Trajectory Tracking of a Set Volleyball with an iPhone using Deep Learning Models

Except where reference is made to the work of others, the work described in this project is my own or was done in collaboration with my advisory committee. Further, the content of this project is truthful in regards to my own work and the portrayal of others' work. This project does not include proprietary or classified information.

Brian Malsan

Certificate of Approval:

Micheal Reale
Assistant Professor
Department of Computer Science

Bruno Andriamanalimanana
Associate Professor
Department of Computer Science

Jorge Novillo
Professor
Department of Computer Science

Trajectory Tracking of a Set Volleyball with an iPhone using Deep Learning Models

Brian Malsan

A Master's Project

Submitted to the
Graduate Faculty of the
State University of New York Polytechnic Institute
in Partial Fulfillment of the
Requirements for the
Degree of

Master of Science

Utica, New York
May 10, 2019

Trajectory Tracking of a Set Volleyball with an iPhone using Deep Learning Models

Brian Malsan

_____
Signature of Author

_____
Date of Graduation

Master's Project Abstract

Trajectory Tracking of a Set Volleyball with an iPhone using Deep Learning Models

Brian Malsan

Master of Science, May 10, 2019
(B.S., State University of New York Polytechnic Institute, 2018)

27 Typed Pages

Directed by Michael Reale

Computer vision and deep learning techniques have seen an increase in use in the world of sports. One sport that this paper has interest in is volleyball. This work presents an iOS application to be used to determine the trajectory of a set volleyball using deep learning models. Using training data that I gathered, a You Only Look Once (YOLO) object detection model is trained and then converted to a CoreML model to be used in the application. This provides a fast way to detect a volleyball, net, and person. The goal is to then display the trajectory that the ball travelled to the screen so the player is able to adjust and get better. The easy to use capabilities of the application can be used to improve the setting abilities of players of all skill level. Also, the application could be easily modifiable to work with other sports.

Acknowledgments

I would like to dedicate this project to my parents for supporting me in everything I do. My brother for helping me discover some of the issues. My roommates, Ian Kurzrock and Derek Smith for keeping me focused. Lastly, my teammates and coach for helping come up with the idea.

Style manual or journal used Journal of Approximation Theory (together with the style known as "sunypolyms"). Bibliography follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX2e) together with the style-file `sunypolyms.sty`.

Table of Contents

1 Introduction 1

2 Related Work 3
2.1 Object Detection Models . . . 3
2.2 Computer Vision in Sports . . . 4

3 Method 6
3.1 YOLO Models . . . 6
3.1.1 YOLO9000 . . . 6
3.1.2 YOLO v3 . . . 7
3.2 Training Pipeline . . . 7
3.3 Model Conversion . . . 8
3.4 iPhone Code . . . 8

4 Results 10

5 Discussion 13

6 Conclusion and Future Work 14

Bibliography 15

Appendices 17

A Installation of Training System 18

B Train and convert models 20

Chapter 1

Introduction

Competitive sports have provided an interesting field of research for computer scientists. The use of computer vision and machine learning techniques can provide players and teams an edge over their opponents. One sport that a computer vision application can be used to help players improve their skill is volleyball. However, sometimes set up of such applications could require a lot of time and equipment.

Volleyball is a sport that can be played indoor or outdoor, for the purposes of this paper we will only deal with indoor volleyball. The court is 30 feet wide by 60 feet long, with a net in the middle at heights of 8 feet tall for the men and 7.4 feet tall for women [1]. At a given time each team has six players on the court split up into different positions. The teams have their hitters, their passers and a setter. The toughest position of those being the setter.

The setter is responsible for getting the ball to a hitter in the best way possible so they can score a point. There are many different locations that can be set at any given speed, and every player likes the ball to be set to them in a different way. Generally there is a

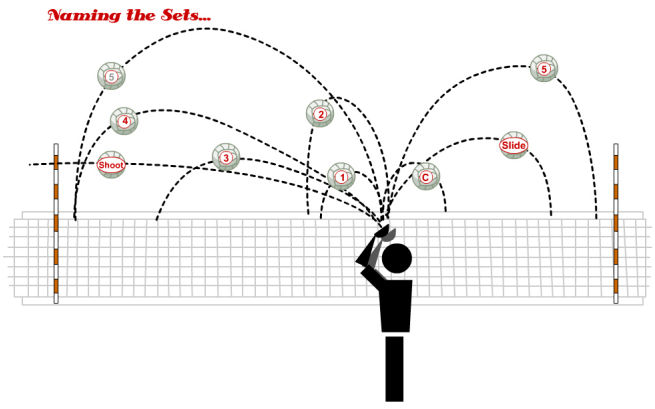


Figure 1.1: Setting Trajectories [2]

perfect trajectory that a setter should put the ball on to the hitter. Figure 1.1 displays the many different trajectories that a ball should be set at. Sometimes getting feedback from your hitters can be difficult. With the use of an application that tracks the volleyball then the setter can have quick and accurate information to adjust how they are playing and hopefully improve.

This paper presents a mobile application that utilizes a "You Only Look Once" (YOLO) object detection model to track the trajectory of a set volleyball. The model is trained on a dataset that I collected myself, and is used to find a ball, person and net in an image. With the model converted into a format that an iPhone can use, the model is tested at real time with a video feed from the camera. With the three items detected the trajectory of the ball will be displayed to the user.

Chapter 2

Related Work

In computer vision the concept of object detection is the ability of a computer to determine where a object is located in an image and what type of object it is. The techniques that do this are object localization and object classification [3]. Once you know where and what the object is, you are able to use that information in a plethora of different ways. An example of this in sports is being able to tell the difference between a player and a ball. To a human this is very easy, but to a computer a lot more work needs to go into it.

2.1 Object Detection Models

There are many different types of deep learning algorithms/models that can be used to do object detection. They all have their own advantages and disadvantages, most are built off of the others to make them faster and more accurate. The reason a normal CNN model would not work for object detection is due to the fact that the length of the output vector is not fixed, because the number of objects in an image is not fixed[4]. The models that are relevant to this project are R-CNN, Fast-RCNN, Faster-RCNN and "You Only Look Once."

The first and most simplistic out of these are region based convolutional neural network (R-CNN). R-CNN was first introduced in a paper by Ross Girshick et al. in 2014 [5]. It describes a system made up of three methods. The system splits up the input image into 2000 regions, then extracts the features from each region using a convolutional neural network, and lastly classifies each region using class-specific linear SVMs to specify if an object appears in that region [5]. It performed better than previous approaches however

not fast enough for real time use. It takes around 47 seconds to test each image and takes a long time to train[4].

The same creator of the previous approach introduced a new model aiming to solve some of the drawbacks. In 2015 Rob Girshick introduced Fast-RCNN [6]. The model takes an image and multiple regions of interest as inputs, extracts feature map, and gives two outputs. The first is the softmax probabilities of the classes and the second is the offset values for the bounding box. These outputs appear for each region of interest [6]. The speed up is due to the fact that you do not have to feed 2000 region proposals through a convolutional network every time, the convolution operation is only done once per input [4]. Fast-RCNN provides comparable fast training and testing time and higher detection quality.

A year later Rin et al. introduced Faster-RCNN, which the name suggests is much faster in testing than the previous approaches[7]. The goal of Faster-RCNN was to increase the speed of the region proposer section of the network. Previous approaches generated regions of interest using a Selective Search algorithms, where Faster-RCNN does not. The biggest advantage of this technique is only a single CNN has to be trained, which does both region proposals and classification.

The last network that is relevant to this paper is called the "You Only Look Once" or YOLO network. It was introduced in 2015 [8]. This one is a little different from the others but has given promising results. It is currently up to version 3. Instead of applying to only parts of an image a YOLO network applies a single CNN to the full image, splitting it up into multiple regions. For each region, a predicted bounding box and probabilities for the object that could be there is outputted[9]. This model greatly out performs the previous.

2.2 Computer Vision in Sports

There has been an increase in the development of computer vision technologies related to sports. Due to the fast pace nature of most sports, neural networks can be use to keep track of the balls and players. Some things that computer vision application could be used

for is to take game statistics (during or after), make calls, or help athletes improve their skill.

There has been many implementations of computer vision in the world of volleyball. Volleyball is a challenging sport to track the ball due to the small size, high speed that causes motion blur, and is a lot of the time occluded by people. In [10] they introduce a technique to track a ball using a physically constraint model that takes into account the movement of the players. They formulate the problem as a Mixed Integer Program, and model the ball in three dimensional space. For their model to work it is necessary for multiple cameras to be covering the playing area.

In 2016, Takahashi et al. published a paper that described a way to track a volleyball in three dimensional space in real time [11]. They trained a model that took input from four different cameras. The model is split up into three modules: one for ball detection, one for 3D position measurement, and the last for ball position prediction [11]. The difficulty lies in extracting 3D information from a 2D image, this is why multiple cameras are used.

Through my research I have not found any mobile applications that do tracking of a volleyball. However, [12] introduces a smartphone application that tracks a pitched baseball from the mound to home plate. This application is able to determine the velocity of the ball and determine the deceleration of the ball. To keep track of the ball they do background subtraction, smoothing and then a series of erosion and dilation to emphasize the ball [12].

Chapter 3

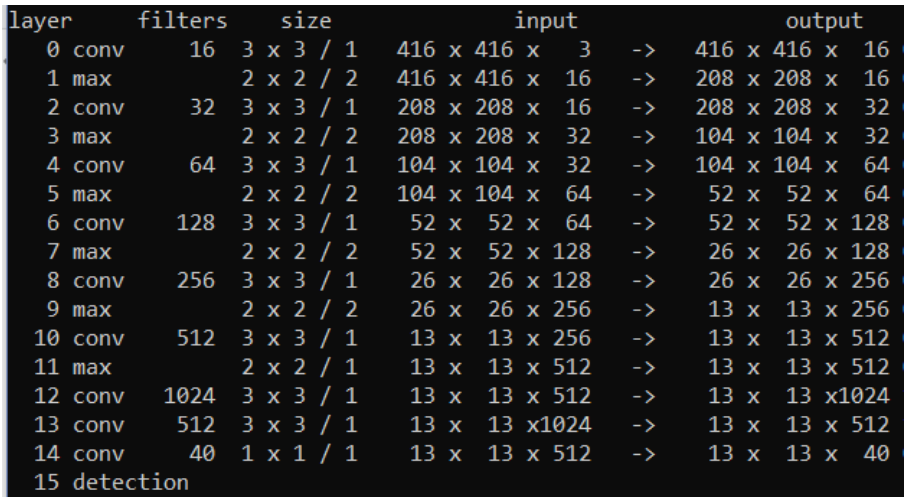
Method

For my project I trained a multiple YOLO models to track the trajectory of a set volleyball, running on a iPhone. A lot of the work in the project was around setting up and training the models.

3.1 YOLO Models

3.1.1 YOLO9000

One of the models that I used was a YOLO9000 model [13]. Figure 3.1 shows the layout of the model.



```
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   13 conv    512  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x 512
   14 conv     40  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x  40
   15 detection
```

Figure 3.1: Model Structure

The model is a single convolutional neural network that for input takes 416x416 images. It is made up of many convolution and maxpool layers, with the last layer being a detection layer,as the name states, detects if an object is in an image. The image is split up into a 13x13 grid, where in each region a specified amount of bounding boxes are created. The

bounding boxes will hold information on weather or not one of the objects you are looking for is found. The model outputs a vector where the size depends on the number of classes and number of bounding boxes per region. My model outputs a vector of size 40x13x13. The 40 is gotten from (numClasses + 5)*boxesPerRegion, I had three classes and was finding five boxes per cell. The vector holds the center point(x,y), height and width of the bounding box and also the confidence value for the class.

3.1.2 YOLO v3

The second model that I trained was a YOLO v3 model. YOLO v3 is the latest version of the model. This model made slight improvements on the previous versions, that made it more accurate. The model still takes a 416x416 image as input and outputs the bounding boxes and object class. This model no longer uses softmax for class predictions, instead replaces it with independent logistic classifiers that improve performance [9]. They also provided a new approach to feature extraction. The model uses feature maps from early layers and concatenates them to up-sampled features from later layers. This allows for more meaningful information and finer detail [9]. The model handles detection of smaller objects better than the previous versions.

3.2 Training Pipeline

The first step of the training pipeline is to gather the data that I need to train the model. I went to the SUNY Polytechinic Institute field house, set up a net, and had two people helping me out. One would toss balls for me to set and the other would video tape. I set the ball in the same direction each time. We tried to do some variation by having myself change color of shirts.

Once I had the videos I needed, the next step was to extract ground truth values. I did this with the Microsoft Visual Tagging Tool v1 [14]. This tool allowed me to traverse each video frame by frame and draw a bounding box around all the objects I want the network to learn. In my case I got ground truth values for the location of the ball, net, and person

in each frame. The tool has the option to export all the information in the format needed to be feed into a YOLO model. The tool also randomly takes 30% of the frames and add them to a set dedicated to testing.

To train the model I used the darknet framework found on Github, this is the framework that the creators of the model use [15]. See Appendix A for how to setup darknet on Windows with use of a GPU. The framework also provided a way to test the models before using them on the iPhone.

3.3 Model Conversion

To be able to use a neural network on an iPhone, there is a need to convert it to a CoreML model. CoreML is Apple's machine learning framework for iOS development. The model had to be converted from a darknet model to a Keras model and then to a CoreML model, there is no direct conversion that I found. To convert from darknet to Keras I used "YAD2k: Yet Another Darknet 2 Keras" [16]. To run it I set up a Python environment using Python version 3.6. Then using the pip command, installed numpy, h5py, pillow, tensorflow, keras, configparser, and coremltools. The coremltools package is used to convert from a Keras model to a CoreML model. The code is in the Converter.py file, to run it needs two arguments: the location of the Keras model and the name you want your CoreML model to have.

3.4 iPhone Code

The code used for the iPhone application was written in the Swift language using version 5 and Xcode IDE using version 10.2. The main frameworks that the project use are AVFoundation: for the use of the camera, CoreML: for machine learning, and Vision: to do requests to test the model. I used the "TinyYOLO-CoreML" project by Matthijs Hollemans as the base of the project [17]. I had to make some changes to his code. First I made it so the model is being tested through a vision request. This is beneficial because the vision request automatically resizes the input to match that of the models input layer. Within

the Yolo.swift file I changed the computeBoundingBox function to the correct number of features(40) and classes(3) for my implementation. Since I changed the orientation of the project to always be in landscape, I changed the conversion of the aspect ratio for the bounding boxes in the show function.

I also added a function that will draw an arc on the screen, that represents the "perfect" set trajectory. Right now it will draw an arc from the center of the screen to a pre-determined location based on the selected set that the player is working on. The goal is once the player and net are detected, it would draw the trajectory from the person to the spot on the net, the arc will have a control point based on what set was selected. The control point determines the height of the arc.

Chapter 4

Results

The training of the models took place on a computer running Windows 10 and a Nividia GeForce 930MX GPU. The training had about 1000 images, did 6000 iterations, it is recommend for 2000 iterations per class, and took about 18 hours to complete. Once completed I ran through the testing suite and visually determined if it was a success or failure in detecting the three objects. The tests were ran on the same computer that did the training. For the YOLO9000 model the testing gave very poor results. For all the images the model was able to detect the net and the person, but was unable to find the ball in any of them. There would also be instances where it would output two bounding boxes for a single person. Figures 4.1 and 4.2 show these two cases. On average the model gave a prediction in around 35 milliseconds.

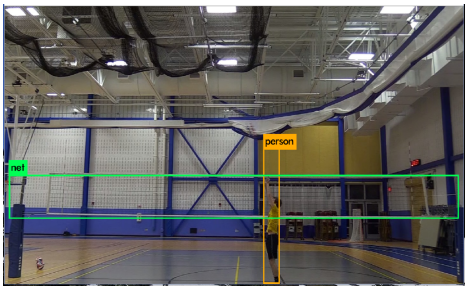


Figure 4.1: Model does not locate the ball

Figure 4.2: Displays two bounding boxes

The training of the YOLO v3 model took place on the same Windows computer with the same GPU. The training took a little bit longer but gave better results. Using the test suite of 164 images and visually determining success, the YOLO v3 model was successful in identifying all objects in the frame 82% of the time. Figure 4.3 shows a successful test of the YOLO v3 model. The reason I could not use this model though, is when running it on the iPhone it would cause the application to crash immediately.

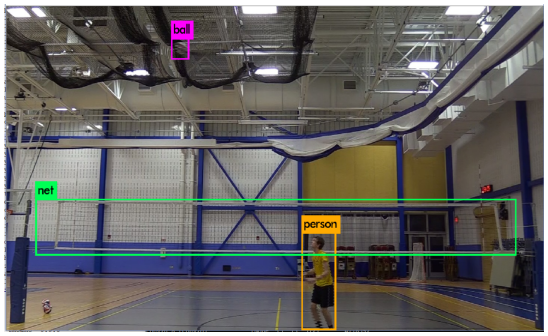


Figure 4.3: Example test of YOLO v3 model

While I was gathering training data I used a Canon XA35, rather than an iPhone, thinking it would give me higher quality data. This was a mistake, because the read out pattern of the image sensors are different between the two. This causes objects that are moving to appear different to both. So when doing testing on the iPhone the same scene will appear differently then when appearing on the other camera. The figures below show how the ball is seen by both devices.

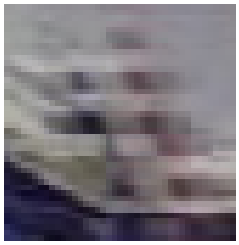


Figure 4.4: Ball moving on Canon XA35

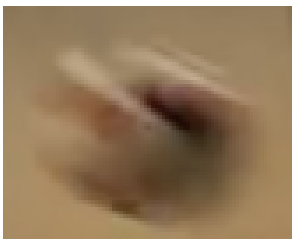


Figure 4.5: Ball moving on iPhone

Since the training data had the wrong read out pattern and was most likely over fitted, the model did not work when testing on the iPhone. I believe with a correctly trained model and a few tweaks the iPhone application would work. Currently when testing on the iPhone no predictions are being made, so no bounding boxes are being drawn to the screen. Figure 4.6 shows what the current app looks like when viewing a scene with a ball net and person. Currently the green line resembles the "perfect" trajectory of a set ball to the outside pin (left side of the net). The line is simply drawn from the center of the screen to a pre-determined point. I made it so there are three options for the arc, that can only

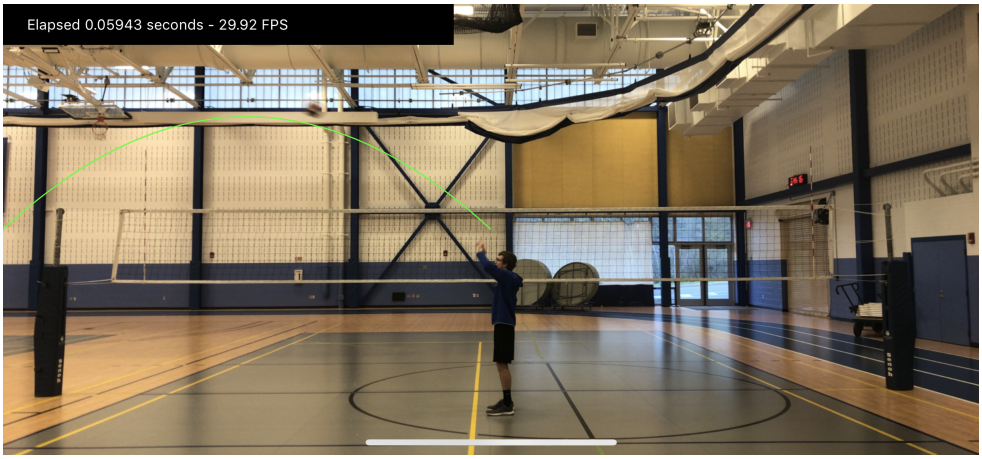


Figure 4.6: iPhone display of application

be changed at compile time. When the model could actually make predictions the values of the bounding boxes for the person and net would be fed to this function and would create an arc based on those values.

Somethings that we discussed to provide better results was to do background subtraction to remove some of the noise in the image. Subtracting the background would emphasize a moving object from frame to frame and de-emphasize what is staying still. This would be beneficial for the project since we are only tracking the ball when it is moving. Following a tutorial [18], I set up an iPhone application that uses OpenCV that subtracts the previous frame from the current frame. This should leave only things that are moving. The application also does Canny edge detection to emphasize edges in the image. The project can be found at `https://github.com/bfmalsan/Masters-OpenCV`. With the edge information and usage of a well trained model, my application could eventually provide good results.

Chapter 5

Discussion

Over the last four months of working on my masters project, I had to deal with many issues while also learning a lot. The biggest thing I learned was the procedure of setting up, training and testing a neural network with data I had collected. I have done this stuff before in some classes, but never to the extent needed for this project. As a general rule I learned that you should always use the device that the model is going to run on for capturing the training set. I also learned Swift, a programming language that I never used before, but was always a language I wanted to learn.

One issue that I encounter was choosing setting up an environment to train the models in an efficient way. At first I used darknet to try and train the model with my own custom data on a Macbook Pro. Since the Macbook I was using does not have a GPU, the training took forever even with a small amount of data. I ran the training for over two days and only got through 1000 iterations. I then tried doing the training on a Windows machine with a GPU. It took a long time to get the darknet training model working on an Windows machine, but we were eventually able to figure it out. Getting the training to work on a GPU was the largest roadblock that I had to overcome. Even with a GPU the training still took a very long time.

If I could start the project over again from the beginning the biggest thing that I would change is how much time I needed to spend on gathering quality data. A big reason why my model does not work well with new data is the lack of variation of the training data. All the data that was collected had the same background, at the same time of day, and with the same person just with slightly different clothing. With better data collection the project should run much smoother.

Chapter 6

Conclusion and Future Work

The study of computer vision in competitive sports has proven to be an interesting area. My project provides a framework in which a YOLO model can be trained and used to detect objects on the iPhone. While I ran into many issues I learned a lot. Future work that would be done on this project is to increase the size of the training dataset, focusing on adding greater variation to the data. I would also use the iPhone to gather all the training data, knowing what I know now. Additional work would include continue working on my application that uses OpenCV and look into the effects on the application of a human slightly moving the camera while the it is trying to find all the objects. Lastly, a similar framework for this project could be applied to other sports, for example free throw shots in basketball.

Bibliography

[1] "Volleyball," 2019. [Online]. Available: https://www.britannica.com/sports/volleyball

[2] B. Bright, "Naming the sets..." 2010. [Online]. Available: http://brightvolleyball.club/learntoplay/how-sets-are-numbered.html

[3] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *CoRR*, vol. abs/1807.05511, 2018. [Online]. Available: http://arxiv.org/abs/1807.05511

[4] R. Gandhi, "R-cnn, fast r-cnn, faster r-cnn, yoloobject detection algorithms," Jul 2018. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[5] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[6] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[7] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[8] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[10] A. Maksai, X. Wang, and P. Fua, "What Players do with the Ball: A Physically Constrained Interaction Modeling," Dec. 2015. [Online]. Available: https://cvlab.epfl.ch/research/research-surv/research-balltracking/

[11] M. Takahashi, K. Ikeya, M. Kano, H. Ookubo, and T. Mishina, "Robust volleyball tracking system using multi-view cameras," *ICPR*, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7900050/authors#authors

[12] Y. Yamaguchi and M. Miura, "Real-time analysis of baseball pitching using image processing on smartphone," *Procedia Computer Science*, vol. 96, pp. 1059–1066, 12 2016.

[13] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. [Online]. Available: http://arxiv.org/abs/1612.08242

[14] "VoTT: Visual Object Tagging Tool 1.5," Feb. 2019. [Online]. Available: https://github.com/Microsoft/VoTT

[15] AlexeyAB, "darknet," 2019. [Online]. Available: https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

[16] allanzelener, "Yad2k: Yet another darknet 2 keras," 2017. [Online]. Available: https://github.com/allanzelener/YAD2K

[17] M. Hollemans, "Yolo-coreml-mpsnngraph/tinyyolo-coreml," Dec. 2018. [Online]. Available: https://github.com/hollance/YOLO-CoreML-MPSNNGraph/tree/master/TinyYOLO-CoreML

[18] Y. Ni, "Opencv with swift - step by step," Aug 2017. [Online]. Available: https://medium.com/@yiweini/opencv-with-swift-step-by-step-c3cc1d1ee5f1

Appendices

Appendix A
Installation of Training System

Steps for setting up darknet to run on Windows with Nvidia GPU.

Dependencies:

- CMake
- OpenCV 3.4
- Visual Studio 2017
- CUDA 10.0
- cuDNN 7.4 (optional)

Build:

1. Clone darknet repository from `https://github.com/pjreddie/darknet`, I placed the location of the folder on my Desktop
2. Before building darknet make sure
   (a) OpenCV was built with the CUDA features enabled
   (b) OpenCV was built as x64
   (c) Location of CUDA\v10.0\bin and CUDA\v10.0\libnvvp are added to your path variable
   (d) Create System variable CUDA_PATH with value C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.
3. Open Cmake
   (a) Set "Where is the Source code:" to C:*path to darknet-master*
   (b) Set "Where to build the binaries:" to C:*path to darknet-master*\newbuild
   (c) Create the folder
   (d) Add location of OpenCV build to variable OpenCV_DIR
   (e) Check ENABLE_CUDA
   (f) Check ENABLE_OPENCV
   (g) Configure
   (h) Generate

(i) Open Project

4. Set Solution Configurations to Release
5. Set Solution Platform to x64
6. Right click darknet project and select build
7. Open up the new Release folder inside newbuild
8. Add thes .dll files
   (a) opencv_core340.dll
   (b) opencv_highgui340.dll
   (c) opencv_imgcodecs340.dll
   (d) opencv_imgproc340.dll
   (e) opencv_videoio340.dll
   (f) pthreadVC2.dll

Appendix B
Train and convert models

Steps to train a YOLO model using the darknet framework. Once trained we provide steps on how to convert the model to a CoreML model.

1. Clone my repository from `https://github.com/bfmalsan/Masters-Project`, I placed the location of the folder on my Desktop
2. Set up darknet framework using Appendix A.
3. Move "data" folder to the location of the darknet executable
4. Copy the files from "models/myyolo9000" folder to location of the darknet executable
5. Execute command darknet.exe detector train obj.data myyolo9000.cfg myyolo9000_1000.weights
6. Move "myyolo9000.cfg" and "backup/myyolo9000_final.weights" to the yad2k folder
7. activate python environment
8. Create Keras model. Execute command python yad2k myyolo9000.cfg myyolo9000_final.weights myyolo9000.h5

   Note: For yolov3 use YoloConvert.py
9. Create CoreML model. Execute command python Converter.py myyolo9000.h5 myyolo9000.mlmodel

   Note: May have to change the Python version to 2.7