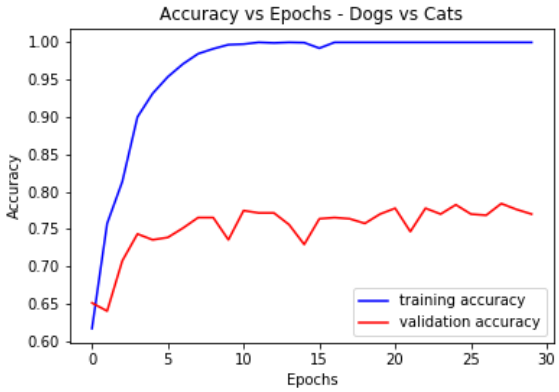
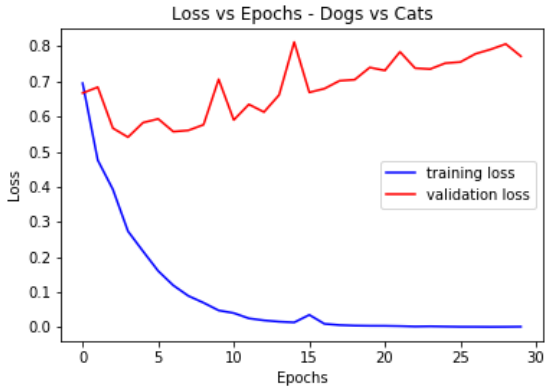
1. Binary classification:

- The total dataset is not actually 4000, but rather 3988, since I had to remove corrupted/unrelated images (such as image of a dog foster logo without any dogs) while running script to reorganize images into appropriate directories to use with image data generator from keras. Using a train/test split and a train/validation split of 80%, there are a total of 2551 images for training set, 639 images for validation set and 798 images for testing set. All datasets have roughly almost the same number of cat and dog images.

a. Base Convolutional Neural net:

- For the convolutional neural network model, I initially started with 1 convolutional block and 1 dense layer. After tuning, the final model was able to achieve stable, reasonably accurate performance and has the following specifications:
  + 4 convolutional layers/ blocks with 32 3x3 filter for the first layer, and 64 3x3 filters for the remaining 3. Batch normalization layers were also added before ReLU activation of a convolutional layers. Thus there are a total of 4 batch normalization layers.
  + 2x2 max pooling
  + 1 hidden dense layer with 512 hidden units
  + Optimizer = Adam with 0.0001 learning rate
- With this model, the performance is as follows:

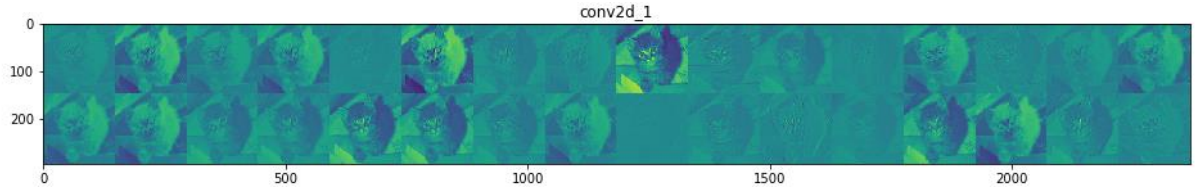


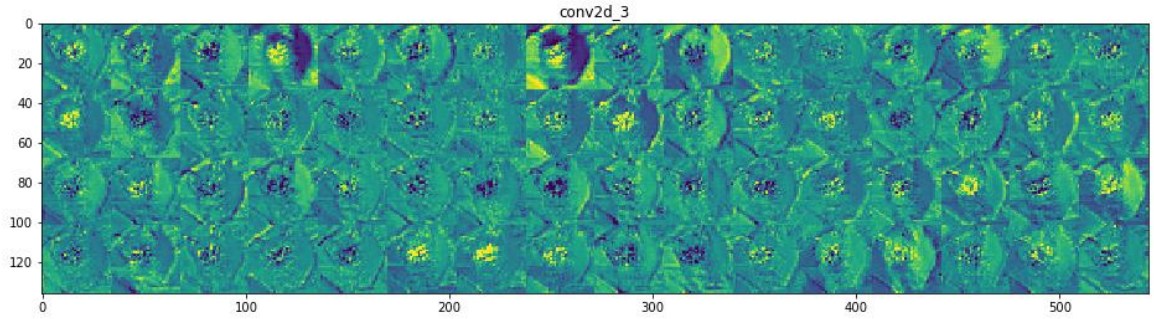




- As can be seen, the model achieved reasonable validation accuracy (around 75%) for such a small dataset, but suffered greatly from overfitting due to insufficient data.

b. Activation Visualization:

- I visualized the activations of all the filters of all 4 conv layers:

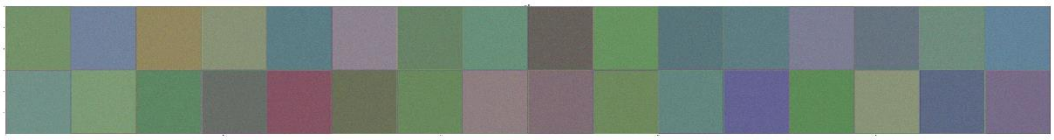


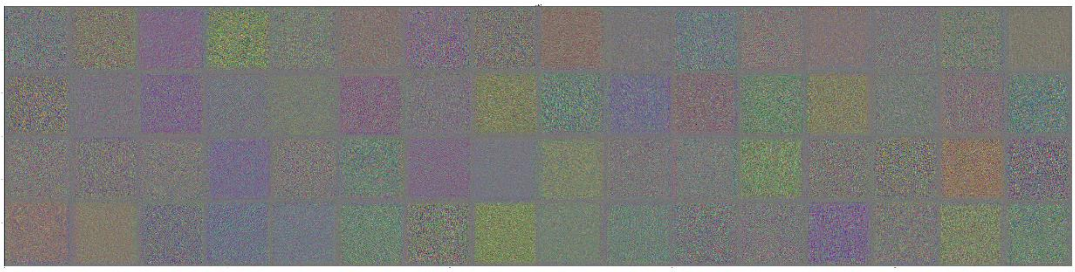



- From the visualization, we reconfirm the fact that convolutional neural net learns higher content features of the image as we go to the deeper layers. For example, conv2d_1 has filters that pick up low level features like edges and blurry circles, but the later layer like conv2_3 abstracts away from low level features, and begins to pick up on the high level content, like parts of the object itself (the cat's head, eyes, mouth like the last row, 6$^{th}$ column picture).

c. Filter Weight Visualization:

- Again I visualized all weights, the following visualizations are from layer conv2d_1 and conv2d_3 (like above):

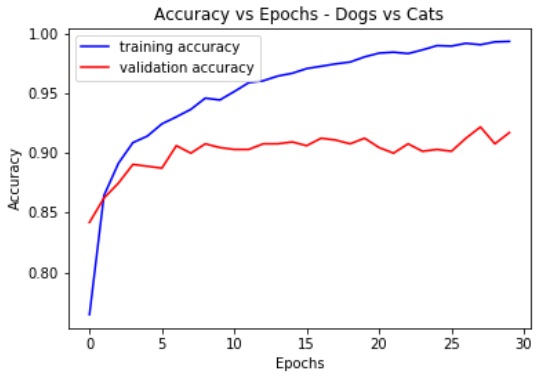


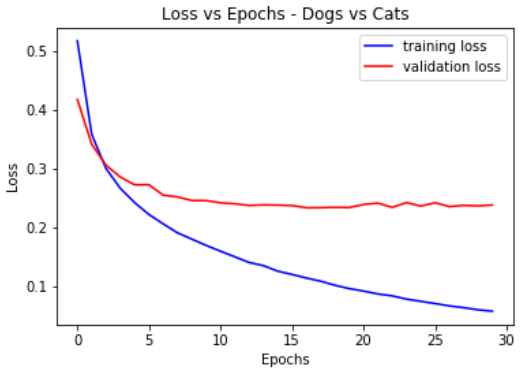


- Theoretically, this visualization should show how the texture information is captured, however to me at least, they are too noisy, there maybe some texture encoded in, but all the noise almost hides the textures away. One possible explanation is that the base model was trained on insufficient amount of data, and thus even though it can capture the content/ features, but it can't capture the texture information yet, hence the noisy visualizations.

d. Train with VGG16 conv base frozen:

- For this model, the fully connected part stays the same as before, that is 1 hidden dense layer with 512 hidden units:

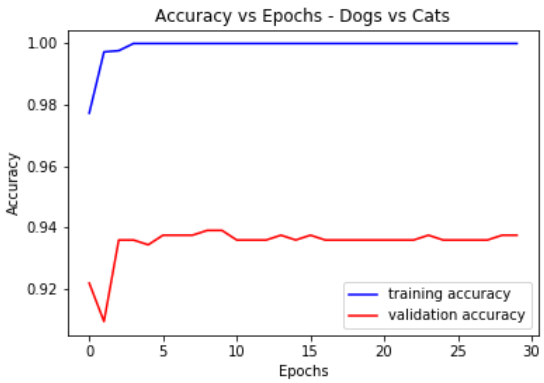


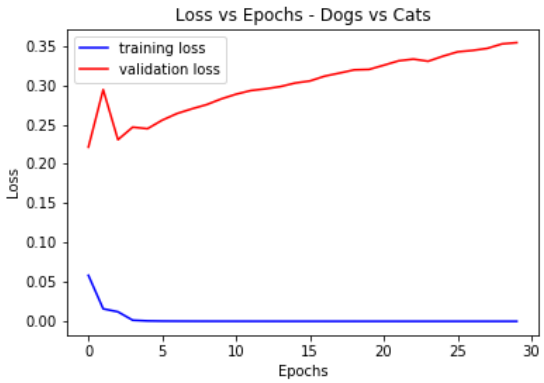




- This model performance improves significantly from the base model, going from 75% validation accuracy up to 90%. This is expected as VGG16 is known to be a very competent network, and has been trained and hundreds of thousands of images. Thus, it should be able to extract relevant features, given that our data are not too different from the data VGG16 was trained on. However, not only that, the overfitting effect also seems not as severe as in the base model. The difference between training accuracy and validation accuracy is only around 9%, while for the base model the difference is nearly 30%. One possible explanation for this is that unlike our model whose convolutional blocks can only learn from ~2500 instances, VGG16 convolutional blocks were trained on a large amount of data, and thus can generalize much better and learn much more relevant image features.

e. Fine tune conv base:

- After finished training in the previous step, I fine tune VGG16 conv base by unfreezing the blocks and continued to train so that the convolutional base can fit to the cats and dogs data better.

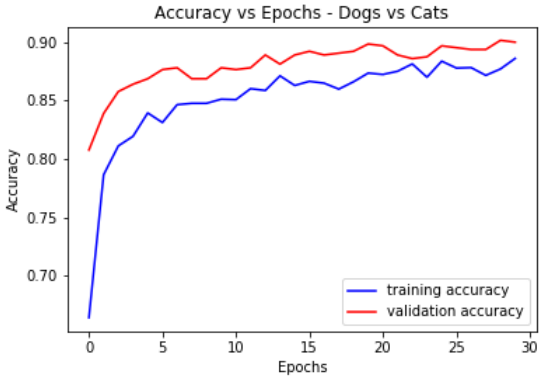


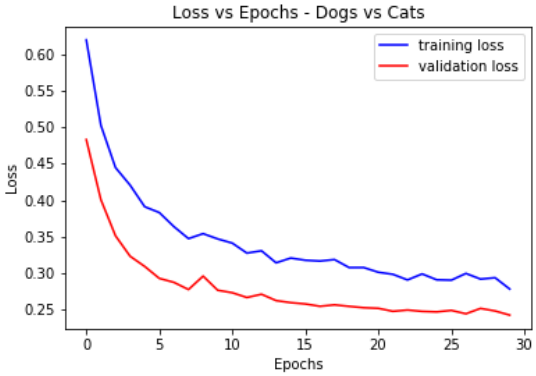




- As expected, the performance improved by a little bit, from approximately 91% to 94%. This is because VGG16 conv base was exposed to the specific dataset for the current classification task. Thus the conv base is able to adapt more to the dataset, and extract features that are more relevant to the cats and dogs dataset. The overfitting effect is roughly the same.

f. Data augmentation and frozen conv base:

- I performed data augmentation on the training dataset with the following specifications:
  + Normalization: 1/255
  + Rotation range: 40
  + Width and height shift range = 0.2
  + Shear range = 0.2
  + Zoom range = 0.2
  + Allow horizontal flip
  + Fill: nearest



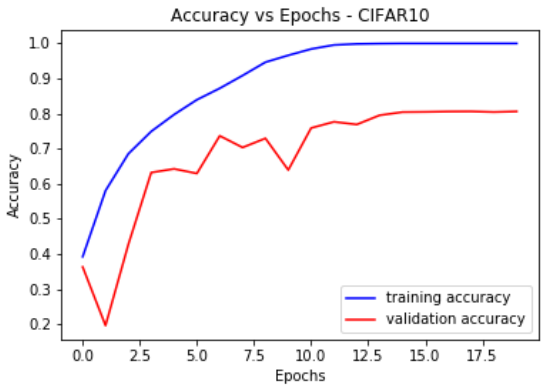

- We can see that data augmentation is indeed a powerful method to prevent overfitting. By creating synthetic images using different transformation method, the model is exposed to a wider variety of data and starts to become more invariant to shifts in spatial positions of the object to be classified. Thus, the model is able to generalize much better. Moreover, data augmentation is able to prevent overfitting without having to sacrifice too much performance as the validation and training accuracy still hovers around 90% mark, compared to the previous more heavily overfitted model with a 91-93% validation accuracy.

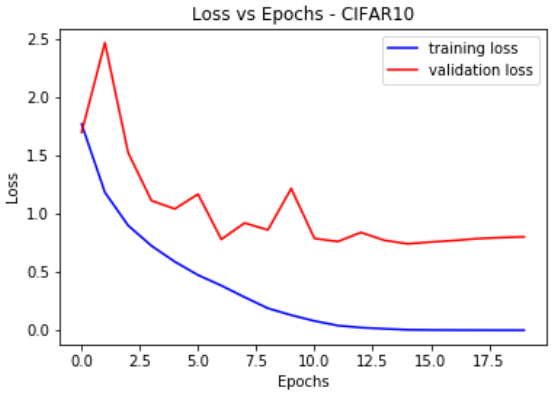
2. Multiclass classification:

a. Base Convolutional Neural net:

- The base model specifications are as follows:
  + 4 convolutional layers/ blocks with 128 3x3 filter for the first layer, and 256 3x3 filters for the remaining 3 with paddings. Batch normalization layers were also added before ReLU activation of the first and last convolutional layers. Thus, there are a total of 2 batch normalization layers. I tried with every layer having batch normalization but did not see any difference in performance, thus I chose only 2 batch norm layers so that I can add inception and residual blocks later without having to worry about resource exhaustion.
  + 2x2 max pooling
  + Dense layer only contains the flattening layer and output layer with 10 units (no hidden layers)

+ Optimizer = Adam with 0.001 learning rate

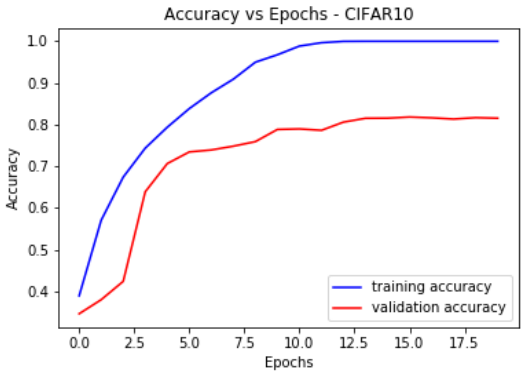


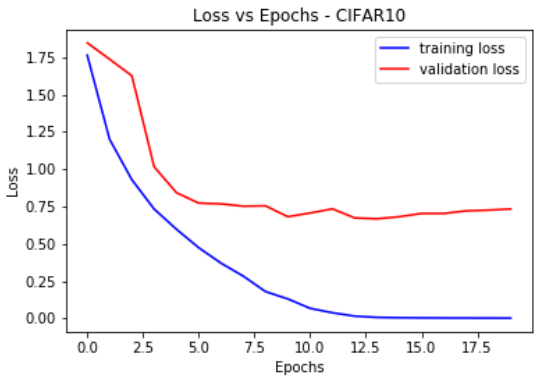




- The model achieved reasonable validation accuracy of around 80%, however overfitted badly.

b. Addition of inception blocks:

- Following the GoogLeNet architecture, I added one inception block before the 4 conv layers of the base model. All the convolutional layers in the inception block have 32 filters with paddings. The inception block also has 1 3x3 max pooling layer. The inception block is implemented with dimensionality reduction, so it has 4 1x1 conv layers, 1 3x3 and 5x5 conv layer. It also has a concatenation layer then feeds the output to the 4 conv layers of the base model



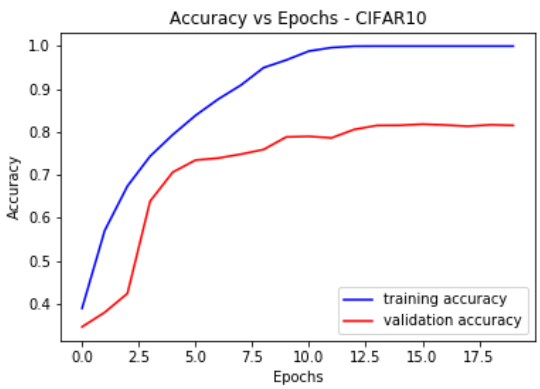



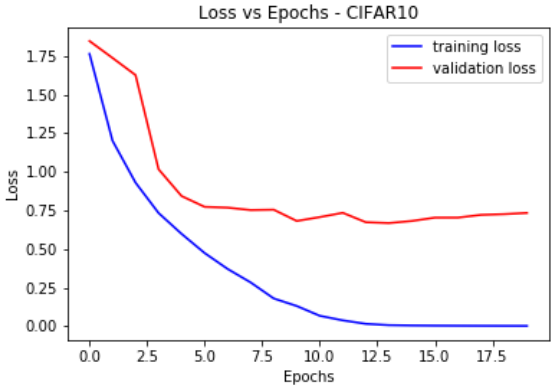

- The addition of inception block actually did not improve the model's performance and the overfitting effect was as severe. My speculation is that 1 inception block is not enough to make any drastic improvement for the model

c. Addition of residual block:

- After trying around with where to add the residual block, I decided to add 1 residual block before the 4 conv layers of the base model, because the performance did not differ much if I added the block elsewhere, and it's easier to compare with inception block as I also added 1

inception block before the 4 conv layers in the previous experiments. The residual block contains a 3x3 conv layer with padding and 128 filters, as well as a 3x3 conv layer with padding and only 3 filters, so that the dimensions are compatible when we add the output with the original input.




- Similar to the previous experiment, the addition of a residual block did not really improve the model's performance. The overfitting effect remained the same. Again, my speculation for this is that 1 residual block is not enough to drastically change the model performance