Automated Jersey Number Recognition: Capstone Project for 494-007

Authors: Noah Wagnon, Jeff Reidy, and Nguyen Nguyen

University of Alabama Department of Electrical and Computer Engineering

***Abstract*—This report details the design specifics and methodologies used for our team's system-based approach to the problem of automated jersey number recognition and indexing in American football. We present our design details, integration procedures, testing infrastructure, and results obtained. We also present a few project management details along with an evaluation of the system's success and ideas for further improvement.**

Keywords—object detection, linked database, optical character recognition, sports footage indexing, data labelling

I. Problem Introduction

A. Background and Problem Statement

As more sophisticated developments in data analysis and storage have arrived, there has been a large increase in the demand for applications of data-intensive analysis to the world of sports. But to meet this demand, however, there needs to be equal improvements in data collection methods over the time-consuming manual methods that are simply impractical for handling the sheer amount of sports footage recorded today. This is especially true when examining past sports footage before recent implementations of player tracking chips. Fortunately, recent advances in advanced image processing techniques have opened the door for many interesting and effective solutions to this problem.

Our project was initiated by our collaborators at the UA School of Library and Information Studies. The specific problem that they sought to solve was the problem of identifying all participating University of Alabama football players on a per-play basis, especially for past UA football games. Their requests have led to our problem statement: we want to develop an automated solution to index each play in a particular game by quarter, game clock start/end time, and include a list of participating UA players within the play. Our solution is designed to work equally as well regardless of what color jersey UA is wearing.

B. Previous Approaches

The main design decision for our team to make was to select an appropriate image processing framework/methodology to perform player jersey number detection. Preliminary research revealed numerous options. The first option we considered was using a single-stage convolutional neural network such as Faster-RCNN [1] to directly detect jersey digits for each frame within a video. The main issue with this approach is the vast amount of noise present in football broadcast videos such as fans, sidelines, motion blur, etc. Trying to perform digit detection in a single stage would be too challenging. We also read reports of pose-estimation-based techniques being successful [2]. This approach, however, would require extensive use of C++ and OpenCv, neither of which our team had extensive experience with. Lastly, we read a report of a team obtaining excellent results in soccer by tracking player position and making jersey number inferences based off positioning, etc [3]. After deliberation, however, we determined that this algorithm could not be easily generalized to American football.

At the conclusion of our background research, we decided to utilize Mask-RCNN [4], a state-of-the-art object detection neural network that is capable of detection and segmentation. To handle the problem of input noise, we decided to split detection into two stages: stage 1 detects persons on the playing field and stage 2 detects the jersey number on an individual person.

C. Subsystem Assignment and Deliverables

As with all systems-based projects, an important step is the partitioning step, which involves separating the problem's solution into subsystems. Our partitioning naturally led to three subsystems, all of which are entirely software-based. The first subsystem is the object detection subsystem. It deals with everything related to the neural network including data collection and labelling, training, evaluation, etc. The main deliverables for this subsystem are the two-stage network mentioned above and an improved football image dataset with labels.

The second subsystem created is the game clock subsystem. This subsystem is responsible for extracting and interpreting the game and play clocks from the broadcast scoreboard within a football game. This extracted data will serve as the index for all our processing, allowing us to determine which players belong to which play. The main

deliverables are a text file containing all clock-related info along with the program to detect and calculate this info.

The final subsystem is the database integration subsystem. This subsystem is responsible for interpreting and manipulating the data from the other subsystems. This system also hosts a simple API that implements a linked database that can accept the coordinated data and store it for future use and persistence. The main deliverable is a functional implementation of all clock-related and participating player information accessible via an API.

II. Object Detection Subsystem

A. Subsystem Introduction

The main goals of the object detection subsystem are to properly recognize the Alabama players' jersey numbers as well as accurately label them, which are done through the two main stages. The first stage is a pre-trained Mask R-CNN that extracts person bounding boxes from footage frames and then generates the person proposals. And the second stage recognizes jersey numbers through training the digit detector. The 0bject detection subsystem mainly utilizes Python3, as well as other tools such as TensorFlow, Keras, VIA, Google Colab, and Mechanical Turk.

B. Analyzing and Adding to the Football Dataset

To begin the labelling process, we needed to capture the frames from footage videos and choose the ones that contain visible Alabama players' jersey numbers. Using Mask-RCNN, we generated person proposals with bounding boxes and set up Mechanical Turk for crowd-sourced labelling. After the workers were finished, we loaded the images into a VIA project to begin the manual labelling process and removed the ones that were not correct. The final dataset was exported into a JSON file in COCO format to prepare for training. Figure 1 below shows an example labelled image.

Figure 1. Example labelled image.

This project and dataset were inherited from another researcher [6], and we have made additions and alterations to it. The figure below shows the final break down and distribution of our dataset. The horizontal axis shows the individual digits, and the vertical axis shows the number of instances within the dataset. These columns are also colored to show the split between home and away jerseys. The total number of digit instances is 28,224. Our dataset originally contained 2401 images. We have identified and labelled 2464 more images from the 2015 Cotton Bowl to add to the dataset. Lastly, we create two copies of each image. One copy is a scaled version, and one copy is a motion blurred version. This brings our final image count to 14595, with 9730 of the images being augmented copies.

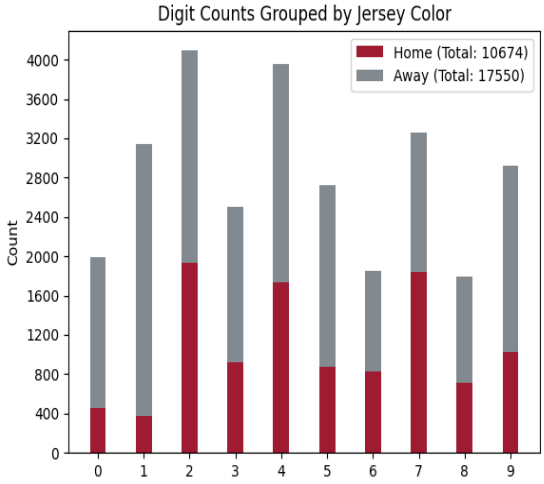


Figure 2. Final state of football dataset.

C. Stage 1: Generate Person Proposals

For stage 1 of the model, we will use transfer learning which involves using a previously trained model for a new task. The pretrained model has been trained on the popular COCO dataset [5] and is designed to identify persons. The dataset contains over 330,000 images and 91 classes, but we are only interested in the person class for this project. The output of this stage will be used as the input for stage 2. Figure 3 below shows the expected input and output for stage 1 of our detection model.

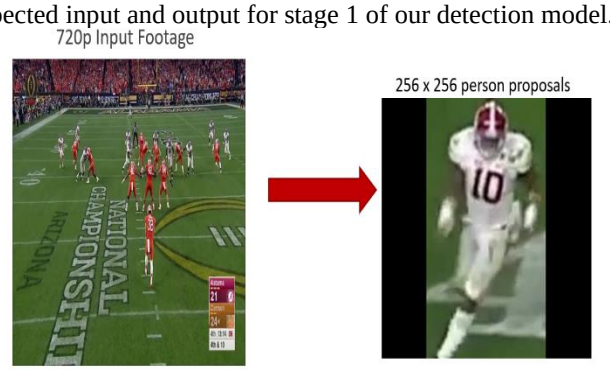


Figure 3. Stage 1 input/output relationship.

D. Intermediate Processing via Color-Filtering

Before passing proposals to the second stage, we must perform an intermediate processing step. This step is responsible for ensuring that our model only performs object recognition on UA players, and it does this by examining the individual RGB channels of each person proposed by the first stage. We know that any image can be split into its 3 color

channels. We can then perform any processing or analysis on the individual channels before merging them to recreate the original. We will be viewing the average intensity value for each channel.

Figure 4 below shows two example person proposals generated by stage 1. We show the full images, but in practice we only view a small strip in the middle to isolate the jersey. The right images are the color histograms of the images where the x axis is the intensity from 0 to 255 and the y axis is the number of occurrences of that intensity. For the crimson player, we can see that on average the red channel intensity value is much higher than blue and green. Examining several other crimson jersey images in a similar manner showed that the average red intensity value is consistently much higher than the green and blue channels. On the other hand, we expect a white jersey to show no dominant color channel. This is how we can differentiate between the two. As expected, the white jersey shows a histogram that is not dominant in any one color and only shows a slight increase in red because the secondary color is red. This analysis is the basis of our color filtering, and we allow the user to provide a command line argument of either "crimson" or "white" to specify jersey color to accept in this stage.

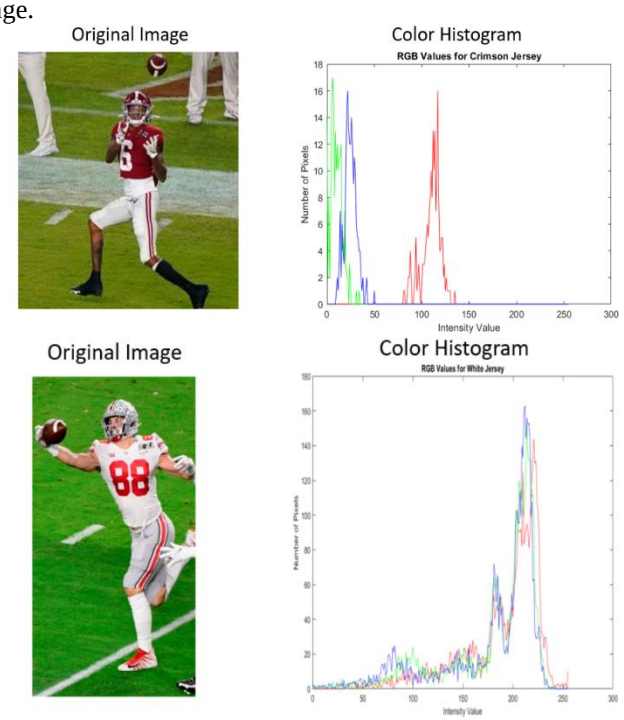


Figure 4. Color Filtering Demonstration.

E. *Stage 2: Digit Detection*

At this point in the recognition process, stage 1 has generated person proposals and the list of proposals has been filtered to ensure UA players only are being examined. So now stage 2 accepts person proposals as input and extracts jersey numbers as output. There is no known pretrained model to accomplish this task, so transfer learning is not an option. Our solution is to train a model on two datasets: the street view house numbers dataset [7] and a football image dataset that we have inherited and will extend. During training we will define our training loss as the sum of the class loss and bounding box loss, where the class loss uses a logarithmic loss formula, and the box loss uses a robust loss formula. Training has been performed on a single RTX 6000 GPU with 24Gb memory and 16.3Tflops for its tensor performance.

Our training methodology was as follows: we trained on the SVHN dataset for 14 epochs and immediately retrained these weights for 15 epochs on the football dataset with train-time augmentations of scaling and motion blur. The training results are shown below in figure 5. Training time was 126 minutes. Our minimum box loss and class loss were 0.0860 and 0.0236, respectively. Figure 6 shows an image of the input and output from stage 2.

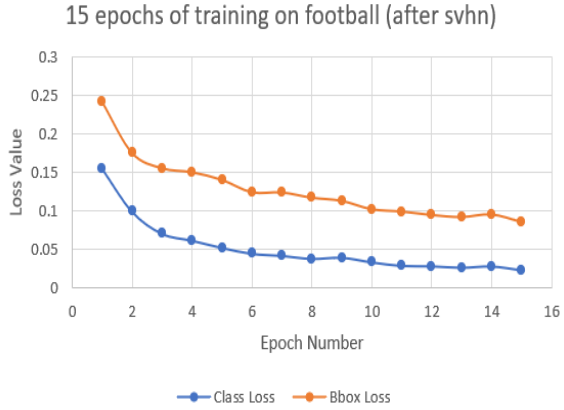


Figure 5. Training results for stage 2.

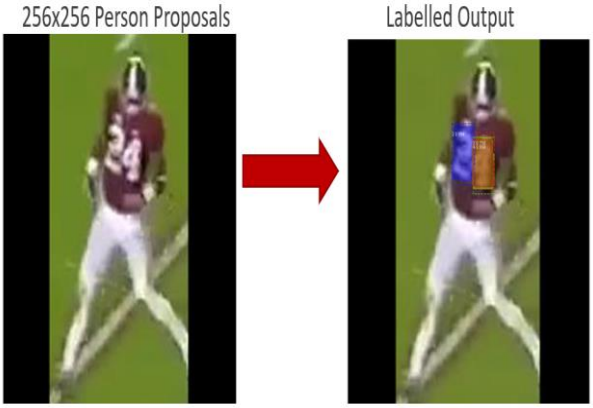


Figure 6. Stage 2 input/output relationship.

F. *Detection Subsystem Testing*

For testing the object detection subsystem, we needed to select a game video in which UA was wearing crimson and a game video in which UA was wearing white. We settled on the 2019 Alabama vs LSU game for our crimson footage [10]. We also settled on the 2019 Alabama vs South Carolina for our white jersey footage [11].

We elected to use abbreviated videos to avoid having 8 plus hours of video footage to process and handle. These videos still include every single play within each game, just with accelerated cuts and skips.

There are several quantitative metrics that we needed to perform verification tests on within the object detection

subsystem. At the beginning of the project, we stated a mAP goal of 0.7. We have surpassed this by achieving a mAP of 0.81. We note that mAP is the accuracy on the training set, so when running on real, new footage the accuracy may vary a bit. Our goal for speed was to reach 2.0 frames per second. Testing the model's speed on crimson jersey footage showed that we were slightly under the requirement, and testing on white jersey footage showed that we were slightly over it. This is not a critical failure, however, since the fps is quite close to the goal. There are methods to improve the fps, but they would be at the cost of detecting less UA players. Testing showed that our color filtering methods have worked very well, since the model now rarely identifies an opposing team player and almost exclusively identifies UA players for both jersey colors. Lastly, we wanted to ensure that our confidence threshold was set at 85% or higher for both Mask-RCNN stages to avoid system noise. Our final parameters used were 93% and 95%. These test results are summarized below in figure 7.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
Accuracy	mAP >= 0.7	Python Evaluation Script	**Pass** - mAP = 0.81
Speed in Crimson	Fps >= 2.0	#Frames Processed/Time	**Fail** - FPS = 1.89
Speed in White	Fps >= 2.0	#Frames Processed/Time	**Pass** - FPS = 2.03
Distraction-Proof	>= 80% of ids are UA	Manual counting on 2-min test footage	**Pass** - more than 90% of ids are UA players with both colors tested.
Detection Confidence	Confidence Threshold is >= 85% for both stages	Examine code of final system implementation	**Pass** – Stage 1 C.T. = 95% Stage 2 C.T. = 93%

Figure 7. Object detection verification results.

After verification, we wanted to validate our detection subsystem to make sure it meets the overall needs of the project. First is a subjective evaluation of the dataset's final state. We have determined this test as a pass since we have increased the overall size of the dataset by a factor of 5 compared to its original state. This was done through augmentation and MTurk labelling as discussed earlier. A larger, more robust dataset will allow for more valuable training and will allow for greater use by others looking for a football dataset in the future. We also wanted to make sure the subsystem is flexible and works nearly the same regardless of UA's jersey color. As we saw in the last paragraph, the speed is nearly the same in both cases and digit detection accuracy is nearly identical, making the test for flexibility a pass. Lastly, we wanted the detection subsystem to be modular, meaning it can operate and be modified independently without having to alter or execute the rest of the system software. This has been achieved by separating the subsystem's code and scripts, allowing us to run detection without necessarily having to run the game clock and database subsystems at the same time. This has helped testing and development. These validation results are summarized below in figure 8.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
Dataset Robustness	Difficulty, Quantity, and Flexibility	Subjective Evaluation of dataset	**Pass** – Augmentation has expanded dataset by a factor of 3. We also have 2604 additional images with labelling nearly complete.
Flexibility	Works on Crimson and White	Compare accuracy with 1-min test footage of each color	**Pass** – Digit detection accuracy is not affected by color. Verification test above showed that 90% or more of players identified are UA players in both colors.
Modularity	Subsystem can be run independent of other subsystems and results are saved.	Python test script	**Pass** – subsystem can be run independently with no input data from other subsystems.

Figure 8. Object detection validation results.

III. Game Clock Subsystem

A. *Subsystem Introduction*

The first main responsibility of the game clock subsystem is to extract the game and play clock data from the game video provided. The second main responsibility of the game clock subsystem is to filter through the game and play clock data with a second filter program to derive individual play windows for each play's start and end times. These play windows feature the absolute range of frames for each play which is used for synchronization with the object detection subsystem to find players present in each play.

B. *Tesseract Optical Character Recognition Introduction*

The main piece of software used for the game clock subsystem was the Tesseract Optical Character Recognition software. The purpose of the Tesseract OCR was to derive the characters present in each image. The stages of running an image through the Tesseract OCR are as follows, first the image must be preprocessed, next the image will go through the OCR for character segmentation and recognition, and finally post processing of the outputted data. The main challenge with using Tesseract OCR in this project was modifying the image to get it to a state where Tesseract OCR could easily recognize the characters present.

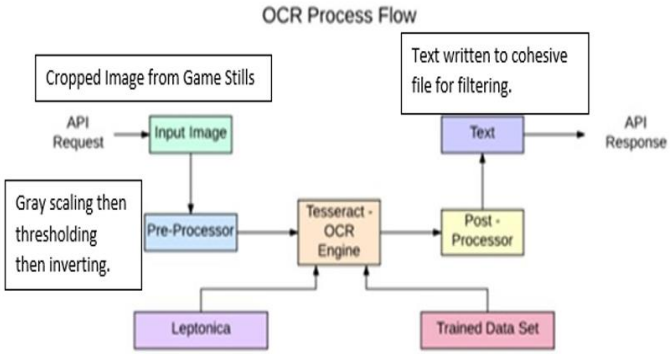


Figure 9. Tesseract OCR Process Flow.

C. *Stage 1: Detection of Game and Play Clock Information*

For stage 1 of this subsystem the game footage desired to be analyzed must be entered into the program via command line argument. Once this happens, the program begins by first breaking the video down into individual frames, specifically taking every 30th frame to achieve an optimal frame rate of 1 fps. From there the individual frames are cropped to the specific region of interest for the frame, specifically the small region surrounding the game and play clock information. This cropped image is then altered three different ways to reach desired OCR detection format. First the image is gray scaled to enhance contrast between the text and the background, next the binary thresholding is applied to make the image black and white, and finally the image is color inverted to reach the desired OCR format of black text on a white background. Once all this has happened the image is then run through the Tesseract OCR software for character detection, and then that output is written to a text file along with the absolute frame information for the filter program to run on for play window detection.

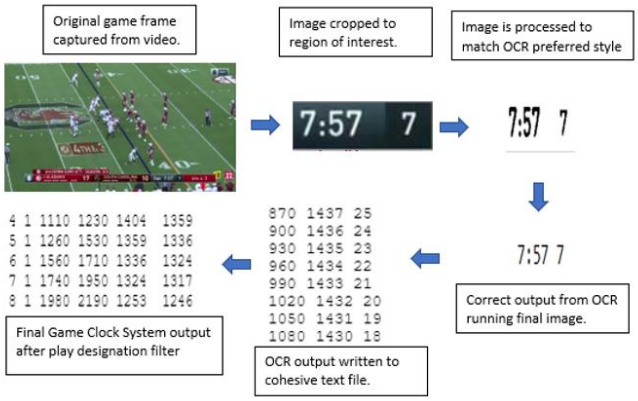


Figure 10. OCR Detection Process Flow.

The information written to the text file from left to right is the absolute frame number, game clock information, and then play clock information. The absolute frame number allows for easy synchronization with object detection subsystem results.

D. *Stage 2: Play Window Detection*

At this point in the play recognition process the Tesseract-OCR combined with the program written have outputted a file with all the game and play clock information for the given video. Stage 2 is to take this file and iterate through it determining when plays start and end during the video. To do this a play designation filter program was designed to sift through the data and output play windows that contain the range of frames for each play, play start time, and play end time. As shown in the figure below, the output of the play designation filter from left to right is play number, quarter number, frame of play start, frame of play end, play start time, and play end time.

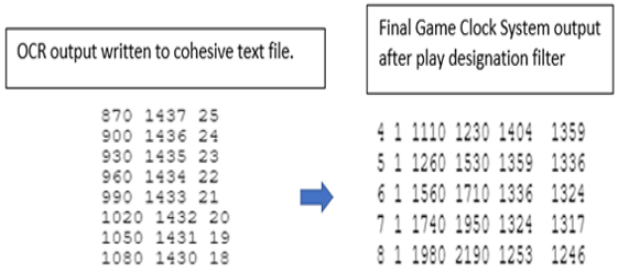


Figure 11. Play Designation Filter Output

The purpose of the range of plays is for synchronization with the results from the object detection subsystem. Determining the start and end times of plays was difficult due to the choice of video chosen for our project. Because of distractions caused by graphics and other media for the object detection subsystem, full length footage of the games could not be utilized. Instead, accelerated video footage was chosen that quickly jumps from the end of one play to the start of the next. This eliminates any standard for play start and end times. In full game footage the play clock reset can be used to track the end and subsequent start of each play. Although there is no standard for play end and start in accelerated footage, by examining the footage we determined that watching for jumps in the play clock could be an efficient method for identifying many of the plays. One shortcoming of this such method is that it excludes plays such as incomplete passes, touchdowns, extra points, and kickoffs. When these plays take place, multiple plays are often grouped together in a single play window. This is not a critical issue as our main goal is to identify players on the field during active plays and our video choice limits itself to only times where plays are actively happening. Finally, this data is outputted to the database subsystem for integration with object detection.

E. *Game Clock Subsystem Testing*

For testing the game clock detection subsystem, we used the same two videos used for testing detection. For verification of the game clock subsystem, we decided that the subsystem must run faster than the object detection subsystem which it easily does as tested in the lab. The subsystem must also be distraction proof which is easily achieved because of the accelerated game footage we use which limits on screen graphics, and python3 code written to give designated output of “0 0” for game and play clock information in which output does not match our formats. The 3rd verification standard is a 90% OCR detection accuracy which was verified as passed by manually checking output against the game stills in which we found an accuracy of 95.6%. The next verification standard is Quarter number accuracy which was checked manually by examining the output of the play window filter program against the youtube game videos. The final verification standard is the play window start and end time accuracy. For our subsystem, a play window is defined as any window in which only a single play occurs. This is normally from the

time the last play ends to the end of the current play. This subsystem failed our verification standards because of errors caused mainly by the choice of video used. We chose to use accelerated game footage to reduce the run time of our system overall and help the object detection subsystem with accuracy of player detection. The formatting of this game footage makes filtering for play windows much more difficult because of the lack of a standardized format for start and end of plays. We are okay with this failure as it allows for greater accuracy of player detection in our database which is considered the more important subsystem of this project. Those test results are summarized in the figure below.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
Speed	OCR execution time <= Detection execution time	Run each subsystem simultaneously on same video	**Pass** - OCR time = 7:45 -OD time = 41:00
Distraction-Proof	OCR only processes the game clock and play clock	Python testing and manually examining output	**Pass** - No extra on-screen graphics are processed
Digit Detection Accuracy	Numbers are read correctly 90% or higher of the time.	Examine and compare raw frames passed to OCR	**Pass** – OCR accuracy was found to be 95.6% accurate (2704/2826)
Quarter # Accuracy	Quarter transitions are handled successfully and the quarter # is always 1-4	Python testing and manually examining output	**Pass** – quarter # increments when clock goes from 0:00 to 15:00 and quarter # is always 1-4
Start/end time Accuracy	90% of play windows begin before the ball is snapped and end before the next snap of the ball.	Python testing and manually examining output	**Fail**- Play designation filter successfully derived 62% of the total plays correctly

Figure 12. Game Clock Verification Results.

After verification, we wanted to validate our detection subsystem to make sure it meets the overall needs of the project. For the validation of the game clock subsystem, we decided the program must always output in a specific format never omitting an information, this was tested and passed manually by examining the play window designation filter output. Secondly the program must be able to run independent of the other subsystems, this was easily passed since originally it was made to run independently before integration. Finally, it needs the ability to be generalized to other broadcasts. This was passed by running the program on game footage from different broadcasting agencies. The only change needed is updating of the cropping window to crop only the region of interest. The results of this testing are shown in the figure below.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
Output Format	Program calculates and displays the following for each play: - Play # - Quarter # - Absolute Frame Reference - Start/end time	Execute on testing footage and manually examine output.	**Pass** – output includes these fields for every play
Modularity	Subsystem can be run independent of other subsystems and results are saved.	Python test script	**Pass** – subsystem can be run independently with no input data from other subsystems.
Generalizable to other broadcasts	Program can easily be run on broadcast footage that formats game clock/play clock differently	Run on separate videos and evaluate how easy it is to transition to different format.	**Pass** – code requires very little change to account for broadcast differences.

Figure 13. Game Clock Validation Results.

IV. Database Subsystem

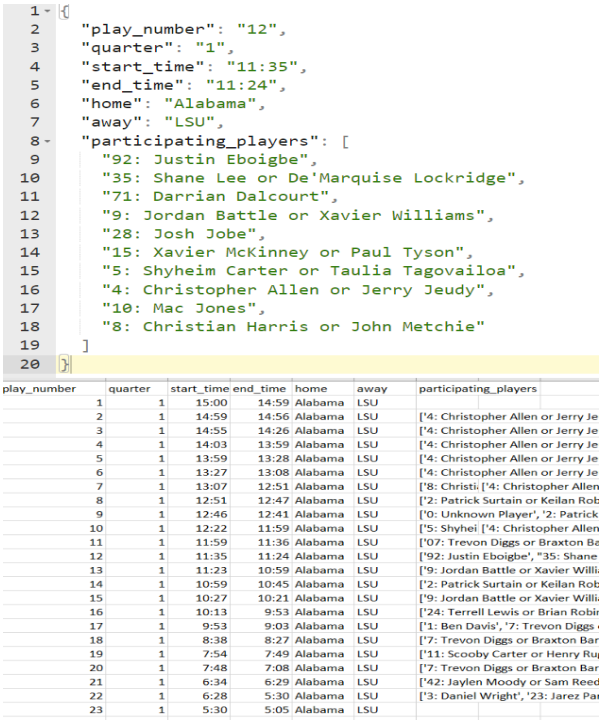
A. *Subsystem Introduction*

The main responsibilities of the database subsystem are to synchronize the results from the game clock and object detection subsystems and serialize these results into a human-readable format. This will result in an API interface that our collaborators can use to interact with the data. Previous projects worked on by our collaborators have produced similar interfaces, but all have lacked important participating player information that our project includes. All serialized data will be in the JSON format.

B. *Swagger UI to Generate Mockups and Server Stubs*

We selected Swagger UI as the tool to generate documentation and initial code for our simple API for accessing the system results. We use the HTTPS protocol to define simple GET and POST commands for uploading and fetching play information. Swagger UI allows users to generate design documents to guide future users. It also offers the option to automatically generate code stubs in a selected language, which is in our case the python programming language.

The code stubs generated are built to utilize Flask, which is a simple python library designed for developing API interfaces that can run on local machines or on web servers. Our project requires us to link a database to this API. Since we are testing with a relatively small amount of data, we are using simple csv files for mimicking the behavior of a relational database. If the number of project users were to grow by a large number, then we would need to add on a true relational database such as MySQL. Figure 14 below shows an example JSON structure generated during testing along with an example of a JSON body after being posted to the .csv database.



```
{
  "play_number": "12",
  "quarter": "1",
  "start_time": "11:35",
  "end_time": "11:24",
  "home": "Alabama",
  "away": "LSU",
  "participating_players": [
    "92: Justin Eboigbe",
    "35: Shane Lee or De'Marquise Lockridge",
    "71: Darrian Dalcourt",
    "9: Jordan Battle or Xavier Williams",
    "28: Josh Jobe",
    "15: Xavier McKinney or Paul Tyson",
    "5: Shyheim Carter or Taulia Tagovailoa",
    "4: Christopher Allen or Jerry Jeudy",
    "10: Mac Jones",
    "8: Christian Harris or John Metchie"
  ]
}
```

play_number	quarter	start_time	end_time	home	away	participating_players
1	1	15:00	14:59	Alabama	LSU	
2	1	14:59	14:56	Alabama	LSU	['4: Christopher Allen or Jerry Je
3	1	14:55	14:26	Alabama	LSU	['4: Christopher Allen or Jerry Je
4	1	14:03	13:59	Alabama	LSU	['4: Christopher Allen or Jerry Je
5	1	13:59	13:28	Alabama	LSU	['4: Christopher Allen or Jerry Je
6	1	13:27	13:08	Alabama	LSU	['4: Christopher Allen or Jerry Je
7	1	13:07	12:51	Alabama	LSU	['8: Christi ['4: Christopher Allen
8	1	12:51	12:47	Alabama	LSU	['2: Patrick Surtain or Keilan Rob
9	1	12:46	12:41	Alabama	LSU	['0: Unknown Player', '2: Patrick
10	1	12:22	11:59	Alabama	LSU	['5: Shyhei ['4: Christopher Allen
11	1	11:59	11:36	Alabama	LSU	['07: Trevon Diggs or Braxton Ba
12	1	11:35	11:24	Alabama	LSU	['92: Justin Eboigbe', '35: Shane
13	1	11:23	10:59	Alabama	LSU	['9: Jordan Battle or Xavier Willi
14	1	10:59	10:45	Alabama	LSU	['2: Patrick Surtain or Keilan Rob
15	1	10:27	10:21	Alabama	LSU	['9: Jordan Battle or Xavier Willi
16	1	10:13	9:53	Alabama	LSU	['24: Terrell Lewis or Brian Robi
17	1	9:53	9:03	Alabama	LSU	['1: Ben Davis', '7: Trevon Diggs
18	1	8:38	8:27	Alabama	LSU	['7: Trevon Diggs or Braxton Bar
19	1	7:54	7:49	Alabama	LSU	['11: Scooby Carter or Henry Ru
20	1	7:48	7:08	Alabama	LSU	['7: Trevon Diggs or Braxton Bar
21	1	6:34	6:29	Alabama	LSU	['42: Jaylen Moody or Sam Reed
22	1	6:28	5:30	Alabama	LSU	['3: Daniel Wright', '23: Jarez Pa
23	1	5:30	5:05	Alabama	LSU	

Figure 14. Example JSON response and database status after posting.

C. *Databse Subsystem Testing*

We used Postman as the primary tool for testing the database subsystem. Postman is a very simple tool that allows the users to send and receive test API requests and responses. It provides response times and response codes through a user interface. The database subsystem is our most lightweight piece of the project. This combined with the use of Postman testing software makes it by far the easiest subsystem to test. We wanted both get and post methods to execute in under 1 second and to return a response code of 200 O.K. We found the average times to be 17 and 15 ms and the response code to be 200 in nearly all test cases with Postman, which passes our verification tests shown below in figure 15.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
POST Speed	Response time <= 1sec	Postman	**Pass** - R.T. = 17ms
Get Speed	Response time <= 1sec	Postman	**Pass** - R.T. = 15ms
Post Correctness	Response code = 200	Postman	**Pass** - R.C. = 200 OK
Get Correctness	Response code = 200	Postman	**Pass** - R.C. = 200 OK

Figure 15. Database verification results.

With the database meeting quantitative metrics, we wanted to define validation tests that measure its effectiveness at a higher-level and measure how well the subsystem contributes to the problem solution. For measuring accessibility, we decided that the results of any post call should automatically and immediately affect the linked database with no manual action. On the flip side, we determined that the GET command should allow users to access and use the data in real-time in their own external software. Each of these tests passed using Postman and python test scripts. Furthermore, the posted data persists in the database after the program is ended, satisfying our test for persistence. Our only subsystem failure is the web access attribute. Our current implementation uses a local http server each time to host the API. With our time restraints, we have not yet gone through the process of officially registering the server for web access, but it should be straightforward, and the database behavior will remain unchanged. Below the validation results are shown in figure 16.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
POST Accessibility	Posted data appears in database	Postman	**Pass** – data appears immediately
GET Accessibility	Data can be retrieved by 3rd parties	Python test script	**Pass** – one line of code imports the data as a python dictionary
Readability	Swagger UI docs define functionality	Subjective evaluation of docs organization	**Pass** – API docs neatly and clearly define expected parameters and behaviors of all methods.
Data Persistence	Data persists after program execution	Python test script	**Pass** – data remains permanently unless manually deleted
Web Access	Database operates as a web server	Subjective evaluation	**Fail** – current implementation requires database to be run as a local http server. Web server registration has not been implemented yet.

Figure 16. Database validation results.

V. System Integration and Testing

A. *Integration Plan and Results*

With all our project components being software pieces, integration is a much less challenging task than managing hardware and software interfaces. The detection and game clock systems operate independently and in parallel and do not require integration. The database subsystem is integrated with each of the other subsystems in a manner that involves manipulating and interpreting subsystem output to match the final expected output of the system.

To test these two integration points, we combined the individual subsystem software into one larger system and executed on real test footage. The python script coordinate.py is responsible for accepting input from both the detection and game clock subsystems and repackaging them into a format fit for the database subsystem. In both cases we were able to confirm that the output from coordinate.py matched the expected results, meaning integration was successful. These results are shown below in figure 17.

Integration	Criteria for Acceptance	Testing Method	Result
Object Detection and Game Clock Subsystems	N/A	N/A	**N/A** – Detection and Game clock subsystems operate independently
Object Detection and Database Subsystems	Database software accepts OD results as input and processes them correctly.	Execute API/coordinate.py on real OD results and examine output	**Pass** – Database maintains correct list of players identified per frame.
Game Clock and Database Subsystems.	Database software accepts GC results as input and processes them correctly.	Execute API/coordinate.py on real GC results and examine output	**Pass** – Database maintains correct list of indexed plays.

Figure 17. Integration results.

With integration complete, we created the chart in figure 18 to visualize the system as a whole execution flow from start to finish. We see that the object detection subsystem and game clock subsystem operate in parallel on the same input footage. The game clock software runs on the lab CPU since it is not intensive enough to require a GPU. This allows both subsystems to run concurrently. Each of their outputs is passed to coordinate.py, which reads the subsystems' output and packages it into the form expected by our API and database subsystem. The api.py python script finally posts the results to our linked database.

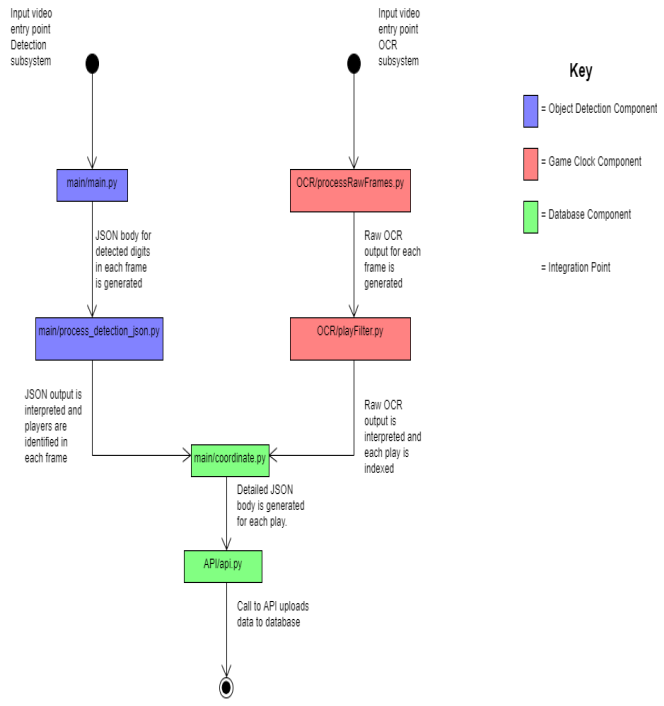


Figure 18. System execution flow.

B. *System-as-a-whole Testing Procedure and Results*

To test the system-as-a-whole, we simply ran detection on the two videos mentioned above (LSU and South Carolina 2019) while simultaneously running the game clock software on the same input footage. We followed the execution steps shown in figure 18 exactly. To verify the system, we decided that the system must operate in the correct order which was tested using UA footage with crimson and white jerseys during which the system worked as previously described. Secondly the system readability is clear and organized, which was evaluated by us a team and enhanced by adding extensive comments in our code. 3rd is portability which failed as switching machines for this program requires significant change, and speed is highly variable depending on GPU strength. Finally, the system must require no manual labeling, this was tested and passed by running the system using only command line arguments. The summary of these tests is shown in the figure below.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
Execution Flow	System executes correctly in order described	Test footage (Crimson and White)	**Pass** – execution flow works exactly as described in integration diagram.
Readability	Software is clearly organized and commented	Subjective evaluation of code structure.	**Pass** – system software uses version control, is well commented, and reasonably organized
Portability	System can be easily transferred to other machines	Attempt to implement on other OS	**Fail** – system requires significant overhead to change OS. Speed performance varies greatly with GPU strength.
Automation	No manual label required	Execute Program and only provide CLAs	**Pass** – system requires no manual labelling.

Figure 19. System Level Verification Results

To validate our overall system, we first tested whether it can be generalized to other teams, to test this we attempted to ID players wearing blue jerseys which the system handled well identifying over 90% of Florida players. Next, we had to make sure our system was faster than manual labeling, we tested this by attempting to manually label players ourselves and we found the system was multiple minutes faster per play. The last two validation criteria were ease of use and overall functionality which was tested by running the system on two full game and examining the results. The summary of the results of this testing is shown in the figure below.

Attribute Tested	Criteria for Acceptance	Testing Method	Result
Generalizable to other teams	System would require little effort to implement with non-UA team	Test for 1 minute trying to id blue jerseys.	**Pass** – System performed equally as well exclusively identifying blue UF jerseys. Over 90% of ids were UF.
Improvement over manual methods	Solution takes substantially less time and effort than manual method.	Attempt to manually id all participating players and compare.	**Pass** – manual method takes several minutes per play longer with much more effort.
Ease of Use	User requires little technical knowledge to use system and access results.	Subjective evaluation of documentation/instructions.	**Pass –** running the system only requires simple command line usage such as "python api.py"
Overall Functionality	Input and output matches what our collaborators expected	Test on 2 full games and examine results.	**Pass** – for both games, input was .mp4 video and output was database containing participating player data

Figure 20. System-level validation results.

C. *Challenges Faced and Evaluation of Results*

As evidenced by our numerous successful test results for each subsystem and for the system-as-a-whole, our project was overall a success. We ultimately achieved what we set out to do and did so in an efficient and timely manner. We have created an end-to-end system that uses advanced image processing techniques to automatically collect participating player info along with relevant clock information. We have included functionality to combine the results to form an API interface that can be interacted with to post and pull stored data.

While our overall evaluation of the project is favorable, we still faced minor issues for each subsystem. For starters, we will examine the game clock subsystem. The primary issue with the game clock subsystem is the vulnerability to non-live football footage. This includes replays, ads, commercials, special camera angles, etc. All these cases often confuse the OCR software and our play calculating logic can often make subtle mistakes here. We consider this issue to be a result of the footage selected, however. A possible solution to this problem is to manually cut and edit videos to remove all footage that is not live footage of a play. This method would be time consuming, however, and we have not yet had a chance to attempt it.

For the database subsystem, the main shortcomings were time related, as we did not get to implement basic delete and patch commands. We also did not get to register our server officially on the web. Aside from these time related issues, the database subsystem was very successful overall and presented the fewest problems of the three subsystems.

For the object detection subsystem, the biggest challenge faced was related to stage 1: person proposals. An inherent problem with detecting persons in American football is the cluster of 10 players (5 offense and 5 defense) on the line of scrimmage. Common R-CNNs struggle to successfully differentiate and segment these clusters of players and often detect less than 2 out of 10. We have done research into modern deep learning solutions to this problem and will present a potential methodology to try in the next section.

VI. Future Work

A potential solution to the problem of detecting and segmenting players in busy, clustered areas such as the line of scrimmage is to retrain our first stage of the object detection model using the relatively new Detection Transformer (DETR) network [9]. DETR combines a basic CNN such as Mask-RCNN in tandem with a transformer encoder/decoder stage. The results are a model that gives comparable speed to top-end RCNNs while outperforming state-of-the-art models on challenging images with overlaid objects. These results, if carried our project, could boost the object detection subsystem performance tremendously. Figure 21 below shows another student's attempt at trying DETR to detect players in a busy frame [8]. The results below are using a model pretrained for person detection. We would like to gather and label football player images to retrain the model on so that it becomes more robust for detecting football players and not just persons in general. We can clearly see that this would be a valuable addition to the project.

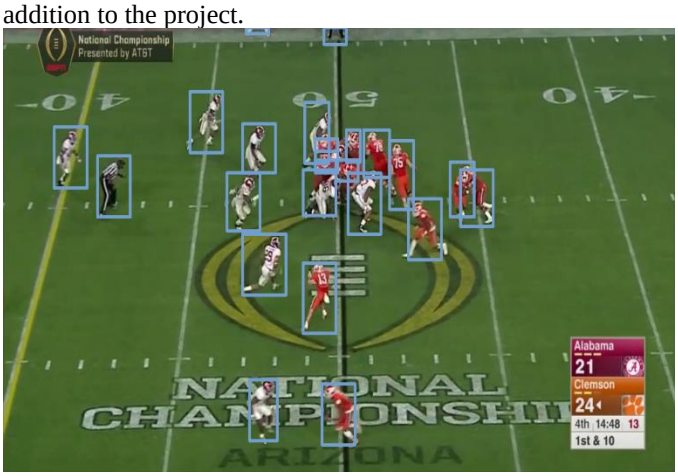


Figure 21. Using pretrained DETR on football players.

VII. Conclusion

A. *Project Budget and Schedule*

For administrative purposes, we have included a figure displaying the final state of the project's budget and schedule from inception to completion.

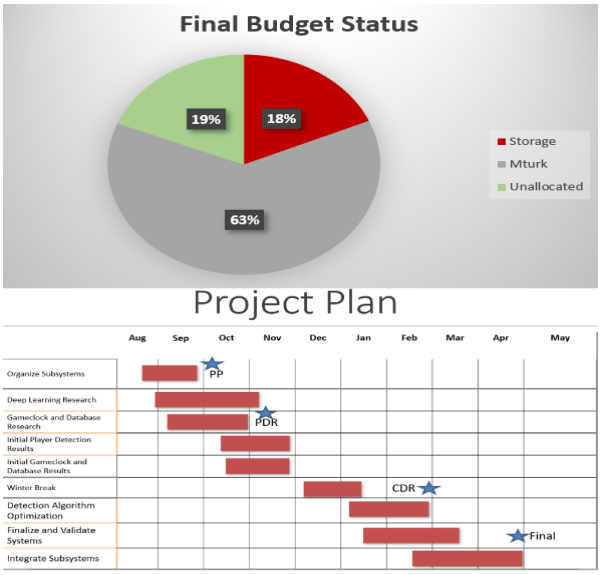


Figure 22. Project budget and schedule.

We spent a total of $187.78 on MTurk labelling fees and $55.49 on external storage, leaving $56.69 of unallocated budget space. In terms of following the schedule, we only had a slight delay in game clock subsystem development due to the learning curve of learning the OCR and pytesseract software. This delay was made up for, however, during late January and early February.

B. Instructions and Access to Software Created

Throughout the project's development, we have used git for software version control. We have stored the project's source code publicly on Github [12]. This Github repository also includes a "494-407 Project Demo.pdf" file. This file serves as a step-by-step instruction/guide for installing and executing our software. The only requirements are a computer with python3, pip3, and anaconda installed.

C. Impacts of Project and Final Thoughts.

Our collaborators approached this project with the primary goal of assisting the Paul Bear Bryant museum store and process footage of UA football games and access the footage based on specific participating players. This would benefit historians and family members of former players collect historic footage. This is not the only impact of the project, however, since our project can provide excellent benefits to sports data collection agencies such as ESPN and iSports. These companies will certainly be interested in automated data collection and indexing methodologies, and they almost certainly possess the processing power to execute a system such as this. As discussed earlier, our project certainly would require a few more improvements before being deployed at a commercial level, but we have proposed an excellent starting point for a solution to the problem of automated jersey number recognition of UA players.

REFERENCES

[1] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[2] H. Liu and B. Bhanu, "Pose-Guided R-CNN for Jersey Number Recognition in Sports," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 2457-2466, doi: 10.1109/CVPRW.2019.00301

[3] S. Gerke, A. Linnemann, and K. Muller. Soccer player recognition using spatial constellation features and jersey number recognition. Computer Vision and Image Understanding, 159:105–115, 2017.

[4] K. He, G. Gkioxari, P. Dollár and R. B. Girshick, "Mask R-CNN", *CoRR*, 2017.

[5] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L. & Dollár, P. (2014). Microsoft COCO: Common Objects in Context.

[6] A. Ramey. American Football Player Jersey Number Recognition. UA Computer Science Department, 2020.

[7] Stephen, Okeke & Jang, Young & Yun, Tae & Sain, Mangal. (2019). Depth-Wise Based Convolutional Neural Network for Street Imagery Digit Number Classification. 133-137. 10.1109/CSE/EUC.2019.00034.

[8] C. Adreon. Detection Transformer on Football Images. UA ECE Department, 2021.

[9] Carion, Nicolas & Massa, Francisco & Synnaeve, Gabriel & Usunier, Nicolas & Kirillov, Alexander & Zagoruyko, Sergey. (2020). End-to-End Object Detection with Transformers.

[10] Test video 1: https://www.youtube.com/watch?v=BRchZ0OggPM&ab_channel=rtsportsuploads

[11] Test video 2: https://www.youtube.com/watch?v=vyQG3WAgUDQ&t=257s&ab_channel=PCAlabamaEnthusiast

[12] Project Source Code: https://github.com/ndwagnon/Automated_Jersey_Number_Recognition