Dog breed identification as A study of CNN for image classification

Problem Introduction

Studying the Udacity Nanodegree Program's Data Science course, I am interested in computer vision and applying Convolutional neural networks (CNN) to a real problem. Convolutional neural networks (CNN) are a type of deep learning neural networks that are commonly used to classify images. CNNs are known for their ability to reduce computational time and adapt to different variations of images. Recognizing a dog's breed from its image is a typical problem that can be applied with CNN.

I also found that the task of assigning breed to dogs from images is challenging but also quite interesting. Dogs in images are in all shapes and sizes, and frequently without pedigrees to describe their breed or heritage. The identities of dogs with unknown or mixed-breed lineages are frequently guessed based on their physical appearance. There are hundreds of dog breed and many people may feel surprised to know "what kind of dog is that? Or so strange dog". Just like a game of curiosity.

Strategy to solve the problem

In following sections, I will try to introduce to you the way we can apply Convolutional Neural Networks (CNN) using the Keras library with python for the dog breed recognization. Due to saving resource (GPU) I will not focus on designing CNN to achieve the best accuracy but I will try to show both ways of building CNN: 1/ building CNN from scratch with an acceptable layer modeling, and 2/ using transfer learning to create a CNN that attains greatly improved accuracy.

Together with dog breed classification, I also test a human face detector using OpenCV's implementation of Haar feature-based cascade classifiers. OpenCV provides many pre-trained face detectors, stored as XML files on github. I downloaded one of these detectors and stored it in the haarcascades directory to use.

EDA

Loading the given data set by Scikit learn's datasets and Keras library' utils I see that 8,351 dog images with 133 different categories (dog breeds).

```
There are 133 total dog categories.
There are 8351 total dog images.
```

The number of available images for the model to learn from is about 62 images per category, this is not a so good number for CNN. By randomly checking I see that the images may have different resolutions, sizes and lightening conditions.

```
There are 6680 training dog images.
```

```
There are 835 validation dog images.
There are 836 test dog images.
```

Roughly checking the training set I see that there are on average 53 samples per category and not balanced (equal) between categories. Another point I see it's quite good is that most of the breeds have almost the same distribution in train, valid, and test datasets.

Modelling

To build a CNN from scratch I created a 3-layer CNN in Keras with relu activation function as below. The target of this CNN to attain a test accuracy of at least 1% and not to consume much GPU and time for training so I intend to reduce and get a CNN with small enough parameters.

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

from keras import datasets, layers, models

model = Sequential()

### TODO: Define your architecture.
model.add(layers.Conv2D(224, (3, 3), activation='relu', input_shape=(224,
224, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (2, 2), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(133, activation='relu'))
```

- The model will get the input image of 224 *224*3 color channels. So the input image is quite big but very shallow, just R, G, and B color information.
- The sequential 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.
- 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 3 convolutional and max-pooling layers, there will be two fully connected layers.

Summarizing the model and its parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 224)     6272
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 224)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 32)      28704
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 54, 54, 32)        4128
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 27, 27, 32)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               4389
=================================================================
Total params: 43,493
Trainable params: 43,493
Non-trainable params: 0
_________________________________________________________________
```

With such design this CNN can run quickly, just more than 100s for each epoch.

```
Train on 6680 samples, validate on 835 samples
Epoch 1/10
6660/6680 [============================>.] - ETA: 0s - loss: 10.5087 - acc: 0.0092Epoch 00001: val_loss improved from inf to 1
0.80553, saving model to saved_models/weights.best.from_scratch.hdf5
6680/6680 [==============================] - 105s 16ms/step - loss: 10.5073 - acc: 0.0091 - val_loss: 10.8055 - val_acc: 0.0108
Epoch 2/10
6660/6680 [============================>.] - ETA: 0s - loss: 12.0236 - acc: 0.0096Epoch 00002: val_loss did not improve
6680/6680 [==============================] - 103s 15ms/step - loss: 12.0264 - acc: 0.0096 - val_loss: 13.0844 - val_acc: 0.0120
Epoch 3/10
6660/6680 [============================>.] - ETA: 0s - loss: 13.2363 - acc: 0.0077Epoch 00003: val_loss did not improve
6680/6680 [==============================] - 103s 15ms/step - loss: 13.2376 - acc: 0.0076 - val_loss: 12.8328 - val_acc: 0.0096
Epoch 4/10
6660/6680 [============================>.] - ETA: 0s - loss: 12.4911 - acc: 0.0078Epoch 00004: val_loss did not improve
6680/6680 [==============================] - 104s 16ms/step - loss: 12.4834 - acc: 0.0078 - val_loss: 12.0970 - val_acc: 0.0096
Epoch 5/10
6660/6680 [============================>.] - ETA: 0s - loss: 11.9593 - acc: 0.0086Epoch 00005: val_loss did not improve
6680/6680 [==============================] - 103s 15ms/step - loss: 11.9549 - acc: 0.0085 - val_loss: 12.3087 - val_acc: 0.0072
Epoch 6/10
6660/6680 [============================>.] - ETA: 0s - loss: 12.1879 - acc: 0.0066Epoch 00006: val_loss did not improve
6680/6680 [==============================] - 103s 15ms/step - loss: 12.1906 - acc: 0.0066 - val_loss: 11.9374 - val_acc: 0.0096
Epoch 7/10
6660/6680 [============================>.] - ETA: 0s - loss: 13.9688 - acc: 0.0090Epoch 00007: val_loss did not improve
6680/6680 [==============================] - 103s 15ms/step - loss: 13.9730 - acc: 0.0090 - val_loss: 15.5573 - val_acc: 0.0108
Epoch 8/10
6660/6680 [============================>.] - ETA: 0s - loss: 15.5876 - acc: 0.0093Epoch 00008: val_loss did not improve
6680/6680 [==============================] - 104s 16ms/step - loss: 15.5870 - acc: 0.0093 - val_loss: 15.6249 - val_acc: 0.0072
Epoch 9/10
6660/6680 [============================>.] - ETA: 0s - loss: 15.0951 - acc: 0.0117Epoch 00009: val_loss did not improve
6680/6680 [==============================] - 104s 15ms/step - loss: 15.0922 - acc: 0.0118 - val_loss: 14.4829 - val_acc: 0.0084
Epoch 10/10
6660/6680 [============================>.] - ETA: 0s - loss: 14.3204 - acc: 0.0092Epoch 00010: val_loss did not improve
6680/6680 [==============================] - 103s 15ms/step - loss: 14.3139 - acc: 0.0091 - val_loss: 14.2894 - val_acc: 0.0108
```

And it still helps to get the accuracy > 1%.

Load the Model with the Best Validation Loss

```
model.load_weights('saved_models/weights.best.from_scratch.hdf5')
```

Test the Model

Try out your model on the test dataset of dog images. Ensure that your test accuracy is greater than 1%.

```
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, axis=0))) for tensor in test_tensors]

# report test accuracy
test_accuracy = 100*np.sum(np.array(dog_breed_predictions)==np.argmax(test_targets, axis=1))/len(dog_breed_predictions)
print('Test accuracy: %.4f%%' % test_accuracy)
```

```
Test accuracy: 1.0766%
```

Metrics

Refering to multi-class classification and the data is slightly imbalanced as noticed in EDA section, I used accuracy evaluation metric and categorical_crossentropy cost function. The dog labels (breed name) should be in a categorical format. The optimizer with RMSprop algorithm to maintain a moving (discounted) average of the square of gradients and divide the gradient by the root of this average and use that average to estimate the variance.

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

Hyperparameter tuning

The created model from scratch is not performing well, accuracy 1%, due to not having enough images data to train the model. As I understand a small training data will be easy to lead to underfitting and if I try to design a CNN with so many layers and parameter it will cost GPU resource to do training and often cause overfitting.

That's why it's reasonable to use pre-trained networks to transfer learning and create a CNN of dog breed classifier.

The pre-trained models can be used as a fixed feature extractor, where the last convolutional output of them is fed as input to the transfer-learning model. I only add a average pooling layer and a fully connected layer containing one node for each dog category and with a softmax.

For VGG16:

```
VGG16_model = Sequential()
VGG16_model.add(GlobalAveragePooling2D(input_shape=train_VGG16.shape[1:]))
VGG16_model.add(Dense(133, activation='softmax'))

VGG16_model.summary()
```

For ResNet50:

```
### TODO: Define your architecture.
Resnet50_model = Sequential()
Resnet50_model.add(GlobalAveragePooling2D(input_shape=train_Resnet50.shape[1:]))
```

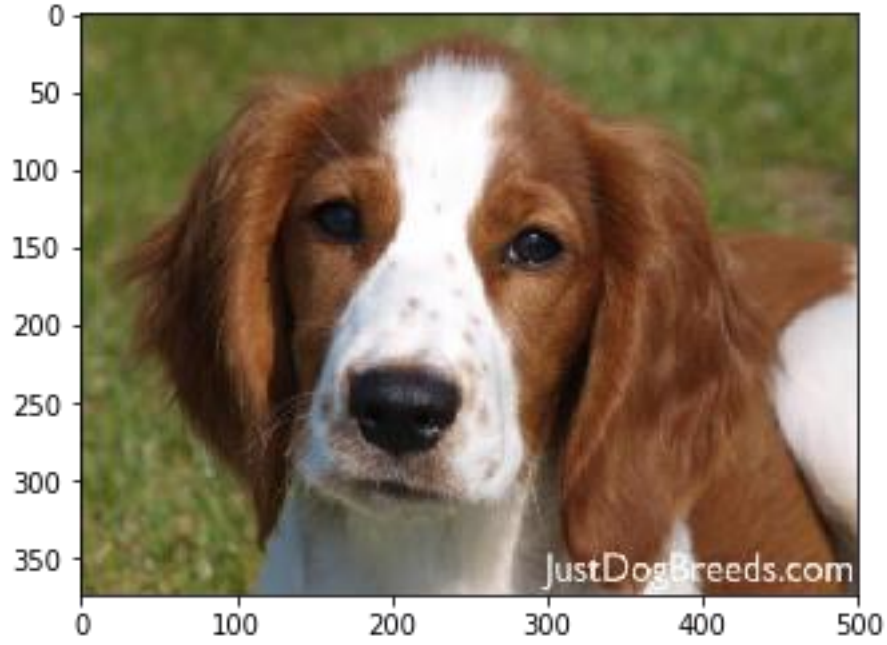
```
Resnet50_model.add(Dense(133, activation='softmax'))

Resnet50_model.summary()
```

I can see that even though these models are not explicitly made for the task of dog breed classifier but they can help to improve the accuracy a lot (to get > 40% with VGG-16 and 80% with ResNet50).

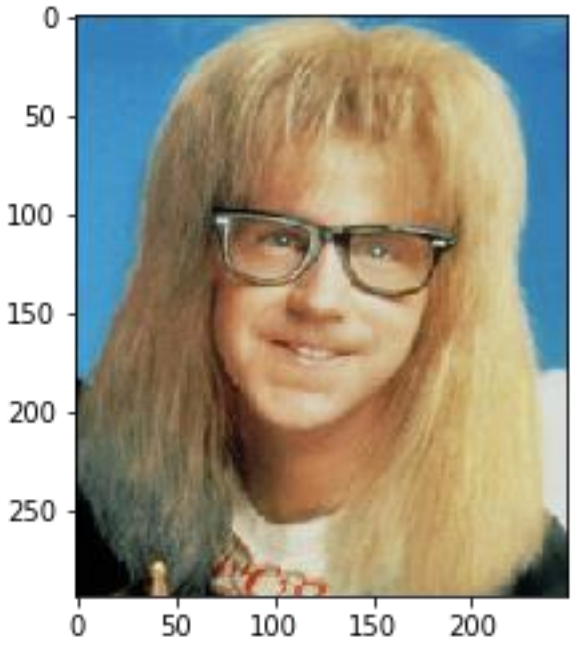
Results

After training the models (with epochs = 20), I can load the model with the best validation loss and test within the test dataset of dog images and calculate the accuracy. The transfer-learning ResNet50 model outperforms the VGG-16 model, it gives an accuracy of 80% and the best validation loss of 0.63336.

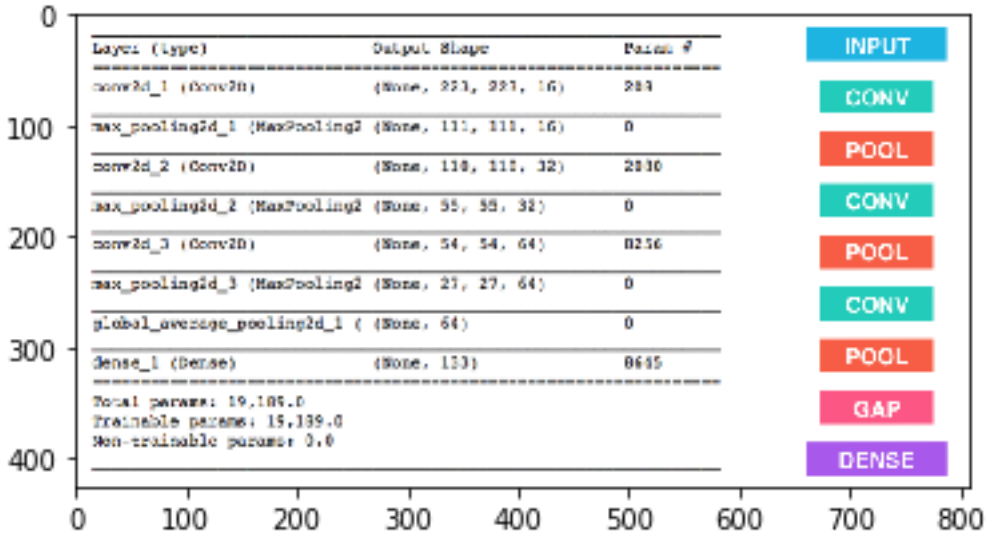
In the end, I wrote a simple application to test the model on the sample photos. It can return the predicted breed if a dog is detected, the resembling dog breed if a human face is detected and an error if neither dog or human face is detected in the image.




```
This photo looks like Welsh springer spaniel
```



```
This photo looks like Silky terrier
```




```
Error: This photo is not human either dog detected!
```

Conclusion/Reflection

In this practice, I have studied about building a CNN from scratch and transfer pre-trained CNN to classify dog breeds and also implemented a haar cascade classifier to detect human face.

For dog breed classification, I created a base line CNN with a few numbers of convolutional layers. The model performed better than a random guess (1%> 1 out of 133 breeds) but underfit. With the limited time and resource, leverageing a transfer-learning architectures to enhance the CNN

performance is a good approach and it resulted in a significant improvement. Basing on the guidelines from transfering VGG-16 model, I also used the pre-trained model ResNet50 which is also light to be loaded to transfer and got the high accuracy around 80%.

Improvements

We can see the problem of not having enough images data to train the model. One potential improvement is data augmentation to add more data. It is possible to modify the images by resizing, cropping, and rotating images randomly. It also enables the model to generalize better without overfitting with the appropriate parameters. Another possible way to enhance the model accuracy is combinining/averaging the predicting results of several CNNs to get the final prediction (known as ensemble techniques).

I also think about the dog breed classification can be used in an attempt to predict such things like matching dogs to families, finding lost dogs... This practice is a chance for me to keep going far ahead of the theory in deep learning later on.