Histopathologic Cancer Detection Challenge

Elizabeth Cutting
Prof. Truang
HW5, Introduction to Artificial Intelligence Summer 2024 July 31, 2024

Introduction

Link to GitHub

Project Goal

The primary goal of this project is to use machine learning to classify images of metastatic cancer cells taken from pathology scans. This project is hosted on Kaggle as a competition, linked here. The Kaggle competition is scored by calculating the area under the ROC curve, which is a standard metric used to test the performance of a machine learning model. This report will document my attempt to participate in the competition by developing a convolutional neural network .

Dataset

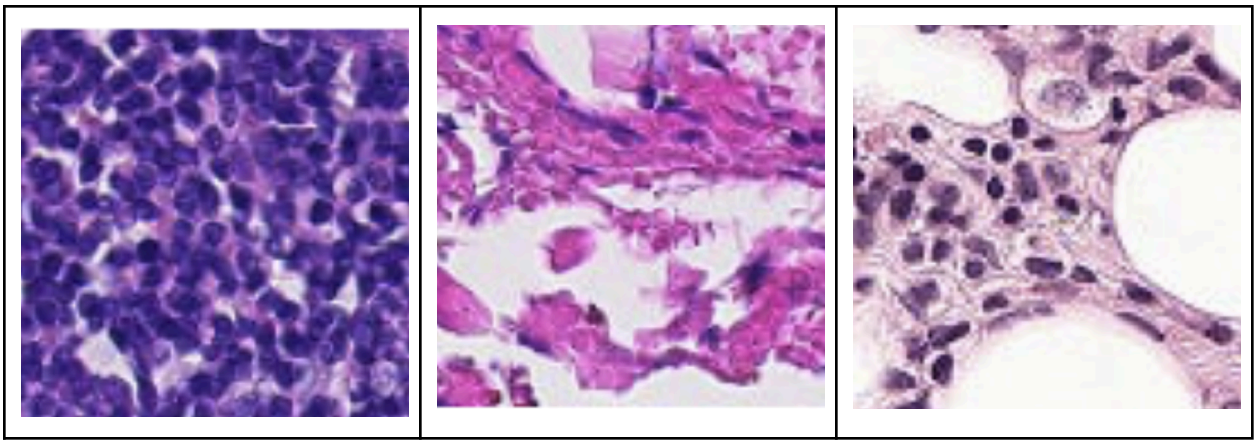
The data used in this competition is a modified version of the data from PCam. The data from PCam is taken via probabilistic sampling, and the dataset on Kaggle has been updated to remove duplicates. The data is split into a training set and a testing set. The labels for the testing set are not provided, and should be submitted and evaluated on Kaggle.

The shape of the data is as follows:

Training Set: 220,025 samples
Testing Set: 57,458 samples

The labels provided in the training set indicate whether or not the center 32 x 32 pixel region of a patch contains tumor tissue. Tumor tissue not in the center is not considered in the labeling process. If a pixel of tumor tissue is present, the label 1 is used to indicate the presence of a malignant tumor.

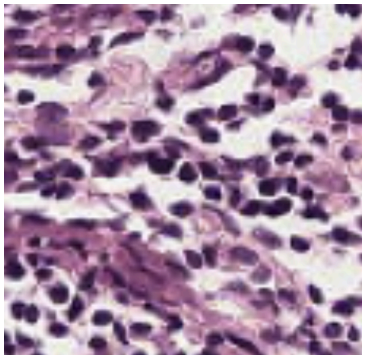
Below are a few examples of the types of images that the model learns from:



Data Exploration

One of the first things that I noticed upon observing the dataset is that there was a lot of variation between the different samples. They all were different colors and intensities, and when I observed them visually, I could not tell which samples would contain cancerous tissue and which were benign.

In the figure above, the leftmost image is benign (0), the center is benign(0), and the rightmost is malignant (1). Yet, the image below, which looks most closely like the rightmost image above, is also benign.



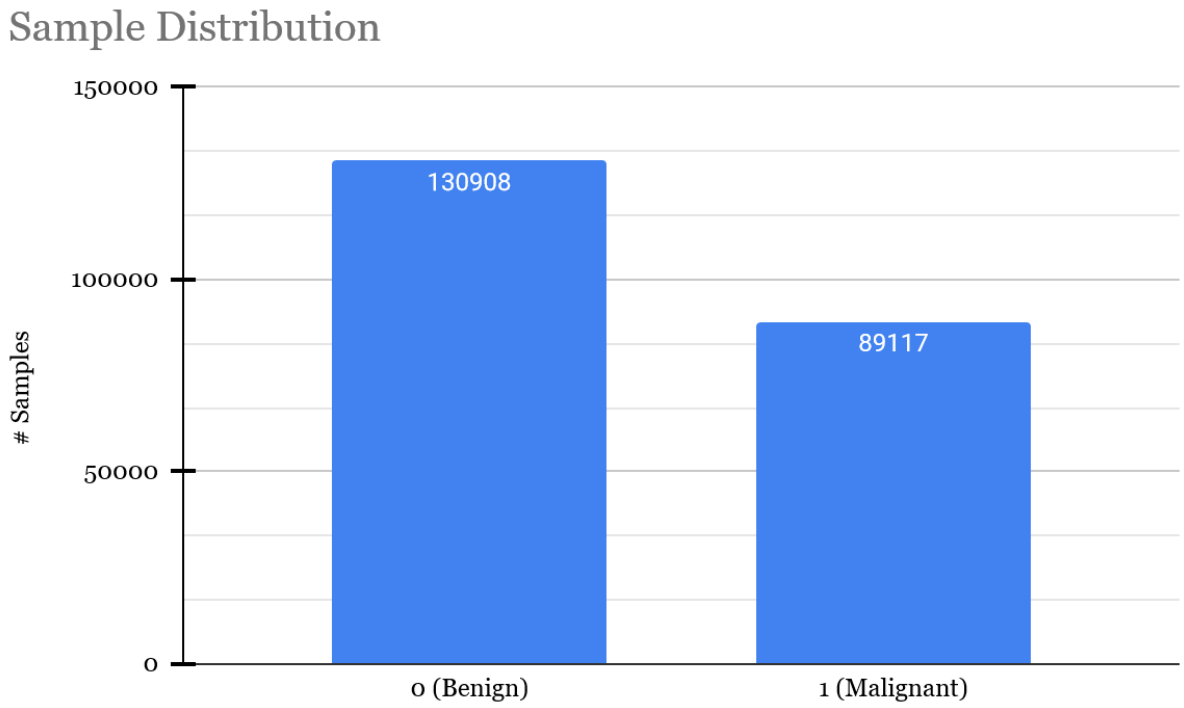
In order to begin the process of data cleaning, I first checked for any strange or missing data points. I imported the training labels, and I checked the labels for any value that was not 0 or 1, including missing values. Thankfully, there were none of these, which made the process very easy.

The next thing I looked at was the variability in the samples. I wanted to see if the dataset was balanced, since a balanced dataset informs the metrics that one can use to meaningfully assess performance. For example, if the dataset is very imbalanced, using accuracy as a metric would produce skewed results. Since the competition is evaluated

with an ROC curve, I decided that it would be most simple, straightforward, and appropriate if I can have a dataset that is as balanced as possible.

During my data exploration, I noticed that the size of the images presented a significant challenge: space and time. In order to download the data efficiently, I used the Kaggle API and unzipped the csv file in Google Colab. However, this took nearly half an hour. I planned on converting images to png files, which took a similar amount of time. This led me to leave out some of the data, which enabled me to create the perfectly balanced dataset I dreamed of.

Initially, I noticed that the dataset looked as follows, with 130,908 class 0 samples and 89,117 class 1 samples.



In order to select an appropriate amount of samples and simultaneously balance my dataset, I took the approach of randomly sampling 10,000 samples from each class, converting these images to png format, and then building my model around these data points. Before selecting, I shuffled the dataset, and then I filtered out the class in order to ensure I ended up with exactly 10,000 of each sample. This concluded my data cleaning, and from here I moved to constructing the model. After I developed a model that I believed performed very well, I went back and increased the amount of data that was used for each class, since I had the time and ability to wait for the images to be processed and the model to be fit. I ended up using 10,000 samples per class, which

helped my model's performance. For the final model, I used 25,000 samples per class to give the model the best change to learn from as much data as possible in a reasonable time frame.

Methods

Convolutional Neural Network

I decided to approach this project by building a convolutional neural network to process the images. I relied upon the `tensorflow` and `keras` libraries in python in order to construct the model, and I had to experiment in order to find suitable combinations of layers. Tuning these parameters was extremely challenging, and every time I needed to refit the model, I would need to allocate at least half an hour for training.

I developed many different CNNs throughout the process, and they all had a similar basic structure. First, I used a rescaling layer (documentation here), which helped with my images. Each network would have a few sets of a 2D convolutional layer followed by a pooling layer. I then used a combination of dense and flattening layers before finally obtaining the output. In some models, I included a dropout layer, but I ultimately did not always use them. They were included to reduce the potential of overfitting, but the models continued to work well even after removing them.

The differences between the models stemmed from two main sources: the main structure and the hyperparameters. As each model was developed, I experimented with various changes to the layout of the layers, and I also worked with several more specific parameters, including control tests to determine the best options for the dataset.

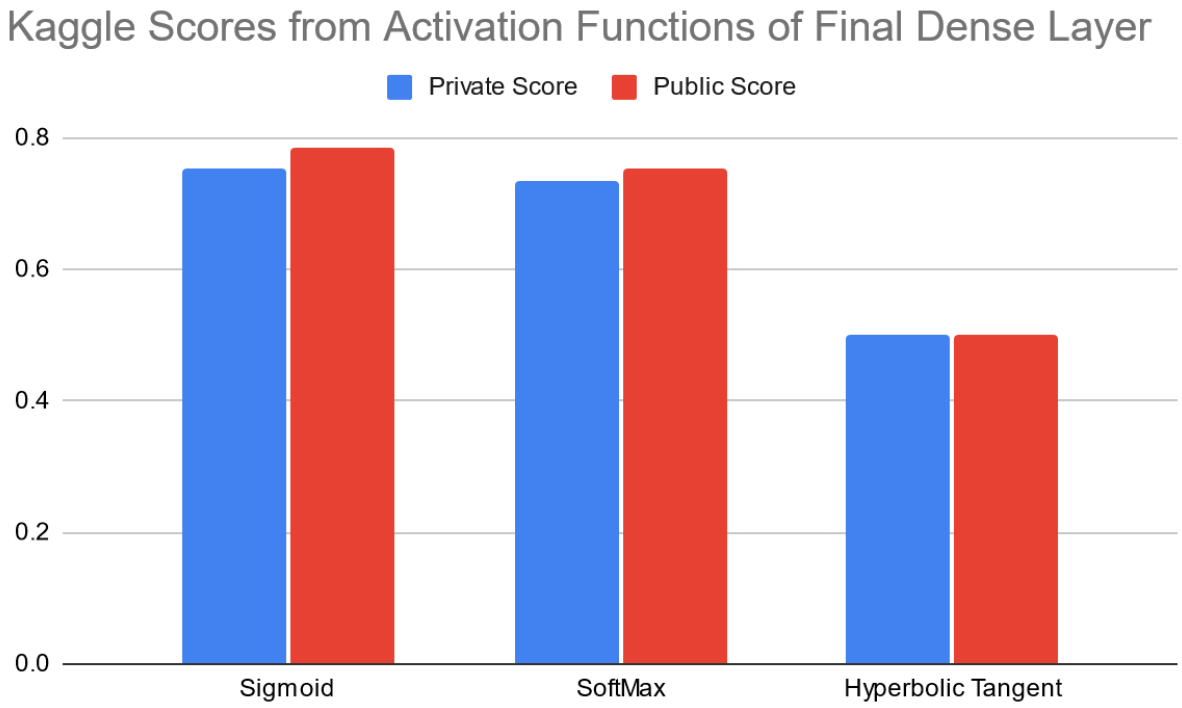
My Approach

Throughout the process of creating my model, I experienced overfit and poorly fit models. There was a lot of trial and error, I would have to experiment before using my knowledge of the various properties of different parameters to improve my model. I eventually arrived at the following parameters. There were some parameters that I did not change much throughout the process, but the ones I spent the most time on are outlined below.

Activation Functions

When deciding on which functions I should use, I ultimately decided to use ReLU on the hidden layers. My main reasoning for this was that many other functions, such as the

sigmoid function, have essentially 0 gradients at their extreme, which means that when back propagation occurs, the features might not have the desired effect. Since the extremes on the images are present in both malignant and benign cases, I assumed that this might translate to a potentially negative result when using this function as an activation function, and thus, I chose ReLU. For the last layer, I chose the sigmoid function because classifying extremes is actually more helpful here. After I had found a good optimizer, I decided that it would be worthwhile to spend more time on the activation function of the final layer. I was expecting either the sigmoid function or the softmax function to perform the best, and I was correct. My results from this test were as follows:



I was surprised to see that Hyperbolic Tangent performed so poorly, but I think this is an instance of a high bias low variance model that simply was too biased to perform well. SoftMax and Sigmoid were similar, which made sense. Sigmoid was ultimately better, most likely because it is the most well-balanced function, so I continued using it in future models.

Batch Size

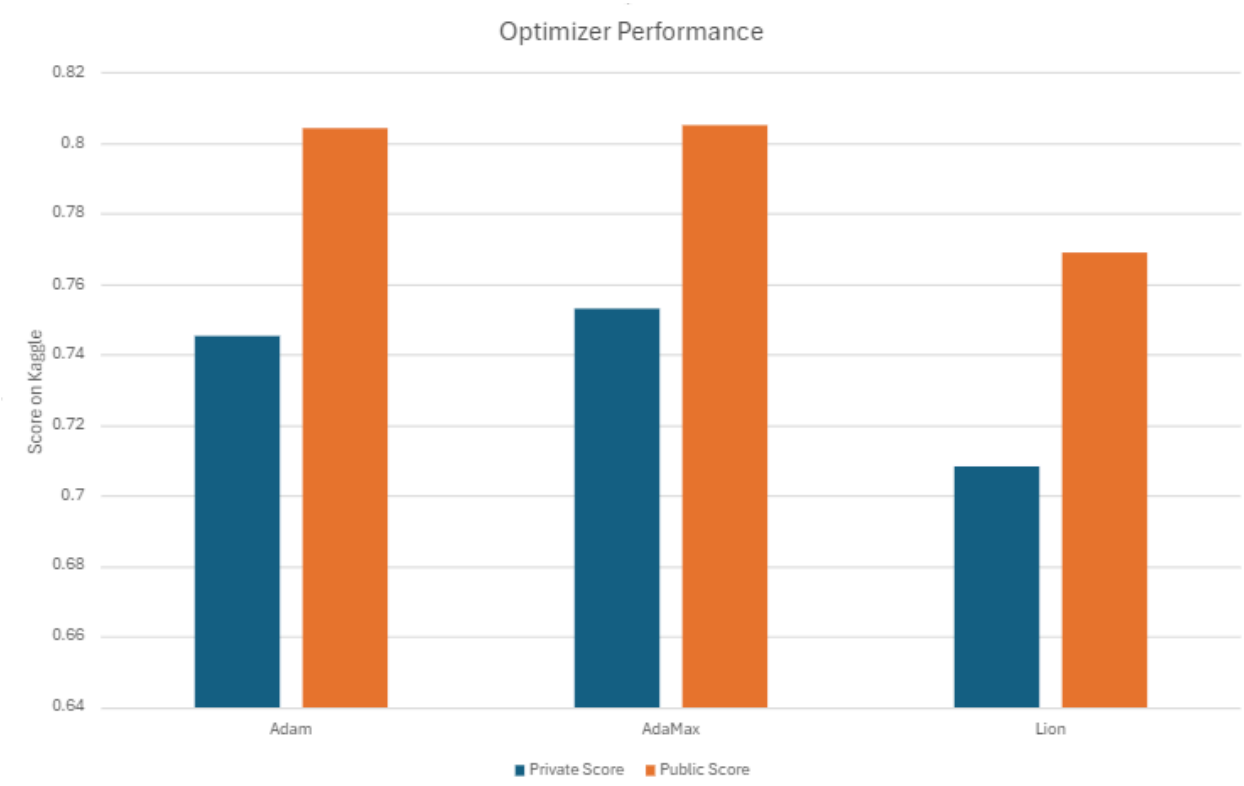
I used a batch size of 64, which is a bit larger than the default value of 32. My motivation behind this was mostly saving time, which was sorely needed with the slow speed of training. Also, given the high amount of variation within the samples, a larger batch size

might even be helpful for the model. I did experiment with batch sizes larger than 64, but since there were a few other factors I was also testing, I wasn't able to isolate it or determine how helpful updating this parameter was. 64 worked well for my models, so I did not test it much further. I tried a smaller size for the final step, but I did not see much difference. Since it took quite a long time, I chose to keep it at 64.

Optimizer

When choosing an optimizer for the model, I ran a simple test with my third model. This was one aspect of my model that I devoted a significant amount of time to perfecting. My third model was a solid model with a score of .7457 with the Adam optimizer, but I was determined to improve it. I decided to run a test with three different optimizers: Adam, AdaMax, and Lion. I chose these three since they seemed to have slightly different specialities and advantages, but all three did not have any disadvantage of being computationally expensive. I learned about the different optimizers and their properties here. The test was conducted with 10,000 samples in each class for 20,000 total samples, which was done to minimize how long the models trained for. I used 3 epochs and kept all other parameters the same.

The results were as follows:



Given the results I found above, I decided that it would be best for me to use the AdaMax optimizer. While there are many factors that influence which optimizer is the best, and AdaMax might not be the best in my final model, since it performed the best here, I decided to use it going forward.

For structuring the hyperparameters of the optimizer, I mostly focused on experimenting with the learning rate. I tried a few trial runs with different values, but I settled on 0.1 learning rate. This was standard, and it seemed to work the best for me with only one epoch.

Metrics

Since it was fairly simple to test different metrics, I opted for including more metrics rather than just a few. For the first few models, I just used accuracy, since I knew that this would be a respectable metric given my prebalanced dataset. I noticed that the accuracy score was given after each metric, and it was very interesting to see how it evolved over time. This made me curious about other metrics that I could look at, and so I decided to use Recall, Accuracy, and Cross Entropy, which I learned how to implement with the documentation here. I also knew that by looking at how these changed over time, I would be able to better understand my model's strengths and weaknesses. For example, if any factor leveled out after a certain number of epochs, I would know that I could decrease the number of epochs without losing any important aspects of my model. I chose to test recall because I know that it is a good test when the dataset is not balanced, which was not true of my set. Thus, I was not surprised when I noticed little from it. Thus, I decided to keep both Accuracy and Binary Cross Entropy, which is just Cross Entropy for binary classification.

Number of Epochs

Originally, I conducted tests with either 3 or 1 epochs, which was mostly done to save time. On some overfit models, I noticed that the accuracy per epoch leveled out and did not change. Since the accuracy was still increasing for some, I decided that in my final model, it would be worthwhile to include more epochs as well as more training data, since my accuracy would likely increase further. I did not experiment much with this however, so I do not know what the optimal choice was.

Challenges

The most difficult part of designing this model was the time. It took me a long time to figure out the best way to import the data into Google Colab, and every time I stopped, I would need to reupload the files. I also had to spend a significant amount of time training the models and waiting for the models to predict all 57,000 data points. I also struggled a bit with formatting the submission files and reading the files incorrectly, which resulted in me spending more time training and submitting to debug this part. This process became even more time-consuming after I ran out of GPU units on Google Colab. Due to time constraints, I did not spend as much time as I would have liked on developing the optimal model structure, because if a model was overfit or performed badly, I would have wasted a lot of time without learning much from my mistake. Instead, I focused more on optimizing the parameters of the model, which I believe was a good decision. For my final model, I made a few logical choices that I thought would improve the model, but it was difficult because I could not test these as much as I would have liked to. However, overall, I still had plenty of time to learn about many other aspects, and I thought this was a very valuable and enriching experience.

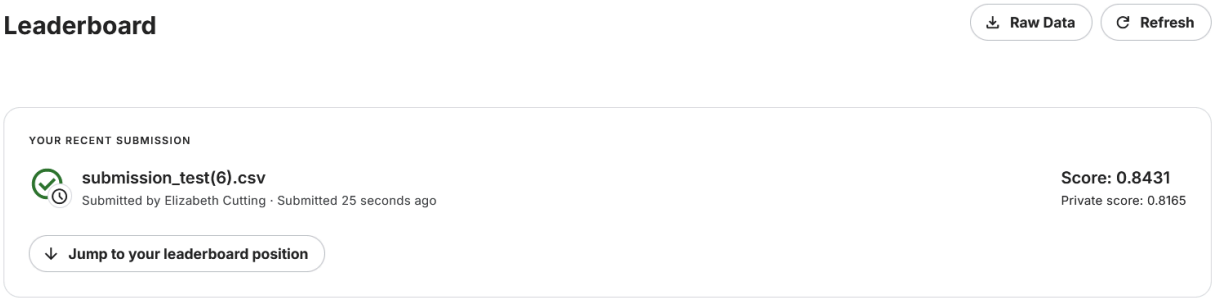
Results

Final Submission

My best model overall received a score of .8431 on the Kaggle competition. This model took the following structure:

```
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 96, 96, 3)         0

conv2d_3 (Conv2D)            (None, 94, 94, 32)        896

max_pooling2d_2 (MaxPoolin   (None, 47, 47, 32)        0
g2D)

conv2d_4 (Conv2D)            (None, 45, 45, 64)        18496

max_pooling2d_3 (MaxPoolin   (None, 22, 22, 64)        0
g2D)

conv2d_5 (Conv2D)            (None, 20, 20, 128)       73856

flatten_1 (Flatten)          (None, 51200)             0

dense_4 (Dense)              (None, 128)               6553728

dense_5 (Dense)              (None, 64)                8256

dense_6 (Dense)              (None, 32)                2080

dense_7 (Dense)              (None, 1)                 33

=================================================================
Total params: 6657345 (25.40 MB)
Trainable params: 6657345 (25.40 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

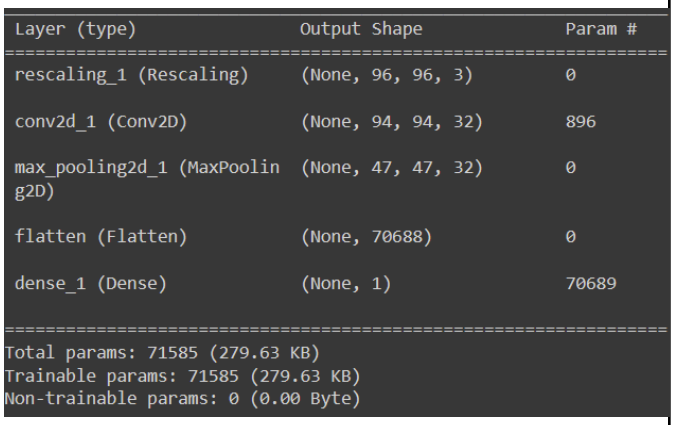
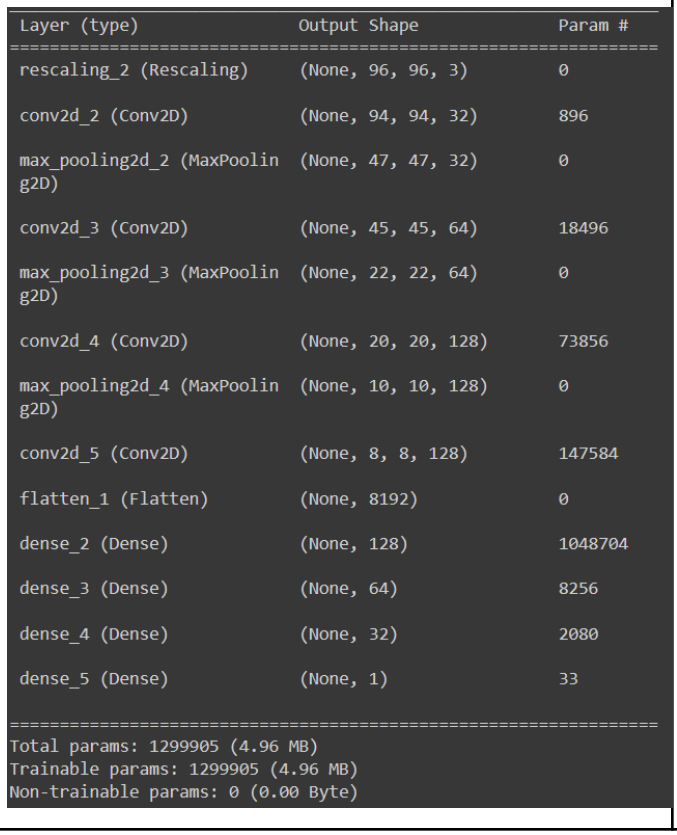
Submission:

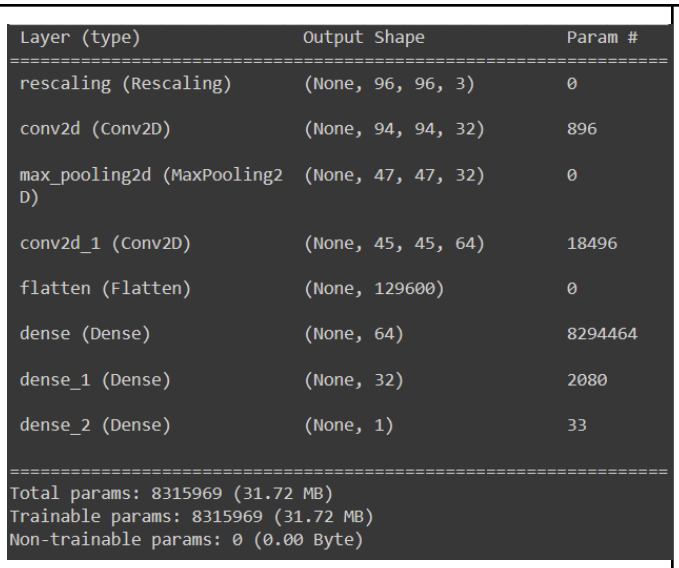
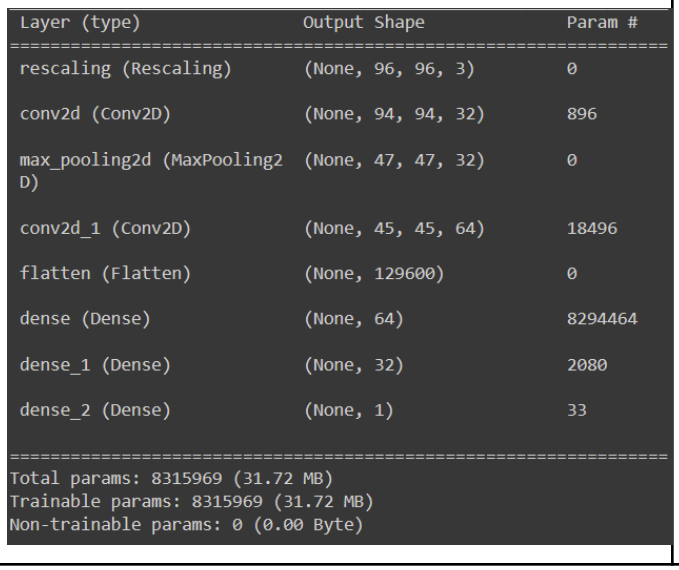


Progress

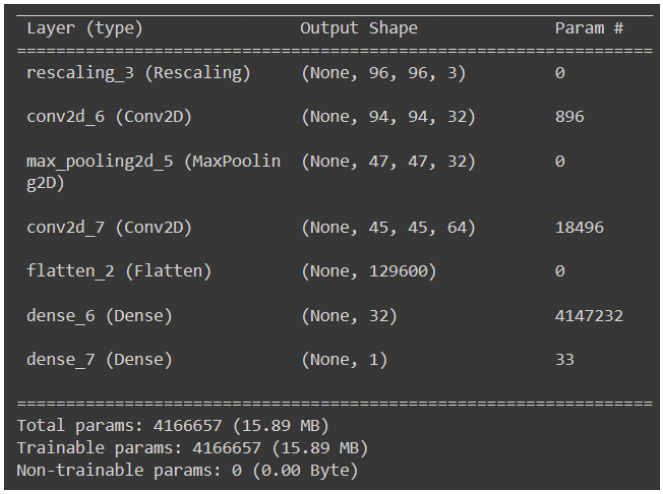
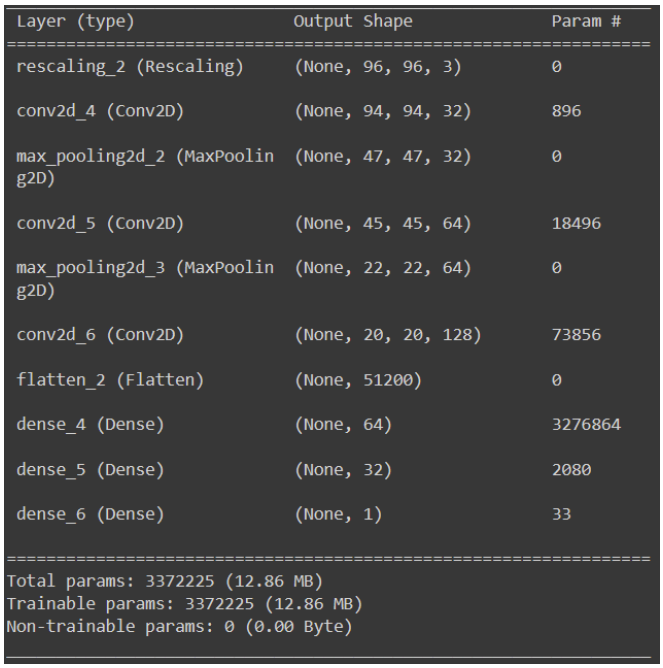
The path to get here was not quite as clean. Below is a table showing the model, the model summary, the amount of training samples, and the eventual score the model received. This does not include every model, but rather outlines the main models I developed and the changes along the way.

Model	Model Summary	Samples	Model Score

Number		Trained On	(ROC on Kaggle)
1 Simple Test		40,000	.7043
2 Complex & Overfit		20,000	.5000

3 Optimizer test - Adam		20,000	.7457
4 Optimizer test: AdaMax		20,000	.7532

5 Optimizer test: Lion	Layer (type) Output Shape Param # rescaling (Rescaling) (None, 96, 96, 3) 0 conv2d (Conv2D) (None, 94, 94, 32) 896 max_pooling2d (MaxPooling2D) (None, 47, 47, 32) 0 conv2d_1 (Conv2D) (None, 45, 45, 64) 18496 flatten (Flatten) (None, 129600) 0 dense (Dense) (None, 64) 8294464 dense_1 (Dense) (None, 32) 2080 dense_2 (Dense) (None, 1) 33 Total params: 8315969 (31.72 MB) Trainable params: 8315969 (31.72 MB) Non-trainable params: 0 (0.00 Byte)	20,000	.7082
6 Testing SoftMax Activation	Layer (type) Output Shape Param # rescaling_3 (Rescaling) (None, 96, 96, 3) 0 conv2d_6 (Conv2D) (None, 94, 94, 32) 896 max_pooling2d_5 (MaxPooling2D) (None, 47, 47, 32) 0 conv2d_7 (Conv2D) (None, 45, 45, 64) 18496 flatten_2 (Flatten) (None, 129600) 0 dense_6 (Dense) (None, 32) 4147232 dense_7 (Dense) (None, 1) 33 Total params: 4166657 (15.89 MB) Trainable params: 4166657 (15.89 MB) Non-trainable params: 0 (0.00 Byte)	20,000	.7354
7 Testing Hyperbolic Tangent Activation	Layer (type) Output Shape Param # rescaling_3 (Rescaling) (None, 96, 96, 3) 0 conv2d_6 (Conv2D) (None, 94, 94, 32) 896 max_pooling2d_5 (MaxPooling2D) (None, 47, 47, 32) 0 conv2d_7 (Conv2D) (None, 45, 45, 64) 18496 flatten_2 (Flatten) (None, 129600) 0 dense_6 (Dense) (None, 32) 4147232 dense_7 (Dense) (None, 1) 33 Total params: 4166657 (15.89 MB) Trainable params: 4166657 (15.89 MB) Non-trainable params: 0 (0.00 Byte)	20,000	0.5000

8 Testing Sigmoid Activation		20,000	.7527
9 Model used for minor parameter tests before final model (above)		50,000	.7821

I decided it would be best to start out with a simple model and do my best to refine and improve it by adding layers and testing parameters. This model worked fairly well, and, as is expected for its simplicity, it did not perform outstandingly. Nevertheless, it was a solid start, and I proceeded from there.

The second model that I created used far too many layers. My approach was that I should test out a complicated model and see how something with many layers performed after my simple model to best understand which factors most needed improvement. I used 4 2D convolutional layers, 3 pooling layers, 4 dense layers, and 1 flattening layer. I used ReLU activation on all layers, and all convolutional layers had 32 filters. Ultimately, this resulted in a model that almost always predicted 1. I am not sure

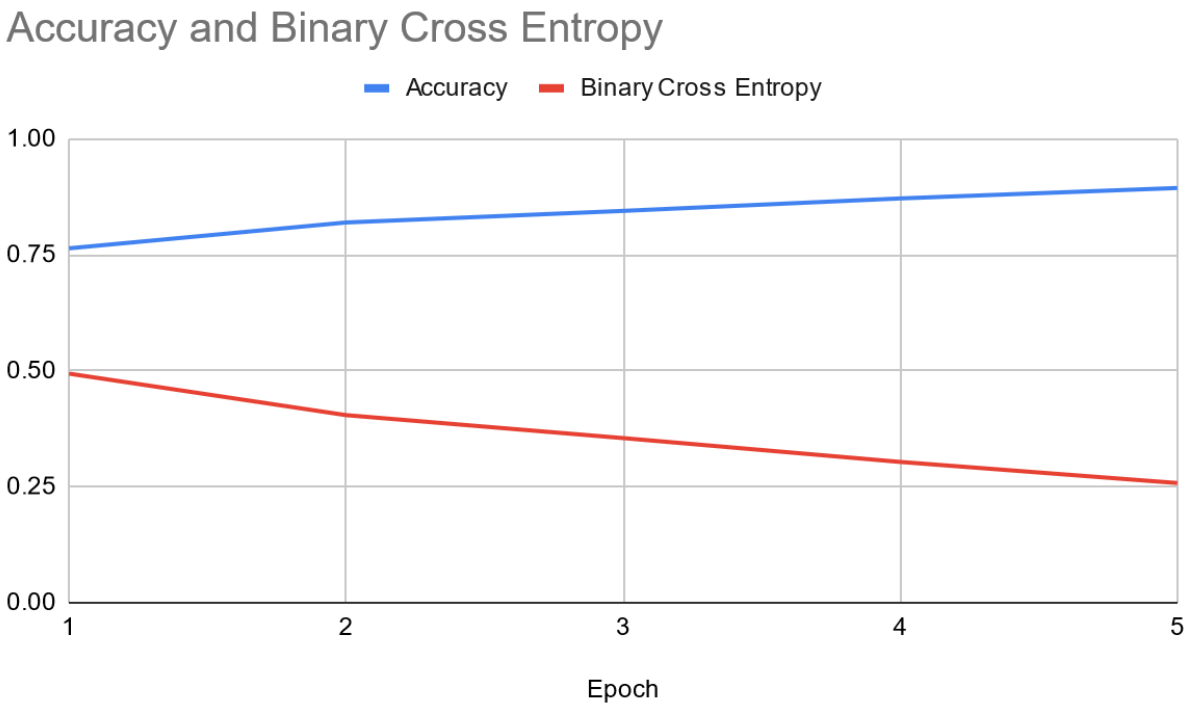
exactly why this occurred, but I know that the overfitting came from having an overly complex structure. I simplified my model and added a dropout layer in the next section.

After this, I found a much more stable model, which I then used as a benchmark for studying optimizers and their associated hyperparameters. After completing this process, I settled on using the AdaMax optimizer.

Next, I performed a test on a few activation functions for my final layer, since I noticed this had quite a large impact. I decided to keep ReLU for my convolutional layers, since this was working well and made sense logically due to ReLU's properties. After testing these layers, I found that the best option for my current model design was the sigmoid activation function.

From here, I ran a few small experiments with the batch size, number of epochs, and training data, but I only was able to change all three parameters at once. I settled with a batch size of 64, which was my original choice, since smaller sizes took too long. I also chose to use 5 epochs, which was more than my original 3. I did this because I saw that there was opportunity for the metrics to improve. After all of this, I put together my final model, which I was rather impressed with. It achieved 89% final accuracy, which I thought was a very solid number given my previous results.

Here is the model performance over all 5 epochs, considering both accuracy and binary cross entropy, which reflects loss:



Conclusion

This project enabled me to learn how to develop and design a convolutional neural network, and along the way, I was able to identify important parameters and learn how to best tune them for the success of the model. I was also able to learn how to properly evaluate a model's performance and apply many of the mathematical theories and foundations learned in class to a model with a valuable purpose.

If I had the opportunity to spend more time on this model, I would like to experiment with the size of the training data. Having the time to let my computer process more images could yield better results, and I am very interested in the correlation between model performance and amount of training data used. I would also like to try working with an imbalanced dataset, so I could learn how to best use some of the available metrics to help a model learn in this case. This is something that often occurs in life, and it is not as easy to simply balance the dataset as I did with this project. Finally, I would like to experiment more with model structure and the number of filters in the convolutional layers. I was not able to spend as much time with the structure because oftentimes, it was very easy to overfit the data, which meant wasting time training the models. Thus, I found a structure that was very solid and proceeded from there. However, since there is still much for me to learn in terms of building the model, I would enjoy spending more time down this avenue.

Most prominently, this project helps show the incredible power of deep learning in many important fields. If a simple model can have such great success, there are many further improvements that can be made to drastically improve public health and safety. The applications are almost limitless, and I find this to be very exciting and encouraging.