DS4420 Project Final Report

Nicholas Chakmakas

April 30, 2021

Abstract

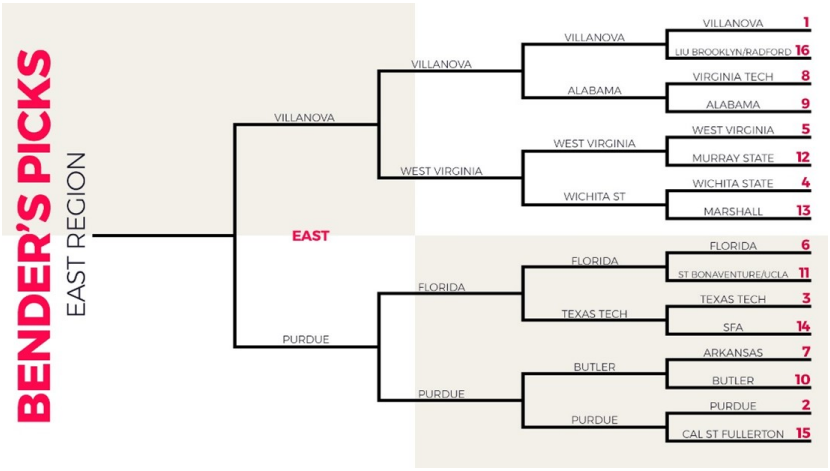
In this project, two neural networks are created to determine the outcome of a March Madness tournament. One method uses more basic data dating back to the 1985 season, while the other method uses data beginning from the 2003 season with more in-depth box scores. Data had to be transformed as part of this project, with the output layer using a binary classification to determine the winner of a game. This project lays a solid foundation for the problem, however, the neural networks perform poorly in this case and still need to be refined for them to be used to actually predict the outcome of a March Madness tournament.

1 Git Repo Link

To access all the code and data for this project, as well as the previous project that this is based on, visit: https://github.ccs.neu.edu/njchakmakas/DS4420-Project

2 Background

March Madness is a college basketball tournament that takes place from mid-March to early April, with 64 teams each competing for one national championship. There are four regions, with each team seeded between 1 to 16, with the favorites to win each game holding the higher seed (lower number, i.e. 1 = highest seed). Matchups in the first round always follows the same seeding order, and the winner of each game advances to the next round until there are no teams left. A single region in the bracket looks like this for a year:



Due to the high volume of games, there are always at least a handful of upsets during the course of the tournament, making predicting a bracket near impossible. The odds of predicting a perfect bracket if deciding the winner of a game by a coin flip is an absurd 1 in 9,223,372,036,854,775,808; with some insight on basketball, more around the range of 1 in 120.2 billion, according to NCAA.com. However, we have years of tournament data at our disposal, as well as regular season statistics about teams to

help us determine the outcome of a given March Madness game. Using machine learning methods, we're aiming to make building the perfect bracket a bit easier. By using regular season stats as well as outcomes from the tournament, we hope to create a model that will do the creation of a bracket that predicts most of the games correctly. This project is an expansion on a previous project, where I used Naive-Bayes and Random Forests to determine the winner of a March madness game, using R.

3 Data

Data was gathered from Kaggle, as part of the March Madness Learning Mania competition: https://www.kaggle.com/c/ncaam-march-mania-2021/data

The CSV's I am using are as follows:

- **MNCAATourneyCompactResults.csv**: Basic results from the tournament; this includes who won and lost the game, the day the game was played, season, and the score of the game. This data dates back to the 1985 season, the first year the NCAA had a 64-team tournament.
- **MNCAATourneySeeds.csv**: Corresponding seed for each team for each season's tournament.
- **MNCAARegularSeasonCompactResults.csv**: This includes basic game results from the 1985 season onward, with the winning team, losing team, and score for each, as well as season, day number, and number of overtimes if necessary.
- **MNCAARegularSeasonDetailedResults.csv**: This dataset includes more detailed box scores for each regular season game, starting in the 2003 season. This includes more a variety of statistics for both the winning and losing teams for each game, which are:
  - **FGM** - Field goals made
  - **FGA** - Field goals attempted
  - **FGM3** - Three pointers made
  - **FGA3** - Three pointers attempted
  - **FTM** - Free throws made
  - **FTA** - Free throws attempted
  - **OR** - Offensive rebounds
  - **DR** - Defensive rebounds
  - **Ast** - Assists
  - **TO** - Turnovers
  - **Stl** - Steals
  - **Blk** - Blocks
  - **PF** - Personal fouls committed by team

4 Data Manipulation

To get the data in a format that would be appropriate for training the neural network model, we first need to attach the regular season stats for each team to its respective tournament game. For example, if team 1 beats team 2 in a game in our data, we include both team's regular season stats for that game.

For the more basic regular season statistics starting in the 1985 season, each season was grouped with the winning and losing team ID's and counted to get each team's wins and loses for that season. In addition, each team's points scored was also tracked; the points scored in a win and loss was recorded and averaged per season. Then, using the number of wins and losses, a weighted average was used to determine the number of points per game (PPG), as well as calculating a team's win percentage in a season. So, for data dating back to the 1985 season, the statistics gathered were:

- Season, TeamID, Wins, Losses, Win %, PPG

This would serve as the baseline for our first neural network. With this dataset, it was merged onto the tournament results from 1985 onward to give us our regular season stats associated with the tournament game data itself.

However, there is still one issue with this. Since we don't know the outcome of the game *before* it happens, giving the model tournament game scores would be "cheating" in a sense, since there would be no way for the neural network to predict the outcome of the game since the data isn't technically available yet. Instead, we use the each team's respective seeds to distinguish between them, since this is known before the game takes place. By replacing the distinguishing factor from game outcome to seeding, we solve this issue.

Our output for our model is if the outcome results in the higher seed winning, resulting in a 0 or 1 label representing if the higher seed wins the given matchup. The data used for the model was given the appropriate labels, since we know the seeding and outcome of each game in the dataset; at this point, the more basic historical data is ready to be trained by the neural network. This dataset will be referred to as the *historical* dataset.

To get the more detailed data ready for the model, we do a similar aggregation on each statistic for the season for each team, taking a weighted average similar to how we calculated PPG earlier. Using that method, we calculated the regular season averages for each statistic listed in *MNCAARegularSeasonDetailedResults.csv*. Once we have those more detailed regular season stats, we associate those with the tournament games from 2003 onwards and convert our data from winning / losing teams to higher / lower seeded teams like earlier, and add the output labels based on the game outcome. This dataset will be referred to as the *detailed* dataset.

5 Methods

For this project, a neural network was built using Keras. The neural network has multiple layers, starting with 256 and getting reduced by half until 16, at which point the final layer uses a sigmoid activation function for binary classification purposes. In addition, it was determined that it would be best to add a dropout layer between each layer in the model until the output layer, to reduce bias in the model. By randomly dropping nodes, this helped ensure that we do not overfit our model on the training data. The loss function used is binary cross entropy loss, with an adam optimizer.

An important factor to the model was the *batch normalization layer*, which normalized the input data in the model before passing it on to any of the hidden layers. Without this normalization layer, it was found that the once the model was trained, it would output the same result for all layers, so there was clearly an issue. Once normalization took place, the features were properly weighted, and each value was an appropriate output corresponding to its probability from the sigmoid output layer.

Once the neural network was built correctly, the appropriate number of epochs to train the model had to be determined. Three different epoch sizes were tried:

- 128
- 256
- 512

5.1 Model Accuracy Plots for each Epoch Size

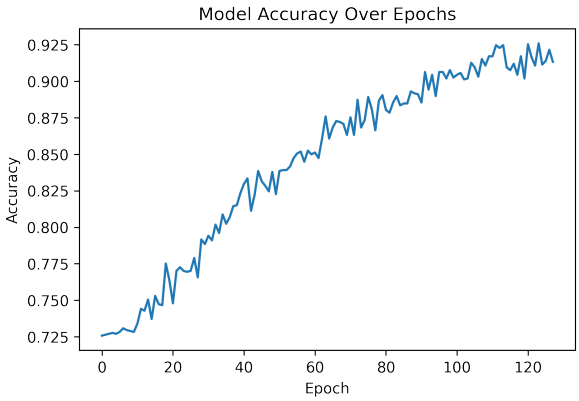


Figure 1: 128 epochs for Historical Data

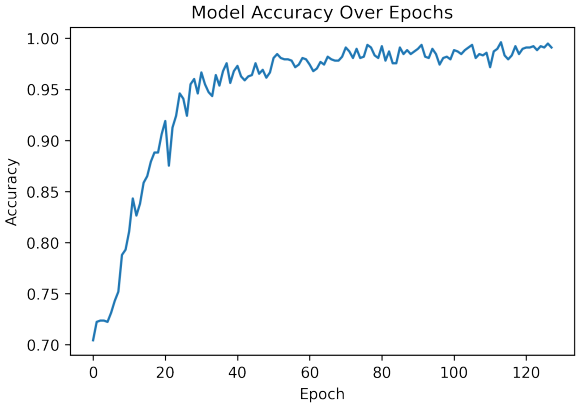


Figure 2: 128 epochs for Detailed Data

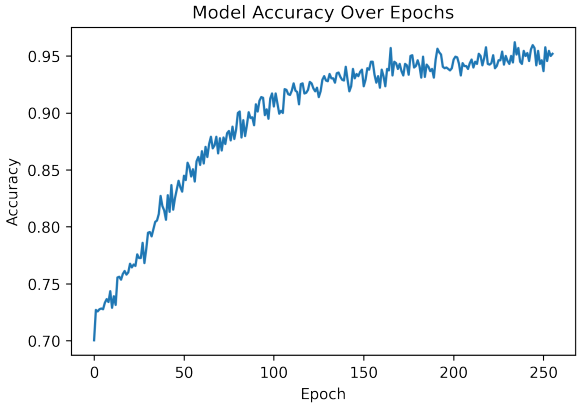


Figure 3: 256 epochs for Historical Data

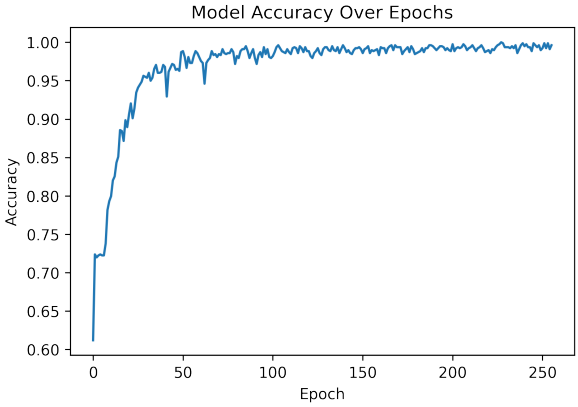


Figure 4: 256 epochs for Detailed Data

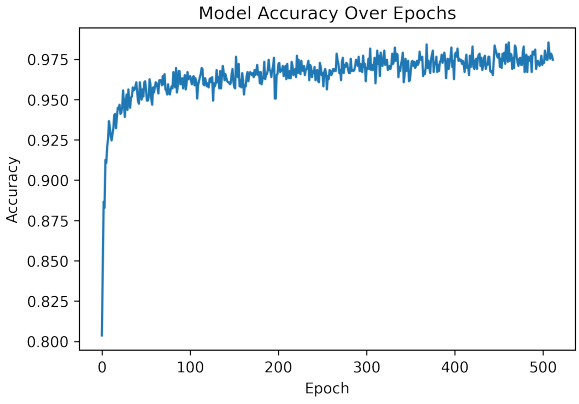


Figure 5: 256 epochs for Historical Data

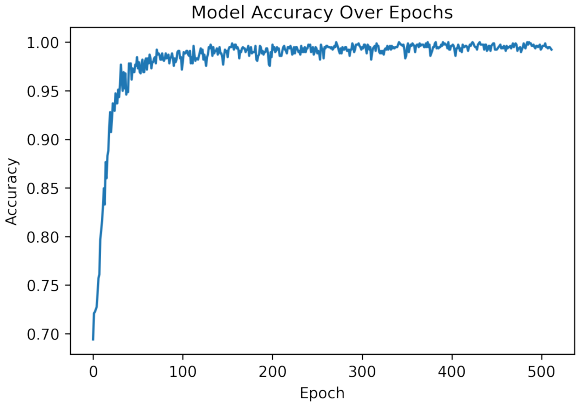


Figure 6: 256 epochs for Detailed Data

5.2 Model Loss Plots for each Epoch Size

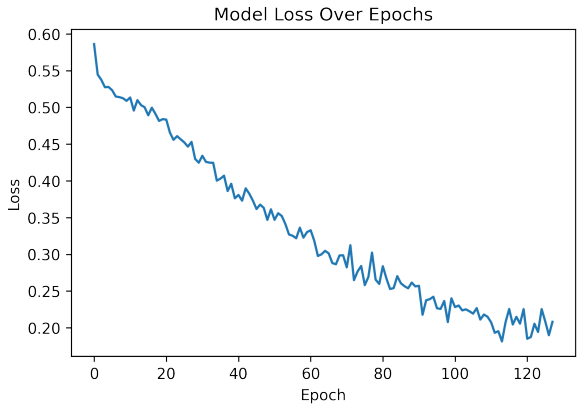


Figure 7: 128 epochs for Historical Data

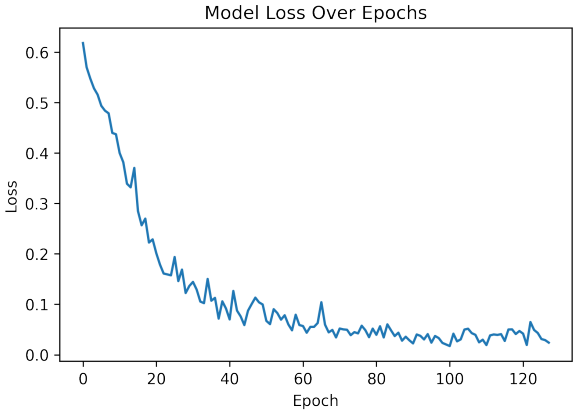


Figure 8: 128 epochs for Detailed Data

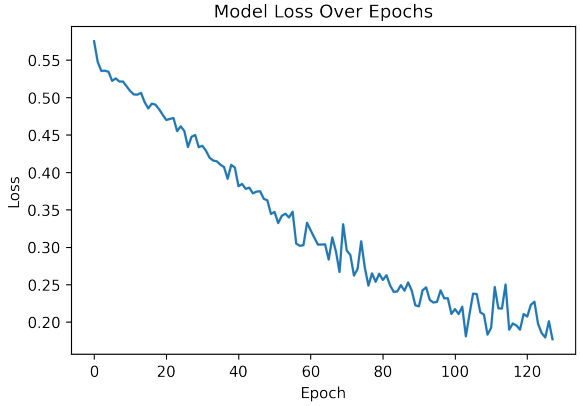


Figure 9: 256 epochs for Historical Data

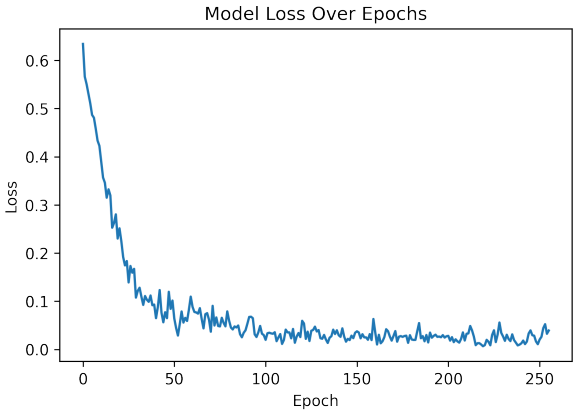


Figure 10: 256 epochs for Detailed Data

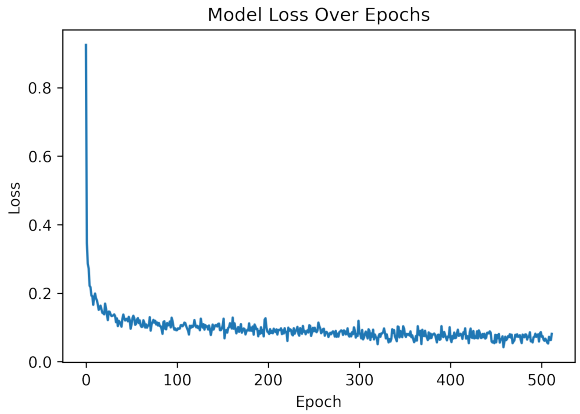


Figure 11: 256 epochs for Historical Data

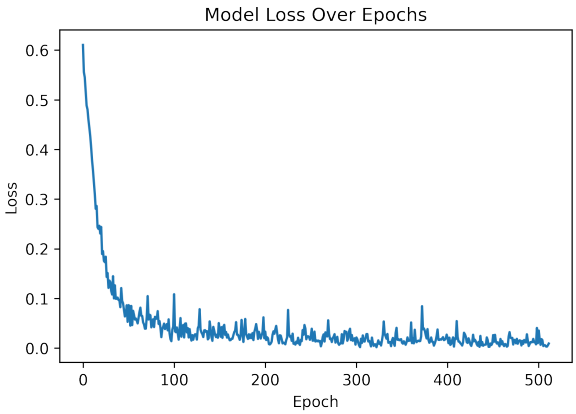


Figure 12: 256 epochs for Detailed Data

For the both neural networks, we see that model accuracy flattens around 150 – 200 epochs, so the neural network that's being trained with 256 epochs will be used for evaluation.

6 Evaluation

6.1 Neural Network 1: Historical Dataset

For the first neural network, we broke our dataset into two sets, using all data from 1985 – 2009 for training and 2010 – 2019 for testing, for about a 70/30 training / testing split. After training the model with 256 epochs, the predictions from the model have a mean squared error of .32 when used on the testing data, leaving us with an accuracy score of about 68%. This is less than ideal for determining a winner to the tournament. The confusion matrix is listed below:

		Predicted Value		
		0	1	Total
Actual Value	0	48	136	184
	1	76	400	476
	Total	124	536	660

The model mostly struggles with predicting the upsets that we're looking to get correct, missing 136 upsets; this makes up 20% of all our testing data, which is a large portion to get incorrect.

6.2 Neural network 2: Detailed Dataset

For the detailed dataset, seasons 2003 – 2014 were used for training and 2015-2019 were used for testing. The model surprisingly performed very similar to the first neural network, with a mean squared error of 0.32, resulting in accuracy of 68% on testing data. The confusion matrix is below:

		Predicted Value		
		0	1	Total
Actual Value	0	30	56	86
	1	50	194	244
	Total	80	250	330

Despite the similar accuracy scores, this model predicted slightly less false positives than above, with only 56 out of 330, and slightly more false negatives. However, this model is still not strong enough to be used as a predictive measure itself.

7 Conclusion

The neural networks that were trained as part of this project did not perform as well as expected, despite a high accuracy achieved during the training phase. This indicates that some overfitting has taken place; an addition to this project could be breaking down the model that currently exists and improving it to account for these errors.

I would hypothesize that under an ideal neural network where no overfitting is taking place, that the second approach would yield better results. The game of basketball is constantly changing, so data dating back to the late 80's and early 90's is more than likely not relevant. Basketball is even different than it was a decade ago, so perhaps only a few years data is needed; advanced regular season statistics provide additional features for our model, and are easily obtainable in these days with up-to-date records on sites like ESPN or SportsReference.

With more time, I would improve the neural network, and hopefully find more conclusive evidence of the hypothesis above.

8 References

1. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
2. https://medium.com/re-hoop-per-rate/training-a-neural-network-to-fill-out-my-march-madness-bracket-2e5ee562eab1
3. https://www.kdnuggets.com/2017/03/machine-learning-march-madness.html
4. https://medium.com/re-hoop-per-rate/new-and-improved-march-madness-neural-network-for-2020-c154aa1041b7
5. https://www.ncaa.com/webview/news%3Abasketball-men%3Abracketiq%3A2021-03-14%3Aperfect-ncaa-bracket-absurd-odds-march-madness-dream: :text=Here's%20the%20TL%2FDR%20version,a%20little%20some