Table of Content

Project Overview

Problem Statement

Metrics

Data Exploration

Visualization

Benchmark

Data Preprocessing Steps

Implementation

Refinement

Results

Justification

Reflection

Future Improvement

Appendix

Project Overview

For the Capstone project, I attempted to tackle Dog Breed Classifier. This project requires implementing Convolutional Neural Networks (CNNs) architecture. In this project, I will build a pipeline to process real-world, user-supplied images. We will utilize a few algorithms to identify a given dog image and predict the canine's breed. However, if supplied an image of a human, the code will identify the resembling dog breed.

Problem Statement

Recognizing an image seems to be an easy task for humans. However, it is a challenging task for computers. Certain factors, such as colors, lighting, shades, angles, can confuse computers to identify an image. The improvement in Computer Vision transfer learning technique using sophisticated neural networks has allowed computers to achieve human-level performance in identifying and classifying images.

Metrics

In this project, I use two different metrics as follows:

1. `time elapsed`: this metric is used to compare the performance of computer vision algorithms in identifying and classifying a given image.
2. `accuracy`: this metric is used to measure the performance of our deep learning model against the test dataset.

Data Exploration

Given a dataset of dog images, I populate a few variables through the use of the load_files function from the scikit-learn library. I divided those images into three dataset, ie. training, validation, and test dataset. I also imported the label so we can use supervised machine learning technique to train our deep learning model.

For dog images:

- 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.

For human images:

- There are 13233 total human images.

Visualization

For data visualization, I use two libraries as follows:

1. OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.
2. Matplotlib to display an image picture on Jupyter Notebook along with the rectangle box to classify the object as identified by OpenCV algorithm.

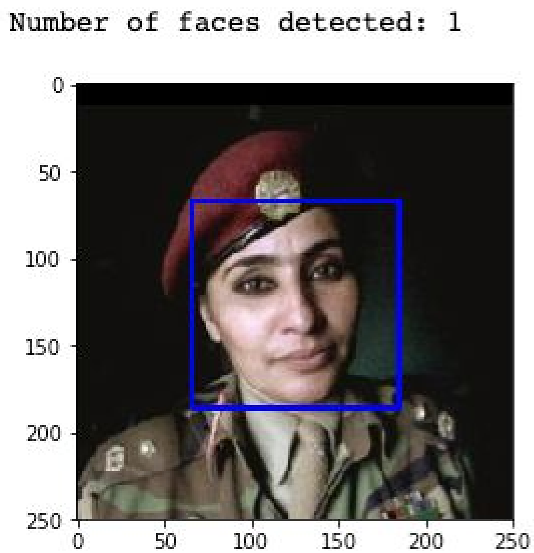


Fig 1: human face detection with OpenCV

Benchmark

OpenCV's Haar Cascade algorithm can easily identify human faces, but has a hard time in identifying dog faces as shown in the following benchmark:

- human files identified : 100% within 2.76 seconds.
- dog files identified : 11% within 14.64 seconds.

Data Preprocessing Steps

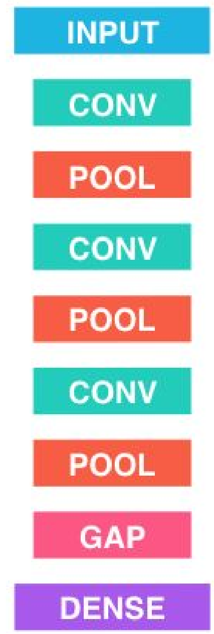
To train the dog images using Keras CNN model, I need to convert the dog images with 224x224 pixels into a 4D tensor with the following tuple, ie. (number of samples, row pixels, column pixels, channels).

The `path_to_tensor` function converts the RGB image as a PIL type with 224x224 dimension. This image is then converted into 224x224x3 (where 3 is the number of channels for RGB images). Finally, it's converted to 1x224x224x3 tensor.

Implementation

Using Keras Sequential model, I built my CNN model from scratch with the following architecture:

```
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
_________________________________________________________________
```



Then, I compile the model with the following configurations:

- optimizer='rmsprop',
- loss='categorical_crossentropy',
- metrics=['accuracy']

Finally, train the model with the following settings:

- epochs=20,
- batch_size=20

Refinement

To further improve the accuracy, I apply transfer learning technique using VGG16 model. VGG16 (also called OxfordNet) is a CNN architecture named after the Visual Geometry Group from Oxford. This model was used to win the ILSVR (ImageNet) competition in 2014. The pretrained VGG16 `npz` file loads a set of weights pre-trained on ImageNet.

To fit into our dog images dataset, I need to replace the output layer and replace it with Keras Dense layer with 133 nodes (ie. number of dog labels) and softmax activation to compute multi-class output nodes.

Results

The accuracy metric on the test dataset jumped from 4.4258% (using my own CNN architecture) to 38.7560% (using VGG transfer learning model). The performance is further improved to 83.1340% accuracy score using ResNet50 transfer learning model.

Justification

Resnets are a kind of CNNs called Residual Networks. They are very deep compared to VGG architecture. Resnet 50 refers to a 50 layers, while VGG16 has 16 layers. In general, deeper and wider architecture can lead to better performance than shallow and small networks. However, bigger networks require bigger computing powers and time to train.

The following models are benchmarked:

Network	Layers	Top-1 error	Top-5 error	Speed (ms)	Citation
AlexNet	8	42.90	19.80	14.56	[1]
Inception-V1	22	-	10.07	39.14	[2]
VGG-16	16	27.00	8.80	128.62	[3]
VGG-19	19	27.30	9.00	147.32	[3]
ResNet-18	18	30.43	10.76	31.54	[4]
ResNet-34	34	26.73	8.74	51.59	[4]
ResNet-50	50	24.01	7.02	103.58	[4]
ResNet-101	101	22.44	6.21	156.44	[4]
ResNet-152	152	22.16	6.16	217.91	[4]
ResNet-200	200	21.66	5.79	296.51	[5]

Fig 2: CNN models benchmark

Reflection

This Capstone project is very educative and fun to do. It introduces challenges and algorithms to identify human faces and dog faces. The project starts with a simple application of OpenCV Haar Cascade algorithm to identify human faces. The project then tried to tackle a harder problem to identify dog faces with a new dataset. I learned that OpenCV library showed poor performance to identify dog images. Therefore, I need to build my own small CNN network. I learned how to preprocess the image data into a 4D tensor, build a custom Keras CNN model, compile the model and train the new dataset. However, a small network is not sufficient to tackle this challenge. Therefore, I applied transfer learning based on pretrained models that have been used to train ImageNet (ie. a collection of large dataset). I learned two different modern CNN models, ie. VGG16 and ResNet50, and customize them to train our dog dataset. I learned that ResNet50 is by far outperformed VGG16 transfer learning, my own CNN architecture, and OpenCV human face recognizer.

Future Improvement

For this project, we tried to identify human face or dog face from an image. Using pretrained models and customize them with our dog images dataset helps to improve the accuracy of our dog breed classifier. Although our classifier performed relatively well, it's only good at identifying narrow task, ie. classifying dog faces. Since our users might be interested in identifying and classifying human faces and dog faces, I will train a model based on a larger dataset that includes both human faces and dog faces.

Appendix

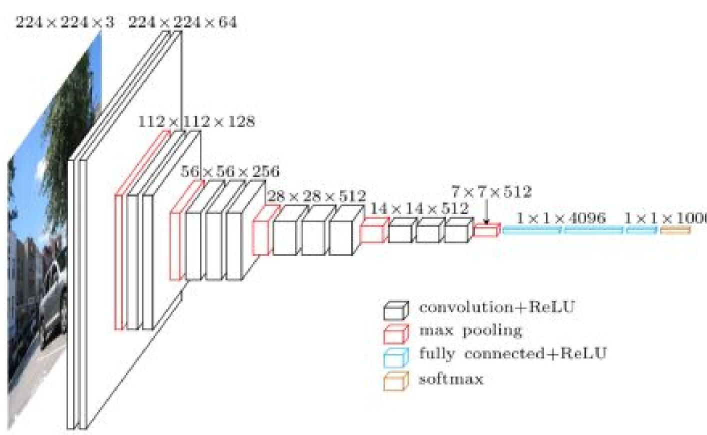


Fig 3: VGG16 Architecture

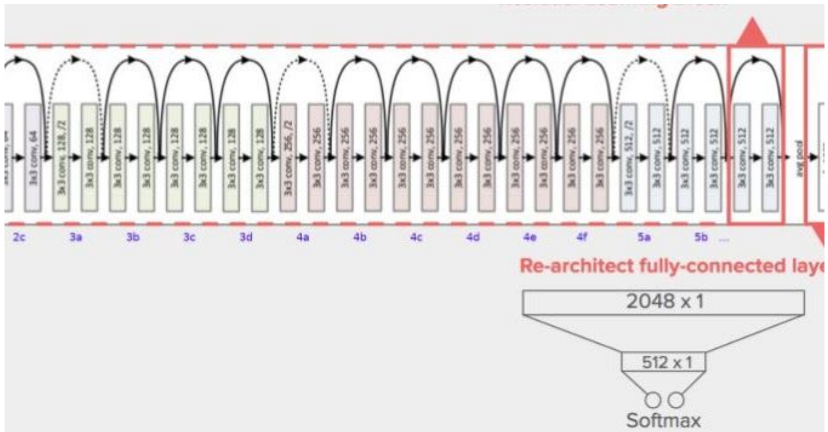


Fig 4: Resnet50 Architecture