Positioning in International Football: An Exploratory Analysis Using Convolutional Neural Networks

Mark Gjuraj '22 Yale University Statistics and Data Science Spring 2022

Contents

Abstract

Drawing from the data made freely available by StatsBomb[1] (via their `StatsBombR`[2] package), we craft *heat maps* using event data from the two most-recent major international football competitions: the 2018 FIFA World Cup and the 2020 UEFA European Football Championship (Euros). We experiment with different bandwidths and image resolutions as we generate these heat maps using 2-dimensional Kernel Density Estimation (KDE), subsequently running these heat maps through a Convolutional Neural Network in the effort of predicting the position their corresponding players were playing. We train this classification model on the heat map images of all players not on the Belgian, Croatian, English, or French national teams, as these European nations were uniquely successful in both competitions, and thus serve as a natural test set. Trained on a collection of just over 3000 images, our classifier correctly predicts the positions for unseen heat maps with an accuracy of approximately 56.5%. The incorrect predictions made by our model for each position effectively illuminate our football intuition by capturing various circumstantial behaviors and individual tendencies of football teams and players that are often difficult to quantify.

Introduction

Analyzing football (soccer) using data is, quite naturally, a daunting prospect. For many, to attempt to boil down the intricate dance that is a football match – one rife with the nuances of human behavior, shrouded in the nebulous, unquantifiable haze of public narrative and personal motivation – is to seemingly strip the beautiful game of its very beauty. By virtue of football's free-flowing format (two 45-minute halves with sparse stoppages), football analytics are uniquely challenged in that there are no "plays" or "at-bats" that partition the match into standardized events with definite beginnings and ends. This said, there are discrete conventions – the very tactics that football coaches use to instruct their players and inform their decision-making – that enable us to achieve analogous categorizations.

One such convention is the notion of *formations*: frameworks which define a team's general shape and positional structure. Knowing the formation a team is playing at any particular moment in a match gives us a rudimentary understanding of that team's strategy and play style. For instance, a *4-4-2* denotes a shape comprised of two wide defenders (a left and right "back"), two central defenders (a left-center back and a right-center back), two wide midfielders (left and right), a left-center midfielder, a right-center midfielder, and two center forwards (left and right). Of course, the rich breadth of variations on the *4-4-2* alone – such as versions which involve advanced wide midfielders (left/right "wingers"), and versions where the central midfielders are either stationed deeper in order to screen the back-line (defensive midfielders) or sit higher up the pitch to aid the forwards (attacking midfielders) – underscores the coarseness of the information a formation provides. The same diversity characterizes individual *positions*; just as two teams who play the same formation might deploy slightly different variants in order to fit their personnel, two players who play the same position might exhibit dramatically different behaviors depending on their individual strengths and weaknesses.

Heat maps are prominent football visualization tools which illustrate the activity of a player spatially throughout the pitch. Their premise is intuitive: based on a given continuum of colors, the presence of high-valued colors in some region of the field means that the player frequented that area relatively often. However, whether it be due to the opacity underlying the construction of heat maps, or a lack of context often accompanying them, the manner in which football consumers and professionals alike should go about assessing these images in order to draw meaningful conclusions is far from obvious. The method we focus on in this exploratory report is predictive modeling using *Convolutional Neural Networks*. CNNs are a flavor of neural networks that are often used with image-structured data (i.e. 3-dimensional arrays, where the third dimension corresponds to the number of channels in an image; RGB images have 3 such channels). Defined by the use of *convolutional layers* – layers consisting of arrays of parameters (filters) that are swept (convolved) over the pixels of an image – CNNs learn the underlying, latent features of the images they are trained on, enabling them to classify novel images with high accuracy.

In **Getting Started** and **Data Exploration**, we guide the reader through our data retrieval and cleaning process, and point the reader to a custom R Shiny application[3] that visualizes the heat maps we create. In **Model Overview and Selection**, our CNN is defined, and the parameters we use to create the heat maps that train the CNN are selected using cross validation. The predictions our CNN makes on our Belgian, Croatian, English, and French heat maps are assessed in **Results**.

Getting Started

Dependencies

To replicate our project's results, begin by visiting our GitHub repository `HeatMapsCapstoneRepo`[4]. Download the `data`, `model`, and `R` folders, and ensure that in your current working directory you have an `.Rproj` object. Also check that you have the following libraries installed:

```r
library(shiny)
library(StatsBombR)
library(SDMTools)
library(tidyverse)
library(dplyr)
library(reticulate)
library(ggplot2)
library(MASS)
library(reactable)
library(gridExtra)
library(tensorflow)
library(keras)
library(kableExtra)
```

The `R` folder houses our R scripts `R/HeatMapRawData.R`, `R/HeatMapPlayerData.R`, `R/HeatMapsCreate.R`, and `R/HeatMapData.R`:

```r
source("R/HeatMapRawData.R")
source("R/HeatMapPlayerData.R")
source("R/HeatMapsCreate.R")
source("R/HeatMapData.R")
```

Data Exploration

Data Retrieval and Overview

The function `get_statsbomb_data()` yields a slew of data frames for both the World Cup and Euros. There is no need to run this function if you've downloaded the `data` folder off of GitHub, as this function is what creates `data/StatsBombData.Rdata`. The function `check_statsbomb_data()` ensures that this `.Rdata` file exists where it should, and instructs you to load it in.

```
get_statsbomb_data()
```

```
check_statsbomb_data()
```

```
Now run the following if you have not already: load("data/StatsBombData.Rdata")
```

```
load("data/StatsBombData.Rdata")
```

- `matches_WC` and `matches_Euros` have 64 and 51 rows respectively, where each observation corresponds to a match. This data frame provides summary information for each match in these competitions. Consult the following document[5] provided by StatsBomb for more information regarding this data's formatting.
- `lineups_WC` and `lineups_Euros` have 128 and 102 rows respectively, where each observation corresponds to a team in a particular match. Nested within each row is another data frame that gives information on each player who was included in that team's squad for that match, and nested within the rows of this data frame are data frames outlining the position(s) that each player played throughout the match. See the following document[6] provided by StatsBomb for more information regarding this data's formatting.
- `events_WC` and `events_Euros` have 227886 and 192695 rows respectively, where each observation corresponds to a match event. Events include everything from *passes* and *shots* to *duels* and *fouls committed*. The following diagrams, included in this data's specification[7] provided by StatsBomb, are essential viewing:
  - **Figure 1 (left)** serves as a reference for the way in which the location coordinates of each event map onto the pitch. The key takeaway here is that the location data is not absolute: in practice, both teams are attacking in opposite directions, and they switch the goals they are defending at halftime, so having the true locations of each event would not be very informative. StatsBomb accounts for this, thwarting this issue by setting the universal direction of attack as going from left to right.
  - **Figure 1 (right)** illustrates the general conventions for how different positions are defined on paper. If we return to our *4-4-2* example, focusing on the positions that comprise this formation makes the self-explanatory nature of its name apparent, as it consists of a line of 4 players at the back (LB, LCB, RCB, RB), a bank of four in the middle (LM, LCM, RCM, RM), and 2 sitting up top (LCF, RCF).

The function `get_involved_players()` yields `data/Players.Rdata`, which possesses three data frames: `allplayers` (4210 rows), `allplayers_WC` (2229 rows), and `allplayers_Euros` (1981 rows). Each row corresponds to a unique player-match-position tuple.

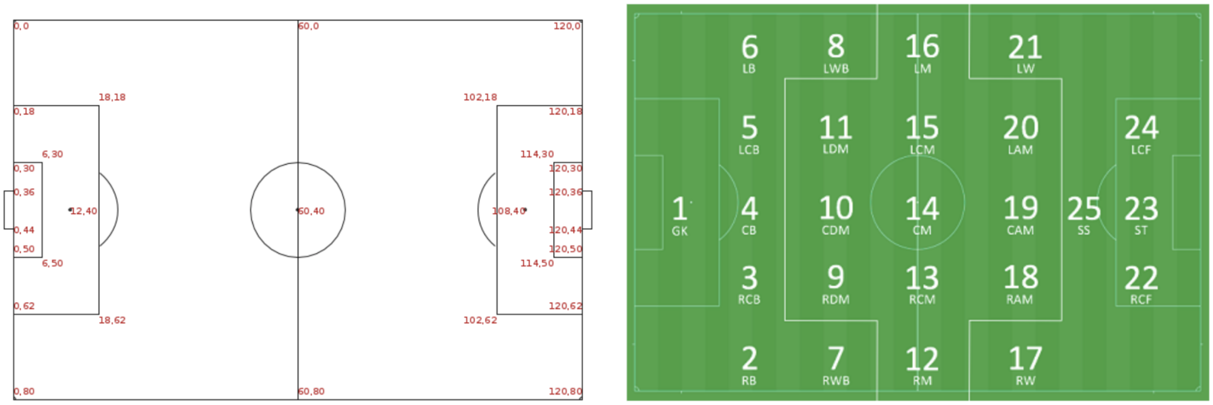


Figure 1: Pitch Coordinates (Left), Positional Template (Right)

```
get_involved_players()
```

```
check_involved_players()
```

```
Now run the following if you have not already: load("data/Players.Rdata")
```

```
load("data/Players.Rdata")
```

For example, suppose our individual of focus is the Belgian winger Eden Hazard (our favorite player). Using `allplayers` (or `allplayers_WC`), we can take a look at his World Cup outings:

playerID	playerName	matchID	team	involvement	position	n_events
3621	Eden Hazard	7536	Belgium	Starting XI	Left Attacking Midfield	164
3621	Eden Hazard	7536	Belgium	Starting XI	Left Wing	47
3621	Eden Hazard	7552	Belgium	Starting XI	Left Wing	118
3621	Eden Hazard	7552	Belgium	Starting XI	Left Center Forward	18
3621	Eden Hazard	7584	Belgium	Starting XI	Left Attacking Midfield	223
3621	Eden Hazard	8650	Belgium	Starting XI	Left Wing	161
3621	Eden Hazard	8650	Belgium	Starting XI	Left Center Forward	39
3621	Eden Hazard	8655	Belgium	Starting XI	Left Wing	218
3621	Eden Hazard	8657	Belgium	Starting XI	Left Wing	236

Eden Hazard started in every game (in which he appeared) at the World Cup, and in numerous matches he shifted positions in the middle of the match. However, there are instances, such as his stint at *Left Center Forward* against Tunisia in the Group Stage (match 7552), where he was only involved in 18 events. This is a relatively small number of involvements compared to, say, his performance against Japan in the Round of 16 (match 7584), where he stayed at *Left Attacking Midfield* the entire match and was, consequently, highly active (223 events).

This alerts us to the fact that there may be instances where players played positions for such a short period of a particular match that they only had a handful of involvements (if any); we want to account for this, as such observations may make it much more difficult for our CNN to correctly classify these positions.

There are 155 such occasions where a player did not register a single event while playing in a particular position during a particular match. Obviously, we should get rid of these rows, as heat maps cannot be made without events.

	playerID	playerName	matchID	team	involvement	position	n_events
151	21261	Jonas Older Wind	3795221	Denmark	Substitute	Left Center Back	0
152	7843	Mason Mount	3795506	England	Starting XI	Center Attacking Midfield	0
153	9638	Jack Grealish	3795506	England	Substitute	Left Center Midfield	0
154	12423	Jadon Sancho	3795506	England	Substitute	Right Back	0
155	22084	Bukayo Saka	3795506	England	Substitute	Right Wing Back	0

It is noteworthy that, if the following removal step is not done, the functions `get_heatmaps()` and `clean_heatmaps_and_labels()` will deal with these event-less instances accordingly. This will become more clear shortly.

```r
allplayers <- allplayers %>%
  filter(n_events != 0)
dim(allplayers)
```

```
[1] 4055   20
```

In fact, we will elect to only consider instances where the player had at least 10 involvements in their position throughout the match:

```r
allplayers <- allplayers %>%
  filter(n_events >= 10)
dim(allplayers)
```

```
[1] 3809   20
```

Football enthusiasts will recall Eden Hazard's stellar performance in the Quarter Finals against Brazil (match 8650). We'll return to this outing in a moment.

Heat Map Creation and Cleaning

Given $(X_1, Y_1), \cdots, (X_n, Y_n)$, 2-dimensional KDE using Gaussian kernels centers a Gaussian distribution with bandwidth `h` (the standard deviation in both the X and Y direction) on each of these points in 2-dimensional space (the pitch). Essentially, it considers a player's events as observed values – a sample from their unknown true event distribution – and seeks to approximate their true distribution through a mixture of Gaussian bumps.

Recall Figure 1's insight into the way in which the field's dimensions are encoded. We know that our location's X values range from 0 to 120, and Y values range from 0 to 80. The novel X and Y values we consider in our KDE are defined by the parameters `x_res` and `y_res`. `x_res = 1`, for example, yields a set of X values $\mathcal{X} = (x_1, x_2, x_3, \cdots, x_{119}, x_{120}, x_{121}) = (0, 1, 2, \cdots, 118, 119, 120)$, while `x_res = 10` yields $\mathcal{X} = (x_1, x_2, x_3, \cdots, x_{11}, x_{12}, x_{13}) = (0, 10, 20, \cdots, 100, 110, 120)$. Thus, a larger *resolution* parameter corresponds to less pixels, which is obviously the opposite of the typical notion of image resolution – aware of this, let's adopt this backward terminology for ease of discussion.

Our 2-dimensional KDE generates a 2-dimensional array – technically a 3-dimensional array with a single channel – of size $|\mathcal{X}|$ by $|\mathcal{Y}|$, where entry (i, j) of the array is:

$$\begin{aligned}\hat{f}(x_i, y_j) &= \frac{1}{n}\sum_{m=1}^{n} K_h(X_m, x_i) * K_h(Y_m, y_j) \\ &= \frac{1}{n}\sum_{m=1}^{n} \frac{1}{\sqrt{2\pi}h} e^{\frac{-1}{2}(\frac{X_m - x_i}{h})^2} * \frac{1}{\sqrt{2\pi}h} e^{\frac{-1}{2}(\frac{Y_m - y_j}{h})^2} \\ &= \frac{1}{nh^2 2\pi}\sum_{m=1}^{n} e^{\frac{-1}{2}\left((\frac{X_m - x_i}{h})^2 + (\frac{Y_m - y_j}{h})^2\right)}\end{aligned}$$

In `R/HeatMapData.R` we find `get_heatmaps()`, which calls upon the data we have loaded into our local environment – as well as the function `heatmap_KDE()` from `R/HeatMapsCreate.R` – in order to create a heat map corresponding to every row of `allplayers`.

Specifically, the function `get_heatmaps()` creates a list of heat map objects (2-dimensional arrays) for each row in `allplayers` given `h`, `x_res` and `y_res`, where very small values (< 0.00001) in each heat map array are zeroed-out. Each array is also normalized so that its values lie between 0 and 1.

We create heat maps for a handful of different combinations of bandwidth and resolution:

```r
dirty_heatmaps_3_1 <- get_heatmaps(h = 3, x_res = 1, y_res = 1)
dirty_heatmaps_3_3 <- get_heatmaps(h = 3, x_res = 3, y_res = 3)
dirty_heatmaps_3_6 <- get_heatmaps(h = 3, x_res = 6, y_res = 6)
dirty_heatmaps_5_1 <- get_heatmaps(h = 5, x_res = 1, y_res = 1)
dirty_heatmaps_5_3 <- get_heatmaps(h = 5, x_res = 3, y_res = 3)
dirty_heatmaps_5_6 <- get_heatmaps(h = 5, x_res = 6, y_res = 6)
dirty_heatmaps_10_1 <- get_heatmaps(h = 10, x_res = 1, y_res = 1)
dirty_heatmaps_10_3 <- get_heatmaps(h = 10, x_res = 3, y_res = 3)
dirty_heatmaps_10_6 <- get_heatmaps(h = 10, x_res = 6, y_res = 6)
```

The function `clean_heatmaps_and_labels()` takes a list of "dirty" heat maps and re-formats it as an array of arrays, while also returning a vector of labels (the positions corresponding to these heat maps) and a vector of row indices that allow us to refer back to `allplayers`. As mentioned above, since we removed our undesirable rows from `allplayers` beforehand, this vector of indices is not particularly useful; this function deals with event-less heat maps by keeping track of which rows of `allplayers` have a non-zero number of events, but if we preemptively remove event-less observations from `allplayers`, then every row will have a defined heat map, making this practice unnecessary.

```r
clean_heatmaps_3_1 <- clean_heatmaps_and_labels(dirty_heatmaps_3_1)
clean_heatmaps_3_3 <- clean_heatmaps_and_labels(dirty_heatmaps_3_3)
clean_heatmaps_3_6 <- clean_heatmaps_and_labels(dirty_heatmaps_3_6)
clean_heatmaps_5_1 <- clean_heatmaps_and_labels(dirty_heatmaps_5_1)
clean_heatmaps_5_3 <- clean_heatmaps_and_labels(dirty_heatmaps_5_3)
clean_heatmaps_5_6 <- clean_heatmaps_and_labels(dirty_heatmaps_5_6)
clean_heatmaps_10_1 <- clean_heatmaps_and_labels(dirty_heatmaps_10_1)
clean_heatmaps_10_3 <- clean_heatmaps_and_labels(dirty_heatmaps_10_3)
clean_heatmaps_10_6 <- clean_heatmaps_and_labels(dirty_heatmaps_10_6)
```

	Center Attacking Midfield	Center Back	Center Defensive Midfield	Center Forward	Center Midfield
# of Heat Maps	145	82	137	265	12

	Goalkeeper	Left Attacking Midfield	Left Back	Left Center Back	Left Center Forward
# of Heat Maps	231	36	211	267	129

	Left Center Midfield	Left Defensive Midfield	Left Midfield	Left Wing	Left Wing Back
# of Heat Maps	253	115	114	246	80

	Right Attacking Midfield	Right Back	Right Center Back	Right Center Forward	Right Center Midfield
# of Heat Maps	37	205	274	129	255

	Right Defensive Midfield	Right Midfield	Right Wing	Right Wing Back	Secondary Striker
# of Heat Maps	131	115	241	84	15

In the table above, we can see the composition of our position labels. There is a clear lack of balance in terms of the positions we have heat map data for, and because we only have 3809 total heat maps, this disparity in our sample sizes suggests that we should re-categorize these positions into more generalized buckets. We elect to do so by grouping together positions that we believe, in practice, tend to correspond to the same fundamental roles in the broader context of the team unit:

Left Wing Back → Left Midfield
Right Wing Back → Right Midfield
Left Center Back, Right Center Back → Center Back
Left Defensive Midfield, Right Defensive Midfield, Center Defensive Midfield → Defensive Midfield
Left Center Midfield, Right Center Midfield → Center Midfield
Left Attacking Midfield, Right Attacking Midfield, Center Attacking Midfield → Attacking Midfield
Left Wing, Left Center Forward → Left Forward
Right Wing, Right Center Forward → Right Forward
Secondary Striker → Center Forward

The result is a shrinkage in our number of position classes from 25 to 12:

	Attacking Midfield	Center Back	Center Forward	Center Midfield	Defensive Midfield	Goalkeeper
# of Heat Maps	218	623	280	520	383	231

	Left Back	Left Forward	Left Midfield	Right Back	Right Forward	Right Midfield
# of Heat Maps	211	375	194	205	370	199

Evidently, for each position we now have at least ~200 observations, making us much more confident that our CNN will see enough of each position to be able to learn their latent features and subsequently classify novel heat maps with high accuracy.

Heat Map Visualization and Shiny Application

The function `heatmap_plotter()` defined in `R/HeatMapsCreate.R` allows us to visualize the heat map arrays. Figure 2 below shows six heat maps corresponding to Eden Hazard's activity while playing *Left Wing* against Brazil in the World Cup *Quarter Finals*. From the top left to the middle left, a bandwidth `h = 5` is used for each heat map, while the number of pixels decreases as we move from left to right. Starting from the middle right, a resolution `res_x, res_y = 3` is used for each heat map, while the bandwidth increases from left to right.

```r
p1 <- heatmap_plotter(heatmaps_data = clean_heatmaps_5_1, matchid = 8650, playerid = 3621,
                      pos = "Left Wing", res_x = 1, res_y = 1)
p2 <- heatmap_plotter(heatmaps_data = clean_heatmaps_5_3, matchid = 8650, playerid = 3621,
                      pos = "Left Wing", res_x = 3, res_y = 3)
p3 <- heatmap_plotter(heatmaps_data = clean_heatmaps_5_6, matchid = 8650, playerid = 3621,
                      pos = "Left Wing", res_x = 6, res_y = 6)
p4 <- heatmap_plotter(heatmaps_data = clean_heatmaps_3_3, matchid = 8650, playerid = 3621,
                      pos = "Left Wing", res_x = 3, res_y = 3)
p5 <- heatmap_plotter(heatmaps_data = clean_heatmaps_5_3, matchid = 8650, playerid = 3621,
                      pos = "Left Wing", res_x = 3, res_y = 3)
p6 <- heatmap_plotter(heatmaps_data = clean_heatmaps_10_3, matchid = 8650, playerid = 3621,
                      pos = "Left Wing", res_x = 3, res_y = 3)
```

We also created a Shiny App to enable users to explore these heat maps without having to write any lines of code. Here, heat map visualizations can be created seamlessly; the user can specify their desired bandwidth and resolution via sliders, and have access to additional map features such as a football pitch overlay, points at the observed event locations, a positional grid (akin to that shown in Figure 1), and a variety of alternate color palettes.

Figure 3 showcases the same six heat maps visible in Figure 2. Other than the fact that our Shiny App sets very small values to `NA` (yielding heat map images that are more akin to those we see in the wild), it is evident that these maps correspond to the same underlying values.

We invite the reader to visit this Shiny App, affectionately called the ***Heat Map Sandbox***[8], to obtain an even deeper familiarity with the data.

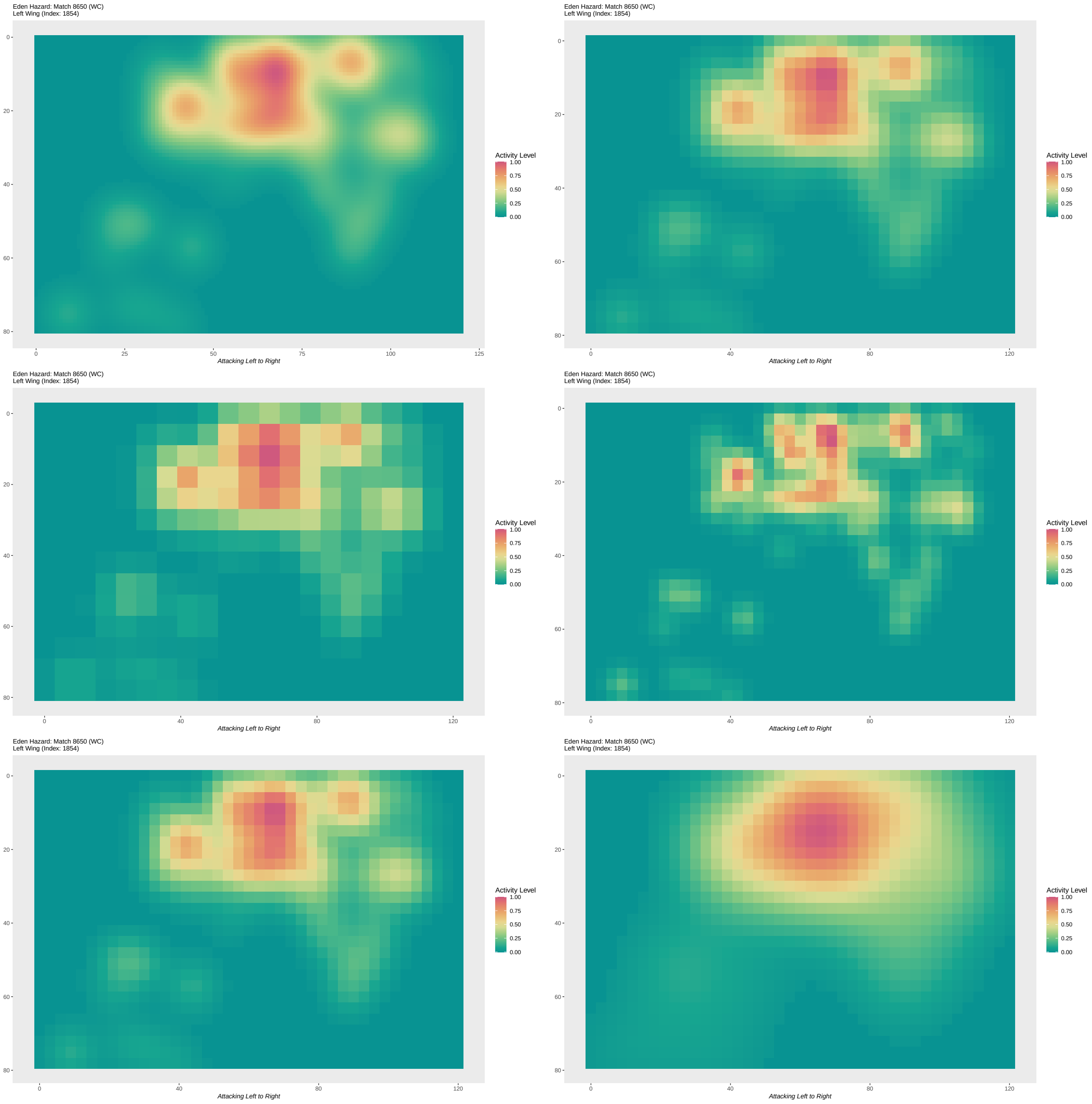


Figure 2: Eden Hazard at Left Wing in Match 8650

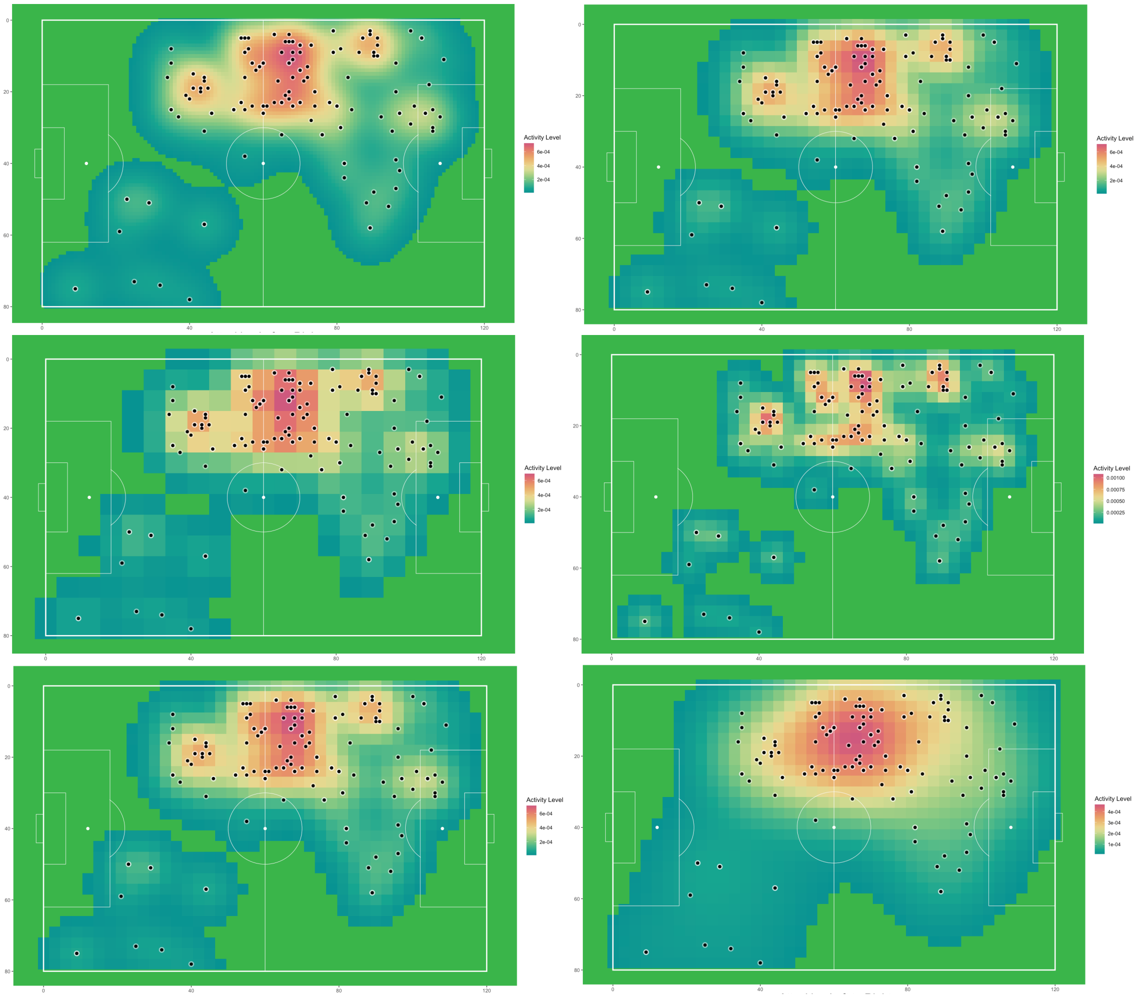


Figure 3: Eden Hazard at Left Wing in Match 8650 (Using Shiny App)

Model Overview and Selection

Training and Test Sets

The following is a list of every nation that appeared in either the 2018 FIFA World Cup or the 2020 UEFA European Football Championship:

```
 [1] "Russia"          "Saudi Arabia"   "Croatia"        "Nigeria"
 [5] "France"          "Australia"      "Argentina"      "Iceland"
 [9] "Peru"            "Denmark"        "Brazil"         "Switzerland"
[13] "Germany"         "Mexico"         "Costa Rica"     "Serbia"
[17] "Belgium"         "Panama"         "Tunisia"        "England"
[21] "Sweden"          "South Korea"    "Poland"         "Senegal"
[25] "Egypt"           "Colombia"       "Japan"          "Portugal"
[29] "Morocco"         "Iran"           "Spain"          "Uruguay"
[33] "Turkey"          "Italy"          "Finland"        "Wales"
[37] "Netherlands"     "Ukraine"        "Austria"        "North Macedonia"
[41] "Scotland"        "Czech Republic" "Slovakia"       "Hungary"
```

Of these 44 teams, only 12 appeared in both competitions:

```
 [1] "Russia"      "Croatia"     "France"      "Denmark"     "Switzerland"
 [6] "Germany"     "Belgium"     "England"     "Sweden"      "Poland"
[11] "Portugal"    "Spain"
```

These teams are particularly interesting to us, as we possess at least six-matches worth – three *Group Stage* matches per competition – of heat map data for each. We accordingly define our test set with heat maps of players from the following four nations (from this candidate set of 12):

```
[1] "Belgium" "England" "France"  "Croatia"
```

Belgium and England are the only two nations who made it beyond the *Round of 16* in both competitions, with the two squaring-off in the *3rd Place Final* of the World Cup as well. Meanwhile, France and Croatia both fell out of the Euros in the *Round of 16*, but it was these two teams who met in the *Final* of the World Cup. Limiting our focus to these four nations enables us to use a sizable portion of our 3809 total heat maps for training:

```r
allplayers_train <- allplayers %>%
  rownames_to_column("index") %>%
  filter(team %in% train_teams)
train_indices <- as.integer(allplayers_train$index)
allplayers_test <- allplayers %>%
  rownames_to_column("index") %>%
  filter(team %in% test_teams)
test_indices <- as.integer(allplayers_test$index)
```

```
[1] "Our training set has 3042 observations, which is approximately 79.9% of our total observations."
```

```
[1] "Our test set has 767 observations, which is approximately 20.1% of our total observations."
```

Essentially, we've manufactured an 80/20 training-test split that enables us to later assess our model's predictions from a team perspective as well as from an individual position perspective.

Of the sets of heat maps that we crafted above, we choose to only consider those with resolution parameters `res_x, res_y = 3`. This decision is justified in detail later.

```
dim(clean_heatmaps_3_3$heatmaps)
```

```
[1] 3809   41   27
```

Our heat maps are currently stored as arrays of dimension `(41, 27)`; as mentioned above, these can be equivalently thought of as arrays of dimension `(41, 27, 1)`. In fact, our CNN model requires 3-dimensional input, so we must undergo another stage of processing: as we partition each of `clean_heatmaps_3_3$heatmaps`, `clean_heatmaps_5_3$heatmaps`, and `clean_heatmaps_10_3$heatmaps` into the training and test sets defined above, we add an extra dimension to each.

```
dim(train_heats_3_3)
```

```
[1] 3042   41   27    1
```

```
dim(test_heats_3_3)
```

```
[1] 767  41  27   1
```

Since these sets of heat maps were constructed with respect to the original order of the rows of `allplayers`, the vectors of position labels for `clean_heatmaps_3_3`, `clean_heatmaps_5_3`, and `clean_heatmaps_10_3` are all identical. This allows us to define our training and test labels `train_positions` and `test_positions` once, and use these universally. Our CNN, which we define in the next section, is implemented using the `keras_model_sequential()` function from the `keras` package. Along with demanding a 3-dimensional input, this model forces us to format our response values as (zero-indexed) integers or one-hot vectors. We elect to use integers; the positions are encoded as follows:

```
positions <- as.integer(clean_heatmaps_3_3$class) - 1
```

0	1	2	3	4	5
Attacking Midfield	Center Back	Center Forward	Center Midfield	Defensive Midfield	Goalkeeper

6	7	8	9	10	11
Left Back	Left Forward	Left Midfield	Right Back	Right Forward	Right Midfield

```
train_positions <- positions[train_indices]
test_positions <- positions[test_indices]
```

Convolutional Neural Network Structure

The structure of the Convolutional Neural Network we will be using is outlined below:

```
CNN_final <- keras_model_sequential()
```

```
CNN_final %>%
  layer_conv_2d(filters = 128,
                kernel_size = c(4, 4),
                activation = 'relu',
                input_shape = c(n_row, n_col, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2),
                       padding = "same") %>%
  layer_dropout(rate = 0.2) %>%
  layer_conv_2d(filters = 32,
                kernel_size = c(4, 4),
                activation = 'relu',
                input_shape = c(n_row, n_col, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2),
                       padding = "same") %>%
  layer_dropout(rate = 0.2) %>%
  layer_flatten() %>%
  layer_dense(units = 32,
              activation = "relu") %>%
  layer_dense(units = 12,
              activation = "softmax")
summary(CNN_final)
```

```
Model: "sequential"
________________________________________________________________________________
 Layer (type)                       Output Shape                    Param #
================================================================================
 conv2d_1 (Conv2D)                  (None, 38, 24, 128)             2176

 max_pooling2d_1 (MaxPooling2D)     (None, 19, 12, 128)             0

 dropout_1 (Dropout)                (None, 19, 12, 128)             0

 conv2d (Conv2D)                    (None, 16, 9, 32)               65568

 max_pooling2d (MaxPooling2D)       (None, 8, 5, 32)                0

 dropout (Dropout)                  (None, 8, 5, 32)                0

 flatten (Flatten)                  (None, 1280)                    0

 dense_1 (Dense)                    (None, 32)                      40992

 dense (Dense)                      (None, 12)                      396

================================================================================
Total params: 109,132
Trainable params: 109,132
```

```
Non-trainable params: 0
--------------------------------------------------------------------------------
```

Taking input of dimension `(41, 27, 1)` (an image of size 41 x 27 with 1 channel) we begin with a convolutional layer comprised of 128 *filters* (also referred to as *kernels*) of dimension `(4, 4, 1)` (128 single-channel 4 x 4 arrays of parameters).

Consider a heat map image of size `(41, 27)` with `c = 1` channel. Starting at the "top-left" of this image – the first `(4, 4)` sub-section of this image – a kernel of parameters of size `(4, 4)` is multiplied component-wise with this sub-section of the image. These products are summed together.

It's important to recognize that this is done for all `c` channels; if `c > 1`, then we'd be doing this component-wise multiplication in each channel and adding all of the products up across the channels.

We take this sum, add a *bias* parameter, and put it through the *Rectified Linear Unit* (relu) activation function

$$f(x) = \max(0, x)$$

The result is the value that goes in the $(1,1)^{\text{th}}$ entry of the output array. Moving over one column in the input image, the next `(4, 4)` sub-section is considered, and the same procedure is done to obtain the value that is set as the $(1,2)^{\text{th}}$ entry of the output array.

Accordingly, the kernel is swept over the entire heat map image, yielding an output of size (41 - 4 + 1, 27 - 4 + 1) = `(38, 24)`.

This entire process is done for each of our 128 kernels, meaning we get 128 such outputs, called *feature maps*. It's hopefully clear to see, then, that what is yielded by this first convolutional layer is an output of dimension `(38, 24, 128)` (an image of size 38 x 24 with 128 channels).

The total number of parameters defined by the model in this layer is:

$$((4 * 4 * 1) + 1) * 128 = 17 * 128 = 2176$$

The subsequent *max-pooling* layer partitions its input into non-overlapping sections of size 2 x 2, finds the maximum value in each of these pieces of the image, and returns these values, essentially reducing the dimensionality of our object. To guard against overfitting, the following *dropout* layer zeros-out each entry of its input with probability *0.2*.

The next three layers are identical, except the second convolutional layer has 32 filters. Convolutional layers are meant to extract the features of an image, so having a second convolutional layer allows us to learn features of features. The total number of parameters defined by the model in this layer is:

$$((4 * 4 * 128) + 1) * 32 = 2049 * 32 = 65568$$

Finally, after a *flatten* layer converts its 3-dimensional input into a 1-dimensional output vector, a *dense* layer (akin to what we'd see in a standard neural network) with 32 neurons feeds into our final layer, which has 12 neurons – one corresponding to each of our positions. The number of parameters defined by the model in these two layers are:

$$1280 * 32 + 32 = 40992$$

$$32 * 12 + 12 = 396$$

The 12 terminal neurons comprise an output vector $\mathbf{z}$ of length 12, where the *softmax* of each component is taken to obtain the vector $\mathbf{p}$:

$$p_k = \frac{e^{z_k}}{\sum_{j=0}^{11} e^{z_j}}$$

The CNN's prediction $\hat{y}$ for the true position y corresponding to an inputted heat map is:

$$\hat{y} = \text{argmax}(\mathbf{p}) = \text{argmax}(p_0, \cdots, p_{11})$$

The 109,132 parameters of this model are tuned through the use of *batch Stochastic Gradient Descent* (SGD) according to the *categorical cross-entropy* (equivalently, negative *log-likelihood*) loss function. For a single inputted heat map (observation i), this loss is defined as:

$$l_i(\theta) = -\sum_{k=0}^{11} \mathbb{I}_{y_i=k} \log p_{i,k}$$

```
CNN_final %>%
  compile(
    loss = "sparse_categorical_crossentropy",
    optimizer = "adam",
    metrics = "accuracy")
```

The decision to settle upon this particular CNN structure beforehand was one that was driven by both a concern for computational cost and a lack of concern for the possibility of overfitting. For starters, these models take a very long time to train, particularly if there are hundreds of thousands of model parameters. Of course, depending on the resolution of our heat map inputs, the CNN's structure will change – the more pixels in a heat map, the more parameters our model must estimate. This reality made us refrain from building models using the heat maps with `res_x, res_y = 1`; for the interpretative ease of thinking in terms of just one model structure, we also shed our `res_x, res_y = 6` heat maps. Thus, we just use our heat maps created using `res_x, res_y = 3`, and tailor our model to handle inputs of the size of just these particular heat maps.

For each CNN (with this given structure) that we train, we implement an "early stopping" condition that circumvents overfitting. A fifth of the training data that is given to the `keras` function `fit()` – which we will see later on – is actually set aside as a validation set. Using the `epochs` parameter, we specify how many episodes, or *epochs*, of training we desire our model to undergo. An epoch is defined as one run-through of the training set; in each epoch, groups of the training heat maps of size `batch` are taken, and for each batch, gradient descent updates to our parameters are calculated using *backpropogation*. At the end of each epoch, predictions are made for validation set, and the loss is calculated and tracked.

Once this validation loss begins to increase, our model recognizes the threat of overfitting; exhibiting a `patience` of 50 epochs, the model ceases training, even if the pre-specified number of epochs has not yet been reached. With such a stopping condition, regardless of the complexity of our model, we can ensure that overfitting is reasonably restrained. Therefore, we feel comfortable with our choice to move forward with a CNN that has many model parameters, but not enough to make training too costly.

Cross Validation

While we do not execute any sort of model selection for different CNN structures, we do indeed find ourselves with three sets of `res_x, res_y = 3` heat maps: heat maps generated using a bandwidth `h = 3`, `h = 5` and `h = 10`. Training our CNN on these different types of heat maps may or may not result in differing predictive performances on our Belgian, Croatian, English and French players' heat maps.

We perform *5-fold Cross Validation* for each bandwidth level under consideration. Please see the accompanying R Markdown to assess our implementation; training 15 CNNs takes quite a while, so we provide each of these models, as well as the training and validation sets that correspond to them, in the `model` and `data` folders available on our GitHub repository[9].

```
mean(SCCE_3_3)
```

```
[1] 2.20643
```

```
mean(SCCE_5_3)
```

```
[1] 2.112924
```

```
mean(SCCE_10_3)
```

```
[1] 1.406507
```

According to our cross-validated approximations of the test loss, **the bandwidth that yields the lowest loss is `h = 10`**. We proceed with training and testing our final model using these heat maps.

Results

Model Performance

Figure 4 illustrates the early stopping mechanism in action; our choice to exercise a `patience` of 50 epochs results in a slight uptick in our validation loss (and corresponding decline in our validation accuracy), but this is not too concerning to us. It doesn't take long – only 25 epochs or so – for the improvement in our validation loss to stagnate and subsequently show signs of increasing.

```r
CNN_final %>% fit(train_heats_10_3, train_positions, epochs = 300,
                  verbose = 0, batch_size = 300, validation_split = 0.2,
                  callbacks = list(callback_early_stopping(min_delta = 0, patience = 50)))
```

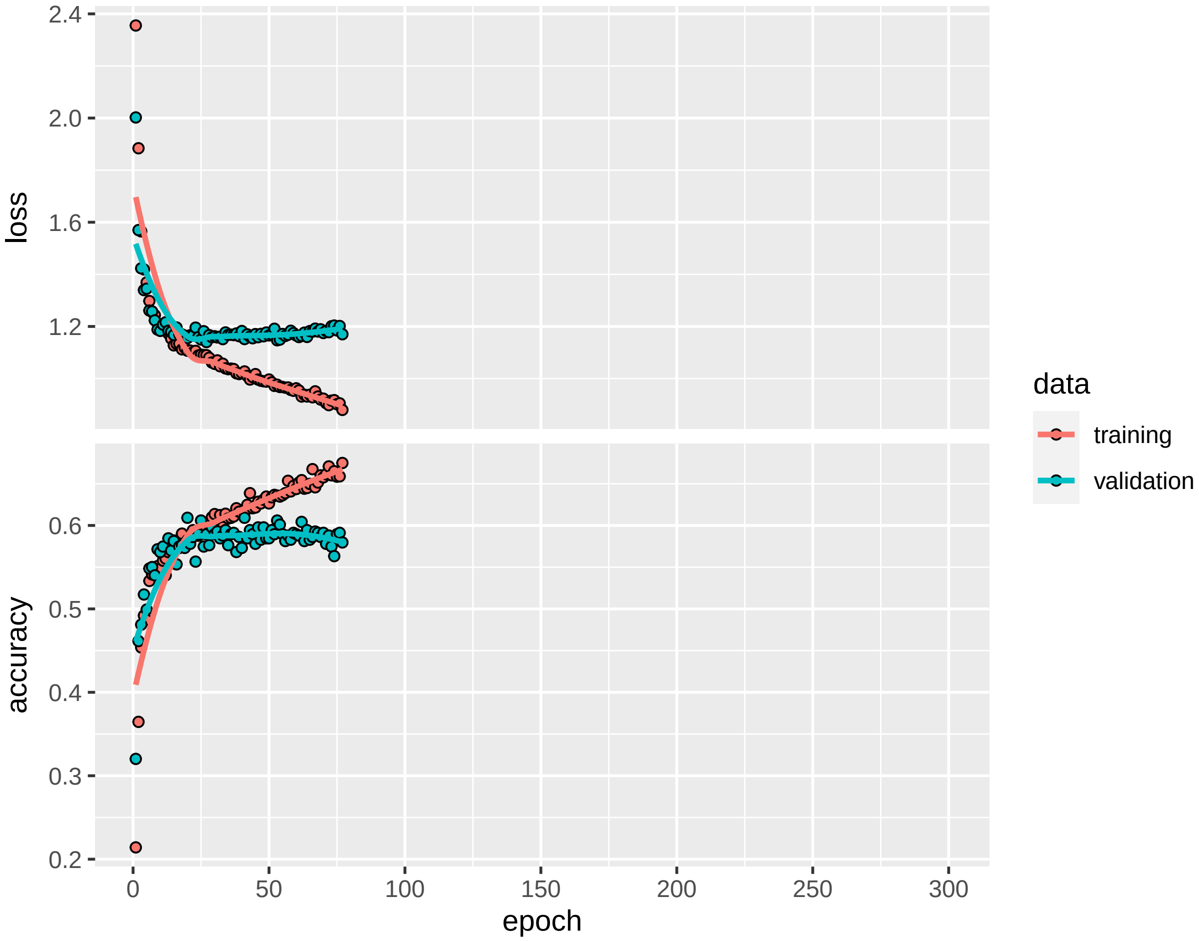


Figure 4: Training and Validation Loss (Top), Training and Validation Accuracy (Bottom) by Epoch

Test Accuracy

Based on the validation accuracy, it appears as if this model correctly predicts the position corresponding to a heat map it hasn't seen at a rate of just under 0.60. Let's see if this proxy for the test accuracy holds for the heat maps of Belgium, Croatia, England and France:

```
predictions <- CNN_final %>% predict(test_heats_10_3)
```

```
position_predictions <- apply(X = predictions, MARGIN = 1, FUN = "which.max") - 1
sum(test_positions == position_predictions) / length(test_positions)
```

```
[1] 0.5645372
```

As we expected, our test prediction accuracy is approximately 0.565. This can also be calculated using the built-in `keras` function `evaluate()`:

```
results <- CNN_final %>% evaluate(test_heats_10_3, test_positions)
results
```

```
     loss  accuracy
1.2335907 0.5645372
```

Teams

The code below augments `allplayers_test` to include these predicted positions, for ease of comparison. We show a subset of its columns to convey what this resulting data frame looks like.

playerID	playerName	matchID	team	position	label	predicted_pos	predicted_label	correct
3444	Danijel Subašić	7529	Croatia	Goalkeeper	5	Goalkeeper	5	TRUE
3471	Dejan Lovren	7529	Croatia	Right Center Back	1	Center Back	1	TRUE
3471	Dejan Lovren	7529	Croatia	Left Center Back	1	Center Back	1	TRUE
5456	Mateo Kovačić	7529	Croatia	Right Wing	10	Right Forward	10	TRUE
5457	Ivan Strinić	7529	Croatia	Left Back	6	Left Back	6	TRUE
5459	Marko Pjaca	7529	Croatia	Center Forward	2	Attacking Midfield	0	FALSE
5460	Andrej Kramarić	7529	Croatia	Center Attacking Midfield	0	Center Midfield	3	FALSE
5461	Šime Vrsaljko	7529	Croatia	Right Back	9	Right Back	9	TRUE
5463	Luka Modrić	7529	Croatia	Right Center Midfield	3	Defensive Midfield	4	FALSE
5463	Luka Modrić	7529	Croatia	Center Attacking Midfield	0	Attacking Midfield	0	TRUE

Examining the table below, there does not appear to be any clear trend between the predictability of the heat maps for each teams' players and their respective performances at the World Cup and Euros. For both competitions, our model correctly predicted the Belgian heat maps approximately 50% of the time, yet the Belgians made it much deeper in the World Cup than the Euros. The positional behaviors of the Croatians (-0.094) and the English (-0.102) appear to be noticeably less predictable in the World Cup than the Euros, however the Croatians were more successful in the World Cup, while the English made it further in the Euros. France was the only team that our model predicted correctly at a rate larger than 0.60 for both competitions – they were more predictable in the World Cup, yet they were the winners of this competition.

team	comp	n	n_correct	p_correct	n_incorrect	p_incorrect	knocked_out
Belgium	EU	72	36	0.500	36	0.500	Quarter-finals
Belgium	WC	117	57	0.487	60	0.513	3rd Place Final
Croatia	EU	63	41	0.651	22	0.349	Round of 16
Croatia	WC	122	68	0.557	54	0.443	Final
England	EU	125	77	0.616	48	0.384	Final
England	WC	107	55	0.514	52	0.486	3rd Place Final
France	EU	61	37	0.607	24	0.393	Round of 16
France	WC	100	62	0.620	38	0.380	Final

```r
t.test(p_correct ~ comp, data = teams_results)
```

```
    Welch Two Sample t-test

data:  p_correct by comp
t = 1.1234, df = 5.9206, p-value = 0.3047
alternative hypothesis: true difference in means between group EU and group WC is not equal to 0
95 percent confidence interval:
 -0.05807652  0.15607652
sample estimates:
mean in group EU mean in group WC 
          0.5935           0.5445 
```

A two-sample t-test comparing the mean classification accuracies for the World Cup and the Euros yields a p-value of `0.3047`, which is larger than any reasonable significance threshold α, meaning we fail to reject the null hypothesis (which states that the true difference in their means is 0).

Goalkeeper

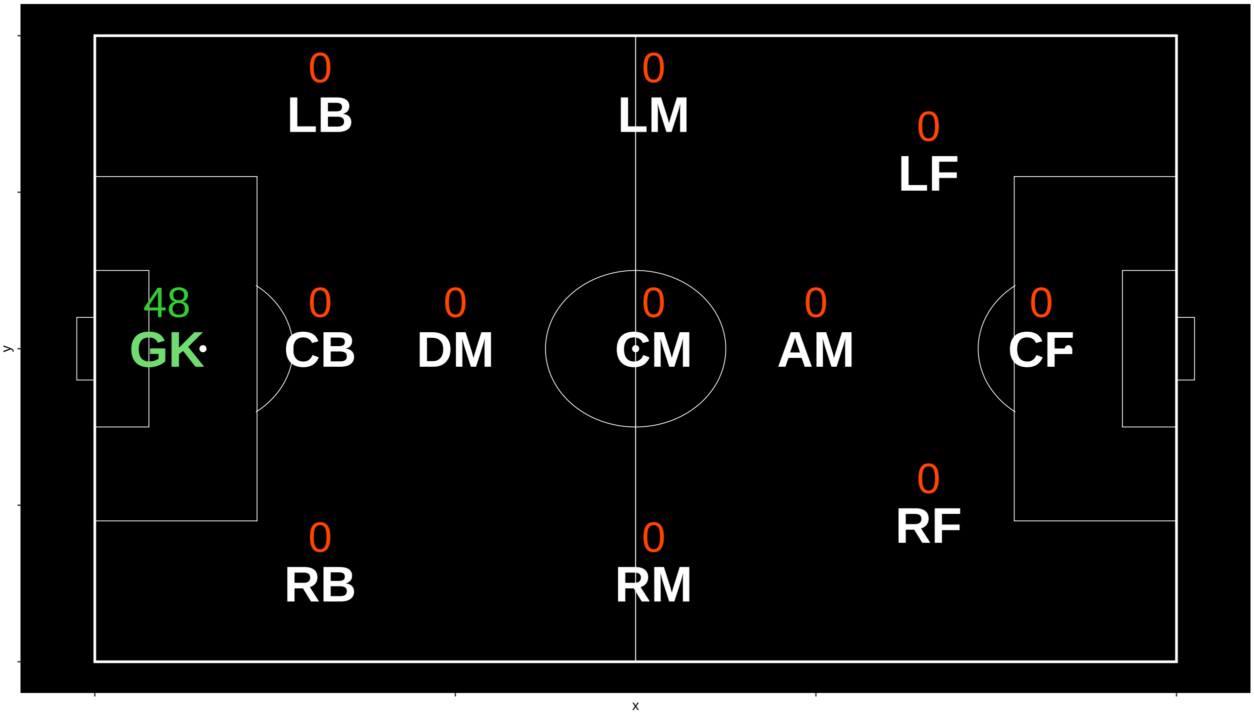


Figure 5: Goalkeeper Predictions

The CNN correctly predicted goalkeepers at a rate of 1. This is not surprising; other than the occasional touch higher up the field – particularly when the team in possession cycles the ball back to the keeper in order to reset their shape or relieve the pressure being put onto them by the opposition – goalkeepers sit in front of their team's goal, meaning the majority of their involvements occur deeper relative to any other player. This result serves as a comforting sanity check, as it demonstrates the ability of a CNN to learn the characteristics of a heat map pertaining to a particular position; when given (as input) a heat map where the vast majority of the activity occurred near the team's goal, our CNN recognizes that, with high probability, the player who this heat map corresponds to is a goalkeeper.

Left Back

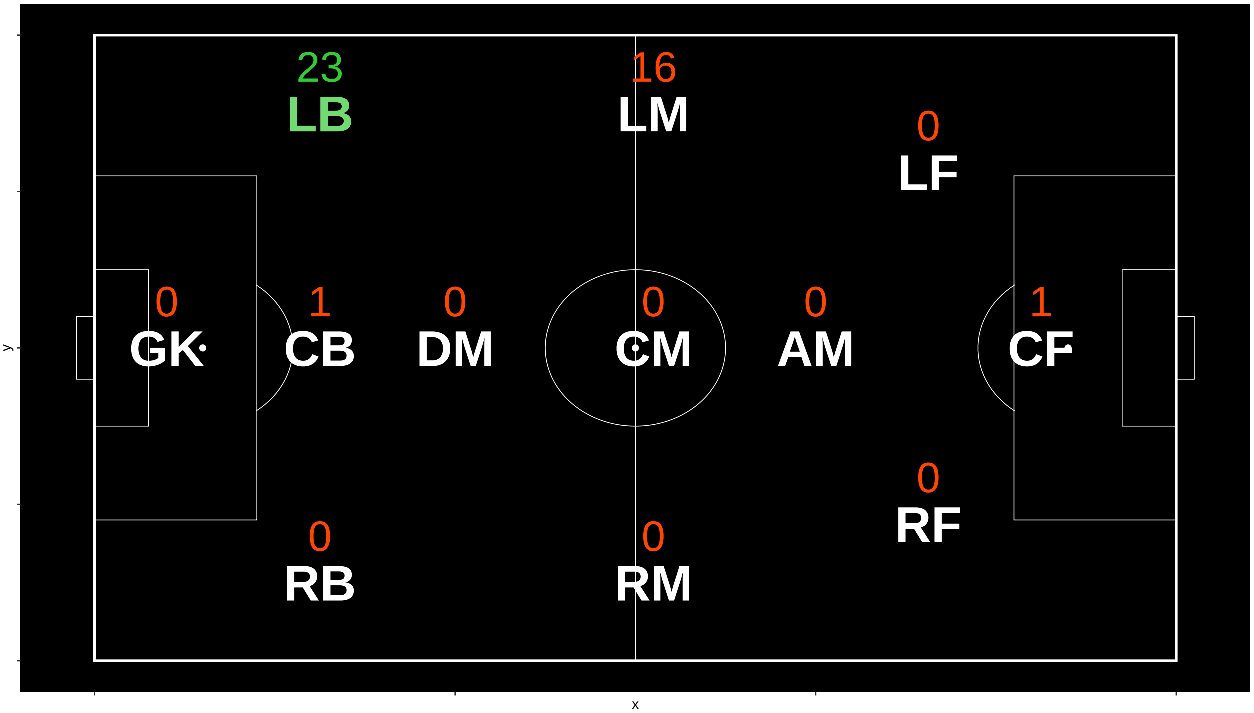
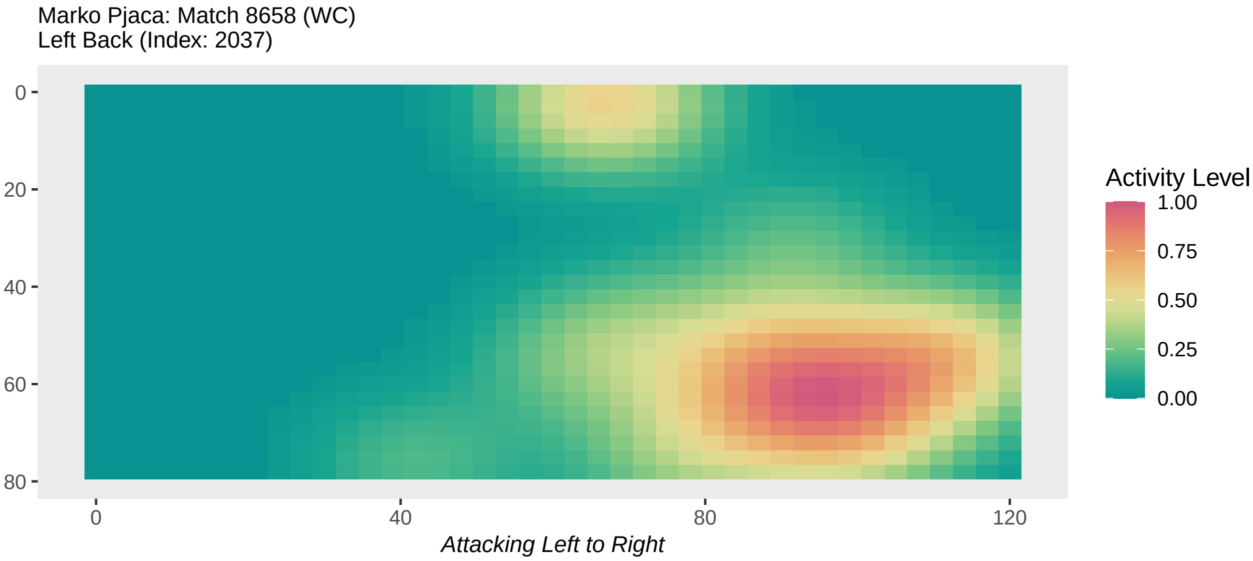


Figure 6: Left Back Predictions

The CNN correctly predicted left backs at a rate of 0.56. Understandably, the majority of the errors were classifications of *Left Midfield*. This makes sense, as many left backs tend to venture farther up the pitch to aid in the attack, and can thus easily be misinterpreted as midfielders. Incredibly, our CNN erroneously believed that one of our left backs was a *Center Forward*. This happened to be Croatian Marko Pjaca, who was subbed into left back in the 81st minute of the World Cup Final against France – a moment in which Croatia were trailing and desperately trying to score. Based on this misclassification, we can assume that Pjaca was – by instruction – ignoring his positional responsibilities in order to aid the attack.



Right Back

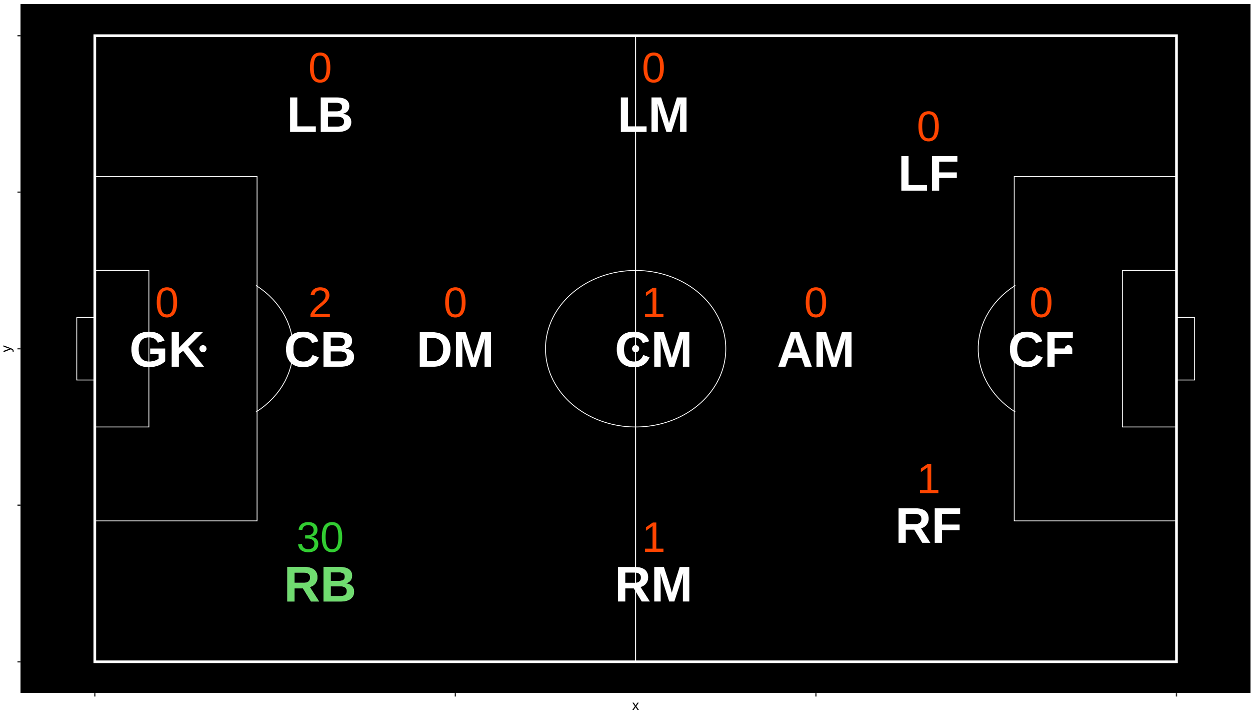
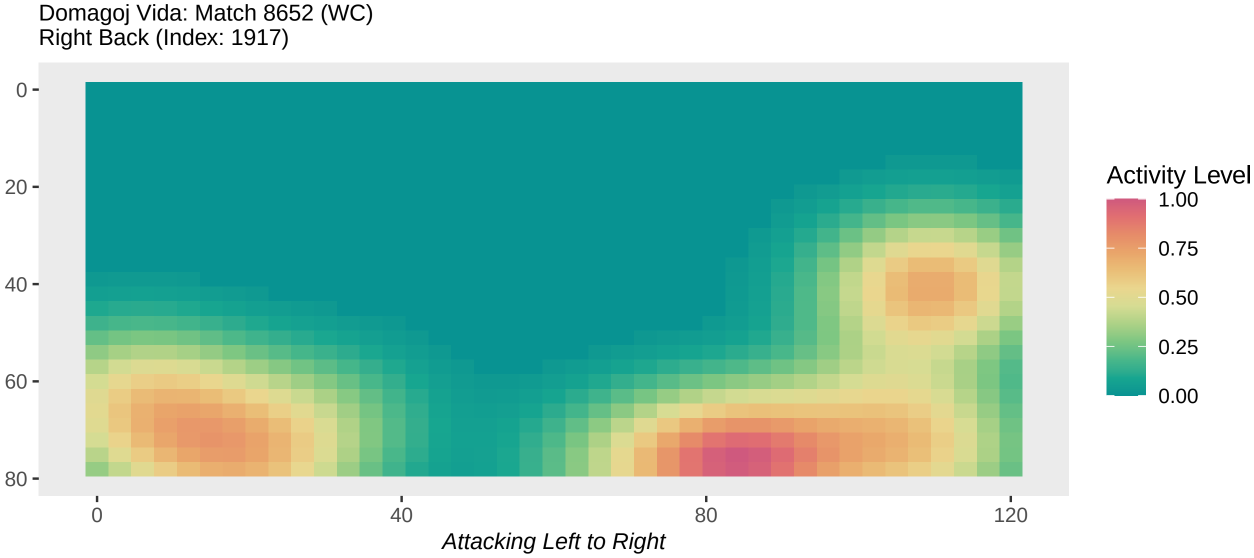


Figure 7: Right Back Predictions

The CNN correctly predicted right backs at a rate of 0.86, which is much better than that of their left-sided counterparts. One right back – another Croatian, Domagoj Vida – was mistaken as a *Right Forward*, implying that he must have been particularly offensive-minded in this performance.



Center Back

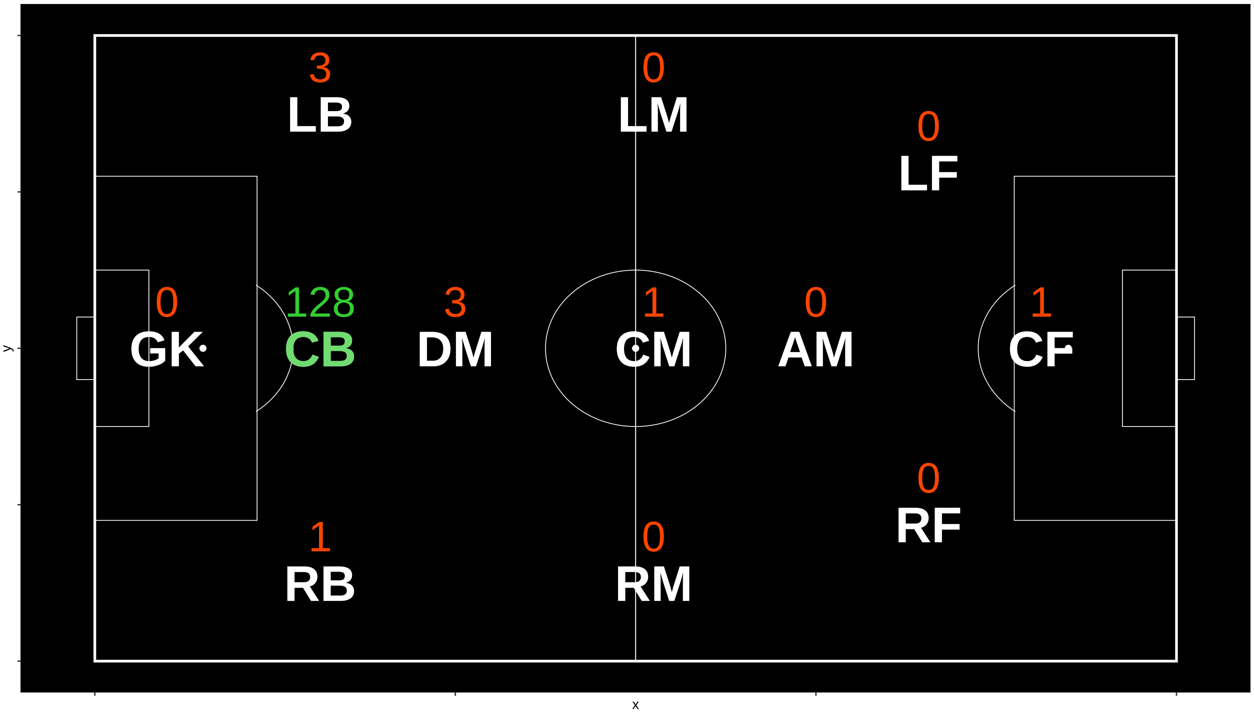
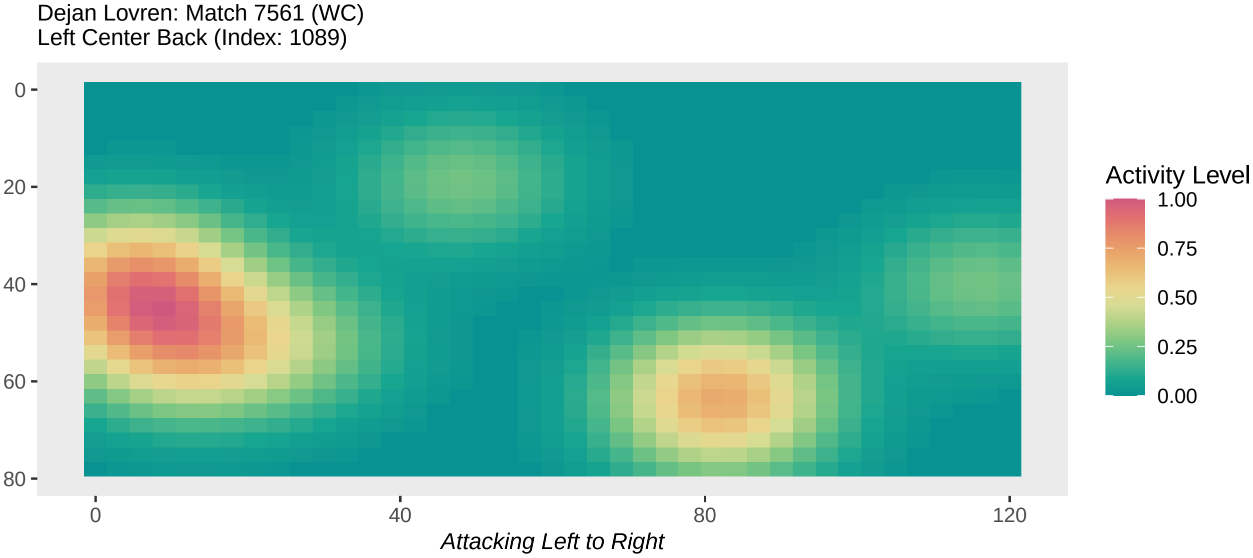


Figure 8: Center Back Predictions

The CNN correctly predicted center backs at a rate of 0.93 – 128 total correct classifications – which is nothing short of impressive considering the large quantity of center back heat maps in the test data. One was incorrectly identified as a *Center Forward* – yet another Croatian!



Definitely an odd error to make, considering the fact that Dejan Lovren's heat map doesn't look much a center forward's at all!

Defensive Midfield

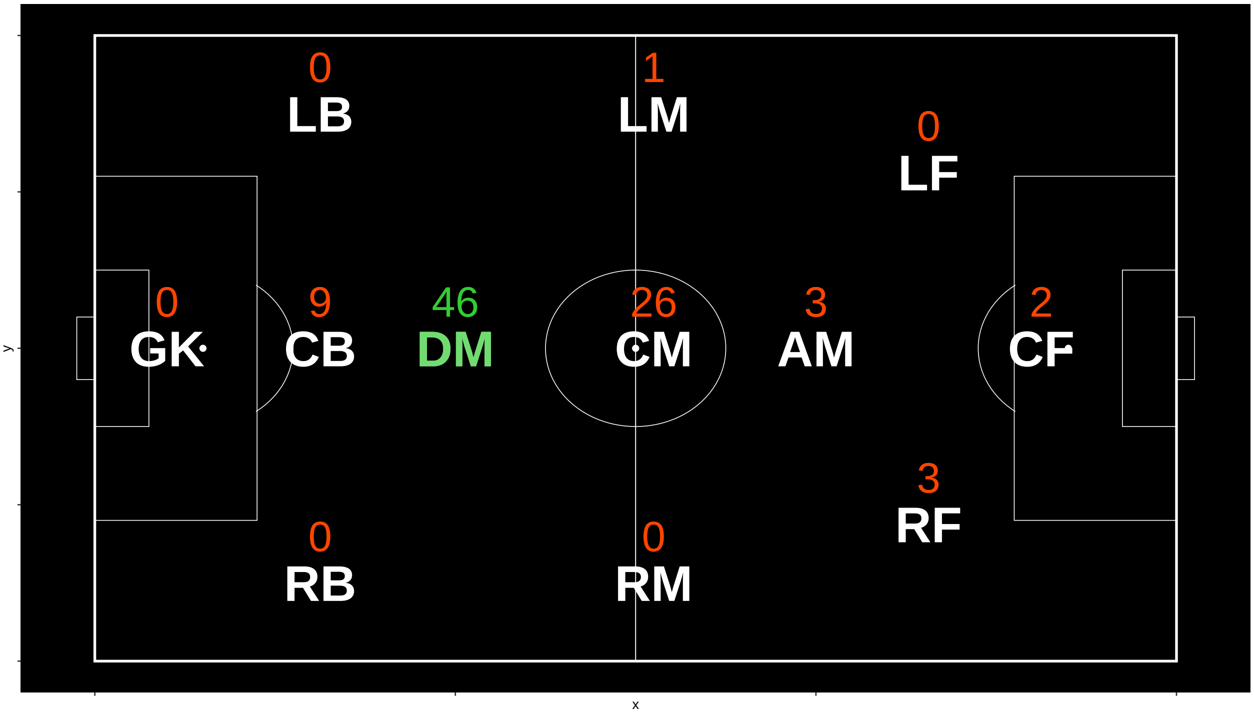


Figure 9: Defensive Midfield Predictions

The CNN correctly predicted defensive midfielders at a rate of 0.51. The majority of its mistakes were predictions of *Center Midfield* or *Center Back*, as we'd expect.

Center Midfield

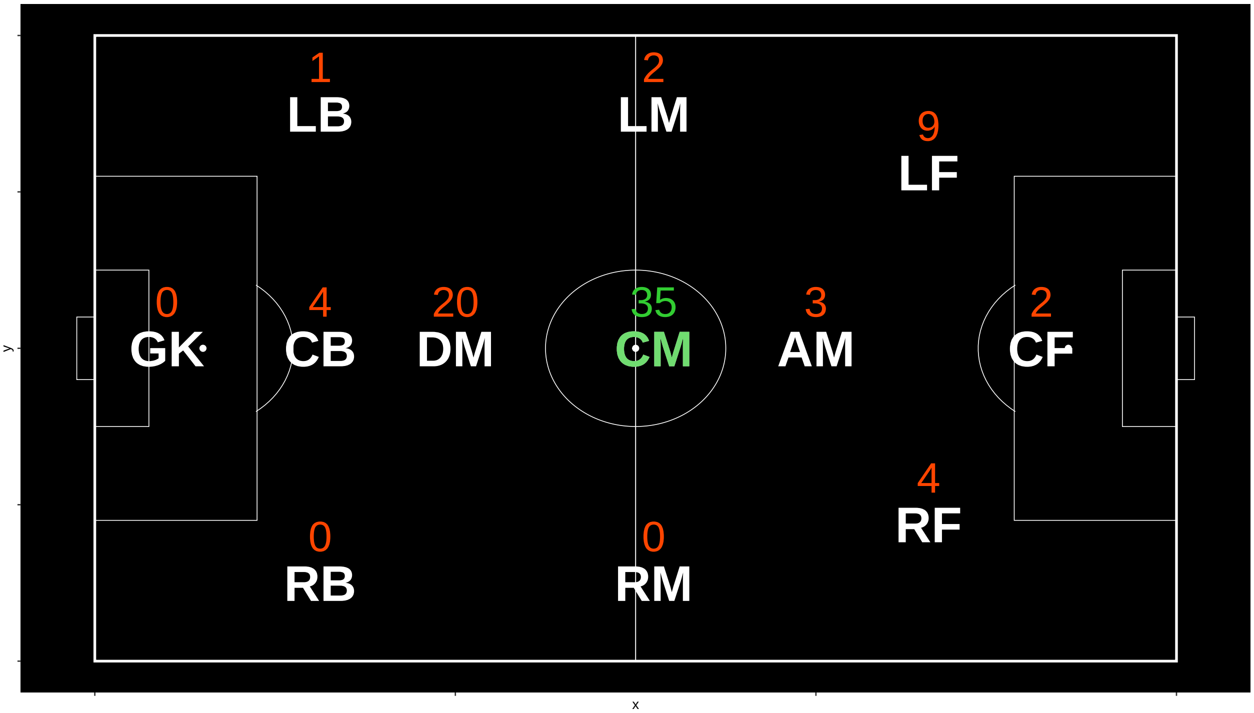


Figure 10: Center Midfield Predictions

The CNN correctly predicted center midfielders at a rate of 0.44. Traditionally serving as the nuclei of their teams, center midfielders tend to touch more unique blades of grass than any other position, as they are tasked with linking the defense and the attack, and thus find themselves all over the pitch. Thus, it is not a surprise that center midfielders were misclassified as almost every other position.

Attacking Midfield

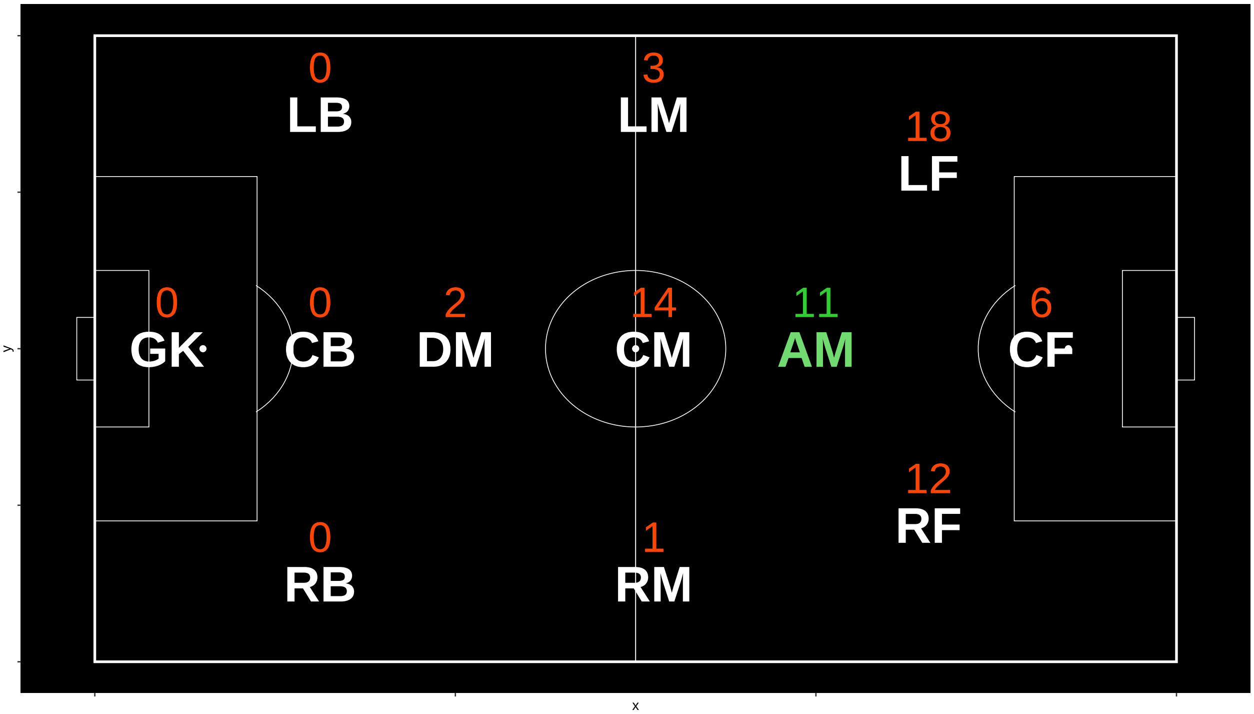


Figure 11: Attacking Midfield Predictions

The CNN correctly predicted attacking midfielders at a rate of 0.16 – by far the worst we've seen thus far. Attacking midfield is a very nuanced position in the sense that the position resides on the boundary of a team's middle third and attacking third – similar to the role of defensive midfielders in relation to the defensive third and the middle third. The predictive performance is much worse here, though, and the misclassifications themselves may give us some insight into why: attacking midfielders were incorrectly labeled as *Left Forwards*, *Right Forwards* and *Center Forwards* quite frequently. This suggests an intuitive fact about defending versus attacking: due to the nature of defending a goal that exists in the center of the pitch (with respect to the direction we've been referring to as Y, which is the width of the pitch), a defender meant to reside in the center of the pitch will not find themselves being involved on the left or right side of the pitch enough to be misclassified as a right-sided or left-sided player. However, further up the field, we do not necessarily expect an attacker meant to reside in the center of the pitch to obey this centrality, as attacks in football tend to require players to constantly be moving, making runs, and providing support for their teammates in unpredictable ways.

Left Midfield

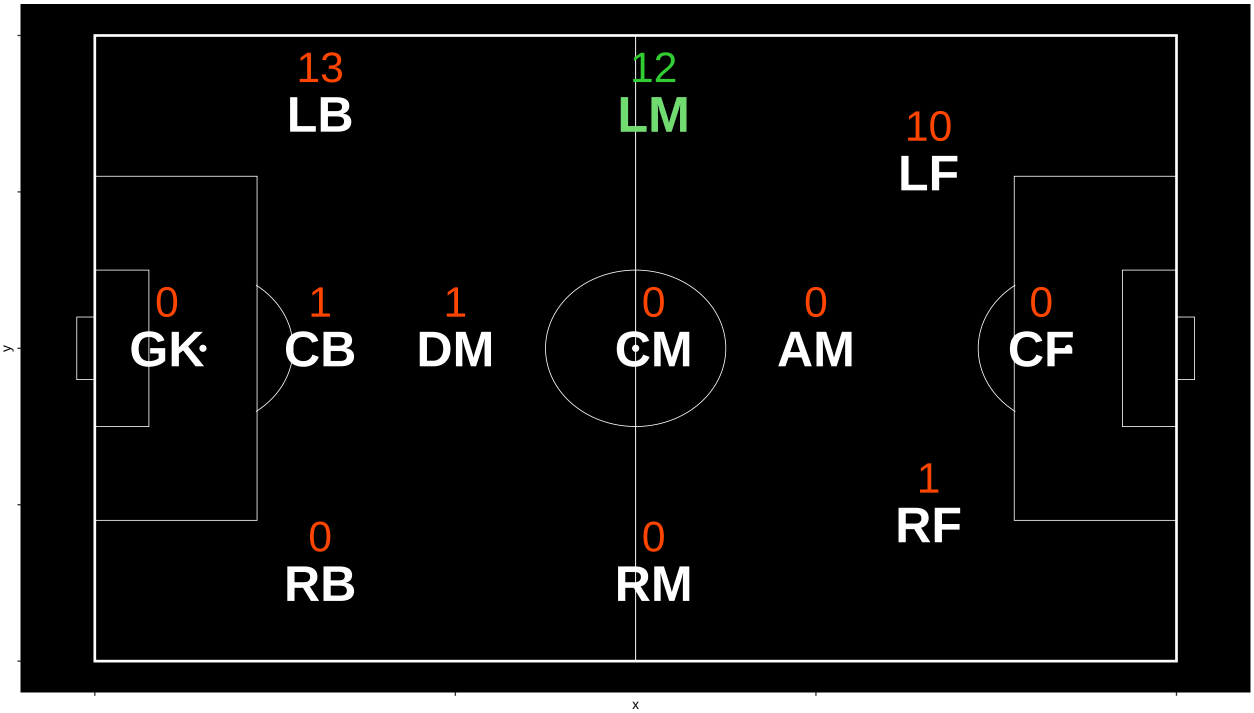
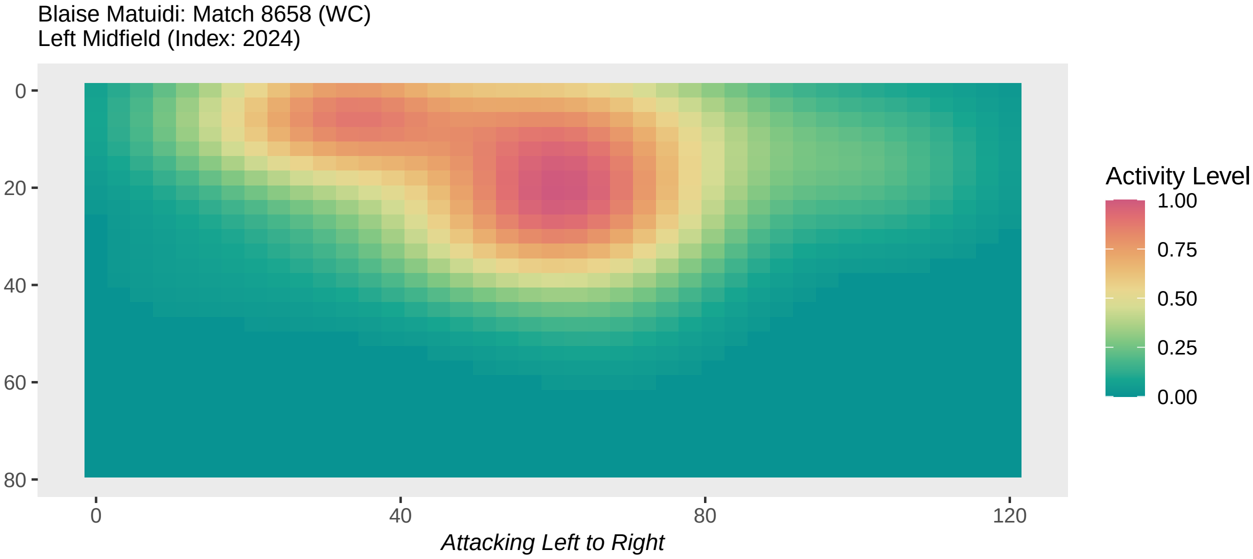


Figure 12: Left Midfield Predictions

The CNN correctly predicted left midfielders at a rate of 0.32. Frenchman Blaise Matuidi was the individual misclassified as a *Center Back*. This outing also happened to be the World Cup Final; Matuidi started the match and played over 70 minutes as a left midfielder, registering a total of 99 events. His heat map, shown below, does not look out-of-the-ordinary, though.



Right Midfield

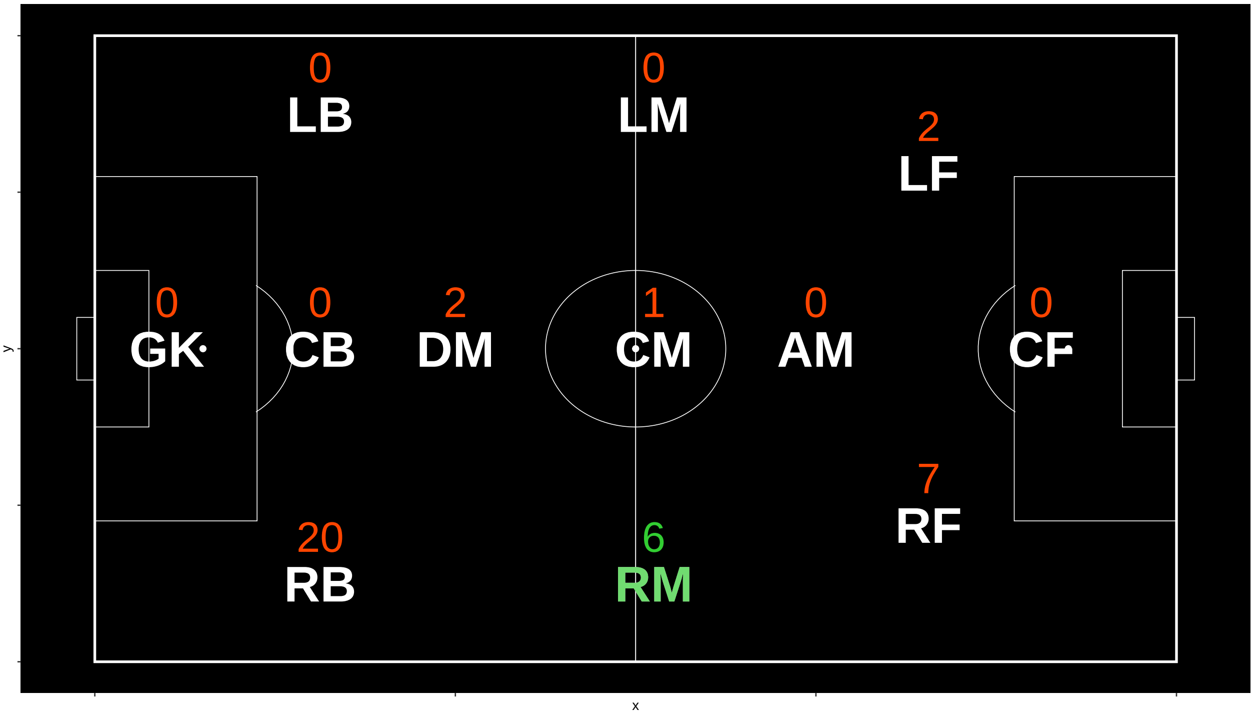


Figure 13: Right Midfield Predictions

The CNN correctly predicted right midfielders at a rate of 0.16. Like left midfield, the CNN was often able to understand that the heat map corresponded to a player playing on the right side of the pitch, but properly assigning *Right Midfield* with a larger probability than *Right Back* or *Right Forward* proved to be a challenge. Our decision to classify *Right Wing Back* as *Right Midfield* is probably why our error rate is so large, as the most frequent mistake was *Right Back*.

Forwards

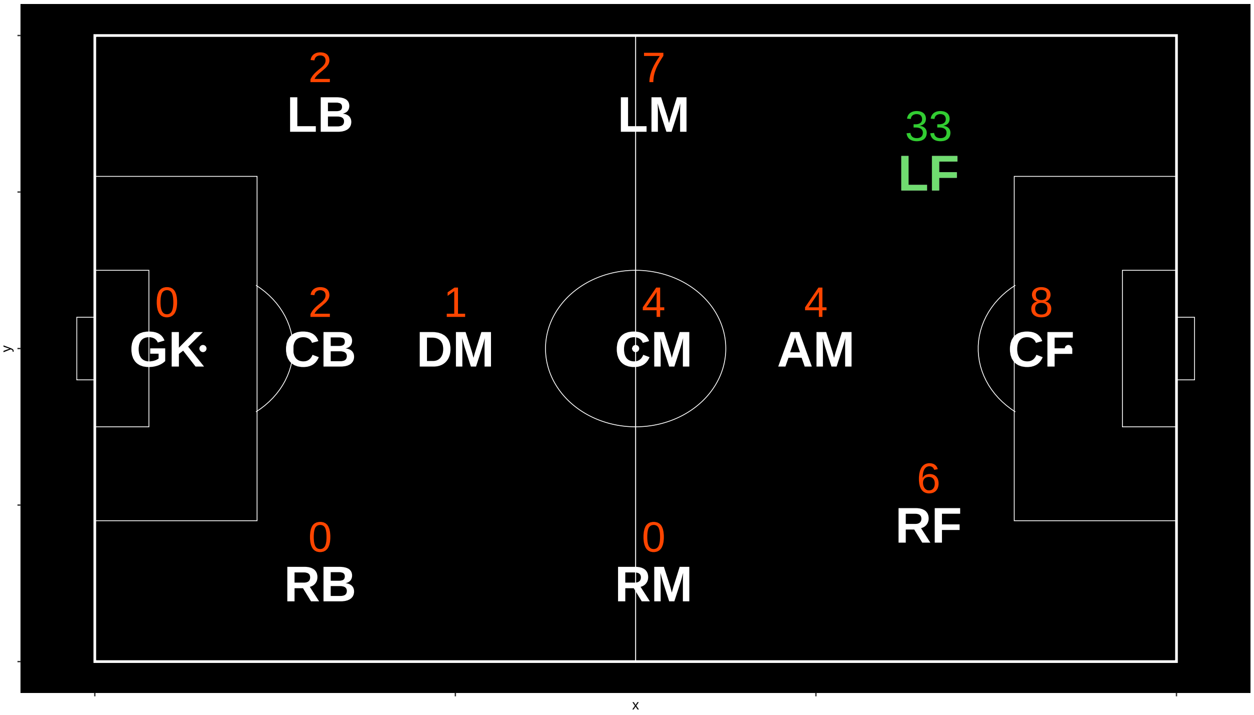


Figure 14: Left Forward Predictions

The CNN correctly predicted left forwards at a rate of 0.49, right forwards at a rate of 0.39, and center forwards at a rate of 0.61. These three positions were frequently misidentified as each other, with a surprising rate of misclassifications as *Center Midfielders*, *Defensive Midfielders*, and other deep-lying players. Perhaps these mistakes correspond to players who were playing a forward position during a moment in the game where they were asked to aid in defense; Belgium, Croatia, England, and France are all very successful teams, and in knockout competitions like the World Cup or Euros, it's not uncommon for teams with a lead to – in the effort of securing victory – sit deeper and get men behind the ball.

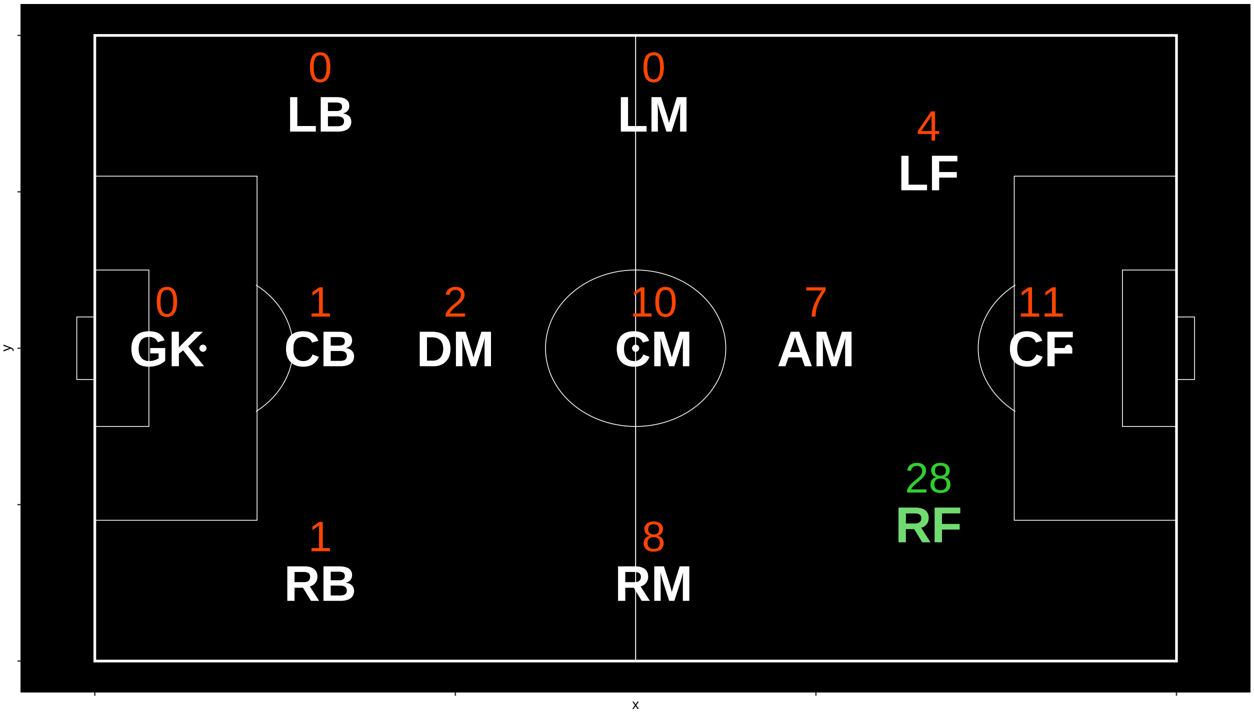


Figure 15: Right Forward Predictions

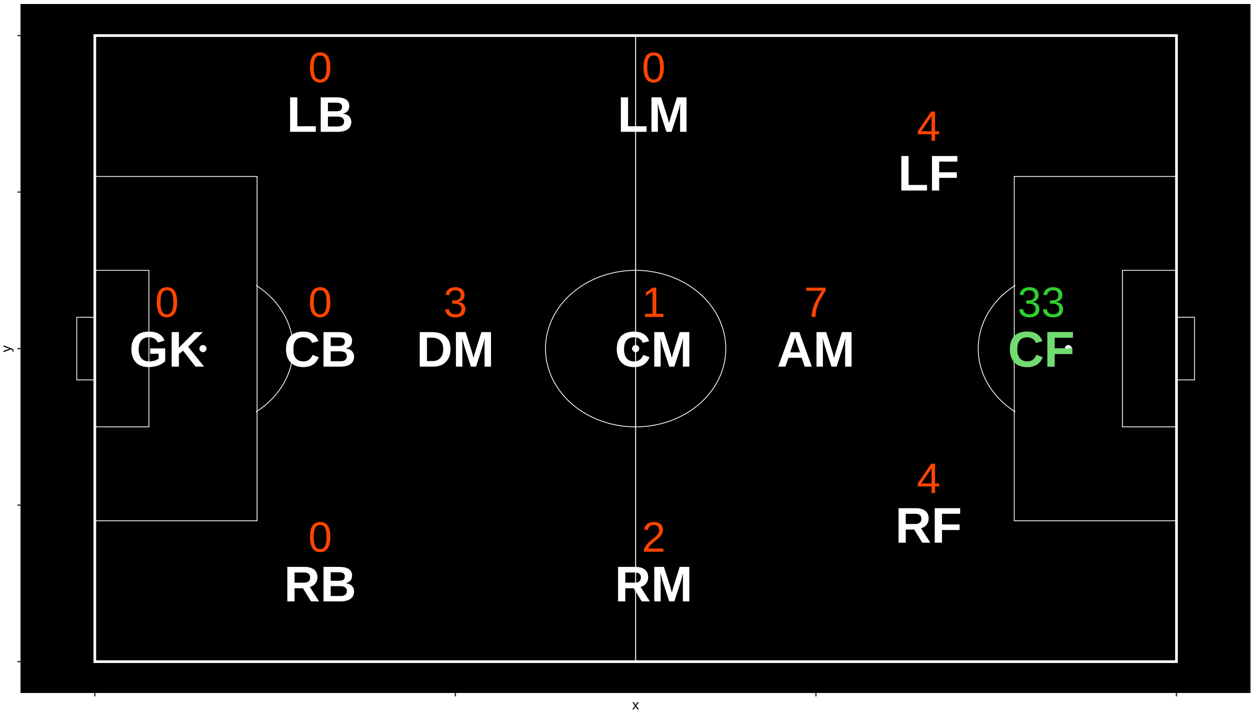


Figure 16: Center Forward Predictions

Conclusion

While we are very grateful to StatsBomb for giving the public access to this data, only having a total of 105 matches to work with obviously limited the potential of our heat map classifier; with more men's international football event data, we would be able to train our CNN on more heat maps, which would surely improve our model's understanding of the latent features of heat maps for different positions. With more heat maps, we could be even stricter with regards to the minimum number of events we require for a heat map to be included in our training and test sets; raising this lower bound would be useful if we desired to filter-out the instances of players who, after being subbed-in late in a match, are technically "playing" a position while being tasked with an entirely antithetical objective (i.e. Pjaca during the World Cup Final). The heat maps in this project were generated using every single event associated with the player of focus – the types of these events were not taken into account. It would be fascinating to see how these different events could be incorporated into our predictions of player positions, whether that be through factoring them into the creation of the heat maps themselves, or using them alongside the heat maps we've already created. Not to mention, there is the entire temporal aspect of a football match that has gone largely ignored throughout this project; perhaps time can be considered somewhere, as this would allow us to track the progression of a team's shape and tactics throughout a match via their players' heat maps.

Links

1) https://statsbomb.com/what-we-do/hub/free-data/

2) https://github.com/statsbomb/StatsBombR

3) https://markgjuraj.shinyapps.io/heatmaps/?_ga=2.2166960.1497053111.1651800795-618419299.1650734225

4) https://github.com/MGjuraj/HeatMapsCapstoneRepo

5) https://github.com/statsbomb/open-data/blob/master/doc/Open%20Data%20Matches%20v3.0.0.pdf

6) https://github.com/statsbomb/open-data/blob/master/doc/Open%20Data%20Lineups%20v2.0.0.pdf

7) https://github.com/statsbomb/open-data/blob/master/doc/Open%20Data%20Events%20v4.0.0.pdf

8) https://markgjuraj.shinyapps.io/heatmaps/?_ga=2.2166960.1497053111.1651800795-618419299.1650734225

9) https://github.com/MGjuraj/HeatMapsCapstoneRepo