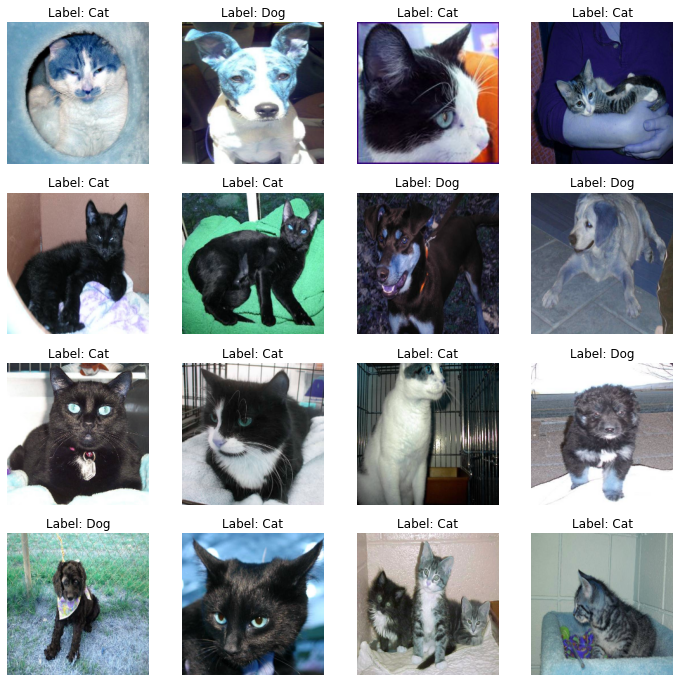
**EDA**

1. Training and test set .zips are unzipped to their respective folder.
2. Create functions to one-hot encode for dogs as 1 and cats as 0.
3. Process data with opencv and resize all images to 224 x 224. Each image is then labeled with their class.
4. I also write a function to visually examine if each picture is labeled correctly

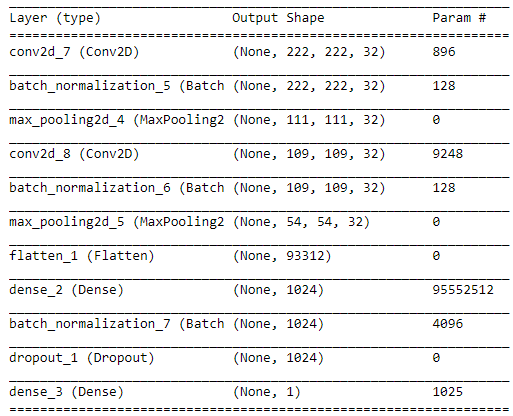


1. I then convert image and labels into a dataframe.
2. Train-test-split is used for a 80-20 split to create a validation set.
3. ImageDataGenerator is used to process the images.
   1. On the training set, several parameters such as rotation\_range and width\_shift\_range are used to introduce randomness into the figures to avoid overfitting.
   2. I used preprocess\_input imported from ResNet50 as the preprocessing function

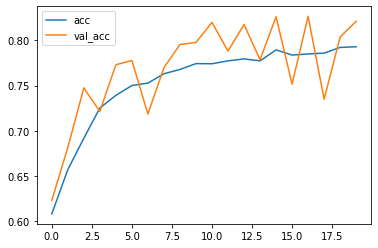
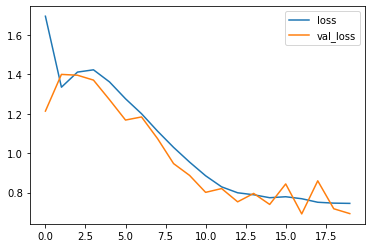
**Model Development**

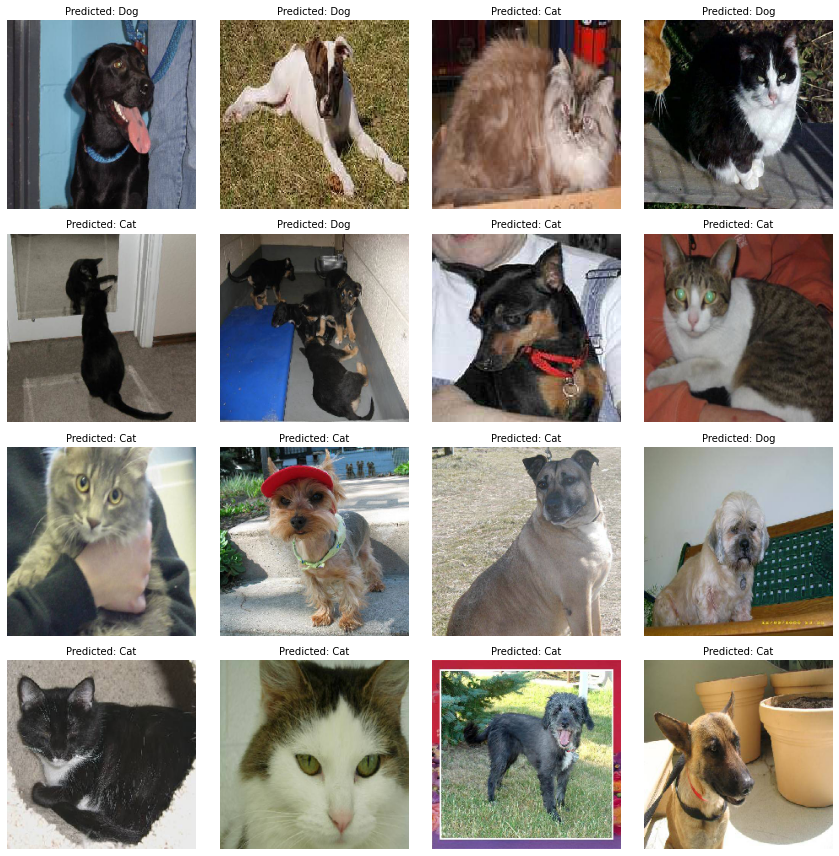
**Model 1:**

The first model I build is a simple 2 layer CNN. Both layers have 32 neurons and use the ReLU function. After each layer I use batch normalization and a 2x2 pooling layer (MaxPool2D). Then the image is flattened for a fully connected layer with L2 regularization, a 50% dropout, then the output layer with a sigmoid activation.



Model was compiled with the RMSprop optimizer at 0.001 learning rate, using the binary crossentropy loss function and accuracy as the metric. The model was then trained for 20 epochs, with weights saved in a hdf5 file.





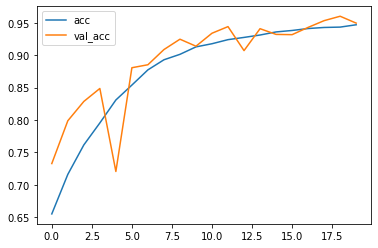
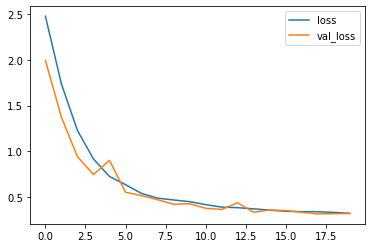
This model barely reaches above 80% accuracy, and 6 out of 16 images in the sample above are misclassified.

**Model 2:**

I increase the number of layers to 8, add padding with zeros, and use the Adam optimizer instead. Accuracy iterated to above 92%, loss did not reach an asymptote within 20 epochs. Only 1 out of 16 samples was misclassified. This was submitted to Kaggle for a baseline result.

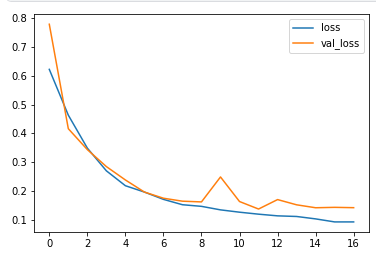
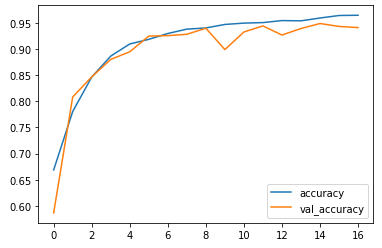
**Model 3:**

Still with 8 layers, learning rate dropped to 0.0001, and have more neurons for each subsequent layer. Results improved significantly.



**Model 4:**

I use Model 3 and reduce the number of neurons at the deeper layers and use 50 epochs instead. I include an early stopping function when the val\_loss stops dropping for more than 3 epochs.

**Model 5:**

Then I also try ResNet50’s pretrained layers with only 10 epochs (since each epoch took around 30 minutes to complete). I use max pooling and imagenet weights. Then I did another 10 epochs to slightly improve the score.

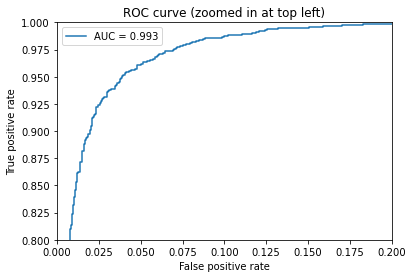
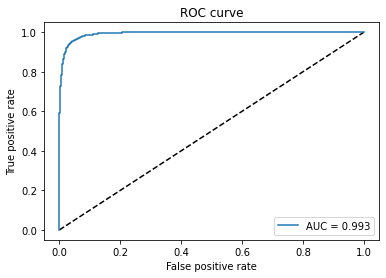
**Model Evaluation**

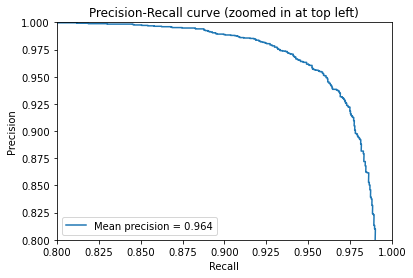
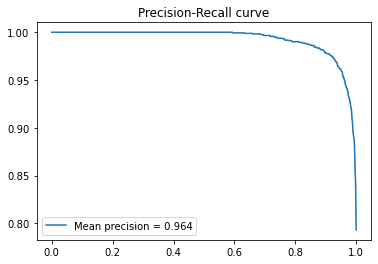
For each model, I plot training loss with validation loss and training accuracy with validation accuracy to see if models are underfit/overfit. ROC-AUC curves and precision-recall graphs are used to evaluate how well the model is done. I also sample 16 images from the test set and visually see if models are working. The submission file to Kaggle needs to contain the probability that each image is a dog, and not a 1/0 encode. I found that capping my predictions between 0.005 and 0.995 lowered my score by 50%, possibly because of how the Kaggle score is calculated (log-loss).

**Kaggle Results**

|  |  |  |
| --- | --- | --- |
| Model | Kaggle | Summary |
| 2 | 0.16709 | 8 Conv2D layers with 32 filters, 1 Dense layer with 1024 units, Adam with 0.001 learning rate, 20 epochs |
| 3 | 0.15725 | 8 Conv2D layers with 32, 64, and 128 filters, 1 Dense layer with 256 units, Adam with 0.0001 learning rate, 20 epochs |
| 4 | 0.15133 | 14 Conv2D layers with 32, 64, 128, and 256 filters, 1 Dense layer with 1024 units, Adam with 0.001 learning rate, 40 epochs |
| 5 | 0.06924 | ResNet50 with imagenet weights, Adam, 0.0001 learning rate, 10 epochs |
| 5+ | 0.06876 | Same as above with 20 epochs |

I used the refined non-ResNet50 model for the following precision-recall and AUC curves.





**Insights**

As expected, the best model comes from the pretrained layers from ResNet50. In my own models, more complex models tend to give better results, but also takes exponentially more time. I can calculate to get a probability. For my best model, the 0.15133 score can roughly be interpreted as an 86% chance that the model predicts the correct answer. For the ResNet50 model, the Kaggle score corresponds to a 93.3% probability. These probabilities are not directly comparable with the accuracy metric.

For my own models, I would also have liked the computational resources or time to run hundreds of epochs to be able to better evaluate them. The following graphs are what happens if I run my second model for 200 epochs (on a Kaggle kernel), where it shows that the model eventually overfits significantly.

