1. Abstract

Since the advent of ImageNet, the field of Computer Vision has witnessed a monumental growth in areas including Image Classification & Recognition, Object Detection, Autonomous Vehicles and etc.

In some or all of the areas, Convolutional Neural Networks (CNN) have achieved close to or even better than human performance. CNNs are inspired by the biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Compared to other image classification algorithms, CNNs require relatively little pre-processing. This independence from prior knowledge and human effort in feature design is a major advantage of CNNs.

2. Introduction

In this project, we try to build neural networks that can achieve high accuracy of classifying dog breed. Specifically, we first build a convolutional neural network (CNN) and assess its effectiveness. Then we experiment with transfer learning by applying pretrained models (e.g., VGG16) to the same testing set, in the hope to compare the performance between CNN and pretrained models.

The dataset used in this project is part of ImageNet (specifically, the Stanford Dogs Dataset), and can be downloaded from here.

The entire dataset contains 20,580 images with a total of 120 possible breeds. Among all of the 20,580 images, 12,000 of them are for training and 8,580 of them are for testing. In this project, we'll further split 20% of the training set to be the validation set.

The dataset also contains a set of annotation files (in XML format) that associate images with bounding boxes, indicating more accurate positions of dogs within the images.

3. Background

Data and computational power are the bread and butter for deep neural networks. On one hand, more data make it possible, in the first place, for deep neural networks to tune its large number of parameters without necessarily overfitting the data. On the other hand, large computational power is the key factor to train deep neural networks faster.

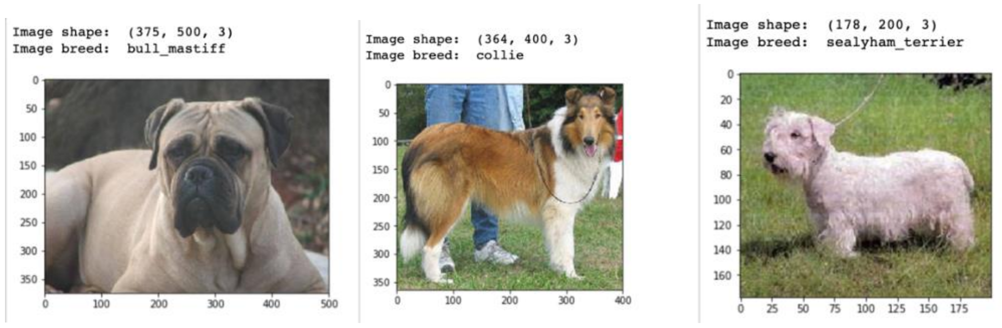
In this project, each dog breed contains roughly 9,600 / 120 = 80 images in the training set, which is relatively few. Therefore, the CNN built from scratch is likely to yield decent results, but nowhere close to the state-of-the-art performance.

Even with a larger set of date, the limited computational power restricts our ability to train the deep neural networks efficiently. An alternative approach is to exploit transfer learning, which applies pretrained models to the domain of interest, with or without additional training.

4. Method

- **Data Exploration**

  A few example pictures from the training set are displayed below, each with the image size and labeled breed.



```
Image shape:  (375, 500, 3)
Image breed:  bull_mastiff
```

```
Image shape:  (364, 400, 3)
Image breed:  collie
```

```
Image shape:  (178, 200, 3)
Image breed:  sealyham_terrier
```

- **Data Preprocessing**

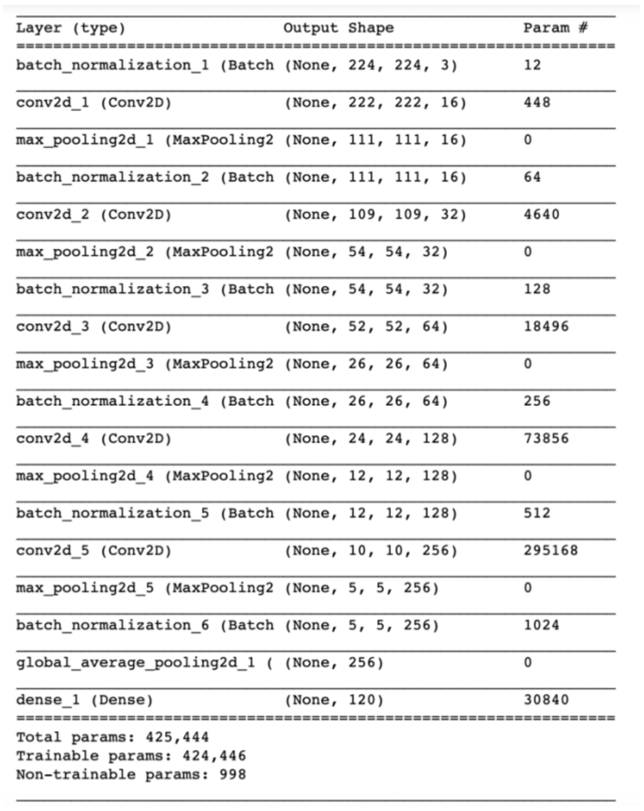
  Firstly, we download the dataset, extract the *.tar* files and organize images under */train* and */test*, under each of which images are grouped by breed. Note that during this step, we split the training set and testing set according to *train_list.mat* and *test_list.mat*. Specifically, exactly 100 images of each breed are assigned to the training set (that is a total of 120 * 100 images), whereas the remaining images of the breed belong to the testing set.

Secondly, we cut the original images according to the bounding boxes as indicated in the annotation files, followed by resizing all images to the same dimension. All processed images are saved under */resize/train* and */resize/test*.

Lastly, we concatenate all training images and all testing images respectively to form matrices *train_X*, *train_y*, *test_X*, and *test_y*. We then split the training set further by the 80/20 rule, leading to the final training set of 9,600 images and validation set of 2,400 images. The testing set contains a total of 8,580 images.

- **Model (CNN) Train and Test**

The CNN architecture used in this project is as follows:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_1 (Batch (None, 224, 224, 3)       12
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 222, 222, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
batch_normalization_2 (Batch (None, 111, 111, 16)      64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 109, 109, 32)      4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 54, 54, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 52, 52, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 26, 26, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 256)       295168
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 5, 5, 256)         1024
_________________________________________________________________
global_average_pooling2d_1 ( (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               30840
=================================================================
Total params: 425,444
Trainable params: 424,446
Non-trainable params: 998
_________________________________________________________________
```

The CNN architecture consists of 5 consecutive (convolutional, max pooling, batch normalization) layers followed by a global average pooling layer and a fully connected layer, with a total of 424,446 trainable parameters and 998 non-trainable parameters.

Additionally, we choose the Adam optimizer since it empirically outperforms other optimization techniques including RMSProp and more. The loss function is sparse categorical cross-entropy. Note that sparse categorical cross-entropy, instead of categorical cross-entropy is used, because our labels are integers labels, rather than one-hot vector labels.

Note also that we fit the training set to an ImageDataGenerator to achieve real-time data augmentation by randomly shifting images horizontally/vertically, as well as flipping horizontally. The purpose of data augmentation is to help CNN avoid overfitting.

In terms of the measurement for the testing accuracy, we simply calculate the percentage of correctly predicted labels on the testing set. We'll also manually examine the top several misclassified dog breeds by visualizations.

- **Transfer Learning**

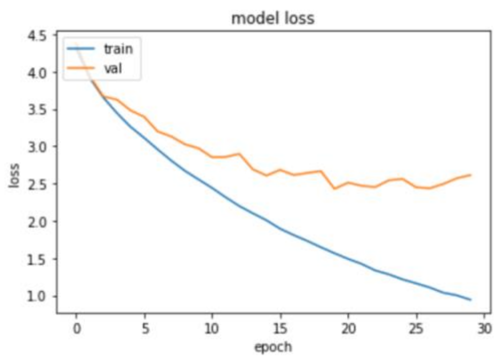
Transfer learning refers to the technique that exploits what has been learned in one setting to improve generalization in another setting.

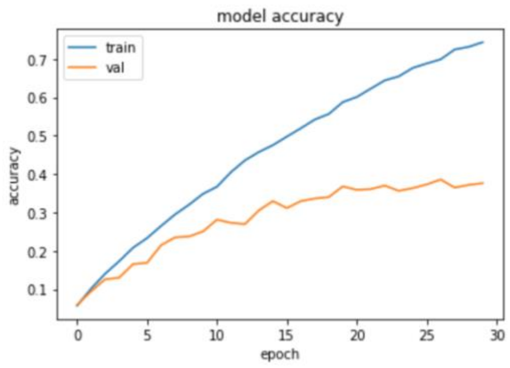
However, as is shown in the **Result** section below, since our CNN architecture already overfits the training set, a more complicated model is more likely to overfit, especially with the limited size of the dataset. Therefore, it is expected that transfer learning will yield a high training accuracy, but a much lower testing accuracy.

Possible solutions (also future directions) include: 1. Retrain the pretrained models with regularization (e.g., batch normalization, L2 penalty and dropout), 2. Apply data augmentation techniques on the training data.

5. Result

- **Train & Validation**

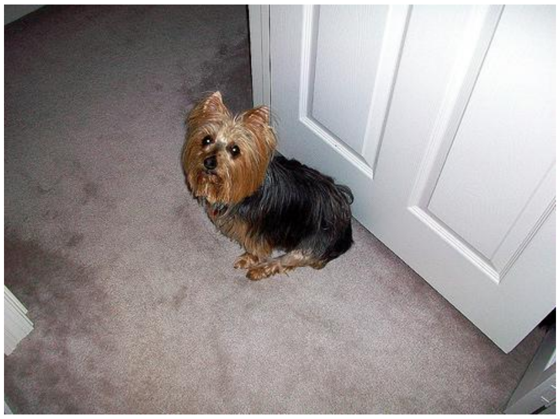




As suggested above, loss and accuracy of the validation set smooth out after around 20 epochs. This suggests that the CNN architecture starts to overfit the training set from the 20th epoch. Given the analysis discussed in the **Background** section, it is not surprising that the CNN architecture overfits the training set.

- **Test**

After being retrained for 20 epochs, the CNN architecture is able to achieve a test accuracy of 36.88%. The following shows several misclassified images in the testing set.



The left image has a truth label of Australian terrier, but is predicted as a silky terrier. The right has a truth label of Gordon setter, but is predicted as a bluetick. As a comparison, an example silky terrier and an example bluetick look like the following, respectively.