Court Vision

By: Jeffrey Sims, Fisher Fugler, Lucas Julian, & Callista Coats

Summer 2020

Introduction

This Court Vision project aims to track players and objects in a high school basketball game using machine learning, computer vision, and artificial intelligence so it could be rewatched in a statistical fashion, such as on a 2D graph with X's and O's as the team players. The primary use is to analyze each player's strengths and weaknesses for coaches to review after the game, but the next goal is to have the product work with a live feed. An example of the statistics gathered by the product is the percentage of successful shots by a player and what percentages of those shots were taken from a certain area in the court, such as a halfcourt, layup, or 3-point shot. From this data, it would be very beneficial for coaches to review these stats of the game during halftime and make useful adjustments to the team. Another purpose of real-time analytics would be so the parents and fans could view the statistics during the game using their phone. The motivation behind the project is that some of these techniques are being used in the NBA and major league baseball, but there is nothing available on the high school level. Therefore, there is a large, uncatered to market for this product for these high school and club level basketball groups.

Our part of this project was court landmark detection (landmarks are points of interest on the court, such as the blocks on the paint, the half-line intersection with the sideline, the corner of the court, etc.) and normalizing court coordinates. These functionalities are necessary in order to pinpoint the locations of players and the basketball for every frame in the video. This was accomplished through computer vision applications of a convolutional neural network to recognize multiple landmarks of the basketball court. Also, our finished project is not going straight to the market; more work will be done later on for the problems we did not get to. Because of this, our final solution requirements are not restrained by strict thresholds of accuracy.

Requirements

Our functional requirements include that:

- Our solution to the court landmark detection integrates correctly with the work already done on the project, such as the differentiation between all the objects in the game, such as the two teams of the game, the ball, and the basket/backboard.
- The model recognizes the basketball court through landmarks and is able to show the position of players and the ball on the output video. The output video is a normalized court (i.e., the coordinates as on the two-dimensional plane looking straight down at the court) with dots as the positions of the objects, where objects refer to players, referees, the ball, the basket, etc. More on this is explained and shown in the **System Architecture** section and the **Appendices** *Calculation* section..
- Preprocessing:
    - Creates altered images from training images to increase the size of the training set
- Neural networks:
    - Accurately identifies specific objects in video
    - Creates reference between object names and their label numbers
- Post-processing:
    - Converts locations on image to real-life coordinates (in feet) on a basketball court
    - Maps player positions on a standard image of a court

    - Plots points on one of two coordinate systems, each with an origin under the a basket
- Animation:
    - Creates an animated video of the normalized court with the ball and players represented with dots

Our non-functional requirements include that:

- Our goal was to create a model that processes prerecorded video after the game takes place and produce accurate results.
- A stretch goal of the project was to have the model process video and display stats in real-time, this would have been worked on after the immediate goal of displaying positions.
- The video fed to the model is from a single, fixed camera position where the quality has to be at least 416x416. This camera placement must be higher up in the audience stands, so the neural network can adequately determine the essential landmarks of the court.
- Post-processing:
    - Represents detection boxes as singular points
    - Converts the image plane to the normalized court plane
    - Converts specific points on the image plane to the court plane
    - Converts between image distance units (pixels) and real-life distance units (feet)
    - Determines what side of the court the image is on
    - Matches landmarks on the image to standard court landmarks
    - Determines the ball's location while accounting for an offset on the z-axis

System Architecture

For the overall architecture of this project, **Figure 1** shows the current Neural Network where the teams, the ball, and the backboard can already be identified. We implemented the new functionality of identifying the positions of each of the players and the ball from the video. Our design architecture can be seen in the lower part of **Figure 1**, where each component of our process is laid out, some of which were supplied to us by the client such as the neural network. The neural network was given to us by the client, which we used for several sessions of training to test the various input methods. The several steps required to prepare and train a neural net to produce these results are detailed in that image. The normalized positions of the players and the ball are relative to the position right under the basket, normalized for a full court. This is shown in **Figure 2**, where the "+" sign marks the position of the basket.

Figure 1 - Architecture and Design Diagram

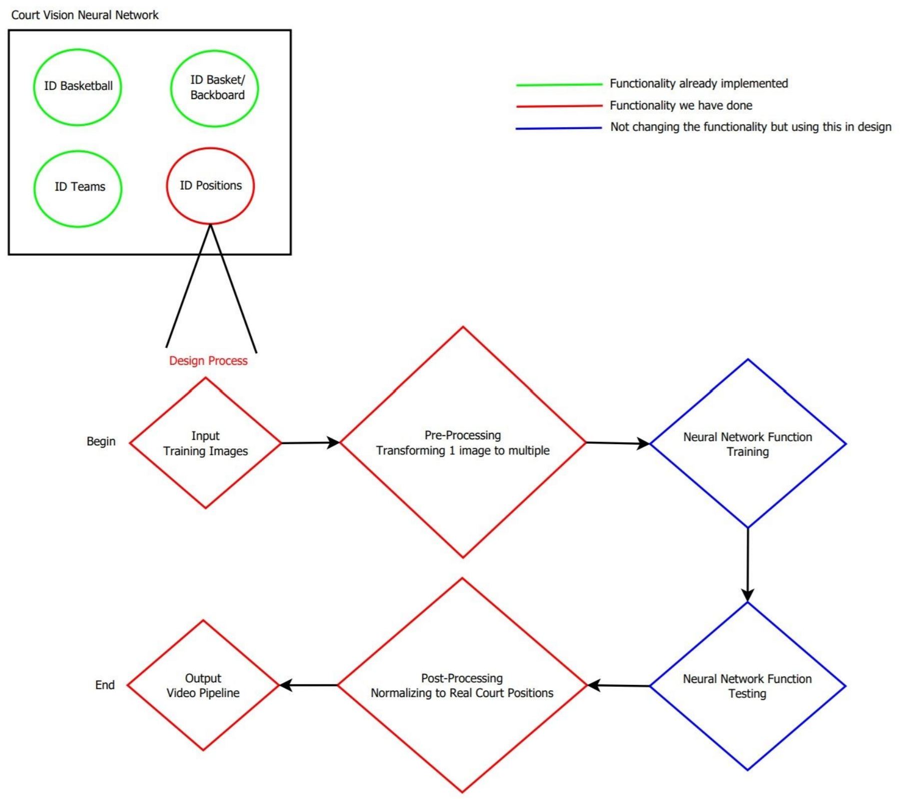
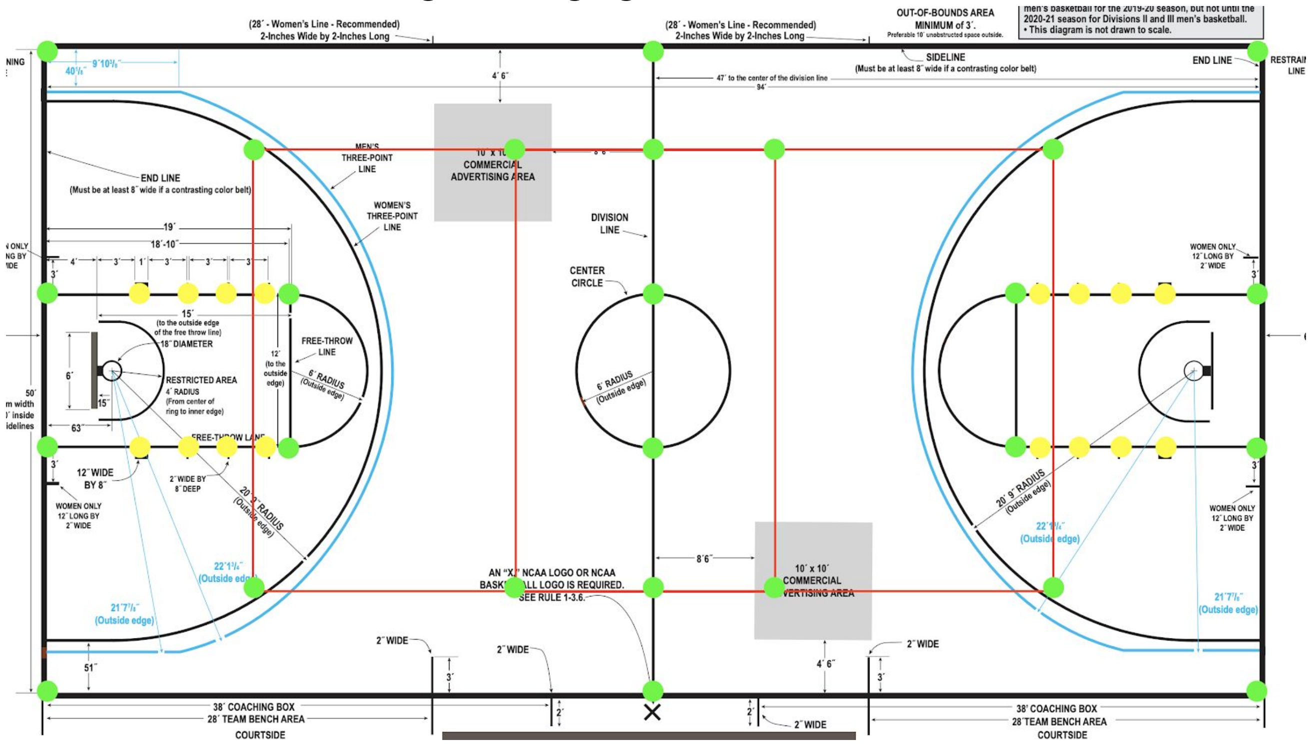

Court Vision Neural Network
ID Basketball
ID Basket/ Backboard
ID Teams
ID Positions
Functionality already implemented
Functionality we have done
Not changing the functionality but using this in design
Design Process
Begin
Input Training Images
Pre-Processing Transforming 1 image to multiple
Neural Network Function Training
Neural Network Function Testing
Post-Processing Normalizing to Real Court Positions
Output Video Pipeline
End

Figure 2 shows all the possible court landmarks with green dots. After the training of the model, at least 4 of these landmarks are visible in most of the frames. 4 corresponding points are necessary in order to calculate the homography matrix to turn the plane of an image into another. The exact equation and mathematics are explained more in the *Calculation* section of the **Appendices**. With the video coordinates known relative to the landmarks, we can normalize these coordinates using the homography matrix to a court (x,y) coordinate.

Figure 2 - Highlighted Court Landmarks



As stated before, the input video to the neural network is from a singular camera feed at a fixed point and fixed angle, and this camera pans to focus on the action of the ball. Because of this, only about half of the court is ever visible at one time.

For the preprocessing method, we chose to do transformations on selected frames of the video to turn 1 training image into 4. This is done by rotating and zooming in or out of the image. This is shown in **Figures 3** and **4**, respectively.

Figure 3 - Rotated Clockwise Pre-Processing

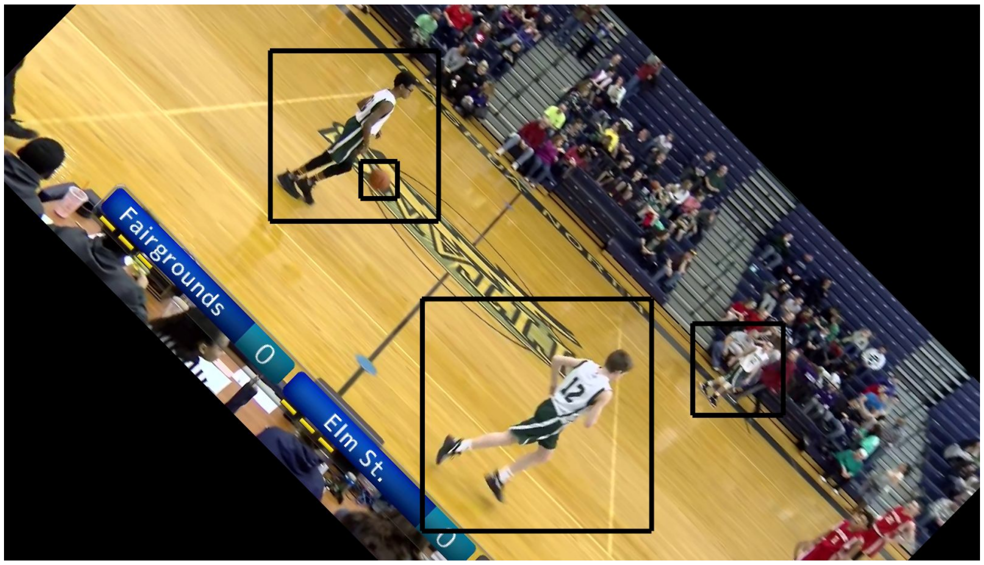
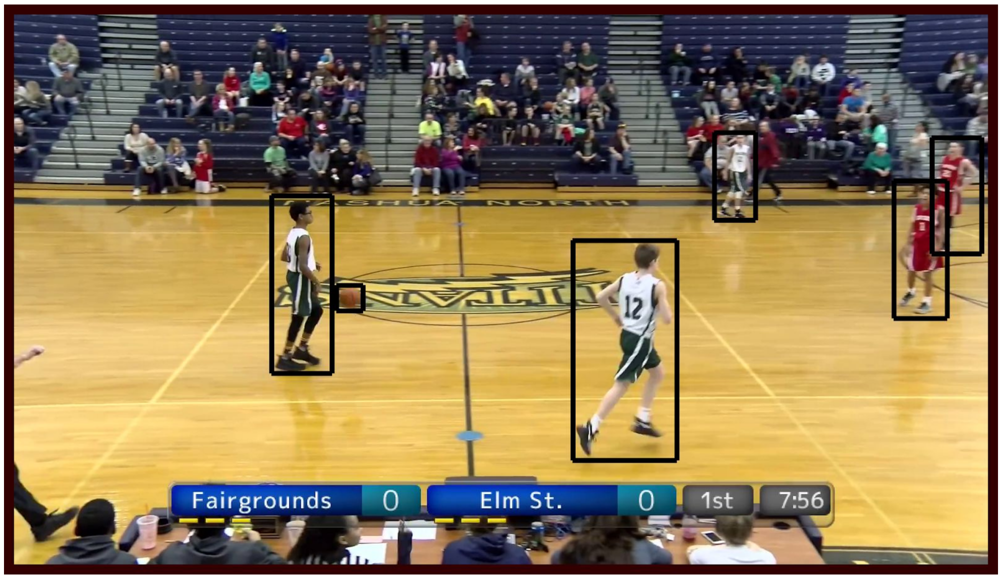


Figure 4 - Zoomed Out Pre-Processing



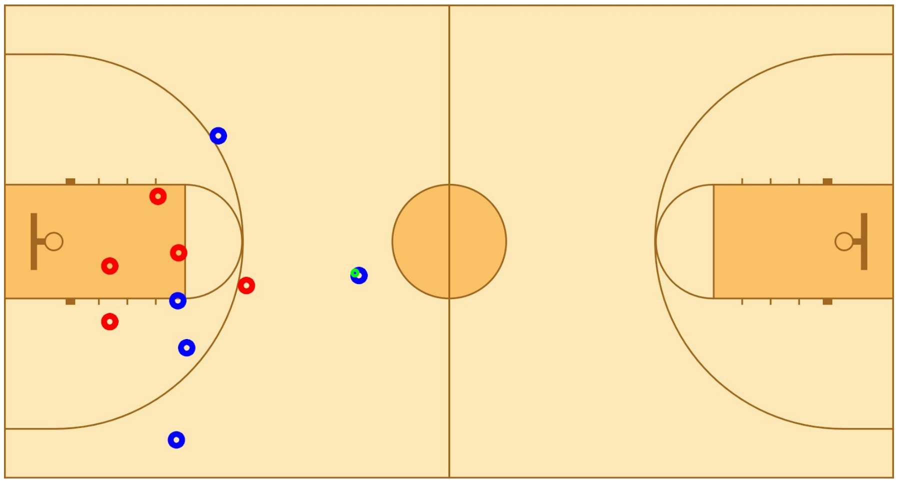
This was done because when training the neural network with only the given video, it did not have enough data to accurately pinpoint certain landmarks. A neural network makes decisions through interconnected nodes with applied weights, and training a neural network is the process of applying these weights. An example of a difficult to identify landmark would be the free throw lines because there are usually many players standing over them which results in a disproportionately smaller amount of them visible in the training data. Therefore the frames selected for this transformation included these (hard to find) landmarks, greatly increasing the amount of training data it has for these points.

For the neural network, this was given to us by the client. That is why in **Figure 1**, the legend says we did not change the functionality of the neural network but it was essential in the design process. We originally tried to get the neural network to identify players as well as the landmarks in the video, but we found that actually using 2 instances of the neural network gives much better results. After the neural network is fed a video, it produces a CSV file. This file stores all of the coordinates where an object or landmark, depending on the neural network, was identified for each frame. A separate txt file is also created at this time which contains the data required to convert the numerical labels used in the CSV to their actual names. The two neural network instances, one for objects and one for court landmarks, each make a result CSV and txt file. We then combine the CSV files using the common data of the video frame number to more easily process the detection results from the two neural networks. The transformation of the normalization can then be run with the combined data.

The role of the post-processing normalization was to convert the object coordinates from the video into their real-life coordinates on the court. The transformation comes in two files: a file that does not run and contains all of the functions used in the program, and another file that is run and transforms the coordinates for a video. The latter file writes all results to a new CSV which can then be read in the next step, animation. The CSV data is also useful later on as it contains the data that the final market product will use for many statistics.

The aforementioned CSV file is used to create an output video of an animation on a normalized basketball court, shown in **Figure 5**. This is just a snapshot shown here, but for the final user acceptance testing, it is a full animation representing the basketball video being tested on. It shows the dots moving around the court just as the players and the ball do..

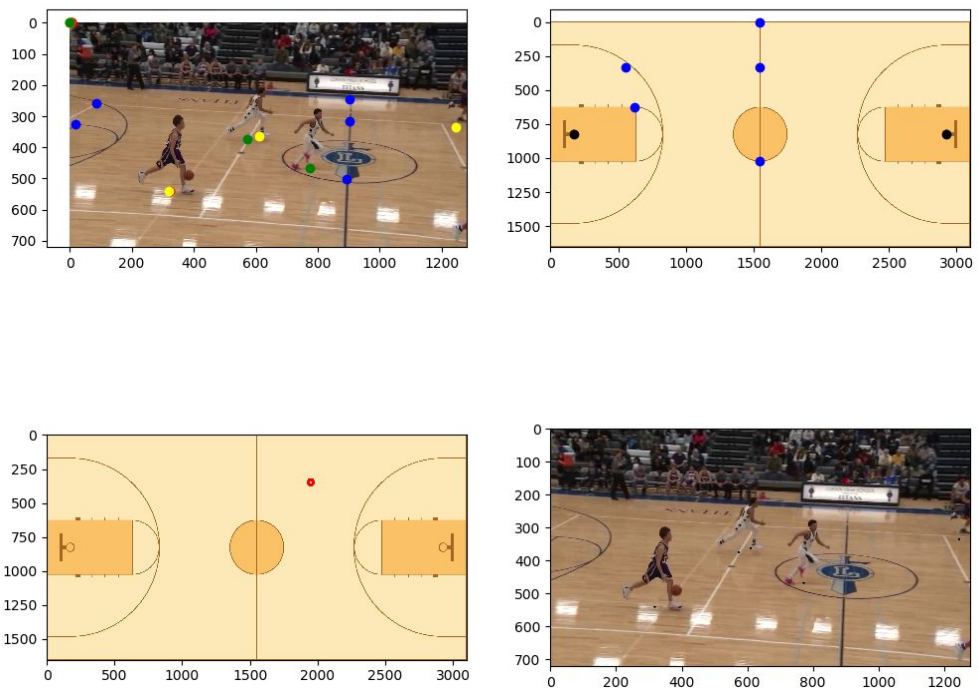
Figure 5 - Example of Animated Output Video



Technical Design

One interesting aspect of our project is that because the input is from a single camera feed, there are frames in which the camera is looking at the middle of the court and neither backboards are in view. This is intriguing because our coordinate system uses the backboards as the base point, so when neither is in view it is difficult to figure out which landmarks are on which side of the court in the code. Also, because the court image has to be cropped to roughly match the section of the court visible in the frame, results fail often at center court, which can be seen in **Figure 6**. The accuracy is best when the frame captures just one side of the court and has a lot of landmark points; this makes the middle of the court a worst-case scenario for the program as currently constructed. Unfortunately, we did not have time to find a solution to this issue. However, we found that this is not a make-or-break issue because the ball is only in the middle of the court for a small percentage of the time.

Figure 6 - Middle Court Example



For explanations of each subplot, see the caption to **Figure 9**.

An additional aspect of our final design that was implemented later on is that we used two neural networks for detection rather than just one. Originally, the plan was to use a single neural network to detect both landmarks and objects. This approach better fits with the idea of YOLO (details in the *Modeling Technique* section of the **Appendices**). However, we discovered that with the use of two different data sets combined, one for landmarks and one for objects, the neural net was negatively reinforcing itself on images that did not have both types of tags. This meaning that when the neural network was training on an image from the landmarks data set, it would see players but since they were not labeled they would not only be ignored but the neural network would actively teach itself that those people are not to be tagged despite them being examples of a dark color and light color team. We additionally tried training on only the intersection of the two data sets, but that drastically reduced the size of the data set available to train on. We decided finally to split off the neural net into two parts, one for objects and one for landmarks, the results of each are seen in **Figures 7 and 8**. For each tag that the neural networks recognize, a bounding box is drawn, and the coordinates of that box are stored. Since we could train each on its own independent data set, the resulting detection results were much more accurate.

Figure 7 - Landmark Detection

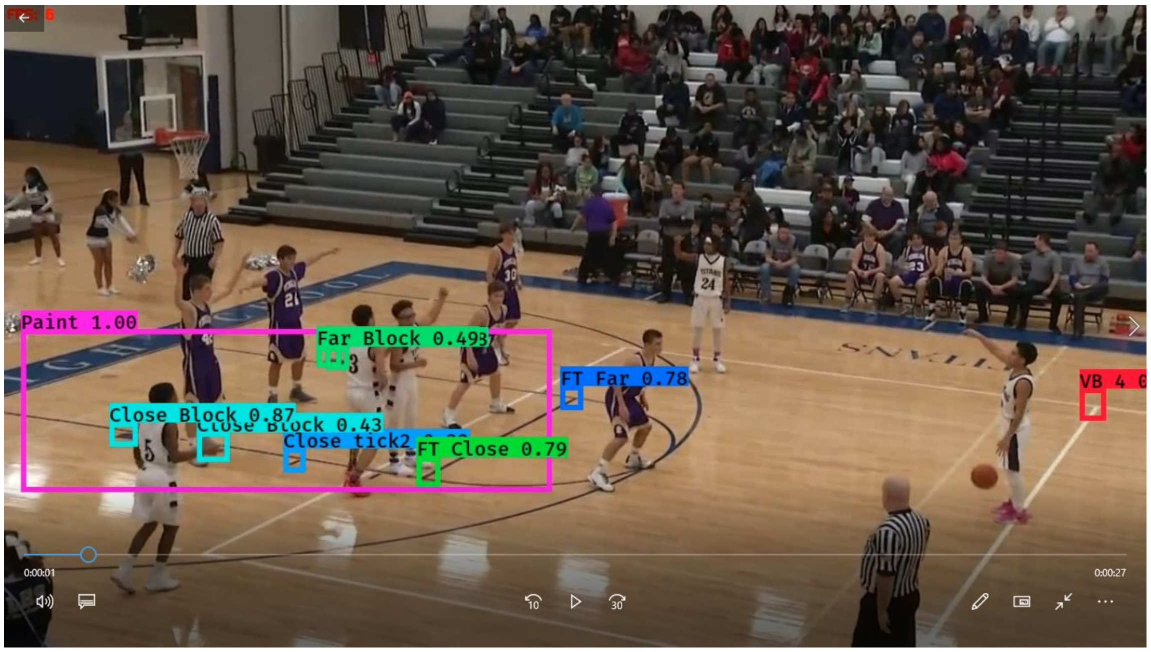
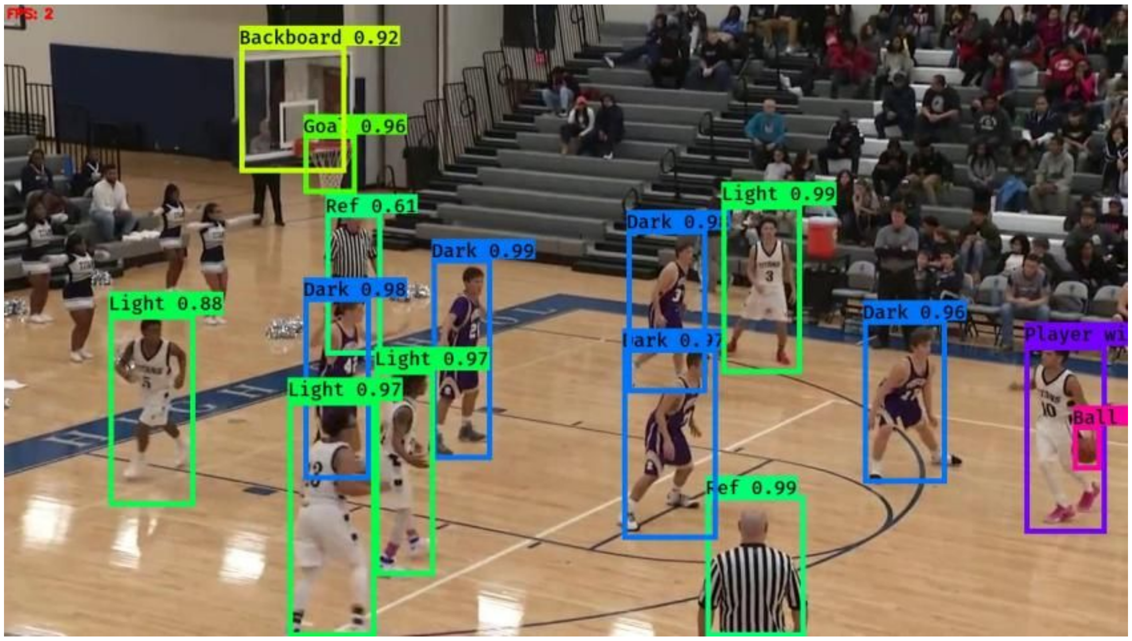


Figure 8 - Object Detection



For future work, a solution could be to expand the logic of the code we have right now. The code goes through a series of object/landmark checks to try to determine which side of the court is in view. If any of the steps return a reasonable guess to which side, it breaks from checking the rest of the points. The points checked in order are:

1. Any backboard detected in the frame
2. Any paint landmarks
3. The half-court line

4. Any detected blocks, such as the volleyball lines (which only some high school courts have)

If a reasonable guess cannot be gathered from checking the above points, it is assumed the courtside the camera is looking at to be the same as the previous frame in the video. To expand this solution, maybe the last case could be tracking the average direction each detected player is moving from the last 10 or so frames. If the average direction of all the players is to the right and there are no useful detected landmarks, a reasonable guess could be the frame in view is the right side or at least going to the right side.

Another task for future work would be to handle frames with less than four landmarks detected. One patch for this would be to assume the same homography matrix from the previous frame. While this would not be as accurate as computing a unique homography matrix with four landmarks, it can be assumed that the differences between most consecutive frames would not be very much; unless there is a fast break in progress, the camera stays pretty steady and only slowly changes angle. It might be better to make the assumption that h matrices are similar between consecutive frames and accept the risk of inaccuracy than to skip an entire frame from the video.

Points are transformed by drawing a green dot on the image; this is so that when the image is warped the point is warped along with it. After the image is warped the dots on the original image are covered up by black dots so that they will not show up in the processing of other objects. Once a warped image is present, it is binarized so that only the green dots are white with everything else being black. This makes it easy to identify the points on the court plane and get their coordinates.

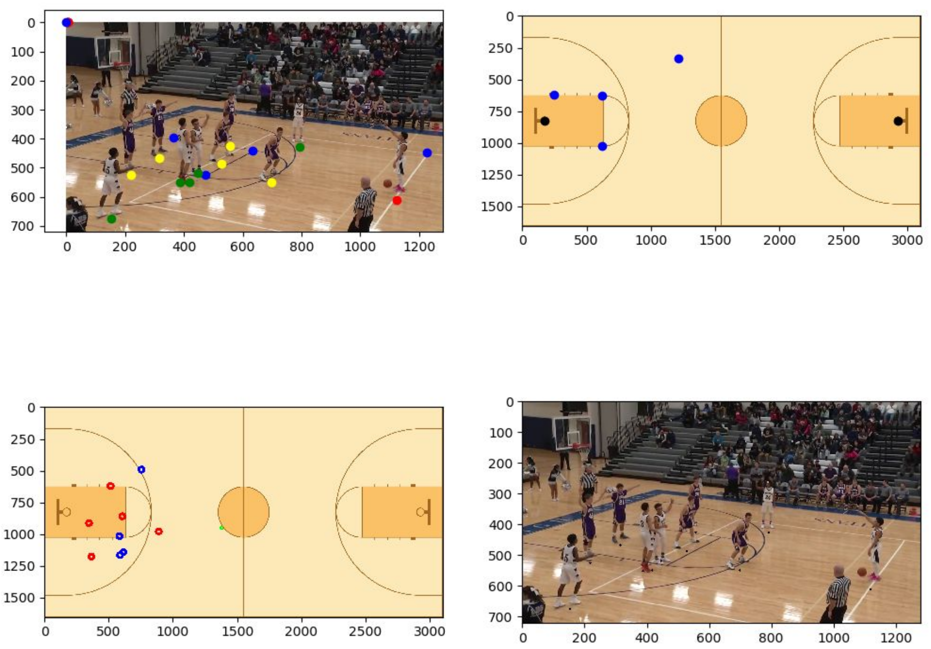
Quality Plan

Unit Testing

Coordinate transformations had to rely on the eye test. A more objective test would have been to transform video in which player coordinates were known in each image, but that was not available to us and would have been far too tedious to set up in the time available. The eye test was more than adequate. **Figure 9** shows the subplot layout that provided the information needed to check all parts tested. The following aspects were what were evaluated in testing:

- All points on the image were within the bounds of the image (**Figure 9**, row 1 column 1)
- Landmarks on-court image corresponded to landmarks labeled on the image (**Figure 9**, row 1 column 2 and match to row 1 column 1)
- Objects transformed on-court image matched the real-life image reasonably well (**Figure 9**, compare row 2)
- Minimize deviations between consecutive frames
- Majority of transformations were accurate
- Percentage of the video frames that could be processed (0.68% for test video)
- Execution time

Figure 9 - Checking the Normalization through Eye Tests



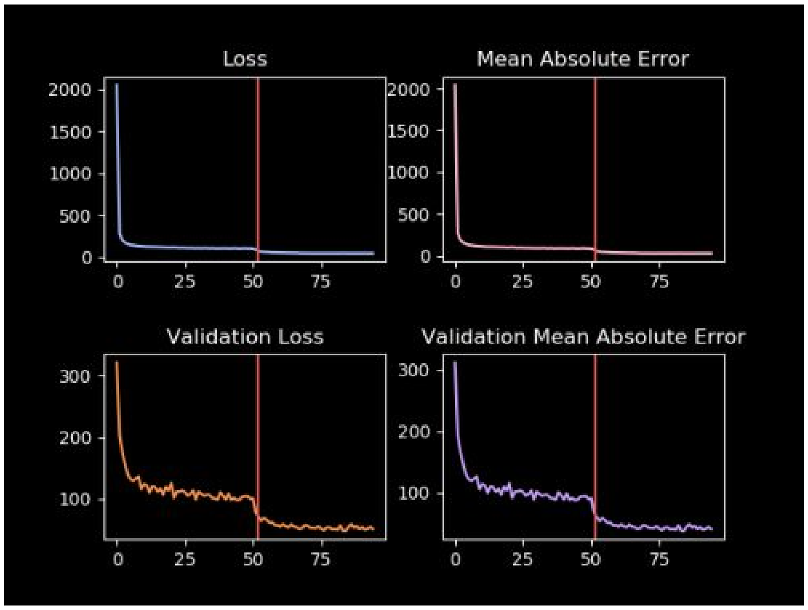
Row 1 column 1, input image with light players denoted by green dots, dark players by yellow dots, the ball by a red dot, and landmarks by blue dots; row 1 column 2, standard image of the court with blue dots correspond to the landmarks in the image to its left; row 2 column 1, visual representation of the output on a standard court where red represents dark players and blue represents light players; row 2 column 2, input image.

For the video pipeline, one essential test was that the video shows both a light and dark team in addition to the ball for most frames of the input video, as some objects are not always in view of the camera. For the coordinate transformations, the transformed images to a normalized court must first pass the eye test, a colloquialism for visually inspecting the output to some degree.

In regards to execution time, a GUI gave a rough estimate, but because some frames are unprocessed the average time per frame is lower than what was calculated; as a greater percentage of the video is able to be processed, the time given by the GUI increases and becomes more accurate. The alternative method used to get accurate execution time tests was to print out the elapsed time of each processed iteration using the clock function from the time library.

When it came to testing the neural network itself we relied primarily on metrics generated during the training process. Throughout the training of the network, we log the loss and the mean absolute error for each epoch's training set and validation set. This data is then graphed to show how the model behaved over time and to make sure everything went as expected. For example, if the validation loss started to increase during the later epochs, meaning that the sum of errors in the validation set started and continued to increase, this would mean that the model began to overfit or take the data too literally, hindering its ability to make accurate predictions with new images. We also separately logged the final values of the aforementioned metrics so that they could be easily compared to other neural networks that we trained with various preprocessing methods in place. From these comparisons, we were able to determine the impacts that these preprocessing methods had and choose a path forward. Not all testing was done during the training phase however, we also have a script that operates on the output generated by the detection portion of the program where the neural net is given a video to identify landmarks or objects in. A CSV file is generated during detection that gives all of the tags found and the confidence in that tag. Utilizing this data we can understand the average confidence for different types of tags, the tagging as a whole, and the total amount of tags found by the neural network. This makes it easier to understand how the neural networks perform on different courts, teams, camera angles, etc. An example of a graph is shown in **Figure 10** where the vertical red line represents where the neural network transfers from doing the bulk of the learning to a fine-tuning stage.

Figure 10 - Graphs of Loss and Error



Integration Testing

Testing that the new functionality of the model meshed with the already existing functionality was one of the most important aspects of the project. This is because our solution must integrate correctly in order to make an all-around product. The solution to identifying real-life coordinates must incorporate with the other functionalities of the project that are already completed in order to satisfy the client. This was tested using a video tagging software and it's output videos to first make sure everything was correctly identified such as the landmarks and objects. Then for the final output video, it must include the light and dark team players, their positions, the ball, and the position of the ball.

Code Reviews

Throughout the project, we performed code reviews as the code was being written due to extensive pair programming on most of the new functionality. We have in addition done one large code review where we met online and reviewed our documentation and code.

User Acceptance Testing

Some performance tests that were performed were running the code, making sure it recognizes both landmarks and objects before the transformation and then making sure it outputs a video of an infographic basketball court with players and the ball as dots in the video with normalized court coordinates.

Code Metrics

There was no specific code structure or documentation required by the client, so we have decided to follow general guidelines taught from classes such as clear variables, starting names with non-capital letters, and proper indentation.

Static or Dynamic Program Analysis

For this project, static analysis was done for every step of the process. This is because, for most of the problems, we had to make/run the code or train the model, and then wait for the process to be done. Afterward, we could study the output and backtrack any errors or issues.

Results

Referring back to our original definition of done: “The minimal useful feature set is the coordinate system of the basketball court using landmarks to identify the positions and then displaying the actual locations using the normalized coordinate system of the court. One test the product will have to pass is when fed sufficient training data, using a new set of testing data can the algorithm correctly identify “clear” positions, such as a non-concealed, single-player standing on the 3 point line.”

We have achieved what we described as the minimal useful feature set as well as setting up appropriate future work such as expanding the neural net to recognize individual players, recognize which team has the ball and stretch goals that we were unable to implement. These goals primarily dealt with just improving the baseline functionality we already have in ways such as remembering where players and landmarks are so they can be mapped to the normalized court even when they are obscured. Right now our model works best at a camera location centered on the court with a bit of elevation. We were unable to take into account a multitude of other camera angles so while there should still be some success at various camera positions and angles, due to the nature of our training data it will most likely not be as extensive as the aforementioned camera situation. Given the current situation of the world, there were certain difficulties with collaboration on the project. Specifically, pair-programming and keeping each other working together at the same time of day was difficult. Despite this, our team learned a plethora of valuable lessons ranging from technical skills to working with people through new mediums. Due to our collective inexperience with machine vision, this project shed light on just how versatile neural networks are. Contrary to our initial beliefs they do not require an exuberant amount of preprocessing to function, in fact, they are capable of great results on relatively small data sets and zero preprocessing. We also learned that despite their aforementioned versatility neural networks perform better when given a specialization, more so in our case where we have two sets of training data with little overlap.

Besides our initial lack of technical knowledge, one of the biggest challenges as a team was working exclusively online. It also did not help our situation that our team was split across time zones. Despite this, our team gained the ability to better plan out each week and block out times to work on the project and communicate our progress and future plans to one another.

Appendices

Development libraries and files:

- OS was used for directory manipulation and file structure.
- CV2 (opencv-python) was used for the homography matrix.
- csv (python-csv) was used for the result files of the transformations.
- Numpy was used throughout the entire architecture.
- Pandas was used for .csv file manipulation.
- Imutils was used to get normalized coordinates.
- Matplotlib was used for animation.

Calculation details:

A homography matrix (H) is a transformation that maps corresponding points in 1 image to another image using the same plane, in our case, this is the image of the court in the video and the real-life court with known measurements. The equation $P = HPw$ is true for all sets of corresponding points as long as they lie in the same plane in the real world, P is the matrix of the points from the image while Pw is the matrix of the real world points (AKA the normalized court). This can be seen in **Figure 11**, where the two camera angles represent the different frames of our videos (as they have different camera perspectives) and the rectangle projected on in the background represents our normalized court. The homography matrix transforms every image from each camera angle into the normalized court. However, at least 4 corresponding points are required to calculate the homography matrix. On our code, we use an open-source library, opencv-python, to estimate the homography matrix for us. In **Figure 12**, this is a normal frame of a video from a basketball game, but the next image, **Figure 13**, is normalized using the homography equation. As you can see from this, the lines of the court in the second image line up geometrically nicely as a normalized rectangle. The positions of the players are calculated by the bounding boxes shown earlier, where the center of the bottom line is estimated to be the player's feet and marked as their positions.

Figure 11 - Expanded Homography equation and Visualization

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

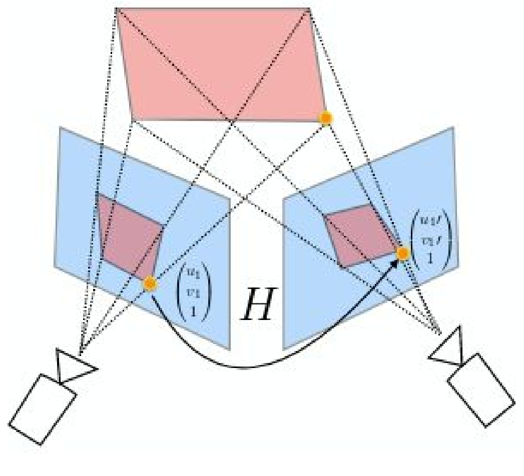

$\begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix}$
H
$\begin{pmatrix} u_1' \\ v_1' \\ 1 \end{pmatrix}$

Figure 12 - Basketball Game Image

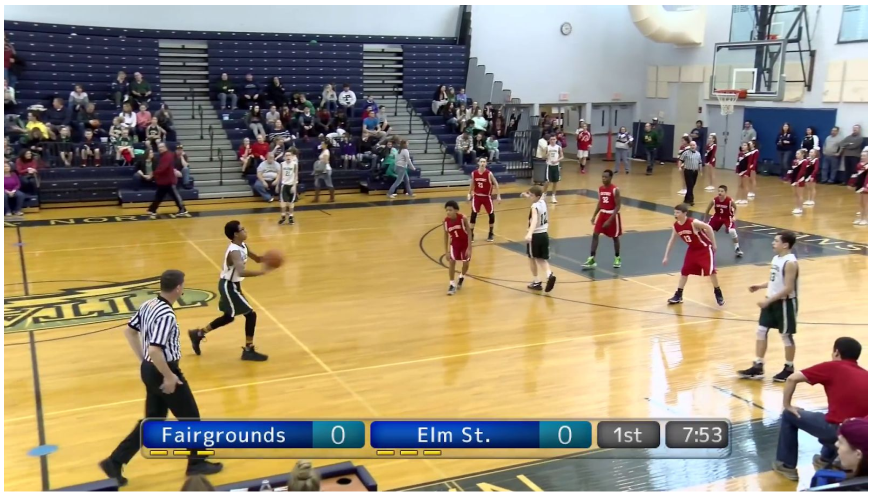
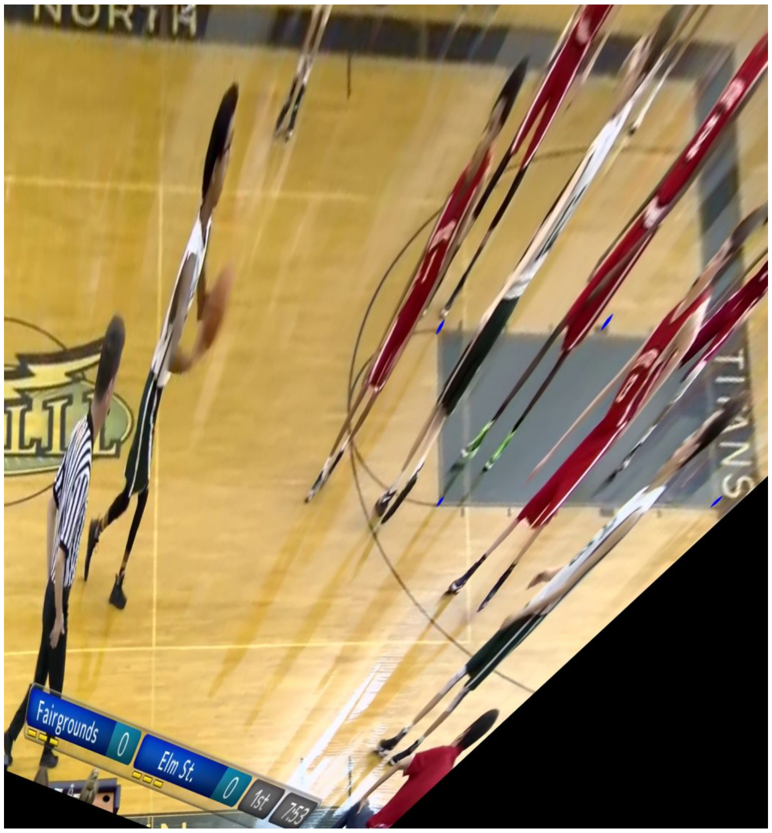


Figure 13 - Normalized image of Figure 12



Modeling Technique:

The modeling technique that is the base for the project is YOLO - You Only Look Once. The principle behind this model is that each image or frame of video is only processed once, therefore it is much faster than other neural network approaches. It is an open-source library that employs real-time object detection and forms predictions from a Convolutional Neural Network (CNN). In our case, it is more appropriate to use a convolutional neural network as opposed to a more standard neural net. CNN's are more suited to images and visual data due to their use of a filter that processes small parts of the image one at a time. CNN's are often compared to neurons in the brain because of their pattern connectivity. A "neuron" in a CNN is just like a neuron in the human brain, as it is specialized for receiving certain environmental factors, and then passes its information along with every other neuron in order to form a full picture. Each neuron in a CNN takes a certain aspect or functionality for determining what the object is or isn't.

Our implementation uses two separate neural networks where one recognizes the players and goal while the other recognizes court landmarks. This is to make the most out of limited data sets with little overlap.

Coding Conventions:

Our client did not have any preferred way of documentation or coding conventions, so we followed the coding conventions taught from Colorado School of Mines described in the Code Metrics. We also added a lot to the README documentation so the project can be continued where we left off.

Acceptance Tests

Because our project is like a demo or proof of concept for our client, our client is the user. His expressed acceptance tests have only been the eye test in terms of accuracy of the positions of the objects in the video. A final acceptance test will be showing our demo video to our client.