NFL 1st and Future Impact Detection

December 2020

Christian Bile (ezanin-christian-prince-carlos.bile@polytechnique.edu)
Marco Garcia (marco.garcia-macias@polytechnique.edu)
Andrei Plekhanov (andrei.plekhanov@polytechnique.edu)

1 Introduction

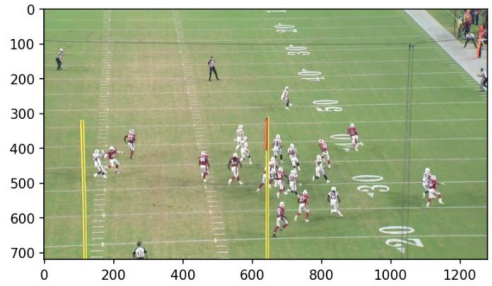
For this project we decided to participate in a computer vision Kaggle challange, the National Football League (NFL) along with Amazon Web Services (AWS) are trying to develop the "Digital Athlete," a virtual representation of a composite NFL player that the NFL can use to model game scenarios to try to better predict and prevent player injuries. For this, The NFL is actively addressing the need for a computer vision system to detect on-field helmet impacts as part of the "Digital Athlete" platform.
The objective is to develop a computer vision model that automatically detects helmet impacts that occur on the field by using labeled video from the sidelines and end zones, and player tracking data. This information is sourced from the NFL's Next Gen Stats (NGS) system, which documents the position, speed, acceleration, and orientation for every player on the field during NFL games.

The National Football League is America's most popular sports league. Founded in 1920, the NFL developed the model for the successful modern sports league and is committed to advancing progress in the diagnosis, prevention, and treatment of sports-related injuries. Health and safety efforts include support for independent medical research and engineering advancements as well as a commitment to work to better protect players and make the game safer.

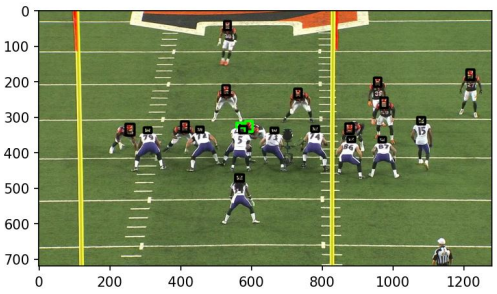
2 The Data

The dataset is divided in three parts:

- **Video files:** 120 .MP4 files with the following format GameKey_PlayID_View.mp4, Each Play has two copies, one shot from the View endzone and the other shot from the View sideline. The video pairs are matched frame for frame in time, but different players may be visible in each view. We only need to make predictions for the view that a player is actually visible in. The training set videos have their corresponding labels in the train_labels.csv file.
- **Train labels:** Helmet tracking and collision labels for the training set.
  - **GameKey:** the ID code for the game.



(a) Video frame example with no labels in image



(b) labeled video frame

Figure 1: Example of data before and after labeling with the information from the csv files

  - **PlayID:**the ID code for the play.
  - **View:** the camera orientation.
  - **Video:** the filename of the associated video.
  - **Frame:** the frame number for this play.
  - **label:** the associate player's number.
  - **left/width/top/height:** the specification of the bounding box of the prediction.
  - **impact:** an indicator (1 = helmet impact) for bounding boxes associated with helmet impacts
  - **impactType:** a description of the type of helmet impact: helmet, shoulder, body, ground, etc.
  - **confidence:** 1 = Possible, 2 = Definitive, 3 = Definitive and Obvious
  - **visibility:** 0 = Not Visible from View, 1 = Minimum, 2 = Visible, 3 = Clearly Visible

- **Sensor data: Each player wears a sensor that allows us to precisely locate them on the field.**
  - **GameKey:** the ID code for the game.
  - **PlayID:**the ID code for the play.
  - **player:** the player's ID code.
  - **time:** timestamp at 10 Hz.
  - **x:** player position along the long axis of the field.
  - **y:** player position along the short axis of the field.
  - **s:** speed in yards/second.
  - **a:** acceleration in yards/second$\hat{2}$.
  - **dis:** distance traveled from prior time point, in yards.
  - **o:** orientation of player (deg).
  - **dir:** angle of player motion (deg).
  - **event:** game events like a snap, whistle, etc.

2.1 Data Manipulation

The data by itself and the way it is given is not very useful, so we had to do some manipulations in order to use it.

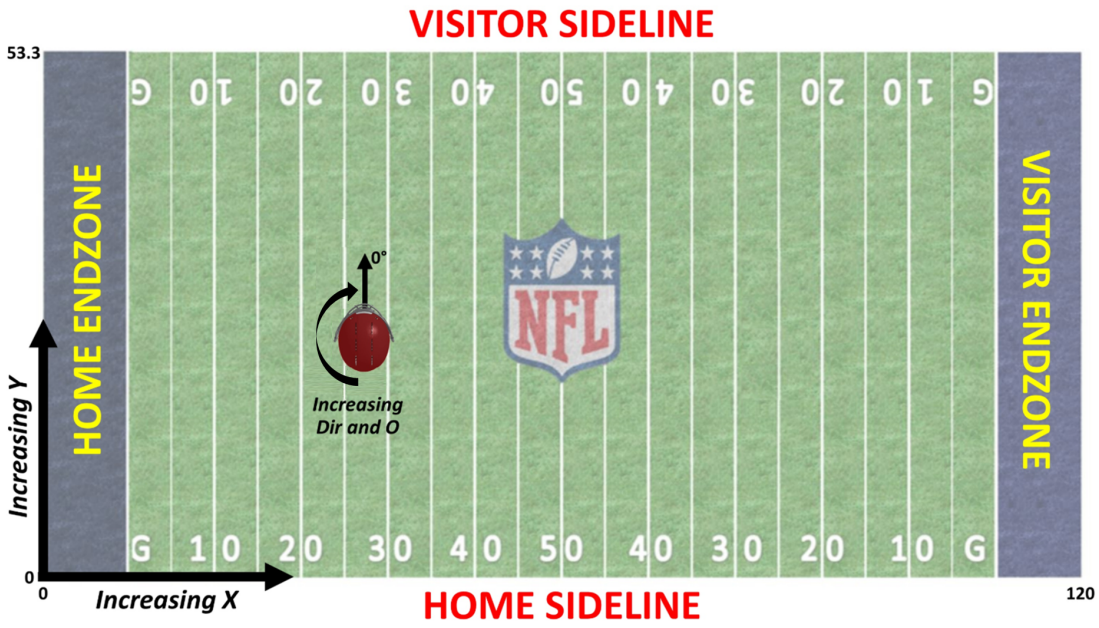


Figure 2: Field example to understand the data from sensors

2.1.1 Matching sensor data with Video data

As mentioned in the data description, the video data has a rate of 60 fps however the sensor in the helmets sample at 10Hz so we have missing information from our sensor data in comparison with the data from the videos. However we have an initial reference that can help us connect the sensor data with the video data, the videos begin 10 frames before the "snap" action and the tracking data contains an "event" column in which the "ball_snap" is recorded. As well The Sideline and Endzone views have been time-synced such that the snap occurs 10 frames into the video. This time alignment should be considered to be accurate to within +/- 3 frames or +-0.05 seconds (60fps) .
Let t be the timestamp at frequency $F_s = 10Hz$ and t_0 be the time at the snap, in which the information of the video and the information of the sensor $S(t)$ are sincronized Let f be the frames of the video with frequency F_v, f_0 be the frame of the "snap" which we know by the given information that $f_0 = 10$.
Then the frame corresponding to the time t is:

$$f(t) = f_0 + (t - t_0)(F_v)$$

$$t(f) = (\frac{1}{F_v})(f - f_0) + t_0$$

Once having this alignment with the sensor data we can now have a direct connection between speed, acceleration, position with the impact and bounding boxes information at least in some frames of the video. If we had more time for the project it would be interesting to use time series to guess the missing information between the frames using video data.
With this information it is possible to do a 2D version of the game, to build the field we have to map it correctly, the rectangular field of play used for American football games measures 100 yards (91.44 m) long between the goal lines, and 160 feet (48.8 m) which is 53 1/3 yards wide. In addition, there are end zones extending another 10 yards (9.144 m) past the goal lines to the "end lines", for a total length of 120 yards (109.7 m).

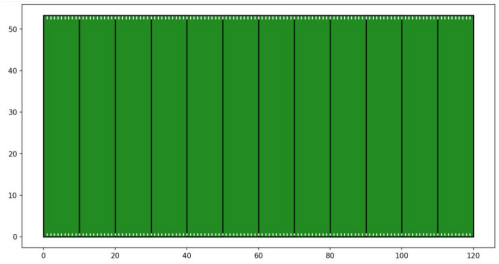
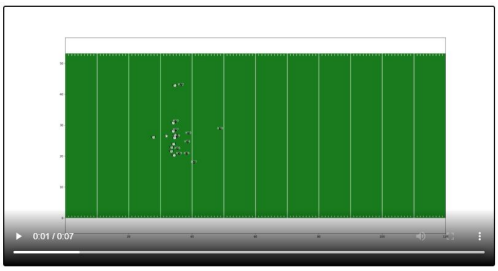
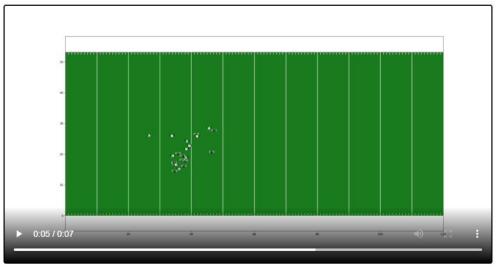


Figure 3: Empty field build for the sensor-video mapping



(a) Virtual reconstruction at second 1



(b) virtual reconstruction at second 2

Figure 4: Evolution of labeled players in virtual field

3 The Metrics

The proposed metric is to evaluate the F1 score at an intersection over union threshold of 0.35, The IoU of a proposed bounding box and a ground truth bounding box is calculated as:

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

The F1 score is:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where the precision

$$precision = \frac{TP}{TP + FP}$$

and the recall

$$recall = \frac{TP}{TP + FN}$$

4 Challanges

- Large images (720x1280) but very small and concentrated bounding boxes ($\approx 0.02 normalizedwidthandheight) Smallam$
- We need to deal with two different camera views
- Lack of computational resources

5 Object detection

For our object detection task we decided to use YOLOv4 model as it is state of the art model for object detection.

6 YOLOv4

YOLOv4 is real-time one stage object detection model which could be trained on single conventional GPU. This model consists of the **backbone** component used for the feature extraction as well as **head** component used for loss calculation and prediction. We will briefly go through explaining each of the component.

6.1 Backbone

Bag of freebies for backbone consists of:

- **CutMix** augmentation: patches are cut and pasted among training examples, ground truth labels are mixed proportionally to the area of the patches.
- **Mosaic** augmentaion: combines 4 training images in 1 with certain ratio.
- **Class label smoothing**: adjusting the target upper bound of a prediction to a lower value (e.g. 0.9)
- **DropBlock regularization**: discards features in a contiguous correlated area.

The backbone **CSPDarknet53** consists of **Dense blocks** - multiple convolution layers composed of batch normalization, ReLU, and followed by convolution. Instead of using only last layer output, each layer takes as input the outputs from all previous layers as well as original input.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 5: CSPDarknet53 model overview

Bag of specials for backbone consists of:

- **Mish activation**: self-regularized, non-monotonic activation function $f(x) = x \tanh(\ln(1 + e^x))$

- **Cross stage partial networks**. CSP separates the input feature maps in two parts. First part bypasses the dense block and goes directly to the next layer.Second part passes through the dense layer.

6.2 Head

Bag of freebies for head:

- **CIoU-loss** - $\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(b,b_{gt})}{c^2} + \alpha v$, where c - diagonal distance of minimal enclosing box, $\rho(b, b_{gt})$ - euclidean distance between central points of BB, v measures the consistency of aspect ratios;
- **Cross mini Batch Normalization**
- **Self-Adversarial Training**
- **Eliminate grid sensitivity**
- **Using multiple anchors for a single ground truth**

7 2-class Object detection using YOLOV4

The first method that we tried consisted in using a pure YOLO approach for 2 classes. Helmet and Collision. To do this we generated a set of images from the videos where there were collisions like in 1 and also another set of images without collisions.

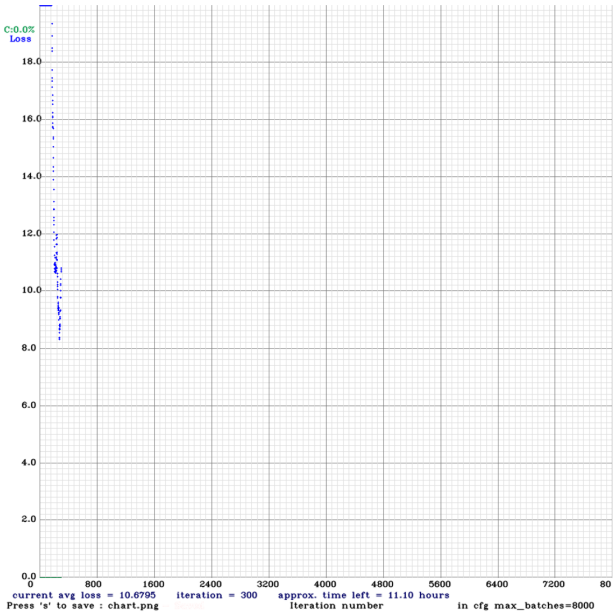


Figure 6: YOLO training curve

As we can see in 6 there was still remaining 12 hours of training but Google Collab gives only a certain amount of time and resources of GPU so we couldn't continue the training.

7.1 Data filtering using sensors

The data from the sensors can be used to filter out false positives from the YOLO Output, it can be used to train a normal classifier given the features of each player at every frame of the video or to take into consideration a short sequence of features by using LSTM, since a collision is directly connected to these physical features.

Training Results			
method	mAP	IOU	Class
YOLOV4	0.35	0.73	0.89

Table 1: YOLOV4 Results.

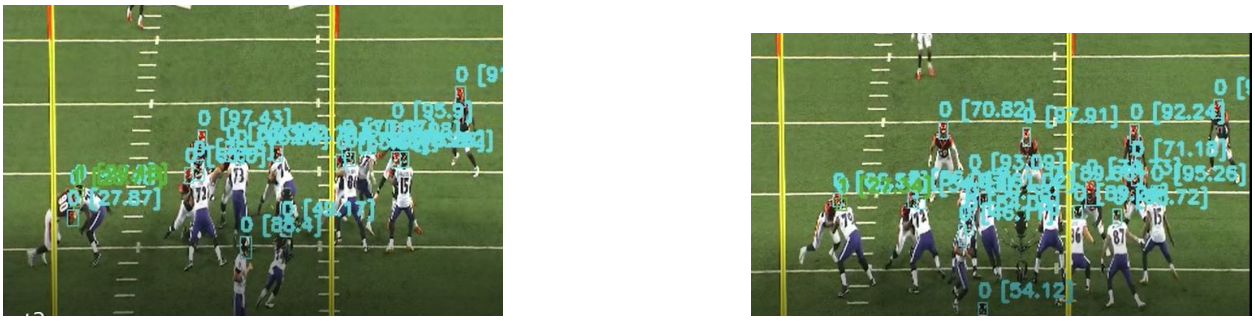


Figure 7: Model Prediction using two class training

8 Stacked Models

One of the methods we tried to address the challenge is subdividing the whole problems into two subproblems and use to a Deep Learning and a Machine Learning model to solve each subproblem. The result is a stacked framework made of two stacked models as shown in Figure 8.

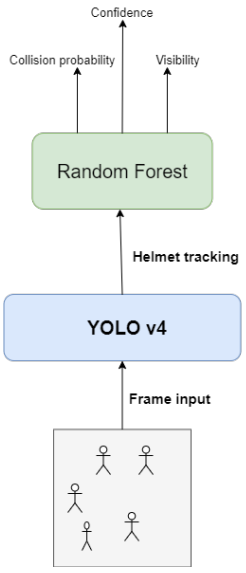
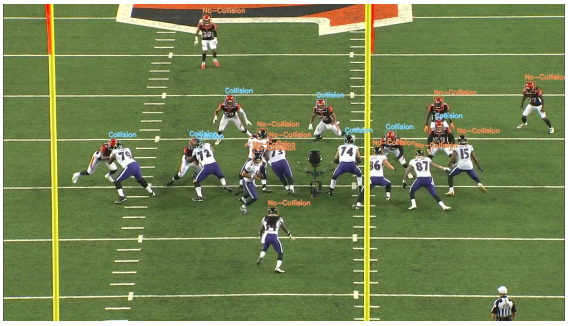
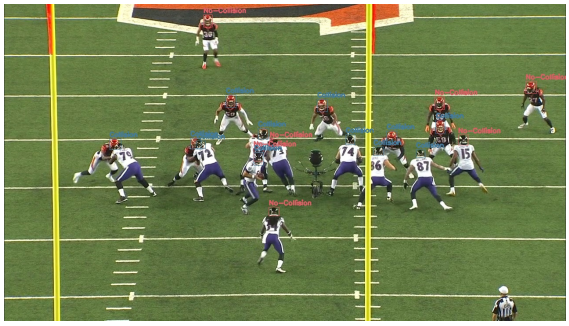


Figure 8: Stacked models

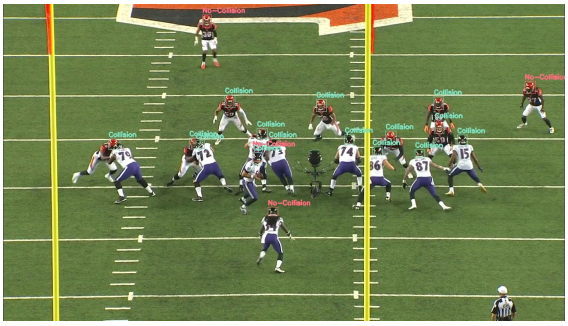
The idea is to detect the helmets collision in a video by tracking the position of pairs of helmets in each frame. The Yolo model takes in input the video frame, tracks the helmets and pass the boundary centers, as long as the view used (Endzone or Sideline), of pairs of players to the Random Forest, which finally that outputs the helmets collision probability, the confidence, and the visibility. We then use the last layer input to display the collision, confidence and visibility greater equals than a threshold as shown in Figure 9. However, we encountered the following issues using this approach:



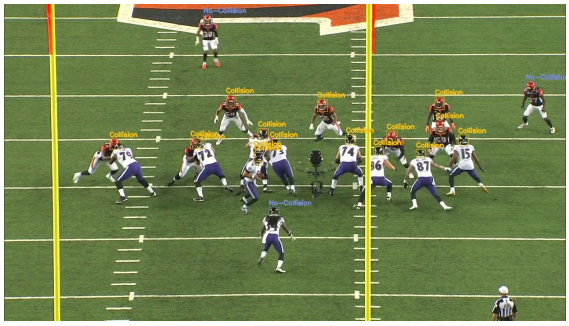
(a) Collision Threshold 0.9, Confidence Threshold 0.7 Visibility threshold 0.5



(b) Collision Threshold 0.8, Confidence Threshold 0.5 Visibility threshold 0.5



(c) Collision Threshold 0.7, Confidence Threshold 0.3 Visibility threshold 0.3



(d) Collision Threshold 0.5, Confidence Threshold 0.3 Visibility threshold 0.3

Figure 9: Collision detection variation based on the thresholds

- **Lack of data:** The dataset we had, contained only 3257 effective collisions among the 4635400 possible interactions (total games × total players × total frames), which is not enough to learn the features of a collision between two players given a view (Endzone, Sideline).
- **Yolo training time:** training the YOLO model was a bit painful as it required resources that we don't have. Plus, using Google collab was also challenging as it was constantly disconnecting.
- **Detection time complexity:** Detecting the helmets is fast but process of displaying the collision is computationally expensive. The time complexity is of the order of $d \times n^2 \times y \times r$ where d is the total of frames of the video, n is the total of players in the game, y is the time of the YOLO model to detect the helmets, and r is the time complexity of the Random Forest to detect a collision.

As we could not solve these issues, we decided to evaluate the performance of this method using simple models on images instead of videos.

9 Results

9.1 Stacked models

The stacked models framework scored pretty well on the helmet detection, but no so well on the collision detection because of the lack of data to learn enough such event as we can see in Figure 9. Indeed, it seems like that our best model (The Random Forest), didn't learn enough the maximum distance between two boundary centers to detect the collision: The players that are far away from a group of players, have a collision probability less that 0.5. Table 2 is a comparison of test scores of some of the machine learning models trained, and the minimum YOLO average loss achieved so far. As we can see from the table, the random forest achieved the best accuracy for the collision detection among the other models, this is why we used it as a final model to use in the second layer of our stacked models framework.

Performances		
Model	**Average Loss**	**Accuracy**
YOLOV4	7.35	
Random Forest		0.86
MLP		0.66
SVM		0.34

Table 2: Comparison of performances of the models tried for collision detection and YOLO

10 Conclusion

One of the main challenges of implementing computer vision projects is the necessity of enough computational resources to train Deep Neural Networks and the amount of data you have to train. For this reason we had to change project since we couldn't get the necessary amount of data. In this project we managed to train and test two different models in a classic but not easy computer vision problem: collision detection. We tried different approaches and only reported the ones that work slightly better, as the lack of data and computational resources encountered during this project, has led us to conclude it with results lower than our expectation, thus not satisfying. However, we learned a lot during this project and we are convinced that we could have done a bit better if we had the time and the resources. A future work on this project would be the usage of the sensor data and the frame to predict an upcoming collision using a Recurrent Neural Network model in specified time window.

11 Implementation and Demo

For this project, we do not provide a video demo but images and a code to run locally for test. However, an installation of darknet YOLOv4 needs to be done and some heavy data needs to be downloaded in order to run correctly the code:

- Link to the Google Colab for the Colision Detection using YOLOv4:https://bit.ly/2K7cuXX
- The file to test the stacked model framework is :helmet_detection.py
- The file to see the training of the model of collision detectioon :helmet_detection.py
- Tutorial on how to install darknet YOLOv4 locally to run the stacked models framework: https://github.com/AlexeyAB/darknet
- Weights of YOLO models + config files + Random Forest model for the stacked models framework: https://drive.google.com/drive/folders/1esmoEQkthV36sVual$_0WcPVrtfJ948UR?usp = sharing$