Rapid Diagnostic Test Prediction via Convolutional Neural Network

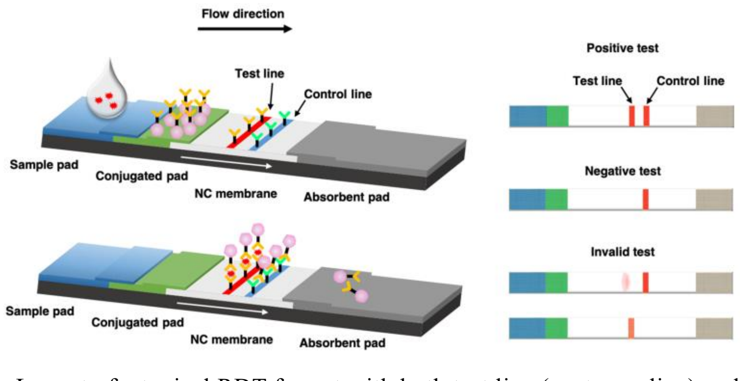
Daniel L. Leon
Department of Applied Mathematics
University of Washington
dleon@uw.edu

Abstract

The use of convolutional neural networks (CNNs) for the prediction of rapid diagnostic test (RDT) results is herein investigated. Photographs of rapid antigen tests for COVID-19, spiked with a relevant range of target recombinant antigen, were taken with several common smart phones under multiple lighting conditions to build training, validation, and test datasets for the network. The CCN architecture was based on AlexNet but was modified to accommodate the cropped test images with aspect ratio 4.3. In classification format, the CNN got 35% accuracy with the validation set after 20 epochs with cross entropy loss of 0.21. In regression format, the CNN achieved 46.8% accuracy with MSE loss of 0.022 on the test set.

1 Introduction

Rapid diagnostic devices are a type of molecular detection platform that can have a wide range of uses. From tests for detecting infectious diseases such as COVID-19, HIV, or Malaria; to home-pregnancy tests, tests for street-drugs, or tests for water impurities; the applications RDTs are wide and varied. The need for quick and accurate interpretation of RDTs at the point of need (PON) could be advanced via deep learning techniques and could have a strong positive impact on public and global health.



[1]

Figure 1: Layout of a typical RDT format with both test line (upstream line) and control line (downstream line). The sample, which can be in many forms, such as a nasal swab, is deposited at the left end of the figure. The sample rehydrates labeled nanoparticles that form a molecular stack on the test line in the presence of target analyte. The control line shows up regardless of the presence of target analyte, and if the control line does not show up, then the test is invalid.

Smartphones offer a practical method for image collection; however variable conditions present many challenges. These challenges include but aren't limited to the quality of the photos taken by any number of smartphone cameras, lighting conditions, orientation of the image, and time since the test was taken. With sufficient image preprocessing, training data, and neural net architecture, I hypothesize that accurate automated RDT interpretation is possible.

I had access to a dataset of over 2,000 rapid COVID-19 RDTs, which I was also involved in in generating in early 2021, thanks to Seattle based company Audere. The images were taken of RDT's with a series of tests using serial dilution of target antigen (Ag) from 10 ng/mL to 19.5×10^{-3} ng/mL, and included negative results (0 Ag) and invalid tests (no control line).

I used the classic convolutional neural network architecture of AlexNet [2] as my foundation, and I modified it to suit my non-square input images. The architecture consists of five convolutional layers and 3 fully connect linear layers. The original network was set up for classification of 1,000 types of images (e.g., tractors, cats, hamburgers, etc.) and I implemented it with the intention of predicting the ten concentrations, negative, and invalid, as twelve separate classes. Later I reconfigured the output and loss function to turn it into a regression problem. Under the classification format of the model, I achieved 35% accuracy on the validation set with cross-entropy loss value of 0.21 after 20 epochs of training. Under the regression format, I achieved 46.8% accuracy on the test set, with a mean-squared error loss of 0.022 on the test set after 30 training epochs.

2 Methods

2.1 Preprocessing images

The images came from five different smartphone: two models of each Apple iPhones and Samsung Galaxys, and a Google Pixel phone. These models were chosen to reflect the relative availability of the devices on the United States market, and they had variable output dimensions, but all around 4,000 px by 3,500 px. The relevant region of the images were all approximately centered in the images, as the phone app that was developed by Audere for capturing the images and metadata had fiducials in the view-screen to assist in capturing the image at the proper, or at least consistent, height from the target. Before the images were loaded into the custom dataloader, they were all center-cropped to uniform dimensions of 350 px by 1500 px. This area was considerably larger than the actual region of interest, but it gave enough slack to allow images to remain in the cropped image. The images all underwent normalization and auto-contrast correction, which helped significantly with the darkest of the images. I took 1,600 images for the training set (80% of 2,000), and the remaining 20% of images were evenly split between the cross-validation and testing sets.

2.2 Classification and regression models

The original implementation of the model used the twelve different target labels (ten concentrations from 19.5×10^{-3} ng/mL – 10 ng/mL [with serial dilutions starting from highest, diluting by half 9 successive times][1], negative target [0 ng/mL], and invalid test, [no test or control line development]) as classes to be classified using cross-entropy loss as the metric. Confusingly, the original classification labels for the model had both negative and invalids labeled as 0, then the highest concentration (10 ng/mL) was label 1, and the higher the classification label, the lower the actual concentration.

$$l_{CrossEntropy} = -\Sigma_{c=1}^{C} w_c \log \frac{\exp(x_{n,c})}{\sum_{i=1}^{C} \exp(x_{n,i})} y_{n,c}$$

To accommodate my problem statement from the original model, which has 1,000 output classes, I made the final layer output be 12. I also had to do some rearranging of the filter kernels, padding, and stride to accommodate the 4.3 aspect ratio of the test strip images, so that the input into the linear layers remained consistent with the original model, which was an arbitrary choice to not modify those.

Since the RDTs themselves have a range of Ag concentrations, it made sense that this model might perform better if it was in regression, rather than classification, format. This is because,

[1] Serial dilutions followed the pattern: $\left\{\frac{10}{2^n} \middle| n \in \mathbb{Z}^+, n < 10\right\}$

with classes, there is nothing in the loss function that predicts if two classes are close or far apart. So, learning makes more sense when considering mean-squared error loss.

$$l_{MSE} = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$

To accommodate regression, I chose the invalid label to be -1. Then I transformed the entire set using $\log_2(x+2)$ so that all the values were in the range $[0, \log_2(10+2) \approx 3.6]$.

The optimizer that I used for this model was stochastic gradient descent (SGD) with momentum and weight decay. I wanted to try other optimizers, like Adam (and its variants) and RMSprop, but those optimizers are more memory intensive than SGD. So, I was unable to used them with my memory limited GPU, which is discussed next.

2.3 Computational aspects of the model

One major benefit of using the AlexNet model was that I was able to fit everything onto my laptops 4 GB NVIDIA GeForce GTX 1650 Ti Max-Q GPU, but just barely. Originally, I was able to use batches of 8 images, but after switching my model to regression format, I was only able to fit batches of 4 images. I did find places where memory kept accumulating but was unable to fix it given the time constraints. I was also able to implement multithreading into my Pytorch dataloader, splitting batches amongst multiple threads. My laptop has an Intel Core i9vPro 10th generation CPU that supports 16 threads. These two optimizations combined gave a significant speed-up over simply running unoptimized on the CPU. For instance, I was able to run through an epoch in less than 5 minutes, including CV, on my optimized model; whereas a single epoch took an unknown amount of time greater than 20 minutes (I couldn't stand to wait to find out) when run strictly on the CPU.

2.4 Measuring model accuracy

Measuring the accuracy of the model was quite straightforward under the classification format since the model output is a likelihood probability for each class. We simply compute the argmax of the 1×12 model output, $\hat{y}_i$, and compare it to the actual label y_i.

The accuracy of the regression model required more thinking since the model output is a single value. In this case, the MSE loss is a good metric for how close you are to the actual concentration (class), but a metric to tell us how many predictions were accurate is not readily apparent. I created a function that compared each prediction to the full list of concentration labels, then took the absolute label that the prediction was closet to as the model prediction. That absolute label was then used as the comparison to the true label to compute the accuracy. The raw model output was still used in the MSE loss calculation for the model training.

3 Results

3.1 Classification format

In the classification format, the model was able to achieve 35% accuracy with the validation set after 20 epochs with cross entropy loss of 0.21 (See figures 2 and 3). I did not run the test set on this model because I switched things around to be in the regression format after consulting with the professor and TA, and upon coming back to it later, I had changed enough code that I was not able to re-code in the short time frame.

3.2 Regression format

In regression format, the CNN achieved a maximum of 46.8% accuracy with MSE loss of 0.022 on the test set. I did try several different activation functions in the model, and the best was when I replaced the basic ReLU functions with LeakyReLU ($\alpha = 0.05$) (See figs 4 & 5).

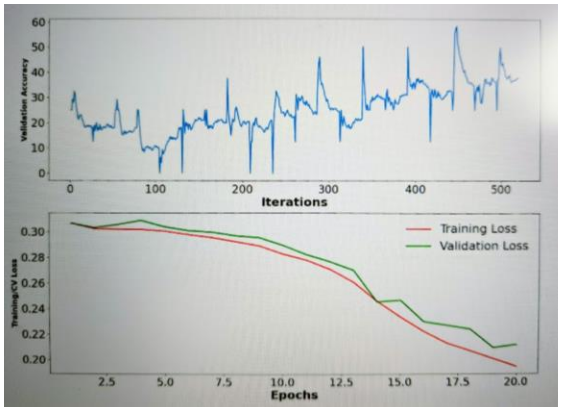


Figure 2: a) Batch validation accuracy that was computed every epoch. b) Training and validation loss, averaged across batches, computed for each epoch. A learning rate scheduler decreased the learning rate every 5 epochs, so it would seem like starting off with a lower learning rate would have been smart, but then I had to contend with vanishing gradients. Larger learning rates gave exploding gradients. It was a delicate balance.

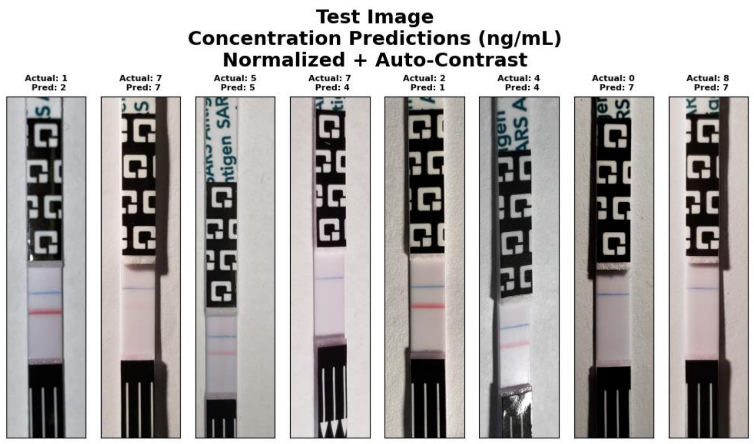


Figure 3: A sample of some of the predictions in classification format, with the actual class label above. In the subset of 8, 3 predictions were made correctly, but the farthest 2 on the right are actually pretty close, since class 8 (78.1×10^{-3} ng/mL) is about the visual limit of detection and class 0 is negative control.

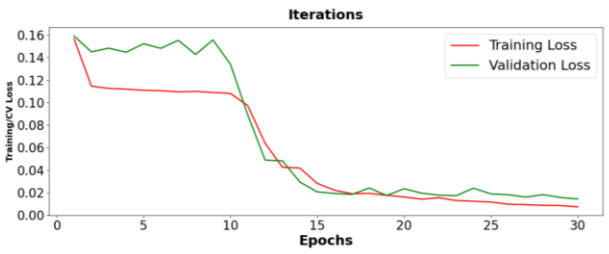


Figure 4: Training and validation loss curves for the model in regression format. Similar to in figure 2, there are considerable improvements after the learning rate scheduler sets in, implying that it might be better to simply start at that learning rate. My experience, however, was the same, as I was constantly combatting exploding and vanishing gradients.

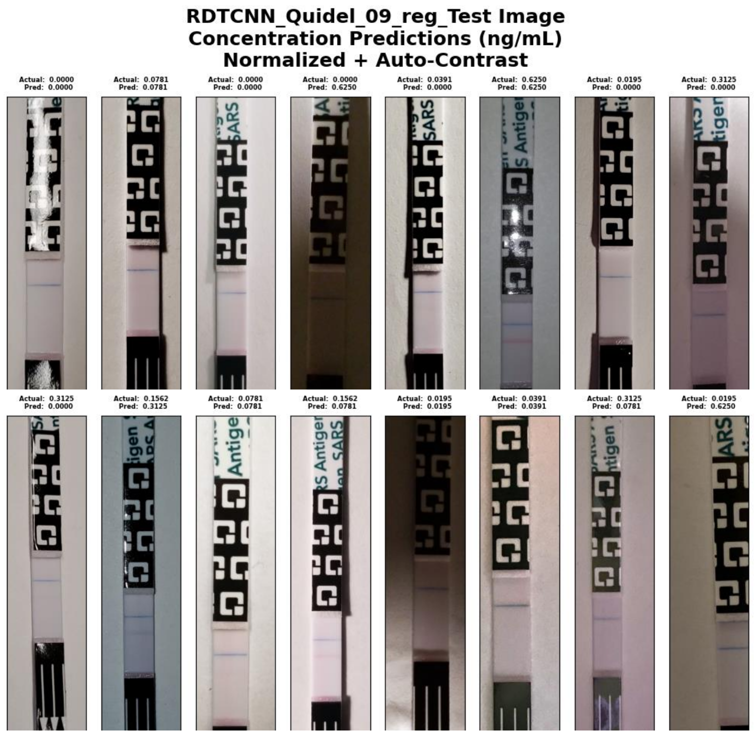

RDTCNN_Quidel_09_reg_Test Image
Concentration Predictions (ng/mL)
Normalized + Auto-Contrast
Actual: 0.0000 Pred: 0.0000
Actual: 0.0781 Pred: 0.0781
Actual: 0.0000 Pred: 0.0000
Actual: 0.0000 Pred: 0.6250
Actual: 0.0391 Pred: 0.0000
Actual: 0.6250 Pred: 0.6250
Actual: 0.0195 Pred: 0.0000
Actual: 0.3125 Pred: 0.0000
Actual: 0.3125 Pred: 0.0000
Actual: 0.1562 Pred: 0.3125
Actual: 0.0781 Pred: 0.0781
Actual: 0.1562 Pred: 0.0781
Actual: 0.0195 Pred: 0.0195
Actual: 0.0391 Pred: 0.0391
Actual: 0.3125 Pred: 0.0781
Actual: 0.0195 Pred: 0.6250

Figure 5: A sample of some of the predictions in regression format, with the actual class label above. In the subset of 16, 7 predictions were made correctly, and several more are close. I was particularly impressed with the accurate prediction on the dark image on the bottom row. These images also highlight some of the difficulty with the images. Dark images aren't uncommon, some have bright glare on them, some aren't centered with the uniform cropping that I applied.

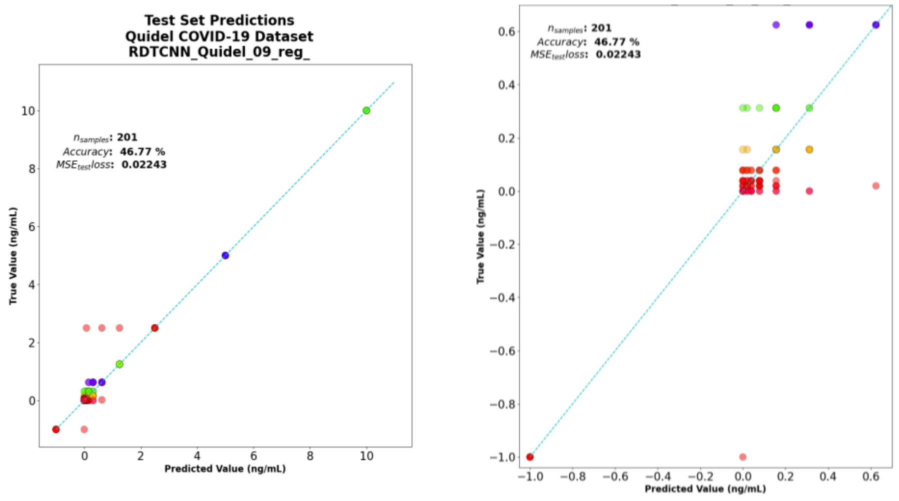


Figure 6: Plot of test predictions vs actual concentration. The transparency of the dots (α) was set to 0.25, so that darker spots indicate that several points are stacked on top of one another. a) This is the full range of concentrations, which includes the invalid results as -1. Note that the model predictions were 100% on the two highest, and the 4th highest (1.25 ng/mL) concentrations, but for some reason it struggled with that 3rd highest concentration. b) This is the zoomed in portion of the plot on the left. At this level, the predictions are less reliable, but promisingly, the invalid tests were mostly scored accurately. At the point where the dots are all red represents the point where human interpretation of the test line starts to fail.

4 Discussion

I have demonstrated that it is indeed possible to predict the RDT results using a CNN architecture, and that the regression format is preferable to the classification format. The dataset of images was specifically intended to simulate real-world difficulties that might be reflected in an actual implementation of this smartphone-based interpretation. Consequently, lighting played an important role, and I saw that the darker images could be more easily mis-identified, though in several cases, such as in figure 5, I showed that correct prediction is possible.

I investigated the filters in the first convolutional layer to try to get an idea about what the model was working with. Figure 7 shows that the test lines are capable of being enhanced, which is good. If I were to continue this work, I would look deeper into the network to get a fuller view of what is going on.

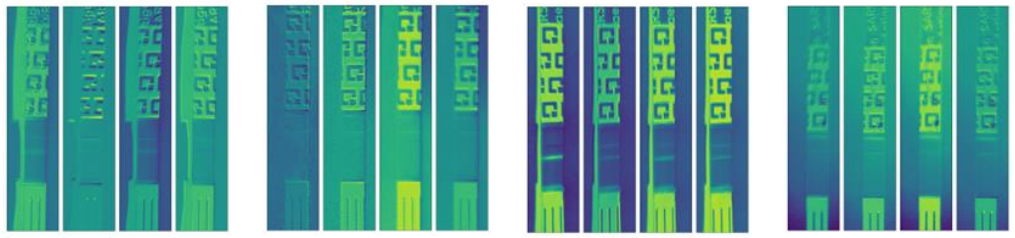


Figure 7: The first 16 filters in the first convolutional layer, shown with 4 different test images. The fist image has strong shadows that are highlighted in the filter. Positive test and control lines show up pretty strongly, but so do the RTD labels. The bright labels may not present an issue, since they are uniform across all RDTs.

I chose to work with AlexNet because it is a relatively small model that I knew would fit on my GPU, and I already had experience working with it. I do think that a more sophisticated model, like ResNet, would have better success. Now that my code is set up to work with cuda and multithreading, I would want to train on a cluster, or a better GPU with more memory, like an RTX-6000, which has 24 GB of GDDR6 memory. In addition to optimizing the model architecture, I would like to work with different optimizers to see how they compare to SGD.

In addition to refining the model, I think further investigation would greatly benefit from better image preprocessing. A large fraction of the cropped image pixels were superfluous to the region of interest, but due to the variability in images taken, and given the compressed timeline, the best that I could do was choosing a uniform image center-cropping that would extract the region of interest. By coming up with a more precise image cropping strategy, I could conceivably reduce the number of pixels per image from $\frac{350px\times 1{,}500px}{50px\times 400px} = 26.25$ times, which would be a huge saver in computational resources needed, and would cut out unnecessary artifacts that could distract the CNN. I would also work on better image adjustments, such as better contrast enhancement, better brightening of dark images, etc. All of these things could conceivably result in higher prediction accuracy and faster training times.

Additionally, I would like to improve my accuracy metric for the regression task. Currently, I only reported accuracy as it applied to exact predictions, but it would be useful to know *how far from actual* a given prediction was. The MSE sort of does this, but interpretation of the model performance would be enhanced by interpretation of this metric.

Lastly, I want to point out a flaw in the dataset, as it relates to the *invalid* samples. Invalid results in this context are classified by the lack of a control line, but not necessarily the lack of a test line. Though uncommon, this type of error can occur. When gathering the dataset, we simply took images of RDTs that had not been run, in order to avoid the automatic control line development. This captures one type of invalid result, but doesn't capture any information about cases where a test line develops, but the control line never does. With this particular RDT, the control and test line used separate nanoparticles (blue for control line, red for test line). One could simulate the scenario outlined above by carefully removing the controlnanoparticle conjugate pad (if it is separate) and re-assembling before running RDTs with varying amounts of target Ag. Another option would be to have the manufacturer provide RDTs that were produced without the control line. This would provide a more robust dataset that would take into account this particular type of error that could be expected to occur, albeit rarely.

5 Conclusions

I set out to see if CNNs could be used in the prediction of RDT results using smartphone images, and while the results weren't 100%, the dataset was difficult, but I now feel confident that CNNs are suited for the task. Under the classification format, the model was able to achieve 35% accuracy with the validation set after 20 epochs. Under the regression format, the model achieved a maximum of 46.8% accuracy with MSE loss of 0.022 on the test set.

This investigation convinced me that the task is a suitable task for a CNN. This same process could be further improved with the recommendations that I laid out, and be applied to future data sets, and eventually implemented with real patient tests.

Acknowledgments

Thanks to Professor Eli Shlizerman for his excellent instruction and mentoring throughout this spring 2022 quarter. Thanks to Ryan Vogt, for his superb work as our T.A. this quarter, and for his thoughtful help and insightful suggestions as they related to this project. Thanks to my fellow classmates, who offered support and comradery throughout the quarter. Special thanks to the people at Audere for letting me use their dataset. Special thanks to Barry Lutz, Jack-Henry Kotnik, and everyone in the Lutz research group. Lastly, special thanks to Diana Reimann, Aaron Reynolds, Bhavica Saxena, Lionel Tukei, who gathered this dataset in January and February of 2021.

References

(IEEE format. Sorry, couldn't get NeurIPS format to work)

[1] T.-N. L. T.-N. L. H.-H. K. H.-C. C. C.-C. L. N. S. C.-K. L. W.-H. C. Wesley Wei-Wen Hsiao, "Recent Advances in Novel Lateral Flow Technologies for Detection of COVID-19," *Biosensors 11.9,* 25 August 2021.

[2] I. S. G. E. H. Alex Krizhevsky, "ImageNet Classification with Deep Convolutional," *Advances in neural information processing systems,* vol. 25, 2012.