IFT6135-H2019 - Prof: Aaron Courville
Assignment 1, Practice - Kaggle

Rim Assouel	**Gunshi Gupta**	**Vikram Voleti**
20120367	20137932	20091845

February 2019

Cats vs Dogs

For all models, input images were scaled to between -1 and 1. The batch size was 128, and SGD was used to optimize training with a learning rate of 0.1. Random 10% of the training set was used for validation.

We experimented with quite a few configurations of architecture and hyperparameters. For each experiment, we draw a figure of the architecture used. All architectures have a softmax at the end (not present in the figures). We also report the loss and accuracy curves for training and validation data in each experiment.

In the following, we experimented with many concepts including :

— Convolutional layers, ReLU, MaxPool, Fully Connected layers
— Data augmentation
— Pre-training on a large dataset (not using a pre-trained model) and finetuning on this one
— ReduceLROnPlateau (self-implemented)
— Skip connections

In addition, for our best model we visualized :

— the first convolutional layer's weights at the start and end of training
— failure cases
— areas of the image the model focuses on for classification

We also mention some possible improvements.

We shall look at each of these points one by one. The code used is available at : GitHub

1. **Overfit**

   We trained a model on the images of the dataset directly.

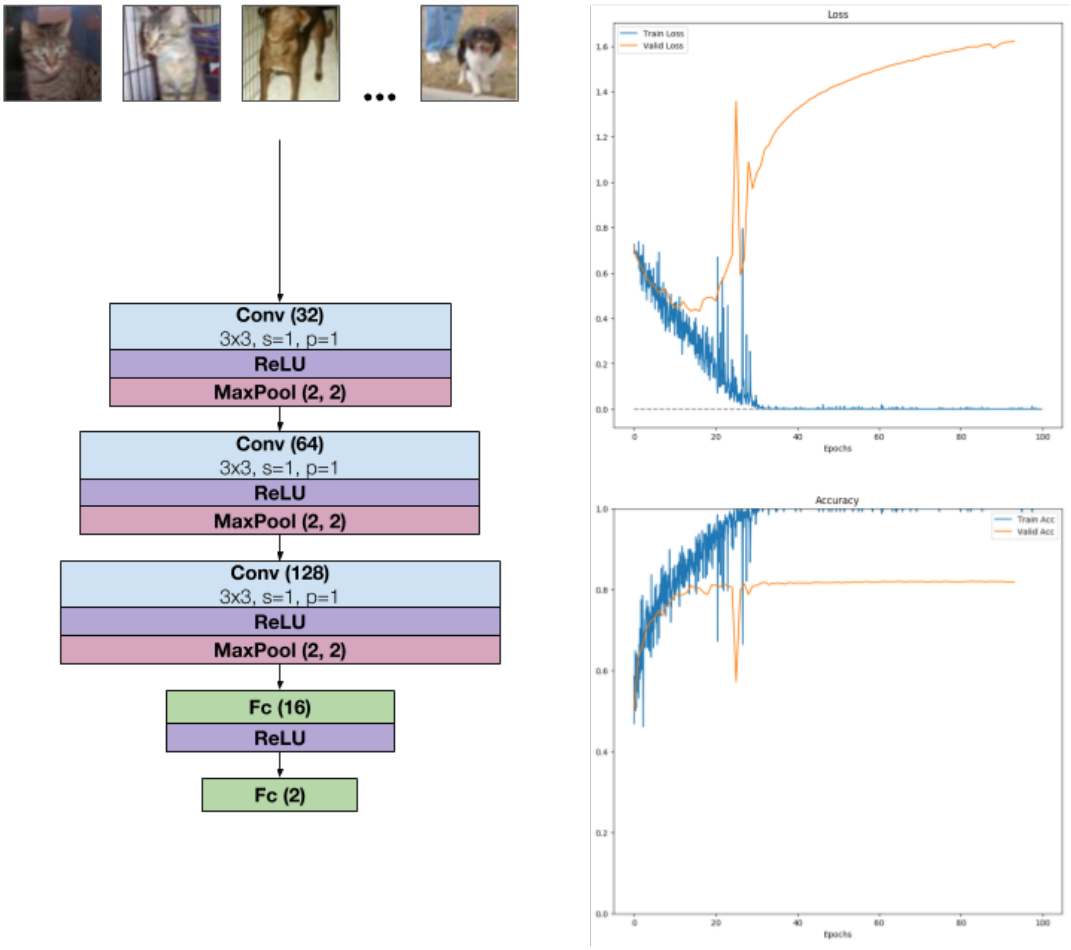

Figure 1 – Overfit

We see that this model overfits the data quite easily.

Instead of reducing the capacity of the model, we wanted to improve its training. So we added data augmentation in the next experiment.

2. Data Augmentation

(a) We then added a lot of data augmentation to let the model learn on a much more diverse set of images.

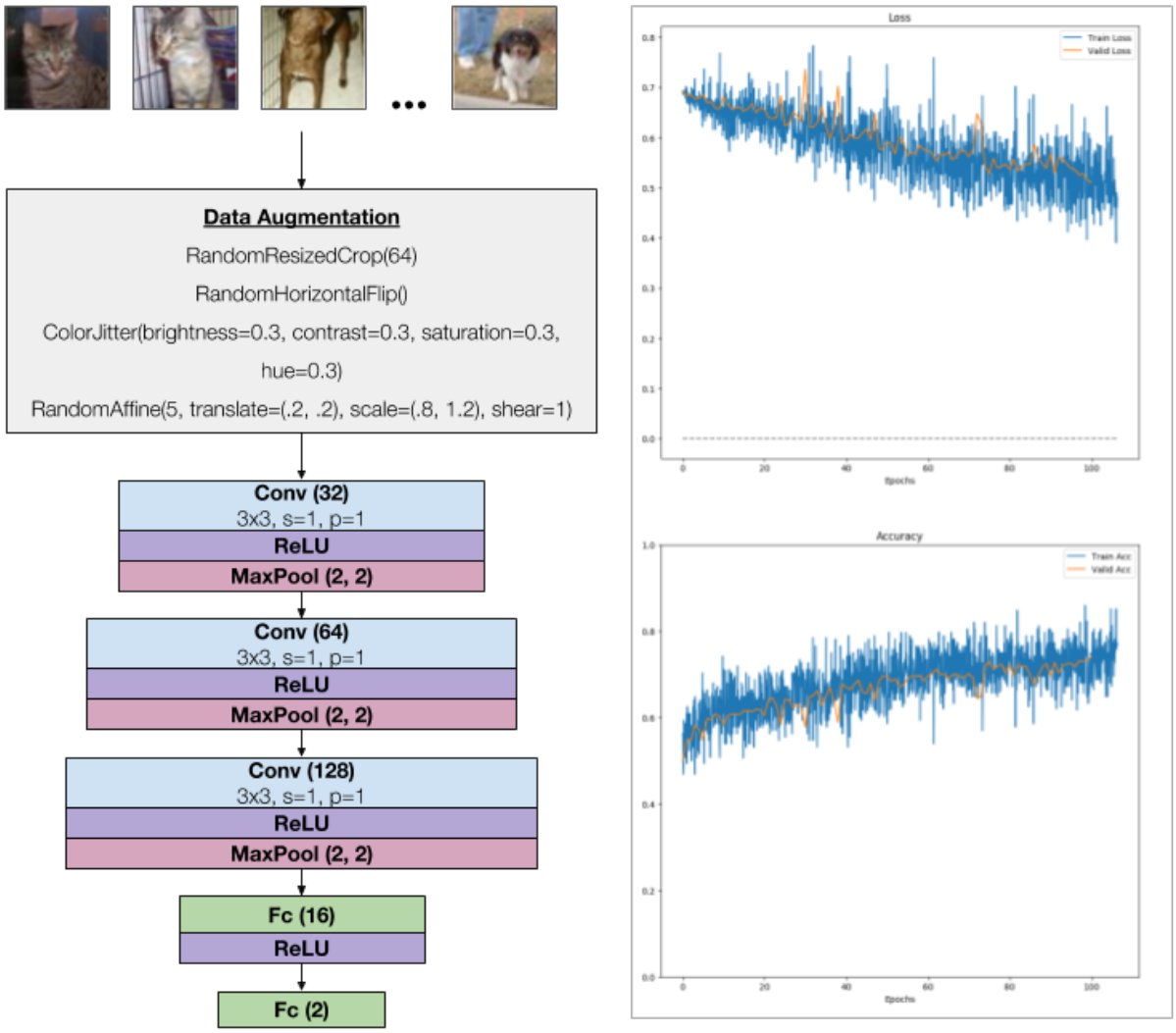

Figure 2 – With Data Augmentation

We see that data augmentation makes the network learn the right things required for this classification, as the validation loss follows the training loss.

We continue this training.

(b) **Continuing with data augmentation**

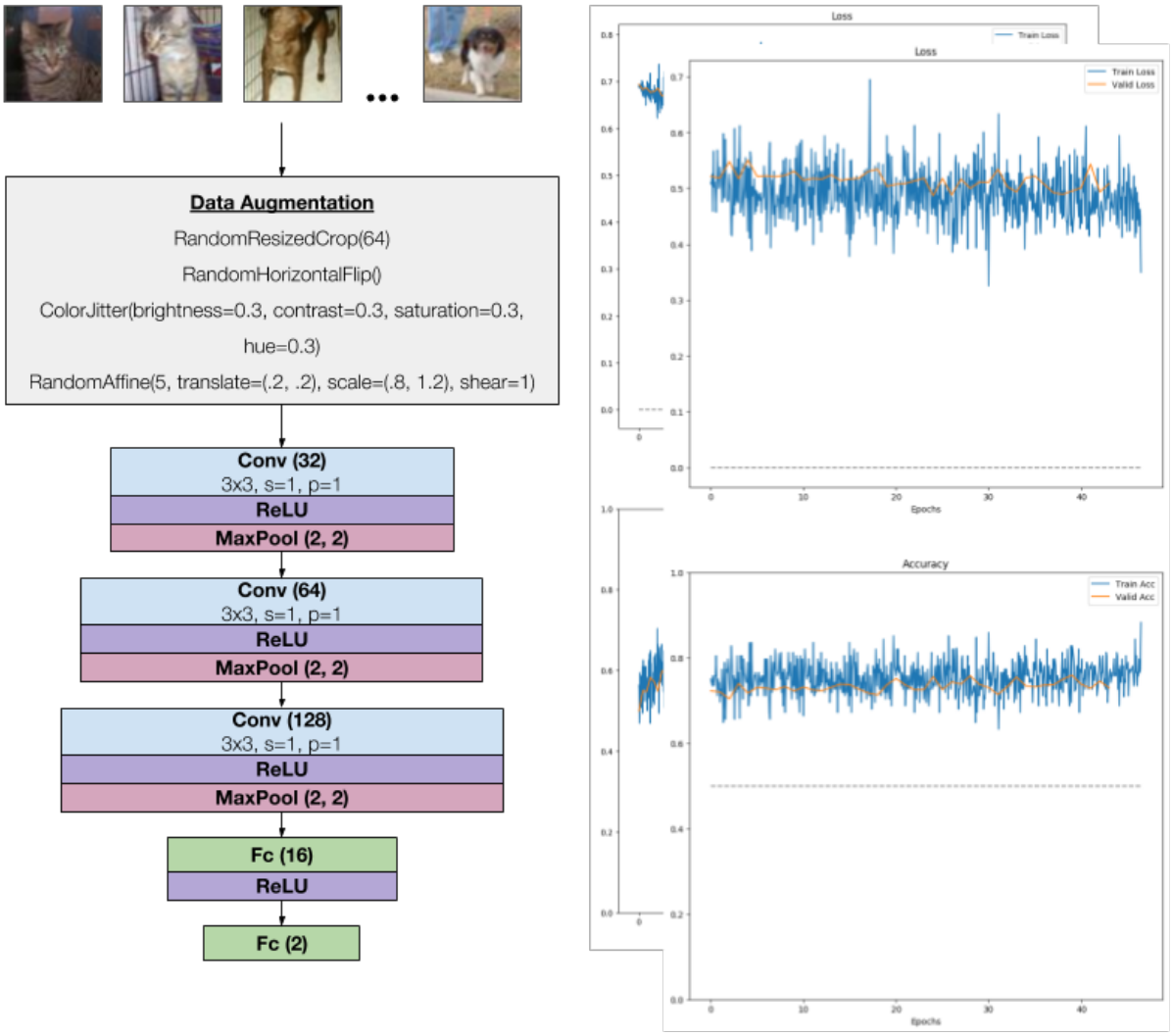

Figure 3 – Continuing with Data Augmentation

We see that this time the network hardly learns anything. This might be because at this stage the network needs more capacity to learn the complicated data that it receives with augmentation.

To tackle this problem, we decrease the amount of augmentation made on the data.

(c) **Continuing by decreasing data augmentation**

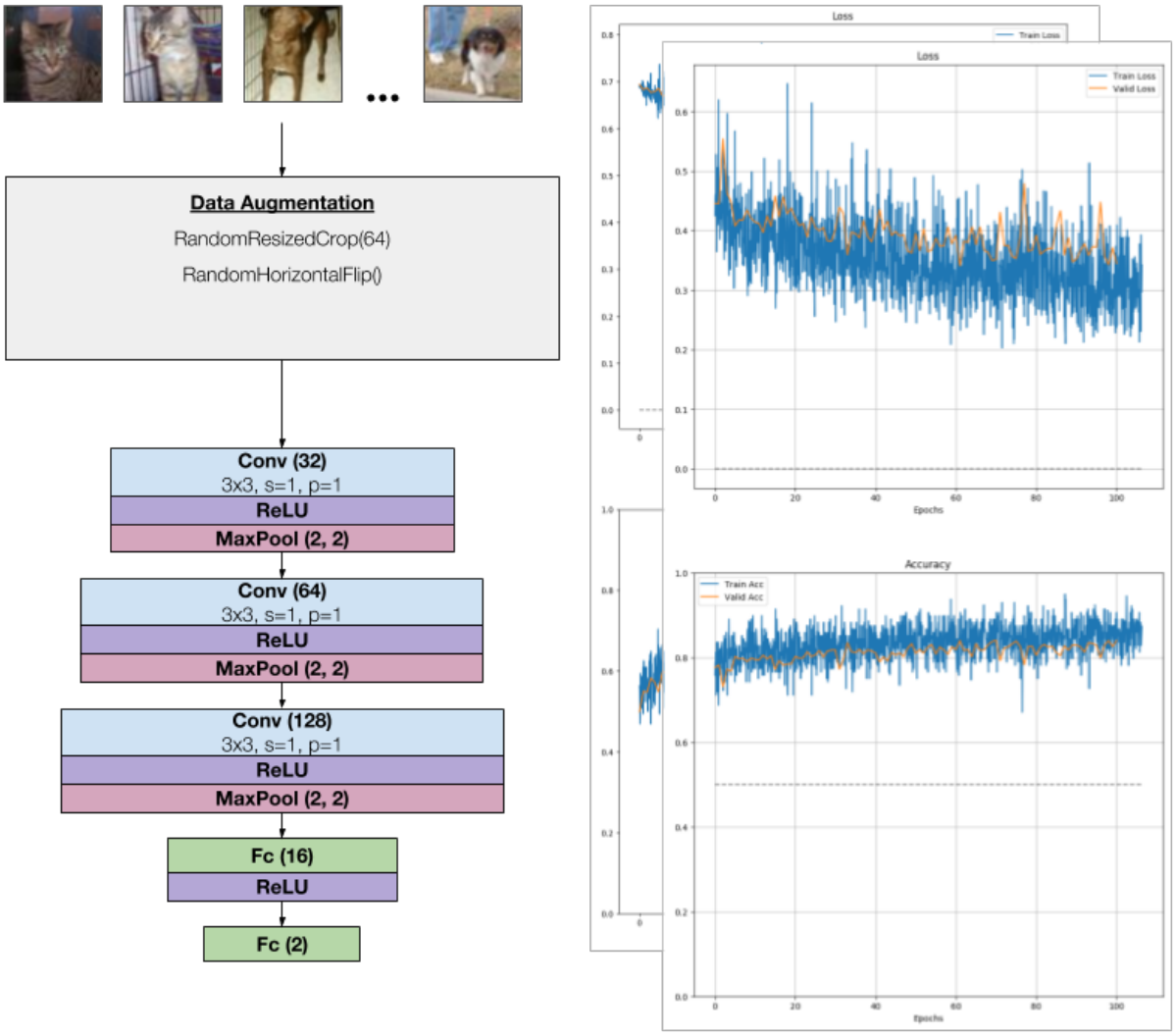

Figure 4 – Less Data Augmentation

We see that the network continues to learn, albeit very slowly.
We then try decreasing the learning rate.

(d) **Continuing by decreasing learning rate**

We decrease the learning rate from 0.1 to 0.05.

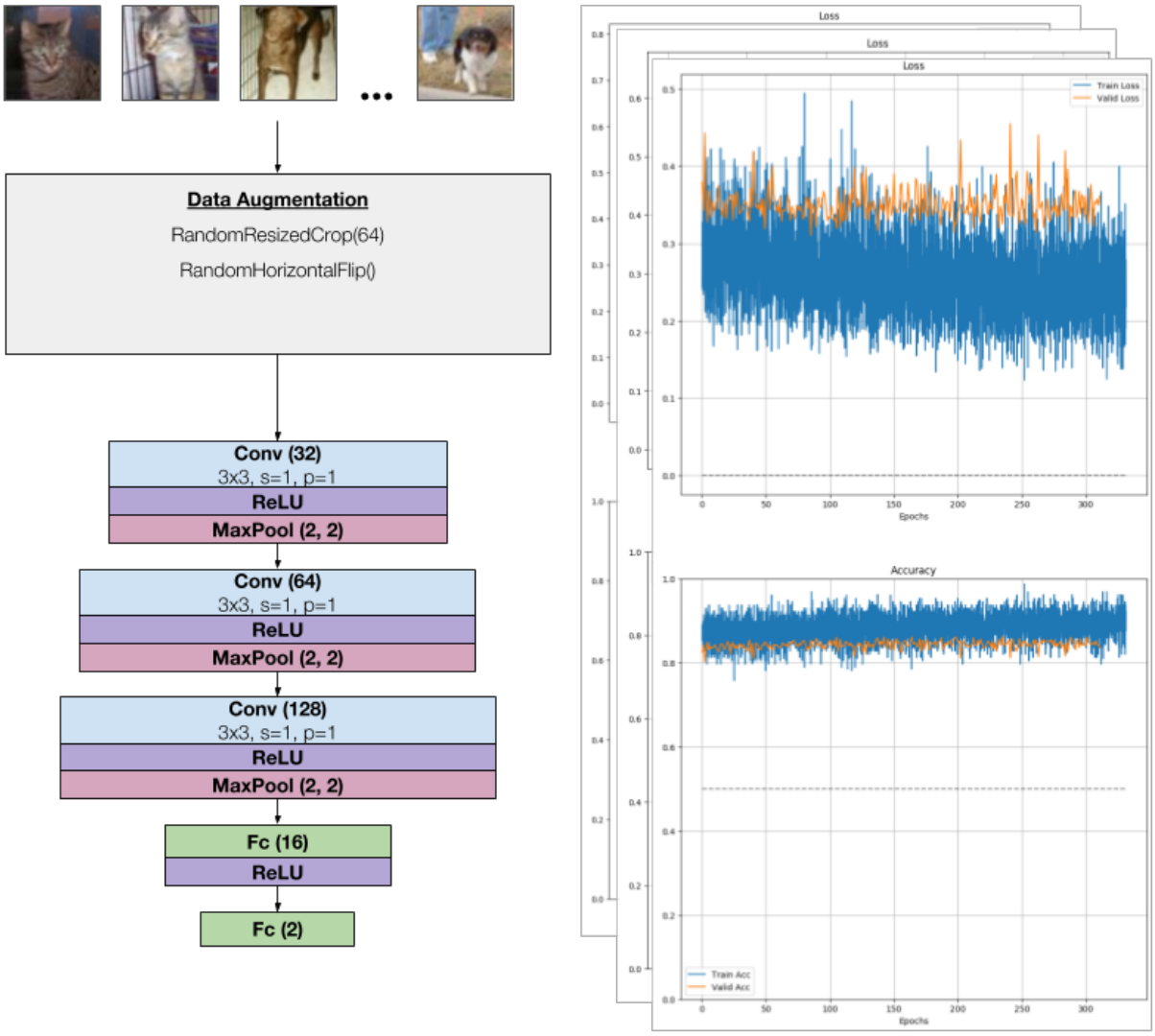

Figure 5 – Half LR

We see that this does not seem to improve the model by much.

Nevertheless, we upload this to check the score on the public leaderboard. It turns out to be **0.86274**.

3. **Pre-train on TinyImageNet**

   (a) We use a bigger model with 1,100,136 parameters, and pre-train it on the TinyImageNet dataset. This is a smaller version of ImageNet, with 200 classes and 500 images per class.

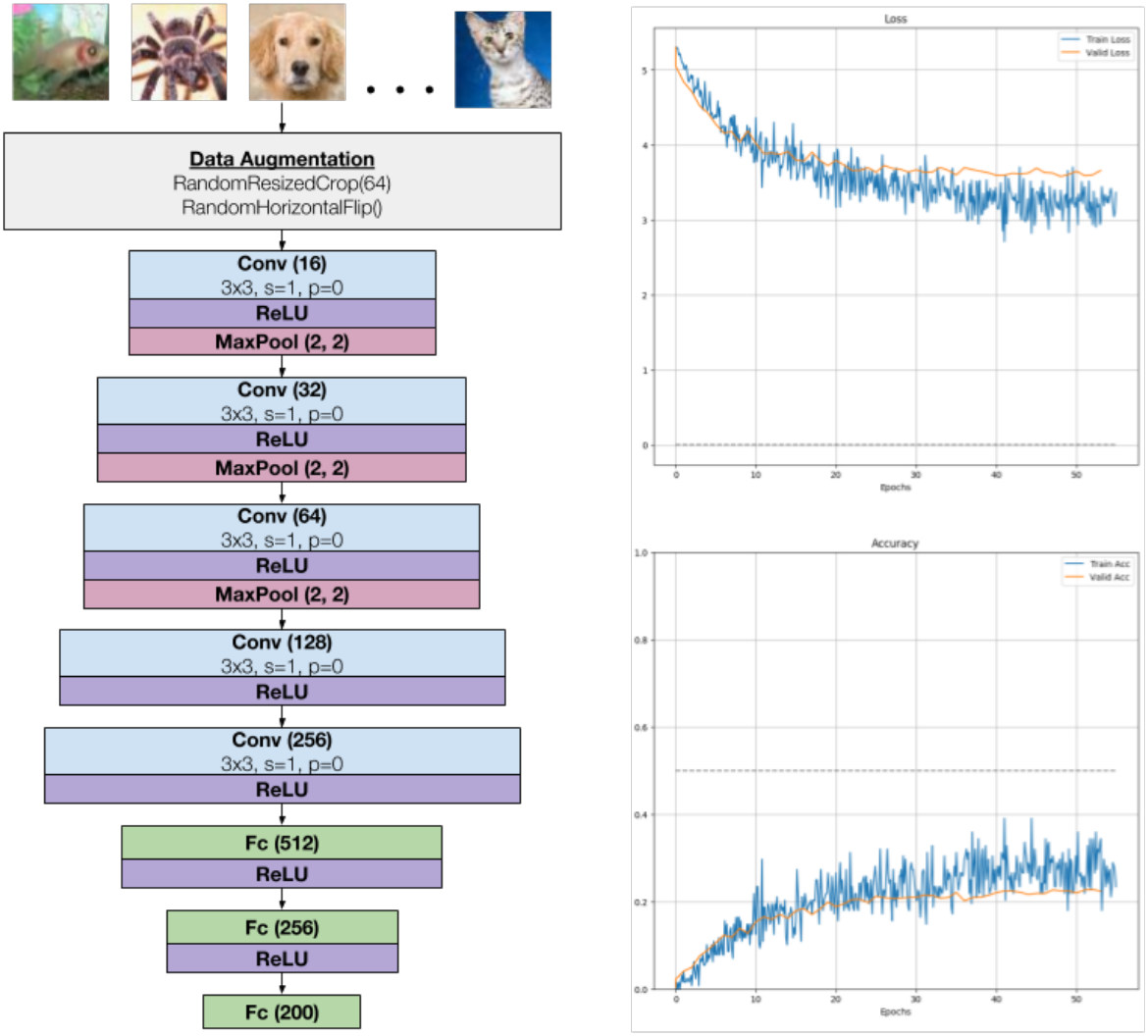

Figure 6 – Pretraining on TinyImageNet

Even though this didn't perform very well on TinyImageNet, it must have learned some features that were useful. Starting from there, we finetuned this network to our Cats and Dogs dataset.

(b) **Finetune on Cats and Dogs**

We try to finetune on Cats and Dogs by freezing the pre-trained weights, and only training on the last layer. This model has only 514 trainable parameters.

Figure 7 – Pretraining on TinyImageNet

We see that the network is not able to learn much. This might be because the features learned from pre-training are not enough to directly differentiate the two classes.

But it might also be because the learning rate is too high. We test this by decreasing the learning rate and seeing what happens.

(c) **Finetune on Cats and Dogs - freeze, half LR**

We reduce the learning rate to 0.05 and finetune.

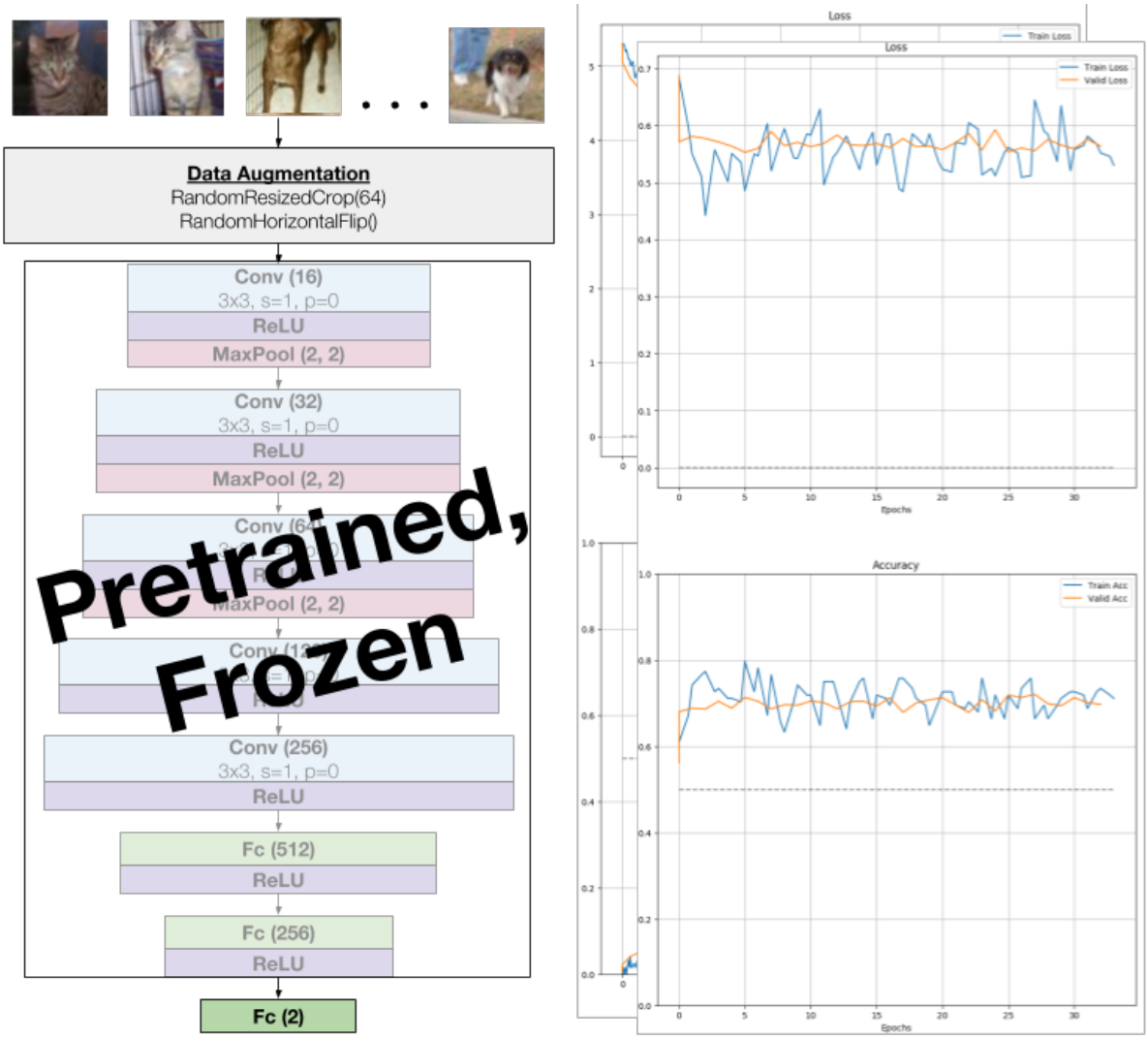

Figure 8 – Finetuning - freeze pretrained network, half LR

We see that the network is still not able to learn much.

To be sure, we introduce early stopping with reduction of LR - ReduceLROnPlateau, and finetune it again.

(d) **Finetune on Cats and Dogs - freeze, ReduceLROnPlateau**

ReduceLROnPlateau : Whenever the validation loss of the network does not decrease for **5 (patience)** epochs, the learning rate is halved.

In the graphs below, the dashed vertical lines are where the LR was halved.

Figure 9 – Finetuning - freeze pretrained network, ReduceLROnPlateau

We see that the network is still not able to learn much.

So we unfroze the pre-trained layers and finetuned over the full network.

(e) **Finetune on Cats and Dogs - unfreeze, ReduceLROnPlateau**

We finetune the full network after pretraining, without freezing any layer. As before, the vertical dashed lines represent the epochs at which the learning rate was halved when the validation loss did not decrease for 5 epochs.

Figure 10 – Finetuning - unfreeze pretrained network, ReduceLROnPlateau

We see that the network does learn more than 80%, but it stays at that level even after halving the learning rate multiple times.

This scored **0.82312** in the public leaderboard.

4. **Bigger model**

(a) The number of parameters in the previous model was 1,100,650. Since the training accuracy also isn't increasing above ≈85%, we increased the parameters to eight times - 8,881,218. We trained this with the same batch size (128), using SGD with a learning rate of 0.1.

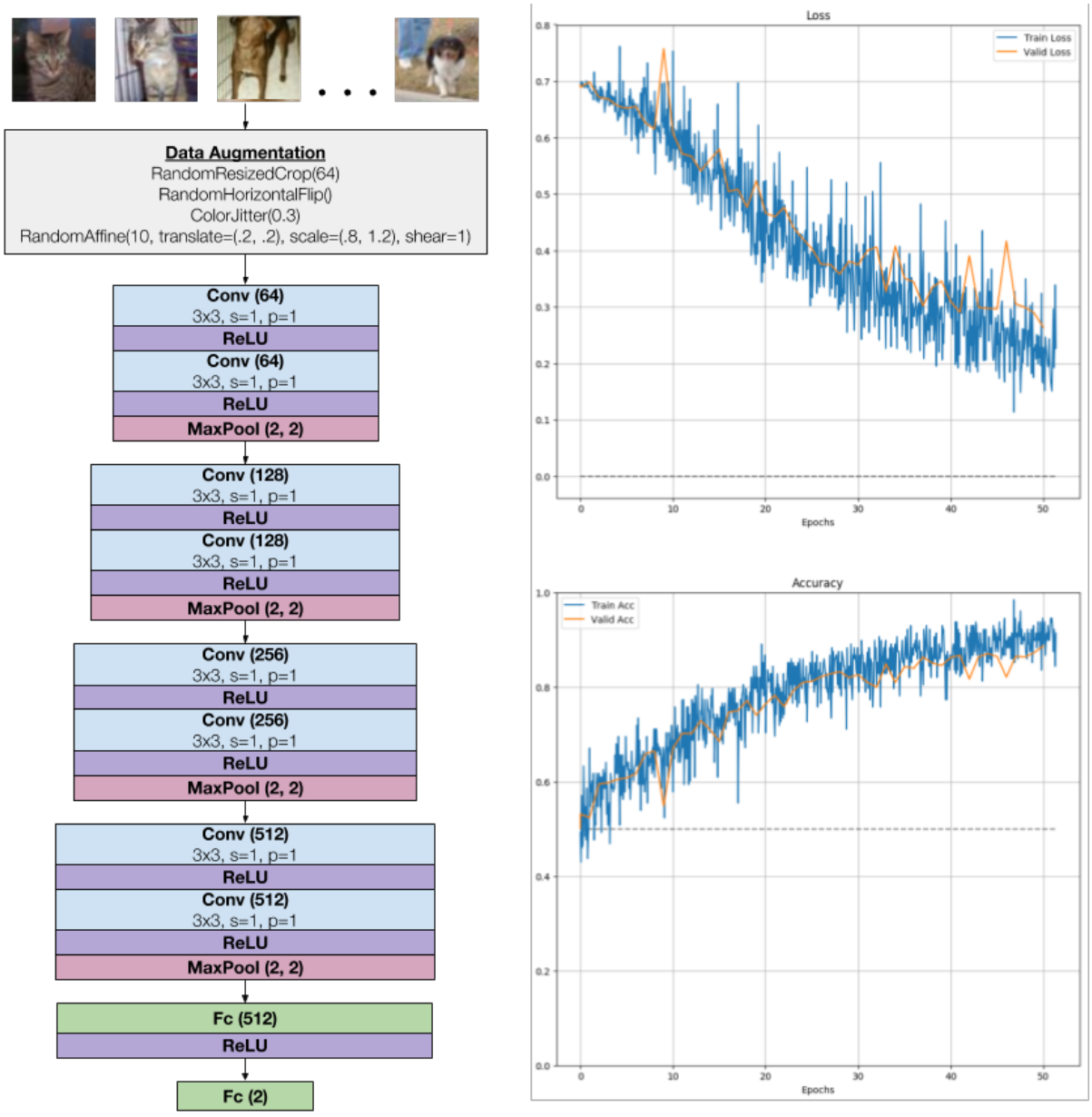

Figure 11 – Bigger network

We see that this network is performing better than the previous ones. We continued to train this with half the learning rate.

(b) **Continue with bigger model, ReduceLROnPlateau**

We continued to train this with the same batch size (128), using SGD with half the previous learning rate - 0.05. We also reduce the LR whenever the validation loss does not decrease for 5 epochs.

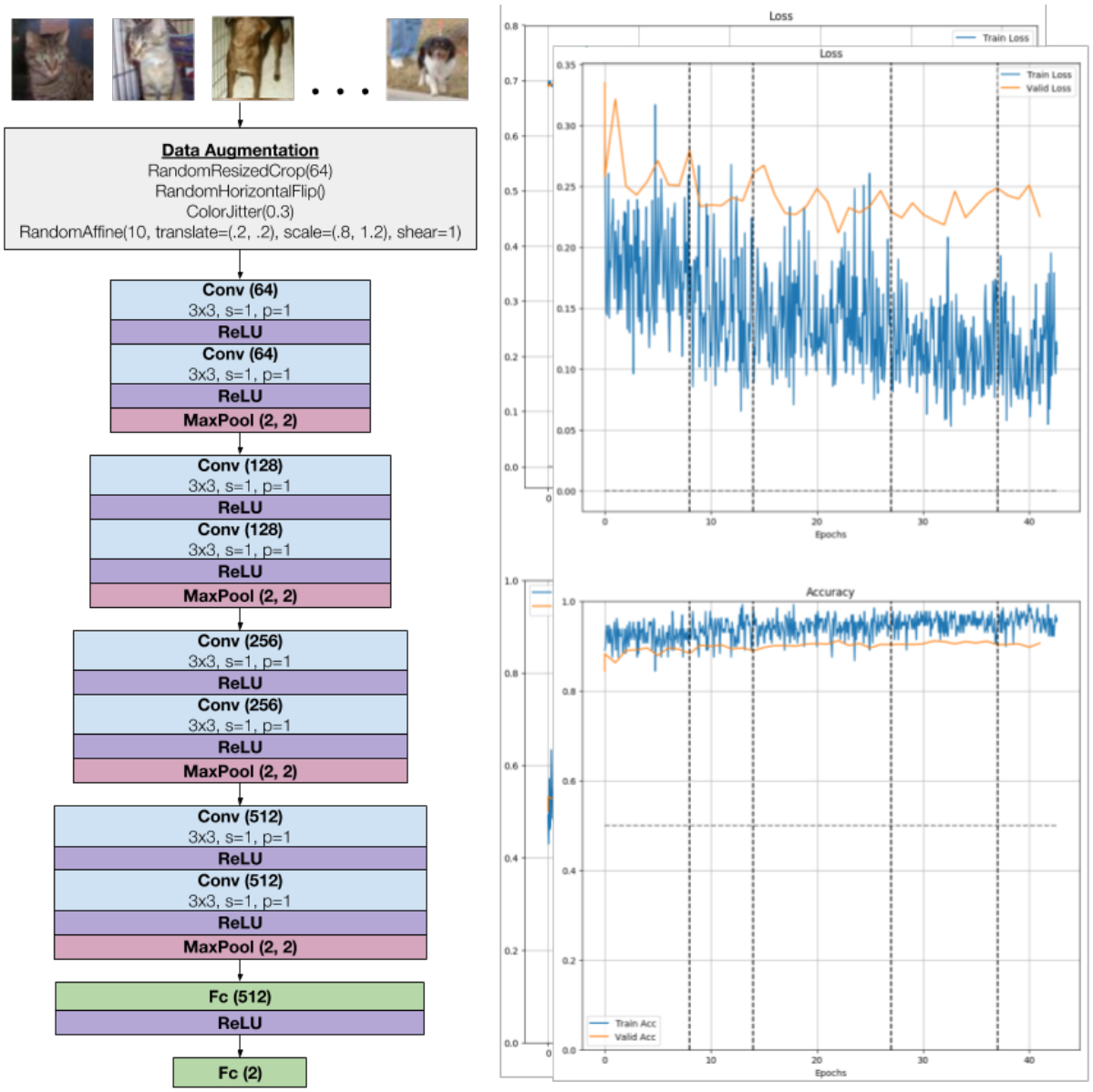

Figure 12 – Bigger network, ReduceLROnPlateau

We see that this network is then trained while the LR was halved 4 more times. This network got a test accuracy of **0.91036** on the public leaderboard.

5. **Skip connections**

Next, we introduce skip connections in this big model to check their effect. Since it is the same model with some skip connections with 1×1 convolutions added, it is only slightly bigger with 9,054,146 parameteres.

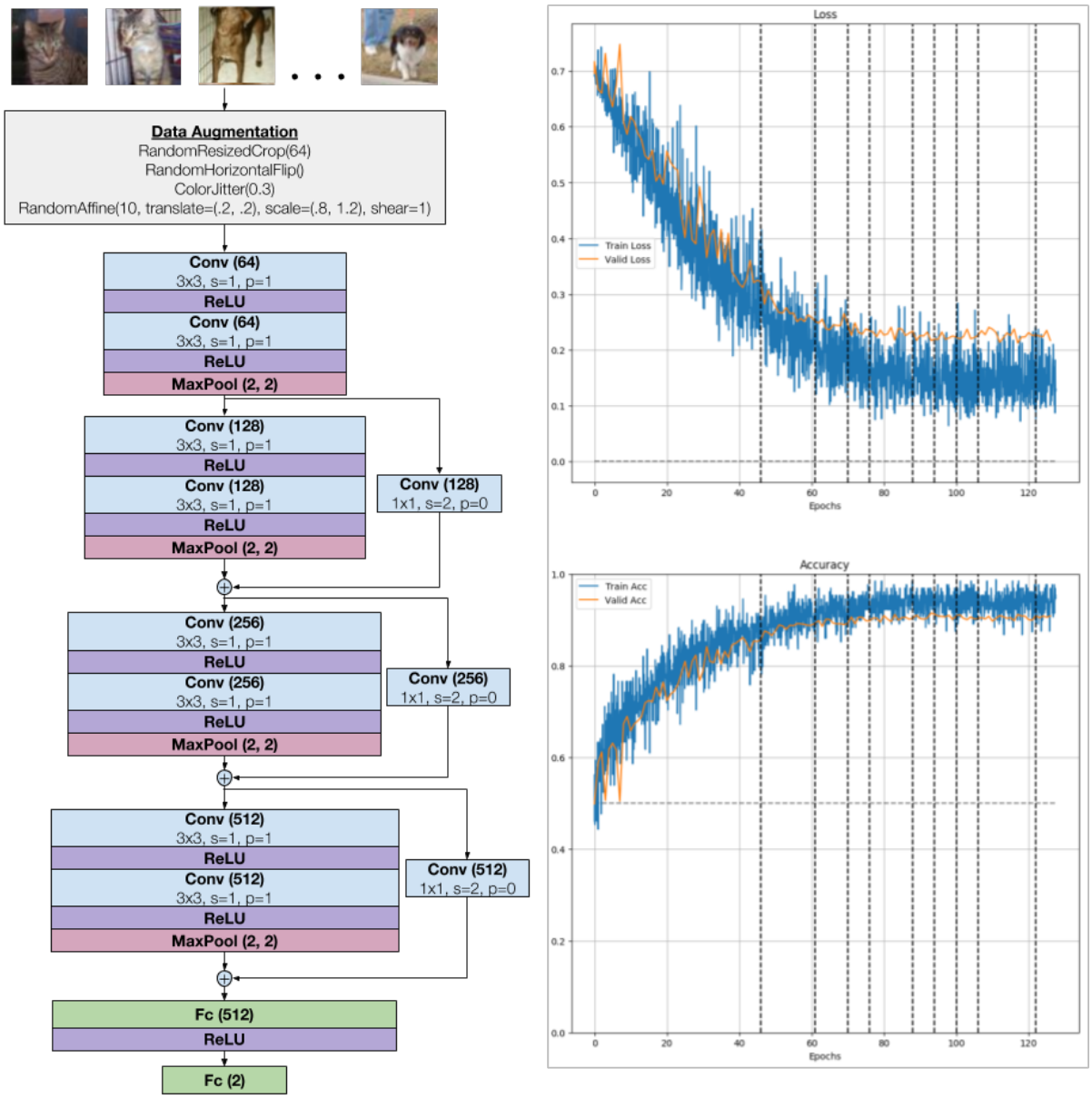

Figure 13 – Bigger network, ReduceLROnPlateau

We see that this model trains very well on the dataset, but it is comparable to the previous network without skip connections. So we conclude that for this task and this model, skip connections do not offer much help.

This got an accuracy of **0.91796** on the public leaderboard.

Skip connections are useful for deeper models. So we go deeper !

6. **ResNet-18 style architecture**

We make an even bigger model inspired from ResNet18 using skip connections, and train it on our dataset. This model has 62,111,234 parameters.

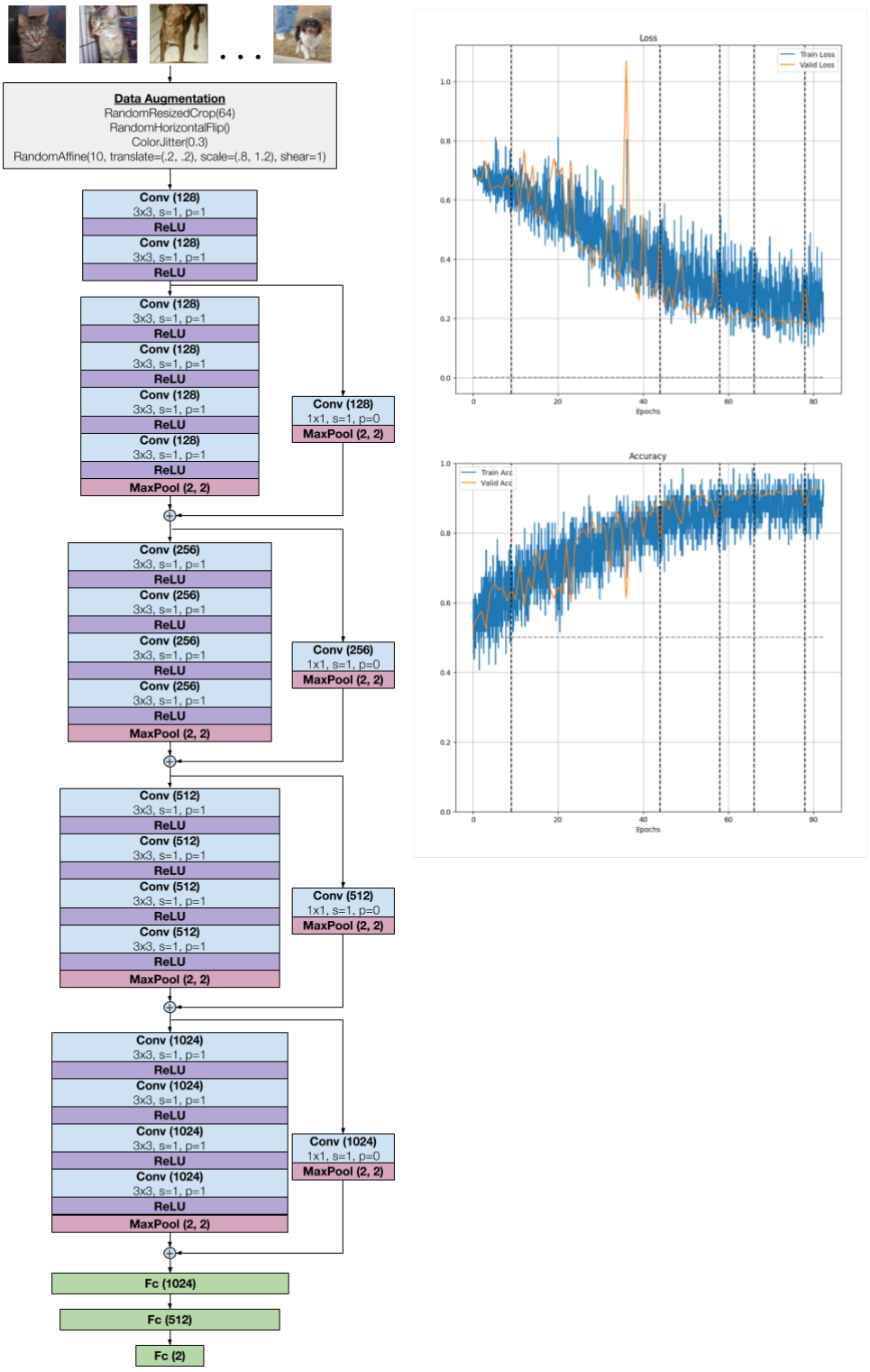

Figure 14 – ResNet-18 style architecture

This achieves an accuracy of **0.93717** in the public leaderboard.

(a) **Visualizing the weights of the convolutional layer**

We visualize the (normalized) weights of the 128 filters of $3 \times 3 \times 3$ size of the first convolutional layer at the start and end of training :

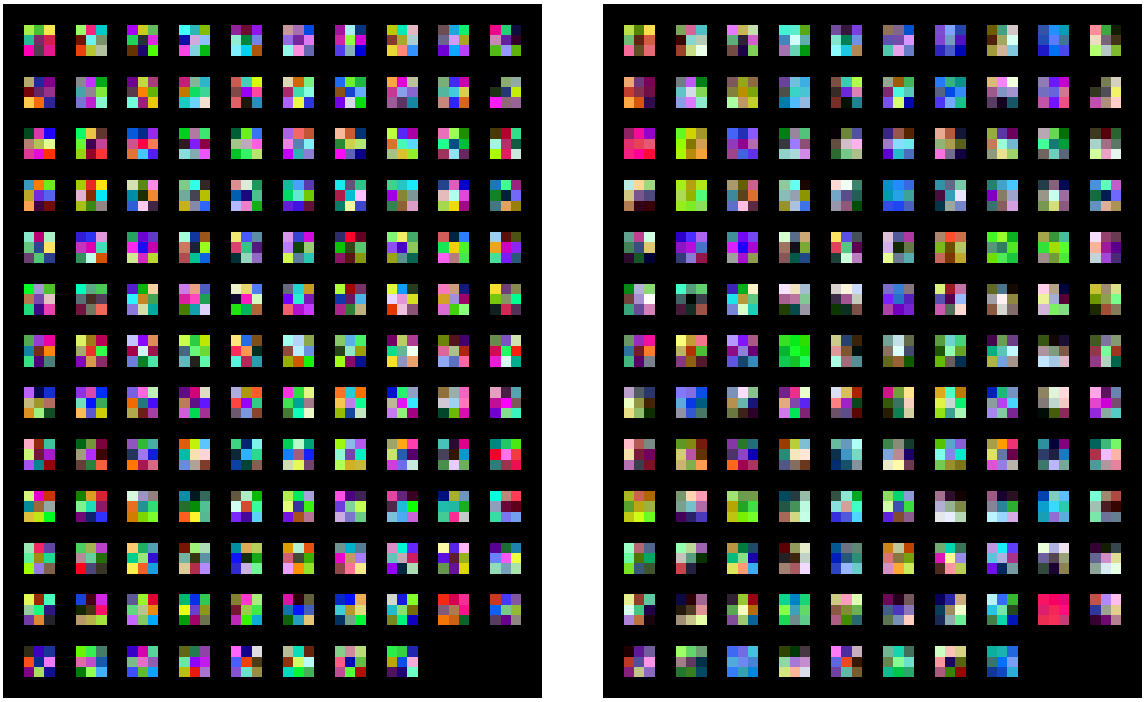

Figure 15 – First convolutional layer weights (a) at start of training ; (b) at end of training

We can see that at the start, all the weights are seemingly random, while at the end, they look for specific patterns such as the color pink, or a blue line, or corners.

(b) **Failure cases**

The predominant failure cases seem to be those in which the face of the cat or dog is not visible due to being occluded, or out of the frame, or not frontal :

Figure 16 – Misclassified samples due to face not visible

or because the pixel area of the face was not as big as in the rest of the images :

Figure 17 – Misclassified samples due to small face

Here's a random sampling of the failure cases :

Figure 18 – Misclassified samples

(c) **Attention by occlusion**

We also tried to observe which parts of the image the model focuses on when trying to classify, by occluding different parts of the image and recording the model output.

We used an occlusion window of 7 pixels.

Figure 19 – (top) Data, and (bottom) respective occluded samples

Using the occluded samples, we forward propagated through the network and found the probability of the correct class, and make a heatmap (in greyscale) of that to find out which areas the model concentrates on.

Below is the log of the heatmap for visual clarity, and the heatmap superimposed on the original image.

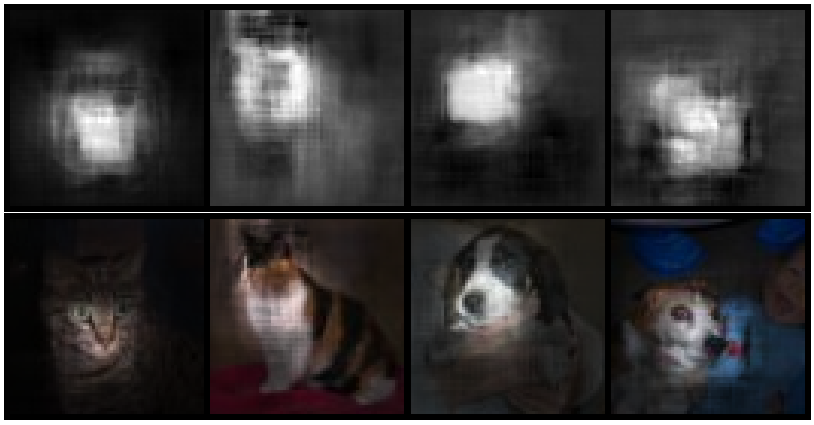

Figure 20 – (top) Log of heatmaps, and (bottom) heatmaps superimposed on the original images

We can see that the network focuses on the face of the cat or dog to classify it correctly. This is in agreement with the fact that most of the failure cases were with those images where the face was not visible or too small.

7. **Improvements**

— We observed that a lot of images in the training set contain frontal faces of the cats and dogs. Indeed, we see that our best model focuses on the face, and fails when it is not able to detect the face properly. Hence, the training dataset could be improved by including more images where the animals are looking elsewhere as well.

— It is possible that BatchNorm shall improve the validation accuracy. Indeed, actual ResNet18 does contain BatchNorm, but we did not implement it.