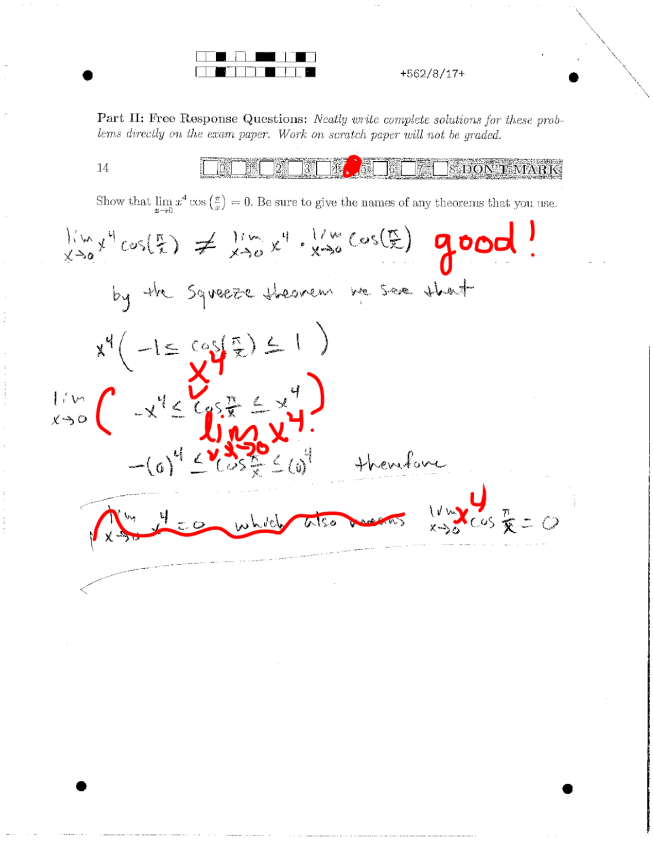
Marissa Graham
CS 501R Final Project Writeup

Dataset



I've been a TA for Math 112 and 113 for five semesters, and my entire time as a TA, the grading for midterms and exams has been electronic. The students answer their free response questions on specially annotated pages, with a bar on the top of boxes that we use to give them a score and a barcode used to match students with their individual pages, as shown on the right. Each question is assigned 1-3 instructors, who jointly write a rubric and each receive a PDF of about 300-500 samples of the question to grade. These are then read by some deterministic software and compiled into a feedback PDF for each student.

Since each page has consistent features and a recognizable "true label" (actually two labels--the correct score and the total points possible), this seemed like a promising classification problem. After consulting with the FERPA office and calculus coordinator, we determined that the use of student exams does not violate their privacy, since there is no identifying student information on each individual page, and I received access to the department's share folders containing graded exams from previous semesters.

I sifted through the folders to find folder names with a consistent enough file naming structure that I could fetch only the graded files, and then compiled and hand-labeled the question number and maximum points possible for the 131 files within these that were larger than 5MB so I could have the maximum amount of data possible without having to do too much hand labeling. I then split these files into their constituent pages and saved each as a 200dpi resolution PNG file in a separate Drive folder that I could feed to a torchvision ImageFolder.

When loading these files as a Dataset, I resized to 820 x 640, center cropped to 640 x 512, and then resized to 2.5*H by W for a fixed H and W; in the getitem function, I only returned the first H pixels of the height dimension, so the final images used for training were size H x W. In all

cases, W = 2H. In total, I collected 36,766 images across seven different 112 exams from fall 2017 to fall 2018.

I also labeled about 1200 randomly selected images with the scores they received and stored those in a separate folder, which I used as a the directory for a smaller fully labeled dataset.

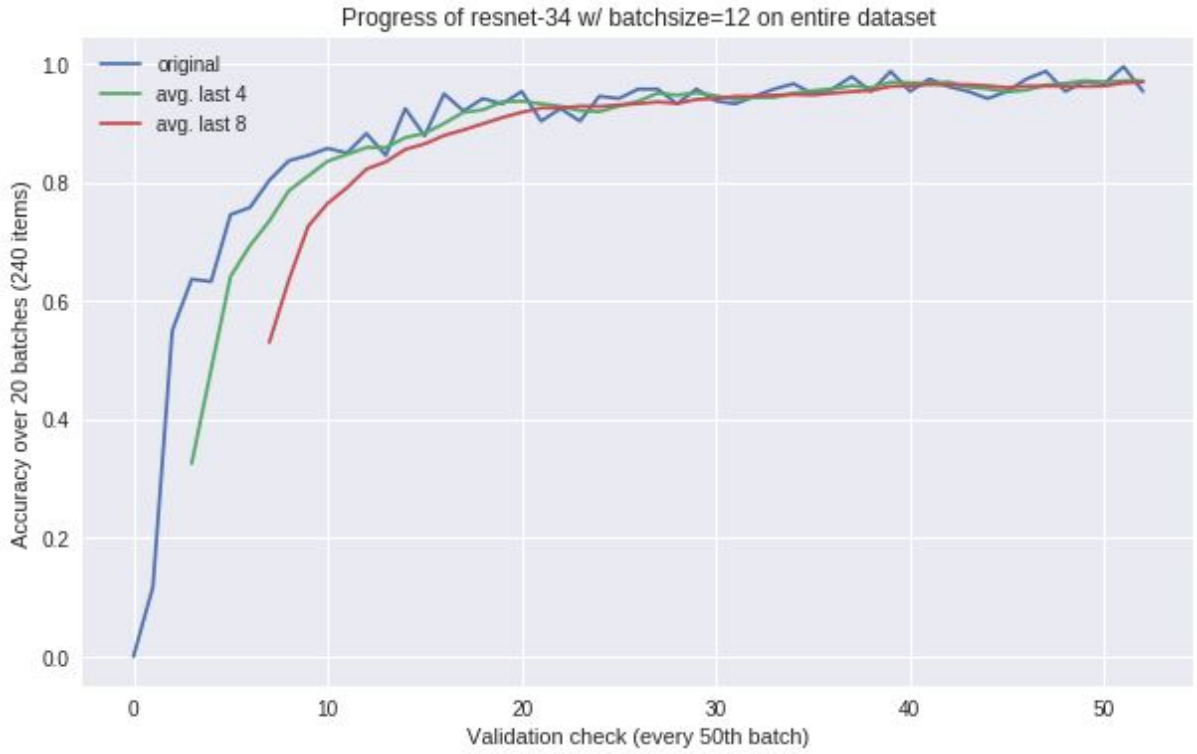
Instead of partitioning the dataset into training and test datasets, I used a RandomSampler that selected a different validation dataset every time (or a different subset of the full dataset for training, if I didn't use the entire dataset). This seemed to help with overfitting--the training results never got significantly better than the validation results in any experiment I ran.

Architecture Experimentation

I originally wanted to try a stock torchvision model, maybe with pretrained weights, but in order to use the image sizes I wanted I needed to reimplement the architecture. I looked at the papers for SqueezeNet, DenseNet, VGG, and ResNet, and chose to use a ResNet since it's a good all-purpose architecture that trains well.

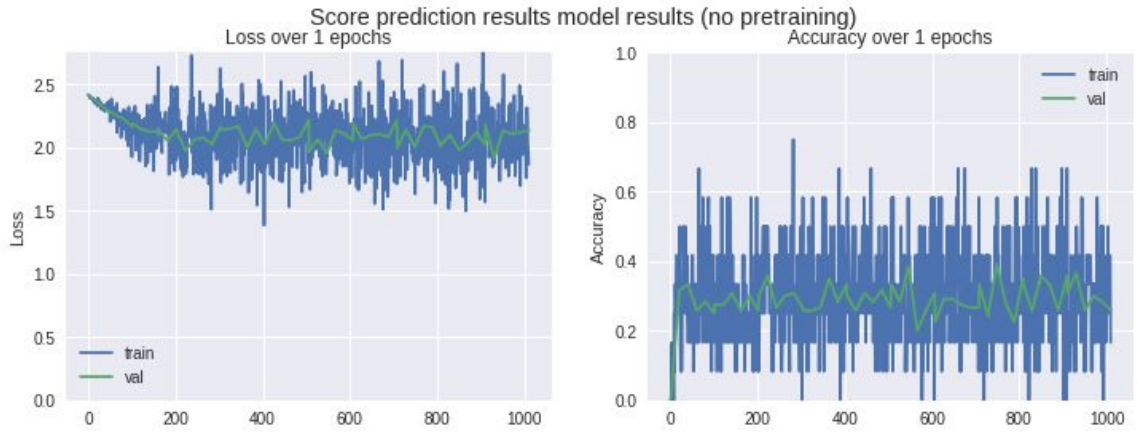
This turned out to be a really good choice--I did essentially no parameter tuning of the model itself, and it did very well. I used the Adam optimizer with a learning rate of 1e-5, and left all other parameters as their default value. I only tried a few different image sizes (32x64, 64x128, 128x256, 256x512) and batch sizes (2, 4, 8, 12, 16). I got better results per batch in less training time with 64x128 images, so I used those for all the rest of my results. Larger batches gave better, but past a batch size of 12, the GPU couldn't handle the amount of RAM necessary, so I used a batch size of 12 for all remaining tests. I tried both a ResNet-18 and ResNet-34. The ResNet-18 trained slightly faster with respect to the number of batches, but it seemed to be slightly less stable--once a certain accuracy level was reached, the ResNet-34 fluctuates less over further training (which you can see a bit in the plot below). Either model probably would have worked just fine, but I used a ResNet-34 for the bulk of my tests.

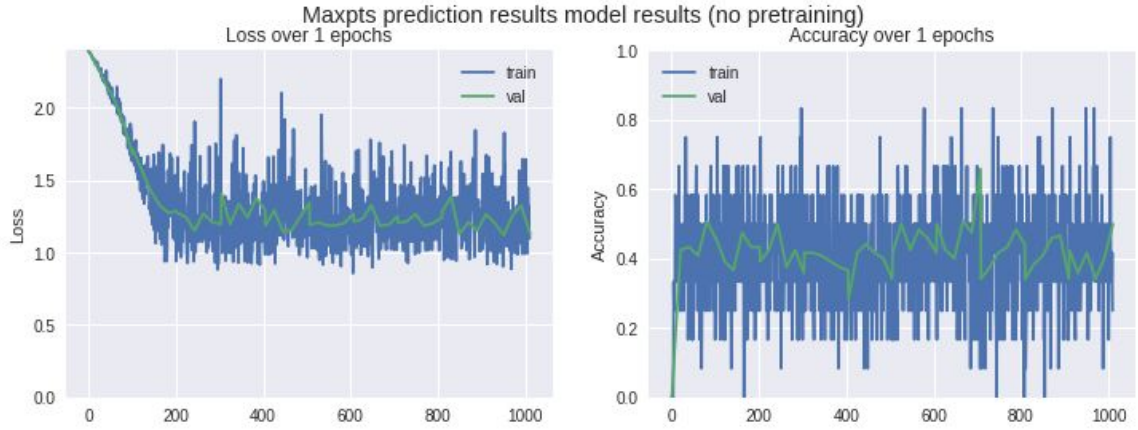
This plot is from a run that got interrupted before the network finished training on the entire dataset (there should be nine more validation checks), but I had kept the accuracy results at each validation check. The model got to a point where it never drops below 95% accuracy, and at one point got as high as 99.6% accuracy (meaning only one page was labeled incorrectly).

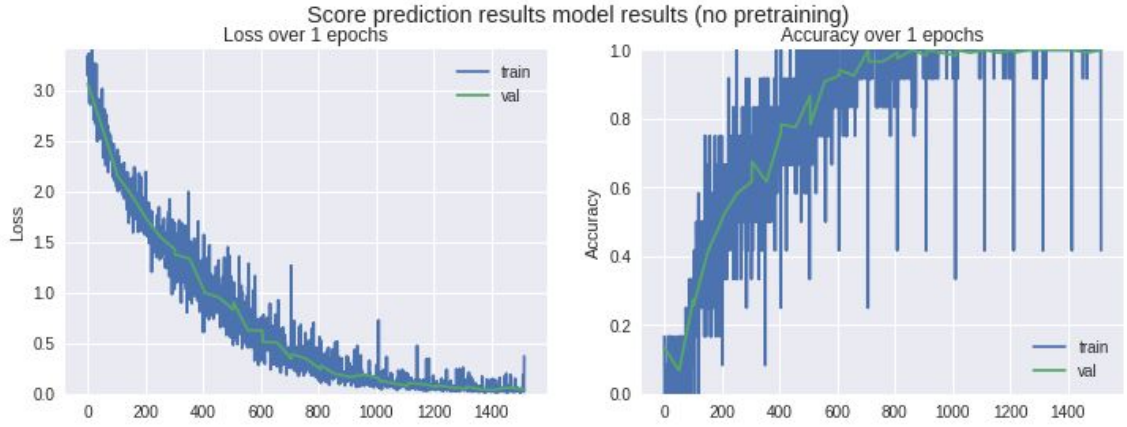


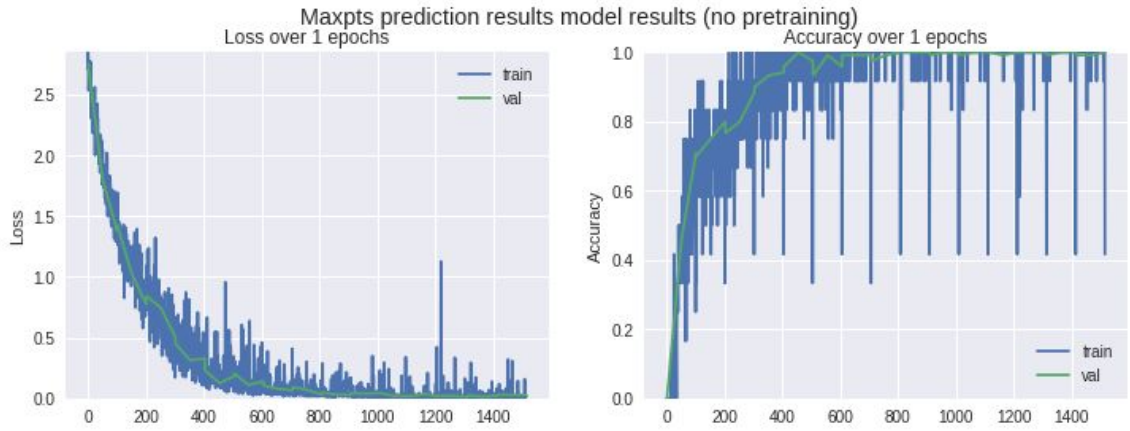
I also adapted the ConvNet architecture used for Lab 3 (MNIST dataset classification) because the problem was most similar to mine. It got to about ~45% accuracy for max score prediction and ~25-30% accuracy on my small dataset fairly quickly, but then never got any better than that. Meanwhile, my ResNet achieved a consistent 99%+ accuracy on the same dataset for both metrics within about 10 epochs (less than 40 minutes of training time). It's likely that ConvNet could have done better with some searching of the parameter space, but there's really no reason to do so when the ResNet works so well. I can therefore conclude that the reason I was able to get such good results is not that the problem is trivial, it's that the ResNet works really, really well.

Plots of these experiments are shown below, ConvNet first, then ResNet-34. Batch size 12 on 64x128 images with a 10% validation split in both cases.


Score prediction results model results (no pretraining)
Loss over 1 epochs
Loss
train
val
Accuracy over 1 epochs
Accuracy
train
val


Maxpts prediction results model results (no pretraining)
Loss over 1 epochs
Loss
train
val
Accuracy over 1 epochs
Accuracy
train
val


Score prediction results model results (no pretraining)
Loss over 1 epochs
Loss
train
val
Accuracy over 1 epochs
Accuracy
train
val



I saved the state_dict of any model with a better score prediction or max score prediction accuracy than previously seen. I ended up with models with a validation accuracy of 100% for both score prediction and max score prediction on my small dataset (likely some overfitting has occurred there), and 99.17% on the entire dataset. All saved models are for a ResNet-34 with an image size of 64 x 128, but can be used on either dataset (since only the weights are saved).

Experimentation with Pretrained Models

When I load and train a pretrained score and accuracy model, the results stay about as good as the validation score of the pretrained model. The same is true when I use max score prediction weights trained on the full dataset for max score prediction on the small dataset. Using a max score prediction model trained on the full dataset as the initial weights for a score prediction model results in an initial accuracy of around 45%, which normally takes about two epochs of training to reach when we start from scratch. This tells us that the features used to predict the total score are somewhat relevant to predicting the actual score, but not equivalently useful.

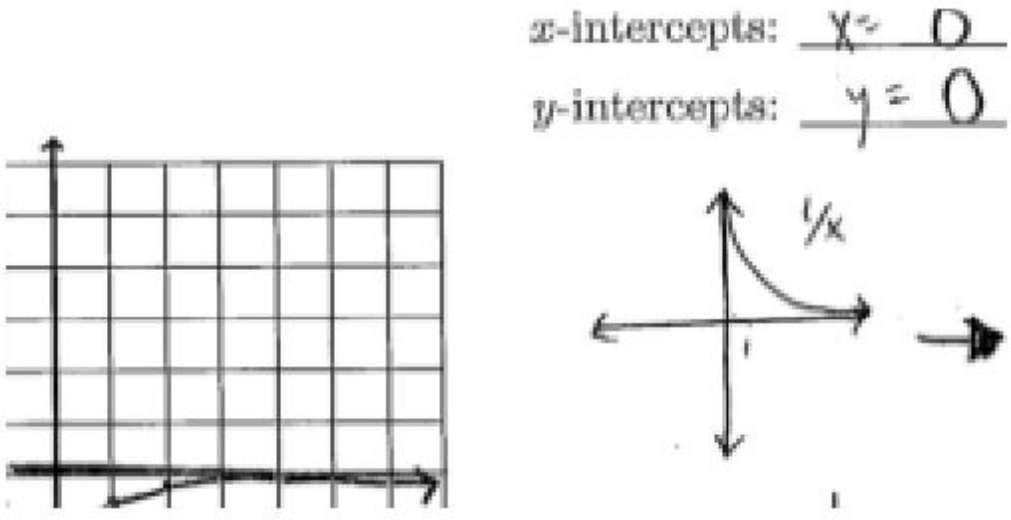
If we use a max score prediction model trained only on the small dataset to predict scores, initial accuracy is only about 25%. This is a little surprising, since about 60% of the points in my small dataset were questions for which the student received full credit; a completely accurate model for the number of points should have gotten 60% accuracy. Likely what is happening is that the model uses features besides the score bar to predict the true score (such as the specific question), more so for the small dataset, which has less variety. This small model is likely overfitted somewhat, but would be a good choice for initial weights in testing a larger dataset.

The high accuracy of the model means it has potential as a replacement for the actual software used to score exams. If I could get labels for the entire dataset (which should exist somewhere, since they were scored at some point), and trained the entire 37k set for the 10 epochs it took for my small dataset to achieve 100% accuracy, the resulting model would likely be just as accurate. But even 99% accuracy is a problem if you don't know *which* questions are scored incorrectly. To help with this, instead of just taking the argmax of the model's results, we could

convert them to the confidence of the model in its answer, and see what confidence in its answer the model needs to have to get 100% accuracy in its guesses on a previously unseen validation dataset. Then we would only need to hand-check the data points where the model's prediction is below this confidence cutoff.

Challenges

- ImageMagick (which is the only way anybody on StackOverflow knows how to split up a PDF and convert it to PNG files) does NOT play nicely with Colab. You need to edit the permissions in the policy.xml file, but colab won't give it to you or let you edit it. I had to get it to print, edit it manually, put it in a triple quotes string, and reload it that way.
- Converting between all the different filenames was really annoying.
- Making all the data structures for reading in the labels from spreadsheets was also really annoying
- The only really annoying part of adapting ResNet was figuring out what size all the Conv2d's needed to be. The source code just uses the naked numbers for 224x224 images and it's not clear how the parameters relate. It was a good review of kernel dimensions, though.
- I had some trouble initially where the way I was doing the training/validation split, it wasn't shuffling on the first epoch. When you don't shuffle them, you get an entire batch of one person's single grading file, and it doesn't train nearly as well
- I had a big chunk of training done, and the accuracy of the max score prediction never got over about 90-95% after a really long time, which confused me because it shouldn't be that difficult of a problem. It turns out I had been center cropping the images at a stage that I should have been resizing them, so when I used images smaller than 256 x 512, the scoring bar was cropped out entirely.
    - Ex:



    - Given that, the 90% really is very impressive, since it was just recognizing the structure of the printed problem definition and students' written work on the page.

- The ‘no module named register_decoders’ PIL.Image bug cropped up again, but the solution I’d been using all semester stopped working, and so did restarting the kernel. Fixing that sucked up a couple of hours.
- The training time seems to depend far more on whether items in the dataset have already been seen than anything else. I got really confused during a single runtime session when my models kept iterating faster and faster despite using a larger resnet-34 architecture and larger batch sizes. This effect overshadows any other conclusions I might draw about the training time of different parameter combinations.
- Colab stopped saving my main test notebook at some point, so I don’t have the training plots for the experiments which produced my best models.