Assignment 1 Report

Lawrence Abdulnour 20019894
Alexandre Marcil C9464
Louis-François Préville-Ratelle 708048

IFT 6135
16/02/2019

Question 1

We refer to the corresponding jupyter notebook for this question.

Question 2

We refer to the corresponding jupyter notebook for this question.

Question 3

3.1)

Hereby is a brief overview of our model. For more details, please refer to the jupyter notebook.

We used a CNN with seven convolutional and two fully connected layers. 3 blocks of 2 convolutional layers with maxpooling were used, followed by an another convolution and two fully connected layers, for a total of 9 layers. We used a kernel size of 5 with a padding of 2 to keep the input dimensions constant inside blocks. Increasing numbers of kernels were used to generate up to 128 feature maps. The 3 Maxpooling steps resulted in a reduced dimension of 8x8x128 for the last convolutional layer. Two linear fully connected layers with 8192 and 200 units each then enabled to get an output for both catagories, cat or dog. In total, this CNN had 3,946,288 paramaters, which accounted for the long training time (about 3h for 100 epochs).

3.2)

Crossentropy loss was used to train this network. 100 epochs were used and the learning rate was reduced at every 25 epochs to enable a fine-tuning of our network. This accounts for the bumps seen in our training curves (figure 1 and 2), mostly visible at epoch 25 (where learning rate was decreased from 0.025 to 0.010) and epoch 50 (0.010 to 0.0025). The last 25 epochs (learning rate 0.001) did not improve our best accuracy of 94.75% on our validation set which was reached at epoch 73. This result is comparable to the accuracy obtained on the kaggle test set (95.2% on the public leaderboard and 93.7% on the private leaderboard).

To achieve this result, a lot of data augmentation was used. We used the transforms functions built-in PyTorch, i.e. ColorJitter, RandomHorizontalFlip, RandomAffine (shear), RandomRotation and RandomResizedCrop. This prevented our network from overfitting too quickly to our training data, which consisted of only 17598 images. Still, we can see on our training curves that our network starts overfitting around epoch 50, with the training accuracy still improving while the validation accuracy barely changes.

Regularization, such as Dropout, would have helped in preventing this kind of overfitting. BatchNorm and better optimizers (Adam, SGD with momentum) would have helped in training this network much more efficiently. More data is always a good idea, so data augmentation could also be improved using specialized library for this purpose, such as ImgAug (https://github.com/aleju/imgaug) or Augmentor (https://github.com/mdbloice/Augmentor).

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

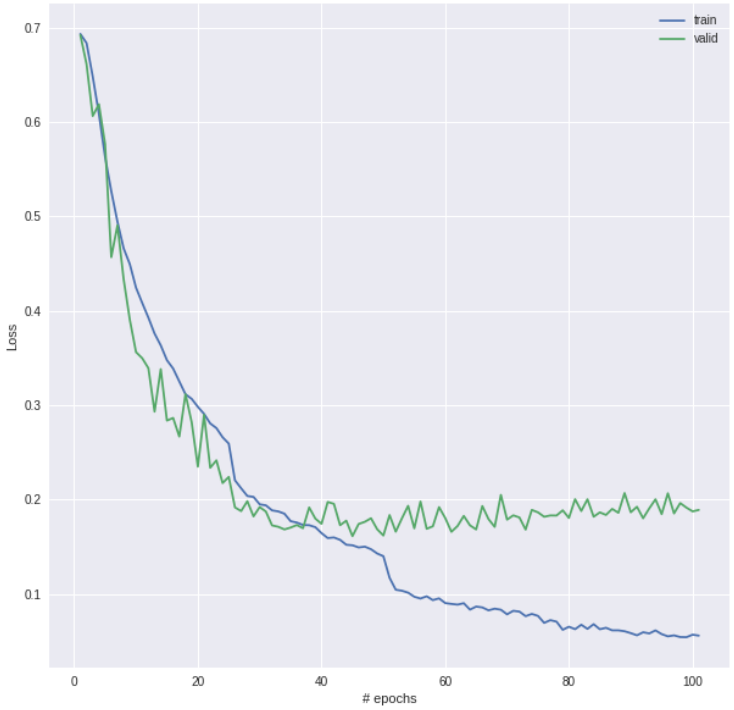

Figure 1: Training curves showing loss decreasing with epochs

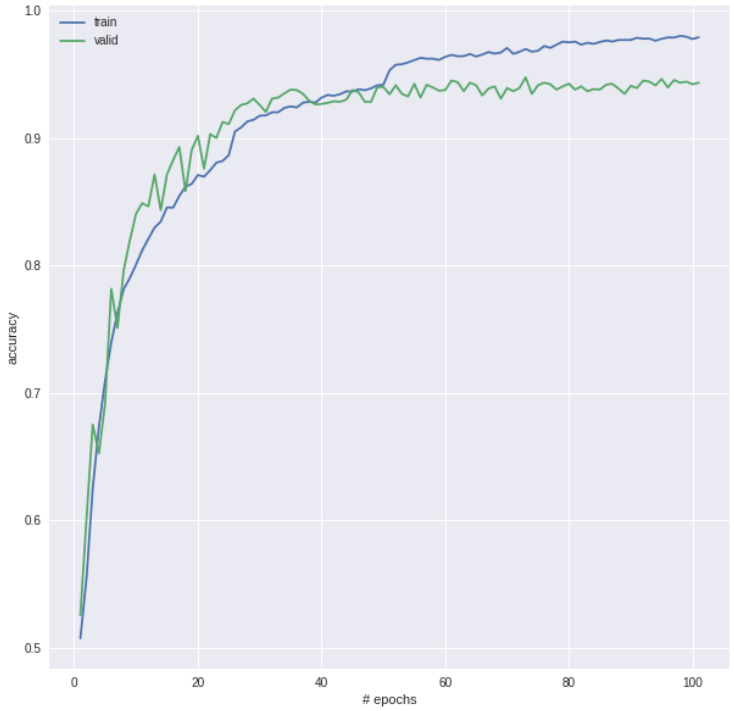

Figure 2: Increase in prediction accuracy with number of training epochs

3.3)

We did not test that many hyperparameters since we only did two submissions on the kaggle website. The first version of our CNN was smaller, having 6 convolutional layers instead of 7, and using kernel sizes of 3 instead of 5. The number of paramaters for that network was about twice as less as our final version (2,175,002 vs 3,946,288). The main difference is that we did not used as much data augmentation while training our first CNN as we did in our final version. Tweaking the learning rate did help us to train our network somewhat faster, and also to fine-tune it by using a small learning rate for the last epochs. Overall, this enabled us to increase our accuracy from 92.6% to 94.75% on our validation set, and from 91.6% to 95.2% on the test set (public leaderboard). We did not pursue to improve our CNN after this result.

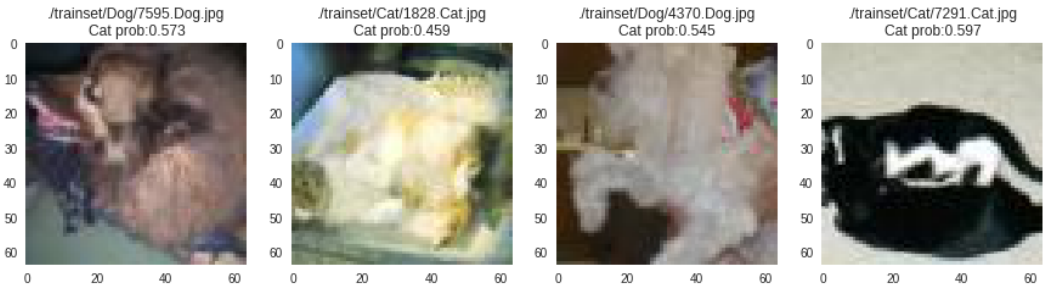

Figure 3: validation set images giving ambiguous predictions

Figure 3 shows images from our validation set that our classifier predicts ambiguously, i.e. around 50% for both classes. The second image (1828) is a cat but gets a cat probability of 45.9%, therefore getting missclassified as a dog. The third image (4370) is a dog but gets a cat probability of 54.5% and gets missclassified as a cat. Given the low resolution of these images, even a human would have a hard time classifying these images.

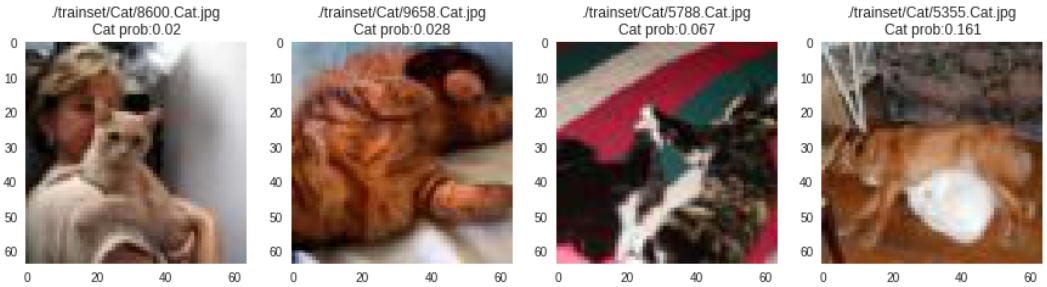

Figure 4: Images of cats getting missclassified with high confidence

Figure 4 shows images of cats that are missclassified as dogs with high confidence (the cat probability is very low). Analysis of theses images show that these photos are taken from odd angles and that they can be hard to classifiy even for a human (picture 2 and 4 in particular).

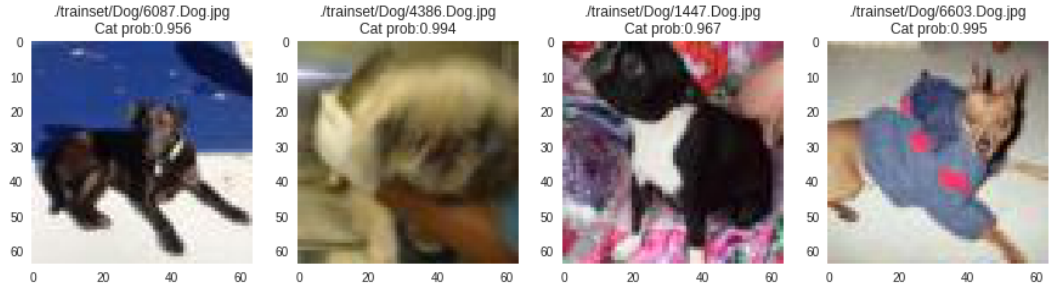

Figure 5: Images of dogs getting missclassified with high confidence

Figure 5 shows images of dogs that are missclassified as cats with high confidence (the cat probability is very high). Dog features are hard to distinguish in these pictures and can explain the missclassifications. Better resolution images would certainly lead to better predictions. More examples of these missclassifications can be seen on our jupyter notebook.

Overall, this qualitative analysis of missclassification reveals that most of these errors are due to poor image quality. Images that cannot be classified by a human maybe should not be used in the training set. This is revealing the importance of having clean datasets to do such machine learning classifiers (garbage in, garbage out!). Or maybe it is simply that some small dogs do resembles cats!

References

Jupyter notebooks for Assignment 1 can be found here:

https://github.com/Zigzagzen7/IFT6135-Assignment-1