FACULTY OF ENGINEERING AND APPLIED SCIENCE

Machine Learning & Data Mining

Project Report NBA Lineup Prediction

Course: SOFE 4620U

Group 11

CRN: 43510

Due Date: March 19,2025

Name:	Student number:
Shiv Patel	100818727
Nathan Yohannes	100749914
Fernando Chan Qui	100844946

Github Repository:

https://github.com/NathanYohannes/NBA_Final_Project

Introduction

The NBA lineup prediction model is designed to determine missing players in previous seasons' lineups using well known machine learning methodologies. In order to generate valid and accurate predictions for the missing player, we used a deep learning neural network, created using PyTorch. The model uses multiple variables to determine the best player, including the existing team on the field, the time at which they are coming, and the season in question. When testing this model, we use a dataset of all lineups and lineup changes that were made during a specific season (from 2007 to 2015).

Feature Engineering

As machine learning is, in essence, a large search problem, it is essential to ensure the data that we are sending into the model is as clean, and effective as possible. As such, we try to ensure that the data that we are training with is as sanitized, and easy to understand for the model as possible, while remaining relevant. To start, we employ one-hot-encoding for the names of players and teams in every dataset. With this, we ensure that the model is being fed information that it can understand, and quantify.

Furthermore, to lower our search-space, we find and train only on data where the home-team changed their players, rather than when anyone on the field (including the away team) changes their lineup. This ensures that our data is trained on only relevant data points, i.e. data that it will be tested against in the real-world. Since this model is required to predict any missing player in the home team (whether it may be player 0 or player 4), we wanted to ensure that the formatting of the testing data was similar to that of the training. As such, we ensured that any replaced player in the testing data was moved to the home_4 location, as this would require the model to only predict the player in home_4, while still remaining unbiased.

For efficiency, we decided it would be best to send data to the model in batches, which is determined by our data loader. Within the data loader, we do a few things to "sanitize" the data and only provide what is necessary for the model to train/predict. In this case, we removed the game code (as it is not required for the training, the season is sufficient). We also remove the home_4 column before sending it into the model, and save it for validation purposes later. We also found that the model was more effective when provided with the full game information, rather than one random game at a time (example: in a game with home team Boston, a batch will contain all relevant lineup changes from that game, rather than random teams). This provides more context to the model, and is able to narrow its search when provided the year and the home team. These batches (games) were then shuffled to lower the possibility of the model recognizing patterns and overfitting.

Model

The first model developed was a baseline model that simply combined the data into one training set and trained a deep learning model on it for a few epochs. The results were mild but the rate of accuracy increases over training showed promise and the team decided to use a deep learning approach.

One reason the team decided to use a deep learning model was, the model could use a powerful tool called an embedding layer. This allows for our player and team encodings to be categorized in a vector space and grants them deep relationships between each other. Our model can use that information to potentially understand that when a point guard is taken out, you would most likely want to substitute another in; or if the opposing team plays a big centre, you would need to respond.

To develop the model, the mixture of experts' approaches was used to maximize the relevance of the training data to the testing. The driving force behind using this approach was the idea that many things change between NBA seasons and attempting to use 2007 data to make predictions in 2015 wouldn't be useful. The algorithm stays the same but each model is trained only on the data available for that year. When making a prediction, the algorithm will look at the data being passed for prediction and choose the closest model in the year to use.

Training process

The training was split up by year, because, as mentioned before, a "mixture of experts" approach was used. This means that instead of having one model trained with un-required/outdated data, we would have multiple that work in unison depending on the data point to be classified. Because of this, we kept the given data, which was split up by year. This ensures no data leakage into other years' models.

When training, we used a training/validation split of 80/20, where on each epoch, all the training data would be run, followed by the 20% batches for the validation. After every epoch, we also check the previous losses, to ensure that the model is not training redundantly, with the same loss, leading to potential overfitting. For every year, we ran 60 epochs (or until the previous the loss was not improving). The loss function is calculated using Cross Entropy with an Adam optimizer. We also implemented GPU acceleration using CUDA cores, in order to more efficiently train this model.

Performance

The following graphs displays the training and validation metrics of the NBA prediction model for all the seasons that we tested for, from 2007 to 2015. These graphs will display four key performance metrics across ~25 epochs.

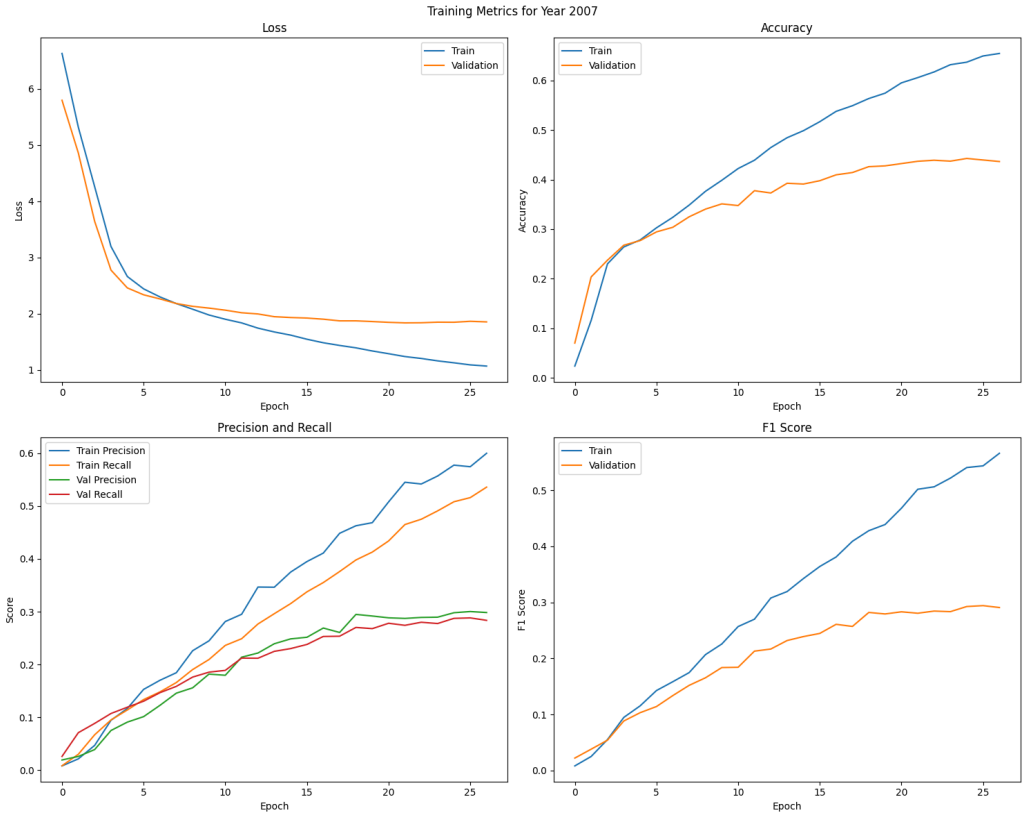
The **Loss graph** (top-left) shows how well the model is learning over time. The loss function will measure the difference between the predicted missing players and the actual ones. A decreasing loss during training means that the model is improving its predictions.

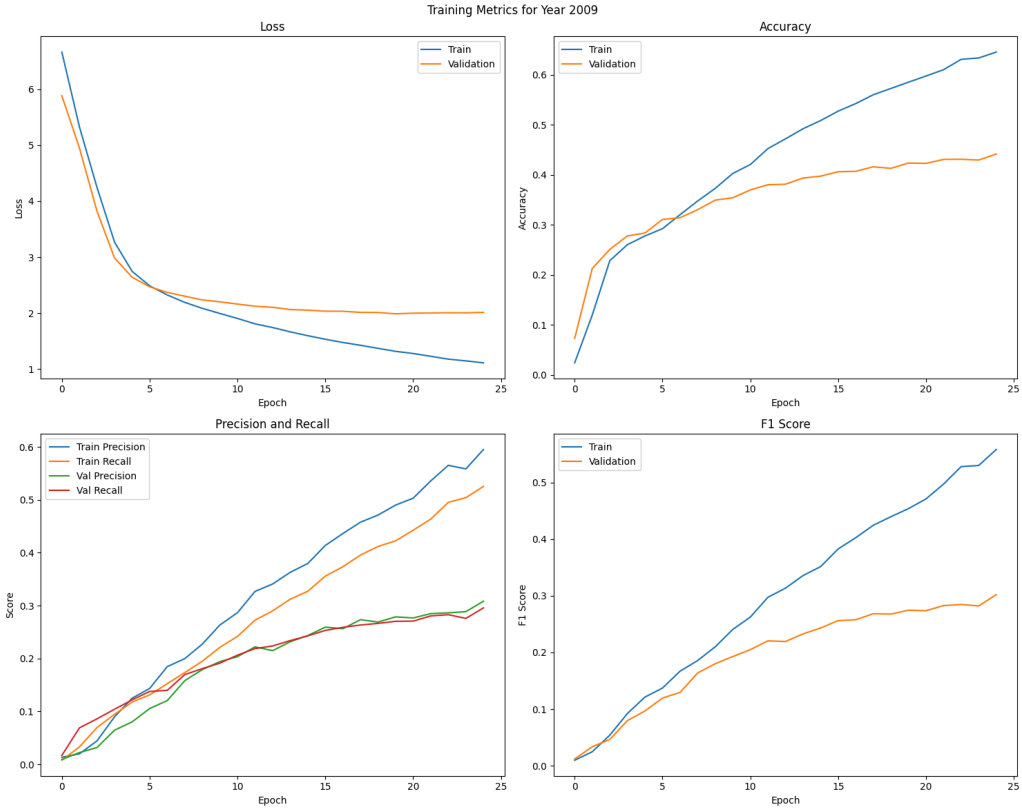
The **Accuracy graph** (top-right) shows the proportion of correct predictions made by the model. A higher accuracy indicates better performance. Over the training period, the accuracy steadily increases, signifying that the model is learning patterns in the data. However, the validation accuracy improves at a slower rate and remains lower than the training accuracy.

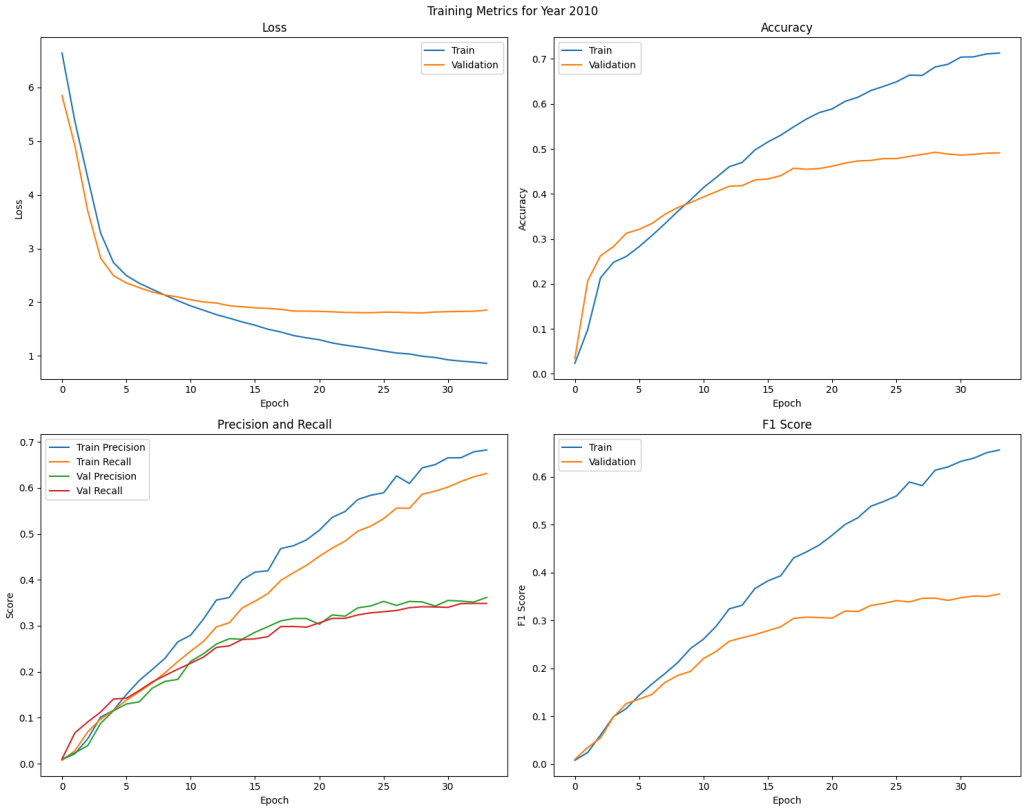
The **Precision and Recall graph** (bottom-left) provides information about the quality of the model's predictions. Precision measures how many of the predicted missing players are actually correct, while recall indicates how many of the actual missing players were identified by the model. Both metrics

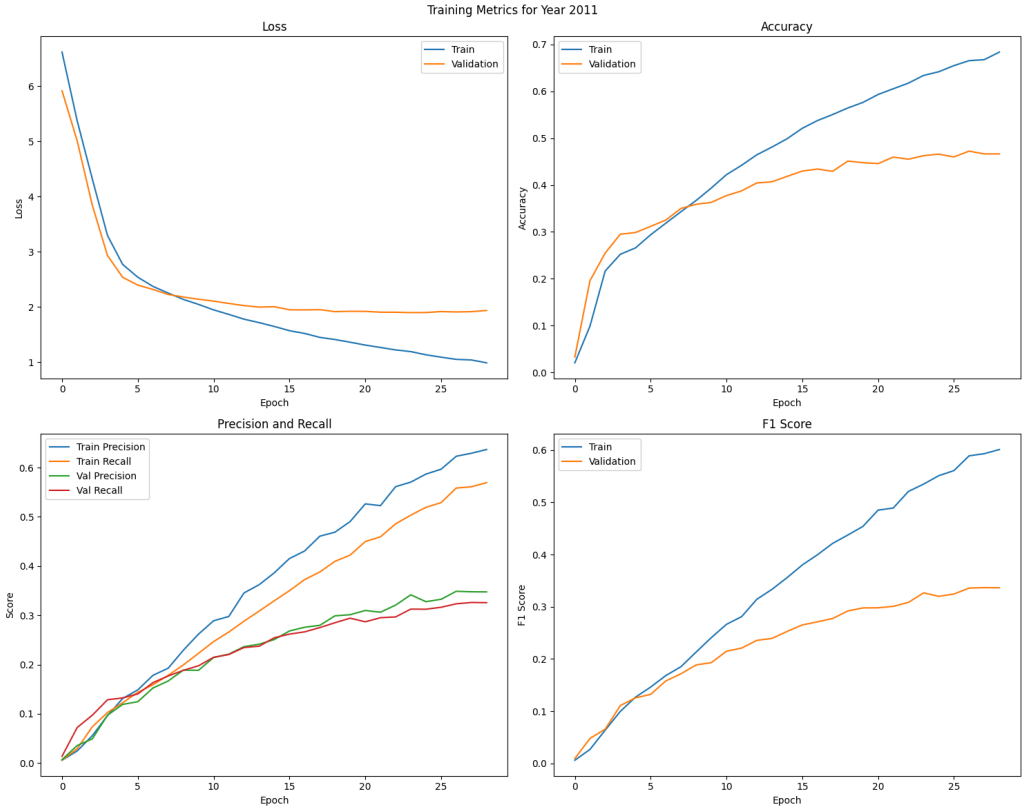
show improvement over time, which means the model is becoming more effective in recognizing missing players.

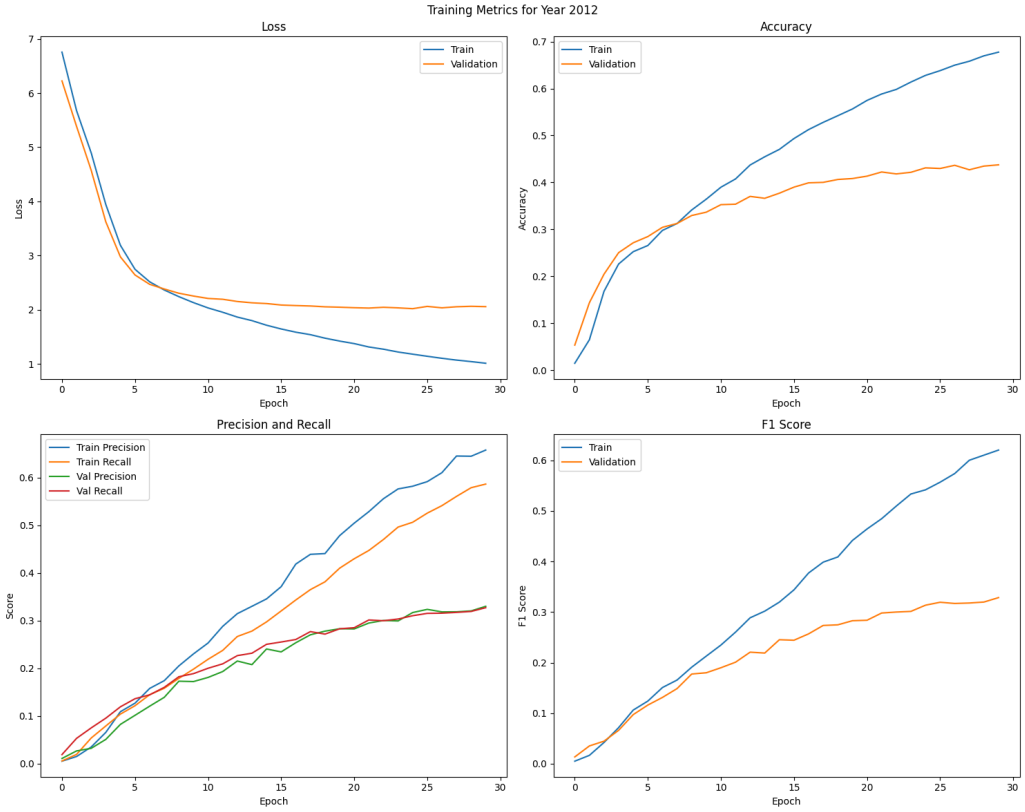
Finally, the **F1 Score graph** (bottom-right) balances precision and recall to provide a more comprehensive measure of the model's effectiveness. A higher F1 score indicates that the model is both accurate and able to identify missing players comprehensively. The training F1 score improves steadily, but the validation F1 score remains significantly lower. This suggests that while the model performs well during training, its ability to make accurate predictions on unseen data remains limited.

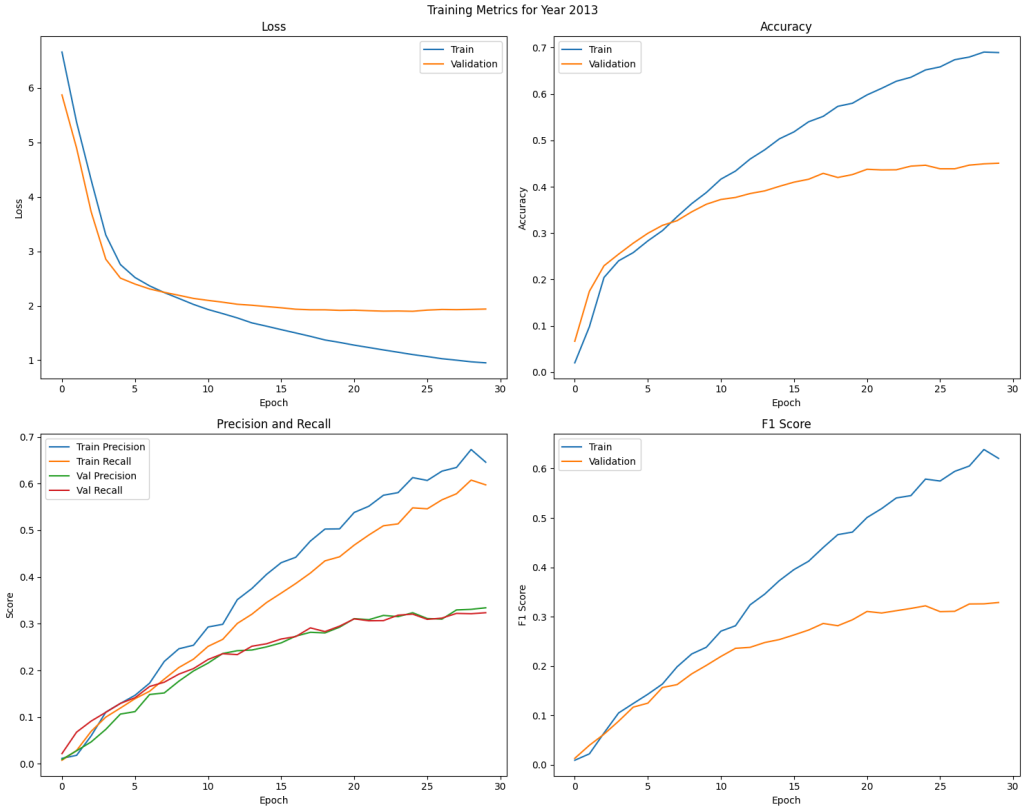


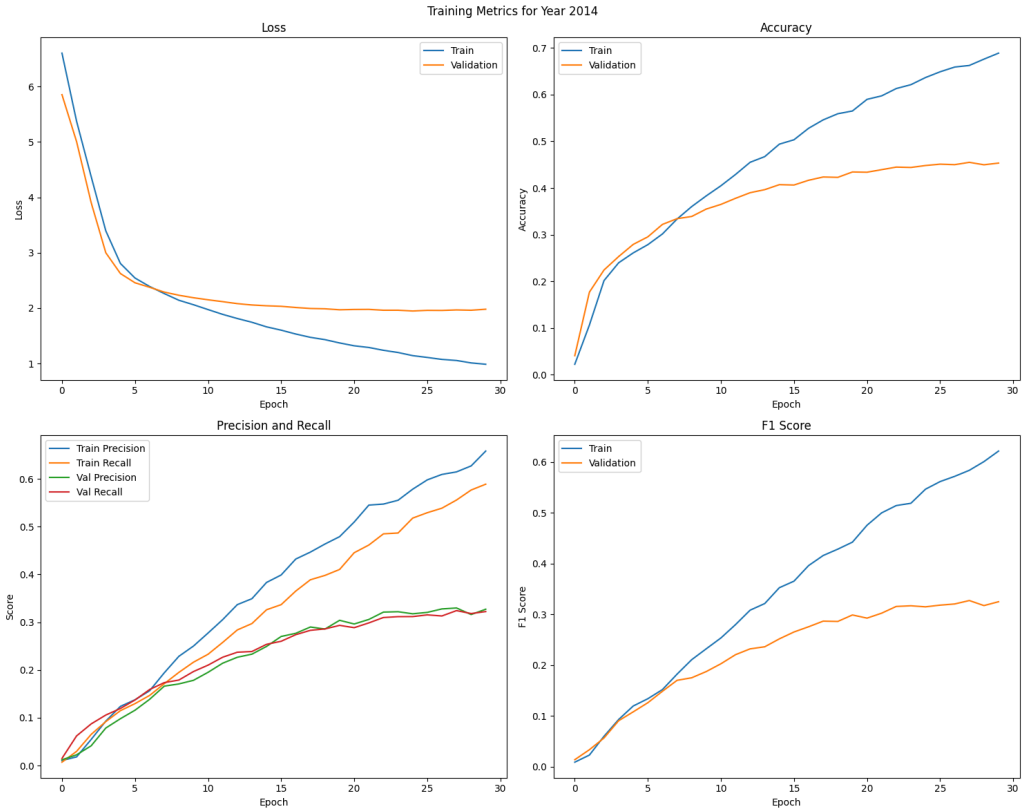

Training Metrics for Year 2009
Loss
Train
Validation
Loss
Epoch
Accuracy
Train
Validation
Accuracy
Epoch
Precision and Recall
Train Precision
Train Recall
Val Precision
Val Recall
Score
Epoch
F1 Score
Train
Validation
F1 Score
Epoch

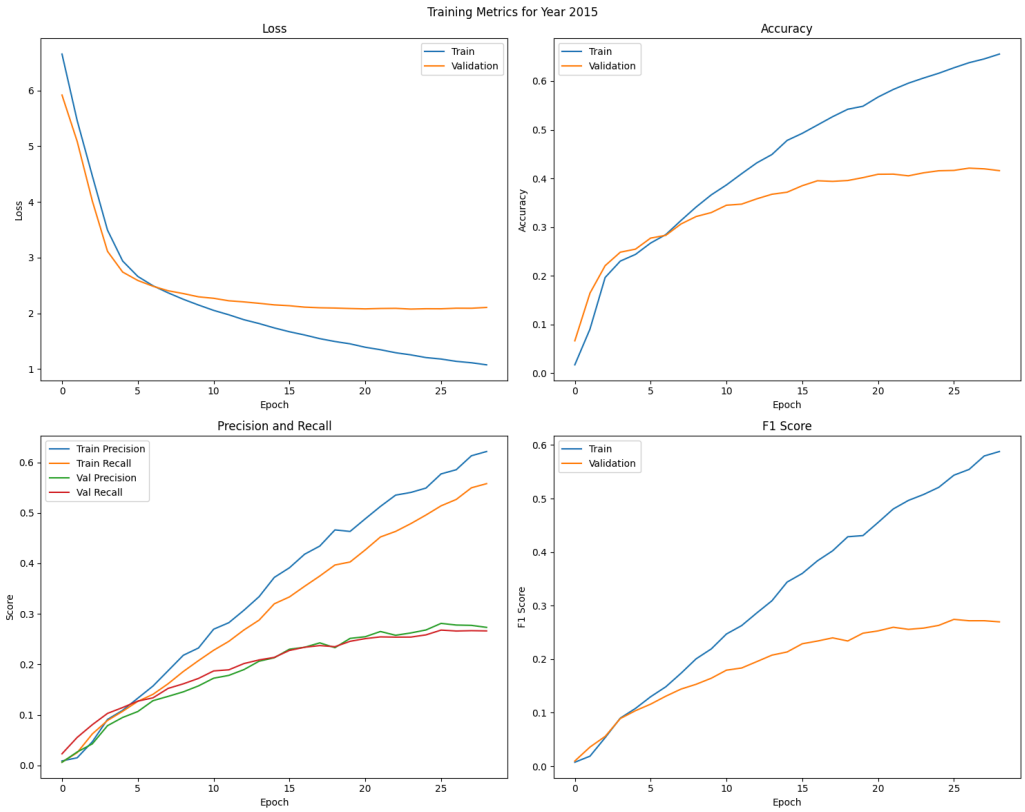

Training Metrics for Year 2010
Loss
Train
Validation
Loss
Epoch
Accuracy
Train
Validation
Accuracy
Epoch
Precision and Recall
Train Precision
Train Recall
Val Precision
Val Recall
Score
Epoch
F1 Score
Train
Validation
F1 Score
Epoch


Training Metrics for Year 2011
Loss
Train
Validation
Loss
Epoch
Accuracy
Train
Validation
Accuracy
Epoch
Precision and Recall
Train Precision
Train Recall
Val Precision
Val Recall
Score
Epoch
F1 Score
Train
Validation
F1 Score
Epoch


Training Metrics for Year 2012
Loss
Train
Validation
Loss
Epoch
Accuracy
Train
Validation
Accuracy
Epoch
Precision and Recall
Train Precision
Train Recall
Val Precision
Val Recall
Score
Epoch
F1 Score
Train
Validation
F1 Score
Epoch


Training Metrics for Year 2013
Loss
Train
Validation
Loss
Epoch
Accuracy
Train
Validation
Accuracy
Epoch
Precision and Recall
Train Precision
Train Recall
Val Precision
Val Recall
Score
Epoch
F1 Score
Train
Validation
F1 Score
Epoch


Training Metrics for Year 2014
Loss
Train
Validation
Loss
Epoch
Accuracy
Train
Validation
Accuracy
Epoch
Precision and Recall
Train Precision
Train Recall
Val Precision
Val Recall
Score
Epoch
F1 Score
Train
Validation
F1 Score
Epoch



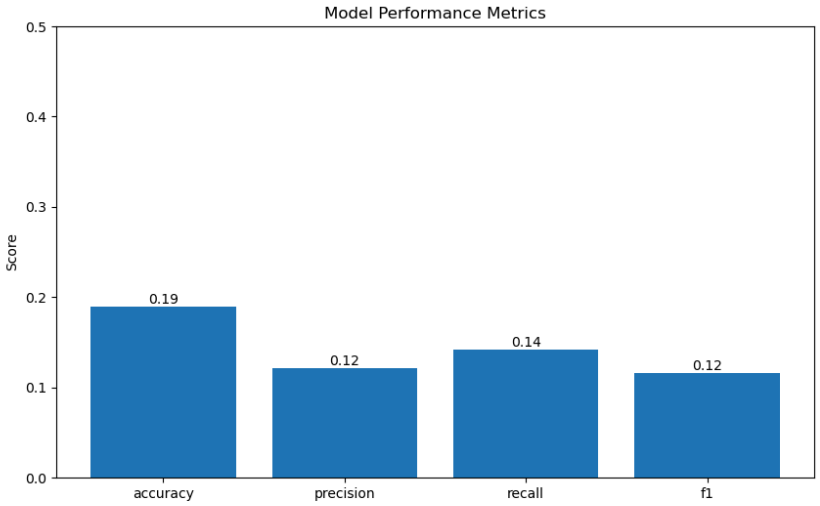
Pre and Post Processing

As mentioned before, in order to get the most out of our data, we needed to ensure that we were lowering the search-space by including only relevant and effective data. To do this, we encode, drop irrelevant data, and make the testing data as similar to real-world as possible.

Post processing on this model is minimal, as it uses the output of the model to create a decision on who the missing player is. With every prediction, the candidate with the highest output score is the one that will be selected. Once this decision is made, that output can be converted to the name of the player, rather than the number we used to encode it.

Evaluation

The model's scoring in evaluation did end up being lower than in our testing and validation. There are many reasons why this could have been the case. Though overfitting could have been of concern, the validation data used in the previous performance calculations was not used to train the weights of the model. There may have been some error in the real world data or our handling of it which would cause lower accuracy, precision, recall and f1-scores reported for evaluation.



Conclusion

The NBA lineup prediction model successfully is able to identify missing players given the historical lineups of home and away teams. By leveraging a deep learning neural network, in addition to well-engineering features, we are able to use this model to get ~70% accuracy while training. Using patterns in player interactions, roles, and matchups, it is able to accurately predict the missing player within a lineup. However, this performance lowers when working with testing versus training datasets. This difference indicates that this model generalizes under different contexts of data.

When training, we see the accuracy on the training set to be about ~70% consistently, through all models, and on validation training sets, we consistently get about 40%-50% accuracy. However, when tasked with predicting missing players on real-world data, that accuracy drops to 20%. Although this could be due to inconsistencies and missing data in the training data (such as specific players playing in seasons that they were not present in), it also indicates how our model generalizes its results based on the training data.

Overall, these results show that while the model generalizes and learns well on training datasets, it faces more challenges and the accuracy, precision, and recall performance of this model degrades as real-world datasets (with potential inconsistencies) are presented. Despite this, it is consistently able to identify missing candidates for a lineup while using multiple variables, such as team combination performance, away team performance/lineup, starting minutes, and more.