieve much higher granularity compared to our model. A custom fully connected layer with 1000 input and 133 output features is added and trained to classify dog breeds.

4.2 Justification

The models used for dog and human detection achieve a very high accuracy and are compliant with requirements for their function.

The Scratch model for dog breed classification achieves 20% accuracy, which is enough to beat the benchmark of 10%. Given there are 133 classes, a random guess would have a probability of less than 1%. Nevertheless, 20% accuracy is a very low performance score and thus the model should not be used in production.

On the Contrary, the Transfer learning model achieves relatively good performance score of 81%, which is significantly above the 60% threshold. The transfer learning technique achieves such a high score compared to the scratch model due to the fact, that computer vision models, trained on a broader class set of images with higher heterogeneity, are generally better at "filtering" images, i.e. detecting relevant edges and visual patterns, that are further passed to the fully connected layers. As it has been discussed above, the model still has significant room for improvement by simply increasing the training duration. Nevertheless, 81% accuracy is good enough to use model in production.

5 Conclusion

5.1 Visualization

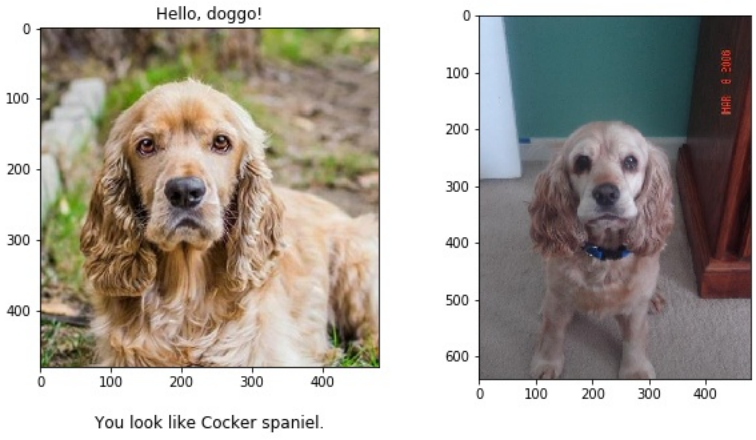

Figure 8: Output example for dog picture

The presented application returns predicted dog breed in the following format. Input image (Figure 8, left) is returned with a predicted dog breed subscript and an example of the breed picture (Figure 8, right). The example picture is selected randomly from the train dataset.

Testing the model on human face input returns, in my opinion, relatively accurate predictions as well (Figure 9).

Figure 9: Output example for human picture

However, randomly selecting one of the pictures from training data as an example of dog breed has its shortcomings, as it can be seen in Figure 10. A suggested improvement would be to pre-select representative images for each class.

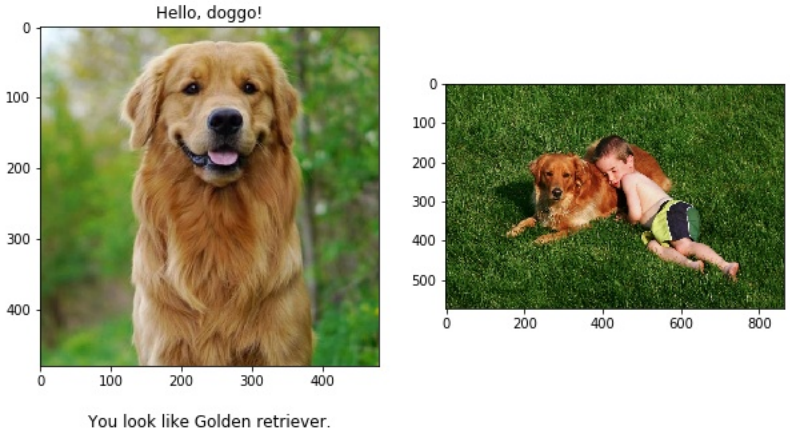

Figure 10: Example of UX shortcomings

5.2 Reflection

This project was a good opportunity for learning how to build a machine learning pipeline with non-linear logic by combining together different models into a single workflow. One

of the important takeaways is that for some tasks, like human face detection, a simple CV model can be sufficient and thus it was not needed to "over-engineer" the app and include complex CNN models for the task.

The part that I have enjoyed the most was building a CNN model from scratch and trying to improve its performance by iteratively making changes in model architecture and being able to see how different techniques affect model performance.

Another important insight for me was learning the best practices for transfer learning. What approach should be selected based on the quality and size of training data? Which baseline model to choose as a core? This has clearly inspired my further interest in this field.

5.3 Improvement

5.3.1 Model performance

Having 81% accuracy is OK but not optimal for such tasks as dog breed classification. Thus following steps can be undertaken to further improve the model performance:

- Increase number of epochs
- Try different baseline models for transfer learning
- Improve training data quality: remove images containing humans or several dogs of different breeds in one picture
- Increase training dataset. 8000 images is a relatively small dataset for deep learning approach. Model performance can be further improved by enriching train dataset with new dog images as well as applying further data augmentation techniques.

5.3.2 User experience

It should be clear that the current output format has been used only for fast prototyping in order to showcase the model results. Ideally, this ML pipeline could serve as a core for a mobile or flask web app, where user uploads a picture and receives a predicted dog breed with a nice breed example picture in return.