Predicting Flood Probability Using Geo-Spatial Environmental Indicators by Sequential Neural Networks

Rohan Keenoy
Dr. Badri Adhikari
University of Missouri - St.Louis

December 4th, 2024

Contents

Chapter 1

Abstract

Floods affect communities around the world. Climate change has been reshaping our rivers with changes in erosion, more rainfall, and several other factors.

Flood detection is important to first responders, as it is important to monitor river conditions, to identify where a flood may occur, and predict the likelihood of a flood in a certain community. Several technologies can be used to help mitigate risk to human life, infrastructure, and environment. Accurate flood prediction models can aid in preparedness and risk management - reducing the impact of the flood overall.

Chapter 2

Problem Statement

Create an efficient model based on the flood prediction dataset.

Chapter 3

Links

- **Link to my notebook:** Google Colab
- **Link to the data:** Flood Prediction Dataset
- **Link to the GitHub repository:** Flood Prediction

Chapter 4

Dataset

Originally, I was interested in using remote sensing tabular data for wildfire prediction, but ran into compute resource overhead issues. To link my interest of an environmental problem I selected the dataset called "Flood Prediction Dataset" on kaggle. The data set has 21 columns. As you will see below the data appears to be normalized already - it all follows pretty close to a normal distribution and is fairly equally distributed. The data source is unknown per kaggle user. They claim it was along a river in the southern hemisphere, but didn't leave any hints in the metadata of which particular river. Additionally, the data data appears to have been categorized up on rank on what may have been thresholding.

- **Hyperlink:** Flood Prediction Dataset
- The dataset contains 50,000 rows with no missing values. While the data curator was not clear about how the data was collected, the following variables are included (seen below) . Data did not require much cleaning except for adding a binary class for likelihood where likely is a probability less than or equal to 50 and not likely is a probability less than 49:

1. **MonsoonIntensity:** Higher volumes of rain during monsoons increase the probability of floods.
2. **TopographyDrainage:** The drainage capacity based on the region's topography. Efficient drainage can help drain rainwater and reduce the risk of floods.
3. **RiverManagement:** The quality and effectiveness of river management practices. Proper river management, including dredging and bank maintenance, can improve water flow and reduce floods.
4. **Deforestation:** The extent of deforestation in the area. Deforestation reduces the soil's ability to absorb water, increasing surface runoff and the risk of floods.
5. **Urbanization:** The level of urbanization in the region. Urban areas have impermeable surfaces (asphalt, concrete), which reduce water infiltration, raising the risk of floods.
6. **ClimateChange:** The impact of climate change on the region. Climate change can lead to more extreme precipitation patterns, including torrential rains that can cause floods.
7. **DamsQuality:** The quality and maintenance status of dams. Well-maintained dams can control floods, while dams with structural problems can break and cause catastrophic floods.
8. **Siltation:** The extent of siltation in rivers and reservoirs. The accumulation of sediments in rivers (siltation) reduces drainage capacity and increases the risk of floods.

9. **Agricultural Practices:** The types and sustainability of agricultural practices. The intensification of agriculture can lead to deforestation, excessive use of fertilizers and pesticides, and inappropriate irrigation practices, reducing soil biodiversity and increasing the risk of floods.

10. **Encroachments:** The degree of encroachment on flood plains and natural waterways. Construction in flood-prone areas impedes the natural flow of water and increases the risk of floods.

11. **IneffectiveDisasterPreparedness:** The lack of emergency plans, warning systems, and simulations increases the negative impact of floods.

12. **DrainageSystems:** Well-maintained and adequately sized drainage systems help drain rainwater and reduce the risk of floods.

13. **CoastalVulnerability:** Low-lying coastal areas are prone to flooding from storm surges and sea level rise.

14. **Landslides:** Steep slopes and unstable soils are more prone to landslides.

15. **Watersheds:** Regions with more watersheds may have a higher or lower risk of flooding, depending on various factors.

16. **DeterioratingInfrastructure:** Clogged culverts, damaged drainage channels, and other deficient infrastructure can increase the risk of floods.

17. **PopulationScore:** Densely populated areas can suffer more severe losses.

18. **WetlandLoss:** Wetlands act as natural sponges, absorbing excess water and helping to prevent floods.

19. **InadequatePlanning:** Urban planning that does not consider the risk of flooding increases the vulnerability of communities.

20. **PoliticalFactors:** Factors such as corruption and a lack of political will to invest in drainage infrastructure can make it difficult to manage flood risk.

21. **FloodProbability:** The overall probability of flooding in the region. This is the target variable for predictive analysis.

4.1 Visualization

Visualization and exploration included creation of histograms and summary statistics, it is clear that many of the columns have a normal or near normal distribution as highlighted below in Fig 3.1 and 3.2 .

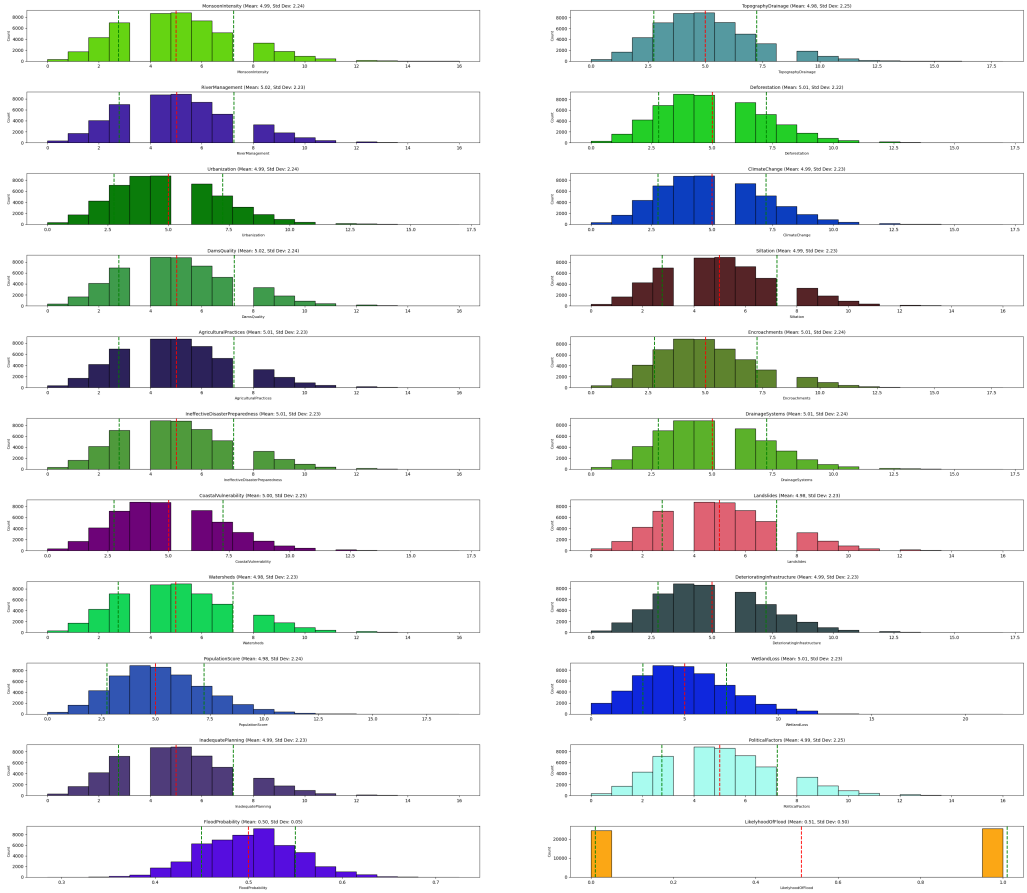

Figure 4.1: Histograms for Each Column

	count	mean	std	min	25%	50%	75%	max
MonsoonIntensity	50000	0.311968	0.139802	0	0.1875	0.3125	0.375	1
TopographyDrainage	50000	0.276894	0.124805	0	0.166667	0.277778	0.333333	1
RiverManagement	50000	0.313496	0.139457	0	0.1875	0.3125	0.375	1
Deforestation	50000	0.294616	0.13075	0	0.176471	0.294118	0.352941	1
Urbanization	50000	0.293474	0.131951	0	0.176471	0.294118	0.352941	1
ClimateChange	50000	0.293432	0.130986	0	0.176471	0.294118	0.352941	1
DamsQuality	50000	0.31346	0.140312	0	0.1875	0.3125	0.375	1
Siltation	50000	0.311787	0.13954	0	0.1875	0.3125	0.375	1
AgriculturalPractices	50000	0.312883	0.139662	0	0.1875	0.3125	0.375	1
Encroachments	50000	0.278132	0.124535	0	0.166667	0.277778	0.333333	1
IneffectiveDisasterPreparedness	50000	0.312814	0.13913	0	0.1875	0.3125	0.375	1
DrainageSystems	50000	0.294474	0.131653	0	0.176471	0.294118	0.352941	1
CoastalVulnerability	50000	0.294113	0.132182	0	0.176471	0.294118	0.352941	1
Landslides	50000	0.311514	0.139234	0	0.1875	0.3125	0.375	1
Watersheds	50000	0.311239	0.139512	0	0.1875	0.3125	0.375	1
DeterioratingInfrastructure	50000	0.293424	0.131243	0	0.176471	0.294118	0.352941	1
PopulationScore	50000	0.262367	0.117804	0	0.157895	0.263158	0.315789	1
WetlandLoss	50000	0.227505	0.101444	0	0.136364	0.227273	0.272727	1
InadequatePlanning	50000	0.312148	0.139376	0	0.1875	0.3125	0.375	1
PoliticalFactors	50000	0.311908	0.14038	0	0.1875	0.3125	0.375	1
FloodProbability	50000	0.487865	0.113715	0	0.409091	0.488636	0.568182	1
LikelyhoodOfFlood	50000	0.51026	0.4999	0	0	1	1	1

Figure 4.2: Summary Statistics

4.2 Distribution

Class distribution as likely to flood had a count of 25513. Count of not likely was 24,487. The calculated baseline 0.5108666666666667.

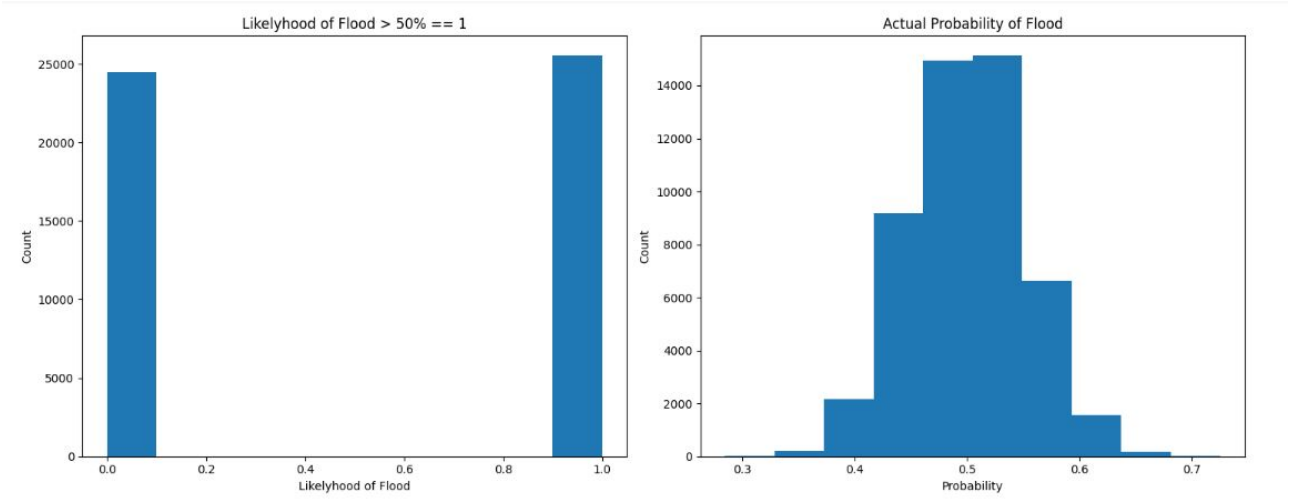

Figure 4.3: Class Distribution of Binary Liklihoods

4.3 Normalization

Training data was carefully normalized. I did not choose min-max normalization and instead opted to use feature scaling using the mean and std per column. To ensure no data leakage occurred, validation data used the mean and standard deviation of the training data after the train-validation split occurred (30 percent validation, 70 percent training). The formula is as follows:

```
def normalize(TRAIN, valid):
    means = TRAIN.mean(axis=0)
    std = TRAIN.std(axis=0)
    trainNorm = (TRAIN - means) / std
    validNorm = (valid - means) / std
    return trainNorm, validNorm
```

$$\text{trainXNorm} = \frac{\text{TRAINX} - \mu_{\text{TRAINX}}}{\sigma_{\text{TRAINX}}} \quad \text{and} \quad \text{validXNorm} = \frac{\text{validX} - \mu_{\text{TRAINX}}}{\sigma_{\text{TRAINX}}}$$

Chapter 5

Model-Selection and Evaluation

5.1 Train-Validation

Training and Validation was shuffled and then split. (30 percent validation, 70 percent training). Normalization occurred on this dataset for training. (see chapter 3)

5.2 Logistic Regression Baseline

This time using logistic regression on an architecture of 8-4-2-1 with activations all being sigmoid and the loss being binary cross entropy, optimizer rmsprop, and metrics for accuracy, a model was trained using properly split data. checkpoint and early stopping was used watching validation accuracy and having a patience of 5 (with weight restoration)

This produced the following curves:

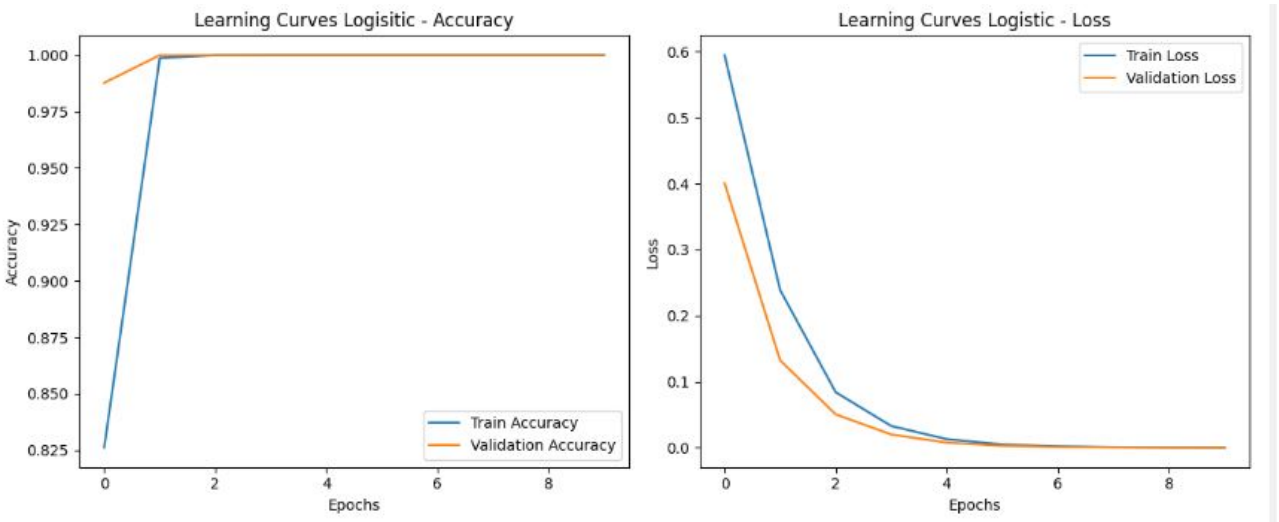

Figure 5.1: Logisitic Baseline Learning

5.3 Trying 6 Different Architectures

A dynamic function for model creation was used to generate 6 different architectures. They differ in the hyper parameters used such as neurons. Each layer had an activation of ReLu, besides the output layer which outputted as sigmoid. The code for the function is below where

it returns the model and is stored in a variable to be used later on. The models would be reduced to neurons per instance. The summaries of the model are below.

```
\def createAModel(neurons, layers):
    model = Sequential()
    model.add(Dense(neurons[0], input_dim=len(XTRAIN[0, :]),
       activation='relu'))
    for i in range(1, layers):
        model.add(Dense(neurons[i], activation='relu'))

    model.add(Dense(1, activation='sigmoid'))

    model.summary()

    model.compile(loss="binary_crossentropy", optimizer="adam",
       metrics=["accuracy"])
    return model
```

Layer (type)	Output Shape	Param #
dense_13 (Dense)	(None, 64)	1,408
dense_14 (Dense)	(None, 32)	2,080
dense_15 (Dense)	(None, 16)	528
dense_16 (Dense)	(None, 1)	17

Total params: 4,033 (15.75 KB)
Trainable params: 4,033 (15.75 KB)
Non-trainable params: 0 (0.00 B)
Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_17 (Dense)	(None, 32)	704
dense_18 (Dense)	(None, 16)	528
dense_19 (Dense)	(None, 8)	136
dense_20 (Dense)	(None, 1)	9

Total params: 1,377 (5.38 KB)
Trainable params: 1,377 (5.38 KB)
Non-trainable params: 0 (0.00 B)
Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_21 (Dense)	(None, 16)	352
dense_22 (Dense)	(None, 8)	136
dense_23 (Dense)	(None, 1)	9
dense_24 (Dense)	(None, 1)	2

Total params: 499 (1.95 KB)
Trainable params: 499 (1.95 KB)
Non-trainable params: 0 (0.00 B)

Figure 5.2: First 3 Larger Architectures

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_25 (Dense)	(None, 8)	176
dense_26 (Dense)	(None, 1)	9

Total params: 185 (740.00 B)
Trainable params: 185 (740.00 B)
Non-trainable params: 0 (0.00 B)

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 4)	88
dense_28 (Dense)	(None, 1)	5

Total params: 93 (372.00 B)
Trainable params: 93 (372.00 B)
Non-trainable params: 0 (0.00 B)

Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_29 (Dense)	(None, 2)	44
dense_30 (Dense)	(None, 1)	3

Total params: 47 (188.00 B)
Trainable params: 47 (188.00 B)
Non-trainable params: 0 (0.00 B)

Figure 5.3: Last 3 Smaller Architectures

5.4 64-32-16-8 Architecture

This architecture trained for 10 epochs before early stopping caused it to stop early. Validation accuracy and training accuracy of 1.0 and a loss of 0.000004 loss. It makes sense that this architecture learned so fast since it had 4,033 parameters.

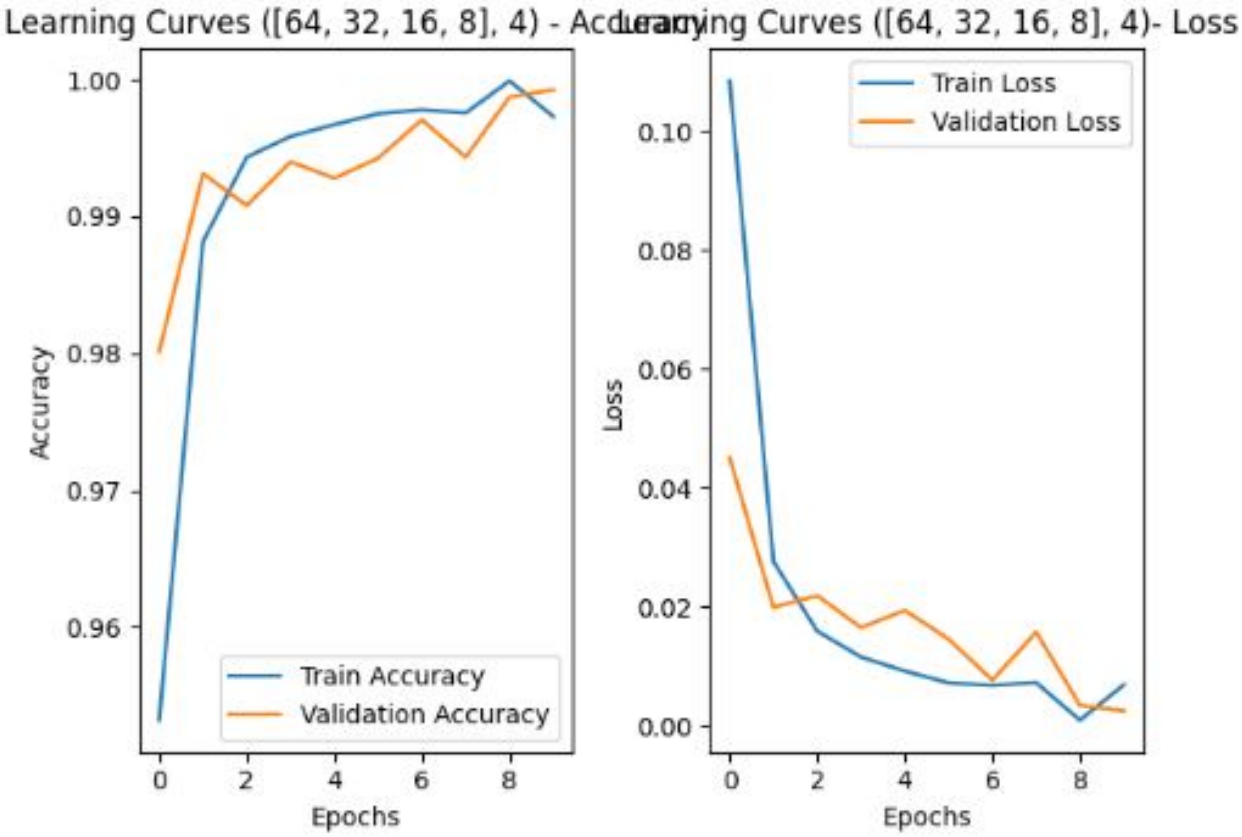

Figure 5.4: Learning curves for 64-32-16-8

5.5 32-16-8-1 Architecture

This architecture also trained in 10 epochs before early stopping stopped it, the curve for loss is nearly identical, however the accuracy varies from the 64 architecture. very similar results with a small loss and .99- 1.0 accuracy.

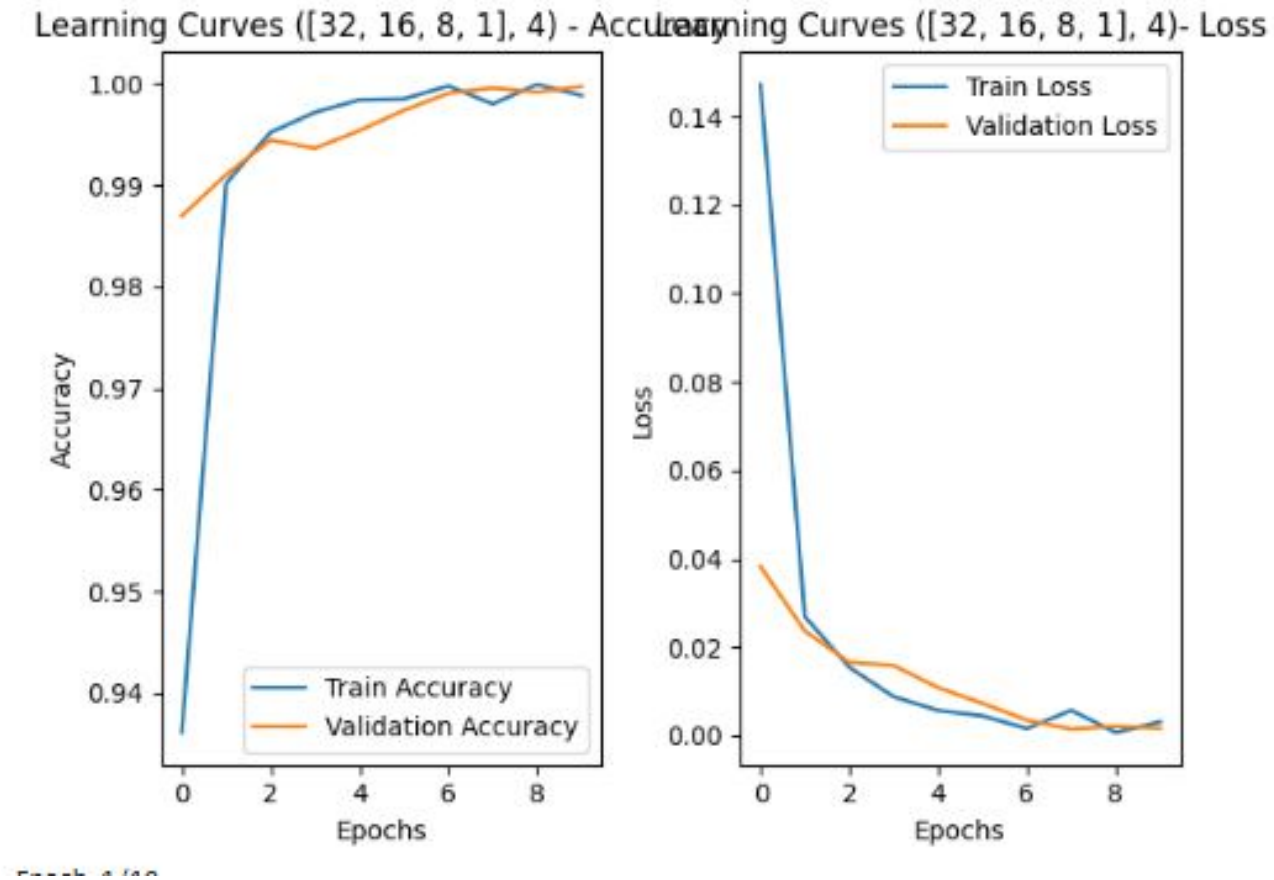

Figure 5.5: 32 Architecture

5.6 16-8-1 Architecture

The 16 architecture early stopped at 10 epochs with similar results, except that loss more gradually falls off. Accuracies are 1.0 and loss is larger than other architectures (articulated in the graph as well) at 0.0015 in the last epoch.

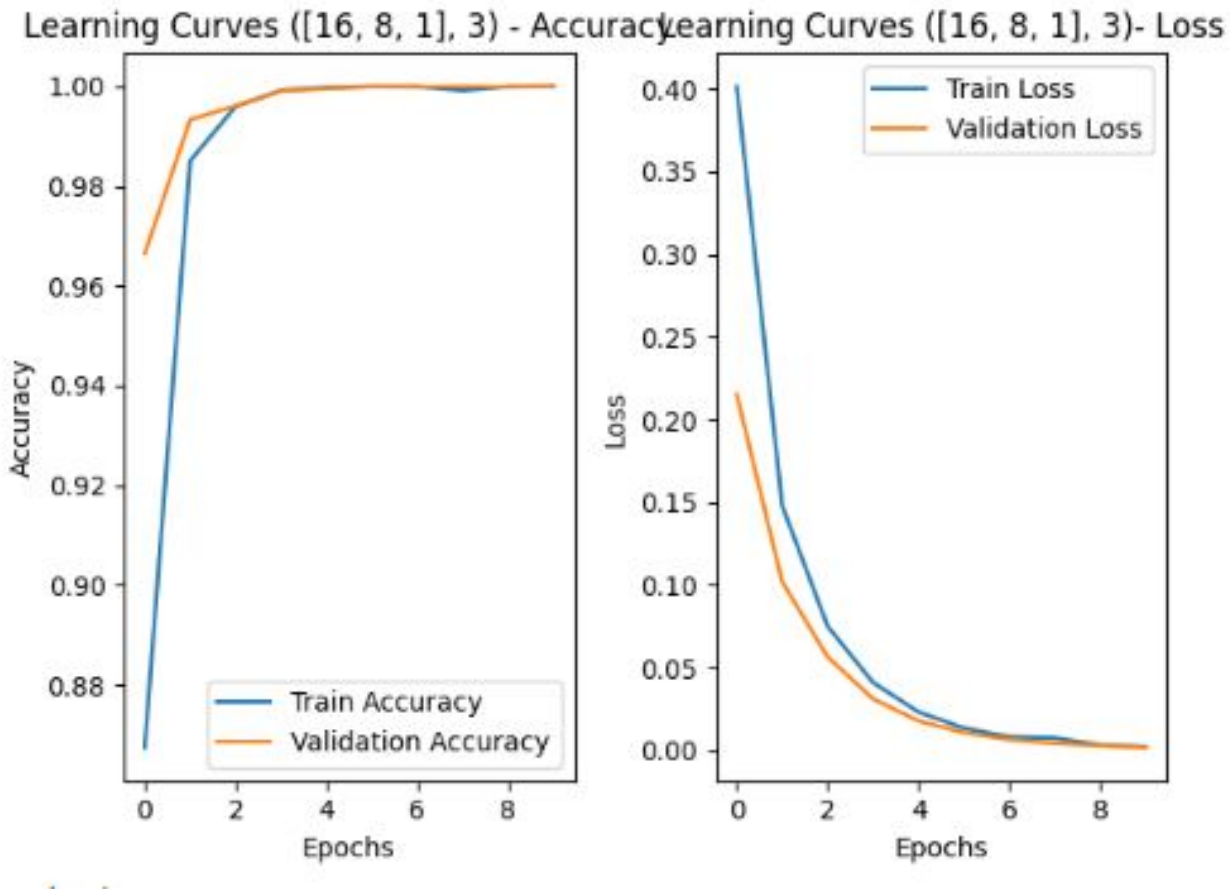

Figure 5.6: 16-8-1 Architecture

5.7 8-1 Architecture

The architecture is odd because again early stopped at 10 epochs, but the Loss curve looks more similar to the larger models that the 16 architecture. Accuracies are 1.0 and loss is very small at 0.00006 in the last epoch.

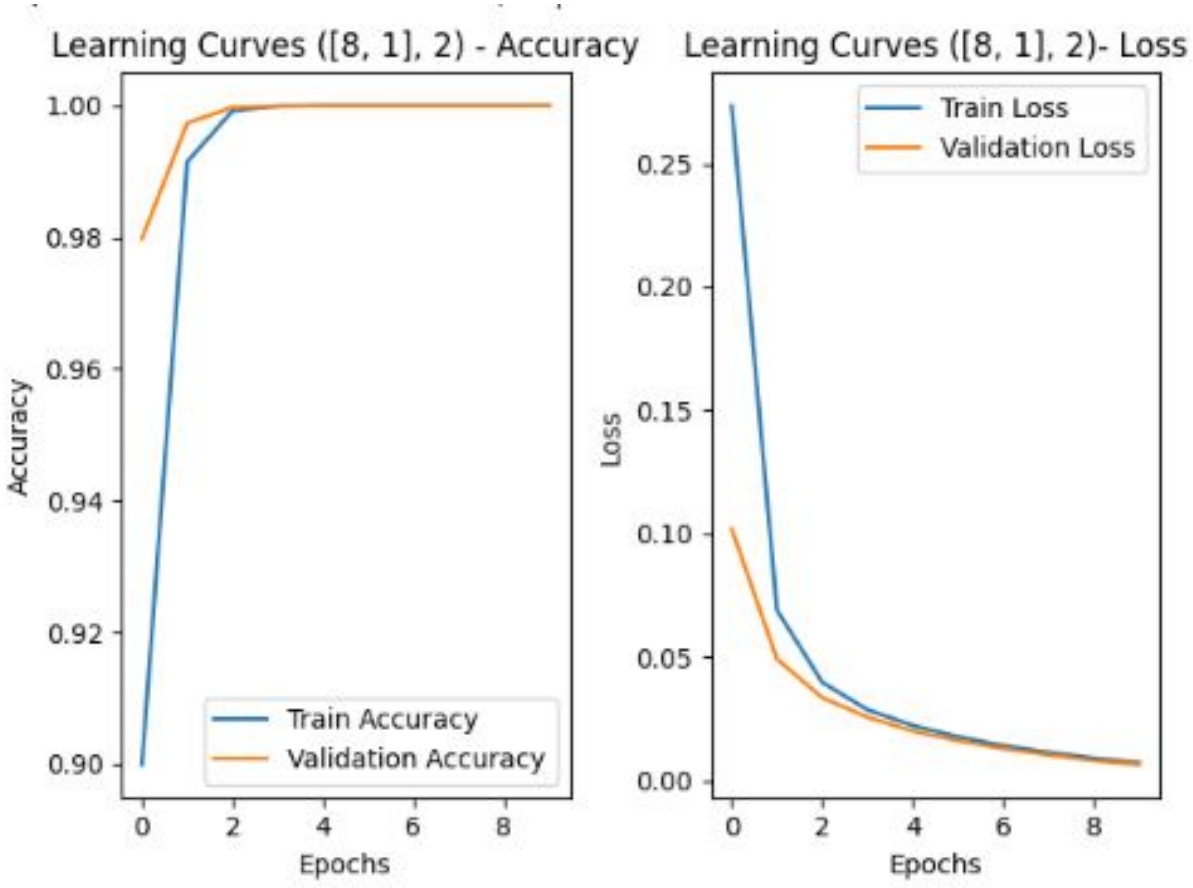

Figure 5.7: 8-1 Architecture

x

5.8 4-1 Architecture

The four architecture performs well too, early stopping at 10 epochs. The accuracies are 1.0 and has a very small loss of 0.0009.

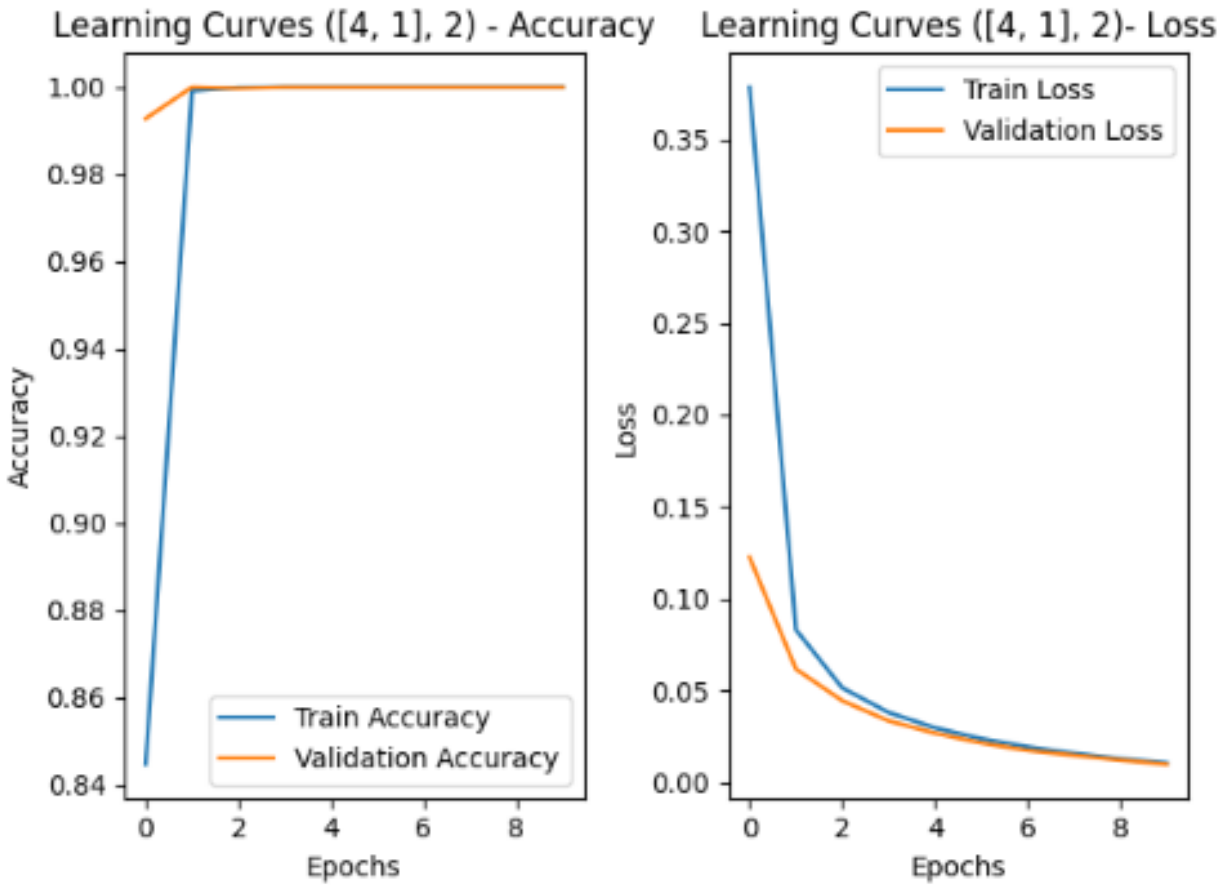

Figure 5.8: 4-1Architecture

5.9 2-1 Architecture

The 2 architecture performs well and early stops at 10 epochs. the loss is on the larger side at 0.0015 in the larger epochs. Accuracies are 1.0.

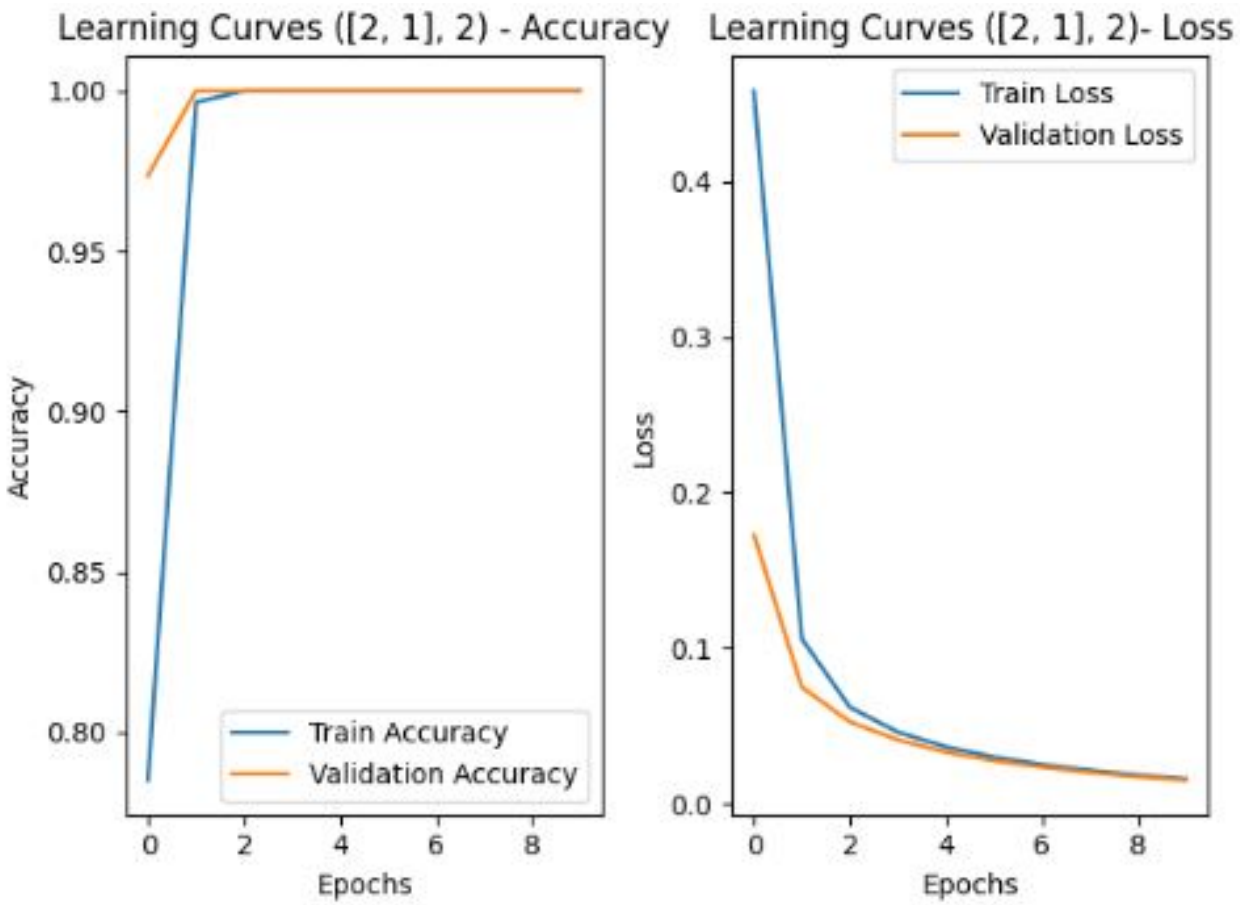

Figure 5.9: 2-1Architecture

5.10 Comparing Architectures and Results

All the results are very similar, which sort of leads me to believe that the dataset is a little "too perfect". Looking at the variables we see a normal distribution across each column which taken at random shouldn't happen. If they aren't normally distributed - they are close to it. I believe this is the reason for all these architectures to be similar. I may switch datasets and give this another write up. That being said, I have never seen this before, and it is interesting.

We really can only look at loss, and architecture [8,1] is a great contender - minimizes validation and training loss. The reasoning is that it performs about the same as the larger models, while being smaller (in loss since all other metrics are perfect).

	neurons	layers	val_loss	val_accuracy	train_loss	train_accuracy	precision	recall	f1
Logistic	16,4,2,1	4	0.000202	1.000000	0.000201	1.0	1.000000	1.00000	1.000000
	[64, 32, 16, 8]	4	0.002528	0.999267	0.000410	1.0	0.998697	0.99987	0.999283
	[32, 16, 8, 1]	4	0.001452	0.999600	0.000567	1.0	0.999348	0.99987	0.999609
	[16, 8, 1]	3	0.001737	1.000000	0.001687	1.0	1.000000	1.00000	1.000000
	[8, 1]	2	0.006357	1.000000	0.006151	1.0	1.000000	1.00000	1.000000
	[4, 1]	2	0.009796	1.000000	0.009439	1.0	1.000000	1.00000	1.000000
	[2, 1]	2	0.015185	1.000000	0.014763	1.0	1.000000	1.00000	1.000000

Figure 5.10: Architecture Results

Chapter 6

Feature Importance and Reduction

6.1 Training model architecture [8,1] with one feature input - 20 times

Model architecture [8,1] was selected as the best model earlier. The loop ran 20 timnes for all features to train and validate. The following results show that the difference between validation varies slightly. Where feature 15 is slightly more effective at this architecture near 60 percent, where the low is feature 14 at around 57 percent. This suggests that the features are all very important to the model. Every time my algorithm goes through a feature, a new model sequential 8-1 is created. This is identical to the model structure as above.

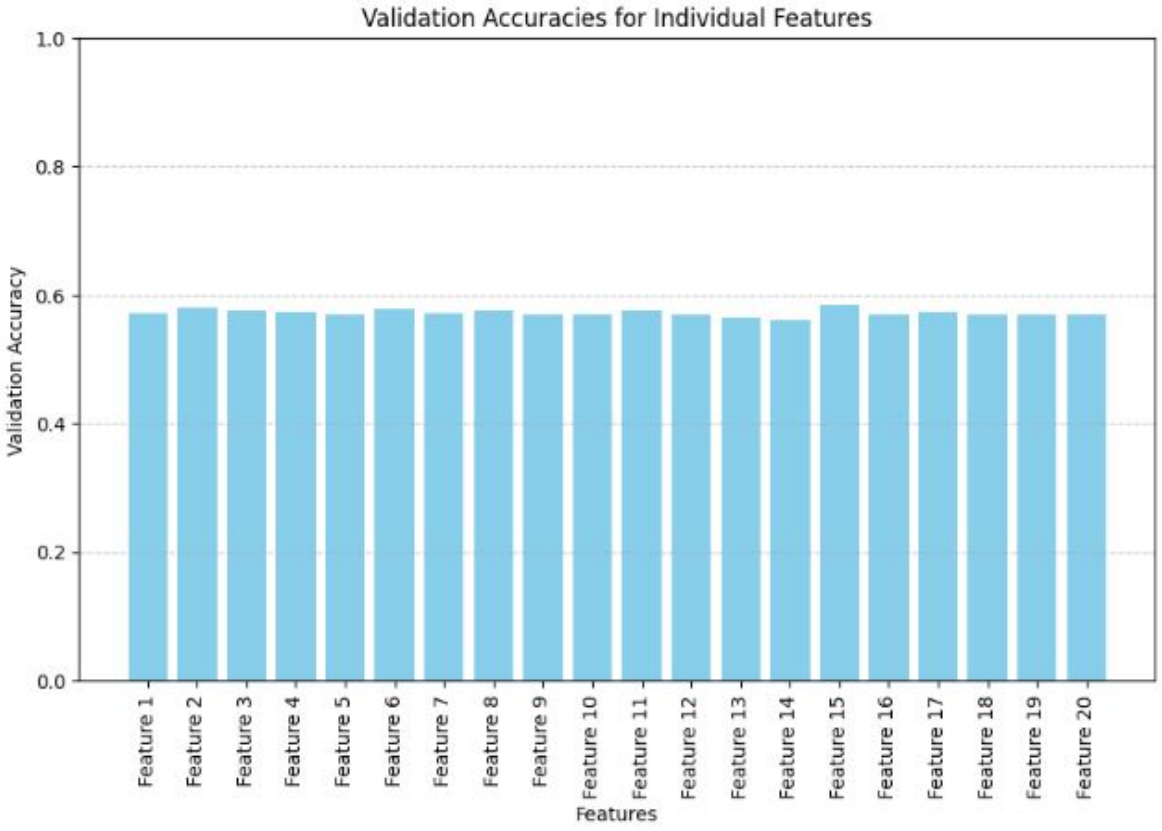

Figure 6.1: Feature Comparison on the 8-1 Model

Because of the extremely normal distribution across feature variables, and this data likely

having been ranked before (only conclusion one can reasonably come up with), suggests that this is fairly accurate.

Chapter 7

Training a Model With Reduced Features

Continuing with feature importance, models were trained gradually removing features. This feature reduction started with the feature that had the lowest validation in the previous step, and then the next lowest, and so on. I sorted the information I had stored into an array to do this. Using the 8-1 model previously mentioned, I found that the largest jump in model performance was the most important feature. Reducing my validation accuracy from 100 percent to 93 percent, the steps following would yield slightly lower steps in accuracy all the way until about 57 percent, still beating the baseline.

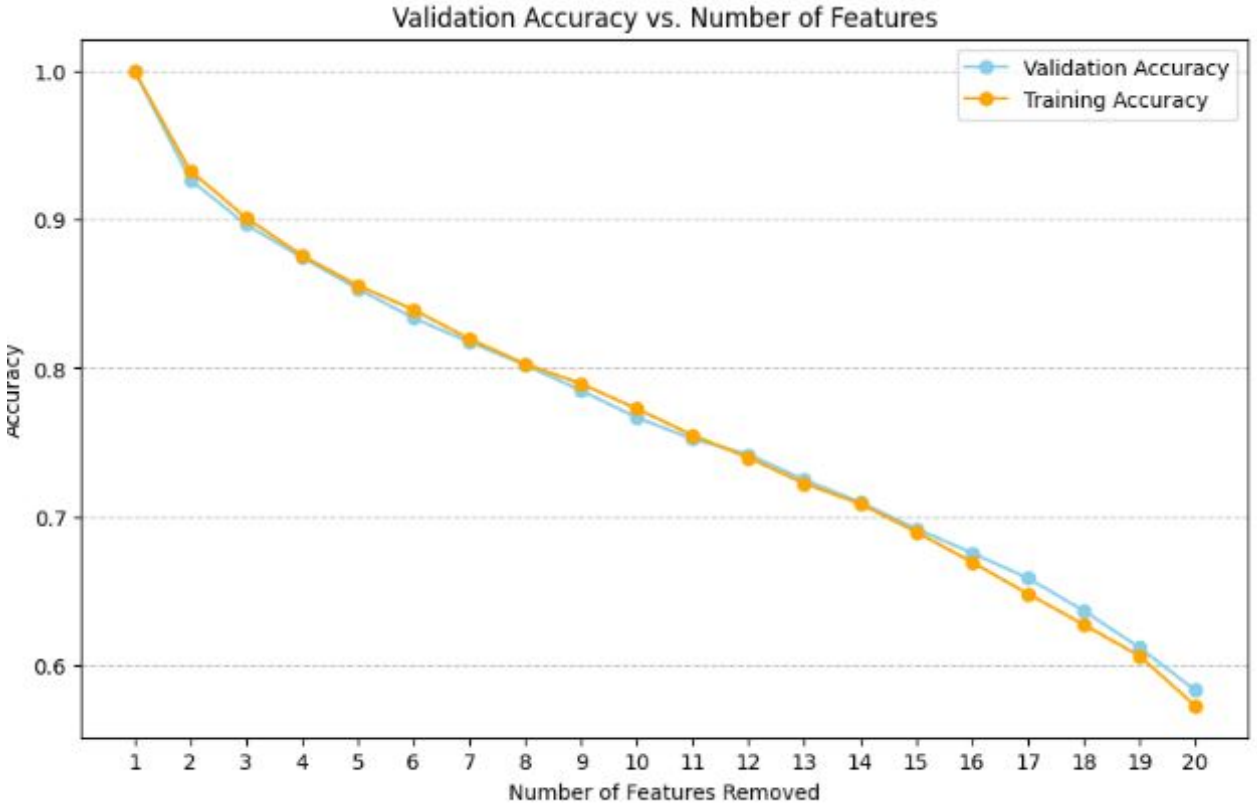

Figure 7.1: FeatureRemoval

Chapter 8

Conclusion

The data with this was suspiciously clean and I it not having reliable meta data besides what I shared is a red a flag. This shows the importance of data selection and review. The model I felt most comfortable using was the 8-1 architecture. This is because the accuracy was 1.0 (among several 1.0's) without it being too small (to ensure no overfitting).

Validation accuracy for all features being so similar and then seeing how it dropped from 100 percent to about 93 percent in the validation data was sort of shocking, this model without the unimportant feature 14, might help generalize the data and not cause it to overfit so much.

All models were binary models, with sigmodial activations and early stopping, models were compiled with binary crossentropy loss. These results were shocking.

I triple checked my results, earlier in the project I did have data leakage (forgot to drop a column), but after fixing that this data still seem very easy to work with, if we have a reliable "ranking" system like I think this data was generated off - paired with this model, we can better brace for mother nature's wrath.