Data Science Capstone Project

Milad Sadat Mohammadi

Convolutional Neural Networks application in Dog Identification App

This notebook introduces one of the most popular projects using convolutional neural networks. The goal is to classify images of dogs according to their breed. At the end of this project, we will develop an algorithm to accept any user-supplied image as input. Then, estimate the dog's breed if a dog is detected in the image. If a human is detected, it will provide an estimate of the dog breed that is most resembling. In this real-world setting, we will piece together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will differ from the CNN that infers dog breed. There are many points of possible failure, and no perfect algorithm exists. This notebook is broken into separate steps as follows:

- Step 0: Load Datasets
- Step 1: Detect Humans
- Step 2: Detect Dogs
- Step 3: Create a CNN to Classify Dog Breeds
- Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 6: Write the Algorithm
- Step 7: Test the Algorithm

1. Import Datasets

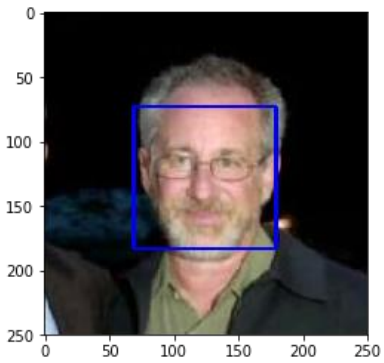
First, we import a dataset of dog images. We populate a few variables through the use of the load_files function from the scikit-learn library:

- train_files, valid_files, test_files - numpy arrays containing file paths to images

- train_targets, valid_targets, test_targets - numpy arrays containing one hot-encoded classification labels
- dog_names - list of string-valued dog breed names for translating labels

2. Detect Humans

We use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. OpenCV provides many pre-trained face detectors, stored as XML files on github. We have downloaded one of these detectors and stored it in the haarcascades directory. Before using any of the face detectors, it is standard procedure to convert the images to grayscale. The detectMultiScale function executes the classifier stored in face_cascade and takes the grayscale image as a parameter.



Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. The first two entries in the array (extracted in the above code as x and y) specify the horizontal and vertical positions of the top left corner of the bounding box. The last two entries in the array (extracted here as w and h) specify the width and height of the box.

3. Detect Dogs

In this section, we use a pre-trained ResNet-50 model to detect dogs in images. Our first line of code downloads the ResNet-50 model, along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10

million URLs, each linking to an image containing an object from one of 1000 categories. Given an image, this pre-trained ResNet-50 model returns a prediction (derived from the available categories in ImageNet) for the object that is contained in the image.

a. Pre-process the Data

When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) 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. The path_to_tensor 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. The function 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).

The paths_to_tensor function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape

(nb_samples,224,224,3).

Here, nb_samples is the number of samples, or number of images, in the supplied array of image paths. It is best to think of nb_samples as the number of 3D tensors (where each 3D tensor corresponds to a different image) in your dataset!

b. Making Predictions with ResNet-50

Getting the 4D tensor ready for ResNet-50, and for any other pre-trained model in Keras, requires some additional processing. First, the RGB image is converted to BGR by reordering the channels. All pre-trained

models have the additional normalization step that the mean pixel (expressed in RGB as [103.939,116.779,123.68] and calculated from all pixels in all images in ImageNet) must be subtracted from every pixel in each image.

Now that we have a way to format our image for supplying to ResNet-50, we are now ready to use the model to extract the predictions. This is accomplished with the predict method, which returns an array whose i-th entry is the model's predicted probability that the image belongs to the i-th ImageNet category. By taking the argmax of the predicted probability vector, we obtain an integer corresponding to the model's predicted object class, which we can identify with an object category through the use of this dictionary.

c. Write a Dog Detector

While looking at the dictionary, you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from 'Chihuahua' to 'Mexican hairless'. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained ResNet-50 model, we need only check if the ResNet50_predict_labels function above returns a value between 151 and 268 (inclusive).

d. Assess the Dog Detector

Question: Use the code cell below to test the performance of your dog_detector function.

What percentage of the images in human_files_short have a detected dog? 0%

What percentage of the images in dog_files_short have a detected dog? 100%

4. Create a CNN to Classify Dog Breeds (from Scratch)

Now that we have functions for detecting humans and dogs in images, we need a way to predict breeds from images. In this step, we will create a CNN that classifies dog breeds. we aim to create CNN from scratch (so, we will not use transfer learning in this step!), due to the limited number of training images and the proposed shallow CNN, we aim to attain a test accuracy of at least 1%. In Step 5 of this notebook, we

will have the opportunity to use transfer learning to create a CNN that attains greatly improved accuracy. In this step, we avoid adding too many layers to maintain low computational complexity. More parameters mean longer training, which means we are more likely to need a GPU to accelerate the training process. Thankfully, Keras provides a handy estimate of the time that each epoch is likely to take; we can extrapolate this estimate to figure out how long it will take for our algorithm to train.

a. Model Architecture

Question 4: Outline the steps you took to get to your final CNN architecture and your reasoning at each step.

Answer: In the proposed CNN architecture we added convolutional layers and max-pooling to extract features from the input image, The number of filters in each step is increased to extract more high-level features in mid-layers. The last layer is a dense layer that outputs the classification probabilities.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 96)        12384
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 27, 27, 96)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 96)                0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               12901
=================================================================
Total params: 27,573
Trainable params: 27,573
Non-trainable params: 0
_________________________________________________________________
```

b. Test the Model

The test accuracy of 5.5024% was achieved by this architecture.

5. Create a CNN to Classify Dog Breeds (using Transfer Learning)

We will now use transfer learning to create a CNN that can identify dog breeds from images. It is expected our CNN model attain at least 60% accuracy on the test set. In Step 4, we used transfer learning to create a CNN using VGG-16 bottleneck features. In this section, we use the bottleneck features from the ResNet-50 pre-trained model.

a. Model Architecture

Question 5: Outline the steps you took to get to your final CNN architecture and your reasoning at each step. Describe why you think the architecture is suitable for the current problem.

Answer: we added three dense layers, where the number of neurons in hidden layers is larger than the last layer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_3 ( (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 520)               1065480
_________________________________________________________________
dense_4 (Dense)              (None, 180)               93780
_________________________________________________________________
dense_5 (Dense)              (None, 133)               24073
=================================================================
Total params: 1,183,333
Trainable params: 1,183,333
Non-trainable params: 0
_________________________________________________________________
```

b. Test the Model

The accuracy of 79.06% was achieved on the test set for the proposed architecture which is a significant increase in accuracy.

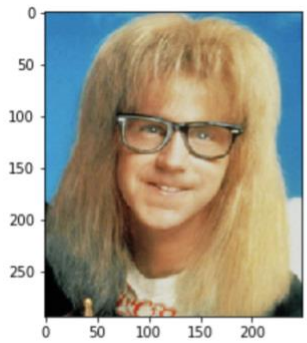
6. Write the Algorithm

We write an algorithm 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 an output that indicates an error.

```
def dog_human_model(img_path):
# read the image
    img = cv2.imread(img_path)
    RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(RGB_img)
    plt.show()
    if dog_detector(img_path):
        features =extract_Resnet50(path_to_tensor(train_files[1]))
        predict_vect=model.predict(features)
        dog_name = dog_names[np.argmax(predict_vect)]
        print("Dogs Detected!\nIt looks like a {0}".format(dog_name))
    elif face_detector(img_path):
        features =extract_Resnet50(path_to_tensor(train_files[1]))
        predict_vect=model.predict(features)
        dog_name = dog_names[np.argmax(predict_vect)]
        print("Hello, human!\nIf you were a dog..You may look like a {0}".format(dog_name))
    else:
        print("Error! Can't detect anything..")
```

7. Test Your Algorithm

We tested our algorithm on different images such as the following image:




```
Hello, human!
If you were a dog..You may look like a ages/train/057.Dalmatian
```

Question 6: Is the output better than you expected? Or worse? Provide at least three possible points of improvement for your algorithm.

Answer: The result showed that our model based on ResNet results in superior accuracy.