Capstone Project

Data Scientist Nanodegree program

April 21st, 2020

I. Project definition

Project Overview

This project aims to create a core algorithm to classify dog images according to their breed, that could be used as part of mobile or web app. The application should provide a fun user experience as it will accept not only dog images but any user-supplied image as input. If a dog is detected, the algorithm should output a prediction of the dog's breed. If a human is detected, the output should be the most resembling dog breed !

Problem Statement

The classifier used is Convolutional Neural Network (CNN), which is known as state-of-the-art of image classification. The following steps are involved :

- Pre-process images
- Build helper functions such as human detector and dog detector
- Experiment with building CNN classifiers from scratch
- Train the classifier using transfer learning
- Predict and analyze the results

The main library used to build CNN architecture throughout this project is Keras.

Metrics

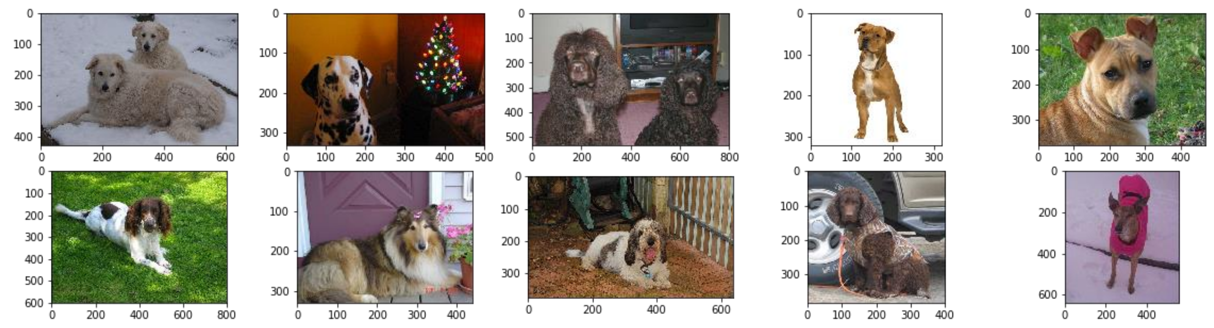
Dog breed classifier is a multi-class classification problem, thus the most relevant evaluation metrics would be accuracy. The accuracy is defined as number of correct predictions over total number of predictions made. It is important to note that the model will be evaluated on dog images only, although it will be used to predict the most resembling dog breed for human images.

II. Analysis

Data Exploration

The main dataset contains 8,351 dog images in total, which is quite small. Those images are pre-labeled with 133 dog breed names. The dataset is divided into train, evaluation and test set with proportion of 80%, 10% and 10% respectively.

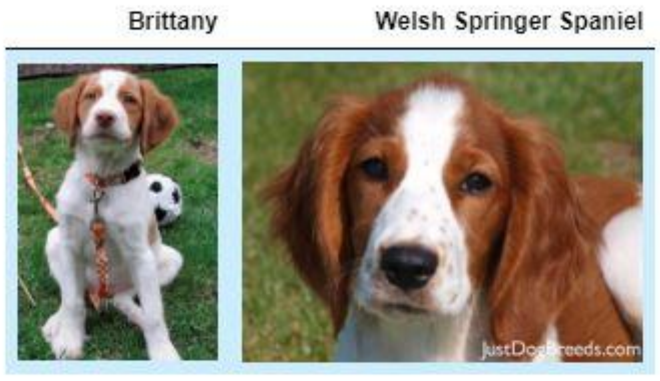
A few examples of dog images are displayed here :



It is easy to spot the following :

- Image size is not homogenous, thus image pre-processing will be compulsory.
- There can be more than one dog in an image, hence the learning task of classification model will be more complex.
- Image are provided with white background or with colored and diverse background, where various objects can be found.

The task of classification is exceptionally challenging since there is a minimal inter-class between some dog-breed pairs that even a human would have difficulty to distinguish.



At the same time, the algorithm will have to find out how to classify different shade dogs as same breed due to the high intra-class variation.

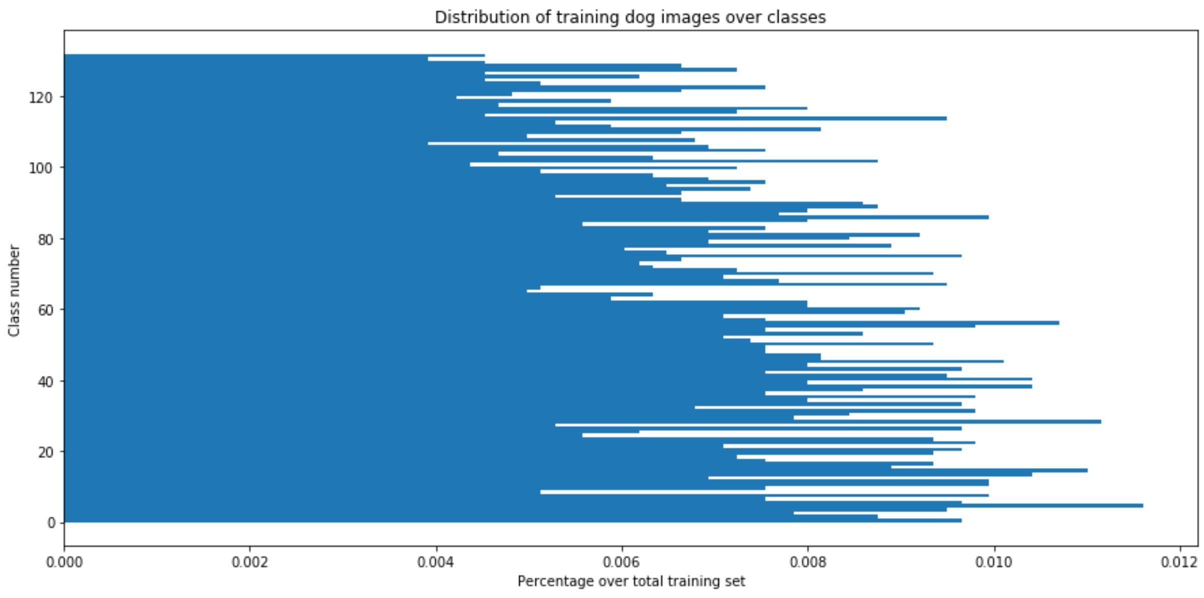
Yellow Labrador – Chocolate Labrador – Black Labrador

Besides, a dataset of 13,233 human images are also provided. It will be useful to test human detector function as well as the final algorithm's output.

Data Visualization

The below plot shows the distribution of training images per classes. It can be seen that there is a imbalance between categories. The class with highest frequency is roughly 1.2% of the training set size, while the one with lowest frequency is about 3 times lower (0.4% of the training set size). This imbalance issue may lead to overfitting where the model may fit more to dog breeds that are frequently seen.

However, the imbalance is not very serious as if there were some classes with very few images. Indeed, in terms of absolute quantity, the number of images per class ranges from approximatively 26 to 80. This will not prevent the model from learning and output predictions at some level of accuracy.



III. Methodology

Data Pre-processing

CNNs built by Keras require 4D input tensors with shape (nb_samples, rows, columns, channels) where nb_samples corresponds to the total number of images, and rows, columns and channels correspond to the number of rows, columns, and channels for each image, respectively.

As seen from above, input images need to be pre-processed. The image pre-processing function takes colored image file path as input, loads the image, resizes it to a square image of 224x224 pixels, converts it to an array and finally outputs a 4D tensor suitable to Keras model. In order to speed up training process, every images are rescaled : every pixels are divided by 255 before being fed to the model.

Helper functions

In order to detect whether input images contain a human or a dog image, two helper functions have been built: the first one is for human face detection and the second one is for dog detection.

Human detector uses OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. After the pre-trained model is downloaded, human detector function accepts an image as input and convert it to grayscale. It outputs a boolean value, according to whether a face is detected in input image.

The face detector function has been tested on 100 human images and 100 dog images with 100% and 89% accuracy respectively. It is worth noting that among 11 dog images where human face is detected by the algorithm, there is one with a clear human face and two with human body presence. Even though it doesn't performe perfectly on dog images, its accuracy is still acceptable.

Similarly, dog detector uses Resnet50 model, which has been pre-trained on ImageNet, a +10 million image dataset from 1000 categories. While the Resnet50 model predicts one among available categories on ImageNet, our dog detector has an additional functionality : it checks whether the predicted class corresponds to a dog category and return True if it is the case and False otherwise.

In terms of accuracy, it performs perfectly on both human and dog images. No human image is detected as having a dog, and 100% of dog images have a detected dog.

Implementation

The implementation process can be split into two main steps :

1. Build CNN models from scratch
2. Use transfer learning to have better performance

The first stage aims to experiment with different Keras basic models. This step should provide a good reflection on how to tune pre-trained model in next stage.

When building CNN from scratch, the main idea is increase the number of channels throughout layers and at the same time reduce the layer length and width.

In order to be able to test performance quickly, I construct small architectures with three to five 2D convolutional layers. Three different models are benchmarked :

- The first one uses 3 convolutional layers with 3 padding layers, followed by 2 fully-connected layers ;
- The second one uses 5 convolutional layers with 5 padding layers, followed by 2 fully-connected layers.
- The third one uses 5 convolutional layers with 5 padding layers, followed by 3 fully-connected layers.

In each case, dense layers are equipped with dropout to prevent overfitting. Final output layer is equipped with a softmax function.

For the first 2D convolutional layer, I choose 16 as the number of channels as this is a reasonable amount that will be increasing by a factor of 2 throughout layers.

Same padding is chosen over valid padding in order to preserve input value at the pixels near image edges.

Relu activation is chosen on 2D convolutional layers due to its reputation on performance in training CNN model.

3x3 kernel size with stride 1 is used throughout the CNN in order to capture the input at every pixel. This is motivated by high intra-class variation of some breeds, thus any value that features the dog patterns must be extracted. This kernel size also helps to reduce number of parameters, compared to 5x5 or 7x7 kernel size. For example: for the second model, network with 3x3 kernel size has 2,616,157 parameters to train, while 5x5 CNN has 3,051,101 and 7x7 CNN has 3,703,517 parameters.

For pooling layers, overlapping max pooling with size 3 and stride 2 is chosen in order to extract the dominant feature value in near neighborhoods. This choice is motivated by the fact that there are dog breed pairs with minimal inter-class variation, thus any feature value that differs must be extracted.

In the second stage, transfer learning is used to improve accuracy and reduce training time. The pre-trained VGG-16 is imported and its last convolutional output is fed to our model. A global average pooling layer and a fully connected layer with 133 nodes are added to the model.

Refinement

As experimented above Implementation section, using transfer learning helps to improve model accuracy. However, in order to have even higher performance and allow the algorithm to be productive, fine-tune the model is necessary.

Pre-computed features of Resnet50 model is imported and several additional layers are added to the model : a Global Max Pooling layer, followed by two fully-connected layers with dropout. Max Pooling allows to detect noise so I think that it should perform well here where there are some minimal variations inter-class. Dropout is added to two dense layers in order to prevent overfitting.

Notice that Global Max Pooling layer's output have 2048 neurons. They are fed into first dense layer equipped by a ReLU activation function in order to filter out the top 500 relevant outputs. Those outputs are then fed into second dense layer that "votes", with softmax activation function, the label with highest probability.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_max_pooling2d_1 (Glob (None, 2048)              0
_________________________________________________________________
dropout_12 (Dropout)         (None, 2048)              0
_________________________________________________________________
dense_13 (Dense)             (None, 500)               1024500
_________________________________________________________________
dropout_13 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_14 (Dense)             (None, 133)               66633
=================================================================
Total params: 1,091,133
Trainable params: 1,091,133
Non-trainable params: 0
_________________________________________________________________
```

Same as the previous experimented models, I keep using categorical cross-entropy loss and RMSProp as optimizer since they worked quite well.

The model with highest accuracy is then used to create final algorithm that accepts a file path as input and first determines whether there is a dog, a human or neither. If a dog or a human is detected, it should returns the predicted dog breed or the most resembling dog breed respectively. If neither is detected, an error message will be displayed. This is where the helper functions built earlier (human face detector and dog detector) come into action.

IV. Results

Model Evaluation and Validation

Each model is trained during 20 epochs. A validation set is used to evaluate the model and checks whether validation loss decreases. If this is the case, the model weights are saved with a checkpointer and will be loaded later for testing.

Three models built from scratch reported 7.65%, 14.35% and 17.10% of accuracy respectively. From this benchmarking, we can conclude that more convolutional layers and more fully-connected layers help to improve model accuracy. When deeper layers are added, the model is able to capture high level features, after edge and shape of images have been detected by shallower layers.

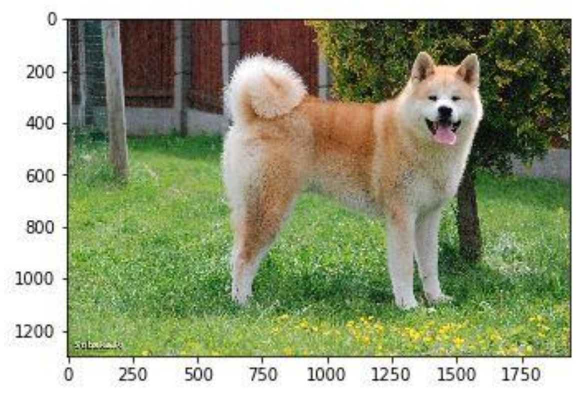
Transfer learning using VGG-16 network reported 42.34% of accuracy on test set. This result is in concordance with the above analysis, since VGG-16 has upto 16 layers and was especially pre-trained on over 10 million images.

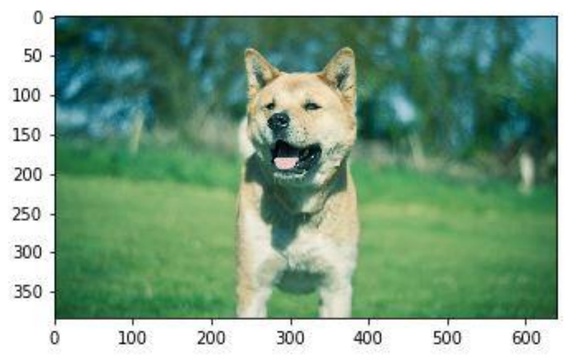
The final model which is a combination of pre-trained Resnet50 model and customized architecture achieved a good accuracy : 80.14%. It is then used to build the final algorithm.

To verify the robustness of this algorithm, several checks were performed with input images from my personal computer. The model gives correct answer when two cat images are fed in and predicts exactly 3/4 dog breeds. Note that it predicts correctly Brittany which is among challenging breeds. As of human images, the results is quite convincing, however, this is the fun part of the application and no accuracy metrics are applied.

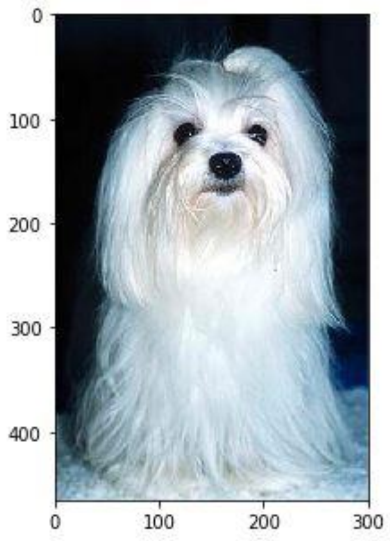
Some examples of model's output :

```
Hi puppy! I predict that you are a ... Akita.
```





```
Hi there! You look like a ... Maltese.
```



Justification

The experiment with models from scratch then transfer learning from the above part shows that pre-trained model is very good to improve model accuracy but to reach acceptable and productive performance, it needs to be fine-tuned, according to the specific problem here. As mentioned earlier, dog breed classification is quite challenging seeing the dataset size and some minimal variations between classes as well as high variations intra-class.

Use a pre-trained model that had learned from big dataset allows the algorithm to quickly acquire the bottleneck features and several added layers help to customize it for dog breed classification task. Notice that adding just one dense layer might not be enough to get acceptable performance (based on what has been experimented with pre-trained VGG-16 model). This observation may open a good way for Improvement reflection in next part.

V. Conclusion

Reflection

To summarize, this project has been passing the following steps :

- Problem overview and statement
- Import dataset, explore data and pre-process input images
- Build helper functions for the final algorithm
- Build different CNN architectures and benchmark the classifiers
- Choose final model and integrate it to final algorithm
- Evaluate and verify the algorithm robustness

The most challenging and interesting part was building models from scratch. Seeing the specific classification task's difficulties, it was not easy to create something from 0 that gives a minimum accuracy, even though the requirement was 1%. This part requires me to look for documents and research papers but it is also where I learned the most. There is no better way to understand in deep

CNNs than building and experiment with my own models. Finally the results were much better than I expected.

Improvement

To improve the model performance, several pathways are worth considering :

- Use more fully-connected layers: From Building model from scratch part, more dense layers are likely to improve model accuracy.
- Perform data collection or data augmentation: The dataset is quite small with imbalanced classes. It might be a good idea to collect additional images and/or create transformed images for classes with low frequency.
- In order to align with fun user experience objective of the application, it will be great to implement a cat detector then train the model so that it will output the most resembling dog !

References:

- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ["ImageNet Classification with Deep Convolutional Neural Networks."](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) NIPS 2012

- Quora: [How does pooling control overfitting in CNN?](https://www.quora.com/How-does-pooling-control-overfitting-in-CNN)

- Quora: [What is the motivation for pooling in convolutional neural networks (CNN)?](https://www.quora.com/What-is-the-motivation-for-pooling-in-convolutional-neural-networks-CNN)