Parallel Acceleration of EpicFlow with Red/Black Successive Over Relaxation Using GPUs with CUDA

Amogh Param
704434779
aparam@cs.ucla.edu

June 2016

Abstract

We propose a strategy to improve the computational speed of state-of-the art optical flow estimation methods, targeted at large displacements with significant occlusions, by exploiting the substantial computational potential of Graphics Processing Units (GPUs) under the CUDA environment. The optical flow estimation method involves an energy minimization phase which we solve using an efficient and parallel numerical approximation scheme, Successive Over Relaxation with Red-Black ordering (Red-Black SOR). In addition, we show that choosing the right memory allocation and access patterns are critical in maximizing the computational and memory throughput for the estimation. The primary contribution of this project extends EpicFlow (Edge Preserving Interpolation of Correspondences), the state-of-the-art in optical flow estimation, by replacing the serial optimization procedure with a parallel optimization algorithm deployed on the GPU in the CUDA environment. The proposed energy minimization variation, along with efficient access patterns, performs on par with the state of the art on the MPI-Sintel, Kitti and Middlebury datasets.

1 Introduction

Optical flow is a classic computer vision problem where the primary goal is to track moving objects, surfaces and edges in a sequence of image frames by way of computing the displacement of pixels between every pair of frames in the sequence. To achieve this goal, we assume that the gray values of corresponding pixels are the same in both image frames of the pair, despite a displacement of the pixels. Optical flow is important to providing visual systems the ability to discern possibilities for action within an environment. In addition, optical flow stimulus is important for the perception of movement by an observer in the environment; perception of the shape, distance and movement of objects in the environment; and for locomotive control. As a result, it is used as a fundamental framework for applications such as image and surface reconstruction; object tracking and detection; and robot navigation.

Animals in the wild can sense the motion of other animals and as a result, make decisions whether to attack or flee based on visual stimulus derived from the motion of predators or prey. It is paramount that they make such decisions almost instantaneously to save their own lives (in case of prey) or to ensure a successful hunt (in case of predator). As a result, this provides motivation for building optical flow estimation techniques that are fast and almost in real time, since optical flow provides vision systems a fundamental way to discern motion.

[Types of Optical Flow approaches]

Energy based methods have the ability to deal with small displacements by using coarse-to-fine approaches where, optical flow estimation techniques are applied to smaller resolutions of the image and propagating the flow to upscaled versions of higher resolutions. Although this works quite well, errors at coarser levels are propagated to finer levels. Although there have been an abundance of such approaches it remains challenging to deal with occlusions, motion discontinuities and large displacements in estimating optical flow from real-world videos. EpicFlow handles these problems by initiating the optical flow estimation by interpolating a sparse set of matches to produce a dense correspondence field. It handles motion discontinuities by using an edge-aware geodesic distance instead of Euclidean distance in the interpolation. This estimate is then used to initialize an energy minimization to obtain the final optical flow estimate. Our project primarily extends this optical flow estimation procedure by accelerating the successive over relaxation process of solving the energy minimization, since it is inherently sequential in its original form. Drawing inspiration from applications in computational fluid dynamics and heat transfer, we use a multi colored ordering technique for the successive over relaxation where alternating elements are colored red or black and can be computed parallely for elements of the same color, thereby decreasing the computational time for the estimation. In addition, we explore the use of different memory access patterns and different memory types as optimizing parallel algorithms on the GPU within the CUDA environment can be challenging.

The rest of the paper is organised as follows: Section 2 describes fast methods for obtaining similarity measures. The detailed optical flow estimation method is described in Section 3. Section 4 shows the experimental results obtained using our fast image motion estimation method applied to a variety of images. Section 5 discusses the reliability and computation speed issues of our algorithm. Section 6 gives concluding remarks

1.1 Objectives

Our goal was to identify key factors in predicting NCAA tournament wins and to find a model that would perform well in the Kaggle competition. The Kaggle competition had the following stages:

- Predict outcomes of the 2015 NCAA Tournament
- Evaluate performance against benchmarks
- Evaluate log loss with actual outcomes

For each stage we submitted a list $\hat{y}$ of probabilities (values between 0 and 1) that each team in the tournament would defeat every other team in the tournament, regardless of whether this match-up actually occurs. For this year, this was m = 2278 predictions. We were judged based on the log-loss L(y—$\hat{y}$), or the predictive binomial deviance, of the games that actually occurred.

$$L(y|\hat{y}) = -1/n * \sum_{i=1}^{n} [y_i.log(\hat{y}_i + (1-y_i).log(1-\hat{y}_i))]$$

where n is the actual number of games played in the tournament (67), y_i is the actual binary outcome of each game, and $\hat{y}_i$ is the corresponding submitted probability. If the opponents in game i are teams A and B, then $\hat{y}_i(A,B)$ is the predicted probability that team A beats team B, and $\hat{y}_i(B,A) = 1 - \hat{y}_i(A,B)$

The goal is to find a set of predictions $\hat{y}$ that minimizes L(y—$\hat{y}$) for the unknown outcome vector y. This scoring method heavily penalizes being simultaneously confident and wrong. If we say Team A will beat Team B with probability 1 and Team B wins, then our log-loss is infinite (although Kaggle bounded the log-loss function to avoid infinite scores). At the same time, it is important to balance between being too conservative and too confident. If we are too conservative (e.g. $\hat{y} \approx 0.5$), then we will never accrue points, but if we are too confident, we make ourselves vulnerable to huge losses.[**1**]

Table 1: Dataset provided by Kaggle

Field	Description
season	indicates the year in which thetournament was played
dayzero	tells you the date corresponding to daynum=0 during that season.
region W/X/Y/Z	the four regions in the final tournament are always named W X Y and Z.
wscore	this identifies the number of points scored by the winning team.
lteam	this identifies the id number of the team that lost the game.
lscore	this identifies the number of points scored by the losing team.
numot	this indicates the number of overtime periods in the game - an integer 0 or higher.
wloc	this identifies the location of the winning team. If the winning team was the home team - this value will be H.
wfgm	field goals made
wfga	field goals attempted
wfgm3	three pointers made
wfga3	three pointers attempted
wftm	free throws made
wfta	free throws attempted
wor	offensive rebounds
wdr	defensive rebounds
wast	assists
wto	turnovers
wstl	steals
wblk	blocks
wpf	personal fouls

2 Challenges

2.1 Upsets

Every game is unpredictable, and the teams that are supposedly the ”better team” sometimes end up losing. This is called

an upset in basketball lingo, and happens regularly throughout the tournament. Because of these upsets, it can be difficult to correctly guess the winner of each game. The tournament can be so unpredictable.[**2**]

2.2 Chance of predicting a perfect bracket

Since there are 64 games played in the NCAA tournament, the odds of forecasting a perfect bracket are astronomical. Each game win/lose has a probability of 1/2. The complexity of the bracke prediction is 1/2*1/2..1/2 (63 times) which is 1 in *approx* 9.2 quintillion channce. This is lower than the chance of winning a lottery.

2.3 Unpredictable events

There are multiple events that happen during a match that cannot be predicted. The best example being injury of a key player. A new player being added to a team is one other such example. Playing at home ground vs playing else where sometimes also affects the outcome of the match.

2.4 Team Dynamics

The dynamics of a team change yearly. New players could be added, older key players could be removed. This makes it difficult to compare the performance of a new team to a old team as the logistics and features of an older team do not represent the newer team accurately.

3 Data

The main source of data is Kaggle. The data is made availabe as .csv files. Kaggle provides regular season wins, losses, point differences, dates, and game locations (home/away) as well as tournament wins, losses, point differences, seeds, and dates of games. This data comes from the season and tournament matches for the years 2005-2015. [**1**]. The data provided by Kaggle mainly consists of match statistics (seasons and tournaments) and team information. We extract average team statistics from this data. As a result we broadly end up with two types of data: [**1**]

3.1 Match Statistics

The match statistics include each team's performance for that individual match instance. This type of data is available for all the matches in all the seasons and tournaments. *Table 1* provides the format of this data.

teamId	season	wscore	wfgm	wfga	wfgm3
1102	2003	68.75	22.58333333333333	40.0	10.0
1103	2003	87.76923076923076	30.0	55.384615384615394	5.461538461538462
1104	2003	74.70588235294117	25.823529411764714	58.352941176470594	7.0588235294117645
1105	2003	79.42857142857143	25.57142857142857	61.85714285714285	9.142857142857142
1106	2003	68.30769230769229	24.76923076923077	53.846153846153854	5.846153846153847

Table 2: Snapshot of the extracted team level data

team1	team2	wscore	wfgm	lscore	lfgm
1411	1421	-2.205128205	-1.61965812	1.5625	1.354166667
1112	1436	14.41052632	4.482105263	16.8	5.366666667
1113	1272	5.896135266	2.403381643	-5.151515152	-0.651515152
1141	1166	2.658170915	-1.250374813	-4.75	-3.5
1143	1301	-1.793650794	1.904761905	6.083333333	4.416666667
1140	1163	-11.14699793	-6.538302277	-0.125	-3.416666667

Table 3: Snapshot of the training data used in the models

3.2 Team Statistics

Using the match statistics provided by Kaggle, we extract team level statistics for an entire season for all the teams. This represents a cumulative performance of a team for the previous four seasons starting from the year before the target season that we want to predict for. Some of the most important team statistics are:

- Two point field goals made
  Three point attempts
- Assists per game
- Average scoring margin
- Blocks for season
- Defensive rating
- Field goal percentage
- Total free throw attempts

3.3 Scale of the Data

Kaggle provides data from the NCAA seasons and tournaments between 2005 and 2015. The statistics below give an

idea about the scale of the dataset.

- Number of seasons: 10 seasons
  Number of tournaments: 10 tournaments
- Number of matches in each season: 5000
- Number of matches in each tournament: 64
- Total number of matches in the training set 50,640
- Total number of matches in testing set: 64
- Total number of columns in the training set/testing set number of data fields or features ≈ 40

4 Implementation

Here we describe the implementation details that we used in our approach to predict the outcomes of matchups.

4.1 Features

As described in the section 3.1 and 3.2 the data provided by Kaggle has a lot statistics compiled for every basketball game, so it was difficult to decide which ones to use. Hence we used all the features initially to construct a training set.

4.1.1 Feature Representation

Using the match level statistics provided by Kaggle, we extracted the team level data for every team in every season. Table 2 shows a small snapshot of this data, but it only contains 4 of the 33 features that we actually use in the dataset. Each of the values for the teams represent the average statistic of that team for that specific season. For example, 'wscore' represents the average points a team scored in a game in which it was the winning team. A statistic starting with a 'w' represents that statistic when the team was a winning team. Similarly, the 'l' represents a statistic when the team was a losing team.

Using the above team level statistic data and the matchup data, we create the training set for the machine learning models. For every match up, we look at the teams involved and their corresponding statistics in the team level statistics that we created previously. For each team, we look at all the matches they won, and compute the average of the 'w' statistics. Similarly, we look at the matches that the team lost and compute the average of the 'l' statistics for that team.

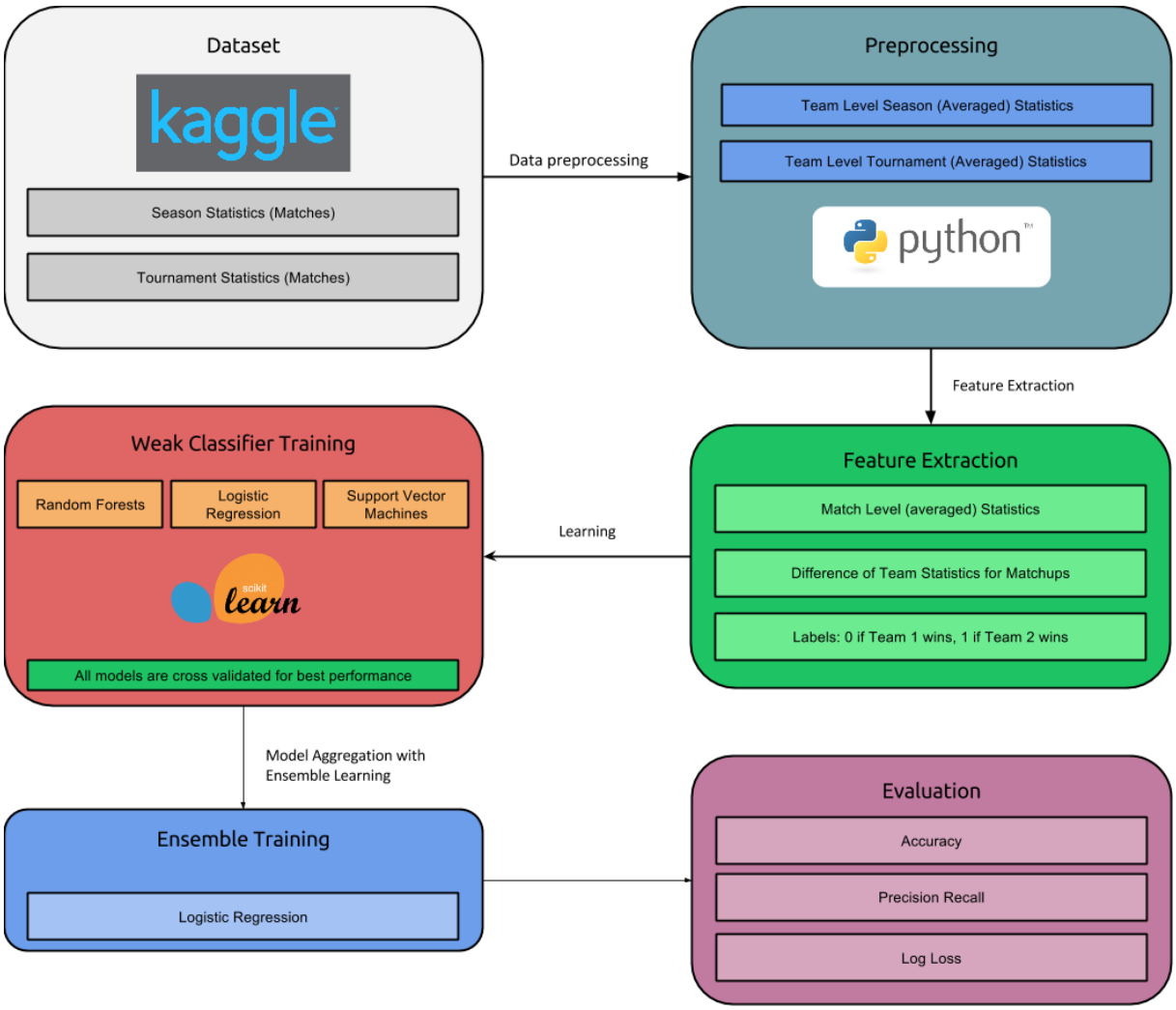

Figure 1: Architecture

We then combine the two vectors for a matchup for a specific team which results in the team vector representing the average statistics of the teams involved in the matchup. This results in two team vectors for every matchup. We can use two approaches to represent these vectors as data points in our training dataset, as follows:

- Vector of differences: Each match is represented by the vector of difference between the individual team vectors. This essentially means that the training vector/data point is represented as:

  $X = mod(TeamAStats - TeamBStats)$.

  The result or the label vector Y is represented by 0/1. The result is 1 if Team A beats Team B and 0 if Team B beats Team A.

- Concatenated Vector of both the team statistics: In this representation the statistics of a match is represented by keeping both Team A and Team B statistics in the vector next to each other. Hence the training vector

  $X = [TeamAStats, TeamBStats]$.

  The result or the label set representation is same as the previous case.

In our implementation, we use the first 'Vector of differences' approach to represent each data point for the training model. When we consider a match between two teams, we sort them by their ids and then take their differences. As mentioned above, if the first team wins, we label the outcome as 0, and 1 if the second team wins. This gives us a distribution of target labels to predict.

4.1.2 Additional Features

We considered applying a few additional ideas to apply different feature extraction algorithms to the existing team and match statistics and consider the output from the algorithm as a feature itself.

Team Rank A method similar to PageRank algorithm called the GEM method was applied to the NCAA dataset [**3**]. The margin of victory $(v1 - v2)$ is used as the weight the "link"

between two football teams, where v1 and v2 are the teams' scores against each other. Then the process same as PageRank is followed to make it row stochastic. This will help us obtain the final ranking. Below is an example of the team Rank calculation.

$$H = \begin{array}{c} \\ \text{NO} \\ \text{PHL} \\ \text{ATL} \\ \text{CAR} \end{array} \begin{array}{c} \begin{array}{cccc} \text{NO} & \text{PHL} & \text{ATL} & \text{CAR} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 13 \\ 26 & 0 & 0 & 0 \\ 11 & 27 & 0 & 9 \\ 10 & 28 & 8 & 0 \end{pmatrix} \end{array}.$$

Figure 2: Team Rank Matrix

This team rank matrix is represented as a graph.

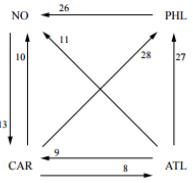

Figure 3: Team Rank graph

Next, we continue to make it row stochastic and follow the PageRank algorithm to get the final ranking. We obtain (0.330 0.252 0.087 0.332) for the PageRank vector, which produces the ranking 1. CAR, 2. NO, 3. PHL, 4. ATL.

4.1.3 Statistical Features

Statistical Features on Player Statistics: One more extra feature that was added to the training set at the list of top players that belong to every team. Only the top players that are playing a particular match are considered in the training set.

5 Architecture

The project can be divided into six important parts; Data collection, Data Preprocessing, Feature extraction, Weak Classifier Learning, Ensemble Training, Prediction/Evaluation. Figure 1 is a pictorial representation of the architecture of the project. The first three parts of the architecture are explained in Section 4. The remaining are explained in the following sections.

6 Machine learning Models

For reasons stated above in section 3, we wanted to use many different machine learning techniques and leverage them against one another so that we would gain the benefits from each and hopefully mitigate the weaknesses of each. We used Support Vector Machines, Random Forests, and Logistic Regression for the base level models.

6.1 Support Vector Machines

Each game in the training set X has a series of characteristics based on the difference in team statistics and a label yi (win or loss). These can be plotted in a p-dimensional Euclidean space. The support vector machine then finds the hyperplanes $w\Delta X - b = 1$ and $w\Delta X = 0$ with the greatest distance between them that partitions the space such that wins and losses are separated (where w is the normal vector to the plane and $\|w\|/b$ is the offset of the hyperplane from the origin along w.)[**4**]. Let the output of an SVM be $f(X)$. Then

$$\hat{y_{i1}}(A,B) = Pr(y_i = 1|f(X)) = 1/1 + exp(Q * f(X) + V),$$

where Q and V are parameters chosen through maximum likelihood estimation from a training set made up of coordinates $(f(X)_i, y_i)$.[**4**]

6.2 Decision Trees and Random Forests

Decision trees split a data set using a set of binary rules that make up the branches of the tree. Sets further down in the tree contain increasingly similar class labels. The objectives are to produce the purest sets possible and to be able to classify any object based on comparison of features to the rules in the tree. A random forest is a bootstrap of the decision tree. Like in the standard bootstrap method, the available training data can be resampled to build many different decision trees. Additionally, the random forest method takes a random subset of available predictors to build its new decision tree. By resampling both the cases and the input variables, many decision trees are grown. Final classification decisions are made based on a majority rule of the many decision trees. Given a decision tree with height k, let $Pr(r_d)$ be the probability associated with node d on branch path r. Then the probability of success for a new case is:

$$y_i(A,B) = \prod_d Pr(r_d)$$

For the random forest of m trees where $Pr(r_{(d,j)})$ is the probability associated with node d on branch path r on decision tree j, the probability of success for a new case is:

$$y_i(A,B) = \sum_j \prod_d Pr(r_{(d,j)})$$

6.3 Logistic Regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). In logistic regression, the dependent variable is binary or dichotomous, i.e. it only contains data coded as 1 (TRUE, success) or 0 (FALSE, failure). In our case the dependent variable is the outcome of the match and independent variables are the features themselves. Logistic Regressio uses a logit function which measures the log odds of Team A winning a match against Team B:

$$\text{logit(p)} = \ln(\text{p}/(1\text{-p}))$$

where P is the probobality of A winning.

6.4 Ensemble Model

Ensemble methods are commonly used in classification problems. These methods allow predictions based on different models to be combined, often using a type of voting schema, to determine one final prediction. The benefit to using an ensemble method is that we can leverage the information gleaned from different methods, combining the benefits from each method while hopefully negating or minimizing the flaws in each. Ensemble methods have been used to combine predictions from different machine learning techniques in order to increase stability and to more accurately classify test cases. The combination of methods can also reduce the variance of the predictions as they are less sensitive to irregularities in a training set.[**4**] In order to ensure that our predictions were in (0,1), we fit a logistic regression model where each explanatory variable is a prediction from one machine learning technique, and the response is the binary outcome. Our model takes the form:

$$y_i(A,B) = 1/e^{-(\beta_0+\beta_1 y_1(A,B)+..\beta_3 y_3(A,B)+\varepsilon)}$$

where y_1 is from Support Vector machines, y_2 is from Random Forests and y_3 from Logistic Regression.

7 Evaluation

To evaluate the performance of our models, we use a number of performance statistics. Primarily, we use:

Model	Dataset	Accuracy
Random Forest	Training	0.97
Logistic Regression	Training	0.76
Support Vector Machine	Training	1.0
Random Forest	Testing	0.61
Logistic Regression	Testing	0.59
Ensemble	Testing	0.64

Table 4: Snapshot of the extracted team level data

- Accuracy
- Precision
- Recall
- F1 Measure
- Log Loss

7.1 Performance

From the tables that follow, we see that our classifiers perform very well when we test the learned model on the training data. On the testing data, the classifiers do not perform as good as they did on the training data, but they still perform fairly well, definitely better than a random coin flip.

	precision	recall	f1-score	support
0	0.99	0.96	0.98	109
1	0.96	0.99	0.97	92
avg / total	0.98	0.98	0.98	201

Table 5: Random Forest on Training Data

	precision	recall	f1-score	support
0	0.76	0.79	0.78	103
1	0.77	0.74	0.76	98
avg / total	0.77	0.77	0.77	201

Table 6: Logistic Regression on Training Data

	precision	recall	f1-score	support
0	1.00	1.00	1.00	106
1	1.00	1.00	1.00	95
avg / total	1.00	1.00	1.00	201

Table 7: Support Vector Machines on Training Data

	precision	recall	f1-score	support
0	0.79	0.62	0.70	48
1	0.38	0.58	0.46	19
avg / total	0.67	0.61	0.63	67

Table 8: Random Forest on Testing Data

	precision	recall	f1-score	support
0	0.55	0.68	0.61	31
1	0.66	0.53	0.58	36
avg / total	0.61	0.60	0.60	67

Table 9: Logistic Regression on Testing Data

	precision	recall	f1-score	support
0	0.74	0.67	0.70	42
1	0.52	0.60	0.56	25
avg/total	0.65	0.64	0.65	67

Table 10: Ensemble Learning on Testing Data

We see from the above tables that the random forest classifier performs the best among the individual classifiers on the testing data and the ensemble learner does make a few gains over the random forest classifier. Thereby it does improve the prediction ability by combining the underlying base models.

Although, we observe that since the models perform extremely well on the training data and not as well on the testing data. This is primarily because the models overfit to the training data.

7.2 Log Loss

In addition to the performance metrics, we also evaluated our performance on a log loss function, provided by Kaggle to evaluate how confident our model makes predictions. The loss function is:

$$L(y|\hat{y}) = -1/n * \sum_{i=1}^{n} [y_i.log(\hat{y}_i + (1-y_i).log(1-\hat{y}_i))]$$

where, y represents the actual 1 or 0 outcomes and $\hat{y}$ represents model prediction probability values.

Benchmark	Score
Winners	0.529
Seed Benchmarks	0.6
All 0.5 Benchmarks	0.693
All 0 Benchmarks	19.19
OUR PERFORMANCE	0.62444662

Table 11: Log Loss Benchmarks

We see from the table below that, although we are way off the best performing score, we still did better than 2 of the benchmarks. It seems like just following the seed predictions leads to better performance than our classifier. These issues and areas of improvement are discussed in the following section.

8 Conclusion

Even with large amounts of statistical information, there are some parts of a basketball game you simply cannot predict. Injuries cannot be forecast, and the mental state of players playing the game cannot be observed. However, these things can drastically affect the outcome of a basketball match. Perfect brackets will therefore continue to be impossible to calculate, no matter how much data is provided to the learning models.

However, we believe that our classification accuracy can improve to above 75% if the proper features are used in training. Vital statistical information we were unable to gather include the following:

- Venue of the match including the distance
- Importance of a match-up
- Average number of players who play consistently

All of this data is expected to play a role in how teams perform. However, this type of data was out of our scope to accumulate. Organizations with more access to this type of data will be able to study whether these features have a profound effect on the success of a basketball team. This research would also benefit greatly with more data. It is recommended that much more data be gathered to help avoid overfit. However, since basketball continually changes throughout the years, using data from more than ten years back is also useless. We would also recommend experimenting with using several feature reduction techniques together as opposed to using the reduction techniques one by one. This is something we did not explore as a part of the project.

The overfitting problem can be solved by dividing the training set into 2 parts - training and evaluation. In this way the model can be built using training set, and fine tuned using evalution set. It can also be reduced by using some kind of regularisation in the model. It could L1 or L2 reguralisation depending on the machine learning model and parameters.