Capstone Project
Data Scientist Nanodegree

Hiral
December 9th, 2021

Problem Introduction

Given an image of a dog, my algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed. The goal is to classify images of dogs according to their breed.

Strategy to solve the problem

This project uses Convolutional Neural Networks (CNN, or ConvNet)! It is a class of artificial neural network, most commonly applied to analyze visual imagery. They are used for image classification and recognition because of its high accuracy. It is currently the best algorithm for the automated processing of images.

The following steps will be involved:

- The images will be pre-processed.

- We will be building helper functions for detecting human and dog face.

- We will be experimenting with building of CNNs from scratch.

- We will be using pre-trained ResNet-50 CNN.

- We will be predicting and analyzing the results.

We will be using Keras (neural network library) for building CNN architecture. The algorithm that detects humans in an image is different from the CNN that infers dog breed. The model will be evaluated on dog images only. We will also use it to predict the most resembling dog breed for human images.

Metrics

We have a multi-class classification problem, that's why the most relevant evaluation metrics would be **Accuracy**. It calculates how often predictions equal labels.

Exploratory data analysis (EDA)

The dataset contains following information:

1. There are 133 total dog categories.
2. There are 8351 total dog images.
3. From total dog images, 6680 are used as training dog images.
4. From total dog images, 835 are used as validation dog images.
5. From total dog images, 836 are used as test dog images.

Random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imbalanced, a random guess will

provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

The task of assigning breed to dogs from images is considered exceptionally challenging. For example, by looking at following images, even a human would have great difficulty in distinguishing between a Brittany and a Welsh Springer Spaniel.

Another example, Labradors come in yellow, chocolate, and black. Algorithm will have to conquer this high intra-class variation to determine how to classify all of these different shades as the same breed.

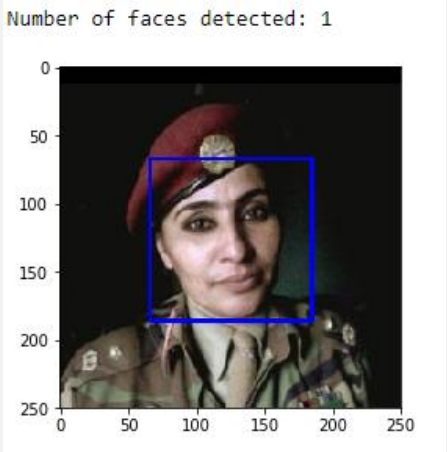
Pre-process the Data

Keras CNNs require a 4D array as input, with shape (nb_samples, rows, columns, channels), where nb_samples corresponds to the total number of images (or samples) and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

Our function takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. It first loads the image and resizes it to a square image that is 224 * 224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since we are working with color images, each image has three channels. Likewise, since we are processing a single image (or sample), the returned tensor will always have shape (1, 224, 224, 3). We rescale the images by dividing every pixel in every image by 255 before being fed to the model.

Functions

Human Face Detector: OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. Returns True if a human face is detected in an image and False otherwise.



***Dog Detector**:* pre-trained ResNet-50 CNN model to detect dogs in images. Returns True if a dog is detected in an image (and False if not).

Modelling

Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. We will create a CNN that classifies dog breeds. We will be doing in two different ways:

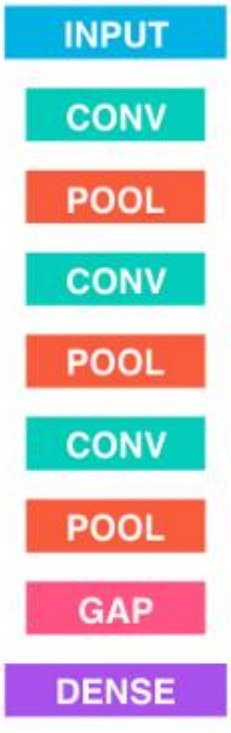
1. From Scratch.

2. Using Transfer Learning (ResNet-50 CNN).

Practice is far ahead of the theory in deep learning. Experiment with many different architectures, and trust your intuition.

Create a CNN (from Scratch)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 223, 223, 16)      208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 32)      2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 54, 54, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 27, 27, 64)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               8645
=================================================================
Total params: 19,189.0
Trainable params: 19,189.0
Non-trainable params: 0.0
_________________________________________________________________
```



We can experiment with different Keras basic models. The main idea is to increase/decrease the number of channels throughout layers and at the same time increase/reduce the layer length and width.

After experimenting with some architectures, our model architecture contains:

- 3 convolutional layers with 50, 75, and 125 feature respectively maps with size 3×3 and with a rectifier function.
- After each convolutional layer, there is a pooling layer that takes the maximum value called MaxPooling2D, it is configured as a 2×2 pool size.
- After second and third pooling layer, there is a dropout layer for regularization. It is set to randomly exclude 25% of the neurons in the layer to avoid overfitting.
- After that there is the flattened layer that converts the 2D matrix data into a vector called Flatten. It allows the output to be fully processed by a standard fully connected layer.
- After above, there is the fully connected layer with 1024 neurons and rectifier activation function.
- Finally, the output layer has 133 neurons for the 133 dog categories and a softmax activation function to output probability-like predictions for each class.

Create a CNN (using Transfer Learning)

I have used pre-trained model ResNet-50 model. The bottleneck features are available in Keras and can be easily extracted from there. After extracting the bottleneck features, model architecture is extended as follows:

- GlobalAveragePooling2D as pooling layer takes input.
- After that, there is the fully connected layer with 1024 neurons and rectifier activation function.
- Finally, the output layer has 133 neurons and a softmax activation function.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 133)               136325
=================================================================
Total params: 2,234,501
Trainable params: 2,234,501
Non-trainable params: 0
_________________________________________________________________
```

Hyper parameter tuning

We will be using following:

1. Categorical cross entropy*:* used as a loss function for multi-class classification model where there are two or more output labels.
2. Accuracy *:* used as metrics.

3. RMSprop: root mean square propagation, used as an optimization algorithm/method designed for artificial neural network training.

Both the models have been compiled with above parameters. Both the models have been trained with 20 epochs and 20 batch size.

```
Resnet50_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

```
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Resnet50.hdf5', verbose=1, save_best_only=True)

Resnet50_model.fit(train_Resnet50, train_targets, validation_data=(valid_Resnet50, valid_targets),
          epochs=20, batch_size=20, callbacks=[checkpointer], verbose=1)
```

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 (Keras function *ModelCheckpoint)* and will be loaded later for testing.

Since the pre-trained Resnet50 model and customized architecture (using Transfer Learning) has highest accuracy, it has been used to write a function that takes an image path as input and returns the dog breed (`Affenpinscher`, `Afghan_hound`, etc) that is predicted by model.

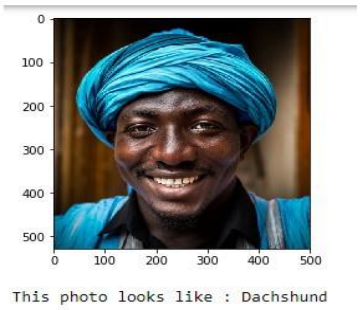
Results

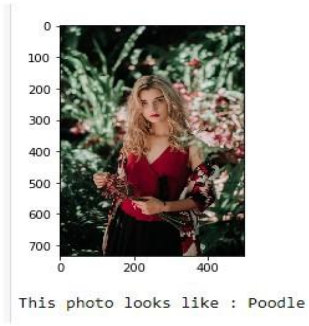
The CNN model built from scratch has given a test accuracy: 9.5%

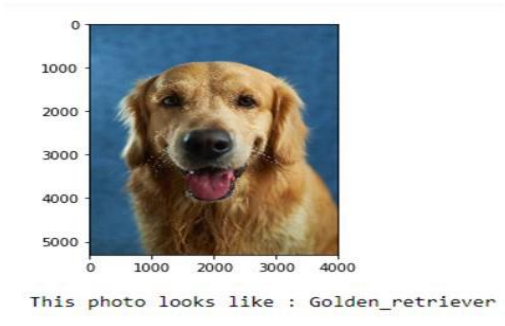
The final CNN model which is a combination of pre-trained ResNet-50 CNN and additional customized architecture has given a good accuracy: 79%.

Test has been performed with 6 random different images from internet. A Algorithm has been written that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

- If a dog is detected in the image, return the predicted breed.
- If a human is detected in the image, return the resembling dog breed.
- If neither is detected in the image, provide output that indicates an error.







Conclusion/Reflection

The model from scratch and then model from transfer learning shows that pre-trained ResNet-50 CNN model is very better for improving model accuracy but to reach acceptable and productive performance, it

also needs to be fine-tuned. There is no better way to understand CNNs than building and experimenting.

ResNet-50 is 50 layers deep and is trained on a million images of 1000 categories from the ImageNet database that makes it better for image recognition and can be easily customize for dog breed classification task.

The findings here are observational, not the result of a formal study.

To see more about this analysis, see the link to my GitHub available here.

Improvements

For improving the CNN model performance, following can be considered:

1. If model building from scratch is to be used then, more fully-connected layers (dense layers) can be used to see if accuracy would be improved.
2. The dataset is quite small with imbalanced classes. Additional images can be collected to increase the training and validation data.
3. Data augmentation (increase the amount of data by adding slightly modified copies of already existing data) can also be performed.