Project definition

Project Overview

This project is similar to Kaggle challenge to identify dog breed [https://www.kaggle.com/c/dog-breed-identification] using the Stanford Dogs dataset [http://vision.stanford.edu/aditya86/ImageNetDogs/]. This project goes further, also identifying human faces using OpenCV Haar Feature-based Cascade Classifiers pre-trained. The dataset with human faces is the Labeled Faces in the wild (lfw)[http://vis-www.cs.umass.edu/lfw/] from University of Massachusets.

To accomplish these tasks we apply multiclass classification using convolutional neural network and transfer learning techniques using the ResNet50 network pre-trained on the imagenet dataset.

Problem statement

In this project, there are two steps:

- Identify a human face, a dog or none in the image;
- If human face or dog, identify the dog breed more related;

Metrics

The metrics used to monitor this problem is the accuracy, how many predictions the model got right. Here we have the formula to binary classification accuracy, using true positive (TP), true negatives (TN), false positives (FP) and false negatives (FN):

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Analysis

Data exploration

The Stanford Dog dataset is a collection of dog images and the version available from Udacity [https://github.com/udacity/dog-project/] is a well-organized dataset ready to go. There are training, testing and validation folders. Within each folder is a subfolder for each dog breed with images, with the corresponding dog breed's file name and a number. This dataset contains 8351 RGB images with different sizes divided into 133 races. The train-validation-test division contains 6680, 835 and 836 dog images, respectively. The folder structure described above is important because it gives us the annotations about the dog's breed and can be used with the Keras → ImageDataGenerator and Fastai → ImageDataLoaders python libraries. The Stanford Dog dataset, listed at the beginning, is larger with 20,580 images and 120 breeds, but was not used in this project. The lfw dataset contains 13,233 images of 5749 people with a normal size of 150x150 pixels. Each image is identified with the name of the person portrayed.

Methodology

Data preprocessing

The data preprocessing to lfw dataset was convert from RGB to gray scale.

To dog dataset is necessary to convert from RGB format to BGR, apply the normalization between 0 and 1 and resize the image to 256x256 pixels.

Implementation

The project consist of Jupyter notebook and it was executed mainly in the Colab [colab.research.google.com] platform using python 3.8. A detailed list with all python libraries to run this project is available in requirements file [https://github.com/edbroni/dog_breed/requirements.txt] and the most important are:

-Keras;

-Tensorflow ;

-OpenCV.

To the first task, identify human faces, the implementation is above and is the application of Haas Cascade classifier available in OpenCV, using the pre-trained model to identify frontal faces:

```
# extract pre-trained face detector
face_cascade =
cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

This function, using the CascadeClassifier shows 97% of accuracy in lfw dataset and 88% of accuracy in dog dataset.

The dog detector apply the ResNet50 convolutional network pre trained with imagenet dataset. This is a multiclass classificator with 1000 diferent class. The dogs classes are between classes 151 e 268. There is two functions envolved here, the class prediction label and the detector itself (return true or false according class number).

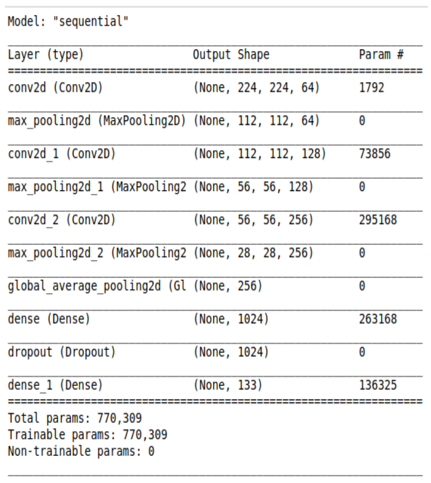
```
def ResNet50_predict_labels(img_path):

    # returns prediction vector for image located at img_path
    img = preprocess_input(path_to_tensor(img_path))
    return np.argmax(ResNet50_model.predict(img))


def dog_detector(img_path):

    prediction = ResNet50_predict_labels(img_path)
    return ((prediction <= 268) & (prediction >= 151))
```

The accuracy of this detector is 99% in lfw dataset and 100% in dog dataset.

To classify the dog breed two approaches was suggested: train a CNN from scratch and apply the transfer learning.

- Training from scracth:

CNN implemented (fig. 1)



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 112, 112, 128)     73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 256)       295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
global_average_pooling2d (Gl (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              263168
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               136325
=================================================================
Total params: 770,309
Trainable params: 770,309
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Model trained from scratch.

Optmizer – RMSProp

Loss function – categorical cross entropy

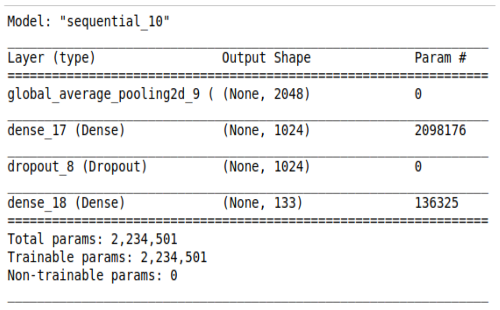
Metric – accuracy

Trained using data augmentation (rotation, horizontal flip, reduction in height and width) for 20 epochs and batch size of 20 images. This model give an accuracy of 18.6%. This is very simple network and a better approach is the transfer learning method.

- Transfer learning

Start with pre trained ResNet50 without the fully connected layers, this CNN add the residual block between layers to better propagate the gradient, the original paper describing the ResNet is Deep Residual Learning for Image recognition [http://arxiv.org/abs/1512.03385v1].

It was created this network bellow to use after the ResNet50 (using Keras) in function create_CNN_predictor:



```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_9 ( (None, 2048)              0
_________________________________________________________________
dense_17 (Dense)             (None, 1024)              2098176
_________________________________________________________________
dropout_8 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_18 (Dense)             (None, 133)               136325
=================================================================
Total params: 2,234,501
Trainable params: 2,234,501
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Layers to substitue the last layers in ResNet50.

Optmizer – RMSProp

Loss function – categorical cross entropy

Metric – accuracy

Trained using data augmentation (rotation, horizontal flip, reduction in height and width) for 6 epochs and batch size of 20 images. This model give 79.9% of accuracy.

To train this second model or to predict using this second model was necessary to extract the bottleneck features, which is pass the images through ResNet50 without the last three layers. Below is a drawing showing this setup.

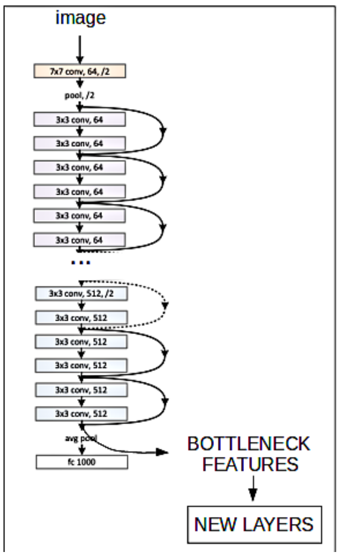


Figure 3: ResNet50 archtecture and the residual loops. Edited from Kaimin et all, 2015.

The function return_dog_breed is where those steps of extract the bottleneck features and the class prediction take place. Your results are the predicted breed of dog and a array of predicted probabilities to all classes.

To deploy the model on line using Heroku, the tensorflow and Keras libraries became an issue due the size and performance. After some comparison with other implementations I found a possible path using the fastai library. With that in mind, a new model was trained, using the same architecture to apply the transfer learning using the ResNet50 pre trained. This model gave the accuracy of 95%.

Refinement

The function identify_and_classify implement the solution to identify the subject (human, dog or neither) and classify. Here we have a difference between the solution in the notebook, using Keras and Tensorflow and the version deployed in Heroku. To be efficient the dog_detector function wasn’t deployed, it was replaced by a confidence value. The comparison bellow shows the pseudo code to better comprehension. With this approach the execution is faster than run the prediction two times when the input is a dog image.

Implementation in jupyter notebook	Implementation in Heroku app
Input image	Input image
If is a human face:	If is a human face:
Identify dog breed	Identify the dog breed
Display dog breed	Display dog breed
Else if is a dog face:	Else if dog breed probability > 45%
Identify the dog breed	Display dog breed
Display dog breed	Else
Else	Display error
Display error	

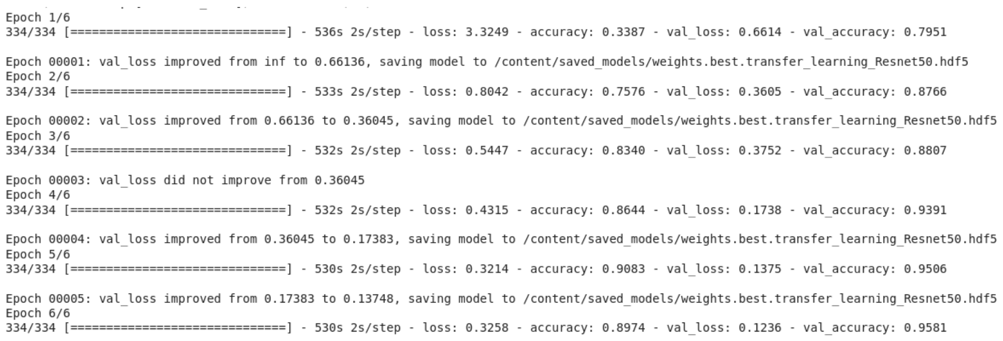
Results

Model evaluation and validation

Summarizing the results:

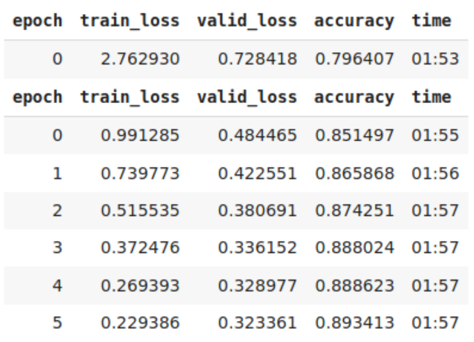
Function	Lfw dataset accuracy	Dog dataset accuracy
face_detector	97%	88%
dog_detector	99%	100%

The figures bellow show the results to two CNN, using Keras and TensorFlow (fig 4) and using Fastai (fig 5). These figures show us the better performance of tensorflow in the validation dataset, but the results with test dataset shows better performance of .



```
Epoch 1/6
334/334 [==============================] - 536s 2s/step - loss: 3.3249 - accuracy: 0.3387 - val_loss: 0.6614 - val_accuracy: 0.7951

Epoch 00001: val_loss improved from inf to 0.66136, saving model to /content/saved_models/weights.best.transfer_learning_Resnet50.hdf5
Epoch 2/6
334/334 [==============================] - 533s 2s/step - loss: 0.8042 - accuracy: 0.7576 - val_loss: 0.3605 - val_accuracy: 0.8766

Epoch 00002: val_loss improved from 0.66136 to 0.36045, saving model to /content/saved_models/weights.best.transfer_learning_Resnet50.hdf5
Epoch 3/6
334/334 [==============================] - 532s 2s/step - loss: 0.5447 - accuracy: 0.8340 - val_loss: 0.3752 - val_accuracy: 0.8807

Epoch 00003: val_loss did not improve from 0.36045
Epoch 4/6
334/334 [==============================] - 532s 2s/step - loss: 0.4315 - accuracy: 0.8644 - val_loss: 0.1738 - val_accuracy: 0.9391

Epoch 00004: val_loss improved from 0.36045 to 0.17383, saving model to /content/saved_models/weights.best.transfer_learning_Resnet50.hdf5
Epoch 5/6
334/334 [==============================] - 530s 2s/step - loss: 0.3214 - accuracy: 0.9083 - val_loss: 0.1375 - val_accuracy: 0.9506

Epoch 00005: val_loss improved from 0.17383 to 0.13748, saving model to /content/saved_models/weights.best.transfer_learning_Resnet50.hdf5
Epoch 6/6
334/334 [==============================] - 530s 2s/step - loss: 0.3258 - accuracy: 0.8974 - val_loss: 0.1236 - val_accuracy: 0.9581
```

Figure 4: Transfer learning using Keras and Tensorflow



epoch	train_loss	valid_loss	accuracy	time
0	2.762930	0.728418	0.796407	01:53

epoch	train_loss	valid_loss	accuracy	time
0	0.991285	0.484465	0.851497	01:55
1	0.739773	0.422551	0.865868	01:56
2	0.515535	0.380691	0.874251	01:57
3	0.372476	0.336152	0.888024	01:57
4	0.269393	0.328977	0.888623	01:57
5	0.229386	0.323361	0.893413	01:57

Figure 5: Transfer learning using Fastai.

Justification

The transfer learning is an option very useful to speed-up the process of training a CNN. It's take all the optimization already done in very complex models and allow the application in different classifications. This is very clear comparing the results of model made from scratch and the other with transfer learning.

Conclusion

Reflection

The proposed image classification problem was carried out, classifying human faces, dogs and dog breeds. Starting from an image, the methods implemented in this project are able to return an answer. There are differences between the jupyter notebook version and the web-deployed version due to resource constraints on the Heroku server. This project is very easy to follow on the jupyter notebook and features are no problem, but deploying to the web is different. Some restrictions had to be applied to provide a functional and accurate application.

In this project, space and performance constraints forced us to use the fastai library at the end of the project. This obligation was a little daunting at first, but in the end it proved to be very useful as the fastai library is quite simple to use. Thus, this project led me to look for another solution that I intend to use more than Keras and TensorFlow.

Improvement

I’m satisfied with the results of this project but it can be improved of course. I applied the data augmentation but other techniques could improve the performance.

The first step, detect human faces, could be improved using other model with better performance or other neural network. I found other haarcascades models but I didn’t try due lack of time.

The transfer learning technique is very useful and worked really well but the last layers added over ResNet50 could be tuned using the grid search procedure. Other improvement is the fine tuning of all weights in the neural network, training few epochs with this dataset. Other possible improvement is prunning the network, removing some connections with low weight values, this could benefit the performance of the CNN.

References

Kaiming He et. all, Deep residual learning for image recognition, 2015, https://arxiv.org/abs/1512.03385