The purpose of this document is to review our team’s performance during Kaggle’s Prostate-cANcer-graDe-Assessment-PANDA-Challenge. Our team ended up placing 654 of 1010, putting our final submission in the top 65%. We had a lot of fun during this competition, and while some of what follows may seem critical it is not taken personally.

What worked: Progressive resizing had a strong positive impact on our models performance. Running our projects on both Kaggle’s kernels, Colab sessions, and our own machines allowed us to run a lot of experiments, sometimes simultaneously, without having to worry about Kaggle’s limited GPU quota. The class binning method and the BCELogitsLoss function mentioned in this forum post, <https://www.kaggle.com/c/prostate-cancer-grade-assessment/discussion/155424> , really helped us to stay competitive. Blending the probabilities from multiple models gave a nice boost in performance and allowed us to merge some of our projects. The python library Cupy also turned out to be an invaluable tool for preprocessing.

What didn’t work: Certain experiments we ran gave poor results due to low batch size in training (which resulted in some false conclusions early on in the competition). Cycling loss functions caused the models to overfit wildly. In many cases unfreezing pretrained models to train early layers caused overfitting which would not be visible until running the model on test data.

What we could have improved: We did not include Efficientnet or Resnext in our final submission as we were not able to get good performance out of those models (many other teams did extremely well with those models). Late in the competition we found some weird quirks with training models on images of a particular size, then at inference using a different size which the model hadn’t been trained on, there were some positive results that warranted further exploration. While blending models (averaging the results on many models) worked well, we never tested model stacking (using the predictions of many models as the input of a final model). A few of the notebooks could have used better documentation as much of the code was not straightforward enough to skim through.

The most creative and novel approach in our group was striplication, however we didn’t manage to include it in the final submission. The early results of striplication were a bit too memory and time intensive to include in our submission. Towards the end of the competition, striplication had some promising results, but they required a team effort to implement and we were each too focused on our own projects to implement. A remaining untested hypothesis is that we could have trained a model on the striplicated images, then retrained the same model on images that had a computationally cheaper preprocessing method to retain the same performance for less compute on new images.

We had fun in this competition. We learned a few new things, we picked up some new tools, and even built some new tools we may bring into future competitions.