Deep Learning: How to build a dog detector and breed classifier using CNN?!

Rahil Bagheri Follow
Oct 8, 2019 · 10 min read

This work is part of Udacity's Data Science capstone project.

. . .

Overview:

You might think recognizing a dog's breed in an image is an easy task for you. You are right! It might not be difficult to find dog breed pairs with minimal inter-class variation for instance, Curly-Coated Retrievers and American Water Spaniels.

But, how about these two?!

Huh, not that easy!

In the last decade, it is much easier to use deep learning techniques with a few lines of python code to distinguish between dog breeds in images. In this blog, I will walk you through how to create Convolutional Neural Networks (CNN) from scratch and leverage the latest state of art image classification techniques on ImageNet. This model can be used as part of a mobile or web app for the real world and user-provided images. Given an image to the model, it determines if a dog is present and returns the estimated breed. If the image is human, it will return the most resembling dog breed. You can find the code in my GitHub repo.

Human detector

I used OpenCV's implementation of the Haar feature-based cascade object classifier to detect human faces. The cascade function is a machine learning-based approach trained on many images with positive(with a face) and negative(without any face) labels. The detectMultiScale gets the coordinates of all the faces then returns them as a list of rectangles. But don't forget to convert the RGB image into grayscale before using it.

The following face_detector function counts up how many human faces are in the photo:

```
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
```

The performance of face_detector evaluated on 100 samples of human and dog images. This detector recognizes all the human faces from human data but didn't perform well on the dog dataset. It had about 11% false positives.

Dog detector

It is time to use another detector that performs better on the dog dataset. I used pre-trained ResNet50 weights on ImageNet in Keras, which is trained on over 10 million images containing 1000 labels.

In the following code, paths_to_tensor takes the path to an image and returns a 4D tensor ready for ResNet50. But, all the pre-trained models in Keras need additional processing like normalization that can be done by using preprocess_input. The dog_detector function returns "True" if a dog is detected in the image stored at img_path.

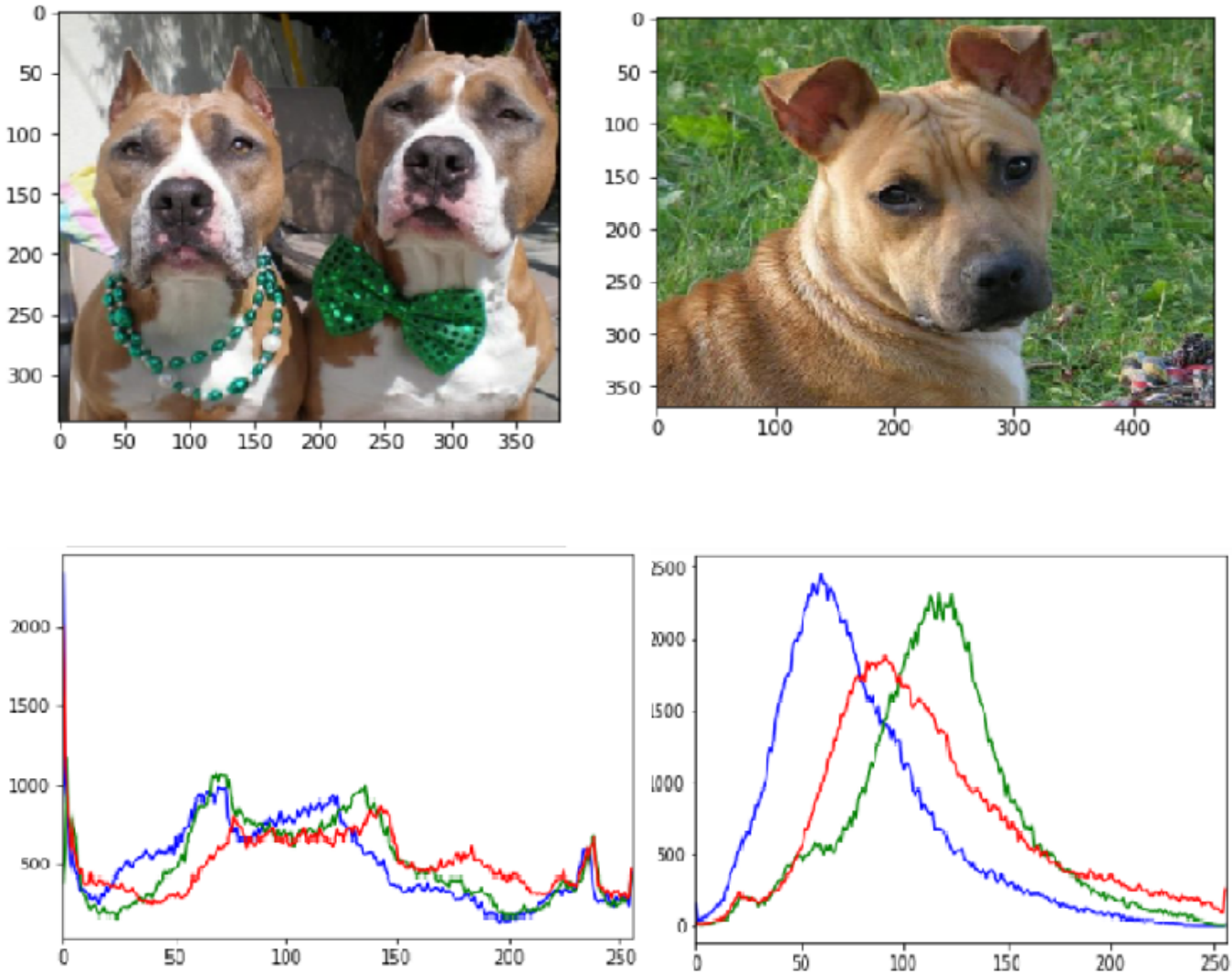
```
from keras.preprocessing import image
from tqdm import tqdm
from keras.applications.resnet50 import preprocess_input,
decode_predictions

def path_to_tensor(img_path):
    img = image.load_img(img_path, target_size=(224, 224)
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in
tqdm(img_paths)]
    return np.vstack(list_of_tensors)

def ResNet50_predict_labels(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))
```

To assess the dog detector, I checked if the predicted class of RestNet50 on ImageNet falls into the dog breed categories. The dog detector performs well without any false negatives.

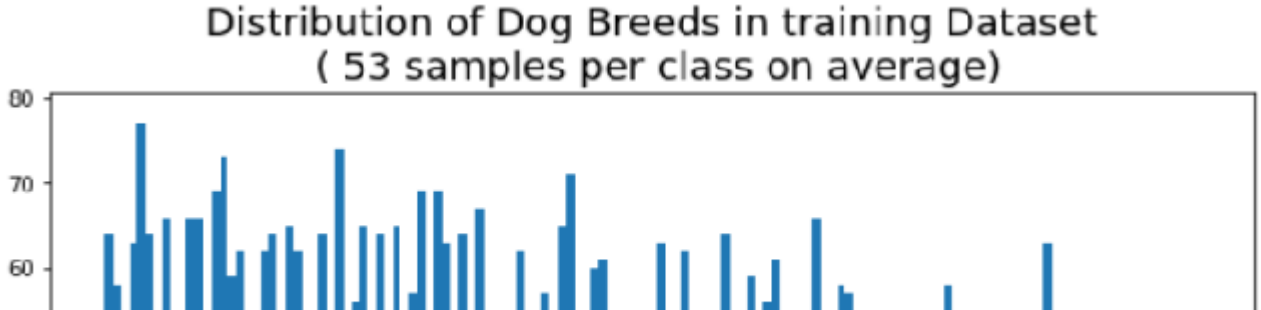
Now that we recognized dogs in images, it is time to predict breeds. But first we should analyze train dataset a little more in details:

Data set

This data set has 8,351 total images with 133 different breeds. The number of available images for the model to learn from is about 62 per kind, which might not be enough for CNN. In this real-world setting, the images have different resolutions, sizes, lightening conditions, also some images have more than one dog. By comparing the pixel intensity distribution of the same labeled images, I noticed, for example, photos for American_staffordshire_terrier are varying in contrast, size, and brightness. The Red, Green, and Blue channels' intensity values found in these two images are distributed differently. This variation in data makes the task of assigning breed to dogs even more challenging.



The distribution of breed labels in training data shows data is slightly imbalanced, with, on average, 53 samples per class. Most of the breeds have almost the same distribution in train, valid, and test datasets.



Also, as part of data processing, I descaled the images by dividing every pixel in every image by 255.

Metrics

Since we are dealing with a multi-classification problem here and the data is slightly imbalanced, I used accuracy evaluation metric and categorical_crossentropy cost function. But, first, the labels have to be in a categorical format. The target files are the list of encoded dog labels related to the image with this format. This multi-class log loss punishes the classifier if the predicted probability leads to a different label than the actual and cause higher accuracy. A perfect classifier has a loss of zero and an accuracy of 100%.

Classify dog breeds

Here I created a 4-layer CNN in Keras with Relu activation function. The model starts with an input image of 224 *224*3 color channels. This input image is big but very shallow, just R, G, and B. The convolution layers squeeze images by reducing width and height while increasing the depth layer by layer. By adding more filters, the network can learn more significant features in the photos and generalize better.

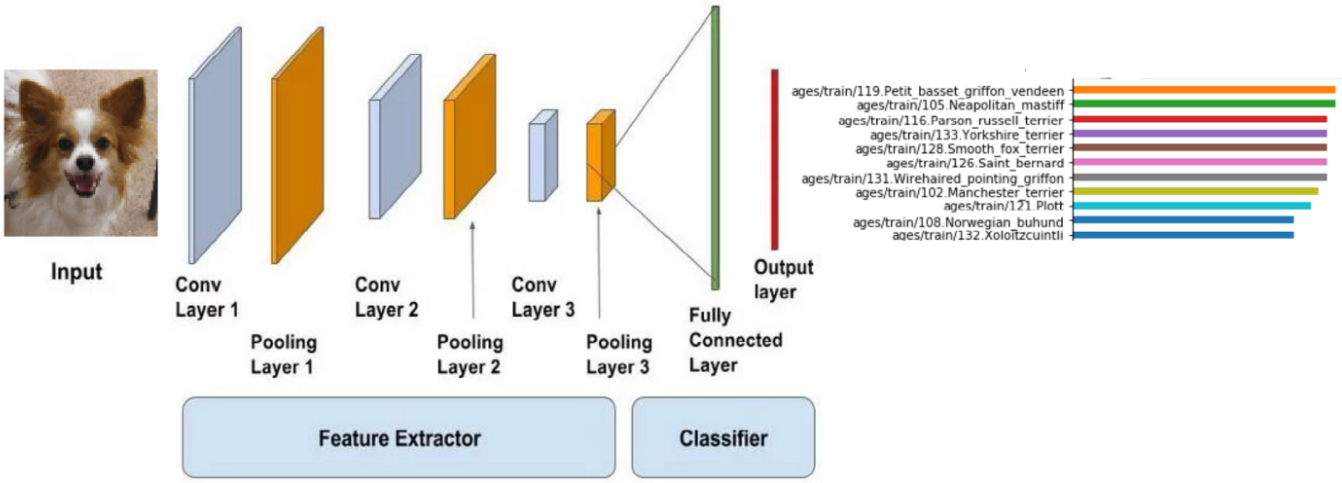
My first layer produces an output with 16 feature channels that is used as an input for the next layer. The filters are the collection of 16 square matrices, output feature maps, which are weighted sums of input features and kernel. The kernel's weights are calculated during the training process by ImageNet data, and what it does is to slide across the input feature maps and produce output features. So, the shape of output features depends on the size of the kernel and input features.

Check this page for a better understanding of how CNN works.

I think it would be ideal for input and output features to have the same size. So, I decided to use same padding to go off the edge of images with zero pads for all the

layers with the stride of 2. I also used the max-pooling operation to ensure I am not losing information in the picture while lowering the chance of overfitting. Max pooling takes the maximum of pixels around a location.

After four Convolutional layers and max-pooling, followed by two fully connected layers, I trained the classifier. The Convolutional layers extract the image features, and the classifier classifies them based on the previously obtained features. The image below shows how the sequence of feature blocks and classifier on top transfer the information from the raw image and predict requested target values.



The CNN model architecture is:

```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

# Model Architecture
model = Sequential()

model.add(Conv2D(filters=16, kernel_size=2, padding='same',
activation='relu', input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=2))


model.add(Conv2D(filters=32, kernel_size=2 , padding='same' ,
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=64 , kernel_size=2 , padding='same' ,
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.4))

model.add(Conv2D(filters=128 , kernel_size=2 , padding='same' ,
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.4))
```

```
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(133,activation='softmax'))

model.summary()
```

The created model from scratch is not performing well, accuracy 12%, due to not having enough images data to train the model. One potential improvement is data augmentation to add more data. This strategy modifies the images by padding, cropping, and rotating images randomly. It also enables the model to generalize better without overfitting, of course, with the appropriate parameters.

As it was mentioned before, training a CNN classifier made from scratch on a small data like this will lead to underfitting and with so many layers, and parameter tuning often causes overfitting. So, it is time to utilize transfer learning pre-trained networks to create a CNN breed classifier even though these models are not explicitly made for this task. But one advantage of these networks is that they are trained on large datasets, ImageNet with millions of labeled images, and reached the 90% accuracy. Also, they can generalize to other images outside the ImageNet.

I implemented the pre-trained models like VGG, Inception V3, and ResNet in Keras and compared their accuracy. I used them as fixed feature extractors and fine-tuned the fully-connected layer by minimizing the cross-entropy loss function using stochastic gradient descent, the learning rate of 0.001 and Nesterov momentum. Also, the last fully connected layer is changed to the number of dog breeds, 133. Then, each model was trained on 25 epochs with 20 samples each batch.

```
# model Architecture
Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=
(train_Xception.shape[1:])))
Xception_model.add(Dense(133, activation='softmax'))

checkpointer =
ModelCheckpoint(filepath='saved_models/weights.best.Xception.hdf5',
verbose = 0, save_best_only=True)

sgd = SGD(lr= 1e-3 , decay=1e-6, momentum=0.9 , nesterov = True)

# compile the model
Xception_model.compile(loss='categorical_crossentropy',
optimizer=sgd, metrics=['accuracy'])

# Training the model
Xception_model.fit(train_Xception , train_targets,
               validation_data = (valid_Xception, valid_targets),
```

```
                shuffle = True,
                batch_size = 20,
                epochs = 25,
                verbose = 1)
```

The Xception model outperforms all the other models with an accuracy of 86% and a loss of 0.4723 on test data. In general, Inception based models slightly outperform VGG and ResNet on ImageNet; also, it is more computationally efficient. The model performance could be improved by applying more appropriate fine-tuning techniques. For example, training a few top layers related to specific dog features and freeze the others that detect more generic features. Because they are already captured in the ImageNet weights.

Result

Now, it is time to combine the detectors and Xception-predict-breed, also, test them on several images with different sizes and resolutions. As mentioned before, accuracy was chosen as a model evaluation metric for this classifier. This code takes a path to an image and first determines whether the image is a dog, human or neither then returns the predicted resembling breed:

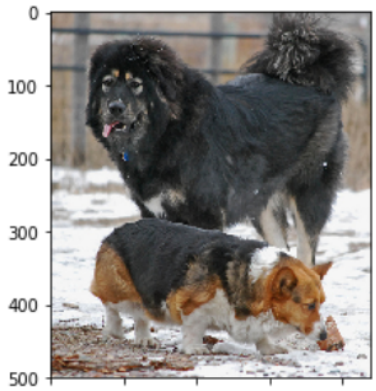
```
def Xception_predict_breed (img_path):
    bottleneck_feature = extract_Xception(path_to_tensor(img_path))
    predicted_vector = Xception_model.predict(bottleneck_feature)
    return dog_names[np.argmax(predicted_vector)]

def display_img(img_path):
    img = cv2.imread(img_path)
    cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    imgplot = plt.imshow(cv_rgb)
    return imgplot

def breed_identifier(img_path):
    display_img(img_path)
    prediction = Xception_predict_breed(img_path)
    if dog_detector(img_path) == True:
        print('picture is a dog')
        return print (f"This dog is a {prediction}\n")

    if face_detector(img_path) == True:
        print('picture is a human')
        return print (f"This person looks like a {prediction}\n")

    else:
        return print('The picture is neither dog nor human')
```

The result shows that the model is performing well. For example, it could predict American_staffordshire_terrier in test data while it was trained on images with different

brightness, sizes, and zoom level.

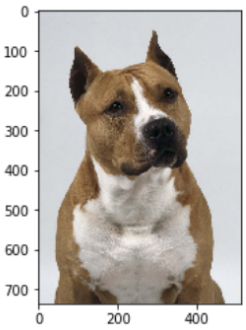
```
breed_identifier('../../../data/dog_images/test/129.Tibetan_mastiff/Tibetan_mastiff_08156.jpg')
```

```
picture is a dog
This dog is a ages/train/129.Tibetan_mastiff
```




```
breed_identifier('../../../data/dog_images/test/008.American_staffordshire_terrier/American_staffordshire_terrier_00538.
```

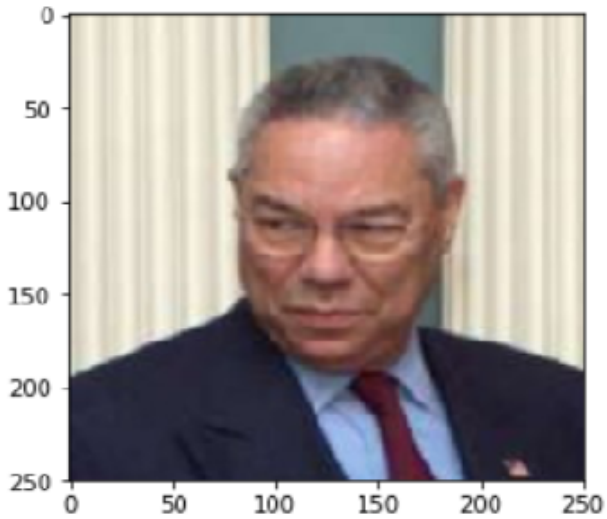
```
picture is a dog
This dog is a ages/train/008.American_staffordshire_terrier
```



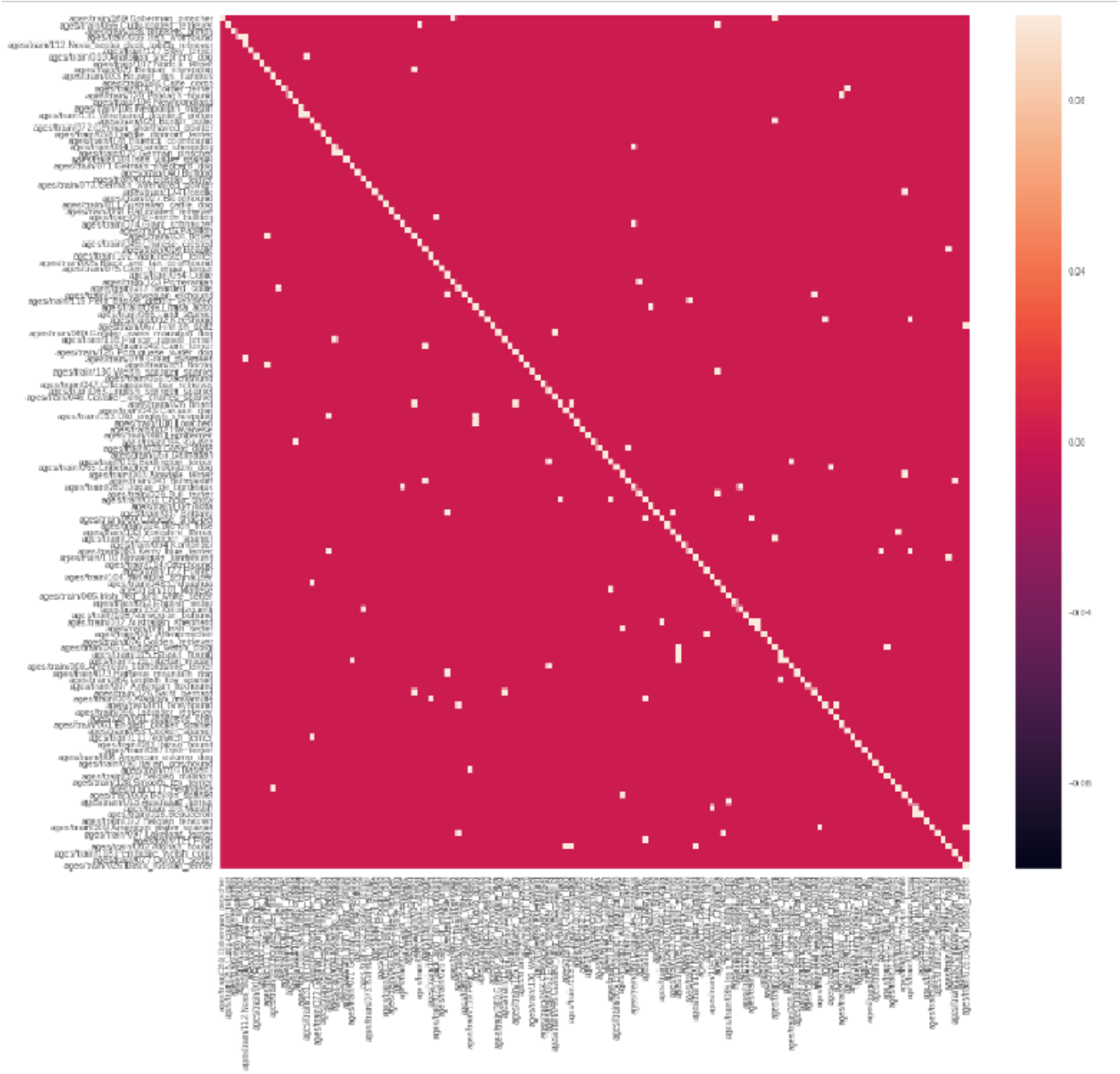
And the following is the result on human images:

```
breed_identifier(human_files[20])
```

```
picture is a human
This person looks like a ages/train/049.Chinese_crested
```



The confusion matrix of the predictions on the test data is given below. But, it is not easy to visualize which breeds are most misclassified.



For a better understanding of misclassified cases, I paired the predicted and actual values. The model couldn't classify between Great_pyrenees and Kuvasz, which both are white, big, and fluffy. It also couldn't distinguish between Mastiff and Bullmastiff, which is a mix between a bulldog and a mastiff. By adding more data, the model would be able to learn more specific features and distinguish between these similar breeds, which is even confusing for humans.

Potential improvements

The model accuracy could improve by data augmentation and adding more training labeled data since the model was slightly imbalanced. With acquiring more data, the Convnet models would be more capable of learning the more relevant and specific dog features from the images. But, it requires a considerable amount of time and memory

usage. Another possible way to enhance the model accuracy is averaging the predictions of several networks to get the final prediction known as ensemble techniques. As I mentioned before, one improvement could be applying more appropriate fine-tune strategies to adjust the weights based on the data. Also, a proper analysis of the final model would be to add some noise to the images and see how it is performing.

Conclusion

In this work, I have implemented the haar cascade classifier and ResNet50 pre-trained weights to detect human faces and dogs. I also spent some time exploring the data to find images characteristics, which helped me to apply more appropriate techniques. For dog breed classification, I have started with a few numbers of Convolutional layers as a baseline network. The model performed better than a random guess, 1 in 133, but still underfits.

With the given time and memory power, I decided to leverage several transfer learning architectures to enhance the CNN performance, which resulted in a significant improvement. I used the pre-trained models as fixed features extractor and changed the global average pooling layer, fully connected layer, and applied other fine-tuning approaches. The Xception and ResNet50 produced higher accuracy than others, around 86%. It is quite challenging to get a higher accuracy due to the complexity of these kinds of datasets with some similarity between mixed breeds.

In the end, I tested the model on the same labeled photos with different intensity distributions, and it performed quite well.

The most exciting part of this project was how modern ConvNets with a few lines of codes made it easy for us to do the tasks that even humans find challenging. But it is quite hard to improve the model with the given time and memory power. I plan to explore the mentioned improvement approaches and will update this blog with better results.

. . .

please feel free to use my code if you found it interesting.

Machine Learning Data Science Image Classification Deep Learning Transfer Learning