Team Members:

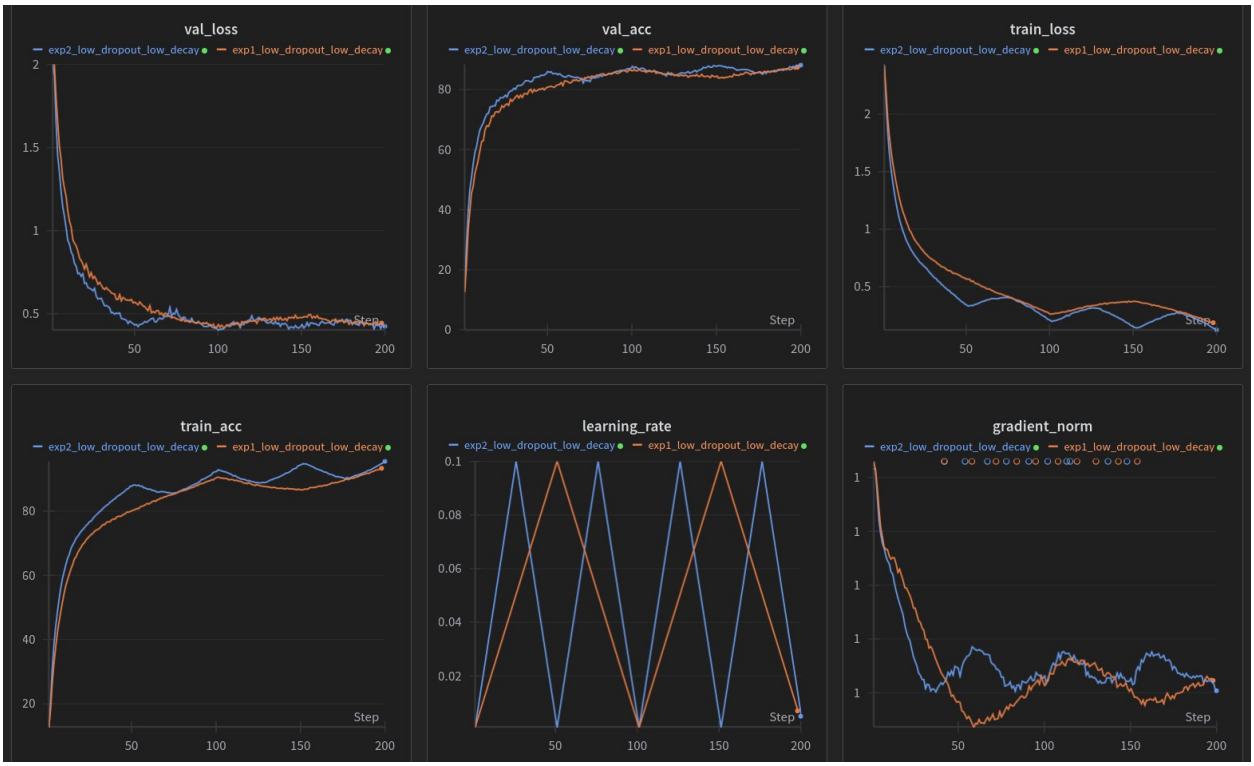
- Sohith Bandari
- Kushagra Yadav
- Nishanth G Palaniswami

Table of Contents

Our Experiment Timeline

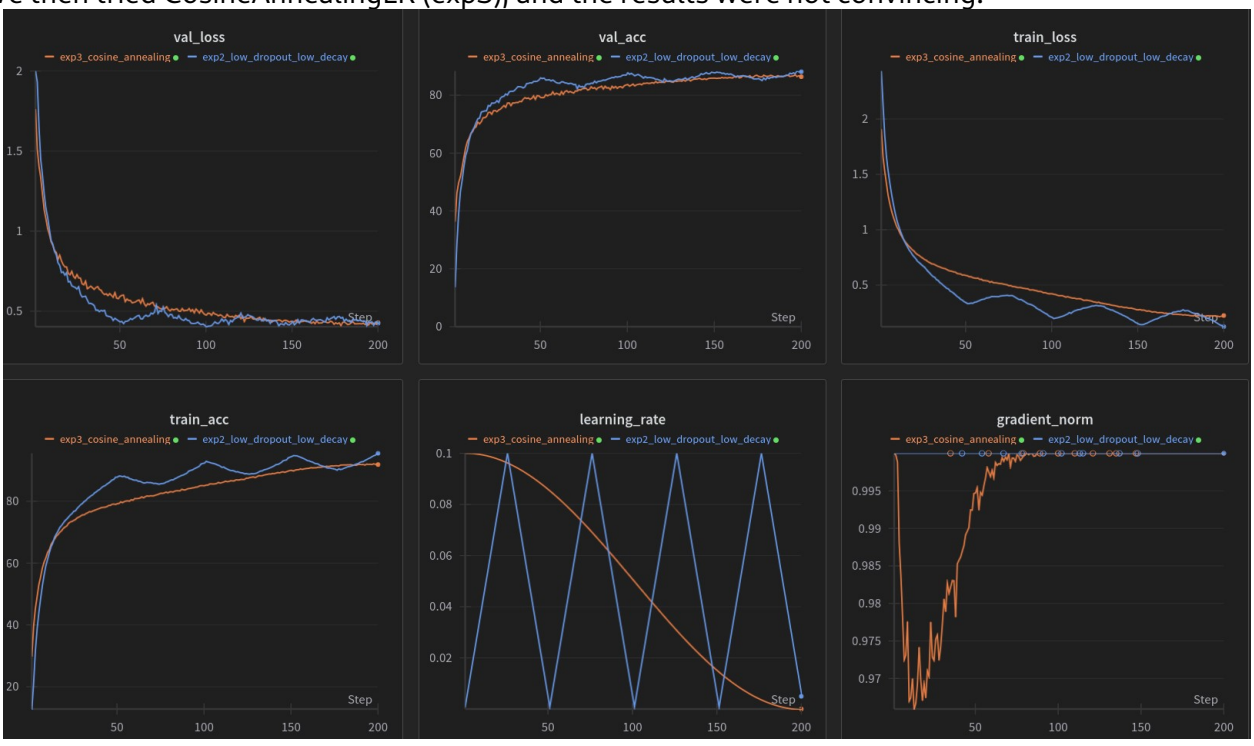
Our first run consisted of testing a custom built ResNet18 while checking if either of the two schedulers were better:

- CyclicLR with `mode=triangular` and step_size=200 (exp1)
- CyclicLR with `mode=triangular` and step_size=25 (exp2)

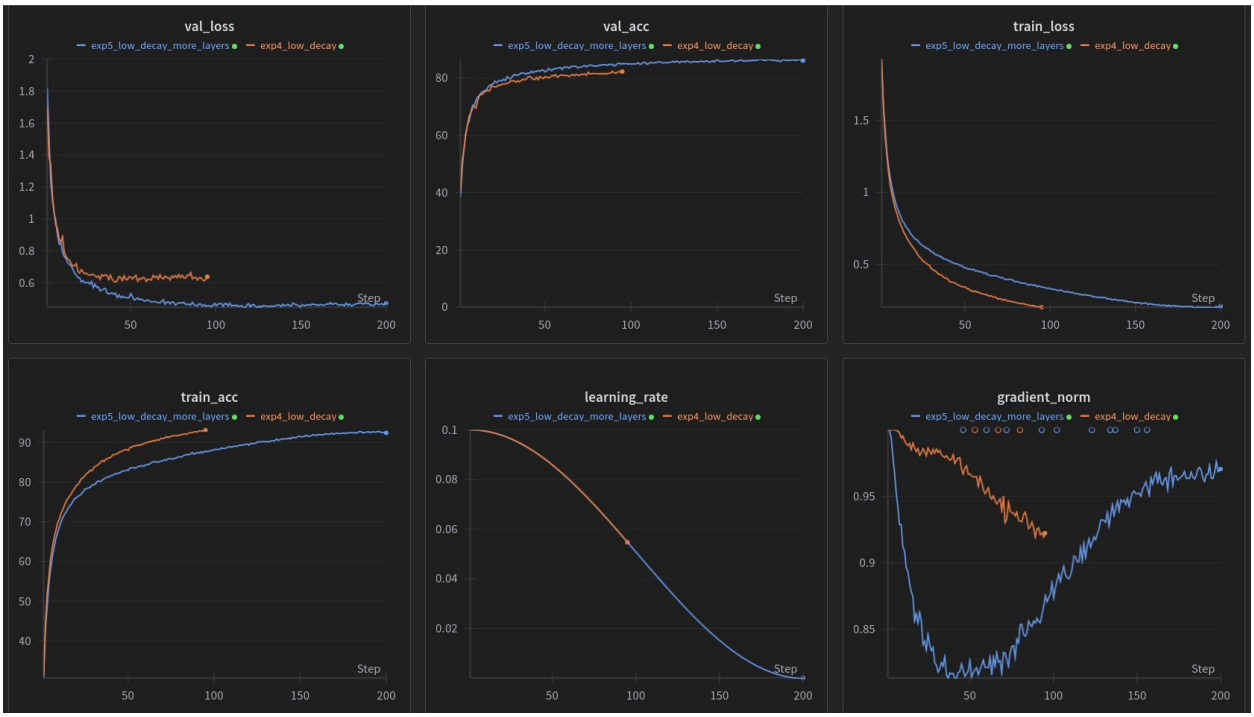


Exp 2 had an edge over exp1 with a 1% validation accuracy difference (88.37 vs 87.37)

We then tried CosineAnnealingLR (exp3), and the results were not convincing:



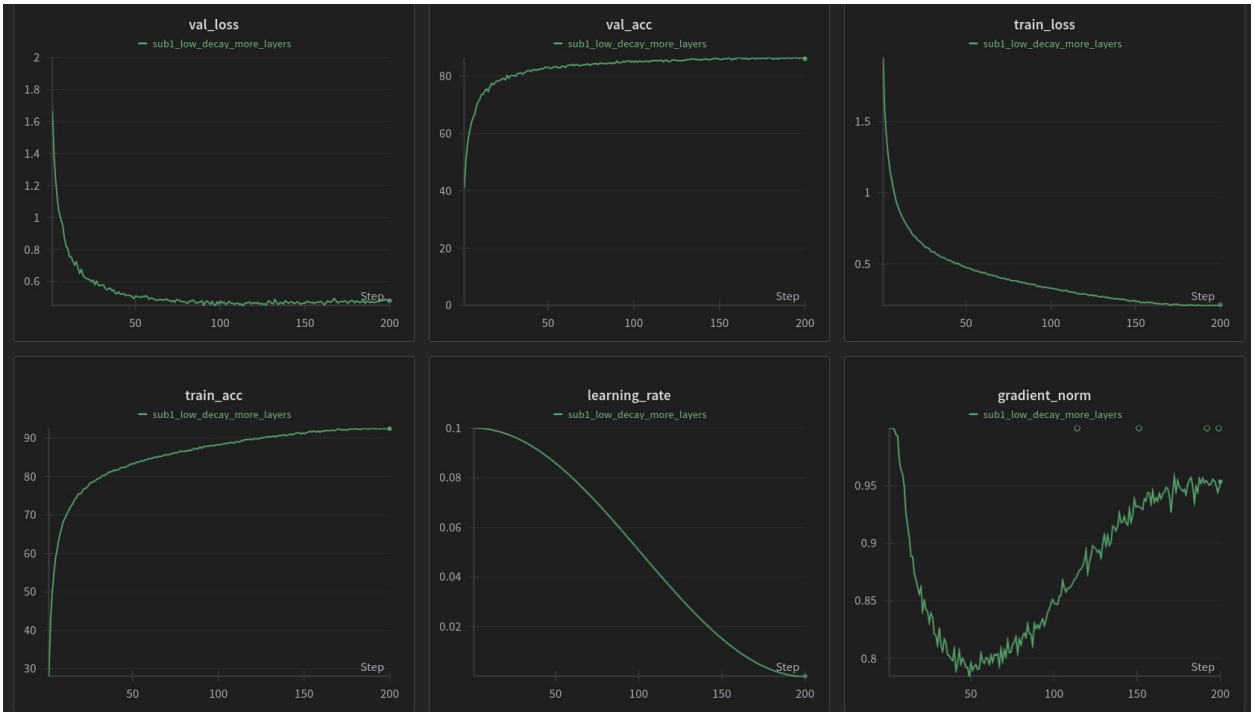
We then tried to lower the weight decay, but increase the number of blocks (from [2, 2, 2, 2] to [3, 2, 3, 2]), but for this experiment, we ran one experiement for 100 epochs and another for 200 epochs:



Clearly, 200 epochs was the answer here.

First Submission

We went ahead with our first submission:

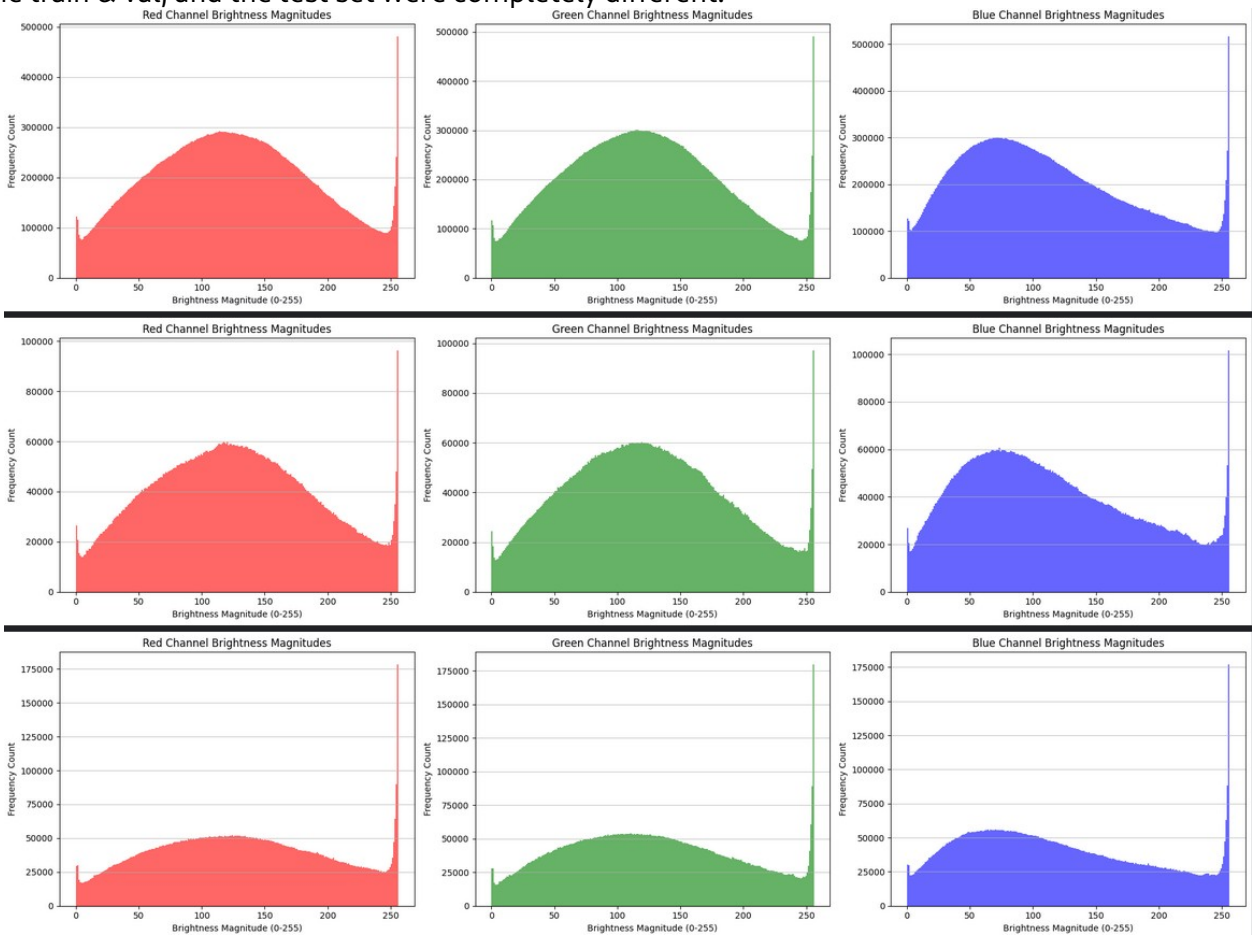


Public Leaderboard Score: 0.77114

But our validation accuracy was 86.5%. We felt that something was amiss in the training data. So we decided to investigate.

Investigating the Test Dataset

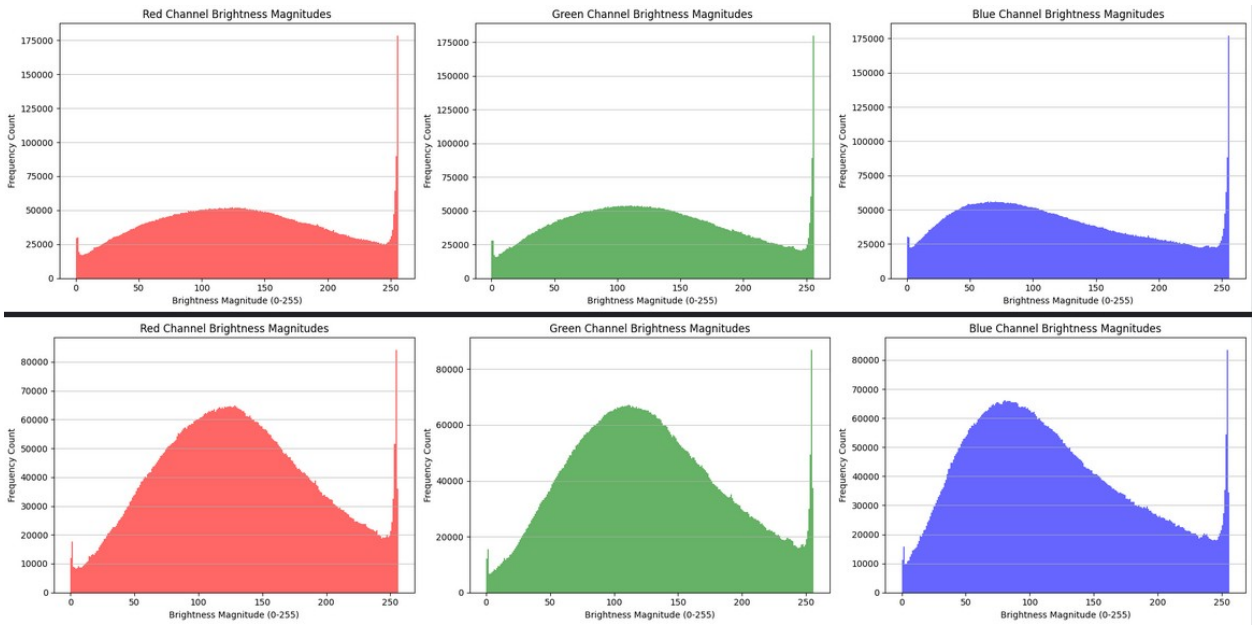
On plotting the histogram of the train, val and the test set, we found out that the distribution of the train & val, and the test set were completely different:



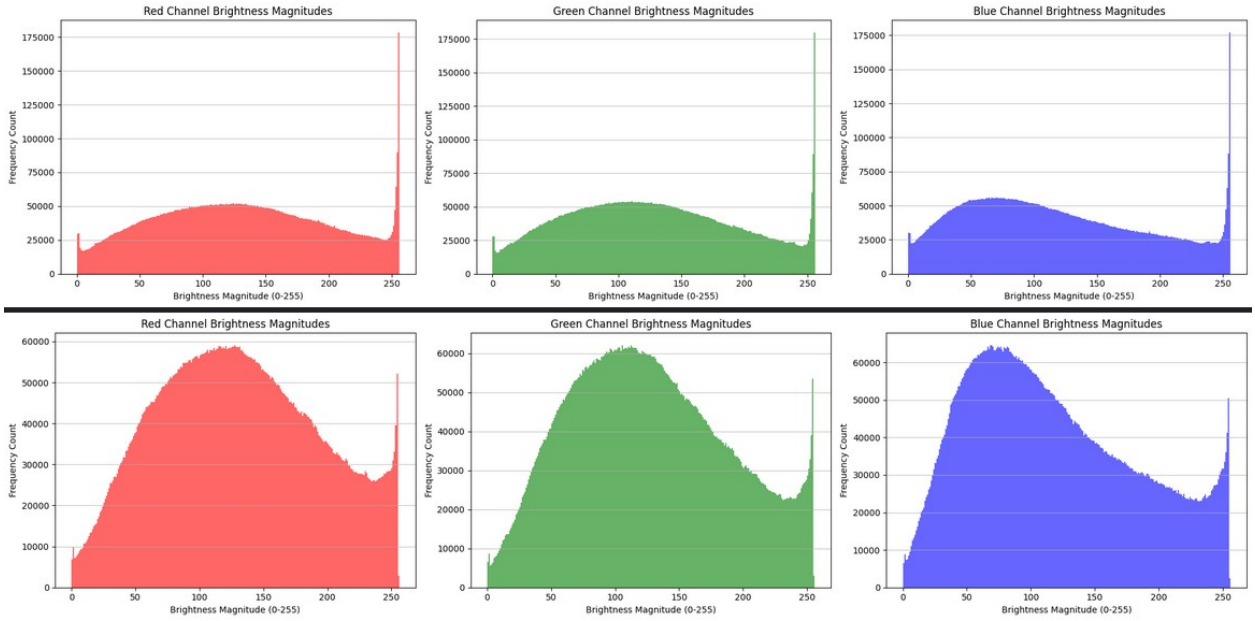
Top to Bottom: Train, Val and Test

Testing various de-noising techniques

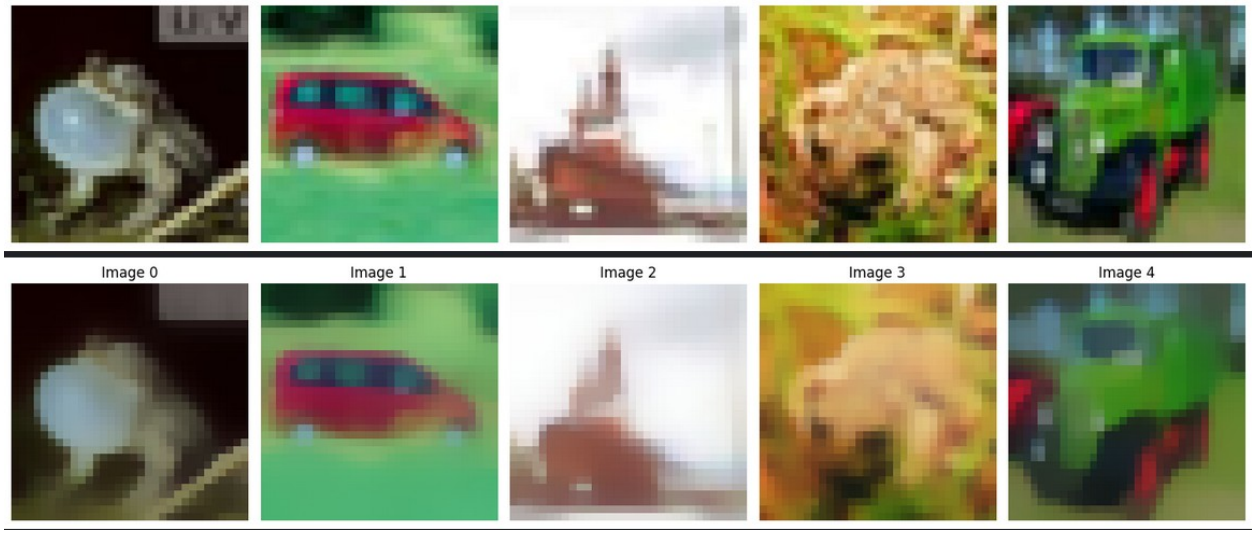
We mistakenly thought that the test set had noise in it, so we set out to test different denoising techniques:



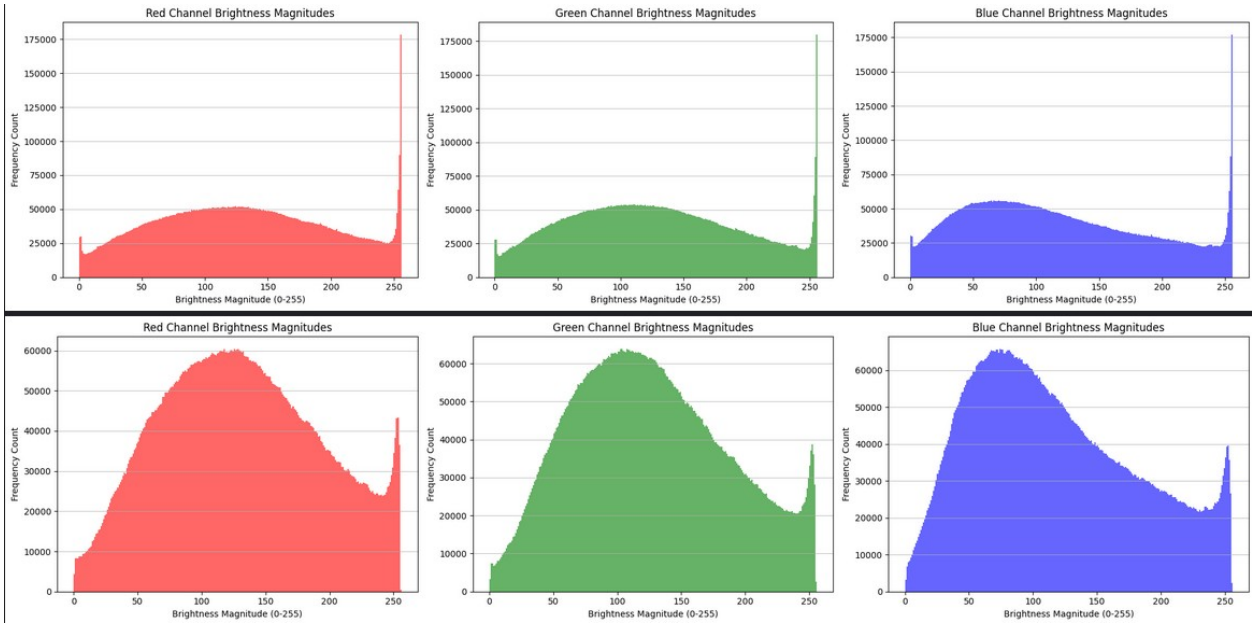
Above: Effect of Alpha-trimmed mean filter on the test dataset



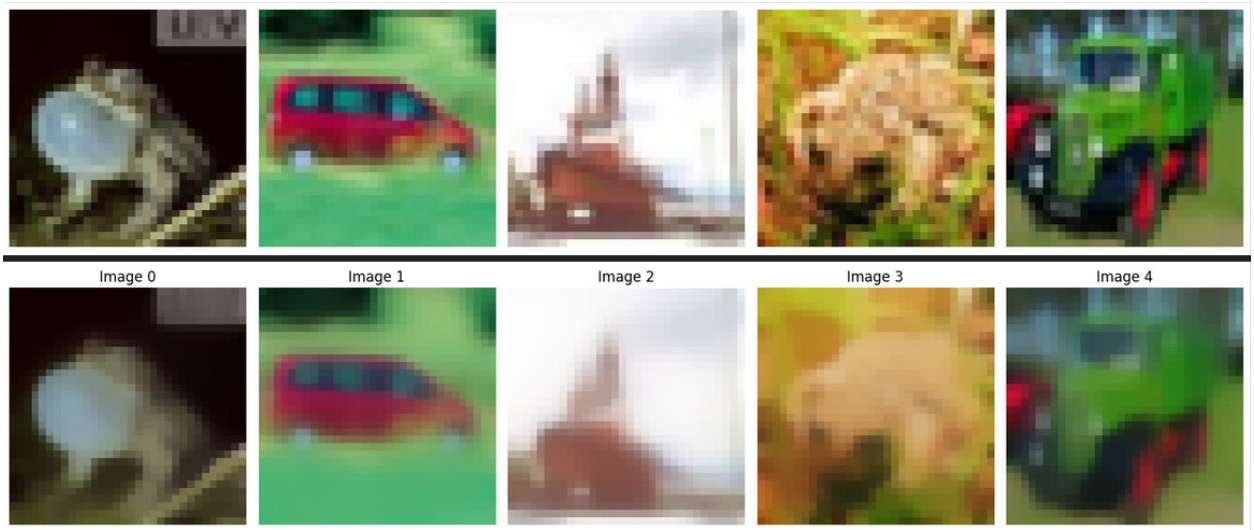
Above: Effect of Total Variance - Chambolle filter on the test dataset



Above: Effect of Total Variance - Chambolle filter on the images of the test dataset



Above: Effect of Total Variance - Bregman filter on the test dataset



Above: Effect of Total Variance - Bregman filter on the images of the test dataset

We felt that we were losing edge information with de-noising, therefore we explored various edge detection methods to pass an additional channel into the model, along with the denoised images:

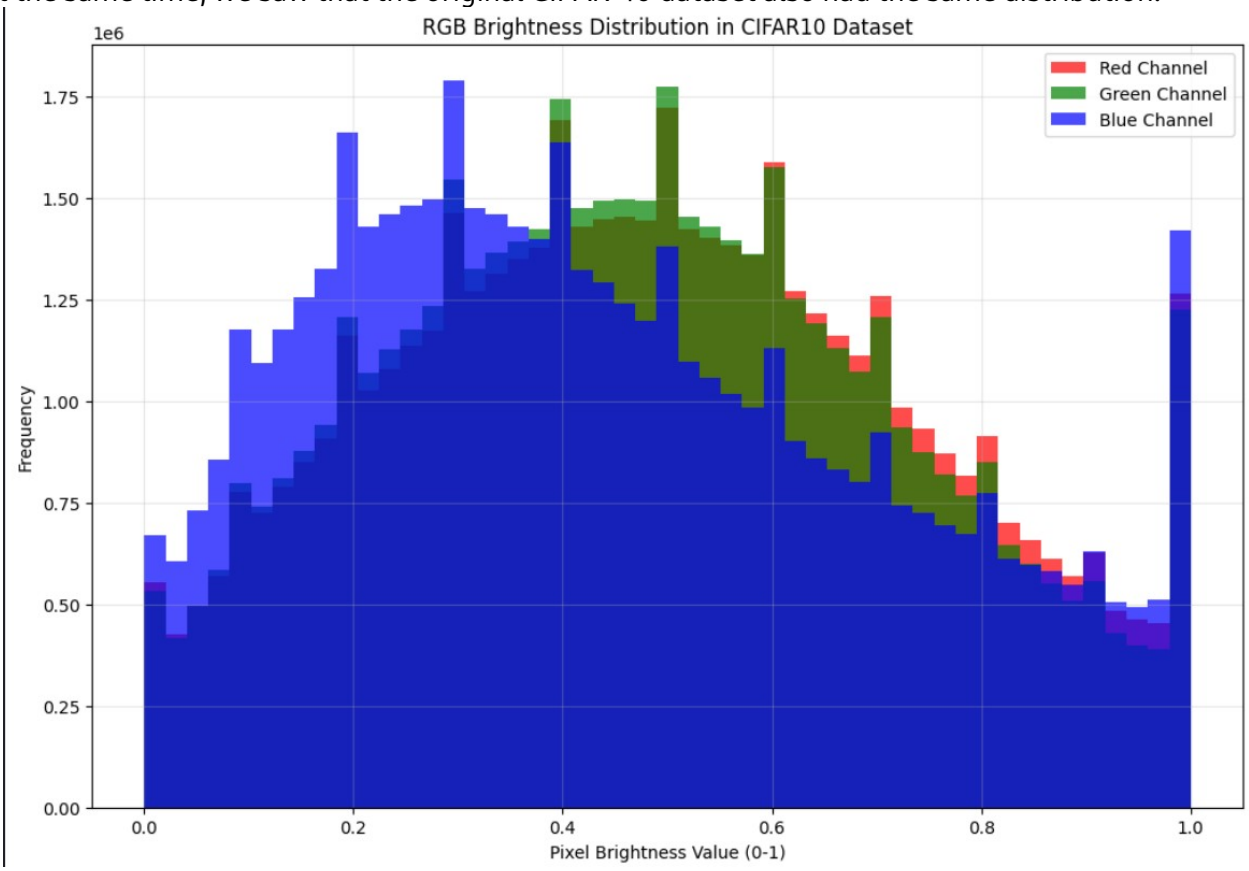
Above: Edge detection using Sobel filter on both the original test dataset, and the denoised test images

Second Submission

Our second submission with denoised images ended up with a disastrous result:

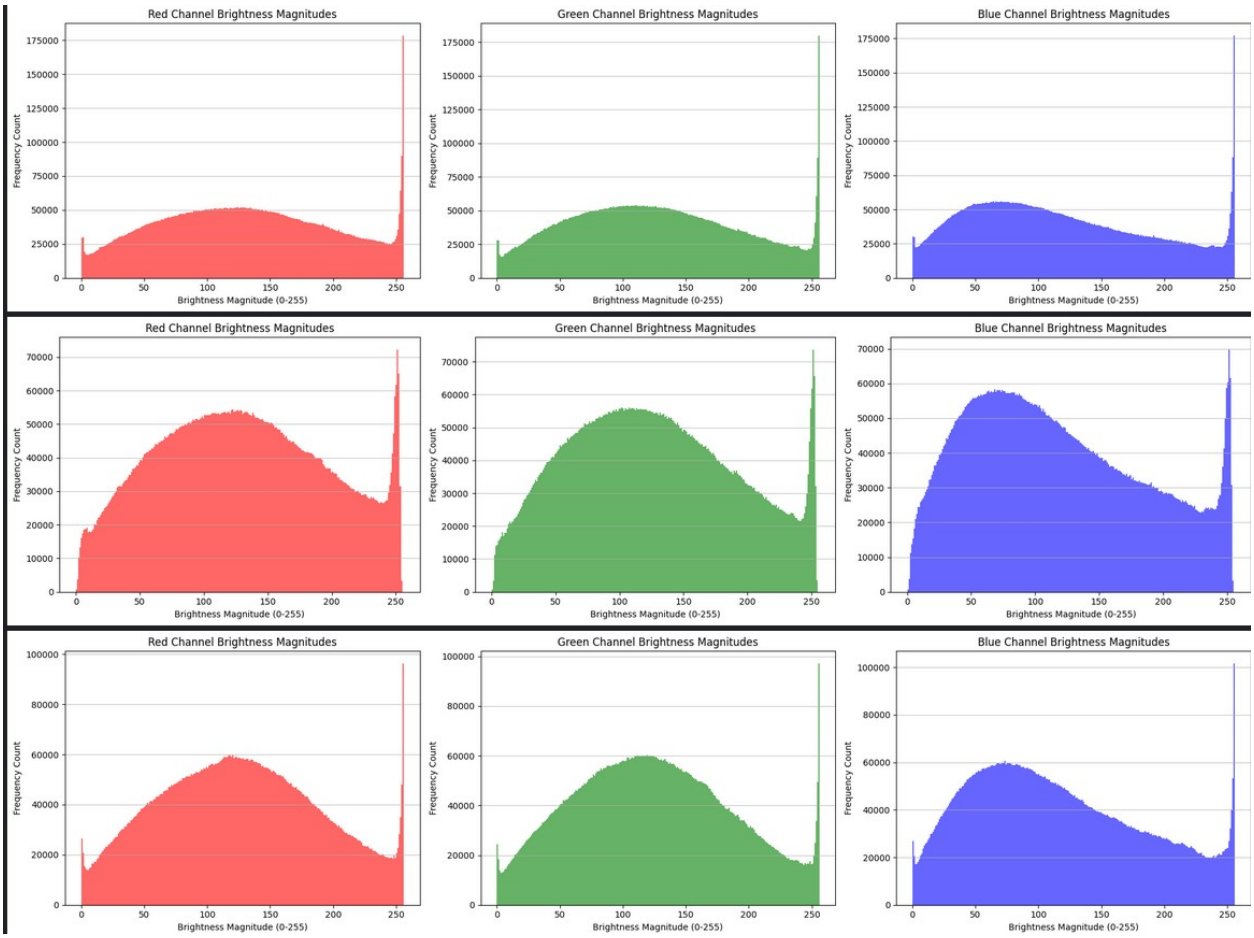
Public Leaderboard Score: 0.66919

At the same time, we saw that the original CIFAR-10 dataset also had the same distribution:

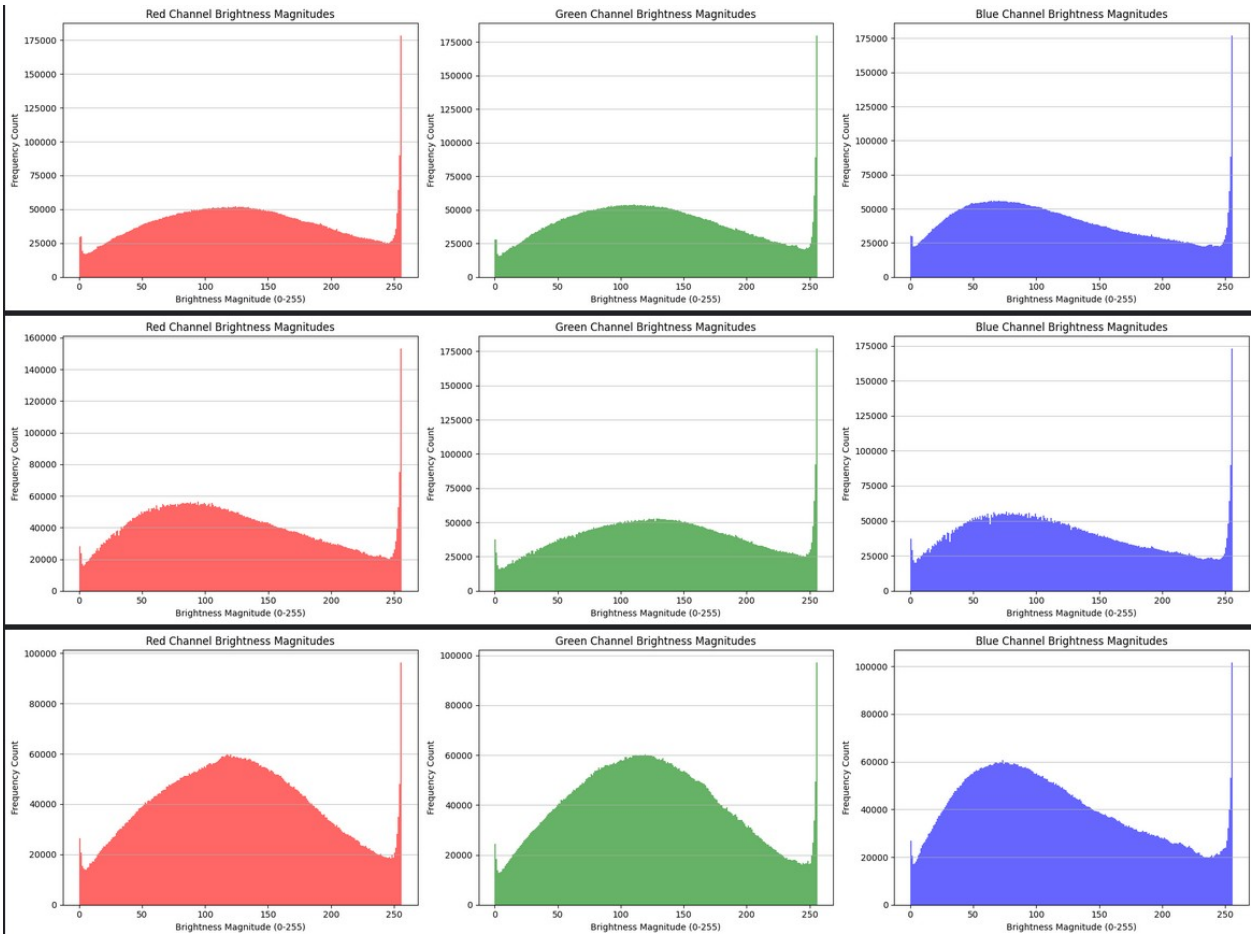


Color Jitter Case

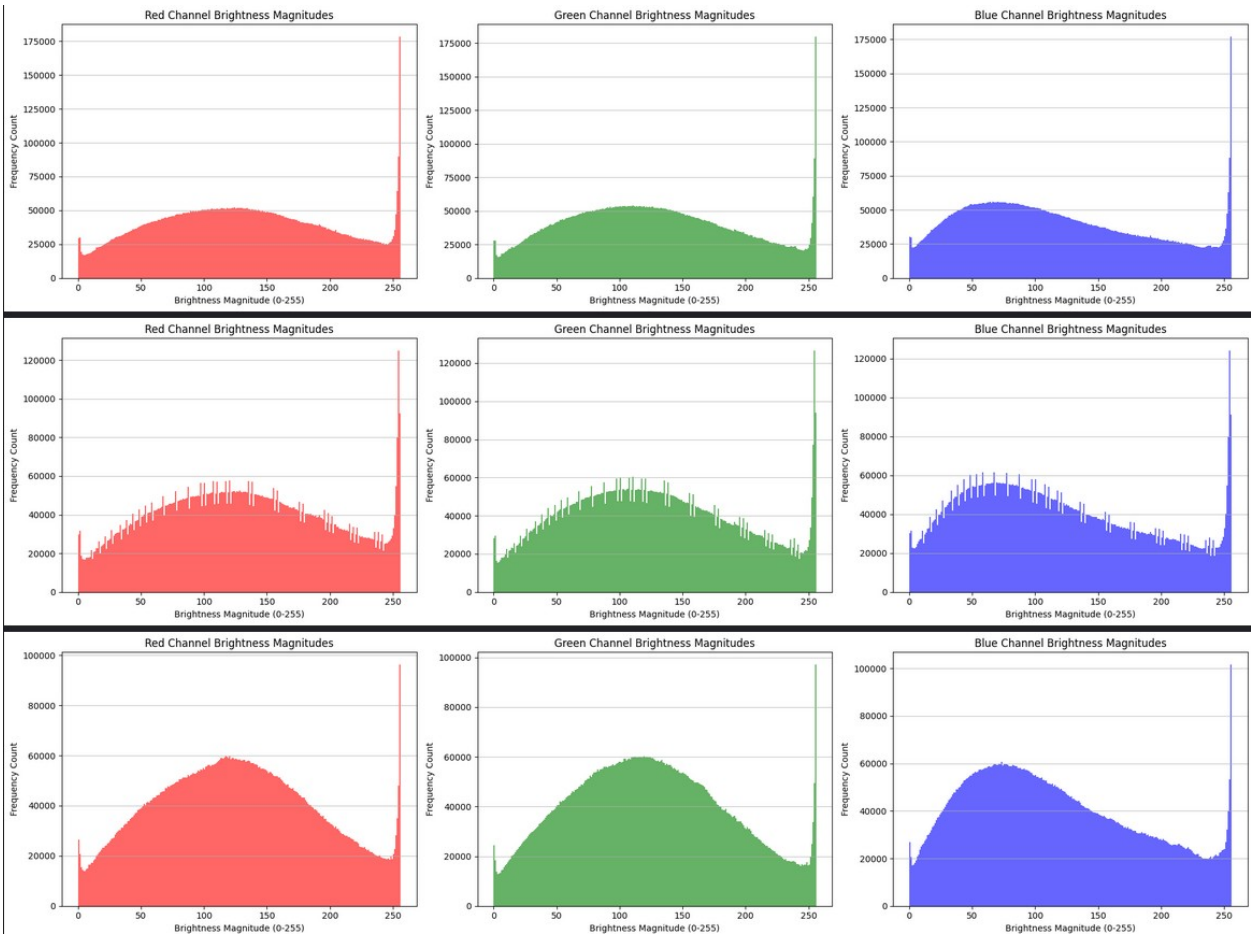
We then realised that if noise wasn't the issue, then surely there was some cosmetic changes, maybe something related to color jitter:



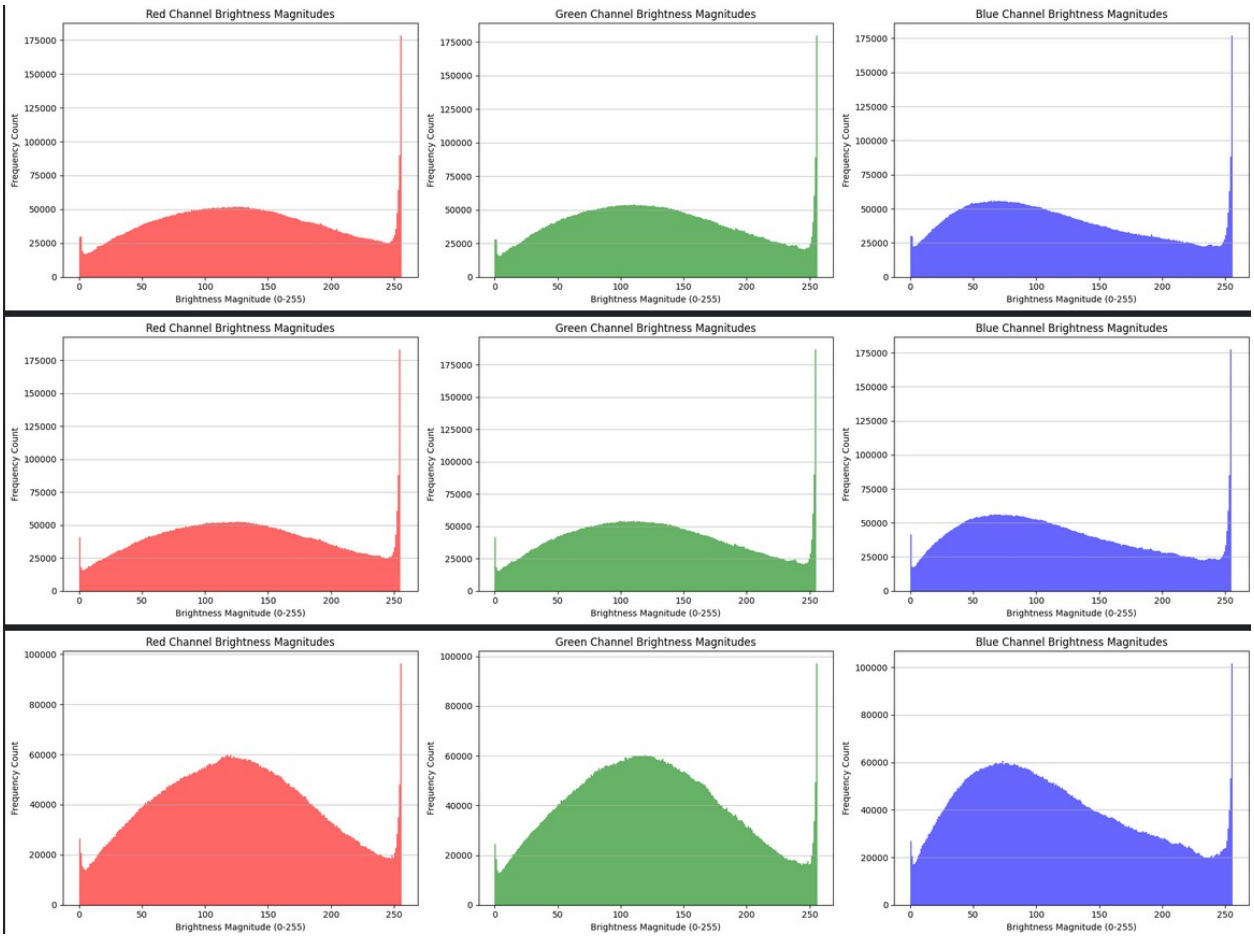
Above: changing contrast, top: original test set distribution, middle: modified test set distribution, bottom: val set distribution



Above: changing hue, top: original test set distribution, middle: modified test set distribution, bottom: val set distribution



Above: changing sharpness, top: original test set distribution, middle: modified test set distribution, bottom: val set distribution

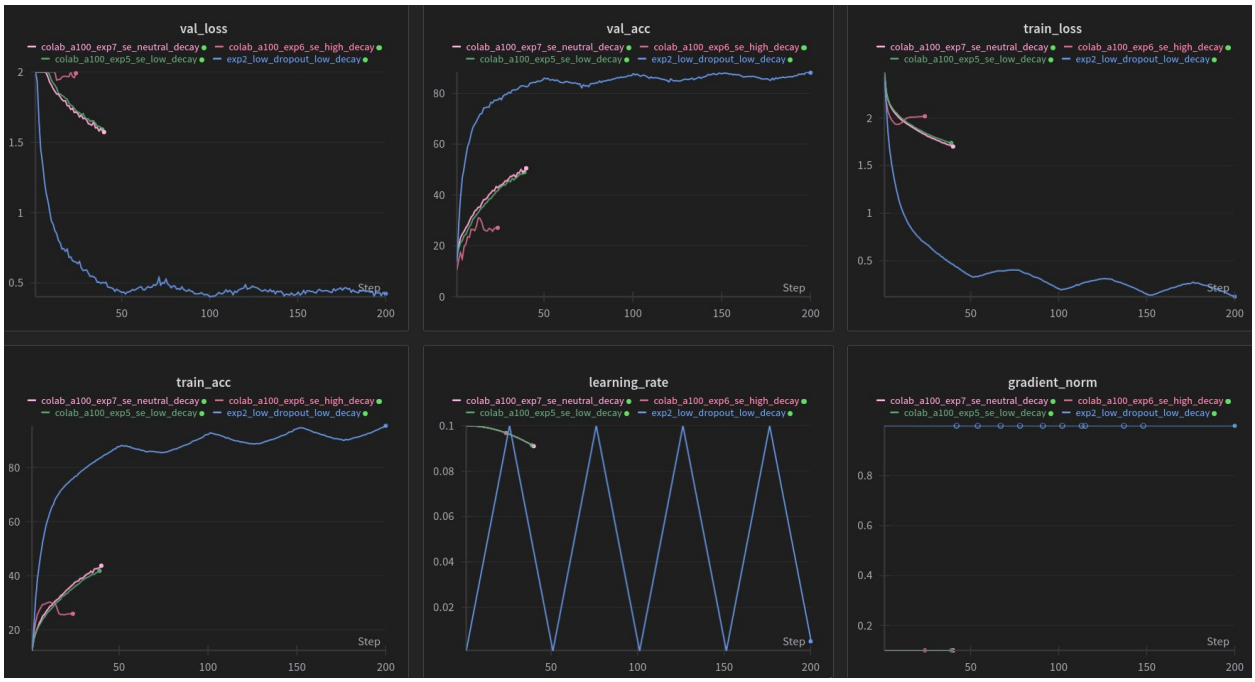


Above: changing saturation, top: original test set distribution, middle: modified test set distribution, bottom: val set distribution

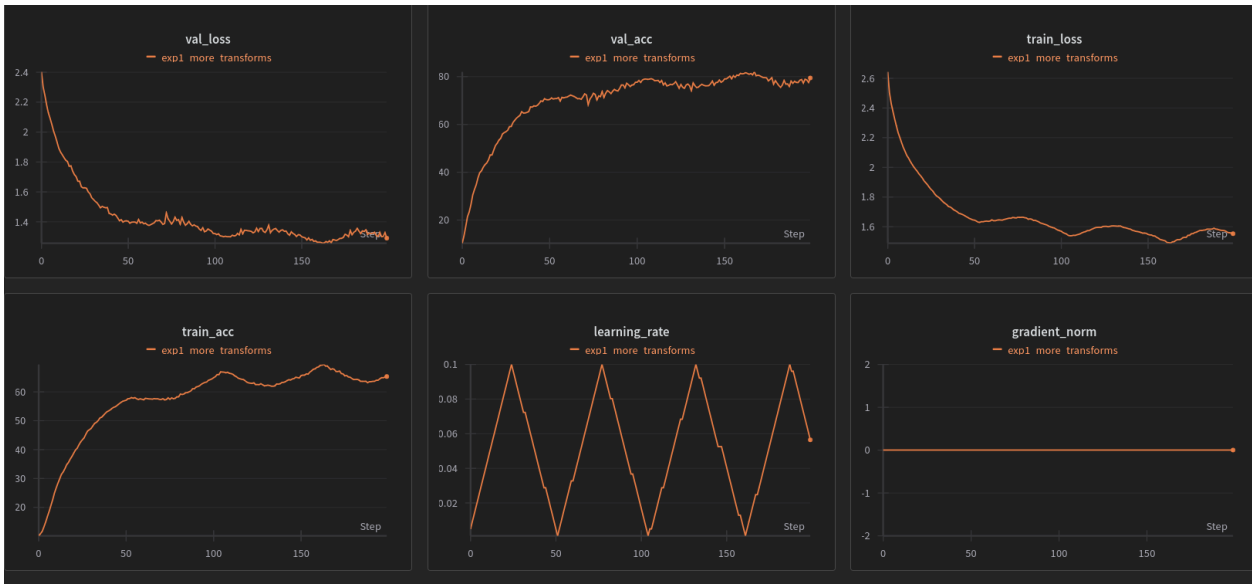
At this point, it was pretty clear that something was amiss with the contrast of the images, therefore we started adding a bit of color jitter to the test dataset. This gave an extra 0.2% boost in accuracy in the leaderboard standings.

Effect of weight decay

We also tested three different types of weight decays, each with dractically different results. The training was stopped mid-way as it was clear that the model would not get a good score.

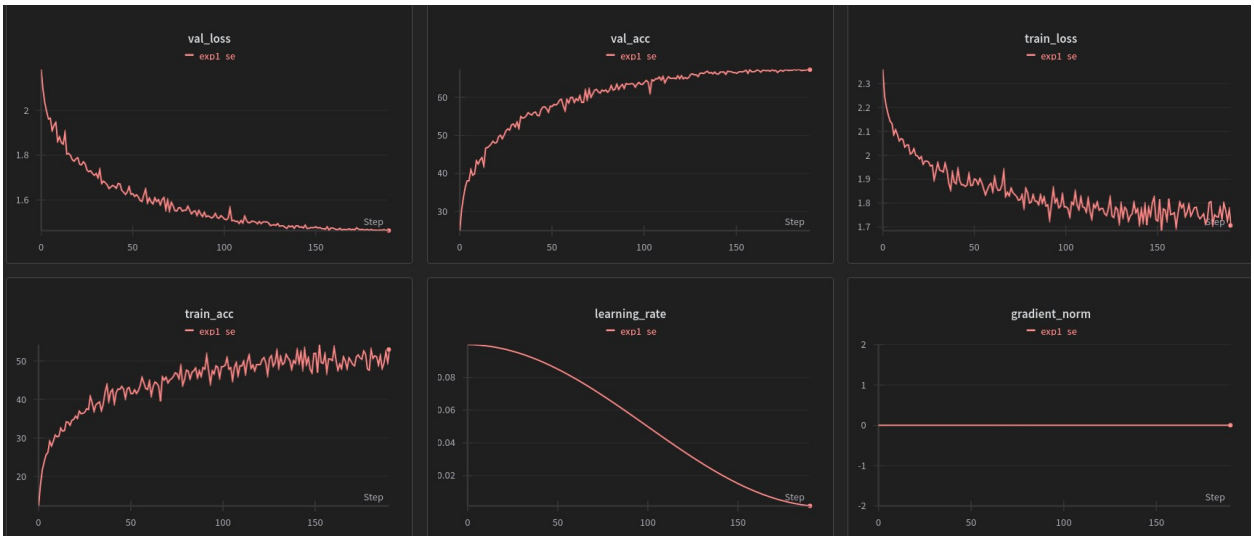


Effect of too many transforms



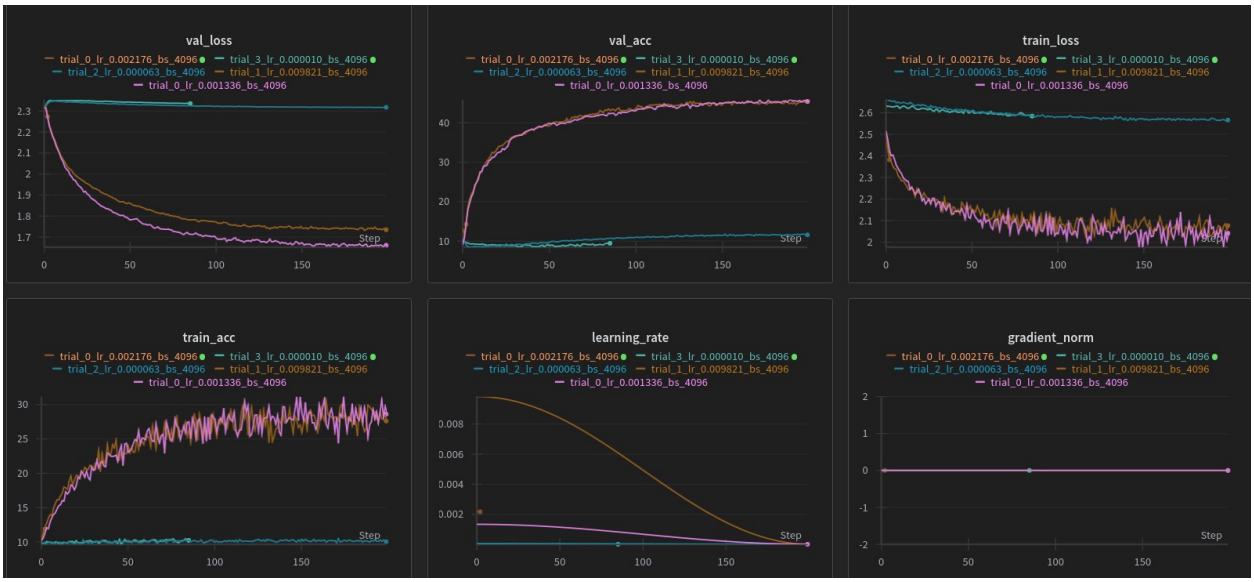
Too many transforms were slowing down our model, artificially making training very difficult with very bad results

Effects of Squeeze & Excite Block



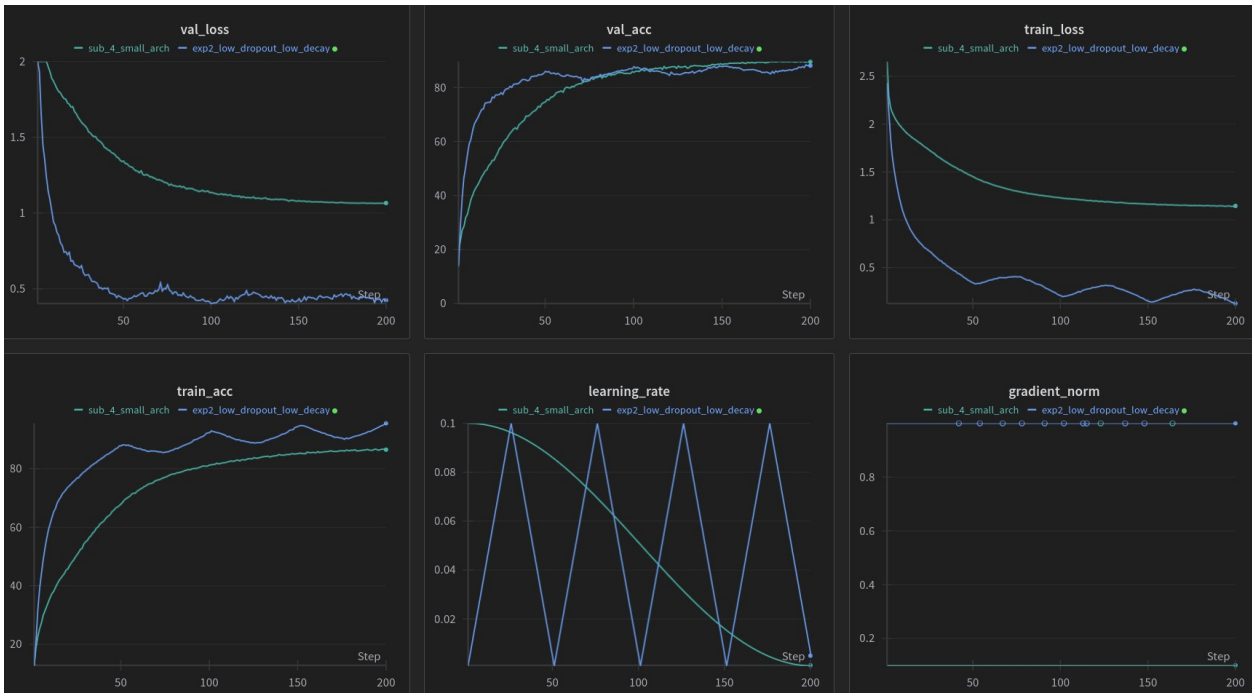
We particularly could not find the correct parameters for the squeeze and excite architectures, therfore we had to abandon this idea, as we could not get good performance

Hyperparameter tuning with Optuna



We had thought of finding hyperparameters with Optuna, but 5 trials in 9 hours was infeasible with the deadline of the project, therefore we had to abandon this as well.

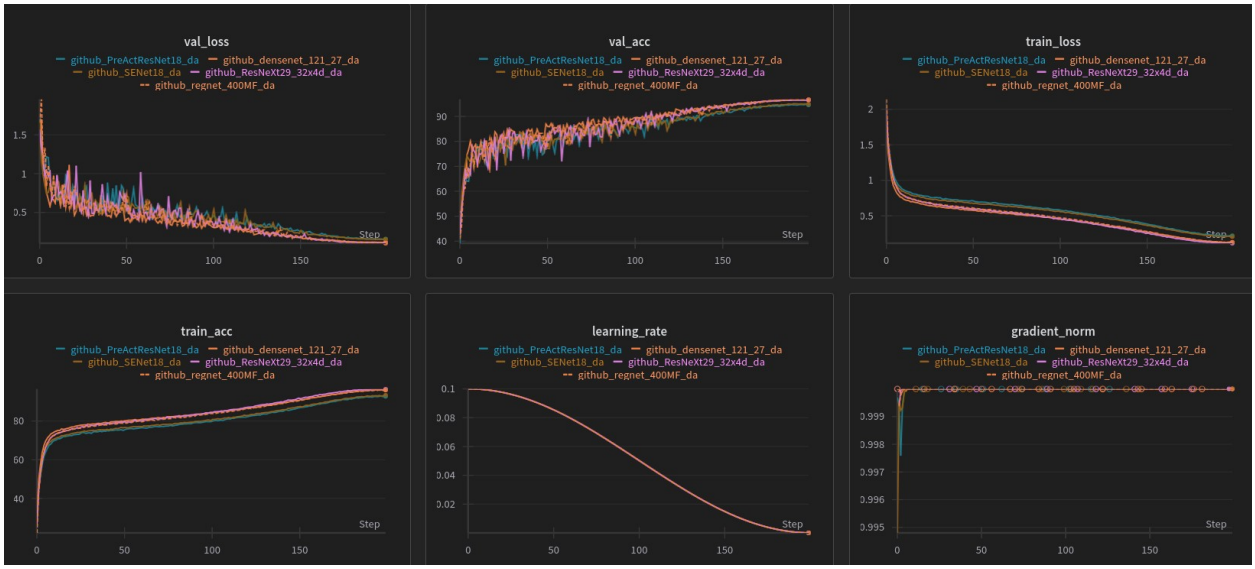
Effect of no. of parameters in the model



3M parameters vs 4.8M parameters did not make much of a difference, with only 1% difference in the accuracy metric

Changing the Model

At that point, we understood that there was something wrong with the architecture of the model, so we checked out the repository provided in the course, and the results were immediate:

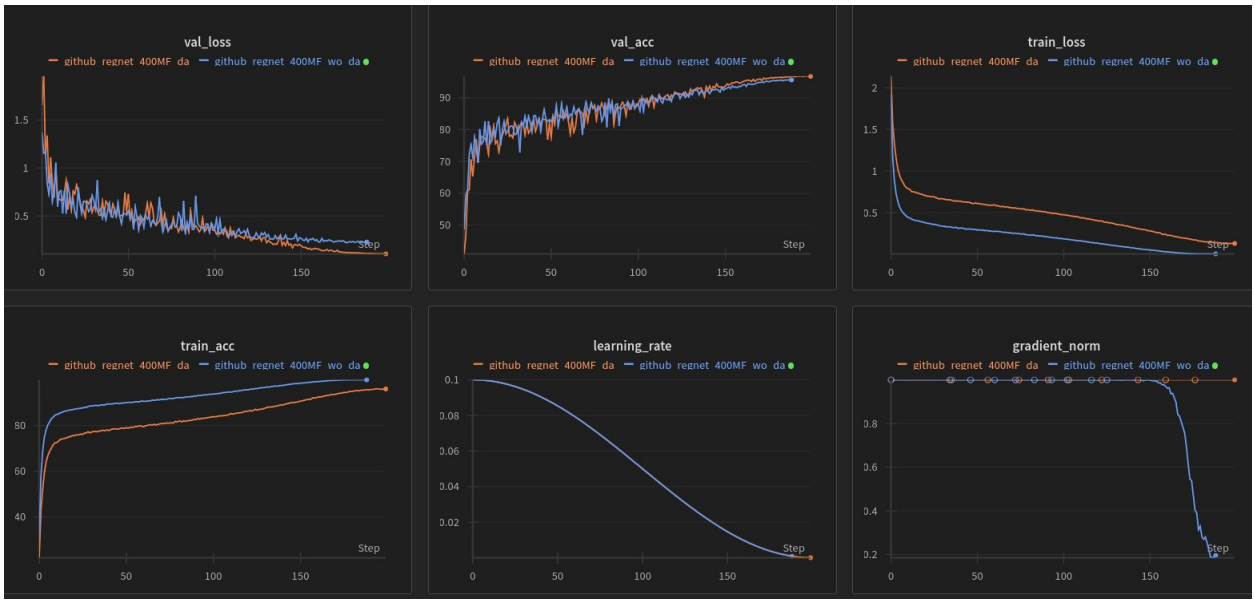


Our Leaderboard score shot upto 0.86971

We still don't understand how every model has almost the same curve though. Maybe the correct hyper-parameters mattered.

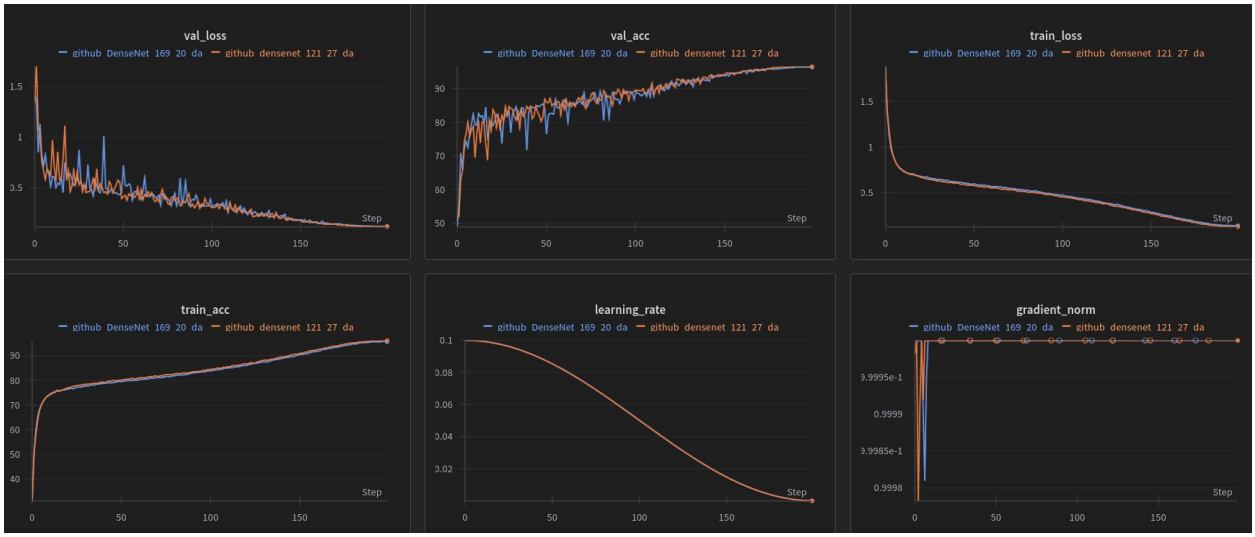
We eventually settled with DenseNet, as it was within the rules of the competition

Effect of Data Augmentations



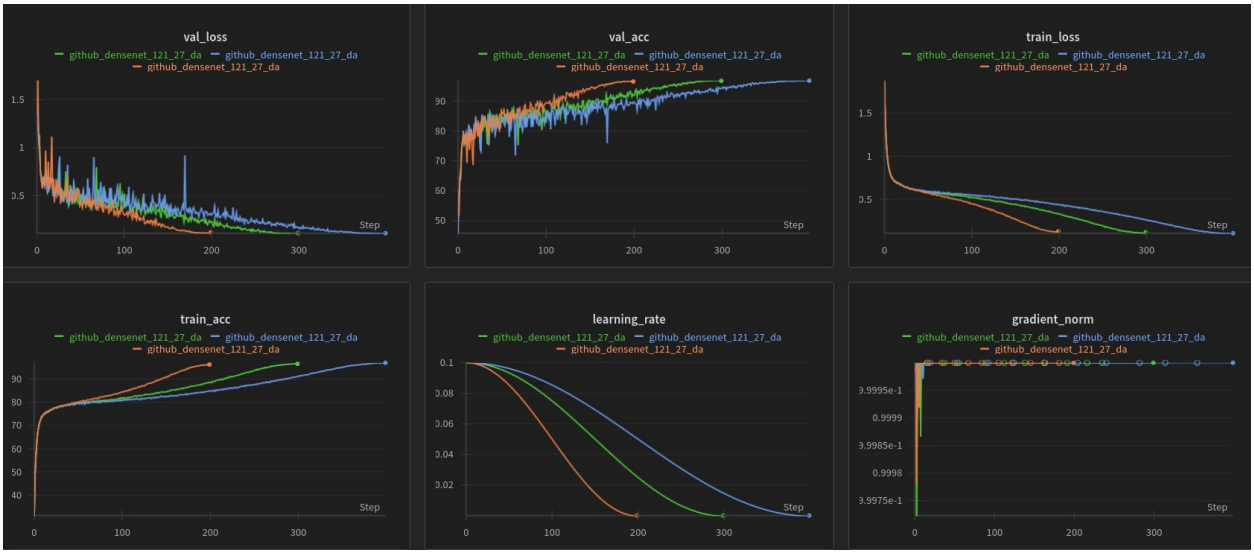
Tests on RegNet-X with and without data augmentations. Data Augmentations improved the accuracy by 1%

Effects of Changing Layers and growth rate

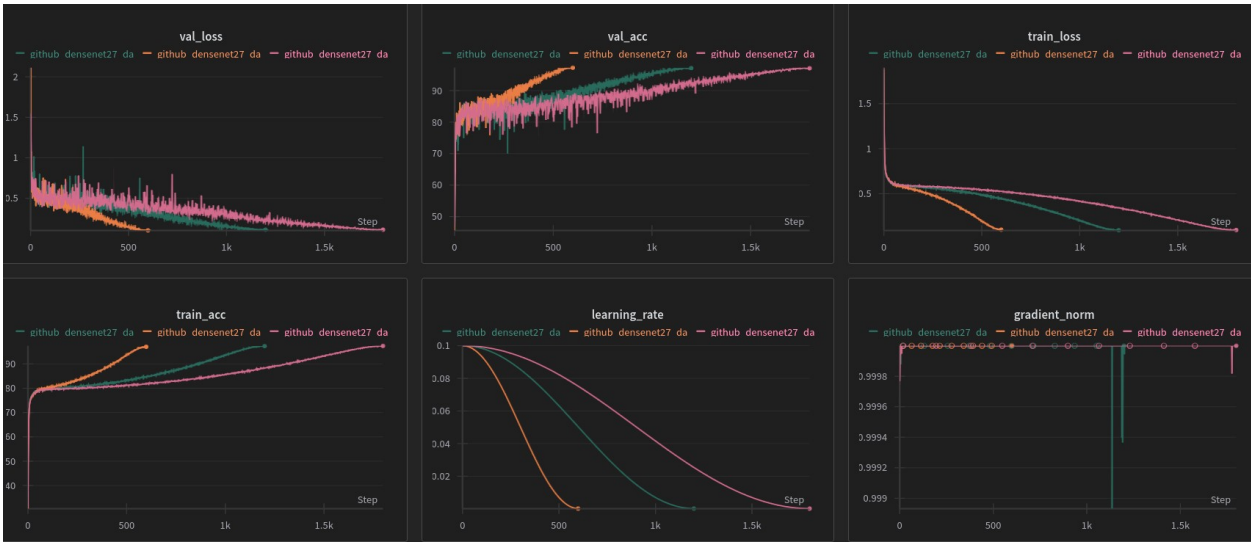


We eventually settled for DenseNet with 121 layers and a growth rate of 27

Effects of increasing the no. of epochs

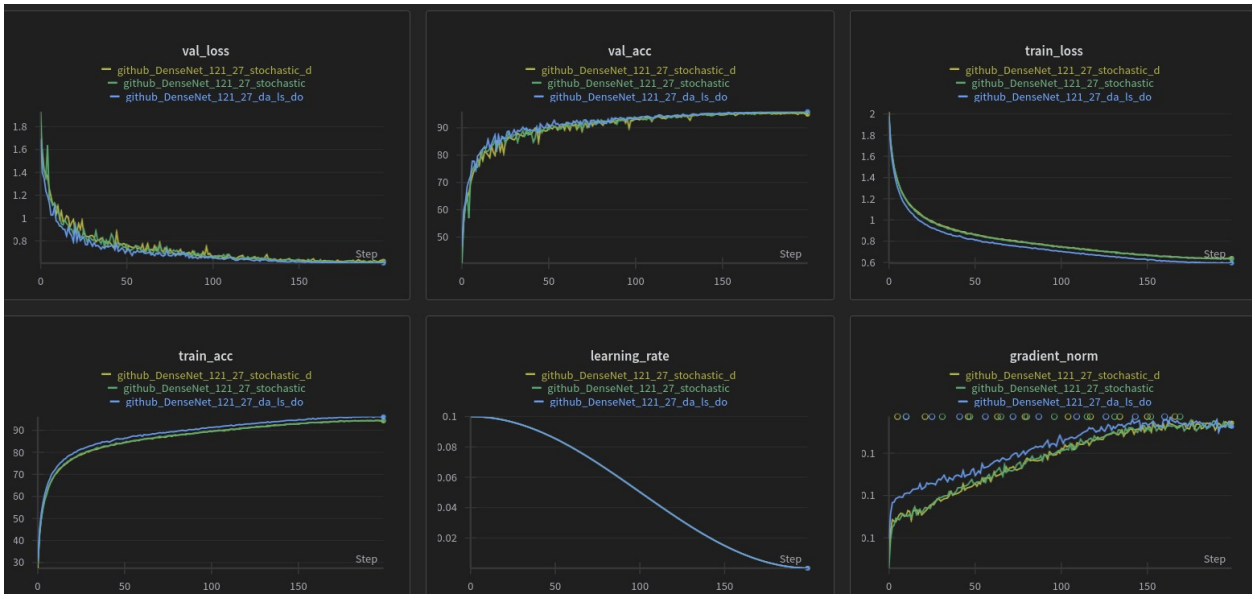


Clearly more epochs resulted in more accuracy. But how much more?



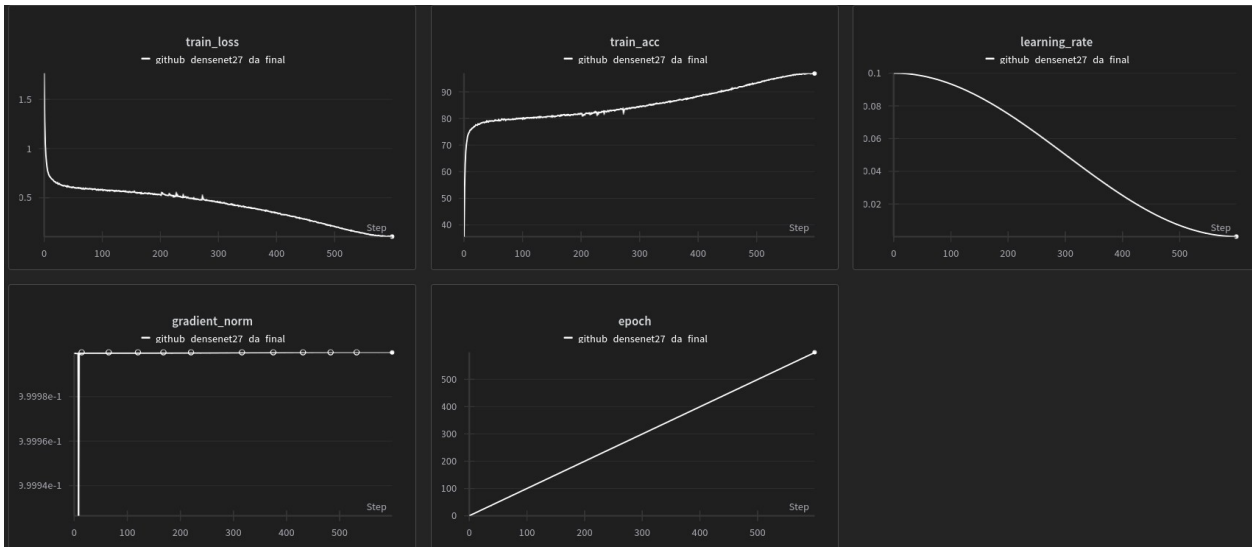
600 epochs performed better than 1200 epochs and 1800 epochs

Effect of Architechtural changes



Dropout, Stochastic Drop, and ResNet-D modifications. All performed worse than the original model.

The Final Submission



The final submission, which yielded us a Public Leaderboard score of 0.88372. This was trained on both the train and the val dataset, so there is no val_acc and val_loss to track.