PROCESSING AND CLASSIFYING AMERICAN FOOTBALL IMAGES

Andrew T. Annnestrand

University of Texas at Austin
Electrical and Computer Engineering

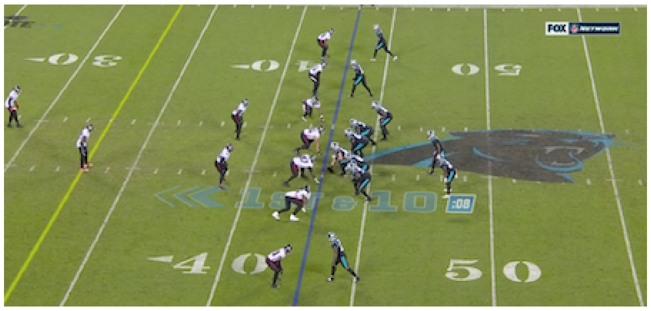


Fig. 1: Defense on the left, offense on the right

ABSTRACT

American Football (or just Football) is the most popular sport in the United States of America due to its dynamic and exhilarating games. The sport is not only dependent on talented players, but it also heavily relies on tactical coaching. Every week, the coaching staff will tirelessly analyze the opponents tendencies, strengths, and weaknesses through past game film. Before the film is analyzed by coaches, it has to be tagged and organized so that the players and coaches have ease when going through past games. Most of this tagging and preparation of film is done by humans. In this paper, I explain an attempt to find ways to move this process of preparing film from human hands into the hands of digital image processing and machine learning.

1. INTRODUCTION

Before getting into processing images, it is important to understand the essential components of football. There are two teams in the game and each team is either defense or offense in a "possession." Each possession consists of multiple plays. A play occurs once the ball is snapped from the offensive line into the quarterbacks hands. Moreover, rules dictate that unless a player is "in motion" (moving from one set position to another) they must be still and cannot abruptly move. Only the offense must follow this rule, whereas defensive players are free to move wherever. This means that we will have static moments throughout the whole game which will allow us to capture images of "pre-snap" formations and analyze them.

2. GATHERING DATA

Perhaps one of the most time consuming parts of this project was the gathering and labeling of the data. All of the data was captured through a python script that took screenshots of youtube clips. This part was not all that difficult, but once I began building the various classifiers (offensive formation, defense vs. offense) it required labeling the data so that a degree of accuracy could be achieved. The data is from a wide range of NFL games which results in many different colors, angles, fields, logos, and more. Due to the diversity and resolution of the images, many details had to be cut out before usage.

3. PRE-PROCESSING IMAGES

Before we can make any inference or predictions on the formations of the football teams, we need to decrease the noise of the image. For example, the image in **Fig. 1** is an example of a formation we would like to analyze. However, having the defense in the picture is simply just unnecessary noise when it comes to predicting the formation of an offense. Thus, in the following sections we will apply traditional image processing techniques to give us more useful data.

3.1. Finding the line of scrimmage (LOS)

The line of scrimmage is the line that runs through the footballs location, orthogonal to the sidelines of the football field. This is an important line because the defense and offense line up on it before every snap. Both the offense and the defense must stay on their respective sides before the ball is snapped, other wise it will result in a penalty. Thus, we have a natural separation between the players of the offense and defense. This line will be the basis of how process the image. The LOS can be marked in a variety of different colors, depending on the network that is covering the game (see **Fig. 2**). However, the majority of them use a dark blue to denote this, something we will leverage later.

The very first idea that I chose to pursue from class is edge detection, given the large amounts of lines. In **Fig. 3** an image is displayed after applying a canny edge detection. While it is interesting to look at, unfortunately it doesn't do too much

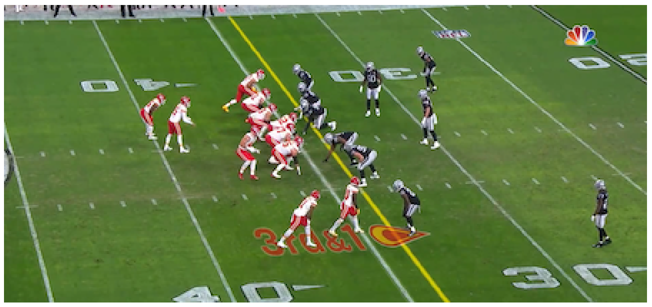


Fig. 2: Example of NBC using a white line for LOS

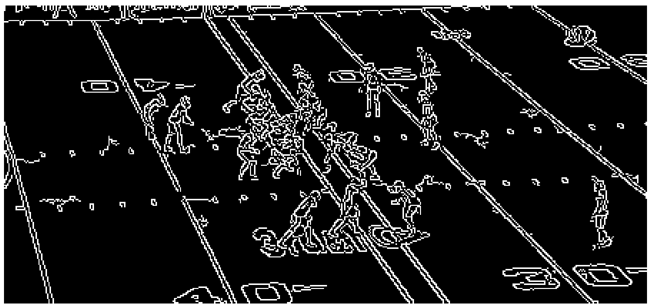


Fig. 3: Image following a canny edge detection

for us by itself. However, we can go a step further and use a Hough Transform to pick specifically on lines, not just all edges. The Hough Transform is an algorithm that uses an accumulator matrix in order to detect if a straight line exists between pixels. The transform also uses voting to see what sets of pixels to truly be certain if there is a line present or not. When taking the Hough Transform, we specify different parameters to ensure that we are detecting the correct type of lines. Some of the import ones include maxLineLength, maxLineGap, and a threshold for number of votes required for a line to be true. The maxLineLength and maxLineGap parameters allow us to specify how lenient we are with gaps, as well as how long a line can be before it becomes a small, separate line.

Depicted in **Fig. 4** is the result of running our edge image in **Fig. 3** though a Hough Transform for line detection. As you can see, it picks up on most of the yard line markings along the field, as well as the LOS. If we decrease the threshold value and maxLineGap, we can expect a much different picture where there is too many detected lines (**Fig.5**). As you can see, there is quite a lot of freedom to detect lines at a variety of angles. Our first set of parameters does a great job in detecting the lines which we value most. But, how exactly do we separate the LOS from the rest of the line markings? In order to do this accurately I had to come up with my own algorithm. I took advantage of one of the constant happenings in every image of the dataset, the players concentrated at the line.

Given the capturing of the players through the canny edge

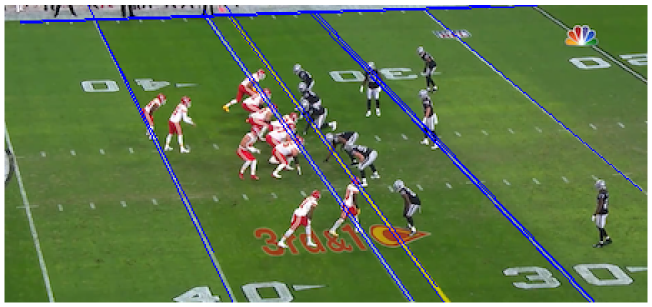


Fig. 4: Example of NBC using a white line for LOS

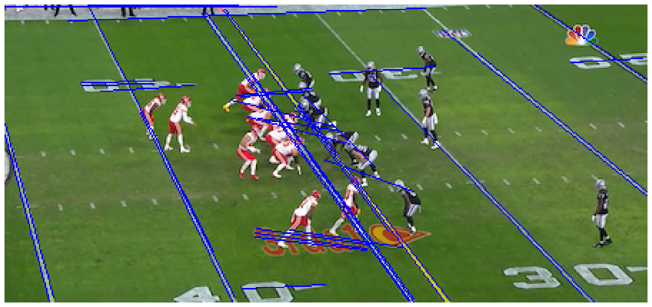


Fig. 5: Image following a canny edge detection

detection, I decided that if I could find the location of the highest player density, then the LOS would be near. In order to find this player density, I took the gradient of the image using a Sobel kernel (in the y direction). The reason why I used Sobel was to specifically highlight the players gradients, and not so much the vertical lines in the image. Once the image gradient was calculated, I decided to employ a common technique in engineering – the sliding window. The sliding window was chosen to be around (200,100) pixels and it was slid across the entire image at 50 pixel intervals. During each iteration of a new window, the average absolute value of the window was calculated and compared to the current max value. If there was a window with a higher average gradient, its pixels were saved and new max value updated. As it turns out, this is very accurate and gives us a window that includes the LOS. Some examples can be seen in **Fig. 6**.

Now we have a window in which we know the LOS resides, as well as all of the straight lines from the Hough Transform. Putting these two algorithms together, I created a voting system to determine which Hough line was the LOS. For each line identified, the number of pixels the line had in the window was calculated. Ultimately, the line with the most amount of pixels inside the window containing the LOS was determined to be the LOS. Now we have determined a crucial part of the processing, on to separating the defense and the offense.

```
Algorithm 1: LOS Detection
Data: $I(i, j)$
Result: $i_{max}$,$j_{max}$
max = 0;
$i = 0$;
while $i < numrows$ do
    $j = 0$;
    while $j < numcols$ do
        $W = Sobel[I(i : i + 200, j : j + 100)]$;
        if $mean(abs(W)) > max$ then
            $max = mean(abs(W))$;
            $i_{max}, j_{max} = i, j$;
        else
            nothing;
        end
        $j+ = 50$;
    end
    $i+ = 50$;
end
```

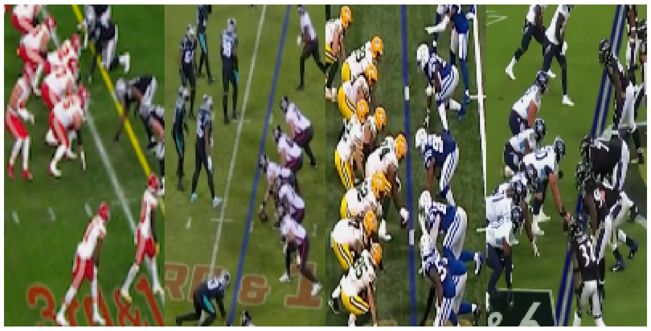


Fig. 6: Windows of LOS using algorithm

3.2. Rotating and splitting the image

In order to separate the image, we must divide it along the LOS to have two images of the offense and defense respectively. Given that the LOS in the image can be at a variety of angles, we should generalize the process so that the LOS is always vertical before splitting (this also makes it easy to split the image on one x value). Therefore, we should rotate the image based on the angle that the LOS makes with the x-axis of the image. We can calculate this angle using simple trigonometry of right triangles. Displayed in **Fig. 7** is the result of rotating the image based on the angle of the LOS, as you can see the LOS is now a vertical line (sits on a constant x value). Now, we simply use indexing into the image array to divide our image into offense and defense. The result is show in **Fig. 8**. Again, the importance of these steps is that we want to analyze the formations of each side separately. That is, we don't want the unneeded noise of the other side increasing our training times, causing less inference power, or totally misleading our models. Now that the main part of

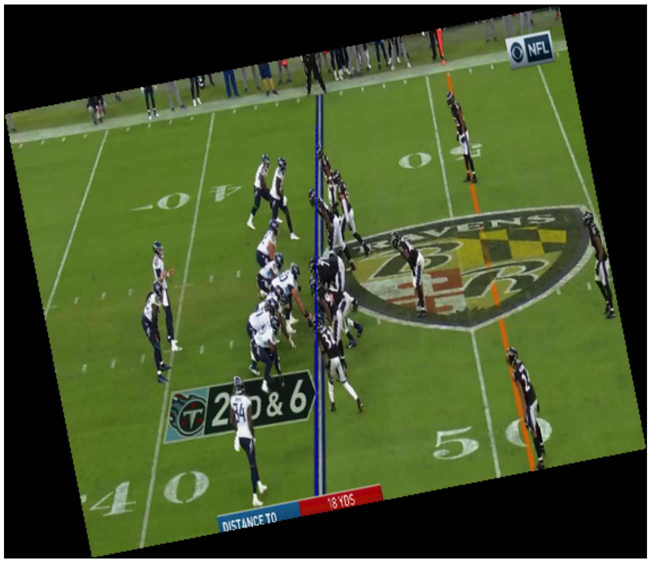


Fig. 7: Rotated image so that the LOS is now a vertical line

the image processing is done (there are a few more processing techniques that were used), we can start labeling the data (if the half is offense of defense, and what formation) so that it is ready to train a ML model.

3.3. Additional processing

As mentioned before, there was some additional steps that needed to occur before training models could occur. The main one was the fact that these images were very large and took up gigabytes of memory. Given my limited access to computation power, I had to convert all of the images to grayscale so that my machine could actually load all of them into memory. Of course this means that I lost valuable information in the color of the images, but the models still performed well (mainly because inference is based on spatial relationships in the image). Lastly, one minor detail was decreasing the overall brightness of the image. This technique not only helped decreased defects such as glare (during an afternoon game) and very bright white jerseys, but it also helped delineate the LOS in most cases.

4. CLASSIFYING THE IMAGES

Back to the main point of this project, we want to be able to make game film processing more efficient and take it out of the hands of humans. One of these processes includes labeling a play, or who is doing what on a given play. For example, a coach may want to look at all defensive possessions of a team in a game to see how they line up. This brings us to the first type of classifier that was trained, predicting if an image was of an offensive formation or defensive formation using a Convolutional Neural Network. The second type of classifier

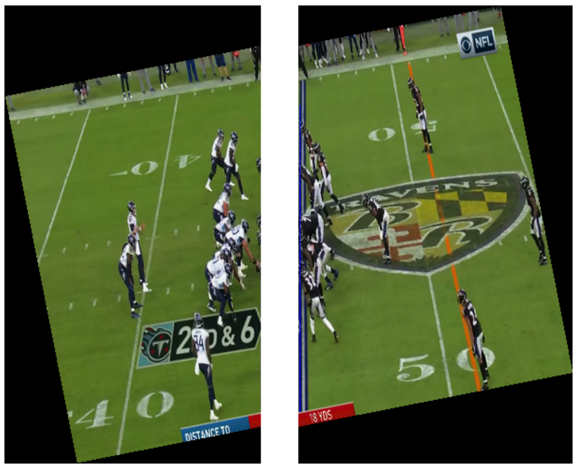


Fig. 8: Two images, split into offense and defense

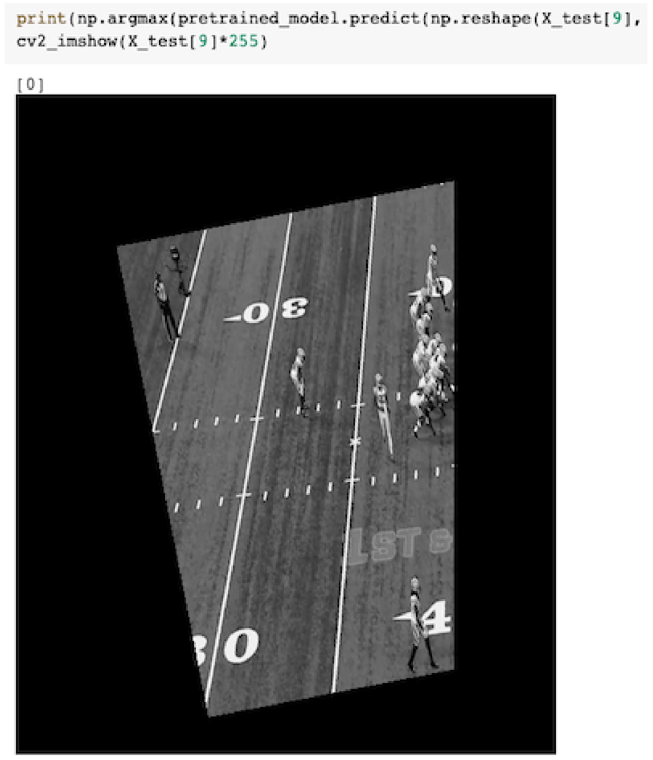
that was built was another CNN, but this time used to predict the type of formation the offense was in.

4.1. Defense or Offense

For the first classifier, I created a CNN with 3 convolutional+pooling layers, as well as a few dense layers after flattening the convolutions. Given that the classification problem that I was solving was based on two classes, I used a sigmoid activation as my last layer to predict either a 0 or 1 (offense or defense). The data that I used was split up into training, validation, and testing data. Below you can see the validation loss/accuracy of the model over training, as well as the testing accuracy after. training was complete. Overall the testing accuracy would range from 81 $-$ 86% for most of the trials that I ran. Considering the size of the data set (around 200 images for each class), I was satisfied with the overall quality of prediction. I believe that if I were to grow the dataset into the order of 1000's, as well as increase the depth of the neural network, I could see accuracies being in the 90s.

4.2. Formation of Offense

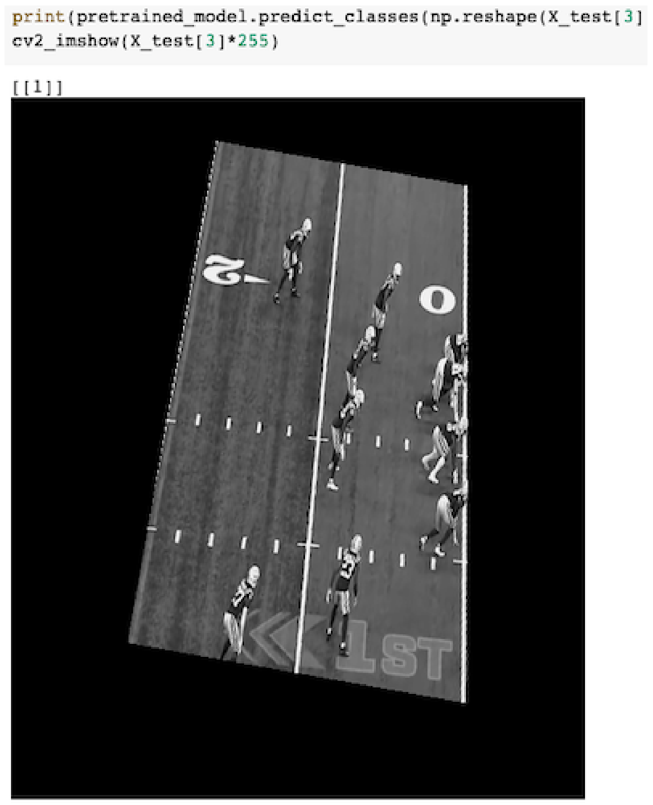
There are many different offensive formations that can be run in football, however I chose the 4 most common ones. Each formation was encoded with the following values for ML training: 0-empty set (only QB in backfield), 1-single set (QB under center, RB in backfield), 2-shotgun (QB and RB in backfield), 3-iformation (QB under center, FB and RB in backfield). I used the exact same architecture as the first, but I decreased the number of units in each layer by around half. The reason I did this was because the size of the dataset for this task was significantly smaller (around 40 images per



```
print(np.argmax(pretrained_model.predict(np.reshape(X_test[9],
cv2_imshow(X_test[9]*255)
```

```
[0]
```

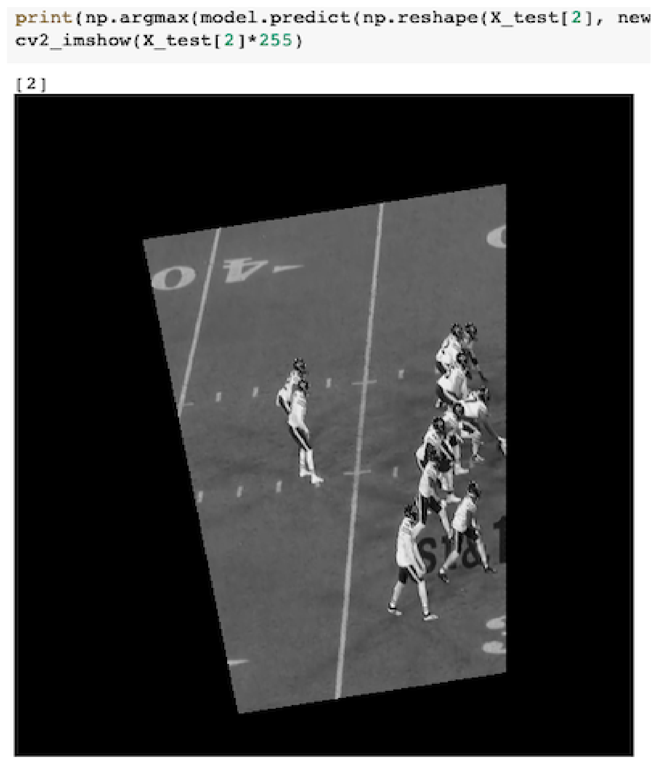
Fig. 9: CNN correctly classifying an offense



```
print(pretrained_model.predict_classes(np.reshape(X_test[3]
cv2_imshow(X_test[3]*255)
```

```
[[1]]
```

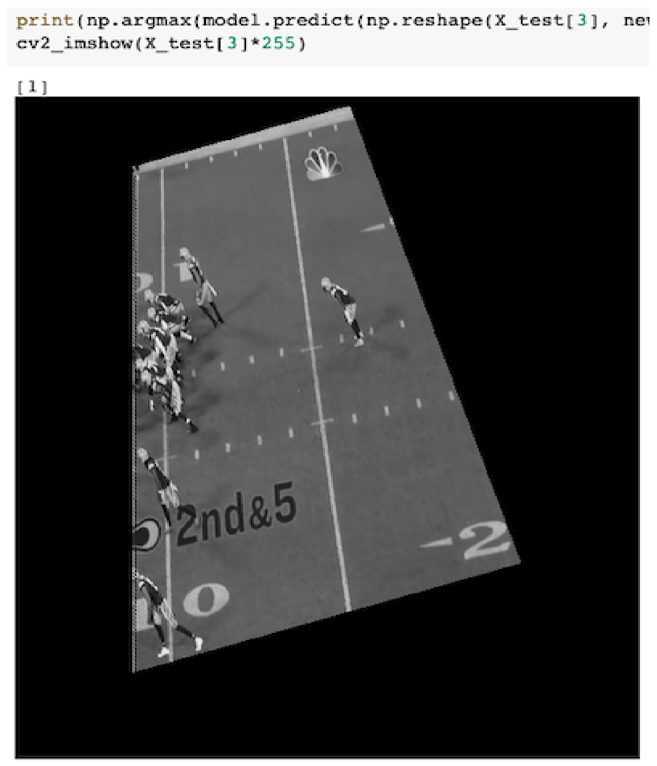
Fig. 10: CNN correctly classifying an defense



```
print(np.argmax(model.predict(np.reshape(X_test[2], new
cv2_imshow(X_test[2]*255)
```

```
[2]
```

Fig. 11: Model correctly classifying shotgun formation



```
print(np.argmax(model.predict(np.reshape(X_test[3], ne
cv2_imshow(X_test[3]*255)
```

```
[1]
```

Fig. 12: Model correctly classifying single set-back formation

class). Another modification I had to make was to make the last layer of the network a softmax output so that our output from the CNN is the probability of the image being each separate class. Due to lack of data, as well as increase dimensionality (4 classes instead of 2 now), the accuracy of this model was much lower than the previous. In fact, even after adding dropout layers and some type of overfitting occurred in the training process. However, my testing accuracy was actually much higher than my training or validation – boasting in the 90% range. Again, these results are hard to trust due to the lack of data, but there are some examples in **Fig. 11, Fig. 12** of the model correctly inferring an offensive formation.

5. CONCLUSION

5.1. What I learned

This project allowed me the opportunity to apply many of the learnings from class into an actual dataset. Specifically, I leaned the value and effectiveness of edge detection and image gradients. I also learned about the value of labeled data. It took quite a lot of effort to build up my dataset, and I can definitely say I will be more grateful for complete datasets in the future. When it comes to machine learning, I learned how to create Convolutional Neural Networks using TensorFlow. Not only did I learn how to construct them, but I also learned the importance of each type of layer (convolutional, pooling, activation, etc.) and how they process an image. CNNs are currently the state of art when it comes to ML tasks with image processing, and I am glad to have learned a valuable skill.

5.2. Future work

This project was very fulfilling, however I believe that there is still so much more that can be done. One major feature I would like to build upon is the captioning of images. The current state of my classifiers allows me to understand what class an image belongs to, but I believe implementing some type of RNN that could turn an image to a sentence description about the play could be achievable. Furthermore, the method used in this project was to manually take screenshots of gameplay – which is quite undesirable. I plan to take Digital Video Processing next semester, and perhaps I can use techniques to automatically capture those images from a large video (a whole game).