Dog breed classifier using CNN 1

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 223, 223, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 111, 111, 16)      0
_________________________________________________________________
batch_normalization_1 (Batch (None, 111, 111, 16)      64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 55, 55, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 55, 55, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 54, 54, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 27, 27, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 27, 27, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 27, 27, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 26, 26, 128)       32896
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 128)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 13, 13, 128)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
global_average_pooling2d_1 ( (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               17157
=================================================================
Total params: 61,557
Trainable params: 61,077
Non-trainable params: 480
```

used VGG16 to demonstrate the use of Transfer Learning. Bottleneck features is the concept of taking a pre-trained model and chopping off the top classifying layer, and then providing this “chopped” VGG16 as the first layer into our model.

format for use with Caffe. The overall image resizing steps are shown in Fig. 3 below.

Fig. 3: Image Resizing Steps Taken

The learning framework/software that I decided to use was Caffe. There were many reasons for this. First, the class encourages students to learn Caffe, which is an open-use software package that has an optimized tradeoff between ease of use and speed. Also, porting the training from CPU to GPU is very easy, as it only involves setting a flag. The prototxt format is also quite easy to understand and make work. However, I had many issues getting Caffe to work for my project. I started off installing Caffe in Ubuntu on Virtualbox, and running the training on the CPU in the virtual machine. However, I abandoned the effort soon after I realized that this would not go anywhere, as the machine would be able to do only 500 or so iterations over 12 hours. The training went much faster after I moved to Terminal and used their GPUs to train, but I had lost a large chunk of time in the process.

All of the results from the testing of the nets are summarized in Table 2. The testing accuracy are based on the cropped and resized testing dataset, using Top-1 accuracy. All learning was done using learning rate of 0.0005, momentum of 0.9, weight_decay of 0.0005. All CONV layers were regularized with the weight_decay. The hyperparameters were determined through 10+ fine-tuning iterations using different parameters from the parameter space to determine the optimal one for the problem as I have defined it.

I first started testing with LeNets, going from 3 layers of CONV-RELU-POOL, all the way to 6 layers. I experimented with varying the convolutional filter depths (50 and 500) along the way, while sticking to a filter size of 5x5. The accuracy results I obtained after 100,000 iterations were not good (< 2%) when I have less than 6 layers with 1 fully-connected layer, and when I increased the network to be 6 layers with 2 fully-connected layers, it went up to 9.4% accuracy.

For my testing with GoogLeNets, I used a similar approach with the number of layers, going from 3-7 layers. I kept the size of the filters per layer the same as in Fig. 2, and only varied the depths of the filters in my testing.

One thing I noticed early on with training is that the quality of convergence of the nets vary wildly depending on the learning rate. For example, if I use a learning rate of 0.0001, even at 100,000 iterations, the accuracy of the model will still be less than 2%, while if I use a learning rate of 0.0005, the model converges quite well to the maximum accuracy, and a learning rate of 0.001 causes the loss to blow up within 2000 iterations.

I visualized some filters below from one example of LeNet that I trained. Unfortunately I experienced a weird bug in pycaffe when I tried to visualize some of the larger LeNets or GoogLeNets. In these situations, if I load the model with pycaffe, all the weights would appear to be NaN's and the test accuracies would appear to be uniformly distributed across all classes, but when I use the caffe –test interface in terminal, it would give an accuracy with a low testing loss (~0.5) as well as the accuracy it displayed while testing (~9%). I am not sure why this would be the case, and I wonder if my computer has memory issues which make it unable to load larger nets. Therefore, the visualization below is only from a 3-layer LeNet, with depth-50 filters, which achieved below 2% accuracy during testing. Although I would have preferred a visualization of a CNN that did better, I figured that it was better to have some visualization than none at all.

The visualizations below show the original image (Fig. 4) that was fed into the filters, the first and second layer filters themselves, and outputs from the first and second convolutional layers. The first layer filters (Fig. 5) are low enough in number, with small number of channels (3 channels for RGB since they are the first layer), that we can visualize them by simply combining the 3 channels and representing them in RGB format. Therefore, the first layer filters are shown in color, and there are 50 of them, representing a depth of 50. The outputs from the first layer Fig. 6 are plotted in grayscale, and show the result of convolving the image below with the filters. To make it convenient for viewing, I am only showing the outputs from the first 36 channels, so the resulting image matrix is square. We can already see some interesting results from the output of the first layer. Firstly, in some of the outputs, the contrast between the white and black portions of the dog are maintained (though they may be reversed so that black becomes white and vice versa), but in other outputs there are barely any differences between the two. We can see that in the 50 inputs to the second layer for this particular image, some inputs will have barely any signal and contrast, while others will maintain their strong contrasts for edge detections and other non-linear operations.

The second layer filters, while visualized in Fig. 7, are not really able to be viewed in an easy manner. This stems from the fact that there are 50 input channels per filter, and there are 50 filters in the layer as well. The Caffe tutorial suggests that these filters be plotted in a row-column matrix format,

where every row indicates all of the input channels of the filter, and the column represents the depth of the filters.

I do note here that the outputs from the second layer (Fig. 8) are not too different from the first layer. This is a good representation of a network that is not well-learned, because there is not much transition of hierarchy of low-level to high-level features as we move through the levels of the network.

Fig. 4: Original Image for Visualization

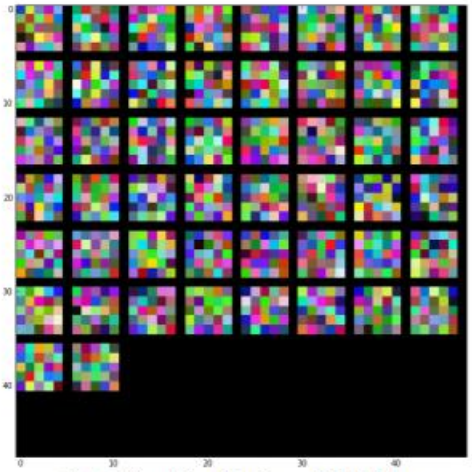

Fig. 5: Visualizing First Layer CONV Filters

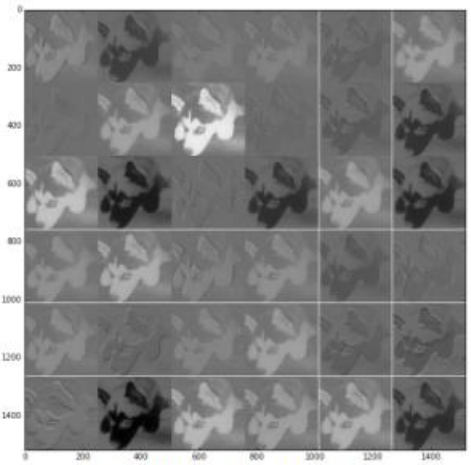

Fig. 6: Output from First CONV Layer

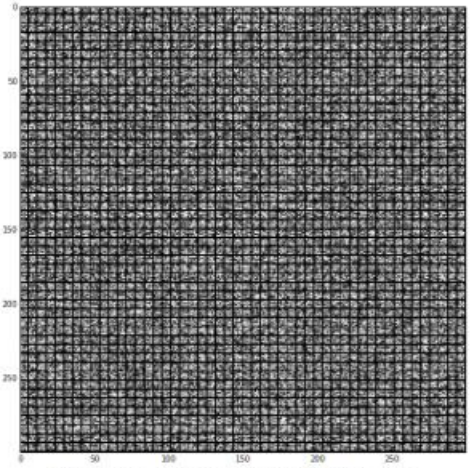

Fig. 7: Visualizing the Second CONV Layer

Dog breed classifier using CNN 3

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 110, 110, 32)      4640
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 53, 53, 64)        18496
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 26, 26, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 24, 24, 128)       73856
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 10, 10, 64)        73792
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 3, 3, 64)          36928
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 1, 1, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               16640
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 133)               34181
=================================================================
Total params: 258,981
Trainable params: 258,981
Non-trainable params: 0
_________________________________________________________________
```

used “rmsprop” optimizer and “loss=categorical_crossentropy” for now. I used accuracy in compiling the model as it is suitable for this kind of problem. Also, analysis can be done on F1 score as well because this dataset is imbalanced and for minority classes F1 score would produce more realistic results.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_3 ( (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_6 (Dense)              (None, 133)               136325
=================================================================
Total params: 2,234,501
Trainable params: 2,234,501
Non-trainable params: 0
_________________________________________________________________
```