UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Máster del
Máster Universitario en Ingeniería y Ciencia de Datos

Redes Neuronales Recurrentes para predicción de Posibilidad de Jugada de Gol

Fernando Martín Martín

Dirigido por: Mariano Rincón Zamorano

Rafael Pastor Vargas

Curso: 2022-2023: 1ª Convocatoria Septiembre

Resumen

Actualmente, en todos los estadios de fútbol de primera y segunda división de La Liga de Fútbol Profesional, existen unas cámaras ópticas que envían en tiempo real la posición (x,y) de cada jugador y de la pelota (x,y,z) 25 veces por segundo. A partir de esa información se calculan ciertas características asociadas a cada jugador (distancia a la portería, distancia al portero, ángulo a la portería, número de jugadores rivales en el cono de visión,..) y se ha entrenado un algoritmo (regresión logística) que utilizando dichas variables predice la probabilidad de marcar un gol por cada jugador en cada uno de los momentos registrados (25 por segundo). Este modelo se utiliza en la retransmisión de televisión de los partidos para ofrecer la probabilidad de marcar un gol en las repeticiones de los disparos o remates a puerta.

El actual sistema tiene algunas limitaciones como, por ejemplo, que no incorpora las jugadas a balón parado, no predice satisfactoriamente las jugadas de gol en las que sea relevante considerar la altura del balón o no contempla el contexto de la jugada previo al remate. Estas limitaciones están asociadas al diseño de las variables explicativas del modelo, las cuales se han construido a partir de la posición de los jugadores y del balón en el momento del remate a puerta y, por tanto, no tiene en cuenta la información temporal de lo acontecido en la jugada en los segundos previos al disparo.

En el presente Trabajo Fin de Máster, con el objetivo de contemplar las situaciones de juego que ahora están limitadas en el algoritmo actual, se ha diseñado y entrenado un modelo alternativo basado en Redes Neuronales Recurrentes (RNN) que incorpora la información de cómo trascurre la jugada en los 5 segundos anteriores al disparo a puerta para predecir la probabilidad de que un jugador marque gol.

Para utilizar la información temporal de la jugada previa a un remate a puerta, se han generado unas representaciones bidimensionales construidas a partir de los datos de la posición (coordenadas espaciales x,y) de cada jugador y de la pelota en los 125 instantes anteriores al disparo, las cuales conformarán las secuencias de elementos que alimentarán la Red Neuronal Recurrente; y para utilizar la información contenida en las relaciones espaciales de la posición de cada jugador y del balón, se ha empleado un Autoencoder Convolucional que ha permitido obtener de cada instante de juego su representación latente de baja dimensionalidad.

Con el objetivo de ir confirmando que las decisiones metodológicas tomadas son acertadas, se ejecutan diferentes experimentos en los que se prueban distintas aproximaciones de manera incremental, comprobando que los experimentos que combinan las Redes Neuronales Recurrentes con los Autoencoders Convolucionales son los que ofrecen los mejores valores en las métricas de evaluación del rendimiento; unos valores además similares a los obtenidos por el algoritmo actual que ha desarrollado La Liga de Fútbol Profesional, cuando ambos sistemas se han evaluado sobre un mismo conjunto de datos.

Abstract

Nowadays, in all the first and second division football stadiums of the Spanish Professional Football League, there are optical cameras that send the position (x,y) of each player and of the ball (x,y,z) in real time 25 times per second. Based on this information, certain characteristics associated with each player are calculated (distance to the goal, distance to the goalkeeper, angle to the goal, number of rival players in the cone of vision,...) and an algorithm has been trained (logistic regression) that, using these variables, predicts the probability of scoring a goal for each player in each of the moments recorded (25 per second). This model is used in the television broadcast of matches to offer the probability of scoring a goal in replays of shots on goal.

The current system has some limitations, such as, for example, it does not incorporate goal plays originated with the ball stopped, it does not satisfactorily predict goal plays in which it is relevant to consider the height of the ball, or it does not take into account the context of the play prior to the shot. These limitations are associated with the design of the explanatory variables of the model, which have been built based on the position of the players and the ball at the time of the shot on goal and, therefore, do not take into account the temporal information of what happened during the play in the seconds before the shot.

In this Master's Thesis, an alternative model based on Recurrent Neural Networks has been designed and trained, which incorporates the information on how the play takes place in the last 5 seconds before the shot on goal to predict the probability that a player will score a goal, with the aim of contemplating the game situations that are now limited in the current algorithm.

In order to use the temporal information of the play prior to a shot on goal, two-dimensional representations have been generated based on the positional data (spatial coordinates x,y) of each player and of the ball are generated in the 125 instants prior to the shot, which will make up the sequences of elements that will feed the Recurrent Neural Network; and to use the information contained in the spatial relations of the position of each player and the ball, a Convolutional Autoencoder has been used, which has allowed to obtain the low-dimensional latent representation of each instant of the play.

With the aim of confirming that the methodological decisions made are correct, different experiments are carried out in which different approaches are tested incrementally, verifying that the experiments that combine Recurrent Neural Networks with the Convolutional Autoencoders are those that offer the best values in the performance evaluation metrics; these values are also similar to those obtained on homogeneous data sets by the current algorithm developed by the Spanish Professional Football League.

Índice general

1. Introducción general y objetivos **3**

1.1. Planteamiento del problema, motivación y objetivos 4

1.1.1. Introducción al algoritmo "Goal Probability" 4

1.1.2. Limitaciones del modelo "Goal Probability" 6

1.2. Motivación y Objetivos 7

1.3. Estructura de la Memoria 8

2. Fundamento teórico y estado del arte. **9**

2.1. Estado del arte del cálculo de probabilidad de gol 9

2.2. Fundamento teórico 11

2.2.1. Aprendizaje profundo 11

2.2.2. Redes Neuronales Recurrentes 13

2.2.3. Redes Neuronales Convolucionales 18

2.2.4. Combinación de CNNs y RNNs 20

2.2.4.1. Arquitectura combinando CNNs y RNNs elegida para el trabajo . . 22

2.2.5. Autoencoders Convolucionales 24

2.2.6. Calibración de probabilidades 25

3. Metodología y herramientas **31**

3.1. Recolección de datos 31

3.1.1. Datos de Tracking y Eventing 32

3.1.2. Conjunto de datos unificados Tracking-Eventing 33

3.2. Flujo de proceso 35

3.2.1. Paso1: Preprocesamiento de datos 37

3.2.1.1. Inclusión de filtro gaussiano a las representaciones bidimensionales 39

3.2.2. Paso2: Extracción de características de cada jugada 41

3.2.2.1. Modelos de autoencoder 41

3.2.2.2. Entrenamiento del autoencoder y generación de las representaciones latentes de las imágenes 47

3.2.3. Paso 3: Modelos de predicción basados en RNNs 47

3.2.3.1. Modelos de RNN . . . 48
3.2.3.2. Entrenamiento de la RNN . . . 50
3.2.3.3. Evaluación rendimiento del modelo . . . 52
3.2.4. Paso 4: Calibración de las probabilidades . . . 55
3.3. Clasificador Base . . . 56
3.3.1. Conjunto de datos de entrenamiento del clasificador base . . . 56
3.3.2. Entrenamiento y evaluación del clasificador base . . . 57
3.4. Entorno computacional y herramientas de trabajo . . . 58
3.4.1. Azure Machine Learning como entorno de desarrollo y ejecución . . . 58
3.4.1.1. Notebooks . . . 58
3.4.1.2. Data Storage . . . 59
3.4.1.3. Jobs . . . 59
3.4.1.4. Enviroments . . . 60
3.4.1.5. Compute . . . 61
3.4.1.6. Pipelines . . . 61
3.4.1.7. Models . . . 63
3.4.2. Librerías utilizadas . . . 64

4. Experimentos, resultados y discusión 67
4.1. Descripción de los conjuntos de datos . . . 67
4.1.1. Conjunto de datos de entrada . . . 67
4.1.2. Conjunto de datos con las representaciones bidimensionales . . . 67
4.1.3. Conjunto de datos para entrenamiento del modelo . . . 68
4.2. Experimentos realizados . . . 70
4.2.1. Modelo Base . . . 70
4.2.2. Sistema con autoencoder no convolucional . . . 71
4.2.3. Sistema con autoencoder convolucional . . . 72
4.2.4. Sistema con filtro gaussiano y autoencoder convolucional . . . 72
4.2.5. Resumen de experimentos . . . 73
4.3. Evaluación de resultados . . . 73
4.3.1. Experimento Modelo Base con RNN basada en LSTM (MB_LTSM) . . . 74
4.3.2. Experimento Sistema con autoencoder no convolucional y RNN basada en LSTM (AES_RNN_LSTM) . . . 75
4.3.3. Experimento Sistema con autoencoder no convolucional y RNN basada en GRU (AES_RNN_GRU) . . . 77
4.3.4. Experimento Sistema con autoencoder convolucional y RNN basada en LSTM (AEC_RNN_LSTM) . . . 79

4.3.5. Experimento Sistema con autoencoder convolucional y RNN basada en GRU (AEC_RNN_GRU) . . . 81
4.3.6. Experimento Sistema con filtro gaussiano, autoencoder convolucional y RNN basada en LSTM (AEC_FG_RNN_LSTM) . . . 82
4.3.7. Experimento Sistema con filtro gaussiano, autoencoder convolucional y RNN basada en GRU (AEC_FG_RNN_GRU) . . . 84
4.3.8. Resumen de resultados . . . 86
4.4. Comparación del sistema respecto al algoritmo "Goal Probability" de LLFP. . . . 88

5. Conclusiones y trabajos futuros 91
5.1. Conclusiones . . . 91
5.2. Trabajos futuros . . . 94

A. Planificación y presupuesto 105
A.1. Planificación . . . 105
A.1.1. Planificación Inicial . . . 105
A.1.2. Planificación Final . . . 107
A.2. Presupuesto . . . 108
A.2.1. Costes de Personal . . . 109
A.2.2. Costes de Infraestructura Técnica . . . 109
A.2.3. Costes totales . . . 110

Nomenclatura

AUC Area Under the Curve, página 51

AUC-PR Area Under Precición-Recall Curve, página 52

AWS Amazon Web Services, página 10

BCE Binary Cross Entropy, página 43

CAE Convolutional Autoencoder, página 24

CNN Convolutional Neural Network, página 11

FCN Fully Convolutional Networks, página 20

GPU Graphics Processing Unit, página 19

GRU Gated Recurrent Units, página 15

LLFP LaLiga de Fútbol Profesional, página 3

LRCN Long-Term Recurrent Convolutional Network, página 20

LSTM Long Short Term Memory, página 15

MSE Mean Squared Error, página 41

PRC Precición-Recall Curve, página 52

RNN Recurrent Neural Network, página 7

ROC Receiver Operating Characteristic, página 51

TFM Trabajo Fin de Master, página 7

Índice de figuras

1.1. Ejemplos de grafismos con información contextual y métricas avanzadas sobre fútbol 3
1.2. Ejemplo de cálculos y estadísticas avanzadas relativas al rendimiento deportivo de jugadores . . . 4
1.3. Ejemplo grafismo "Goal Probability" . . . 5
1.4. Ejemplo de valores de las variables explicativas del modelo "Goal Probability" . . . 6
1.5. Gráfico con la Arquitectura del algoritmo "Goal Probability" . . . 7

2.1. Red Neuronal Recurrente básica . . . 13
2.2. Red Neuronal Recurrente profunda . . . 14
2.3. Neurona Long Short Term Memory . . . 15
2.4. Neurona Gated Recurrent Unit . . . 16
2.5. Arquitectura Transformer original . . . 17
2.6. Filtro Convolucional . . . 19
2.7. Arquitectura tradicional de una CNN . . . 20
2.8. Long-Term Recurrent Convolutional Network . . . 21
2.9. Arquitecturas de redes neuronales para procesamiento de videos . . . 22
2.10. Arquitectura de referencia del presente trabajo combinando CNN y LSTM . . . 23
2.11. Arquitectura de un Autoencoder . . . 25
2.12. Ejemplo de Autoencoder Convolucional . . . 25
2.13. Diagrama de Confiabilidad . . . 27

3.1. Representaciones de posición de jugadores y balón en una jugada . . . 37
3.2. Flujo de proceso del trabajo . . . 38
3.3. Ejemplos de imágenes en escala de grises generadas durante el preprocesamiento . . . 39
3.4. Ejemplos de imágenes en escala de grises con filtro gaussiano . . . 41
3.5. Arquitectura del autoencoder de la etapa de extracción de características . . . 42
3.6. Estructura del autoencoder no convolucional . . . 44
3.7. Estructura del autoencoder convolucional . . . 47
3.8. Estructura de la RNN basada en GRU . . . 49
3.9. Estructura de la RNN basada en LSTM . . . 50

3.10. Curvas ROC y Precisión-Recall 55
3.11. Interfaz de Usuario de Azure Machine Learning Studio 58
3.12. Entorno de ejecución de Azure Machine Learning utilizado en el trabajo 60
3.13. Canalización definida en Azure Machine Learning 62

4.1. Conjunto de representaciones bidimensionales en una jugada 69
4.2. Conjunto de representaciones latentes de los instantes de juego de una jugada . . . 70
4.3. Gráficas métricas entrenamiento RNN del experimento MB_RNN_LSTM 74
4.4. Diagramas de confiabilidad del experimento MB_RNN_LSTM 74
4.5. Curvas ROC y PR del experimento MB_RNN_LSTM 75
4.6. Gráficas métricas entrenamiento RNN del experimento AES_RNN_LSTM 76
4.7. Diagramas de confiabilidad del experimento AES_RNN_LSTM 76
4.8. Curvas ROC y PR del experimento AES_RNN_LSTM 77
4.9. Gráficas métricas entrenamiento RNN del experimento AES_RNN_GRU 77
4.10. Diagramas de confiabilidad del experimento AES_RNN_GRU 78
4.11. Curvas ROC y PR del experimento AES_RNN_GRU 78
4.12. Gráficas métricas entrenamiento RNN del experimento AEC_RNN_LSTM 79
4.13. Diagramas de confiabilidad del experimento AEC_RNN_LSTM 80
4.14. Curvas ROC y PR del experimento AEC_RNN_LSTM 80
4.15. Gráficas métricas entrenamiento RNN del experimento AEC_RNN_GRU 81
4.16. Diagramas de confiabilidad del experimento AEC_RNN_GRU 81
4.17. Curvas ROC y PR del experimento AEC_RNN_GRU 82
4.18. Gráficas métricas entrenamiento RNN del experimento AEC_FG_RNN_LSTM 83
4.19. Diagramas de confiabilidad del experimento AEC_FG_RNN_LSTM 83
4.20. Curvas ROC y PR del experimento AEC_FG_RNN_LSTM 84
4.21. Gráficas métricas entrenamiento RNN del experimento AEC_FG_RNN_GRU 84
4.22. Diagramas de confiabilidad del experimento AEC_FG_RNN_GRU 85
4.23. Curvas ROC y PR del experimento AEC_FG_RNN_GRU 85

5.1. Ejemplos de inferencia de probabilidad de gol 94

Índice de tablas

3.1. Tabla con las variables que conforman el conjunto de datos principal 36
3.2. Arquitectura del encoder no convolucional 42
3.3. Arquitectura del decoder no convolucional 43
3.4. Arquitectura del encoder convolucional 45
3.5. Arquitectura del decoder convolucional 46
3.6. Arquitectura de la RNN basada en GRU 48
3.7. Arquitectura de la RNN basada en LSTM 50
3.8. Matriz de Confusión 52
3.9. Tabla con las variables que conforman el conjunto de datos del modelo base 57

4.1. Volumetría del conjunto de datos fuente principal del TFM 68
4.2. Resumen características de los experimentos 73
4.3. Métricas de rendimiento del experimento MB_RNN_LSTM sobre conjunto de test . . 75
4.4. Métricas de rendimiento del experimento AES_RNN_LSTM sobre conjunto de test . 76
4.5. Métricas de rendimiento del experimento AES_RNN_GRU sobre conjunto de test . . 78
4.6. Métricas de rendimiento del experimento AEC_RNN_LSTM sobre conjunto de test . 80
4.7. Métricas de rendimiento del experimento AEC_RNN_GRU sobre conjunto de test . . 82
4.8. Métricas de rendimiento del experimento AEC_FG_RNN_LSTM sobre conjunto de test 83
4.9. Métricas de rendimiento del experimento AEC_FG_RNN_LSTM sobre conjunto de test 85
4.10. Resultado de evaluación de los experimentos sobre el conjunto de datos de test . . . 86
4.11. Resultado de evaluación de los experimentos sobre todo el conjunto de datos 87
4.12. Comparativa de evaluación del sistema AEC_RNN_GRU respecto "Goal Probaility" de LLFP 89

A.1. Estimación inicial de esfuerzo de desarrollo del TFM 107
A.2. Planificación inicial de elaboración del TFM 107
A.3. Planificación y valoración de esfuerzo real de elaboración del TFM 108
A.4. Costes uso infraestructura de cómputo 110
A.5. Costes totales elaboración TFM 110

Capítulo 1

Introducción general y objetivos

En los últimos años hay un creciente interés por enriquecer la experiencia de los usuarios aficionados al fútbol con información contextual, estadísticas y cálculos avanzados como apoyo a los análisis pre y post partido o incluso sobreimprimirla en las retransmisiones de los encuentros en vivo por televisión; integrando así la experiencia digital con la audiovisual de forma interactiva. En las figuras 1.1 y 1.2 se pueden ver algunos ejemplos.

Figura 1.1: Ejemplo de grafismo que se utiliza sobre imágenes de los partidos de LaLiga de Fútbol Profesional mostrando información contextual de lo que ocurre en una jugada (LaLiga Beyond Stats).

Una de las métricas avanzadas que LaLiga de Fútbol Profesional (LLFP) introdujo a partir de la temporada 2021-2022 en las retransmisión de televisión de los partidos fue la probabilidad de que un remate a puerta acabe en gol, la cual se muestra como grafismo en las repeticiones de los disparos o remates a puerta, como se puede ver en la figura 1.3. El objetivo de esta métrica es ofrecer a los fans una mejor y más precisa compresión de la dificultad que supone la realización de un gol en una determinada jugada, sin tener en cuenta las propiedades particulares del rematador a la hora de encarar la portería, como su calidad, la técnica o la efectividad de sus disparos.

Figura 1.2: Ejemplo de cálculos y estadísticas avanzadas relativas al rendimiento deportivo de jugadores de fútbol (LaLiga Beyond Stats).

1.1. Planteamiento del problema, motivación y objetivos

1.1.1. Introducción al algoritmo "Goal Probability"

El algoritmo "Goal Probability" que ha desarrollado LLFP se calcula a partir de dos fuentes de información:

- **Eventing**: Información acerca de los eventos ocurridos en los encuentros de fútbol (pases, faltas, tiros a puerta, saques de esquina,...) acompañados del momento en que se produjeron y otros cualificadores de contexto como el partido, el equipo, el jugador involucrado, la dirección de ataque del equipo del jugador, tipo de falta, realización del tiro (gol o no gol), etc...

  Esta información es proporcionada a través del fichero F24 generado por la compañía Stats Perform, especializada en captura de eventos y generación de datos y métricas deportivas[1].

- **Tracking**: En los estadios de fútbol de primera y segunda división de LLFP se han instalado un sistema de seguimiento óptico (TRACAB system desarrollado por la empresa Chyronhego [2]), que permiten registrar en tiempo real tanto la posición (coordenadas espaciales x,y) de cada jugador y de la pelota, como la la altura del balón, 25 veces por segundo.

Ambas fuentes de información se combinan para generar un set de datos con información histórica de más de 2.000 partidos, recogiendo información de cerca de 21.000 tiros a puerta. Cada disparo contiene las coordenadas espaciales (x,y) de los 22 jugadores, las coordinadas espaciales (x, y, z) del balón, la identificación del jugador y equipo que realiza el remate y la etiqueta de si el disparo fue gol o no gol.

[1] https://www.statsperform.com/opta-feeds/
[2] https://chyronhego.com/wp-content/uploads/2019/01/TRACAB-PI-sheet.pdf

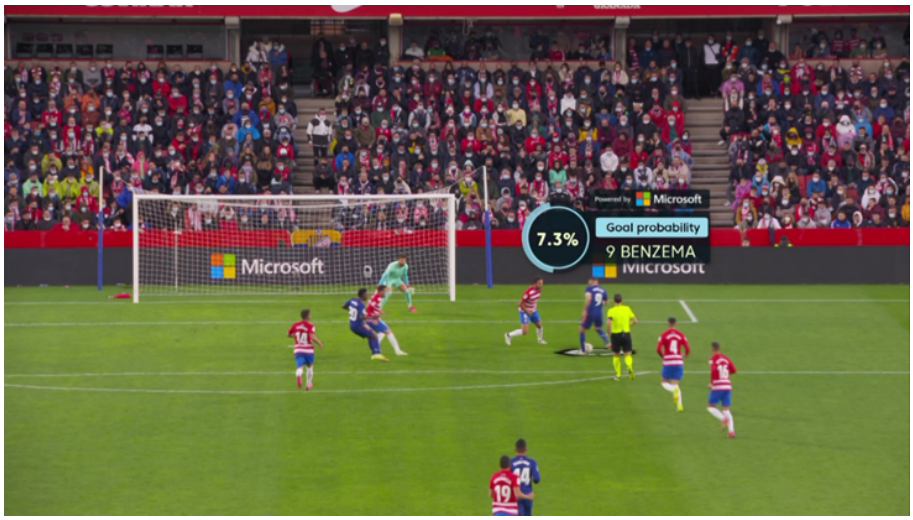

Figura 1.3: Ejemplo de grafismo que se utiliza en la retransmisión de los partidos de LaLiga de Fútbol Profesional para mostrar el resultado del algoritmo de predicción de probabilidad de gol a partir de variables de contexto calculadas en el momento del disparo a puerta. Imagen cedida por LLFP.

Este conjunto de datos sirve a su vez para generar ciertas variables de contexto, que serán las variables explicativas usadas en el entrenamiento de un modelo de aprendizaje automático que predice la probabilidad de que cada disparo acabe en gol. Las variables de contexto, características o variables explicativas que hacen de entrada al modelo son: la distancia del rematador a la portería, la distancia al portero, la distancia al jugador rival más cercano, el número de jugadores rivales existente en el triángulo formado por el jugador que dispara y los palos de la portería, el ángulo respecto los palos de la portería que tienen el jugador que dispara, el ángulo respecto al portero, etc...Por otro lado, la variable objetivo o variable "target" es la etiqueta de gol o no gol asignada a cada disparo en función de si el remate finalmente acabó o no en gol.

Por tanto, LLFP se encuentra ante un problema supervisado de clasificación binaria que, en este caso, es resuelto mediante un modelo de regresión logística, cuya salida es identificada como la probabilidad de que la jugada acabe en gol. En la figura 1.4 se ilustra un ejemplo de jugada con los valores de las variables explicativas asociadas al momento del disparo, junto con la predicción de probabilidad de gol (o salida del modelo) que se muestra en la retransmisión del partido.

Aunque otros modelos de aprendizaje automático más sofisticados pudieran potencialmente conseguir mejores predicciones, el uso de la regresión logística se ajusta a las necesidades de negocio por dos principales motivos:

- La regresión logística permite conocer el peso y la importancia que otorga cada variable explicativa a los resultados del modelo, lo que proporciona interpretabilidad a los resultados arrojados por el algoritmo, hecho de extrema importancia para LLFP por querer entender y poder explicar a los medios y a los fans los motivos exactos por los que una probabilidad de gol de una jugada es alta o baja.

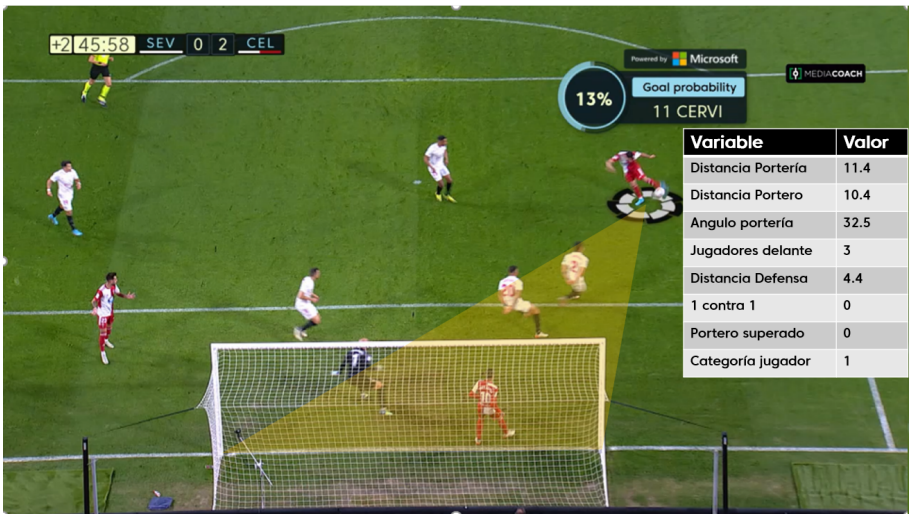

Figura 1.4: Ejemplo de imagen de ocasión de gol junto con los valores asociados a las variables explicativas del modelo "Goal Probability" que predice la probabilidad de que el remate acabe en gol. Imagen cedida por LLFP.

- Es computacionalmente muy eficiente en tiempo de inferencia. Puesto que LLFP está utilizando el algoritmo para mostrar la probabilidad de gol en un grafismo sobreimpreso en las repeticiones de las jugadas de gol durante las retransmisiones en directo de los partidos de fútbol, se hace necesario que todo el flujo de datos (captura de información por las cámaras ópticas, registro de las coordenadas espaciales, calculo de las variables explicativas, inferencia del modelo y presentación del resultado al operador de grafismos) esté ejecutado en menos de 30 segundos. El modelo de regresión logística en tiempo de inferencia tiene una latencia mínima por lo que contribuye eficientemente a que se cumplan los tiempo de respuesta exigidos por negocio. En la figura 1.5 se muestra la arquitectura del modelo de solución que soporta la ejecución en tiempo real del algoritmo "Goal Probability" de LLFP.

1.1.2. Limitaciones del modelo "Goal Probability"

Pese a su interpretabilidad y eficiencia computacional, el modelo "Goal Probability" que ha desarrollado LLFP tiene algunas limitaciones debido, principalmente, al diseño de las variables explicativas, entre las que se destacan:

- No incorpora las jugadas de balón parado: saque de esquina, faltas indirectas, etc..
- No predice satisfactoriamente distintas tipologías de jugadas de gol, como boleas o remates de cabeza,.. ya que las variables explicativas no integran la información de la altura del balón y, por tanto, considera que todos los tiros se han realizado con el balón controlado por el jugador con sus pies.
- No contempla el contexto de la jugada previo al disparo a puerta que, en muchas ocasiones, condiciona la dificultad del remate y por tanto la probabilidad de marcar el gol.

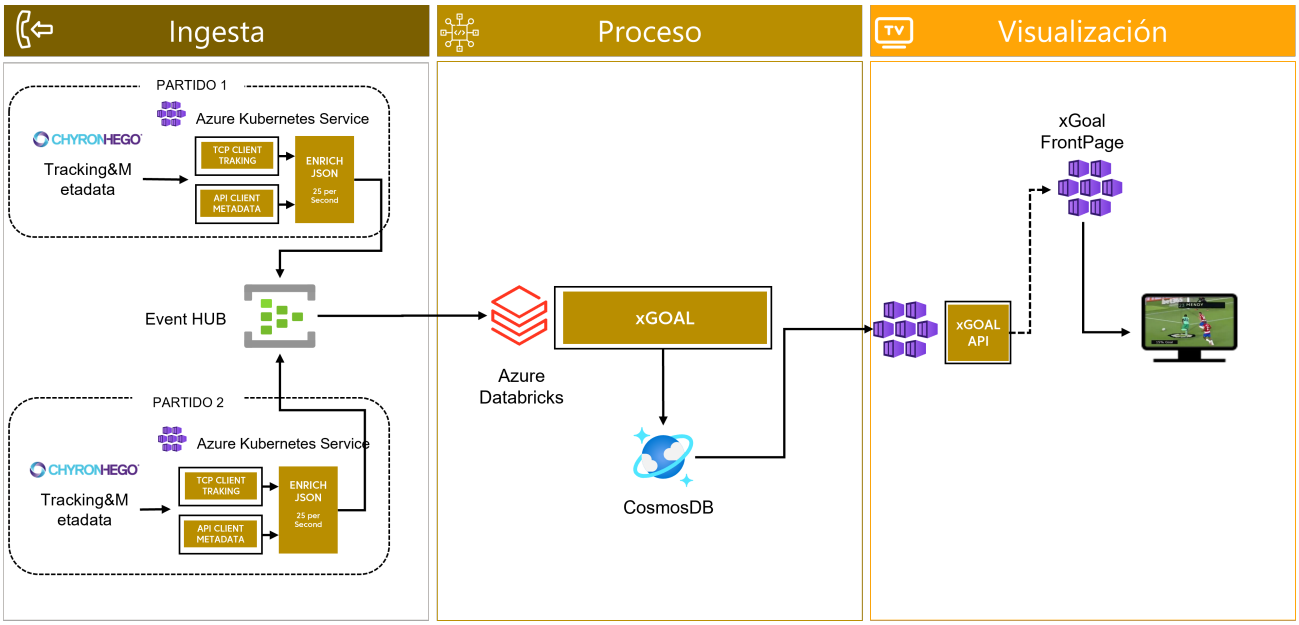

Figura 1.5: Gráfico mostrando la arquitectura técnica que soporta la ejecución del algoritmo "Goal Probability" de LLFP para poder ofrecer la probabilidad de realizar un tiro a puerta durante la repetición de las ocasiones de gol en las retransmisiones en directo de los partidos de fútbol, en menos de 30 segundos. Ilustración cedida por LLFP.

1.2. Motivación y Objetivos

El presente Trabajo Fin de Master (TFM) tiene como motivación predecir la probabilidad de marcar un gol que tiene una jugada de fútbol, resolviendo las limitaciones que presenta el modelo "Goal Probability" que ha desarrollado LLFP actualmente. Estas limitaciones, según se han descrito en la sección 1.1.2, están asociadas al diseño de las variables explicativas del modelo, las cuales se han construido a partir de la posición de los jugadores y del balón en el momento del remate a puerta y, por tanto, no tienen en cuenta la información temporal de lo acontecido en la jugada en los segundos previos al disparo.

Por tanto, el objetivo principal del trabajo es diseñar, construir y evaluar un modelo de predicción basado en Redes Neuronales Recurrentes (Recurrent Neural Networks o RNN) que incorpore la información del trascurso de la jugada y no sólo del momento del disparo.

Para la consecución de este objetivo principal se pretende alcanzar los siguientes objetivos específicos:

- Revisar las aproximaciones realizadas en estos últimos años al problema de predecir la probabilidad de gol en el fútbol, resumiendo el "estado del arte" en cuanto a las soluciones propuestas.
- A partir de las fuentes de información disponibles, construir procesos y algoritmos que permitan extraer la información espacio-temporal de la jugada previa a un remate a puerta y representarla de manera que sea apropiada para alimentar el sistema predictivo propuesto.
- Construir un conjunto de datos correctamente etiquetado para resolver el problema de apren-

dizaje supervisado de clasificación del que se ocupa el trabajo.

- Diseñar y entrenar modelos de aprendizaje profundo basados en RNNs que permitan cumplir el objetivo general propuesto.
- Evaluar los resultados de las estrategias propuestas para determinar cual es el sistema que mejor resuelve el problema planteado.
- Interpretar y calibrar los resultados del modelo para conocer el grado de cumplimiento del objetivo general propuesto.
- Hacer uso de la herramienta Azure Machine Learning y el framework de trabajo que implementa, para diseñar los flujos de trabajo, registrar los experimentos y gobernar los modelos y los conjuntos de datos.

1.3. Estructura de la Memoria

La memoria de este proyecto se estructura en torno a los siguientes capítulos:

- **Capítulo 1** - Introducción general y objetivos: Es el presente capítulo introductorio, en donde, además de presentar y contextualizar el problema que se intenta resolver, se describe la motivación y los objetivos del Trabajo Fin de Máster.
- **Capítulo 2** - Fundamento teórico y estado del arte: En esta capítulo se repasa la bibliografía o "estado del arte" de trabajos previos relacionados con el problema que se pretende resolver y se describe el fundamento teórico que sustenta el presente trabajo y las referencias bibliográficas que respaldan las metodología empleadas para llevarlo a cabo.
- **Capítulo 3** - Metodología y herramientas: En este capítulo se describe el conjunto de datos, se expone el diseño de los experimentos detallando los flujos de proceso, los métodos y los modelos utilizados; y también se indica tanto el entorno/herramientas de trabajo, como el entorno computacional disponible.
- **Capítulo 4** - Resultados y discusión: En este capítulo se muestran los resultados de los experimentos realizados y se comparan mediante las métricas de evaluación.
- **Capítulo 5** - Conclusiones y trabajos futuros: Último capítulo en donde se enumeran las conclusiones obtenidas, se valoran los resultados y se apuntan las posibles líneas de trabajo que se abren a partir del estudio realizado.

Capítulo 2

Fundamento teórico y estado del arte.

En este capítulo se repasa la bibliografía o "estado del arte" de trabajos previos relacionados con el problema que se pretende resolver y se describirá el fundamento teórico que sustenta el presente trabajo y las referencias bibliográficas que respaldan las metodologías empleadas para llevarlo a cabo.

2.1. Estado del arte del cálculo de probabilidad de gol

El uso de los datos y las estadísticas deportivas para extraer patrones y resumir características del rendimiento de los jugadores y los equipos de fútbol es algo que ya se encuentra en estudios publicados en 1968 (Reep and Benjamin (1968)) con numerosas tablas de métricas y estadísticas de encuentros de futbol de la Premier League. En aquel momento se asumía que el fútbol era un juego en el que intervienen muchos factores variables y aleatorios que hacen muy difícil predecir los resultados.

Posteriormente, al calor de la industria de las apuestas deportivas, se desarrollan numerosas metodologías y técnicas para predecir las probabilidad de victoria de un equipo usando modelos estadísticos y métricas deportivas históricas, como en Jones et al. (2004); Karlis and Ntzoufras (2003); Heuer et al. (2010); incluso se empiezan a utilizar modelos supervisados de machine learning entrenados a partir de eventos observados en los partidos (Kerr (2015); Hoekstra et al. (2012)), pero siempre orientados a estimar las posibilidades de victoria de los encuentros más que la probabilidad de marcar un gol en particular.

Las primeras referencias que se encuentran relativas al termino probabilidad de gol son en los estudios de Barnett de 1993 (Barnett and Hilditch (1993)) sobre el impacto de los campos de césped artificial en el rendimiento de los equipos de cara a gol. En los trabajos Pollard and Reep (1997); Pollard et al. (2004); Ensum et al. (2004) se estudian diferentes factores que afectan al éxito de los tiros a puerta analizados en importancia mediante una regresión logística, con el objetivo de determinar las estrategias de juego más efectivas.

Posteriormente, ya son publicados numerosos trabajos siguiendo la misma línea de análisis, entre

los que se destaca Rudd (2011), en donde, utilizando cadenas de Markov que describen las transiciones entre distintos estados de juego, se calculan las probabilidades de gol como medio para entender las estrategias de juego más eficaces; y Rathke (2017) en donde se divide el campo en 8 zonas y también, con el uso de una regresión logística, se relacionan los disparos realizados en cada zona con la probabilidad de gol.

No obstante, no ha sido hasta la mitad de la década pasada en Bialkowski et al. (2014b,a); Lucey et al. (2015a), una vez que la tecnología ha avanzado hasta el punto de poder obtener de manera automática la posición espacio-temporal de la pelota y de los jugadores en cada instante de juego (tracking), cuando aparecen estudios que utilizan dicha información para explicar lo que ocurre en los encuentros y generar métricas avanzadas que ayuden a entender las características de juego de los equipos y los jugadores. Estos mismos autores en Lucey et al. (2015b) son los primeros que tratan de predecir la probabilidad de marcar gol como un problema de aprendizaje automático supervisado, usando datos históricos de tracking, diferentes características explicativas extraídas a partir de dichos datos, y entrenando regresiones logísticas como modelos para la inferencia.

En esta última línea de estudio se encuentran varios trabajos, como Eggels (2016); Rowlinson (2020); Anzer and Bauer (2021) donde, usando los mismos tipos de datos y similares variables explicativas que en los estudios previos, se entrenan diferentes modelos de aprendizaje automático (regresiones logísticas, árboles de decisión, Random Forest, Ada Boost, XGBoost) para obtener la probabilidad de gol en el momento del disparo. Además, en el trabajo de Eggels se introduce el concepto de calibración de la probabilidad de gol, en donde mediante diferentes metodologías se ajustan los valores de salida de los modelos en tiempo de inferencia para que realmente ofrezcan una probabilidad estadística de ocurrencia del evento (ver sección 2.2.6).

Fuera del ámbito académico, aparte de LLFP, también se encuentra cómo otras grandes ligas de fútbol europeo están incorporando a las retransmisiones televisivas métricas avanzadas extraídas a partir de la información de tracking, incluida la probabilidad de gol, como ocurre en la Bundesliga alemana en colaboración con Amazon Web Services (AWS)[1].

Adicionalmente, tradicionales compañías proveedoras de estadísticas y métricas deportivas, como Stats Perform, ofrecen la métrica de probabilidad esperada de gol, la cual, según se indican en su página web[2], es construida de la misma manera que lo apuntado en los artículos académicos previos: recogida de datos históricos de tracking de los jugadores y el balón, cálculo de características explicativas del contexto de la jugada y entrenamiento de un modelo de regresión logística.

Finalmente, se ha encontrado un artículo en el sitio web de AWS [3] que describe la construcción de un indice de intensidad para cuantificar la probabilidad de que un equipo marque un gol en cada instante del juego, utilizando aprendizaje por transferencia (o transfer learning) sobre una arquitectura

[1] https://www.bundesliga.com/de/bundesliga/news/expected-goals-xgoals-torwahrscheinlichkeit-analyse-statistik-aws-amazon-13252

[2] https://www.statsperform.com/resource/expected-goals-in-context/

[3] https://aws.amazon.com/es/blogs/machine-learning/predicting-soccer-goals-in-near-real-time-using-computer-vision/

de Redes Neuronales Convolucionales llamada Inflated 3D Networks (Carreira and Zisserman (2017)) y alimentada con imágenes reales del partido.

Aparte de esta última mención, no se han localizado referencias bibliográficas de artículos o trabajos científicos que intenten predecir la probabilidad de gol mediante el uso de redes neuronales profundas a partir de la información espacio-temporal que proporcionan unas secuencias de representaciones bidimensionales de los datos de tracking que, como luego se mostrará, es el enfoque metodológico seguido por el presente TFM. Esto anima a pensar que esta memoria documentará una línea de estudio novedosa y pionera en el sector de la generación y análisis de métricas deportivas.

2.2. Fundamento teórico

Según se ha descrito en los objetivos del presente TFM (ver sección 1.2), para librar las limitaciones del actual modelo "Goal Probability" que ha desarrollado LLFP, se pretende diseñar y entrenar un modelo basado en RNNs, por lo que se puede enmarcar el TFM en el ámbito del aprendizaje automático profundo, al ser las RNNs consideradas un tipo de modelos englobados en dicha disciplina.

Por otro lado, también se verá a lo largo de la memoria que, para extraer la información espacio-temporal de la jugada previa a un remate a puerta, se han generado unas representaciones bidimensionales construidas a partir de los datos de la posición o tracking (coordenadas espaciales x,y) de cada jugador y de la pelota. Estas representaciones bidimensionales, en donde a cada jugador en función del rol que toma en la jugada se le asigna un valor diferente, son asimilables a imágenes en escala de grises. En el campo de la resolución de problemas supervisados y no supervisados a partir de secuencias de imágenes también se ha demostrado la eficacia de las RNNs; no obstante, generalmente las RNNs para tratamiento de imágenes son combinadas con Redes Neuronales Convolucionales (Convolutional Neural Networks CNN) para extraer, no sólo las relaciones temporales, sino también las relaciones espaciales de la naturaleza de la fuente de información. Las CNNs también son consideradas parte de la disciplina del aprendizaje profundo y, junto con las RNN y la combinación de ambas arquitecturas, serán objeto de descripción en la presente sección de la memoria.

Finalmente, indicar que el problema que el presente trabajo pretende resolver es un problema de clasificación binaria (gol o no gol) supervisado, en donde no se toma como resultado la clase inferida a partir de un umbral aplicado a la salida del modelo, sino la probabilidad de ocurrencia del evento. Para transformar la salida del modelo en probabilidades reales se realizará un ejercicio de calibración de los resultados, que también será justificado de manera teórica en el presente capítulo.

2.2.1. Aprendizaje profundo

El aprendizaje profundo (o Deep Learning) fue introducido por Geoffrey Hinton, Yoshua Bengio y Yann LeCun en Lecun et al. (2015) como un subconjunto del aprendizaje automático (o Machine

Learning) que implica entrenar redes neuronales artificiales compuestas por muchas capas para reconocer patrones en los datos. En el aprendizaje profundo, las redes neuronales se utilizan para aprender características cada vez más complejas de los datos. Estos métodos de aprendizaje descubren las características inherentes a la información de entrada, sin requerir de unas reglas predefinidas, sino automáticamente durante el proceso de entrenamiento.

El aprendizaje profundo se ha aplicado con éxito en una amplia gama de campos, incluido la visión por computadora, el procesamiento del lenguaje natural y más; y ha logrado un rendimiento de vanguardia en tareas como clasificación de imágenes, reconocimiento de voz y juegos.

Generalmente, en el aprendizaje profundo las redes neuronales se entrenan utilizando grandes cantidades de datos etiquetados y los patrones aprendidos se pueden usar para hacer predicciones o clasificaciones en datos nuevos no etiquetados. No obstante, el aprendizaje profundo es utilizado en otras categorías de problemas en los que no es necesario el etiquetado de información. Atendiendo a esta división se agrupan los tipos de problemas en las siguientes categorías:

- **Aprendizaje supervisado**: un algoritmo aprende a asignar entradas a salidas en función de ejemplos etiquetados proporcionados en un conjunto de datos de entrenamiento. El algoritmo usa estos datos etiquetados para aprender una función de mapeo desde la entrada hasta la salida. El objetivo es generalizar este mapeo a ejemplos nuevos de datos de entrada; es decir, en el aprendizaje supervisado, el algoritmo aprende a predecir las etiquetas de salida en función de los datos de entrada.

- **Aprendizaje no supervisado**: un algoritmo aprende a identificar patrones y estructuras en datos no etiquetados. A diferencia del aprendizaje supervisado, no se proporcionan etiquetas de destino y el algoritmo tiene que descubrir la estructura subyacente de los datos por sí mismo. Los algoritmos de aprendizaje no supervisado tienen como objetivo encontrar relaciones o similitudes entre puntos de datos y agruparlos en grupos, o reducir la dimensionalidad de los datos conservando las características importantes.

- **Aprendizaje semi-supervisado**: combina el aprendizaje supervisado y no supervisado. Implica entrenar un modelo en datos etiquetados y no etiquetados, donde la cantidad de datos etiquetados es significativamente menor que la de los datos no etiquetados. Los datos etiquetados proporcionan información sobre las etiquetas de salida, mientras que los datos no etiquetados ayudan al algoritmo a identificar patrones y estructuras en los datos. El objetivo del aprendizaje semisupervisado es mejorar el rendimiento del algoritmo para predecir las etiquetas de salida de los datos no etiquetados utilizando la información de los datos etiquetados.

- **Aprendizaje por refuerzo**: un algoritmo aprende a tomar decisiones en un entorno para maximizar una señal de recompensa acumulativa. El agente interactúa con el entorno realizando acciones, recibiendo recompensas o penalizaciones y observando el estado resultante. El objetivo del aprendizaje por refuerzo es aprender una política, que es un mapeo de estados a

acciones que maximiza la recompensa acumulada esperada a lo largo del tiempo. El agente aprende a través de prueba y error, ajustando su política en función de la retroalimentación que recibe del entorno.

El aprendizaje profundo, en función del caso de uso que se pretende resolver, utiliza distintas arquitecturas de redes neuronales, siendo las Redes Neuronales Convolucionales (CNN) y las Redes Neuronales Recurrentes (RNN) las que se han empleado en el presente trabajo y que, a continuación, se pasan a describir .

2.2.2. Redes Neuronales Recurrentes

Si los datos que se necesitan procesar están conformados en secuencias temporales con correlaciones entre los diferentes elementos de la secuencia, como por ejemplo señales de texto y voz, se suelen aplicar RNNs.

La razón por la que las RNNs pueden aprender y memorizar correlaciones temporales en las secuencias de entrada es debido a que mantienen ciertos estados ocultos que dependen no sólo del elemento de la secuencia actual, sino también de los estados anteriores. De esta manera, los estados ocultos implícitamente contienen información sobre la historia de toda la secuencia.

La representación de gráfica de una RNN simple se encuentra a la izquierda de la figura 2.1. En esa misma imagen, pero a la derecha, se muestra que la RNN también se pueden representar "desenrollando" la secuencia de entrada. De esta manera se consideran las salidas de las neuronas ocultas en los diferentes pasos de tiempo, como si fueran salidas de diferentes neuronas en una red neuronal ordinaria con una simple capa oculta; lo que permite aplicar las técnicas habituales de entrenamiento de redes neuronales (retropropagación del gradiente) para calcular las matrices de pesos de la RNN.

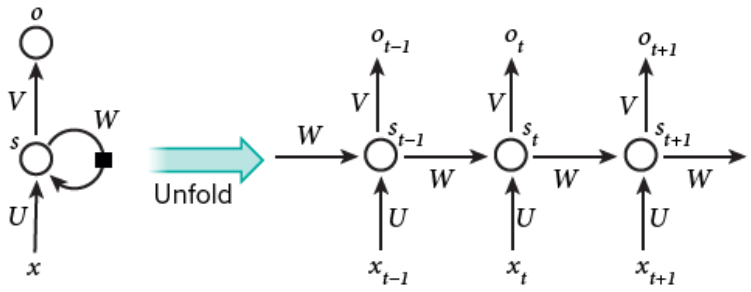

Figura 2.1: Representación de una Red Neuronal Recurrente con una única neurona oculta (izquierda) y su representación "desenrollada" a lo largo de los diferente pasos de tiempo de la secuencia de entrada. Ilustración obtenida de Lecun et al. (2015).

El funcionamiento básico de las RNN es el siguiente, dada una secuencia de entrada $x{=}(x_1, ..., x_T)$, se calcula un vector oculto $h = (h_1, ..., h_T)$ y un vector de salida $y{=}(y_1, ..., y_T)$ mediante la siguiente

ecuación. Desde t=1 a T:

$$h_t = \mathcal{H}(\mathrm{U_{ih}x_t} + \mathrm{W_{hh}h_{t-1}} + \mathrm{b_h})$$

$$y_t = V_{h0}h_t + b_0$$

donde, U, W, V, representan las matrices de pesos (U_{ih} representa la matriz de pesos desde la entrada a la celda oculta, W_{hh} representa la matriz de pesos desde la celda oculta en $t-1$ a si misma en t y V_{h0} representa la matriz de pesos desde la celda oculta a la salida), b representa el termino independiente y $\mathcal{H}$ la función de activación de la capa oculta. Los valores de las matrices de pesos son calculados en el proceso de entrenamiento.

Las matrices de peso U, V, W se comparten en todos los pasos. Esto se debe a que se está aplicando la misma operación en las diferentes entradas que componen la secuencia temporal. Compartir estos vectores de pesos en todos los pasos de tiempo reduce en gran medida el número de parámetros que la RNN necesita aprender.

Se pueden crear RNNs profundas apilando múltiples capas ocultas de RNNs, siendo las secuencias de valores de salida de una capa, la secuencia de entrada para la siguiente, como se muestra en la figura 2.2 de dos maneras diferentes.

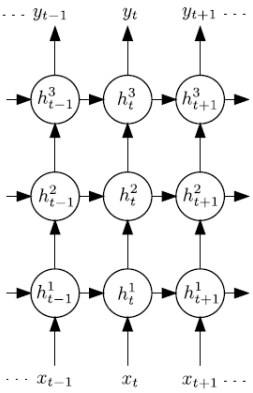

Figure 3. **Deep Recurrent Neural Network.**

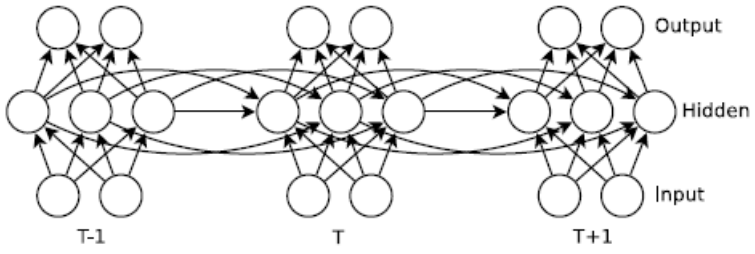

Figura 2.2: Representaciones de una Red Neuronal Recurrente profunda obtenida de Graves and Jaitly (2014) y Sutskever et al. (2011).

Las RNNs son sistemas dinámicos muy poderosos, pero entrenarlos resultó ser problemático porque los gradientes a lo largo de muchos pasos de tiempo típicamente explotan o se desvanecen (Bengio et al. (1994)). El motivo de ocurrencia de este problema es que los gradientes utilizados en el proceso de aprendizaje de los valores de las matrices de pesos U, W, V son obtenidos multiplicando el error de los gradientes en cada periodo de tiempo. Debido a que el número de pasos de tiempo puede ser grande, si los errores de los gradientes en su gran mayoría son más pequeños que la unidad,

los resultados generales de la multiplicación se aproximarán a cero. Por otro lado, si los errores de los gradientes son en su mayoría mayores que la unidad, entonces los gradientes pueden explotar hasta el infinito. Tanto los gradientes que se desvanecen como los que explotan impedirán la convergencia del entrenamiento. Si bien el problema del gradiente explosivo puede mitigarse mediante técnicas de control de gradiente, el problema del gradiente desvanecido debe ser abordado mediante nuevas arquitecturas de RNNs.

Una arquitectura de RNN compuesta por células especiales denominadas Long Short Term Memory (LSTM) propuesta en Hochreiter and Schmidhuber (1997), ha proporcionado una solución popular y eficaz a este problema. La figura 2.3 ilustra una células LSTM.

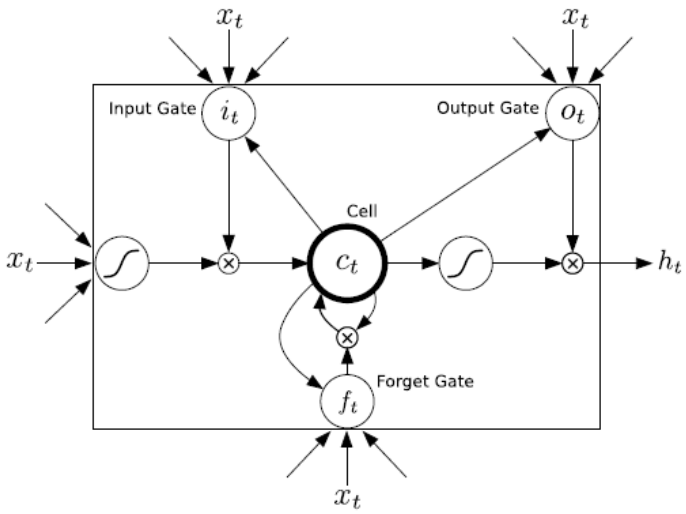

Figura 2.3: Representación de una neurona Long Short Term Memory obtenida de Graves and Jaitly (2014).

Las LSTM han demostrado ser estables para modelar problemas con secuencias de datos largas, como se aprecia en numerosos estudios (Graves (2013); Sutskever et al. (2014)). La principal innovación de las LSTM es su celda de memoria c_t la cual actúa esencialmente como un acumulador de la información de estado en cada secuencia. La celda es accedida, escrita y borrada por varios controles llamados "puertas" (o "gates" en inglés) auto-parametrizados durante el aprendizaje. Cada vez que llega una nueva entrada, su información se acumulará en la celda si la puerta de entrada i_t está activada. Además, el estado anterior de la celda c_{t-1} podría "olvidarse" en este proceso si la puerta de olvido f_t está activada. Finalmente, la salida de la celda que se propagará al estado final h_t se controla mediante la puerta de salida o_t. Una ventaja de usar la celda de memoria c_t y las puertas para controlar el flujo de información es que el gradiente quedará atrapado en la celda (también conocido como carrusel de error constante Hochreiter and Schmidhuber (1997)), y así se evita el problema del desvanecimiento del gradiente.

Las Gated Recurrent Units (GRU) introducidas en Cho et al. (2014b), son otra adaptación de la arquitectura RNN para superar el problema del desvanecimiento del gradiente. Su diseño es muy

similar al LSTM (ver figura 2.4) y tienen un rendimiento comparable a las LSTM (como se demuestra en Jozefowicz et al. (2015); Chung et al. (2014)) pero, como son más recientes y pueden considerarse como una versión simplificada de los LSTM, son más rápidas de entrenar. Las GRU también controlan el flujo de datos a través de las puertas, pero en lugar de tres puertas, solo contienen dos: la puerta de borrado y la puerta de actualización. Estas puertas regulan qué información mantener en la salida y qué información excluir. La puerta de actualización se encarga de la memoria a largo plazo, lo que significa que controla cuáles son las contribuciones de pasos anteriores que merece la pena conservar en el siguiente paso de tiempo; mientras que la celda de borrado se encarga de la memoria a corto plazo ya que determina cuanta información del pasado tiene que ser olvidada.

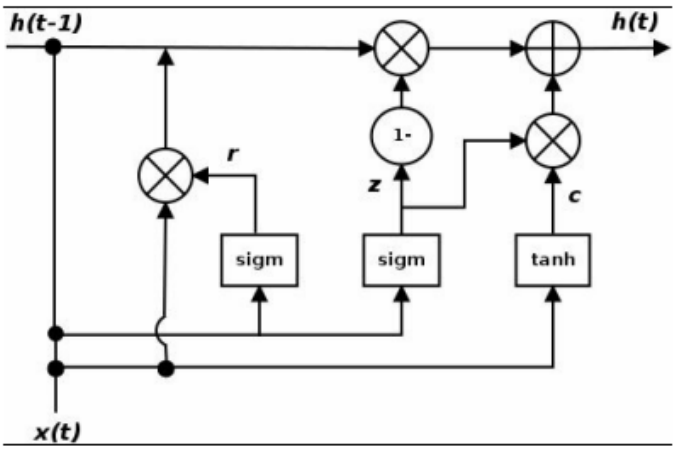

Figura 2.4: Representación de una neurona Gated Recurrent Unit obtenida de Kapoor et al. (2022).

Gracias a los avances en el diseño de nuevas arquitecturas y en las formas de entrenamiento de las redes, las RNNs dieron buenos resultados en trabajos para predecir el siguiente carácter de un texto (Sutskever et al. (2011)) o la siguiente palabra de una secuencia (Mikolov et al. (2013)); pero también en tareas más complejas del ámbito del procesamiento del lenguaje natural, como reconocimiento de voz (Hinton et al. (2012); Graves and Jaitly (2014)), traducciones automáticas (Cho et al. (2014a)), análisis de sentimiento (Gopalakrishnan and Salem (2020)), resúmenes de texto (Fan et al. (2014)), etc...

Por último, indicar que en 2017 Vaswani (Vaswani et al. (2017)) introdujo una nueva arquitectura de RNN llamada Transformers (ver figura 2.5). Los transformers fueron diseñados para abordar las limitaciones de las RNN tradicionales en el procesamiento de largas secuencias de datos. Resumidamente, los Transformers utilizan un mecanismo de "autoatención" para calcular la relevancia de las diferentes partes de la secuencia de entrada. Esto permite que la red se centre en la información más importante de la secuencia e ignore las partes irrelevantes. A diferencia de las RNN, los Transformers pueden procesar toda la secuencia de entrada en paralelo, lo que los hace mucho más rápidos y eficientes para secuencias largas.

Los Transformers, en la actualidad, han desplazado totalmente al uso de RNN tradicionales para

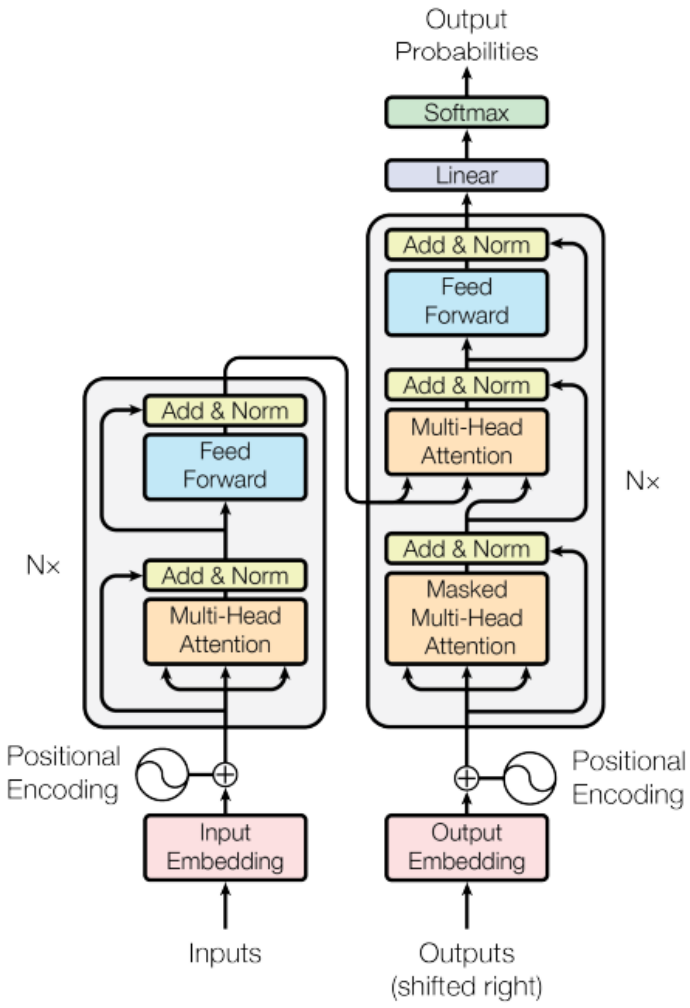

Figura 2.5: Figura que representa la arquitectura de un Red Neuronal Transformer según fue definida originalmente en Vaswani et al. (2017).

realizar todo tipo de tareas asociadas al procesamiento de lenguaje natural como la clasificación de textos, identificación de entidades, el análisis de sentimiento, respuesta automática de preguntas (chatbots), traducciones entre idiomas, reconocimiento y transcripción de voz, generación y auto-completado de textos, etc..

Entre las arquitecturas más populares de redes neuronales basadas en Trasformers, se destacan:

- BERT (Bidirectional Encoder Representations from Transformers): BERT fue introducido por Google en 2018 (Devlin et al. (2018)).

- GPT (Generative Pre-trained Transformer): GPT fue introducido por OpenAI en 2019 (Radford et al. (2019)). Actualmente se encuentra disponible la versión GPT-4 del modelo, y ya la versión GPT-3 contenía alrededor de 175 miles de millones de parámetros.

- T5 (Text-to-Text Transfer Transformer): T5 fue introducido por Google en 2019 (Raffel et al.

(2019)).

- RoBERTa (Robustly Optimized BERT Pretraining Approach): RoBERTa es una variante de BERT introducida por Facebook en 2019 (Liu et al. (2019)).

2.2.3. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales fueron introducidas en Lecun et al. (1998) por Yann LeCun, Yoshua Bengio y otros, a partir de los estudios previos de Fukusima publicados en 1980 (Fukushima (1980)), con el objetivo de aplicar arquitecturas de redes neuronales profundas a datos cuyo formato de entrada fueran matrices múltiples.

La información que se suele representar en forma de matrices múltiples son las imágenes en color, las cuales están compuestas por tres matrices de 2 dimensiones (2D) que contienen intensidades de píxeles en los tres canales de color (RGB); no obstante, muchos otras modalidades de datos están en forma de matrices múltiples: matrices de 1 dimensión (1D) para señales y secuencias, incluido el lenguaje; matrices 2D para imágenes o espectrogramas de audio y matrices 3D para video o imágenes volumétricas.

Hay tres ideas clave detrás de las CNNs que las permiten aprovechar las propiedades de la naturaleza matricial de los datos de entrada:

- **Capas de convolución**: son capas que aplican filtros (matrices de pesos compartidos) a bloques de puntos de la matriz de entrada (ver figura 2.6). Estos filtros permiten extraer rasgos intrínsecos de las imágenes en bloques como, por ejemplo, bordes verticales, bordes horizontales, contrastes, inclinaciones, texturas, ... en lugar de ser procesadas pixel a pixel. La salida de las capas de convolución vuelven a ser grupos de matrices llamadas mapas de características.

- **Capas de agrupación**: son capas que también recorren las matrices de entrada, pero esta vez agrupando los valores obtenidos tras la aplicación de los distintos filtros de las capas convolucionales, con el objetivo de reducir la dimensionalidad de las matrices.

- **Múltiples capas**: las arquitecturas de CNN más comunes para procesamiento de imágenes suelen contener cientos de capas, con el objetivo de ir aprendiendo rasgos y características cada vez más complejas contenidas en las imágenes.

La arquitectura típica de una CNN (ver figura 2.7) está estructurada como una serie de bloques de capas. Los primeros bloques se componen de dos tipos de capas: capas de convolución y capas de agrupación. Las capas convolucionales se organizan como conjuntos de mapas de características calculados al aplicar los filtros convolucionales a los mapas de características de la capa anterior. Según se ha indicado las capas convolucionales se encargan de extraer características subyacentes de pequeñas áreas de las imágenes. Por otro lado, las capas de agrupación fusionarán las características

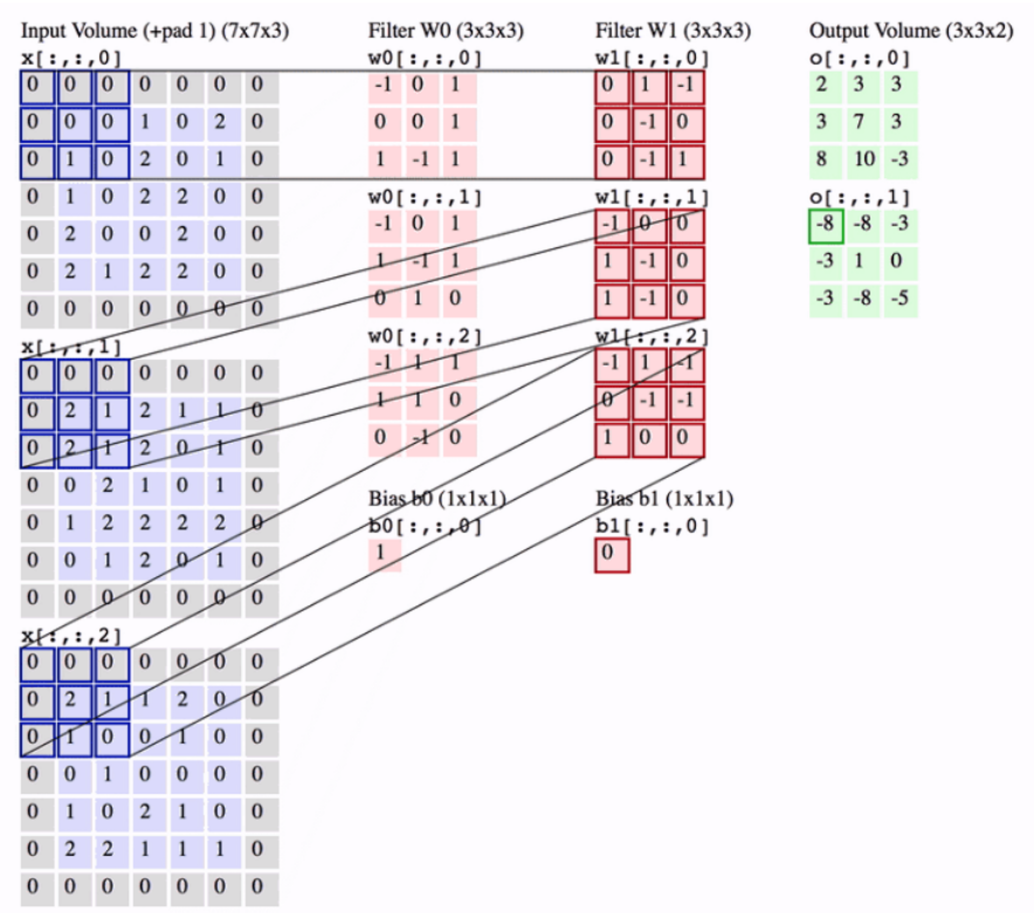

Figura 2.6: Figura que ilustra el procedimiento de filtrado de la capa de convolución. En la imagen se muestran unos filtros tridimensionales de tamaño 3×3×3. Cada filtro se utiliza para recorrer toda la imagen y generar otra imagen (llamada mapa de características) como salida. Imagen obtenida de https://colab.research.google.com/github/juansensio/blog/blob/master/042_cnns/cnns.ipynb.

similares en una única representación, con el objetivo de simplificar y resumir la captura de información. Los últimos bloques de capas, suelen estar formados por grupos de neuronas simples totalmente conectadas.

Desde principios de la década de los 2000 las CNNs han sido utilizadas con gran éxito en detección, segmentación y reconocimiento de objetos y regiones en imágenes, como por ejemplo en reconocimiento de señales, segmentación de imágenes médicas, reconocimiento facial, detección de personas, etc...; sin embargo, no fue hasta 2012 cuando Krizhevsdky introdujo la arquitectura convolucional AlexNet (Krizhevsky et al. (2012)), cuando realmente se produjo un gran avance que aceleró la adopción de las CNNs en la comunidad científica para aplicarlas en el campo de la visión artificial. Este éxito fue debido al uso eficiente de los recursos computacionales a través de unidades de procesamiento gráfico (Graph Proccesing Units o GPU), a la introducción de una nueva técnica de regularización llamada "dropout" y a la capacidad de aumentar la variedad de imágenes de entrenamiento replicándolas pero modificando ligeramente sus características (rotándolas, girándolas,

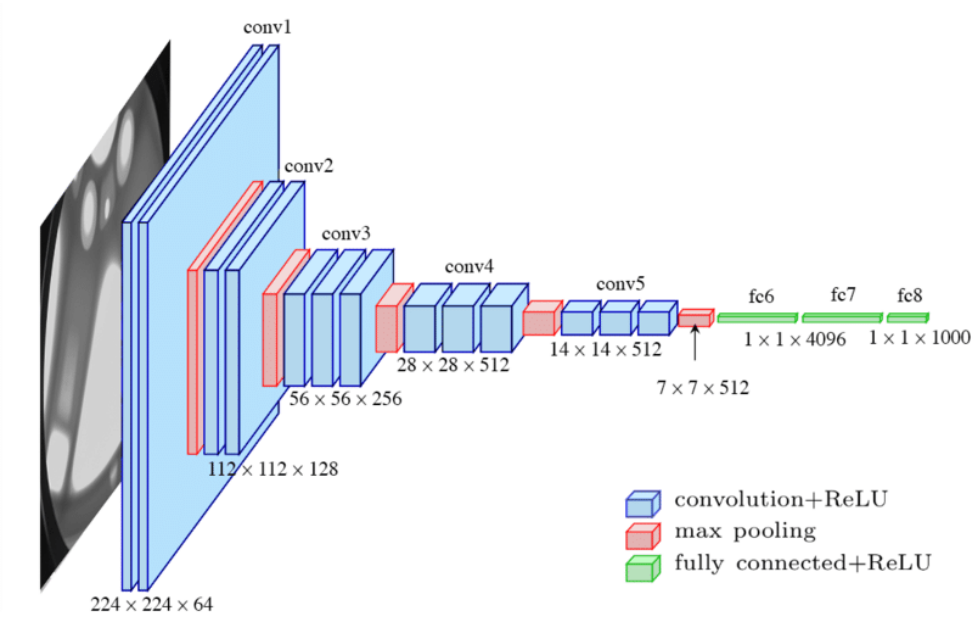

Figura 2.7: Figura que representa la arquitectura habitual de una Red Neuronal Convolucional obtenida de Simonyan and Zisserman (2014b).

etc..).

Seguidamente a la arquitectura AlexNet aparecieron otras CNNs que fueron incorporando mejoras y evoluciones para ofrecer cada vez mayores prestaciones. Entre ellas se destacan las siguientes: SeNet en 2013 (Uijlings et al. (2013)), GoogLeNet en 2014 (Szegedy et al. (2014)) , VGGNet en 2014 (Simonyan and Zisserman (2014b)), ResNet en 2015 (He et al. (2015)) y XCeption en 2017 (Chollet (2017)).

Finalmente, indicar que las arquitecturas de CNNs han seguido evolucionando para estar conformadas por cientos de capas, todas ellas de naturaleza convolucional, lo que ha permitido mejorar la capacidad predictiva en tareas de clasificación, pero también resolver con éxito problemas complejos de detección de objetos y segmentación de imágenes, como puede ser el seguimiento de objetos en movimiento. Algunas de las arquitecturas más populares de este tipo de redes, denominadas por algunos autores redes totalmente convolucionales (Fully Convolutional Networks o FCN), son: U-Net en 2015 (Ronneberger et al. (2015)), SSD en 2015 (Liu et al. (2015)), DeepLab en 2016 (Chen et al. (2016)), Mask R-CNN en 2017 (He et al. (2017)), RetinaNet en 2017 (Lin et al. (2017)), YOLOv3 en 2018 (Redmon and Farhadi (2018)) y EfficientDet en 2020 (Tan et al. (2020))

2.2.4. Combinación de CNNs y RNNs

Aunque la aplicación más habitual de las RNNs se enmarca en el procesamiento de lenguaje natural, lo cierto es que se pueden utilizar para procesar cualquier tipo de datos secuencialmente relacionados a lo largo del tiempo, como series temporales numéricas o incluso secuencias de imágenes

(videos), como se describe, por ejemplo, en Srivastava et al. (2015).

En el campo de la resolución de problemas supervisados y no supervisados a partir de secuencias de imágenes, generalmente las RNNs son combinadas con CNNs para extraer no sólo las relaciones temporales sino también las relaciones espaciales de la naturaleza de la fuente de información. Este concepto se muestra en Donahue et al. (2015), con el diseño de una arquitectura de redes neuronales llamadas Long-Term Recurrent Convolutional Network (LRCN).

Las LRCN combinan una CNN para extraer características visuales jerárquicas profundas de las imágenes, con una RNN basada en células LSTM para aprender a reconocer y sintetizar dinámicas temporales que involucran datos secuenciales (ver Fig2.8).

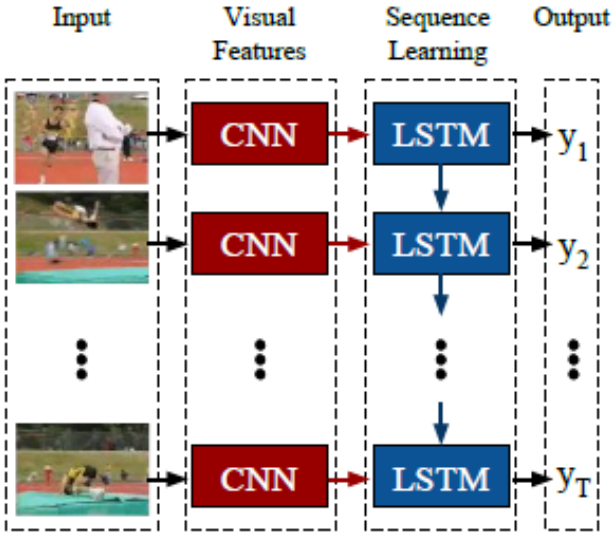

Figura 2.8: Figura que representa el concepto Long-Term Recurrent Convolutional Network, arquitectura de redes neuronales que combina CNNs con RNN-LSTM, obtenida de Donahue et al. (2015).

En Ng et al. (2015) también se argumenta que las CNN son capaces de aprender características complejas de las imágenes para resolver problemas de reconocimiento visual de objetos, pero a la hora de procesar videos, un enfoque ingenuo sería tratar las secuencias del video como imágenes fijas y aplicar CNN para reconocer cada fotograma y, posteriormente, promediar las predicciones a nivel de video, ya que se estaría perdiendo la información temporal. En su lugar, se propone combinar la extracción de características obtenidas mediante las CNN con su procesamiento secuencial a través de una RNN basada en LSTMs.

Este tipo de arquitecturas combinadas permiten resolver con éxito problemas de clasificación de vídeos, pero también otros más complejos como extraer descripciones semánticas de lo que sucede en el video (Shi et al. (2015); Finn et al. (2016); Yang et al. (2021)).

En la literatura se encuentran otros enfoques de arquitecturas de redes neuronales que también permiten con éxito el procesamiento de videos:

- **Redes Convolucionales 3D** (3D Convnet): son como redes convolucionales estándar, pero

con filtros convolucionales espacio-temporales, por lo que crean directamente representaciones espacio-temporales (Tran et al. (2015); Varol et al. (2017)). Ver primera imagen de la figura 2.9.

- **Redes de 2 Flujos** (Two-Stream Networks): se compone de dos flujos de datos, uno que procesa cada imagen mediante una CNN profunda y otro que procesa, también a través de una CNN profunda, un grupo de secuencias de flujo óptico[4] calculadas externamente (Simonyan and Zisserman (2014a); Feichtenhofer et al. (2016)). Ver segunda imagen de la figura 2.9.

- **Combinación de 3D Convnets y Two-Stream Networks** (TwoStream Inflated 3D ConvNets): bien se superpone una 3D Convnet sobre una Two-Stream Networks, o bien se construye una Two-Stream Networks procesada mediante 3D Convnets en lugar de CNN profundas (Carreira and Zisserman (2017)). Ver tercera y cuarta imagen de la figura 2.9.

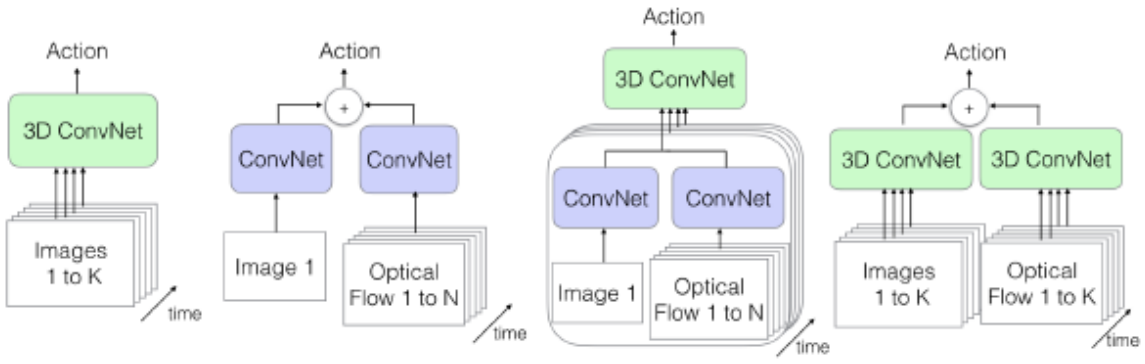

Figura 2.9: Figura que muestra diferentes diseños arquitectónicos de redes neuronales para procesamiento de videos, obtenida de Carreira and Zisserman (2017). La primera imagen corresponde a una 3D-ConvNet, la segunda a una Two-Stream Network y la tercera y cuarta a una TwoStream Inflated 3D ConvNets.

2.2.4.1. Arquitectura combinando CNNs y RNNs elegida para el trabajo

Según se ha ido introduciendo en la presente memoria, el enfoque elegido y que será desarrollado posteriormente para calcular la probabilidad de gol de una jugada, es procesando a través de una RNN la información espacio-temporal contenida en secuencias de representaciones bidimensionales construidas a partir de los datos de la posición (coordenadas espaciales x,y) de cada jugador y de la pelota en los instantes previos al remate a puerta. Estas representaciones, al ser matrices de dimensiones (2D), son asimilables a imágenes en escala de grises.

Por tanto, el TFM se enfrenta a un problema que involucra el procesado de secuencias de imágenes en escala de grises para su clasificación (en gol o no gol), y aplica el uso de arquitecturas

[4]Las secuencias de flujo óptico se refieren a una técnica para seguir el movimiento de objetos en una secuencia de video. Implica calcular el desplazamiento de píxeles entre secuencias sucesivas de un video para estimar el movimiento de los objetos en la escena.

de aprendizaje profundo como las descritas en la sección anterior. En este tipo de escenarios la bibliografía analizada recomienda usar la combinación de CNNs con RNNs.

En este caso, se decide utilizar una arquitectura como la reflejada en la figura 2.10, en donde, para procesar las secuencias de imágenes en escala de grises, primeramente se hace pasar cada imagen por una CNN para extraer las características espaciales y luego dichas características se tratan a través de una RNN basada en neuronas LSTM o GRU.

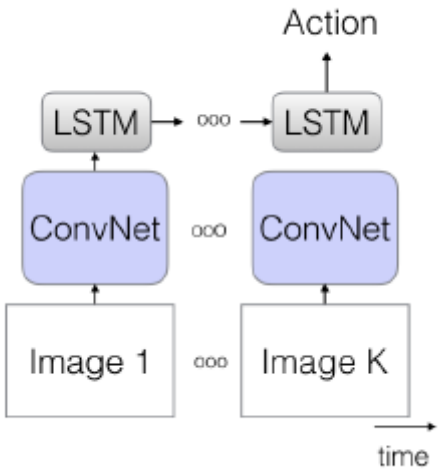

Figura 2.10: Figura que representa la arquitectura de redes neuronales sobre la que se basa el presente trabajo, la cual combina CNNs con RNN-LSTM, obtenida de Carreira and Zisserman (2017).

Estas arquitectura de redes se entrenan de una sóla vez, tomando lotes de secuencias de imágenes de entrada y calculando los pesos, tanto de la parte convolucional como de la parte recurrente, en el mismo proceso mediante la tradicional técnica de retropropagación del gradiente a partir de una única función de pérdida.

En el trabajo presente se ha optado por dividir el procesamiento y el cálculo de los pesos en dos pasos:

- Un primer paso para extraer las características jerárquicas espaciales de las imágenes en escala de grises mediante el uso de CNN.
- En un segundo paso, una vez obtenidas dichas características (en forma de vectores numéricos), agruparlas en secuencias que serán procesadas por una RNN basada en LSTM o GRU con el objetivo de obtener la probabilidad de gol.

El motivo que fundamenta dicha decisión es doble:

- Por un lado, aliviar las exigencias computacionales que supone el entrenamiento de una red profunda compuesta por capas convolucionales y recurrentes. Debido a que los requisitos de

recursos computacionales, es necesario dividir dichos esfuerzos y entrenar por separado la CNN y la RNN.

- Por otro, tener flexibilidad para variar los parámetros y las arquitecturas tanto de la CNN como de la RNN por separado y evaluar más rápidamente cual combinación ofrece el mejor resultado. Es decir, primero se entrenan distintas arquitecturas de CNN haciendo variar los parámetros (número de capas, tipo de capas, número de neuronas, tasas de aprendizaje, optimizadores,..) para elegir aquella combinación que ofrezca mejores resultados, y una vez elegida la CNN óptima se repite el proceso con los parámetros de la RNN.

Para poder entrenar la CNN de manera independiente con el objetivo de extraer las características espaciales subyacentes de las imágenes en escala de grises en forma de vectores numéricos, se enfoca la tarea como un problema de aprendizaje no supervisado, en donde el algoritmo (en este caso la red CNN) aprende a identificar patrones y estructuras en datos no etiquetados. Según se indica en Géron (2019) y en Kapoor et al. (2022), de manera genérica, los Autoencoders son una arquitectura de redes neuronales apropiada para realizar esta tarea y, específicamente, la variante en donde las capas ocultas de la red están conformadas por capas de convolución (llamados Autoencoders Convolucionales), son los sistemas más adecuados para extraer las características subyacentes de los datos de entrada cuando estos son conjuntos de imágenes.

2.2.5. Autoencoders Convolucionales

Los autoencoders son arquitecturas de redes neuronales capaces de aprender representaciones densas de los datos (llamadas representaciones latentes) sin necesidades de disponer de datos etiquetados. Estas representaciones latentes suelen tener una dimensionalidad mucho menor que los datos de entrada, por lo que habitualmente son utilizados, como en el presente TFM, para labores de preprocesamiento de información antes del entrenamiento de una red neuronal profunda (Géron (2019)).

Los autoencoders simplemente tratan de copiar las entradas a las salidas, haciendo pasar la información por capas de neuronas intermedias con el objetivo de forzar el aprendizaje de las características subyacentes de los datos. En la figura 2.11 se puede observar la típica arquitectura de un autoencoder, la cual se compone de dos bloques:

- El "encoder" que se encarga de convertir los datos de entrada en las representaciones latentes.
- El "decoder" que se encarga de generar las salida (reconstruir los datos de entrada) a partir de las representaciones latentes.

Tanto el encoder como el decoder pueden estar compuestos por múltiples capas ocultas de neuronas, y la estructura del enconder y del decoder suele ser simétrica.

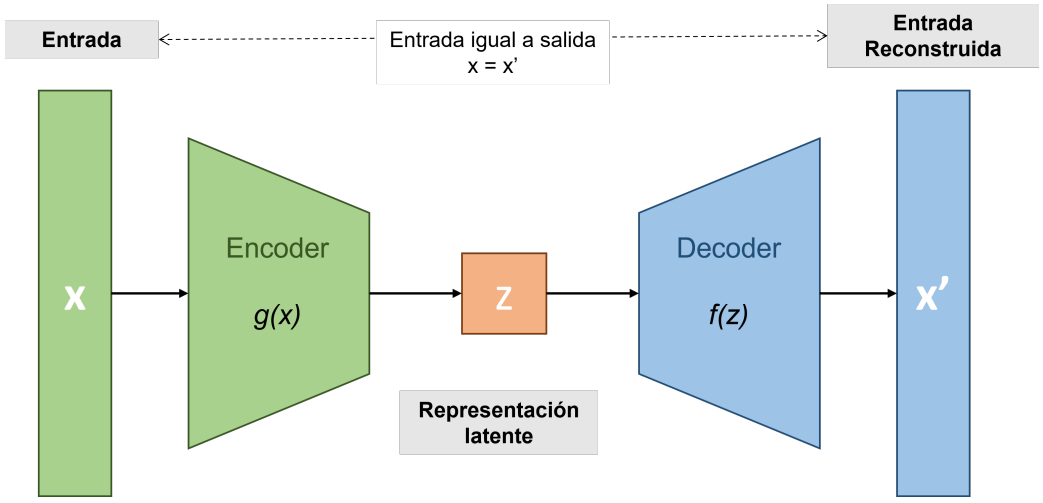

Figura 2.11: Figura que representa la típica arquitectura de un autoencoder.

En el caso de querer utilizar un autoencoder, siendo los datos de entrada imágenes, la mejor opción según se indica en Géron (2019) y en Kapoor et al. (2022) es utilizar un autoencoder convolucional (Convolutional Autoencoder o CAE), introducido en 2011 en Masci et al. (2011). En un CAE las capas ocultas del encoder y del decoder están conformadas por capas de convolución y capas de agrupación al igual que las contenidas en una CNN tradicional. La figura 2.12 ilustra este concepto.

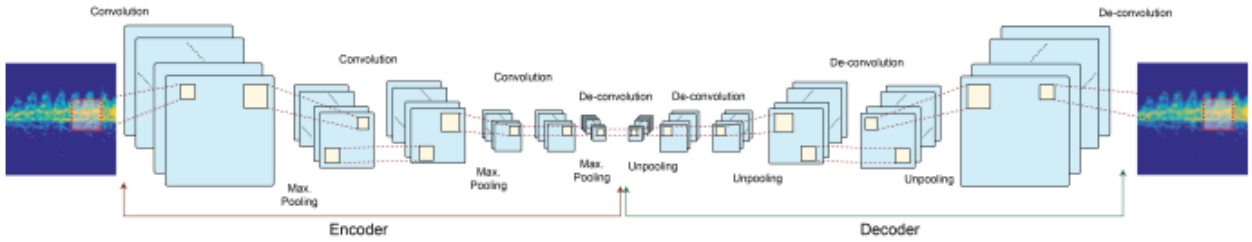

Figura 2.12: Figura que representa la arquitectura estándar de una autoencoder convolucional, obtenida de Seyfioglu et al. (2018).

Los autoencoders convolucionales han resultado de gran utilidad para resolver complejos problemas de aprendizaje no supervisado sobre imágenes, como se puede comprobar en Seyfioglu et al. (2018); Toma et al. (2021); Catak et al. (2022).

2.2.6. Calibración de probabilidades

El problema que el presente trabajo pretende resolver es un problema supervisado de clasificación binaria (gol o no gol), en donde no se toma como resultado la clase inferida a partir de un umbral aplicado a la salida del modelo, sino la probabilidad de ocurrencia del evento.

En los problemas supervisados de clasificación binaria, lo más frecuente es modelar la salida del algoritmo como una distribución logística a través de la función sigmoide, definida por la fórmula:

$$S(g(x)) = \frac{1}{1 + e^{g(x)}}$$

, siendo $g(x)$ la función que representa la salida numérica del algoritmo sobre la instancia x.

Los valores de la función sigmoide están entre 0 y 1, y se suelen interpretar como la probabilidad de asignación a la clase positiva ($P(y = 1|x)$). De esta manera, una vez elegido un umbral (generalmente 0,5), la salida de la función determina si la instancia se asigna a la clase positiva o la clase negativa.

Pero realmente las probabilidades de asignación de clase que devuelve la función sigmoide deben ser interpretadas como un valor de confianza del modelo sobre la pertenencia de una instancia a una u otra clase, y no en todos los casos estos valores están ajustados a las probabilidad reales asociadas a la distribución de los eventos de interés que se pretenden modelar. Para transformar estos valores en estimaciones de probabilidad de ocurrencia de la clase, hay que calibrar las salidas para que realmente reflejen dicha probabilidad.

Como se demuestra empíricamente en Niculescu-Mizil and Caruana (1999) algunos modelos como Suport Vector Machines, Naive Bayes o Boosted Trees, son más propensos que otros a generar probabilidades no calibradas; y en Guo et al. (2017) se argumenta que las redes neuronales modernas generalmente también producen resultados mal calibrados.

Para determinar el grado de calibración, es decir, cuánto están o no las salidas de la función sigmoide alejadas de las probabilidades reales de ocurrencia de la clase, se suele seguir la siguiente técnica:

1. Se agrupan en bloques (también llamados "bins") las instancias de datos del conjunto de entrenamiento que presentan valores de salida de la función sigmoidal parecidos.

2. Se estima la probabilidad real de ocurrencia de la clase positiva en cada bloque de instancias calculando la frecuencia relativa del bloque (ratio de las clases positivas):

   $$p_k = \frac{1}{n_k} \sum_{j=1}^{n_k} y_j$$

   , donde n_k es el número de instancias del bloque k, siendo R el número de bloques $k\epsilon\,\{1, \ldots, R\}$.

3. En cada bloque se compara la estimación de la probabilidad real con la media de los valores de la función sigmoide aplicada a cada instancia del bloque. La media de los valores de la función sigmoide formalmente se representa con la siguiente fórmula:

   $$\bar{y_k} = \frac{1}{n_k} \sum_{j=1}^{n_k} S(g(xj))$$

4. Se agregan los resultados mediante alguna fórmula que permita comparar diferentes situaciones. Tres de las fórmulas más habituales son:

- **Brier Score** (Brier (1950)): En donde se suman las diferencias al cuadrado entre p_k e $\bar{y_k}$ de cada bloque

$$Brier = \sum_{k=1}^{R} \frac{n_k}{n} (p_k - \bar{y_k})^2$$

- **Expectec Calbration Error** (Pakdaman Naeini et al. (2015)): En donde se hace una media del valor absoluto de las diferencias entre p_k e $\bar{y_k}$ de cada bloque

$$ECE = \sum_{k=1}^{R} \frac{n_k}{n} |p_k - \bar{y_k}|$$

- **Maximun Calibration Error** (Pakdaman Naeini et al. (2015)): En donde se toma la máxima diferencia entre p_k e $\bar{y_k}$ de todos los bloques

$$MCE = max_{k\epsilon\{1,...,R\}} |p_k - \bar{y_k}|$$

Para obtener una perfecta calibración, tanto MCE como ECE o MCD deben ser igual a cero.

Gráficamente, se suele representar el grado de calibración mediante los llamado Diagrama de Confiabilidad o Reliability Diagram (Bröcker and Smith (2007)), en donde para cada bloque se comparan las probabilidades posteriores reales p_k, con la media de los valores que arroja el algoritmo $\bar{y_k}$. Unos ejemplos se muestran en la figura 2.13.

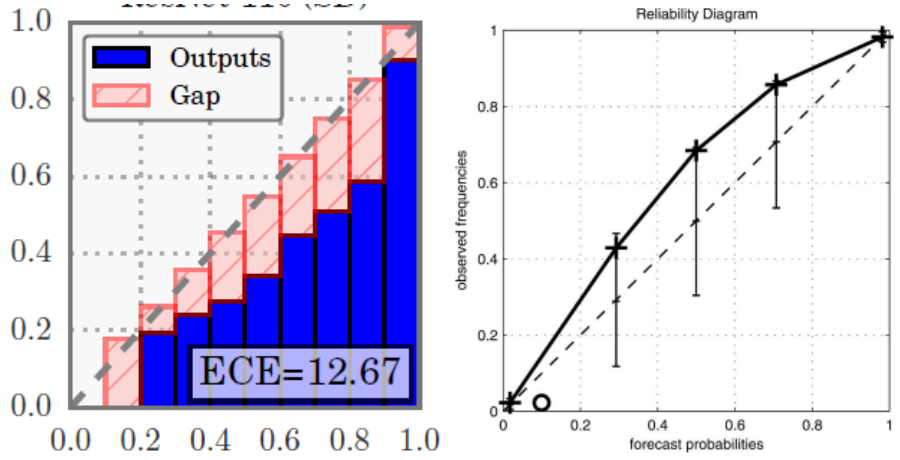

Figura 2.13: La figura de la izquierda muestra un ejemplo de Diagrama de Confiabilidad y su Expected Calibration Error asociado, obtenido de Guo et al. (2017). En figura de la derecha se muestra otra representación del Diagrama de Confiabilidad, también llamada Curva de Calibración, obtenida de Bröcker and Smith (2007).

Un modelo correctamente calibrado será aquel que presente una curva de calibración que coincida con la diagonal principal de los ejes de coordenadas. Si los valores están por debajo de la diagonal principal, el modelo sobrestima, es decir, arroja probabilidades superiores a la realidad; sucediendo la situación contraria (infraestimación) si los valores están por encima de la diagonal principal.

Existen varios métodos para calibrar las salidas de los algoritmos y ajustarlas a las probabilidades reales. Los más habituales son:

- **Platt Scaling** (Platt (1999)): Una vez entrenado el algoritmo predictor $g(x)$, los valores se estiman mediante una regresión logística, produciendo probabilidades calibradas:

$$S(g(x)) = \frac{1}{1 + e^{(Ag(x)+B)}}$$

  Los parámetros A y B se ajustan usando el método de máxima verosimilitud a partir de un conjunto de datos de entrenamiento.

- **Regresión Isotónica o Isotonic Regresion** (Zadrozny and Elkan (2001)): Una vez entrenado el algoritmo predictor y obtenidas las probabilidades no calibradas mediante la función sigmoide $S(g(x))$,se entrena una regresión isotónica la cual producirá probabilidades calibradas.

  La regresión Isotónica se plantea como una regresión lineal $y = \hat{y} + \epsilon = m(S(g(x)) + \epsilon$, siendo y la clase real a predecir (1 o 0). El parámetro m se calcula igual que en la regresión lineal mediante el método de los mínimos cuadrados, pero cumpliendo la restricción que $\hat{y_i} \leq \hat{y_j}$ siempre que $S(g(x_i) \leq S(g(x_i)$, es decir:

$$min \sum_{i=1}^{n} (\hat{y_i} - \hat{y_j})^2$$

  sujeto a $\hat{y_i} \leq \hat{y_j} \forall (i,j) \epsilon E$ donde $E = \{(i,j) : S(g(x_i) \leq S(g(x_i)\}$

Finalmente, indicar que otro bien conocido motivo por el que los algoritmos de clasificación binaria arrojan probabilidades posteriores mal calibradas, es cuando se utilizan técnicas de rebalanceo sobre los set de datos de entrenamiento de los modelos, debido a tener que resolver problemas donde las clases están fuertemente desbalanceadas(Dal Pozzolo et al. (2015)). Como se muestra más adelante en la presente memoria, esta situación se da en la naturaleza de los datos con los que se va a trabajar debido a que las instancias etiquetadas como gol tienen una proporción muy inferior a las etiquetadas como no gol en el conjunto de datos disponible, y se usará una técnica de rebalanceo de las clases para configurar el conjunto de datos con el que se va a entrenar el modelo RNN.

En estos casos, se propone en Saerens et al. (2002) una sencilla técnica para calibrar las probabilidades posteriores obtenidas del modelo, consistente en aplicar la siguiente fórmula:

$$p_f(x) = \frac{\frac{\Pi_t^+}{\Pi_s^+} p_s(x)}{\frac{\Pi_t^+}{\Pi_s^+} p_s(x) + \frac{\Pi_t^-}{\Pi_s^-}(1 - p_s(x))}$$

, siendo:

- $p_s(x)$ la probabilidad de ocurrencia de la clase positiva estimada por el modelo para la instancia x, $P(y = 1|x) = S(g(x))$.

- Π_t^+ la proporción de la clase positiva presente en el conjunto de datos, $\frac{1}{n}\sum_{j=1}^{n} y_j$.
- Π_t^- la proporción de la clase negativa presente en el conjunto de datos, $1 - \Pi_t^+$.
- Π_s^+ la proporción de la clase positiva presente en el conjunto de datos balanceado para el entrenamiento, por lo que generalmente será $0{,}5$.
- Π_s^- la proporción de la clase negativa presente en el conjunto de datos balanceado para el entrenamiento, por lo que generalmente será $0{,}5$.

Capítulo 3

Metodología y herramientas

3.1. Recolección de datos

Según se ha introducido en 1.1.1 LLFP ha usado dos fuentes principales de información para construir el algoritmo "Goal Probability": Eventing y Tracking, las cuales también serán la base para desarrollar el presente trabajo de cálculo de la probabilidad de que un remate a puerta acabe en gol. Existen otros datos que son susceptibles de aportar valiosa información para determinar la dificultad de un disparo a portería y por tanto contribuir al entrenamiento de un modelo que infiera la posibilidad de que el tiro sea gol, pero dichas fuentes de información no están disponibles. Estas fuentes de información complementarias podrían ser:

- **Datos de la calidad del jugador**: capacidades técnicas, eficacia histórica de cara a puerta, estado de forma, posición habitual del jugador (delantero, centro, defensa,.), etc... y no sólo referida al jugador que remata a puerta, sino también a los defensores o del portero.
- **Datos ambientales**: información meteorológica (temperatura, dirección y velocidad del viento, lluvia,..), información sobre la sensación de presión ambiental generada por los aficionados (si se juega en casa o fuera de casa, el aforo disponible y el porcentaje de ocupación, la distribución de aficionados locales y visitantes en el aforo, la cercanía de las gradas al terreno de juego, el nivel de ruido que genera la afición,...), etc...
- **Información de la competición**: posición del equipo en la clasificación, marcador del encuentro en el momento del disparo, criticidad del resultado del partido para cumplir los objetivos del equipo, catalogación del partido (si es una final, un derbi, un clásico,...), etc..
- **Información sobre la dificultad del remate**: velocidad del balón tanto antes como después del remate, si el remate se realiza con la pierna consonante con la catalogación del jugador (zurdo o diestro), si la posición del jugador que remata no es natural sino forzada (bolea, chilena, disparo en caída, remate de tacón, remate con una parte del cuerpo que no es la pierna o la cabeza,...), etc..

3.1.1. Datos de Tracking y Eventing

Por tanto, las fuentes originales de información utilizadas como base para el desarrollo del TFM son las siguientes:

- **Eventing**: Información acerca de los eventos ocurridos en los encuentros de fútbol (pases, faltas, tiros a puerta, saques de esquina,...) acompañados del momento en que se produjeron y otros cualificadores de contexto como el partido, el equipo, el jugador involucrado, la dirección de ataque del equipo del jugador, tipo de falta, realización del tiro (gol o no gol), etc... La fuente de datos de Eventing recoge hasta 60 tipos de eventos diferentes.

  Esta información es proporcionada a través del fichero F24 generado por la compañía Stats Perform, especializada en captura de eventos y generación de datos y métricas deportivas[1], y viene en formato XML.

- **Tracking**: En los estadios de fútbol de primera y segunda división de LLFP se han instalado un sistema de seguimiento óptico (TRACAB system desarrollado por la empresa Chyronhego [2]), que permiten registrar en tiempo real tanto la posición (coordenadas espaciales x,y) de cada jugador y de la pelota, como la la altura del balón, 25 veces por segundo. Esta información es proporcionada mediante ficheros planos de texto, conteniendo una fila por cada cada jugador, lo que significa que albergan alrededor de 3,6 millones de registros por partido.

Ambas fuentes de información, al ser proporcionadas por proveedores diferentes tienen distintos convenios, nomenclaturas y valores para las mismas variables y, por tanto, deben de ser homogeneizadas y coordinadas. Algunos ejemplos de esta situación son:

- El identificador de los jugadores es diferente en ambas fuentes de datos, siendo el número de dorsal el identificador en el fichero de Tracking y un valor numérico único definido por Stats Perform en el caso del fichero de Eventing.

- La referencia en el terreno de juego desde la que se miden la posición de los jugadores y el balón: la esquina inferior izquierda en el fichero de Eventing, mientras que es el centro del campo en el fichero de Tracking.

- En el fichero de Eventing, el momento temporal en el que ocurre una acción viene identificado mediante un Timestamp, mientras que en el fichero de Tracking cada instante de juego viene identificado por un valor secuencial (IdFrame) que cuenta las fracciones de tiempo de duración $1/25$ segundos ocurridas desde media noche. Además estas dos medidas de tiempo no están sincronizadas, pudiendo haber desfases temporales de hasta 3 segundos.

[1] https://www.statsperform.com/opta-feeds/
[2] https://chyronhego.com/wp-content/uploads/2019/01/TRACAB-PI-sheet.pdf

Este proceso de limpieza y homogeneización de información entre ambas fuentes de datos es realizado por LLFP para conseguir un conjunto de datos coordinado, contiendo exclusivamente la información necesaria para entrenar los modelos que se contemplan en el presente trabajo. El proceso consta de las siguientes tareas principales:

- Se relacionan los dorsales de los jugadores presentes en el fichero de Tracking con el identificador de jugador de Stats Perform contenido en el fichero de Eventing, gracias a una tabla auxiliar de relaciones ofrecida por Stats Perform.

- Se seleccionan del fichero de Eventing exclusivamente los eventos de disparo a puerta.

- Se identifica el momento temporal en el que ocurren los eventos contenidos en el fichero de Eventing con el IdFrame que maneja el fichero de Tracking (identificador único de la fracción de tiempo de duración $1/25$ segundos contada desde media noche). Esto no es posible realizarlo con una simple transformación a partir del Timestamp asociado a cada evento en el fichero de Eventing, puesto que ambas fuentes de datos no están sincronizadas temporalmente. Para conseguirlo se toma una ventana de secuencias del fichero de Tracking alrededor del Timestamp asociado al evento y se identifica el instante en el que el balón se encuentra mas cerca del jugador que dispara a puerta. El IdFrame asociado a ese momento es el que se asigna a la información del evento contenida en el fichero de Eventing.

- Se seleccionan del fichero de Tracking los registros asociados a los 125 instantes anteriores a cada evento de disparo que indica el fichero de Eventing, junto con el momento del remate.

- Se transforman las coordenadas posicionales x,y de los jugadores y del balón del fichero de Tracking, las cuales vienen referenciadas tomando el centro del campo como el origen de datos (0,0), a posiciones referenciadas tomando como como origen la esquina inferior izquierda del terreno de juego. Esto es posible gracias a que la información de Eventing también contienen las dimensiones del terreno de juego en el que se disputa cada partido.

- Finalmente se fusionan ambas fuentes de datos y se filtran las variables que no se consideran de interés para el estudio.

3.1.2. Conjunto de datos unificados Tracking-Eventing

Según se han indicado, el conjunto de datos con información fusionada de Eventing y Tracking proporcionado por LLFP y fuente principal del trabajo, contiene la posición (coordenadas espaciales x,y) de cada jugador y de la pelota, así como la altura del balón, en cada uno de los 25 instantes (o frames) de los últimos 5 segundos de cada remate a puerta. Concretamente el conjunto de datos recoge la siguiente información:

- De la fuente de datos de Eventing, los tiros a portería, indicando el jugador que lo ha realizado, el equipo, si ha sido gol o no y otros calificadores de contexto de la jugada que contiene el disparo: saque de esquina, falta, penalti, etc...
- De la fuente de datos de Tracking, las coordenadas x,y de los jugadores y las coordenadas x,y,z del balón en los últimos 5 segundos antes del evento; es decir, 125 representaciones por evento. Las coordenadas viene expresadas en metros y están calibradas considerando la esquina inferior izquierda del terreno de juego como el origen (valor 0,0).

Para cada partido, cada jugada de tiro a puerta y cada instante (126 instantes por jugada), el conjunto de datos tendrá tantos registros como jugadores presentes en el instante (habitualmente 22). El nombre y descripción de las variables contenidas en cada registro se describen a continuación:

- **IdSeason**: Año de inicio de la temporada de LLFP en la que se juega el encuentro.
- **IdGame**: Identificador único del partido de fútbol.
- **QtPitchLengthMeters**: Longitud en metros del terreno de juego.
- **QtPitchWidthMeters**: Anchura en metros del terreno de juego.
- **IdEvent**: Identificador único del evento (tiro a puerta).
- **IdEventType**: Contiene los valores "15" (no gol) o "16" (gol).
- **IdFrameEvent**: Identificador del instante en el que se realiza el disparo.
- **VlPitchDirectionShot**: Indicador de la portería donde se realiza el disparo: 1 (derecha), -1 (izquierda).
- **IdTeamShot**: Identificador único del equipo al que pertenece el jugador que ha hecho el disparo.
- **IdPlayerShot**: Identificador único del jugador que realiza el disparo.
- **IdFrame**: Identificador del instante de juego o frame al que se refiere el registro.
- **IdTeam**: Identificador único del equipo al que pertenece el jugador al que se refiere el registro.
- **IdPlayer**: Identificador único del jugador al que se refiere el registro.
- **NaPosition**: Posición del jugador al que se refiere el registro: "Defender", "Midfielder", "Forward" o "Goalkeeper".
- **NuJersey**: Número de dorsal del jugador al que ser refiere el registro.
- **VlRealX**: Posición en metros en el eje X del campo (longitud del campo) en la que se encuentra el jugador.

- **VlRealY**: Posición en metros en el eje Y del campo (anchura del campo) en la que se encuentra el jugador.
- **VlRealBallX**: Posición en metros en el eje X del campo (longitud del campo) en la que se encuentra el balón.
- **VlRealBallY**: Posición en metros en el eje Y del campo (anchura del campo) en la que se encuentra el balón.
- **VlRealBallZ**: Altura en metros a la que se encuentra el balón.
- **VlBallDistance**: Distancia en metros a la que se encuentra el balón respecto el jugador al que se refiere el registro.
- **VlPitchDirection**: Indicador de la portería del jugador al que se refiere el registro: 1 (derecha), -1 (izquierda).
- **Corner**: Indicador de si el tiro a puerta está en una jugada cuyo origen es un saque de esquina: 1 (si) o 0 (no).
- **FaltaIndirecta**: Indicador de si el tiro a puerta está en una jugada cuyo origen es una falta indirecta: 1 (si) o 0 (no).
- **Faltadirecta**: Indicador de si el tiro a puerta está en una jugada cuyo origen es una falta directa: 1 (si) o 0 (no).
- **Penalti:** Indicador de si el tiro a puerta es debido a un penalti: 1 (si) o 0 (no).

En la tabla 3.1 se muestra el formato y un ejemplo del contenido de las variables del conjunto de datos

En la figura 3.1 se muestra gráficamente un ejemplo del contenido del conjunto de datos, representando en un terreno de juego la posición de los jugadores y del balón en 16 instantes de juego anteriores a un disparo a puerta.

3.2. Flujo de proceso

El proceso de creación del sistema para predecir la probabilidad de marcar un gol, se ha dividido en una serie de pasos secuenciales, representados en la figura 3.2 y que se describen a continuación:

- **Preprocesamiento de datos**: En este paso se limpian el conjunto de datos eliminando datos erróneos y se transforman para adaptarlos al formato esperado por el siguiente paso del proceso. La serie de transformaciones contenidas en este paso tienen como objetivo representar cada instante de juego como una matriz bidimensional (asimilada a una imagen en escala de grises) que refleje la posición de los jugadores y del balón, diferenciados por rol: jugador que remata, portero defensor, jugador rival, jugador compañero y balón.

Variable	Formato	Ejemplo
IdSeason	Año de 4 posiciones	2021
IdGame	Entero de 7 posiciones	2219261
QtPitchLengthMeters	Flotante con 1 decimal	105,6
QtPitchWidthMeters	Flotante con 1 decimal	68,0
IdEvent	Numérico de 10 posiciones	2318743403
IdEventType	Numérico de 2 posiciones	15
IdFrameEvent	Numérico de 7 posiciones	1776696
VlPitchDirectionShot	Entero de 1 posición	-1
IdTeamShot	Numérico de 3 posiciones	855
IdPlayerShot	Entero de 6 posiciones	241153
IdFrame	Numérico de 7 posiciones	1776685
IdTeam	Numérico de 3 posiciones	855
IdPlayer	Numérico de 6 posiciones	2412345
NaPosition	Alfabético	Forward
NuJersey	Numérico de 2 posiciones	7
VlRealX	Flotante con 2 decimal	12,62
VlRealY	Flotante con 2 decimal	21,16
VlRealBallX	Flotante con 2 decimal	6,6
VlRealBallY	Flotante con 2 decimal	11,67
VlRealBallZ	Flotante con 2 decimal	0,34
VlBallDistance	Flotante con 2 decimal	13,51
VlPitchDirection	Entero de 1 posición	-1
Corner	Booleano	0
FaltaIndirecta	Booleano	0
Faltadirecta	Booleano	0
Penalti	Booleano	0

Tabla 3.1: Tabla con las variables que conforman el conjunto de datos principal a partir del cual que se desarrollará el trabajo de cálculo de la probabilidad de que un remate a puerta acabe en gol.

- **Extracción de características**: En este paso se pretenden representar cada imagen en escala de grises generada en el paso anterior, mediante un vector de menor dimensionalidad que recoja las características densas de los datos (llamadas representaciones latentes). Esto se consigue mediante el entrenamiento de un autoencoder sobre las imágenes en escala de grises y luego aplicando el encoder a las imágenes para obtener los vectores con las representaciones latentes.

- **Entrenamiento del modelo basado en RNN**: En este paso se entrena un modelo basado en RNN sobre las secuencias de representaciones latentes generadas en el paso anterior, configurado como un problema de clasificación binaria con el objetivo de predecir la probabilidad de que una jugada acabe o no en gol.

- **Calibración de las probabilidades**: En este paso se calibran las salidas del modelo entrenado en el paso anterior para que reflejen la probabilidad real de ocurrencia del evento de gol.

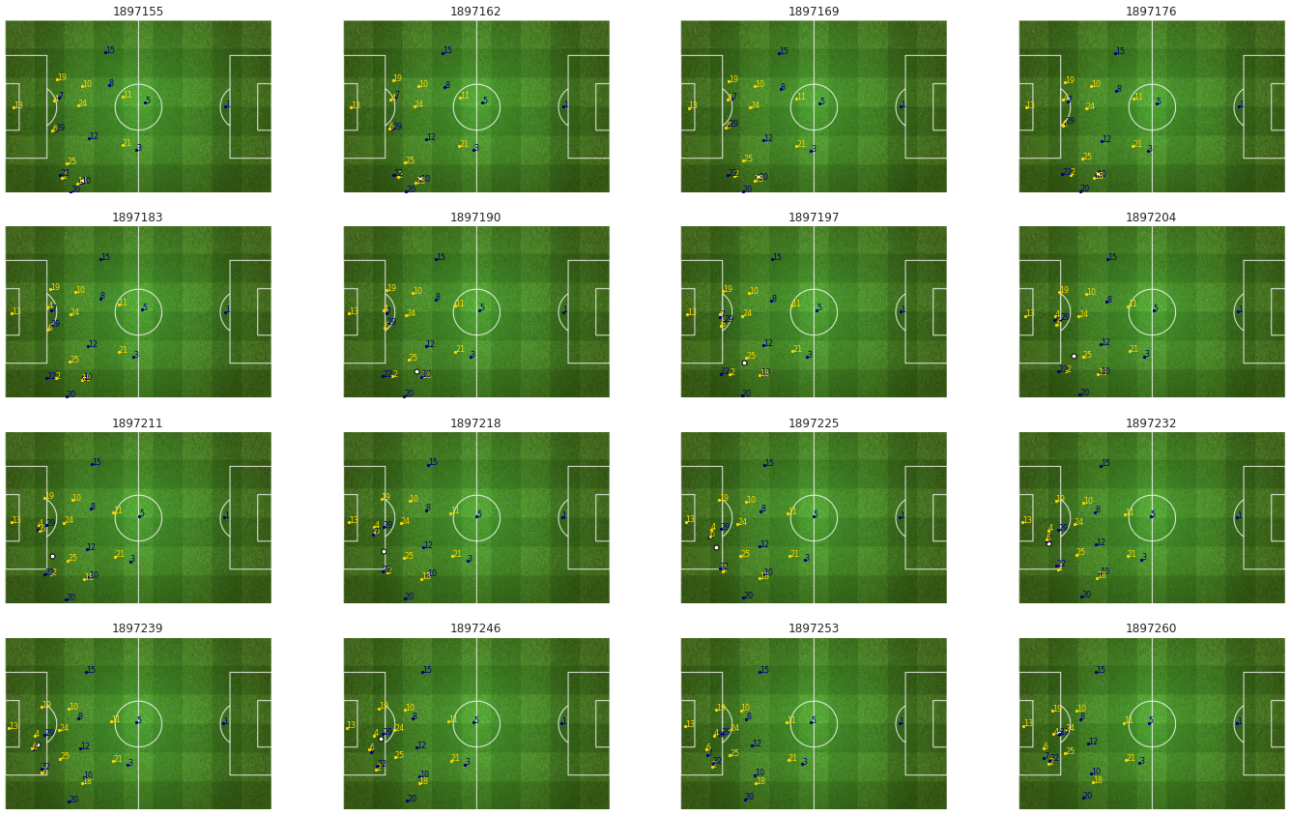

Figura 3.1: Figura con 16 instantes de ejemplo contenidos en los 5 segundos anteriores a un disparo a puerta, cada una representando la posición de los jugadores y del balón en el terreno de juego.

3.2.1. Paso1: Preprocesamiento de datos

En la fase de preprocesamiento, en primer lugar, se realizan las siguientes acciones sobre el conjunto de datos descrito en la sección 3.1.2:

- Se eliminan los registros con clave duplicada al considerarse erróneos. La clave del conjunto de datos es la concatenación de Identificador de Partido (IdTeam), Identificador de la Jugada (IdEvent), Identificador del Instante de juego (IdFrame) e Identificador de Jugador (IdPlayer).

- Se eliminan los registros asociados a jugadas referidas a penaltis, puesto que la posición de los jugadores en la secuencia de instantes de los 5 segundos anteriores a la ejecución del penalti, no aportan información relevante para predecir la probabilidad de que el disparo acabe en gol.

- Se retiran los registros asociados a jugadas que no contengan secuencias de 126 instantes (el instante correspondiente al disparo a puerta y los 125 asociados a los 5 segundos anteriores a dicho evento).

- Se transforman las posiciones de los jugadores y del balón, para considerar que todas las jugadas se realizan en la portería izquierda del campo de juego.

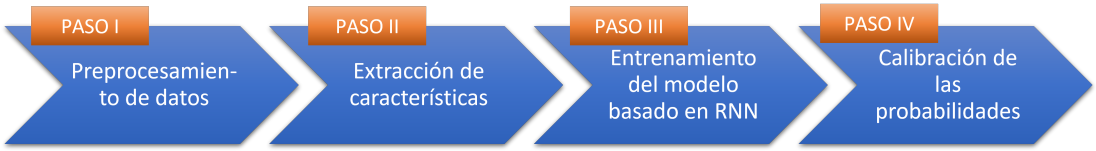

Figura 3.2: Figura que representa los pasos que componen el flujo del proceso de creación del sistema para calcular la probabilidad de que una jugada acabe en gol.

- Con el objetivo de prescindir de información no relevante, se eliminan los registros asociados a aquellos jugadores situados en la mitad de campo contrario a donde transcurre la jugada.

A continuación, para cada instante o frame, se crea una matriz bidimensional de tamaño 110x140, siendo cada celda la representación de un cuadrado de 0,5x0,5 m^2 en la que se divide la mitad del terreno de juego. Esta representación se puede asimilar a una imagen en escala de grises de dimensión 110x140 en donde cada celda representa un pixel.

Cuando un jugador o el balón esté situado en algún punto de la mitad izquierda del terreno de juego, se iluminará el pixel equivalente al cuadrado de 0,5 metros de lado en donde esté localizado. La intensidad del pixel será diferente en función del rol que tiene cada jugador en la secuencia:

- Si el jugador es el que dispara a puerta, se le asigna al pixel el valor 250.
- Si el jugador es el portero defensor, se le asigna al pixel el valor 225.
- Si el jugador es del equipo contrario al del jugador que dispara a puerta, se le asigna al pixel el valor 200.
- Si el jugador es del mismo equipo al del jugador que dispara a puerta, se le asigna al pixel el valor 175.
- Al pixel asociado al balón se le asigna un valor entre 125 y 150 en función de la altura registrada, a través de la siguiente fórmula:

$$Pixel_{balon} = 151 - 5max(0{,}2, min(z, 5))$$

, siendo z la altura del balón en metros. Con esta fórmula se restringe la altura del balón al intervalo $[0{,}2, 5]$, se multiplica por 5 para obtener valores en el rango $[1, 25]$ y finalmente se restan los valores a 151 para situarlos en el intervalo $[125, 150]$. De esta manera, un balón con una altura de 5 metros o superior será representado con el valor 125, un balón con una altura de 0,2 metros o inferior será representado con el valor 150 y un balón situado a una altura entre 0.2 y 5 metros tendrá asignado un valor entre 125 y 150.

Por tanto, los valores asignados a cada pixel sólo tienen como objetivo diferenciar el balón, su altura y los roles que desempeña cada jugador en el instante que representa la imagen en escala de grises.

En la figura 3.3 se muestran dos ejemplos de imágenes en escala de grises generadas en el paso de preprocesamiento y transformación de datos descrito anteriormente.

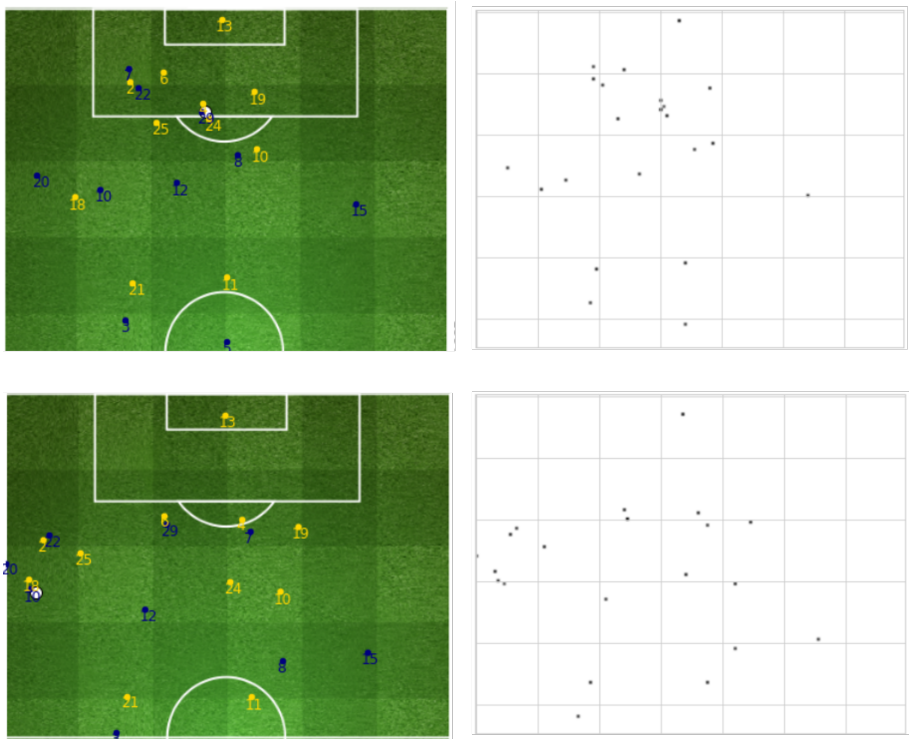

Figura 3.3: Figura en la que se muestra la representación de dos instantes de juego y imagen en escala de grises equivalente generada durante la etapa de preprocesamiento y transformación de datos del flujo del proceso del trabajo.

Finalmente, se etiqueta cada secuencia de las 126 representaciones bidimensionales pertenecientes a una jugada como gol o no gol, para ello se crea un nuevo conjunto de datos asignando para cada evento (jugada) el valor '1' si el disparo acabó en gol, o el valor '0' si el disparo no acabó en gol.

3.2.1.1. Inclusión de filtro gaussiano a las representaciones bidimensionales

Como se puede apreciar en la figura 3.3, las imágenes en escala de grises generadas, tienen una dimensión de 110x140, lo que supone 15.400 píxeles, de los cuales sólo un máximo de 23 (correspondientes a los jugadores y el balón siempre que estén en la mitad del campo donde transcurre la jugada) serán iluminados.

Esta situación hace muy difícil al autoencoder de la etapa de extracción de características aprender las representaciones latentes de la imagen en escala de grises, ya que la mayor parte del valor de los píxeles son ceros.

Para mitigar este problema, se aplica un filtro gaussiano a cada imagen para ampliar el número de pixeles diferentes de cero, iluminando aquellos situados alrededor de los puntos que representan a los jugadores y al balón.

Los filtros gaussianos se utilizan para difuminar imágenes y consisten en aplicar un determinado tipo de filtro convolucional a la imagen (ver sección 2.2.3); es decir, una matriz numérica de baja dimensinalidad que va recorriendo la imagen para combinar la intensidad de los píxeles con los valores de la matriz y generar un nuevo valor para cada pixel de la imagen original.

El filtro convolucional (también llamado kérnel) y la técnica utilizada en este caso para generar la imagen difuminada, sigue los siguientes pasos:

- Se va recorriendo cada pixel de la imagen original tomando porciones alrededor del pixel de dimensión igual a la de la matriz del filtro.
- Se calcula la media ponderada de los valores de los píxeles de cada porción.
- Los pesos que se utilizan en el cálculo de la media ponderada se toman de la matriz del filtro, los cuales siguen una distribución normal $N(\mu, \sigma)$ bidimensional, siendo la media μ igual al valor del pixel central de la porción tomada, y la desviación típica σ igual a un valor fijo predefinido.
- El resultado de la media ponderada será el nuevo valor del pixel central de la porción tomada.
- Como se va recorriendo la imagen original aplicando los pasos anteriores a cada pixel, el resultado será una nueva imagen de igual dimensión, mostrando la imagen original difuminada.

Tras realizar varios experimentos ejecutando el flujo completo del proceso de creación del sistema del presente trabajo pero haciendo variar el valor de σ del filtro gaussiano, se comprueba que el mejor rendimiento del modelo de predicción de gol se obtiene con $\sigma = 2$[3].

Por otro lado, la aplicación del filtro gaussiano reduce la intensidad de los píxeles originalmente iluminados al transformarlos en una media ponderada de los pixeles que lo rodean, los cuales en su gran mayoría son cero. Esto hace que, en parte, se pierda la diferenciación por roles de la posición de los jugadores y del balón establecida a través de las diferentes intensidades de los píxeles, afectando al rendimiento del modelo de predicción de gol. Para mitigar este problema y tras la ejecución de varios experimentos, se comprueba que el rendimiento del modelo de predicción de gol es superior si se fusiona la imagen difuminada mediante el filtro gaussiano con la imagen original. Esto se implementa sustituyendo en la imagen difuminada los píxeles iluminados de la imagen original.

En la figura 3.4 se muestran dos ejemplos de imágenes en escala de grises a las que se les ha aplicado un filtro gaussiano con $\sigma = 2$ y se han fusionado con la imagen original.

Finalmente, se normalizan las intensidades de los píxeles de cada imagen para que estén comprendidos entre 0 y 1, dividiendo los valores entre el máximo (250).

[3] En la librería utilizada para aplicar el filtro gaussiano (Scipy), el tamaño del mismo viene determinado por defecto a partir del valor de σ mediante la fórmula: $2\sigma + 1$

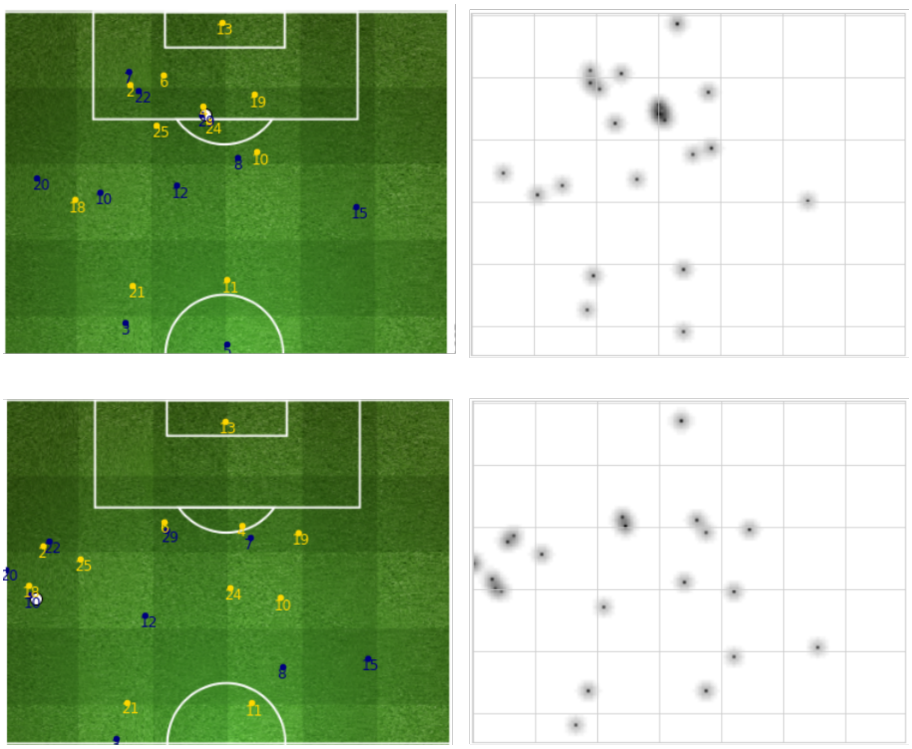

Figura 3.4: Figura en la que se muestra la representación de dos instantes de juego y su imagen en escala de grises equivalente a la que se ha aplicado un filtro gaussiano de varianza 2.

3.2.2. Paso2: Extracción de características de cada jugada

En base a lo descrito en la sección 2.2.5, en la fase de extracción de características, se entrena un autoencoder con el objetivo de representar cada imagen en escala de grises asociada a un instante de juego, con un vector de mucha menor dimensionalidad que la imagen original, pero que condense toda la información de la imagen extrayendo sus características latentes.

En la figura 3.5 se muestra gráficamente el objetivo de este paso del flujo del proceso, donde se representa cada imagen en escala de grises con sus características latentes.

3.2.2.1. Modelos de autoencoder

Para el trabajo se diseñan y prueban varias arquitecturas de autoencoder, para finalmente experimentar con las dos que se describen a continuación:

Autoencoder no convolucional

Autoencoder no convolucional con las siguientes características:

- Las entradas y salidas del sistema serán las imágenes con dimensión 110x140.
- Encoder:

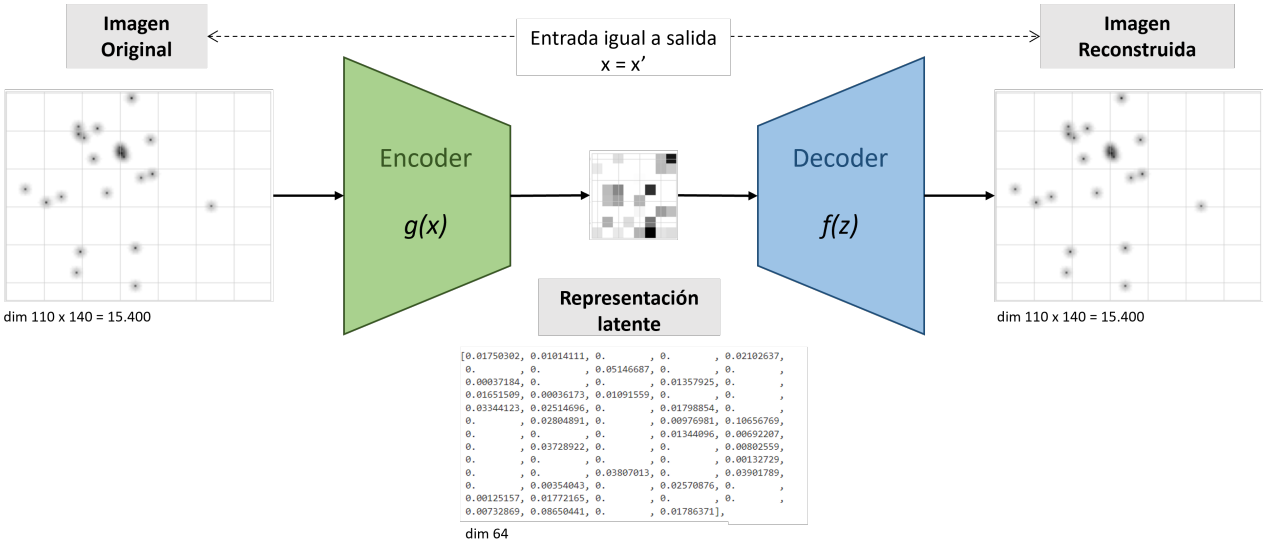

Figura 3.5: Figura que representa la arquitectura del autoencoder utilizado en el trabajo para extraer las representaciones latentes de la imágenes en escala de grises de cada situación de partido.

- Está compuesto de 4 capas densas de 512, 256, 128 y 64 neuronas, todas ellas con función de activación "Selu" [4].
- Esto supone que consta de 8.057.792 parámetros que se calcularán durante el entrenamiento.
- La salida del encoder será un vector de longitud 64, correspondiente a la representación latente de cada imagen.
- En la tabla 3.2 se resume la arquitectura del encoder:

Capa (tipo)	Dimensión Salida	# Parámetros
flatten (Flatten)	(None, 15400)	0
dense (Dense)	(None, 512)	7.885.312
dense_1 (Dense)	(None, 256)	131.328
dense_2 (Dense)	(None, 128)	32.896
dense_3 (Dense)	(None, 64)	8.256

Tabla 3.2: Tabla con la descripción de la capas que componen el encoder del autoencoder no convolucional.

- Decoder:

[4]La función de activación "Selu" (scaled exponential linear unit), atiende a la siguiente fórmula:

$$selu(x) = \lambda \begin{cases} x & x > 0 \\ \alpha e^x - \alpha & x \leq 0 \end{cases}$$

, siendo $\lambda \approx 1{,}0507$y $\alpha \approx 1{,}6733$.

- La entrada del decoder será el vector de características latentes de longitud 64 generado por el encoder.
- De manera simétrica al encoder, el decoder está compuesto de 4 capas densas de 128, 256, 512 y 15.400 (110 x140) neuronas, todas ellas con función de activación "Selu", salvo la última capa que no tienen función de activación.
- Esto supone que consta de 8.073.128 parámetros que se calcularán durante el entrenamiento
- En la tabla 3.3 se resume la arquitectura del decoder:

Capa (tipo)	**Dimensión Salida**	**# Parámetros**
dense_4 (Dense)	(None, 128)	8.320
dense_5 (Dense)	(None, 256)	33.024
dense_6 (Dense)	(None, 512)	131.584
dense_7 (Dense)	(None, 15400)	7.900.200
reshape_6 (Reshape)	(None, 110, 140)	0

Tabla 3.3: Tabla con la descripción de las capas que componen el decoder del autoencoder no convolucional.

- Por tanto, el autoencoder se compone de 16.130.920 parámetros que se calcularán durante el entrenamiento.
- El optimizador elegido que demuestra un mejor rendimiento en los experimentos realizados es el "Adam" (introducido en Kingma and Ba (2014)) con tasa de aprendizaje (o learning rate) inicial igual a 0.0001. El optimizador determina a que tasa modificar los valores de los parámetros a partir del gradiente calculado sobre la función de pérdida. El optimizador Adam va ajustando dinámicamente el valor de la learning rate a aplicar en cada iteración del entrenamiento.
- La función de pérdida elegida es la media de los errores al cuadrado (mean squared error o MSE) la cual mide el error al cuadrado cometido por el autoencoder en los valores de los píxeles a la hora de reconstruir la imagen original, mediante la siguiente fórmula:

$$MSE = \sum_{i=1}^{N_h} \sum_{j=1}^{N_w} \frac{(p_x(i,j) - p_{x'}(i,j))^2}{N_h N_w}$$

,siendo $p_x(i,j)$ y $p_{x'}(i,j)$ los valores de los píxeles de la imagen original y de la imagen reconstruida, y N_h, N_w las dimensiones de la imágenes.

- La estructura final de las capas que componen el autoencoder se muestra gráficamente en la figura 3.6.

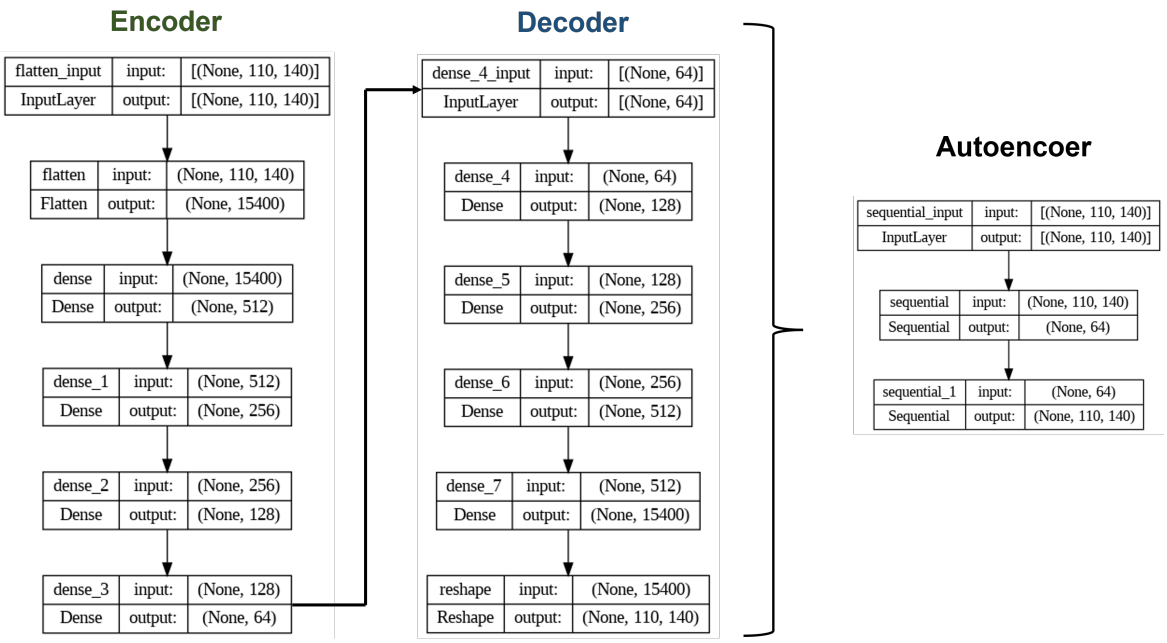

Figura 3.6: Figura que muestra la estructura de las capas del autoencoder no convolucional utilizado en el trabajo para extraer las representaciones latentes de las imágenes en escala de grises de cada instante de partido.

Autoencoder convolucional

Puesto que se está tratando con imágenes en escala de grises, también se diseña una arquitectura de autoencoder basada en capas convolucionales con el objetivo de que la representación latente de cada imagen recoja, no sólo las relaciones lineales que aprende una autoencoder no convolucional con capas densas, sino también las características espaciales intrínsecas de cada imagen.

Tras distintos experimentos, la arquitectura y configuración del autoencoder convolucional elegido se describe a continuación:

- Las entradas y salidas del sistema serán las imágenes con dimensión 110x140.
- Encoder:
  - Está compuesto por 4 capas convolucionales con 8, 16, 32 y 64 filtros de dimensión 3x3, intercaladas con sus respectivas capas de agrupación para ir reduciendo la dimensión de las imágenes hasta dejarlas en 1x1.
  - Todas las capas convolucionales contienen la función de activación "Relu" [5].
  - Esto supone que consta de 24.384 parámetros que se calcularán durante el entrenamiento.
  - La salida del encoder será un vector de longitud 64, correspondiente a la representación latente de cada imagen.
  - En la tabla 3.4 se resume la arquitectura del encoder:

Capa (tipo)	Dimensión Salida	# Parámetros
reshape (Reshape)	(None, 110, 140, 1)	0
conv2d (Conv2D)	(None, 110, 140, 8)	80
max_pooling2d (MaxPooling 2D)	(None, 55, 35, 8)	0
conv2d_1 (Conv2D)	(None, 55, 35, 16)	1.168
max_pooling2d_1 (MaxPooling 2D)	(None, 11, 7, 16)	0
conv2d_2 (Conv2D)	(None, 11, 7, 32)	4.640
max_pooling2d_2 (MaxPooling 2D)	(None, 3, 2, 32)	0
conv2d_3 (Conv2D)	(None, 3, 2, 64)	18.496
max_pooling2d_3 (MaxPooling 2D)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0

Tabla 3.4: Tabla con la descripción de la capas que componen el encoder del autoencoder convolucional.

- Decoder:
  - La entrada del decoder sera el vector de características latentes de longitud 64 generado por el encoder, el cual primeramente será redimensionado a una imagen de 1x1x64 para que pueda ser tratado por las capas convolucionales.
  - De manera simétrica al encoder, el decoder está compuesto de 4 capas convolucionales traspuestas de 32, 16, 8 y 1 filtros de dimensión 3x3, en donde se va indicando un "stride" que permite ir ampliando la imagen desde 1x1 a la dimensión original de 110x140.
  - Cada capa convolucional, se acompaña de una capa de normalización "Batch Normalization". Esta capa estandariza a una distribución normal (normaliza) las salidas de la capa anterior (antes de aplicar la función de activación). Se ha demostrado que la técnica estabiliza el proceso de aprendizaje y ayuda a reducir el número de épocas requeridas para entrenar redes CNN.
  - Todas las capas convolucionales contienen la función de activación "Relu", salvo la última capa que tiene como función de activación la llamada sigmoide ("Sigmoid")[6]. Esto es así puesto que, en este caso, se trata la tarea de reconstrucción de imágenes como un problema de clasificación binaria sobre cada píxel (0 no iluminado, 1 iluminado) utilizando como función de pérdida la "Binary Cross Entropy" (BCE), en lugar de un problema de regresión sobre el valor de los píxeles minimizando la función MSE como se hace con el autoencoder no convolucional. El uso de la función de activación sigmoide asegura que la salida del autoencoder esté comprendida entre 0 y 1 y así poder optimizar la función de perdida BCE. El motivo de afrontar el problema como un ejercicio de clasificación binaria

[5]La función de activación "Relu" (rectified linear unit), atiende a la siguiente fórmula: $relu(x) = max(0, x)$.
[6]La función de activación "Sigmoid" atiende a la fórmula: $Sigmoid(x) = \frac{1}{1+e^x}$.

en lugar de un problema de regresión, es debido a que mediante esta estrategia también se consigue entrenar el autoencoder y además se acelera y estabiliza el proceso, según se indica en Creswell et al. (2017).

- Con la arquitectura descrita, el decoder consta de 24.545 parámetros que se calcularán durante el entrenamiento.
- En la tabla 3.5 se resume la arquitectura del decoder:

Capa (tipo)	Dimensión Salida	# Parámetros
reshape_1 (Reshape)	(None, 1, 1, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 11, 7, 32)	18.464
batch_normalization (BatchNormalization)	(None, 11, 7, 32)	128
conv2d_transpose_1 (Conv2DTranspose)	(None, 55, 35, 16)	4.624
batch_normalization_1 (BatchNormalization)	(None, 55, 35, 16)	64
conv2d_transpose_2 (Conv2DTranspose)	(None, 110, 140, 8)	1.160
batch_normalization_2 (BatchNormalization)	(None, 110, 140, 8)	32
conv2d_transpose_3 (Conv2DTranspose)	(None, 110, 140, 1)	73
reshape_2 (Reshape)	(None, 110, 140)	0

Tabla 3.5: Tabla con la descripción de las capas que componen el decoder del autoencoder convolucional.

- Por tanto el autoencoder se compone de 48.929 parámetros que se calcularán durante el entrenamiento, lo que supone una reducción muy considerable de parámetros respecto al autoencoder no convolucional.
- El optimizador elegido para este autoencoder también es el "Adam" con tasa de aprendizaje inicial igual a 0.0001.
- Como se ha indicado anteriormente, la función de pérdida elegida a optimizar durante el entrenamiento es la BCE, la cual mide el error entre la salida de la función de activación sigmoidal y el valor del pixel (el cual ya se había normalizado en la fase anterior para estar comprendido entre 0 y 1), según la siguiente fórmula:

$$BCE = -\sum_{i=1}^{N_h}\sum_{j=1}^{N_w} \frac{p_x(i,j)log(p_{x'}(i,j)) + (1-p_x(i,j))log(1-p_{x'}(i,j))}{N_h N_w}$$

, siendo $p_x(i,j)$ los valores de los píxeles de la imagen original, $p_{x'}(i,j)$ los valores que arroja el autoencoder tras pasar por la última función de activación sigmoidal, y N_h, N_w las dimensiones de la imágenes.

- La estructura final de las capas que componen el autoencoder convolucional se muestra gráficamente en la figura 3.7:

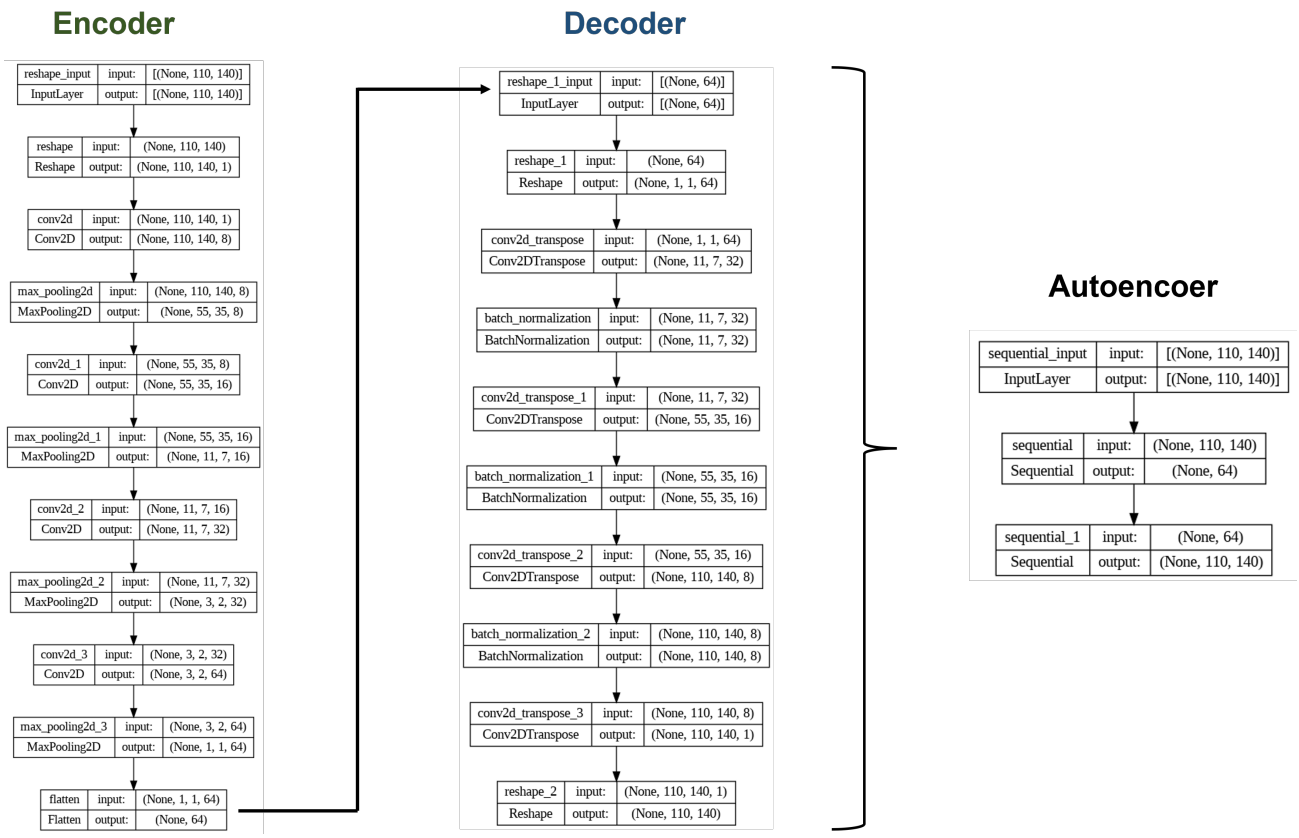

Figura 3.7: Figura que muestra la estructura de las capas del autoencoder convolucional utilizado en el trabajo para extraer las representaciones latentes de las imágenes en escala de grises de cada instante de partido.

3.2.2.2. Entrenamiento del autoencoder y generación de las representaciones latentes de las imágenes

Para reducir y simplificar el tiempo de entrenamiento de los autoencoders se toma 1 de cada 5 imágenes de cada jugada, ya que entre imagen e imagen sólo transcurren 0,04 segundos y, por tanto, la variación de la posición de los jugadores de un imagen a otra cuando estas son consecutivas, no es significativa. Al tomar 1 de cada 5 imágenes se reduce el conjunto de datos de entrenamiento sin perder información relevante, al estar considerando las distintas situaciones de juego ocurridas cada 0,2 segundos.

El entrenamiento se realizará en lotes de 32 instancias y durante 5 épocas, ya que la función de pérdida se estabiliza en el mínimo a partir de la tercera época. Una vez entrenados los autoencoders se hacen pasar todas las imágenes en escala de grises por el encoder, con el objetivo de generar los vectores de representaciones latentes de dimensión 64.

3.2.3. Paso 3: Modelos de predicción basados en RNNs

Una vez generadas las representaciones latentes de cada imagen en escala de grises mediante el paso anterior, se agrupan en jugadas (secuencias de 126 imágenes) y se realiza el entrenamiento de

modelos basados en RNNs, diseñados para que arrojen la probabilidad de que una jugada acabe o no en gol.

3.2.3.1. Modelos de RNN

Para el trabajo se diseña y experimenta con varias arquitecturas de Redes Neuronales Recurrentes, para finalmente seleccionar las dos que se describen a continuación:

Red Neuronal Recurrente basada en GRUs

Red Neuronal Recurrente con capas ocultas compuesta por neuronas de tipo GRU (ver sección 2.2.2) con las siguientes características:

- La entrada del sistema serán secuencias de 126 vectores (representaciones latentes de cada imagen), por lo que tendrán una dimensión 126 x 64.
- Se compone de una primera capa de 128 unidades GRU con función de activación tangencial hiperbólica [7], y cuya salida serán también secuencias de los vectores generados por las neuronas GRU.
- Una segunda capa de 128 unidades GRU y con función de activación tangencial hiperbólica. La entrada son las secuencia de vectores de 128 valores generados por la capa anterior y su salida es un único vector de 128 valores correspondiente al último instante temporal de la secuencia.
- Una última capa densa de 1 neurona con función de activación sigmoidal, para que la salida sea un número entre 0 y 1, representando la probabilidad de que la secuencia de entrada pertenezca a una jugada que acaba en gol.
- Esta arquitectura de red consta de 173.697 parámetros que se calcularán durante el entrenamiento.
- En la tabla 3.6 se resume la arquitectura de la red:

Capa (tipo)	Dimensión Salida	# Parámetros
gru (GRU)	(None, 126, 128)	74.496
gru_1 (GRU)	(None, 128)	99.072
dense_1 (Dense)	(None, 1)	129

Tabla 3.6: Tabla con la descripción de las capas que componen la Red Neuronal Recurrente basada en unidades GRU.

[7] La función de activación tangencial hiperbólica, atiende a la siguiente fórmula: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

- El optimizador elegido para la RNN también es el "Adam" con tasa de aprendizaje inicial igual a 0.0001.
- Como el presente trabajo se enfrenta a un problema de clasificación binaria, la función de pérdida elegida a optimizar durante el entrenamiento es la BCE, la cual mide el error entre la salida de la función de activación sigmoidal y el valor de la etiqueta de la jugada (0 no gol, 1 gol).
- La estructura de las capas que componen la red se muestra gráficamente en la figura 3.8.

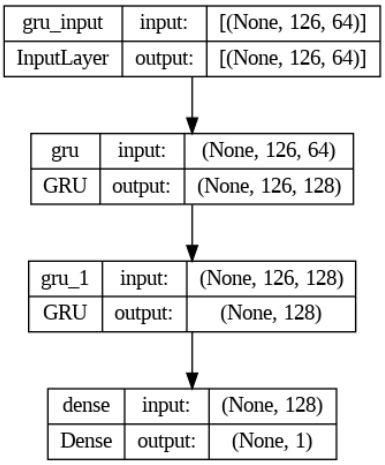

Figura 3.8: Figura que muestra la estructura de las capas de la red Red Neuronal Recurrente basada en unidades GRU utilizada en el trabajo para calcular la probabilidad de gol de una jugada.

Red Neuronal Recurrente basada en LSTMs

Red Neuronal Recurrente con la misma arquitectura que la descrita anteriormente, pero con capas ocultas compuestas por neuronas de tipo LSTM (ver sección 2.2.2):

- La entrada del sistema serán secuencias de 126 vectores por lo que tendrán una dimensión de 126 x 64 (o 126 x 32).
- Se compone de dos capas de 128 unidades LSTM con función de activación tangencial hiperbólica, y una última capa densa de 1 neurona con función de activación sigmoidal.
- Esta arquitectura de red consta de 230.529 parámetros que se calcularán durante el entrenamiento.
- En la tabla 3.7 se resume la arquitectura de la red:
- El optimizador también será el "Adam" con tasa de aprendizaje inicial igual a 0.0001 y la función de pérdida la BCE.

Capa (tipo)	Dimensión Salida	# Parámetros
lstm (LSTM)	(None, 126, 128)	98.816
lstm_1 (LSTM)	(None, 128)	131.584
dense_1 (Dense)	(None, 1)	129

Tabla 3.7: Tabla con la descripción de las capas que componen la Red Neuronal Recurrente basada en unidades LSTM.

- La estructura de las capas que componen la red se muestra gráficamente en la figura 3.9.

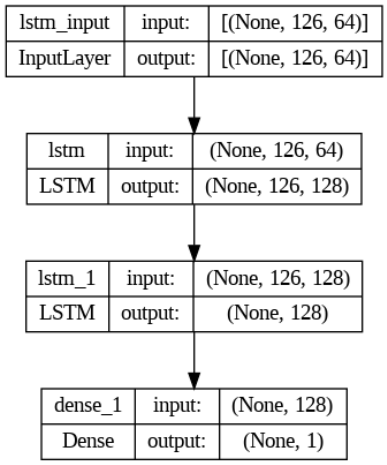

Figura 3.9: Figura que muestra la estructura de las capas de la red Red Neuronal Recurrente basada en unidades LSTM utilizada en el trabajo para calcular la probabilidad de gol de una jugada.

3.2.3.2. Entrenamiento de la RNN

Según se describe más adelante en la sección 4.1.1, dado que los goles son eventos raros en el fútbol, el conjunto de datos disponible para el entrenamiento está fuertemente desbalanceado respecto a la variables objetivo (28.246 jugadas, de las cuales sólo el 9 % fueron etiquetadas como goles).

Según se indica en Japkowicz and Stephen (2002), entrenar un algoritmo sobre un conjunto de datos con pocas instancias de una clase, tiende a clasificar todas las observaciones como la clase mayoritaria, lo que significa que el modelo a menudo es incapaz de generalizar bien el comportamiento de la clase minoritaria. Por tanto, este desequilibrio influiría en gran medida en el rendimiento o precisión predictiva del modelo.

Los métodos habitualmente utilizados para tratar con el problema de desbalanceo de clases en el conjunto de datos de entrenamiento, se pueden clasificar de la siguiente manera (Mollineda et al. (2007)):

- **Métodos de remuestreo para equilibrar el conjunto de datos**: consiste en variar la cantidad de ejemplares de las clases para que el conjunto resulte más balanceado. Estos métodos,

a su vez, pueden agruparse en dos tipos:

- **Submuestreo** (undersampling): consiste en reducir el tamaño de la clase mayoritaria eliminando observaciones al azar hasta que el conjunto de datos esté balanceado. En un conjunto de datos desbalanceado, a menudo es realista suponer que muchas observaciones de la clase mayoritaria son redundantes y que al eliminar algunas de ellas al azar, la distribución de datos no cambiará significativamente. Sin embargo, con esta técnica existe el riesgo de eliminar observaciones relevantes respecto a la capacidad predictiva del modelo, ya que la extracción se realiza de forma no supervisada. Se recomienda aplicar estrategias de submuestreo cuando el nivel de desequilibrio entre las clases es bajo.
- **Sobremuestreo** (oversampling): consiste en aumentar el tamaño de la clase minoritaria aleatoriamente, replicando instancias de dicha clase hasta que las dos clases tengan igual frecuencia. El sobremuestreo no agrega instancias con información adicional al modelo y aumenta el riesgo de sobreajuste (overfitting) por quedar el modelo sesgado hacia la clase minoritaria. Otro inconveniente de este enfoque es que aumentan los tiempos de entrenamiento de los algoritmos, lo que puede ser particularmente ineficaz cuando el conjunto de datos original es muy grande. Se recomienda aplicar técnicas de sobremuestreo cuando el conjunto de datos tiene muy pocos ejemplos de la clase minoritaria.

  Una técnica muy extendida de sobremuestreo es SMOTE (Chawla et al. (2002)), la cual sobremuestrea la clase minoritaria generando instancias sintéticas a partir de otras instancias próximas del conjunto de datos. La idea es formar nuevas instancias interpolando entre ejemplos de la clase minoritaria, por lo que se crean grupos alrededor de cada observación de dicha clase.

- **Modificación de los algoritmos existentes**: consiste en adaptar los modelos y técnicas existentes a las características especiales de los conjuntos de datos desequilibrados. Dentro de este grupo, vale la pena mencionar:

  - El aprendizaje sensible a la pérdida (cost-sensitive learning), que trata de ponderar de manera diferente los errores de la función de pérdida en función de la distribución de cada clase.
  - Los clasificadores de una clase (one-class classifiers), que tratan de modelar una clase de objetos (la clase objetivo) la cual se asume que está bien muestreada, y distinguirla de todos los demás objetos posibles (la clase atípica). La clase objetivo suele ser la clase minoritaria.
  - Los conjuntos de clasificadores (classifiers ensembles), que combinan los resultados de varios clasificadores, cada uno entrenado después de sobremuestrear o submuestrear el conjunto de datos con diferentes tasas de sobremuestreo o submuestreo.

- También dentro de esta categoría se pueden englobar las técnicas que asignan diferentes pesos a las instancias de cada clase para compensar el desequilibrio en el conjunto de datos, pero sin alterar realmente la distribución de clases.

En el presente trabajo, debido a que el conjunto de datos tiene muy pocas instancias de la clase minoritaria (sólo el 9 % de las jugadas son etiquetadas como gol), se opta por utilizar como técnica de equilibrado un sobremuestreo aleatorio, la cual duplica aleatoriamente instancias de la clase minoritaria hasta conseguir balancear el conjunto de datos de entrenamiento.

Finalmente indicar que el entrenamiento se realizará en lotes de 32 instancias y durante 20 épocas, a partir de las cuales se observa que se estabiliza el aprendizaje de los modelos.

3.2.3.3. Evaluación rendimiento del modelo

La mayoría de las medidas de rendimiento de modelos que tratar de resolver problemas de clasificación binaria como al que se enfrenta el presente TFM, se construyen sobre una matriz de confusión de 2×2 como se ilustra en la tabla 3.8.

		Real	
		Clase Positiva (Gol)	**Clases Negativa (No-Gol)**
Predicción	**Clase Positiva (Gol)**	Verdaderos Positivos (TP)	Falsos Positivos (FP)
	Clase Negativa (No-Gol)	Falsos Negativos (FN)	Verdaderos Negativos (TN)

Tabla 3.8: Tabla que muestra la definición de los elementos de una matriz de confusión 2x2.

La matriz de confusión, muestra cuatro medidas simples:

- **Verdaderos Positivos** (TP): indican el número de jugadas que acabaron en gol que el modelo ha clasificado correctamente.
- **Verdaderos Negativos** (TN): indican el número jugadas que no acabaron en gol que el modelo ha clasificado correctamente.
- **Falsos Positivos** (FP): indican el número de jugadas que acabaron en gol que el modelo ha clasificados erróneamente.
- **Falsos Negativos** (FN): indican el número de jugadas que no acabaron en gol que el modelo ha clasificados erróneamente.

A partir de estas 4 medidas simples, el rendimiento de los modelos de clasificación binaria habitualmente se calcula usando las siguientes métricas:

- **Precisión**: $\frac{TP}{TP+FP}$
- **Sensibilidad** (Recall): $\frac{TP}{TP+FN}$

- **Especificidad** (Specificity): $\frac{TN}{FP+TN}$
- **F1-Score**: $2\frac{Precisión x Recall}{Precisión+Recall}$
- **Exactitud** (Accuracy): $\frac{TP+TN}{TP+FN+TN+FP}$
- **Error**: $1 - Accuracy$

Sin embargo, ha sido ampliamente demostrado (por ejemplo en Daskalaki et al. (2006); Huang and Ling (2005); Landgrebe et al. (2006)) que, cuando la distribución de las clases está desbalanceada, estas medidas no son apropiadas porque no consideran los costes de clasificación errónea y están fuertemente sesgadas a favorecer la clase mayoritaria. En el caso concreto del presente trabajo, sólo el 9 % de las instancias del conjunto de datos disponible están etiquetadas como goles y si se diseña una estrategia de clasificación que asigne a todas las instancias como negativas (no gol) se obtendría una medida de exactitud del 91 % aunque esté fallando por completo a la hora de predecir los goles.

Por tanto, en entornos con datos desbalanceados, se necesitan métricas alternativas que midan el rendimiento de clasificación en las clases positivas y negativas de forma independiente. Una forma de combinar las tasas de TP y FP consiste en utilizar la curva ROC (Receiver Operating Characteristic). La curva ROC es un gráfico bidimensional para visualizar, organizar y seleccionar clasificadores en función de su desempeño, ya que describe el compromiso entre aciertos (verdaderos positivos) y fallos (falsos positivos) del clasificador. En la curva ROC, la tasa de verdaderos positivos o sensibilidad se representa en el eje Y y la tasa de falsos positivos o 1-especificidad en el eje X. Para dibujar la curva ROC deben tenerse en cuenta varios umbrales de clasificación sobre los que calcular la tasa de TP y FP.

La curva ROC se puede interpretar de la siguiente manera:

- El punto inferior izquierdo (0,0) representa el caso donde el modelo clasifica todos los ejemplos como negativos.
- El punto superior derecho (1,1) representa el caso donde el modelo clasifica todos los ejemplos como positivos.
- El punto (0,1) representa la perfecta clasificación, indicando un 100 % de sensibilidad (ningún falso negativo) y un 100 % de especificidad (ningún falso positivo).
- La diagonal $y = x$ define una clasificación totalmente aleatoria y divide el espacio ROC: los puntos por encima de la diagonal representan buenos resultados de clasificación (mejor que el azar) y los puntos por debajo de la línea los resultados pobres de clasificación (peor que al azar).

Para evaluar el rendimiento general de un clasificador, se puede medir la fracción del área total que cae bajo la curva ROC, también llamada Area Under the Curve o AUC (Huang and Ling (2005)). La

AUC varía entre 0 y 1, y los valores de AUC más grandes indican generalmente un mejor rendimiento del clasificador. La AUC se puede interpretar como la probabilidad de que el clasificador asigne una puntuación más alta a una muestra positiva elegida aleatoriamente respecto a una muestra negativa.

Adicionalmente, el objetivo de este TFM no es predecir correctamente si las jugadas van a acabar en gol o no, sino calcular la probabilidad de anotar gol de cada jugada y así ofrecer una medida de dificultad de acierto de las jugadas que permita poder comparalas entre si. Por lo tanto, en este sentido, la AUC parece una buena métrica de rendimiento de los modelos del TFM puesto que indicará, con independencia del umbral de predicción seleccionado, que el modelo es capaz de asignar puntuaciones altas a jugadas que realmente acabaron en gol respecto a las que finalmente no acabaron en gol.

Por otro lado, según se indica en Saito and Rehmsmeier (2015), debido a la propia definición de la curva ROC (concretamente debido a la tasa de falsos positivos en donde aparece TN en el denominador), se recomienda el uso de la curva Precision-Recall (PRC) y su área Under the Curve (AUC-PR) para medir el rendimiento de modelos sobre conjuntos de datos altamente desbalanceados. Esta curva, como su propio nombre indica, relaciona la métrica Precisión (representada en el eje de Y) frente a Sensibilidad (representadas en el eje X), reflejando el compromiso de ambas métricas en los distintos umbrales de clasificación definidos sobre los valores arrojados por el clasificador.

La curva PRC se puede interpretar de la siguiente manera:

- Los puntos en el cuadrante superior izquierdo (baja sensibilidad pero alta precisión) representan que el modelo predice pocos positivos pero todos han sido correctamente clasificados (pocos falsos positivos pero muchos falsos negativos).
- Los puntos en el cuadrante inferior derecho (alta sensibilidad pero baja precisión) representa que el modelo predice muchos positivos pero gran parte de ellos están mal clasificados (pocos falsos negativos pero muchos falsos positivos).
- El punto (1,1) representa la perfecta clasificación: 100 % de sensibilidad (ningún falso negativo) y 100 % de precisión (ningún falso positivo).
- En el caso concreto del presente trabajo, la recta $precisión = 0{,}09$ define una clasificación básica donde todas las instancias se clasifican como positivas (gol), debido a que la proporción de instancias etiquetadas como goles en el conjunto de datos de entrenamiento es del 9 %. Una AUC-PR superior al 9 % indicará que el clasificador tiene un rendimiento mayor que el clasificador básico, siendo un AUC-PR de 1 el indicador de que un modelo consigue una perfecta clasificación.

En la figura 3.10 se muestran ejemplos de curvas ROC y PR, y como interpretarlas.

También se calcula como métrica de evaluación de rendimiento de los experimentos la BCE, que es usada por los modelos de clasificación como función de pérdida durante el entrenamiento y es también agnóstica a los umbrales de clasificación.

Figura 3.10: Figura que muestra la forma habitual de las curvas ROC y PR y sus principales interpretaciones, obtenidas de https://en.wikipedia.org/wiki/Receiver_operating_characteristic y de https://analyticsindiamag.com/complete-guide-to-understanding-precision-and-recall-curves/ respectivamente.

Como resultado de lo comentado anteriormente, se eligen como métricas principales para evaluar el rendimiento de los modelos del presente trabajo la AUC-ROC, la AUC-PR y la BCE.

Una vez dicho esto, aunque no se usen para determinar cual modelo ofrece mejor rendimiento, las métricas Precisión, Sensibilidad y F-Score también tienen valor ya que dan información sobre hasta qué punto el clasificador es realmente capaz de otorgar valores altos a las oportunidades de gol claras, por lo que también se ofrecerán asociadas a cada experimento.

3.2.4. Paso 4: Calibración de las probabilidades

Como se explicó en la sección 2.2.6, los resultados de un modelo de clasificación se interpretan como la probabilidad de asignación o grado de confianza de que una cierta jugada pertenezca a una determinada clase (gol o no gol), en lugar de la probabilidad real de ocurrencia del propio evento. El proceso de transformar las salida del modelo a probabilidad de ocurrencia del evento se denomina calibración.

Debido al fuerte desbalanceo entre clases del conjunto de datos de entrenamiento y a la estrategia de rebalanceo seguida durante el entrenamiento de los modelos, se diseña un proceso de calibración en dos pasos:

- Primeramente, con el objetivo de corregir en las salidas del modelo el efecto de rebalanceo del conjunto de datos de entrenamiento por sobremuestreo, se aplica la fórmula propuesta para ajustar las salidas del modelo en Saerens et al. (2002) (ver sección 2.2.6):

$$p_f(x) = \frac{\frac{\Pi_t^+}{\Pi_s^+} p_s(x)}{\frac{\Pi_t^+}{\Pi_s^+} p_s(x) + \frac{\Pi_t^-}{\Pi_s^-}(1 - p_s(x))}$$

- En segundo lugar, para calibrar las probabilidad corregidas en el paso anterior, se entrena una regresión isotónica sobre un conjunto de datos reservado, la cual se aplicará a las predicciones del modelo en el momento de la inferencia y así obtener la probabilidad final de que una jugada acabe en gol.

Para monitorizar y medir la eficacia del proceso de calibración se han empleado diagramas de confiabilidad y se ha calculado el ECE en cada paso (ver sección 2.2.6).

3.3. Clasificador Base

Con el objetivo de confirmar la validez la estrategia metodológica seguida en el presente trabajo, en cuanto al uso de autoencoders para extraer las características latentes de las representaciones bidimensionales (imágenes en escala de grises) de cada instante de juego y así condensar toda la información relevante de la jugada, se diseña un primer sistema de clasificación con las siguientes características:

- El conjunto de datos de entrenamiento de este modelo base se compone de características de cada situación de juego, construidas directamente a partir de la información disponible en la fuente de datos principal del trabajo proporcionada por LLFP (ver sección 3.1.2).

- También está basado en RNNs.

3.3.1. Conjunto de datos de entrenamiento del clasificador base

A partir del mismo conjunto de datos proporcionado por LLFP y descrito en la sección 3.1.2, conteniendo la posición (coordenadas espaciales x,y) de cada jugador y de la pelota, así como la altura del balón, en cada instante de los últimos 5 segundos previos a cada remate a puerta, se confecciona otro conjunto de datos para el entrenamiento del clasificador base.

Este nuevo conjunto de datos, para cada partido y evento (tiro a puerta), tiene tantos registros como instantes componen la jugada (126); y cada registro está conformado por 69 variables explicativas numéricas con 2 decimales, que se describen en la tabla3.9.

Variable	Descripción
TransfBallX	Posición X del balón
TransfBallY	Posición Y del balón
TransfBallZ	Posición Z del balón
TransfX_1	Posición X del jugador que dispara a puerta
TransfY_1	Posición Y del jugador que dispara a puerta
TransfBallDistance_1	Distancia al balón del jugador que dispara a puerta
TransfX_2	Posición X del portero rival
TransfY_2	Posición Y del portero rival
TransfBallDistance_2	Distancia al balón del portero rival
TransfX_3	Posición X del jugador rival más cercano al balón en el momento del disparo
TransfY_3	Posición Y del jugador rival más cercano al balón en el momento del disparo
TransfBallDistance_3	Distancia al balón del jugador rival más cercano al balón en el momento del disparo
TransfX_4	Posición X del segundo jugador rival más cercano al balón en el momento del disparo
TransfY_4	Posición Y del segundo jugador rival más cercano al balón en el momento del disparo
TransfBallDistance_4	Distancia al balón del segundo jugador rival más cercano al balón en el momento del disparo
TransfX_5	Posición X del tercer jugador rival más cercano al balón en el momento del disparo
TransfY_5	Posición Y del tercer jugador rival más cercano al balón en el momento del disparo
TransfBallDistance_5	Distancia al balón del tercer jugador rival más cercano al balón en el momento del disparo
.....	....
TransfX_12	Posición X del jugador compañero más cercano al balón en el momento del disparo
TransfY_12	Posición Y del jugador compañero más cercano al balón en el momento del disparo
TransfBallDistance_12	Distancia al balón del jugador compañero más cercano al balón en el momento del disparo
TransfX_13	Posición X del segundo jugador compañero más cercano al balón en el momento del disparo
TransfY_13	Posición Y del segundo jugador compañero más cercano al balón en el momento del disparo
TransfBallDistance_13	Distancia al balón del segundo jugador compañero más cercano al balón en el momento del disparo
...	....
TransfX_22	Posición X del jugador compañero más lejano al balón en el momento del disparo
TransfY_22	Posición Y del jugador compañero más lejano al balón en el momento del disparo
TransfBallDistance_22	Distancia al balón del jugador compañero más lejano al balón en el momento del disparo

Tabla 3.9: Tabla con las variables que conforman el conjunto de datos a partir del cual se entrenará el modelo base para el cálculo de la probabilidad de que un remate a puerta acabe en gol.

Para poder entrenar los modelos RNNs sobre este conjunto de datos, se redimensiona para que cada instancia esté compuesta por secuencias de 126 vectores de 69 características.

3.3.2. Entrenamiento y evaluación del clasificador base

Para generar el clasificador base, se toma el mismo modelo de RNN basado en LSTMs descrito en la sección 3.2.3.1 , se rebalancea el conjunto de datos de entrenamiento siguiendo la misma estrategia que la descrita en la sección 3.2.3.2, se utilizan las mismas métricas de evaluación que las indicadas en 3.2.3.3 y se calibran las salidas del modelo con la metodología reflejada en 3.2.4.

3.4. Entorno computacional y herramientas de trabajo

Para el desarrollo del TFM se ha empleado Python como lenguaje de programación y se utiliza la nube Azure de Microsoft como entorno integral de aprovisionamiento de Infraestructura como Servicio (IaaS), Software como Servicio (SaaS) y Plataformas como Servicio (PaaS). En esta sección se describen los distintos elementos provistos por Azure que soportan las actividades del trabajo.

3.4.1. Azure Machine Learning como entorno de desarrollo y ejecución

Azure Machine Learning es una plataforma en si misma que ofrece herramientas y servicios a los científicos de datos para que puedan crear, implementar y administrar modelos en poco tiempo y de una manera controlada, segura y colaborativa. A su vez, permite automatizar el ciclo de vida de los modelos para facilitar la adopción de metodologías de integración y despliegue continuo de software de calidad en el ámbito del desarrollo de proyectos de ciencia de datos (conocido como MLOps).

En la figura 3.11 se muestra la Interfaz de Usuario de la Plataforma Azure Machine Learning, llamada Azure Machine Learning Studio.

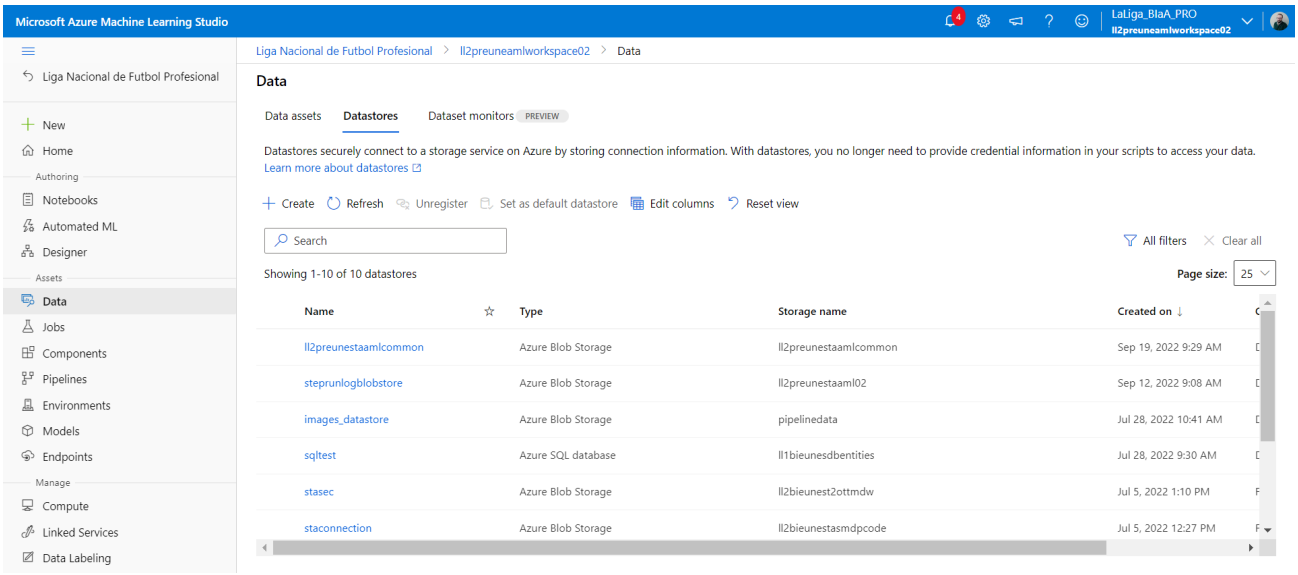

Figura 3.11: Figura que muestra una captura de pantalla de la Interfaz de Usuario Azure Machine Learning Studio.

De entre todos las herramientas y servicios que presta Azure Machine Learning como entorno de trabajo para el desarrollo de proyectos de ciencia de datos, se utilizan en el presente trabajo los que se describen a continuación, agrupados según los distintos módulos en los que se ofrecen:

3.4.1.1. Notebooks

Se usa el módulo de "Notebooks" de Azure Machine Learnig, que disponibiliza un entorno de desarrollo basado en Jupyter Notebook, en donde se construye y organiza el código del trabajo.

Este entorno se ejecuta sobre una máquina virtual, llamada instancia de computación, que está preconfigurada con las librerías y entornos de computación más habituales a la hora de trabajar en proyectos de ciencia de datos. El entorno también dispone de un espacio de almacenamiento para alojar datos y objetos.

3.4.1.2. Data Storage

Este módulo de Azure Machine Learnig permite vincular cuentas y servicios de almacenamiento de Azure (Azure Blob Storage) con el entorno de trabajo Azure Machine Learning, para que los datos almacenados en dichos servicios sean fácilmente accedidos y gestionados por el código desarrollado en Azure Machine Learning.

En el caso concreto del presente trabajo, se crea un Data Storage para el proyecto, vinculado con un Blob Storage de Azure donde se almacenan los conjuntos de datos originales, los conjuntos de datos intermedios que generan los procesos del trabajo, las representaciones bidimensionales de cada instante de juego, etc...

Esta funcionalidad facilita implementar la estrategia dividir el flujo de trabajo en pasos (ver sección 3.2), de manera que los datos generados en cada paso se almacenen en el Data Storage, para que sean accedidos por el paso siguiente.

3.4.1.3. Jobs

Azure Machine Learning permite realizar ejecuciones de código, llamados experimentos, y registrar los elementos generados por la ejecución (métricas, gráficos, objetos, ..) que se desean monitorizar de manera que queden guardados y vinculados al experimento, y así poderlos consultar en cualquier momento futuro.

Cada ejecución de un mismo experimento se denomina "job" y Azure Machine Learning los permite localizar, consultar y administrar en el módulo correspondiente. Junto con los elementos de la ejecución que se desean monitorizar, en los jobs también se almacenan los logs del proceso, el consumo de recursos, el código ejecutado, los parámetros de configuración, etc...

Los jobs y experimientos se puede crear realizando ejecuciones interactivas desde los propios Notebooks; pero también permiten ser lanzados dentro de un entorno de computación efímero, es decir, al solicitar la ejecución de un job bajo esta modalidad, Azure Machine Learning se encarga de aprovisionar la infraestructura de cómputo, de configurar un entorno de ejecución, de realizar la ejecución del job y, finalmente, de decomisar el entorno de computación una vez finalizada la ejecución.

Para utilizar esta funcionalidad de Azure Machine Learning simplemente hay que encapsular el código que se desea ejecutar en un script, indicar que tipo de infraestructura de computo se debe utilizar, que librerías y paquetes se deben implementar en el entorno de ejecución y finalmente lanzar el job, bien desde un Notebook o a través de la Interfaz de Usuario.

3.4.1.4. Enviroments

Un "enviroment" o entorno de ejecución, es la definición de librerías, paquetes y dependencias que Azure Machine Learning debe configurar en una infraestructura de cómputo efímera para poder ejecutar los jobs.

Azure Machine Learning dispone de entornos predefinidos acomodados para trabajos de ciencia de datos, utilizando Python como lenguaje de programación y las librerías y frameworks más habituales (Scikit-Learn, Tensorflow, Pytorch, etc..); pero también permite definir entornos personalizados por el propio usuario.

En el caso concreto del presente trabajo, se define en el módulo correspondiente de Azure Machine Learning, un entorno de Python personalizado para poder ejecutar los pasos en los que se compone cada experimento. Este entorno solicitará la instalación de las librerías que se necesiten (Azure Machine Learnig, Scikit-Learn, Scipy, Numpy, Pandas, Tensorflow, etc...). En la figura 3.12 se muestra el contenido de un fichero .yaml con las librerías, paquetes y dependencias concretas que contiene el entorno de ejecución personalizado definido para el presente trabajo.

```yaml
channels:
  - anaconda
  - conda-forge
dependencies:
  - python=3.6.2
  - 'pip>=21.0,<22'
  - pip:
      - 'matplotlib>=3.3,<3.4'
      - 'psutil>=5.8,<5.9'
      - 'tqdm>=4.59,<4.60'
      - 'pandas>=1.1,<1.2'
      - 'theano>=1.0,<1.1'
      - 'scipy>=1.5,<1.6'
      - 'numpy>=1.10,<1.20'
      - scikit-learn==0.24.1
      - azureml-core==1.26.0
      - azureml-defaults==1.26.0
      - azureml-mlflow==1.26.0
      - azureml-telemetry==1.26.0
      - azureml-train-restclients-hyperdrive==1.26.0
      - azureml-train-core==1.26.0
      - tensorflow-gpu==2.4.0
      - tensorboard==2.4.0
      - horovod==0.20.0
      - 'onnxruntime-gpu>=1.7,<1.8'
name: azureml_271cb08c51c462a516ec0a90ed30bd29

```

Figura 3.12: La figura muestra el contenido del un fichero .yaml con las librerías, paquetes y dependencias del entorno de ejecución Python de Azure Machine Learning con el que se configurarán las instancias de computación en donde se ejecutarán los jobs del TFM.

3.4.1.5. Compute

Según se ha indicado en Azure Machine Learning se pueden aprovisionar distintas infraestructuras de cómputo, tanto permanentes como efímeras, en donde se ejecutan los Notebooks interactivos y los jobs desatendidos. En el caso concreto del presente trabajo, se utilizarán 3 tipos de infraestructuras de cómputo:

- Instancia de computación para desarrollar de manera interactiva a través de Notebooks, la cual tiene las siguientes características:
  - Standard_D2s_v3: 2 cores, 8 GB de memoria RAM, 16 GB de disco.
  - Unidad de procesamiento: CPU - Propósito general [8].
- Clúster efímero de computación en donde se ejecutarán los jobs en los que se aconseje el uso de GPUs, por encargarse del entrenamiento de los modelos basados en redes neuronales, como son los autoencoders y las RNNs. Este clúster tiene las siguientes características:
  - Standard_NC6: 6 cores, 56 GB de memoria RAM, 380 GB de disco.
  - Unidad de procesamiento: GPU - 1 x NVIDIA Tesla K80[9].
  - Número máximo de nodos en uso: 3.
- Clúster efímero de computación en donde se ejecutarán los jobs que no requieran del uso de GPUs, como son los de preprocesamiento y limpieza de datos o los de entrenamiento de las regresiones isotónicas y posterior calibración de las salidas de los modelos. Este clúster tienen las siguientes características.
  - Standard_D12_v2: 4 cores, 28 GB RAM, 200 GB disk.
  - Unidad de procesamiento: CPU - Memoria Optimizada[10].
  - Máximo número de nodos en uso: 3.

3.4.1.6. Pipelines

En Azure Machine Learning se pueden encadenar jobs estableciendo dependencias entre ellos, para poder ejecutar los trabajos en un orden determinado de manera desatendida. A estas estructuras de jobs encadenados se les llama canalizaciones o "pipelines". Cada componente de una canalización se llama paso o "step" y puede tener asociada su propia infraestructura de computación y entorno de ejecución diferente al resto. Además pueden definirse las dependencias entre pasos para que la

[8] https://learn.microsoft.com/es-es/azure/virtual-machines/dv3-dsv3-series
[9] https://www.nvidia.com/en-gb/data-center/tesla-k80/
[10] https://learn.microsoft.com/en-us/azure/virtual-machines/dv2-dsv2-series

canalización permita la ejecución de pasos de manera secuencial, en paralelo o combinando ambas modalidades.

Se utilizaran las canalizaciones de Azure Machine Learning para implementar los pasos en los que descompone el flujo de proceso de los experimentos del TFM, descrito en la sección 3.2. En la figura 3.13 se muestra el tipo de canalizaciones que se definirán en Azure Machine Learning.

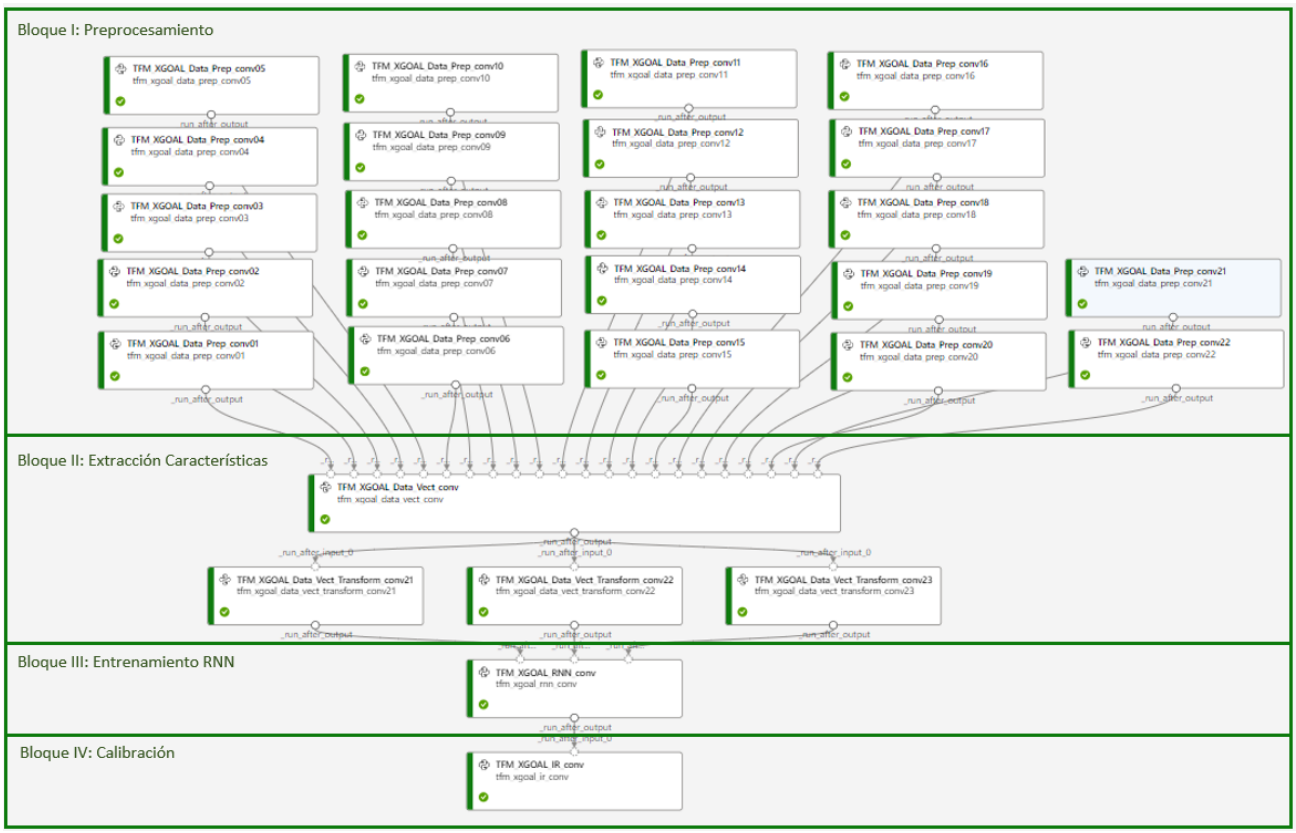

Figura 3.13: En la figura se muestra la composición de los jobs que conforman las canalizaciones de Azure Machine Learning que se utilizan en el TFM, agrupados en los bloques del flujo de proceso definido.

Estas canalizaciones constan de los siguientes grupos de jobs:

- **Bloque de preprocesamiento de datos**: consta de 22 jobs idénticos que se encargan de las tareas descritas en la sección 3.2.1 de limpieza y transformación del conjunto de datos inicial hasta conseguir las representaciones bidimensionales (imágenes en escala de grises) que componen los instantes de cada jugada. Puesto que estas actividades son costosas en términos de consumo de recursos de computación, previamente se ha dividido el conjunto de datos inicial en 22 partes y cada job se encarga de las tareas de preprocesamiento y transformación de una parte, pudiendo de esta manera ejecutarse en paralelo (sin los recursos de computación lo permiten) al no depender de ningún otro proceso.

  Cada job se este bloque se ejecuta sobre el clúster de computación Standard_D12_v2, lo que permite tener hasta 3 ejecuciones en paralelo y así reducir el tiempo de procesamiento de las actividades de esta etapa.

- **Bloque de Extracción de características**: consta de 4 jobs que se encargan de las tareas descritas en la sección 3.2.2 para representar cada imagen en escala de grises mediante un vector de menor dimensionalidad con sus características latentes. Este bloque a su vez se divide en dos partes:
  - Primer job que tienen dependencias con los jobs del bloque anterior, lo que quiere decir que no podrá ejecutarse hasta que no hayan finalizado de manera satisfactoria los 22 jobs del bloque anterior. Este job se encarga de realizar el entrenamiento del autoencoder sobre las imágenes en escala de grises y se ejecuta sobre el clúster de computación Standard_NC6 para aprovechar la potencia de cálculo que ofrece las GPUs y así eficientar los tiempos de entrenamiento del autoencoder.
  - Grupo de 3 jobs idénticos que se encargan de aplicar el encoder generado en el job anterior a las imágenes en escala de grises, para obtener los vectores con las representaciones latentes de cada instante de juego. Puesto que esta activad es costosa en términos de computación, cada job se encargará de vectorizar un tercio de las imágenes, pudiendo de esta manera ejecutarse en paralelo. Los 3 jobs tienen dependencia con el job anterior, por lo que sólo se ejecutarán cuando dicho job haya finalizado satisfactoriamente. Finalmente, indicar que estos jobs también se ejecutan sobre la infraestructura de computo Standard_NC6, lo que permite aprovechar las GPUs y la ejecución en paralelo de los 3 procesos.
- **Entrenamiento del modelo basado en RNN**: consta de un sólo job que se encarga de entrenar el modelo basado en RNN sobre las secuencias de representaciones latentes generadas en el bloque anterior, según se indica en la sección 3.2.3.2. Tiene dependencias con los jobs de dicho bloque, lo que quiere decir que no podrá ejecutarse hasta que no hayan finalizado de manera satisfactoria los 3 jobs anteriores. Este job se ejecuta sobre el clúster de computación Standard_NC6 para aprovechar la potencia de cálculo que ofrecen las GPUs.
- **Calibración de las probabilidades**: consta de un sólo job que se encarga de entrenar la regresión isotónica y calibrar las salidas del modelo basado en RNN, para ajustarlas y que reflejen la probabilidad de real de ocurrencia del evento de gol según se ha descrito en la sección 3.2.4. Este job tiene dependencia de ejecución satisfactoria del job anterior y se ejecutará sobre la infraestructura de cómputo Standard_D12_v2 al no requerir del uso de GPUs.

3.4.1.7. Models

Los diferentes modelos que se van generado en el flujo de proceso de cada experimento, se pueden almacenar y registrar en Azure Machine Learnig, con el objetivo de que estén disponibles de manera sencilla para usos futuros en procesos de inferencia en línea o por lotes. En el trabajo se usa esta funcionalidad para almacenar las distintas versiones de los autoencoders, RNNs y regresiones

isotónicas que se entrenan en los jobs de los experimentos. Esta funcionalidad facilita implementar la estrategia de dividir el flujo de trabajo en pasos y jobs, ya que los modelos entrenados en en un paso se registran en Azure Machine Learning y luego son utilizados para inferencia en jobs posteriores.

3.4.2. Librerías utilizadas

Adicionalmente a las librerías de Azure Machine Learning, para poder trabajar con las funcionalidades descritas en la anterior sección, las principales librerías y frameworks de Python que se han utilizado para desarrollar el trabajo son:

- **NumPy**[11]: Librería por defecto en Python para cálculo numérico eficiente. Ofrece un objeto "array" (estructuras de datos que contienen elementos homogéneos dispuestos en forma de tabla o matriz multidimensional) y funciones comunes necesarias para operaciones sobre los arrays, como por ejemplo, operaciones matemáticas, ordenamiento, selección, operaciones estadísticas, etc..

- **Pandas**[12]: Pandas es una librería potente, flexible y sencilla de utilizar para la ingesta, análisis y manipulación de conjuntos de datos en formato tabla. Ofrece un objeto "dataframe" organizado en filas y columnas, y funciones comunes necesarias para operaciones sobre los dataframes como, por ejemplo, fusión, indexado, selección, agregación, funciones estadísticas, etc..

- **TensorFlow**[13]: Es un framework para machine learning e inteligencia artificial de Google, ampliamente utilizado para construir y entrenar redes neuronales profundas. Utiliza una estructura de grafo para definir y ejecutar cálculos numéricos, lo que le permite distribuir tareas en múltiples dispositivos. Se suele utilizar como backend en los casos en los que se necesita disponer de recursos hardware con GPUs y/o procesamiento distribuido en el entrenamiento de los modelos.

- **Keras**[14]: Es un API de alto nivel para trabajar con redes neuronales profundas usando TensorFlow como backend. Esta librería ofrece la posibilidad de definir, desde modelos sencillos, a arquitecturas más complejas o incluso utilizar aprendizaje transferido (transfer learning) sobre modelos preentrenados.

- **SciKit-Learn**[15]: Es una las principales librerías de Python para aprendizaje automático. Ofrece implementaciones de los principales algoritmos, tanto para aprendizaje supervisado como no supervisado en sus distintas variantes; además de métricas y herramientas para su evaluación, selección de características, ajustes de hiperparámetros, visualización de resultados, etc..

[11] https://numpy.org/
[12] https://pandas.pydata.org/
[13] https://www.tensorflow.org/
[14] https://keras.io/
[15] https://scikit-learn.org/

- **MatPlotLib**[16]: Librería con capacidades gráficas y numerosos recursos de visualización para la explotación visual de los datos.
- **Scipy**[17]: Librería que se utiliza para realizar cálculos científicos y técnicos. Ofrece una amplia gama de herramientas y funciones para el análisis de datos, la optimización, la estadística y la simulación de sistemas físicos y matemáticos. En el caso concreto del presente trabajo se utiliza para aplicar el filtro gaussiano a las representaciones bidimensionales de cada instante de juego.

[16] https://matplotlib.org/
[17] https://scipy.org/

Capítulo 4

Experimentos, resultados y discusión

4.1. Descripción de los conjuntos de datos

4.1.1. Conjunto de datos de entrada

Según se ha indicado en la sección 3.1.2, el conjunto de datos que es fuente principal del presente trabajo es un fichero proporcionado por LLFP conteniendo información de eventing y tracking con la posición (coordenadas espaciales x,y) de cada jugador y de la pelota, así como la altura del balón, en cada uno de los 25 instantes de juego (o frames) de cada uno de los últimos 5 segundos de cada remate a puerta.

La información contenida en el conjunto de datos corresponde a todos los eventos de tiro a puerta ocurridos en 1.884 partidos de la Liga Santander y Liga Smart Bank de las temporadas 2020-2021, 2021-2022 y 2022-2023 (hasta la jornada 26 en la Liga Santander y jornada 28 en Liga Smart Bank), lo que supone contar con 28.880 disparos a puerta, de los cuales 2.998 fueron gol y el resto (25.846) no acabaron en gol.

Adicionalmente, según se ha indicado en la sección 3.4.1.6, para aprovechar el paralelismo de ejecución que ofrece la infraestructura de cómputo disponible en la fase de preprocesamiento de datos del flujo del proceso del trabajo, este fichero previamente se ha dividido en 22 partes. La tabla 4.1 muestra las volumetrías contenidas en cada una de las partes, siendo el total de registros 80.773.564.

Tras los procesos de preprocesamiento y limpieza descritos en 3.2.1, en donde, entre otras tareas, se eliminan registros erróneos y los eventos asociados a penaltis, el conjunto de datos se reduce hasta contener 28.246 disparos a puerta, de los cuales 2.551 fueron gol y 25.695 no fueron gol.

4.1.2. Conjunto de datos con las representaciones bidimensionales

El último paso de la fase de preprocesamiento y preparación es transformar cada evento de tiro en una secuencia de 126 representaciones bidimensionales, asimiladas a imágenes en escala de grises

Id	Temporada	Liga	Jornadas	Partidos	Tiros a Gol	Goles	No Goles	Frames	Registros
1	2020-2021	Santander	1-10	99	1.443	161	1.282	153.591	3.985.590
2	2020-2021	Santander	11-20	100	1.585	185	1.400	170.810	4.383.036
3	2020-2021	Santander	21-30	99	1.598	168	1.430	163.710	4.427.640
4	2020-2021	Santander	31-38	76	1.254	150	1.104	131.067	3.468.780
5	2021-2022	Santander	1-10	100	1.833	199	1.634	189.393	5.073.585
6	2021-2022	Santander	11-20	100	1.806	191	1.615	182.718	4.994.766
7	2021-2022	Santander	21-30	99	1.658	196	1.462	172.051	4.588.542
8	2021-2022	Santander	31-38	8	168	17	151	20.411	465.696
11	2022-2023	Santander	1-6	60	1.178	132	1.046	128.884	3.251.682
21	2022-2023	Santander	7-17	80	1.730	162	1.568	178.423	4.777.542
22	2022-2023	Santander	18-26	99	2.184	215	1.933	215.402	5.913.791
12	2020-2021	SmartBank	1-10	87	740	67	673	84.171	1.870.995
13	2020-2021	SmartBank	11-21	95	649	62	587	76.808	1.980.342
14	2020-2021	SmartBank	22-32	97	1.133	124	1.009	127.690	3.785.897
15	2020-2021	SmartBank	33-42	96	1.276	145	1.131	137.991	3.935.988
16	2021-2022	SmartBank	1-10	100	1.347	132	1.215	147.164	3.717.000
17	2021-2022	SmartBank	11-21	100	1.300	134	1.166	141.529	3.593.646
9	2021-2022	SmartBank	22-32	95	1.431	146	1.285	147.149	3.955.518
10	2021-2022	SmartBank	33-42	28	386	42	344	45.872	1.062.558
18	2022-2023	SmartBank	1-6	66	886	83	803	100.277	2.445.156
20	2022-2023	SmartBank	7-17	100	1.698	135	1.563	176.133	4.686.822
19	2022-2023	SmartBank	18-28	100	1.597	152	1.445	167.969	4.408.992

Tabla 4.1: Tabla que muestra las volumetrías de cada una de las partes en las que se ha dividido el conjunto de datos fuente principal del TFM.

de tamaño 110x140 pixeles, a partir de las posiciones de los jugadores y el balón, a las cuales además se les aplica un filtro gaussiano (ver sección 3.2.1.1).

En la figura 4.1se muestra un ejemplo de 16 imágenes en escala de grises contenidas en una jugada de disparo a puerta.

A partir de las imágenes en escala de grises, se toma 1 de cada 5 para formar otro conjunto de datos que sirve para entrenar los autoencoders que permiten transformar cada imagen en un vector de representaciones latentes que recoja las relaciones espaciales de la posición de los jugadores y del balón en cada instante de juego (ver sección 3.2.2).

Este segundo conjunto de datos tiene un total de 712.538 imágenes de dimensión 100x140. Para el entrenamiento del autoencoder se utiliza el 90 % de las imágenes (641.284), reservando el 10 % (71.254) para validación.

4.1.3. Conjunto de datos para entrenamiento del modelo

Una vez entrenado el autoencoder, se utiliza el encoder generado para transformar las secuencias de 126 imágenes en escala de grises de cada jugada en secuencias de 126 vectores de 64 posiciones.

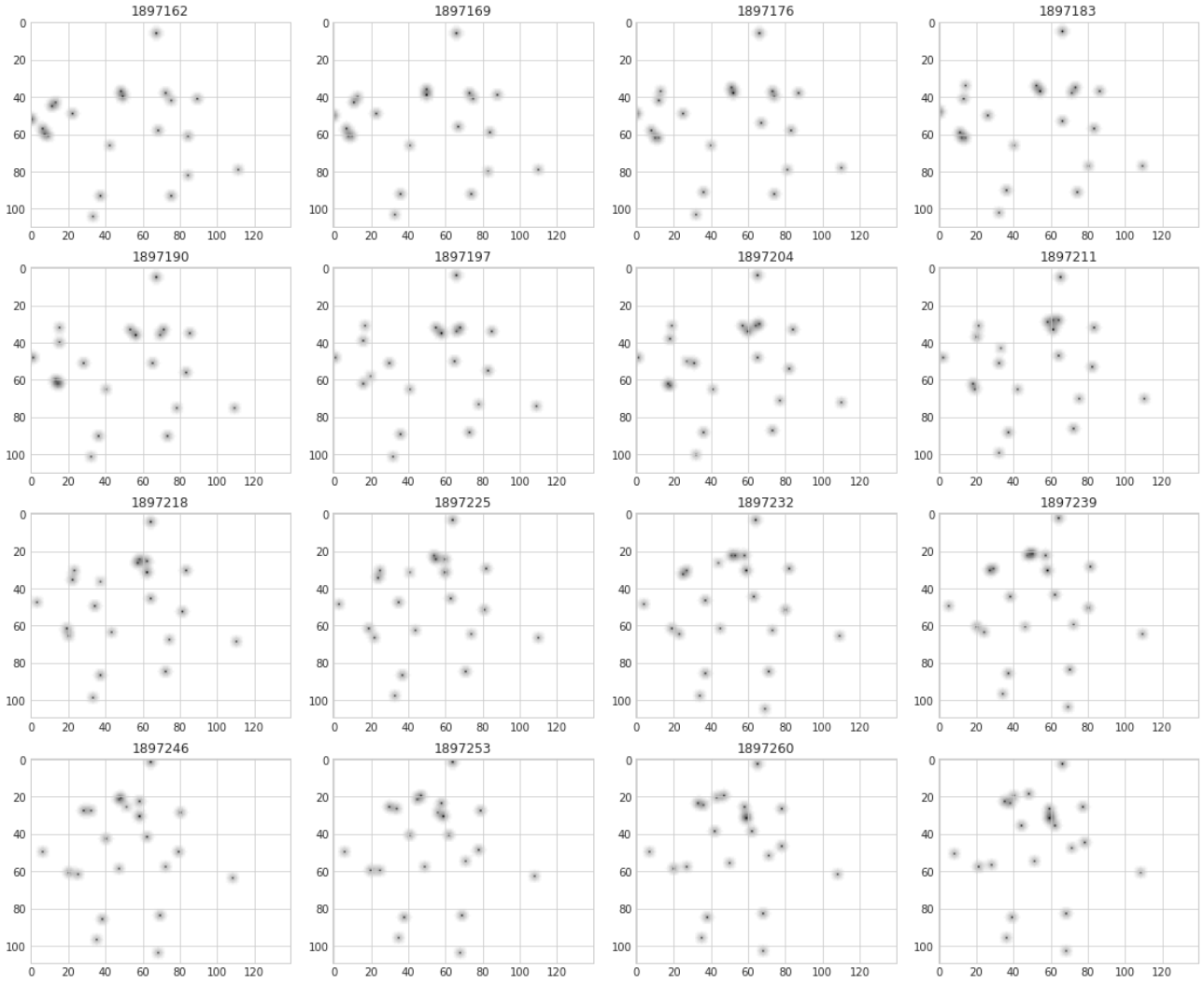

Figura 4.1: Figura con 16 representaciones bidimensionales de ejemplo, asimiladas a imágenes en escala de grises de dimensión 110x140, asociadas a algunos de los instantes de juego contenidos en los 5 segundos anteriores a un disparo a puerta.

En la figura 4.2 se muestra un ejemplo de la representación visual de 16 vectores de 64 posiciones (8x8) con las características latentes de las imágenes en escala de grises contenidas en una jugada de disparo a puerta.

Como resultado de esta transformación, se obtiene el conjunto de datos final para el entrenamiento de los modelos basados en RNNs que permiten obtener la probabilidad de que cada jugada acabe en gol (ver sección 3.2.3).

Este último conjunto de datos recoge 28.246 eventos de disparo (2.551 goles frente a 25.695 no goles), por lo que tendrá la forma de un array de tres dimensiones con tamaño 28.246x126x64.

Para el ejercicio del entrenamiento de los modelos, se divide el conjunto de datos en tres partes:

- **Conjunto de entrenamiento**: 81 % del total (22.878 eventos, de los cuales 2.136 fueron gol frente a los 20.742 no gol) que se utiliza para el entrenamiento de los modelos basados en

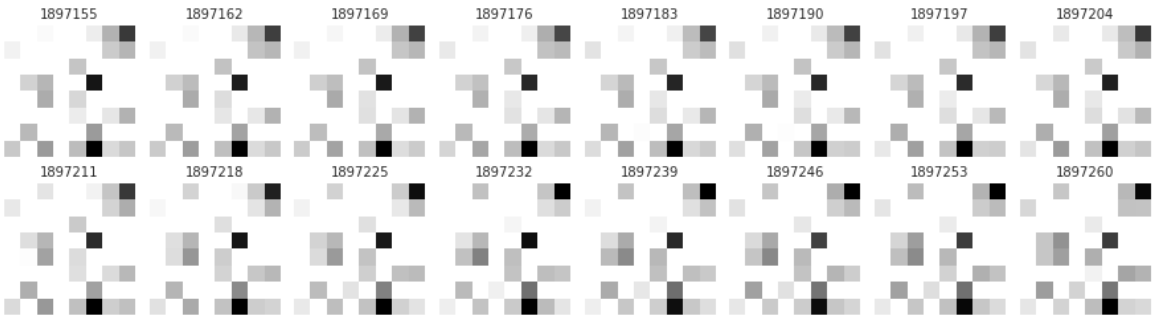

Figura 4.2: Figura con vectores de representaciones latentes de longitud 64 (visualizados como imágenes de 8x8) de ejemplo, asociados a 16 instantes de juego contenidos en los 5 segundos anteriores a un disparo a puerta.

RNN

- **Conjunto de validación**: 9 % del total (2.543 eventos, de los cuales 203 fueron gol frente a los 2.340 no gol) que se utiliza para la evaluación del desempeño del modelo al final de cada época durante la fase de entrenamiento, con el objetivo de determinar la correcta evolución del aprendizaje del modelo.

- **Conjunto de evaluación**: 10 % del total (2.825 eventos, de los cuales 212 fueron gol frente a los 2.613 no gol) que se utiliza para evaluar el desempeño del modelo una vez entrenado y comparar los resultados obtenidos por cada experimento.

Además, debido al desbalanceo existente en el conjunto de datos respecto a las jugadas etiquetadas como gol frente a las que no acabaron en gol, se utiliza una técnica de sobremuestreo aleatorio de las instancias asignadas a la clase minoritaria para equilibrar el conjunto de datos de entrenamiento (ver sección 3.2.3.2), dejando inalterados los conjuntos de datos de validación y de evaluación. Esto lleva a que finalmente el conjunto de datos de entrenamiento tenga 51.392 eventos (25.548 etiquetados como goles y 25.844 como no goles).

4.2. Experimentos realizados

A continuación de especifican los 7 experimentos diseñados para construir el sistema de cálculo de probabilidades de que una jugada acabe en gol.

4.2.1. Modelo Base

Como modelo base sobre el que comparar si las estrategias propuestas mejoran el rendimiento del sistema, se diseña un experimento que utiliza una RNN con unidades LSTM descrita en el apartado 3.3.

Experimento MB_LTSM

- **Fases de Preprocesamiento y Extracción de Características**: se crea el conjunto de datos incluyendo características extraídas directamente de la posición de los jugadores y del balón contenidas en la fuente principal del trabajo. Ver sección 3.3.1.

- **Fase de Entrenamiento del Modelo**: se entrena un modelo RNN con dos capas de 128 unidades LSTM descrito en la sección 3.2.3.1.

- **Fase de Calibración de Probabilidades**: se calibran las salidas arrojadas por el modelo para convertirlas en probabilidades utilizando las técnicas descritas en la sección 3.2.4.

4.2.2. Sistema con autoencoder no convolucional

Se diseñan 2 experimentos (uno basado en RNN con unidades LSTM y otro con unidades GRU) utilizando autoencoders no convolucionales para extraer las características latentes de las imágenes en escala de grises que representan cada instante de una jugada.

Experimento AES_RNN_LSTM

- **Fases de Preprocesamiento:** se limpia y preprocesa el conjunto de datos fuente principal del trabajo para transformar cada instante de juego en una representación bidimensional, asimilada a una imagen en escala de grises, según se indica en la sección 3.2.1.

- **Extracción de Características**: se entrena un autoencoder no convolucional (descrito en 3.2.2.1) para, utilizando el encoder, transformar cada imagen en escala de grises en un vector denso de 64 posiciones con las representaciones latentes. Ver sección 3.2.2.2.

- **Fase de Entrenamiento del Modelo**: se entrena un modelo RNN con dos capas de 128 unidades LSTM descrito en la sección 3.2.3.1.

- **Fase de Calibración de Probabilidades**: se calibran las salidas arrojadas por el modelo para convertirlas en probabilidades utilizando las técnicas descritas en la sección 3.2.4.

Experimento AES_RNN_GRU

El diseño de este experimento es idéntico al descrito anteriormente, pero el modelo utilizado en la Fase de Entrenamiento es un modelo RNN con dos capas de 128 unidades GRU (ver sección 3.2.3.1).

4.2.3. Sistema con autoencoder convolucional

Con el objetivo de condensar la información contenida en las relaciones espaciales de la posición de los jugadores y del balón, se diseñan 2 experimentos (uno basado en RNN con unidades LSTM y otro con unidades GRU) utilizando autoencoders convolucionales para extraer las características latentes de las imágenes en escala de grises que representan cada instante de una jugada.

Experimento AEC_RNN_LSTM

- **Fases de Preprocesamiento:** se limpia y preprocesa el conjunto de datos fuente principal del trabajo para transformar cada instante de juego en una representación bidimensional, asimilada a una imagen en escala de grises, según se indica en la sección 3.2.1.
- **Extracción de Características**: se entrena un autoencoder convolucional (descrito en 3.2.2.1) para, utilizando el encoder, transformar cada imagen en escala de grises en un vector denso de 64 posiciones con las representaciones latentes. Ver sección 3.2.2.2.
- **Fase de Entrenamiento del Modelo**: se entrena un modelo RNN con dos capas de 128 unidades LSTM descrito en la sección 3.2.3.1.
- **Fase de Calibración de Probabilidades**: se calibran las salidas arrojadas por el modelo para convertirlas en probabilidades utilizando las técnicas descritas en la sección 3.2.4.

Experimento AEC_RNN_GRU

El diseño de este experimento es idéntico al descrito anteriormente, pero el modelo utilizado en la Fase de Entrenamiento es un modelo RNN con dos capas de 128 unidades GRU (ver sección 3.2.3.1).

4.2.4. Sistema con filtro gaussiano y autoencoder convolucional

Se diseñan 2 experimentos (uno basado en RNN con unidades LSTM y otro con unidades GRU) utilizando autoencoders convolucionales para extraer las características latentes de las imágenes en escala de grises que representan cada instante de una jugada, pero, a diferencia de los anteriores experimentos, se difumina cada imagen en escala de grises con un filtro gaussiano para aumentar el número de píxeles distintos de cero y favorecer el aprendizaje del autoencoder.

Experimento AEC_FG_RNN_LSTM

- **Fases de Preprocesamiento:** se limpia y preprocesa el conjunto de datos fuente principal del trabajo para transformar cada instante de juego en una representación bidimensional, asimilada

a una imagen en escala de grises (ver sección 3.2.1), y además se le añade un filtro gaussiano según se describe en 3.2.1.1.

- **Extracción de Características**: se entrena un autoencoder convolucional (descrito en 3.2.2.1) para, utilizando el encoder, transformar cada imagen en escala de grises en un vector denso de 64 posiciones con las representaciones latentes. Ver sección 3.2.2.2.
- **Fase de Entrenamiento del Modelo**: se entrena un modelo RNN con dos capas de 128 unidades LSTM descrito en la sección 3.2.3.1.
- **Fase de Calibración de Probabilidades**: se calibran las salidas arrojadas por el modelo para convertirlas en probabilidades utilizando las técnicas descritas en la sección 3.2.4.

Experimento AEC_FG_RNN_GRU

El diseño de este experimento es idéntico al descrito anteriormente, pero el modelo utilizado en la Fase de Entrenamiento es un modelo RNN con dos capas de 128 unidades GRU (ver sección 3.2.3.1).

4.2.5. Resumen de experimentos

En la tabla 4.2 se refleja de manera resumida el conjunto de experimentos con sus características principales

	Fases del Flujo de Proceso de Creación del Sistema			
Código	**Preprocesamiento**	**Extracción Características**	**Entrenamiento Modelo**	**Calibración**
MB_RNN_LSTM	Extracción directa de características		RNN con LSTMs	Regr. Isotónica
AES_RNN_LSTM	Imágenes en escala de grises	Autoencoder No Conv.	RNN con LSTMs	Regr. Isotónica
AES_RNN_GRU	Imágenes en escala de grises	Autoencoder No Conv.	RNN con GRUs	Regr. Isotónica
AEC_RNN_LSTM	Imágenes en escala de grises	Autoencoder Convolucional	RNN con LSTMs	Regr. Isotónica
AEC_RNN_GRU	Imágenes en escala de grises	Autoencoder Convolucional	RNN con GRUs	Regr. Isotónica
AEC_FG_RNN_LSTM	Imágenes B&N + Filtro Gaussiano	Autoencoder Convolucional	RNN con LSTMs	Regr. Isotónica
AEC_FG_RNN_GRU	Imágenes B&N + Filtro Gaussiano	Autoencoder Convolucional	RNN con GRUs	Regr. Isotónica

Tabla 4.2: Tabla que relaciona los experimentos realizados en el TFM con sus principales características, para construir el sistema de cálculo de probabilidades de que una jugada acabe en gol.

4.3. Evaluación de resultados

En esta sección se describirán los resultados de los experimentos junto con la evaluación de los sistemas generados para predecir la probabilidades de que una jugada acabe en gol. La evaluación será realizada utilizando las métricas descritas en la sección 3.2.3.3.

4.3.1. Experimento Modelo Base con RNN basada en LSTM (MB_LTSM)

Las gráficas de evolución de las métricas AUC-ROC y BCE (imagen 4.3) a lo largo de las épocas del entrenamiento del modelo RNN muestra un comportamiento de sobreajuste muy fuerte, ya que la métrica AUC-ROC crece a lo largo de las épocas hasta llegar a un valor de casi perfecta clasificación (0,99), mientras que la AUC-ROC de validación no sólo no mejora sino que empeora ligeramente hasta el 0,68. En cuanto a la BCE de entrenamiento, sigue el mismo patrón que la AUC-ROC, reduciéndose hasta alcanzar un valor de 0,106; pero la BCE de validación no evoluciona y se estanca alrededor del valor 0,62.

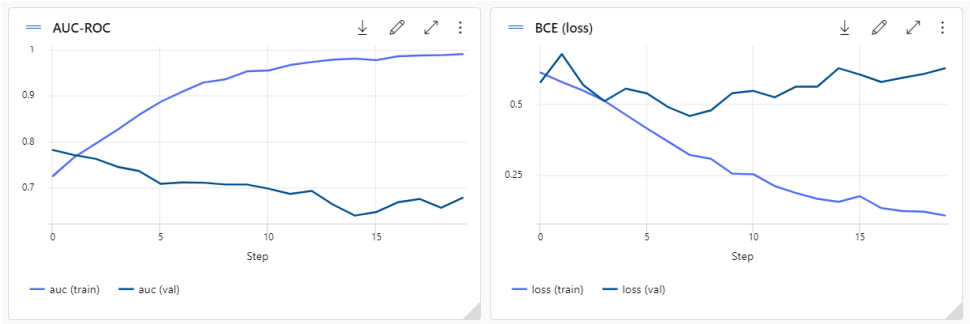

Figura 4.3: Imagen mostrando la evolución de las métricas AUC-ROC y BCE durante el entrenamiento del modelo RNN en el experimento MB_RNN_LSTM.

Respecto a la calibración de las salidas del modelo, se obtiene una ECE de 0,070 que tras el proceso de calibración se corrige hasta un valor de 0,0036. En las gráficas de la imagen 4.4 se muestran los diagramas de confiabilidad antes y después del proceso de calibración.

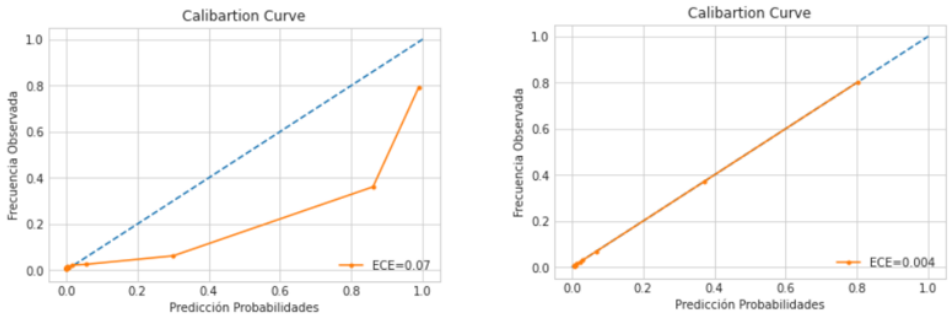

Figura 4.4: Imagen con las gráficas de confiabilidad del sistema construido por el experimento MB_RNN_LSTM, antes y después de la calibración

En cuanto a las métricas de evaluación de rendimiento del sistema, aplicadas sobre el conjunto de test, se obtiene una AUC-PR de 0,144, una AUC-ROC de 0,623 y una BCE de 0,763. En la tabla 4.3 se reflejan estos resultados junto con el valor del resto de métricas secundarias.

	Métricas Principales			Métricas Secundarias		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
MB_RNN_LSTM	0.144	0.623	0.763	0.174	0.294	0.219

Tabla 4.3: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento MB_RNN_LSTM, evaluadas sobre el conjunto de datos de test.

Finalmente, en la imagen 4.5 se muestra los gráficos de las curvas ROC y PR resultantes de la evaluación del sistema sobre el conjunto de datos de test.

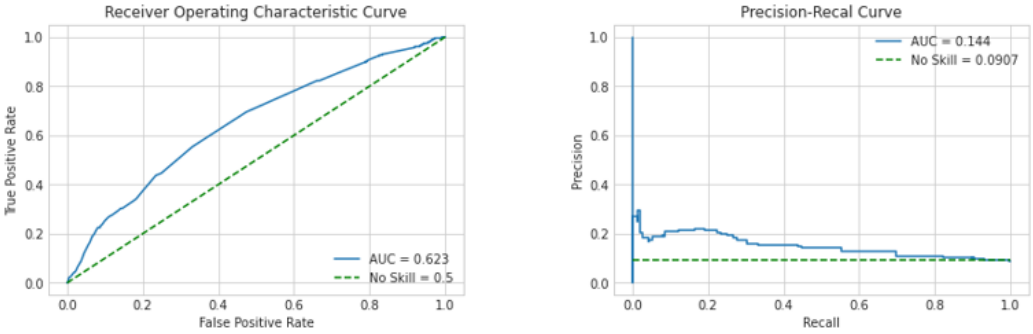

Figura 4.5: Imagen con las gráficas de las curvas ROC y PR resultantes de la evaluación sobre el conjunto de datos de test del sistema construido por el experimento MB_RNN_LSTM.

A la vista de estos resultados, se concluye que el sistema generado por este experimento no es un buen clasificador puesto que sobreajusta fuertemente sobre el conjunto de datos de entrenamiento y generaliza pobremente sobre el conjunto de datos de test, lo que lo invalidada para predecir la probabilidad de que una jugada acabe el gol. Un modelo con sobreajuste indica que ha aprendido a describir errores o ruido presente en las instancias del conjunto de datos de entrenamiento, en lugar de las relaciones subyacentes de los datos, lo que le impide generalizar.

4.3.2. Experimento Sistema con autoencoder no convolucional y RNN basada en LSTM (AES_RNN_LSTM)

En este caso, las gráficas de evolución de las métricas AUC-ROC y BCE (imagen 4.6) a lo largo de las épocas del entrenamiento del modelo RNN, muestran un comportamiento muy similar al descrito en el anterior experimento (MB_RNN_LSTM), apreciándose de nuevo un sobreajuste muy acusado. El comportamiento de la AUC-ROC de entrenamiento crece a lo largo de las épocas hasta llegar a un valor de perfecta clasificación (0,99), mientras que la AUC-ROC de validación se mantiene sin mejoría alrededor del valor 0,60. En cuanto a la BCE de entrenamiento, sigue el mismo patrón que la AUC-ROC, reduciéndose hasta casi llegar a cero (0,0646) y la BCE de validación, no sólo no baja sino que presenta una tendencia creciente.

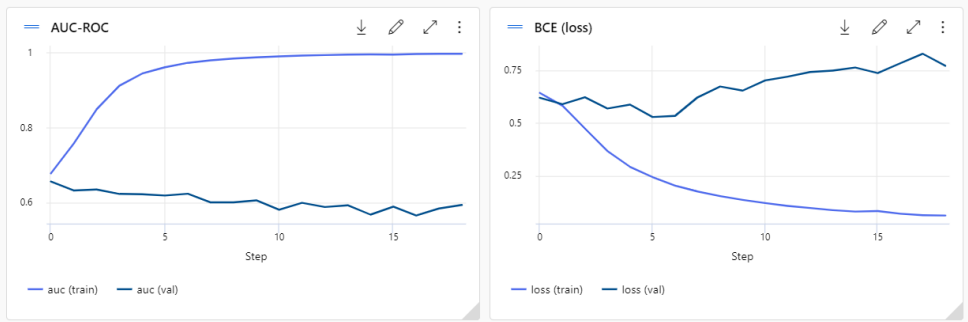

Figura 4.6: Imagen mostrando la evolución de las métricas AUC-ROC y BCE durante el entrenamiento del modelo RNN en el experimento AES_RNN_LSTM

Respecto a la calibración de las salidas del modelo, se obtiene una ECE de 0,049 que tras el proceso de calibración se corrige hasta un valor de 0,0045. En las gráficas de la imagen 4.7 se muestran los diagramas de confiabilidad antes y después del proceso de calibración.

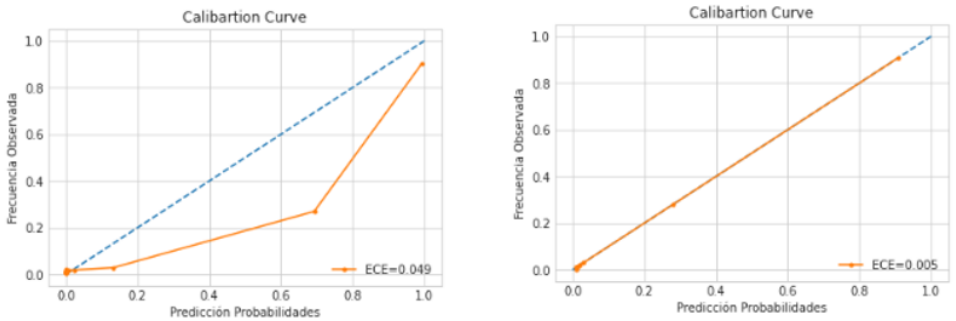

Figura 4.7: Imagen con las gráficas de confiabilidad del sistema construido por el experimento AES_RNN_LSTM, antes y después de la calibración.

En cuanto a las métricas de evaluación de rendimiento del sistema, aplicadas sobre el conjunto de test, se obtiene una AUC-PR de 0,092, una AUC-ROC de 0,564 y una BCE de 0,776. En la tabla 4.4 se reflejan estos resultados junto con el valor del resto de métricas secundarias. Estos valores confirman el acusado sobreajuste del modelo sobre el conjunto de datos de entrenamiento y la pobre generalización sobre el conjunto de test, ya que los valores de la AUC-ROC y AUC-PR son sólo un poco mejores que un clasificador aleatorio o que clasifique todas las instancias a la clase mayoritaria (el cual presentaría una AUC-ROC del 0,5 y una AUC-PR del 0,09).

	Métricas Principales			**Métricas Secundarias**		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
AES_RNN_LSTM	0.092	0.564	0.776	0.109	0.142	0.123

Tabla 4.4: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento AES_RNN_LSTM, evaluadas sobre el conjunto de datos de test.

Finalmente, en la imagen 4.8 se muestra los gráficos de las curvas ROC y PR resultantes de la evaluación del sistema sobre el conjunto de datos de test.

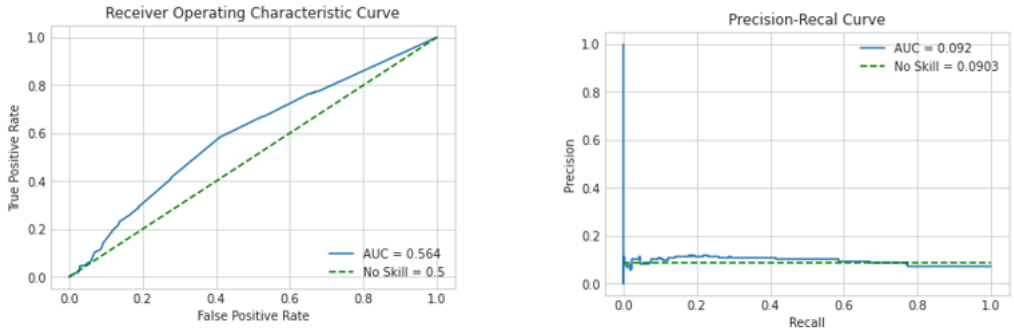

Figura 4.8: Imagen con las gráficas de las curvas ROC y PR resultantes de la evaluación sobre el conjunto de datos de test del sistema construido por el experimento AES_RNN_LSTM.

4.3.3. Experimento Sistema con autoencoder no convolucional y RNN basada en GRU (AES_RNN_GRU)

Las gráficas de evolución de las métricas AUC-ROC y BCE (imagen 4.9) a lo largo de las épocas del entrenamiento del modelo RNN, no mejoran en este experimento respecto a lo descubierto en pasados experimentos: sigue apreciándose sobreajuste muy acusado. La AUC-ROC de entrenamiento crece un poco más lento que en el experimento AES_RNN_LSTM pero vuelve a llegar a valores cercanos a la perfecta clasificación (0,99), mientras que la AUC-ROC de validación se mantiene sin mejoría alrededor del valor 0,60. En cuanto a la BCE se comporta igual que lo descrito en el experimento anterior.

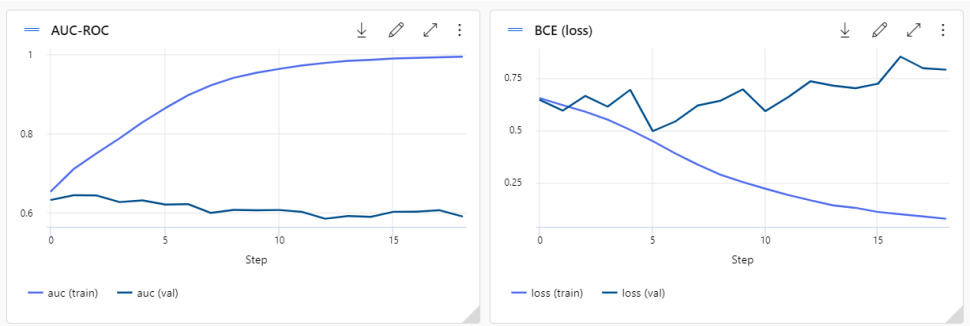

Figura 4.9: Imagen mostrando la evolución de las métricas AUC-ROC y BCE durante el entrenamiento del modelo RNN en el experimento AES_RNN_GRU.

Respecto a la calibración de las salidas del modelo, se obtiene una ECE de 0,049 que tras el proceso de calibración se corrige hasta un valor de 0,0048. En las gráficas de la imagen 4.10 se muestran los diagramas de confiabilidad antes y después del proceso de calibración.

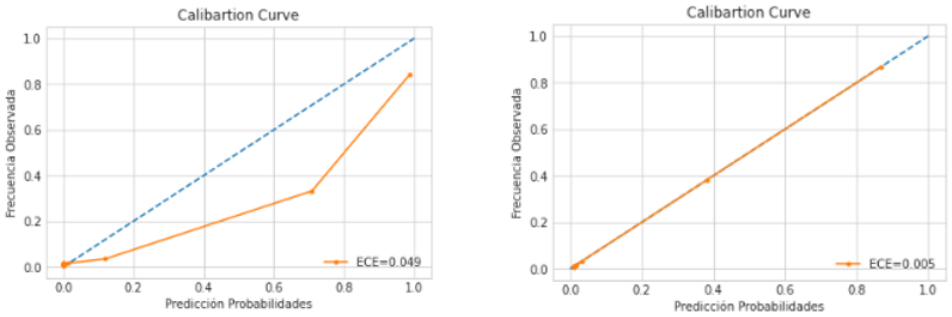

Figura 4.10: Imagen con las gráficas de confiabilidad del sistema construido por el experimento AES_RNN_GRU, antes y después de la calibración.

En cuanto a las métricas de evaluación de rendimiento del sistema, aplicadas sobre el conjunto de test, se obtiene una AUC-PR de 0,097, una AUC-ROC de 0,574 y una BCE de 0,805; confirmando el fuerte sobreajuste en entrenamiento y la baja generalización del sistema sobre muestras no usadas en el aprendizaje. En la tabla 4.5 se reflejan estos resultados junto con el valor del resto de métricas secundarias.

	Métricas Principales			**Métricas Secundarias**		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
AES_RNN_GRU	0.098	0.574	0.805	0.136	0.189	0.158

Tabla 4.5: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento AES_RNN_GRU, evaluadas sobre el conjunto de datos de test.

Finalmente, en la imagen 4.11 se muestra los gráficos de las curvas ROC y PR resultantes de la evaluación del sistema sobre el conjunto de datos de test.

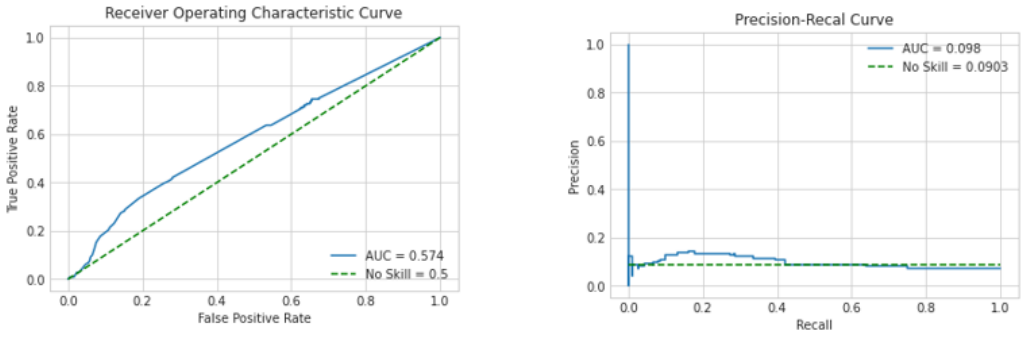

Figura 4.11: Imagen con las gráficas de las curvas ROC y PR resultantes de la evaluación sobre el conjunto de datos de test del sistema construido por el experimento AES_RNN_GRU.

4.3.4. Experimento Sistema con autoencoder convolucional y RNN basada en LSTM (AEC_RNN_LSTM)

Como se puede apreciar en la figura 4.12 las gráficas de evolución de las métricas AUC-ROC y BCE a lo largo de las épocas del entrenamiento del modelo RNN, muestra un comportamiento mejor que en los pasados experimentos, al no apreciarse un sobreajuste tan acusado. La evolución de la AUC-ROC de entrenamiento mejora lentamente a lo largo de las épocas hasta llegar a un valor de 0,74, mientras que la AUC-ROC de validación permanece estable sin mejorar en valores medios del 0,66. En cuanto a la BCE de entrenamiento, sigue el mismo patrón que la AUC-ROC, reduciéndose lentamente hasta alcanzar un valor de 0,60; pero la BCE de validación, aunque con tendencia decreciente, experimenta un comportamiento de "sierra" fluctuante alrededor de la curva BCE de validación. Durante la fase de ajuste de hiperparámetros de la RNN se ha intentado corregir sin éxito el comportamiento fluctuante de la BCE de validación, modificando la tasa de aprendizaje, variando el número de instancias por lote, probando diferentes optimizadores e incorporando técnicas de regularización; por lo que se considera que dicho comportamiento puede deberse a que, a diferencia del conjunto de datos de entrenamiento, el conjunto de datos de validación no está balanceado y el número de instancias etiquetadas como gol son sólo 203 de 2.340. Puesto que la AUC-ROC no muestra un comportamiento tan confuso, será esta la métrica que se utilizará como guía para determinar la correcta evolución de los entrenamientos de los sistemas desarrollados en el presente experimento y siguientes.

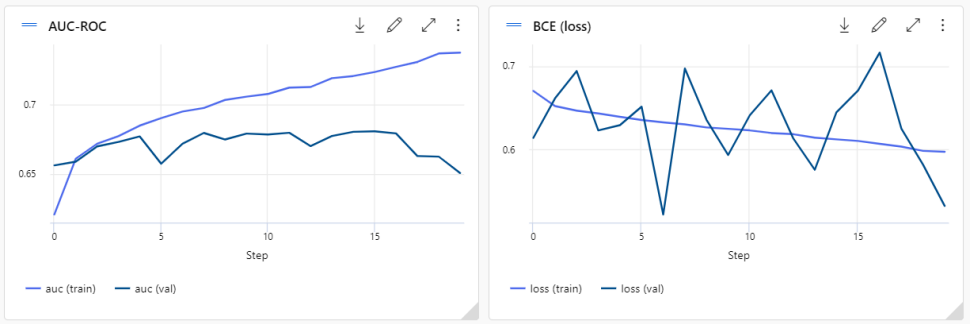

Figura 4.12: Imagen mostrando la evolución de las métricas AUC-ROC y BCE durante el entrenamiento del modelo RNN en el experimento AEC_RNN_LSTM.

Respecto a la calibración de las salidas del modelo, se obtiene una ECE de 0,274 que tras el proceso de calibración se corrige hasta un valor de 0,0027. En las gráficas de la imagen 4.13 se muestran los diagramas de confiabilidad antes y después del proceso de calibración.

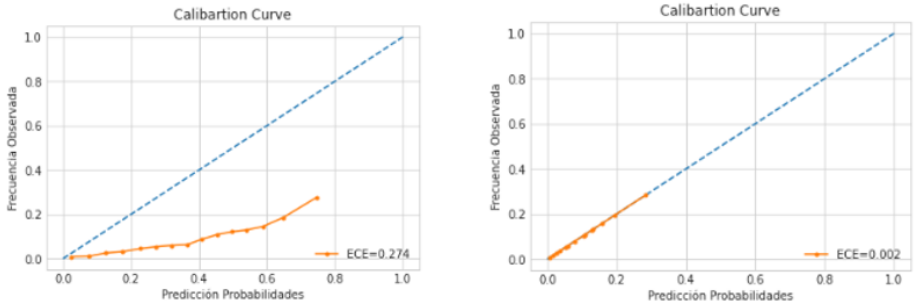

Figura 4.13: Imagen con las gráficas de confiabilidad del sistema construido por el experimento AEC_RNN_LSTM, antes y después de la calibración.

En cuanto a las métricas de evaluación de rendimiento del sistema, aplicadas sobre el conjunto de test, se obtiene una AUC-PR de 0,146, una AUC-ROC de 0.693 y una BCE de 0.504. En la tabla 4.6 se reflejan estos resultados junto con el valor del resto de métricas secundarias.

	Métricas Principales			**Métricas Secundarias**		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
AEC_RNN_LSTM	0.146	0.693	0.504	0.149	0.528	0.232

Tabla 4.6: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento AEC_RNN_LSTM, evaluadas sobre el conjunto de datos de test.

Finalmente, en la imagen 4.14 se muestra los gráficos de las curvas ROC y PR resultantes de la evaluación del sistema sobre el conjunto de datos de test.

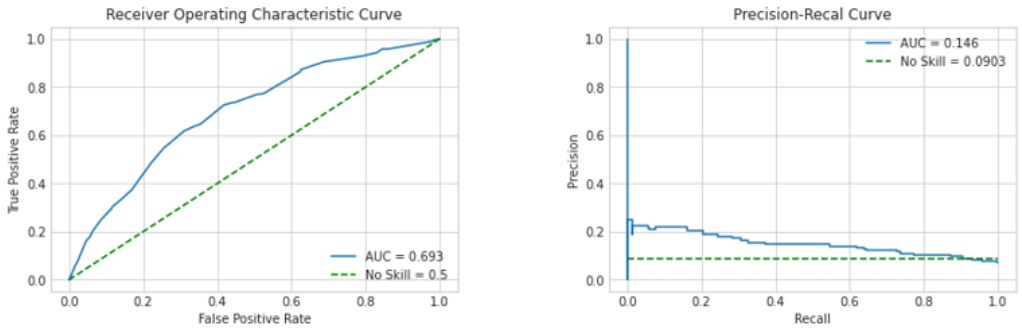

Figura 4.14: Imagen con las gráficas de las curvas ROC y PR resultantes de la evaluación sobre el conjunto de datos de test del sistema construido por el experimento AEC_RNN_LSTM.

Según los resultados reflejados en el experimento, el sistema construido presenta un sobreajuste moderado, no obstante consigue obtener una AUC-PR sobre el conjunto de test del 0.146 y una AUC-ROC del 0.693, por lo que mejora considerablemente el rendimiento de un clasificador aleatorio o que clasifique todas las instancias a la clase mayoritaria y también supera el rendimiento del modelo base (MB_RNN_LSTM).

4.3.5. Experimento Sistema con autoencoder convolucional y RNN basada en GRU (AEC_RNN_GRU)

Las gráficas de evolución de las métricas AUC-ROC y BCE (imagen 4.15) a lo largo de las épocas del entrenamiento del modelo RNN, muestran un comportamiento mejor que en el anterior experimento. Los valores de la AUC-ROC de entrenamiento mejoran lentamente a lo largo de las épocas hasta llegar a un valor de 0,72, mientras que la AUC-ROC también presenta una tendencia de mejoría hasta situarse en valores del 0,67, por lo que el diferencial entre la serie de entrenamiento y validación se mantienen en valores aceptables como para hacer pensar que no hay un sobreajuste acusado. En cuanto a la BCE de entrenamiento, se reduce lentamente hasta alcanzar un valor de 0,62 y la BCE de validación, con tendencia decreciente, sigue fluctuando alrededor de la curva de la BCE de entrenamiento.

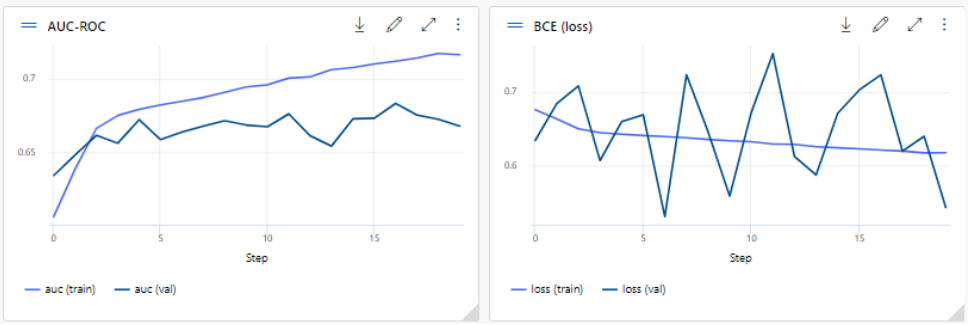

Figura 4.15: Imagen mostrando la evolución de las métricas AUC-ROC y BCE durante el entrenamiento del modelo RNN en el experimento AEC_RNN_GRU.

Respecto a la calibración de las salidas del modelo, se obtiene una ECE de 0,296 que tras el proceso de calibración se corrige hasta un valor de 0,0020. En las gráficas de la imagen 4.16 se muestran los diagramas de confiabilidad antes y después del proceso de calibración.

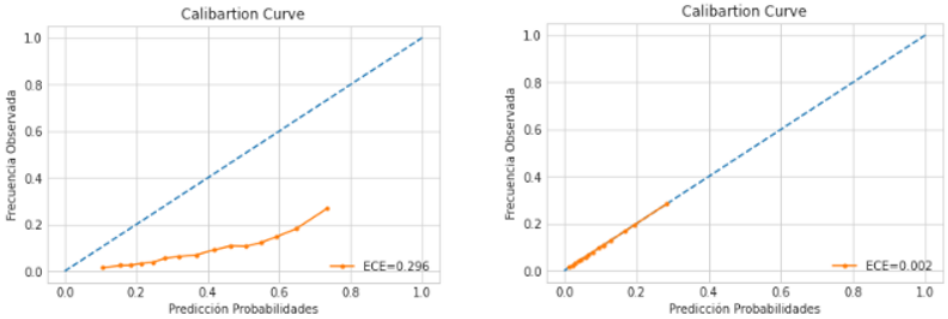

Figura 4.16: Imagen con las gráficas de confiabilidad del sistema construido por el experimento AEC_RNN_GRU, antes y después de la calibración.

En cuanto a las métricas de evaluación de rendimiento del sistema, aplicadas sobre el conjunto de test, se obtiene una AUC-PR de 0,174, una AUC-ROC de 0,724 y una BCE de 0,511. En la tabla

4.7 se reflejan estos resultados junto con el valor del resto de métricas secundarias.

	Métricas Principales			Métricas Secundarias		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
AEC_RNN_GRU	0.174	0.724	0.511	0.156	0.580	0.246

Tabla 4.7: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento AEC_RNN_GRU, evaluadas sobre el conjunto de datos de test.

Finalmente, en la imagen 4.17 se muestra los gráficos de las curvas ROC y PR resultantes de la evaluación del sistema sobre el conjunto de datos de test.

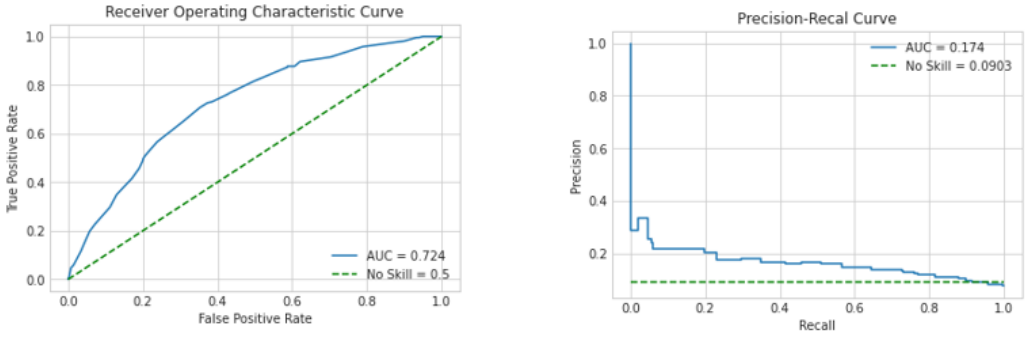

Figura 4.17: Imagen con las gráficas de las curvas ROC y PR resultantes de la evaluación sobre el conjunto de datos de test del sistema construido por el experimento AEC_RNN_GRU.

Los resultados obtenidos en este experimento, con una AUC-PR del 0,174 y una AUC-ROC del 0,724, indican que el sistema construido tiene un rendimiento aceptable para predecir la probabilidad de que una jugada acabe en gol.

4.3.6. Experimento Sistema con filtro gaussiano, autoencoder convolucional y RNN basada en LSTM (AEC_FG_RNN_LSTM)

Las gráficas de evolución de las métricas AUC-ROC y BCE (imagen 4.18) a lo largo de las épocas del entrenamiento del modelo RNN, muestran un comportamiento prácticamente idéntico al reflejado en el experimento AEC_RNN_LSTM, indicando un sobreajuste moderado: la AUC-ROC de entrenamiento mejora lentamente a lo largo de las épocas hasta llegar a un valor de 0,72, mientras que la AUC-ROC de validación permanece estable en valores medios del 0,66. Por su parte la BCE de entrenamiento se reduce lentamente hasta alcanzar un valor de 0,60, y la BCE de validación experimenta el mismo comportamiento fluctuante.

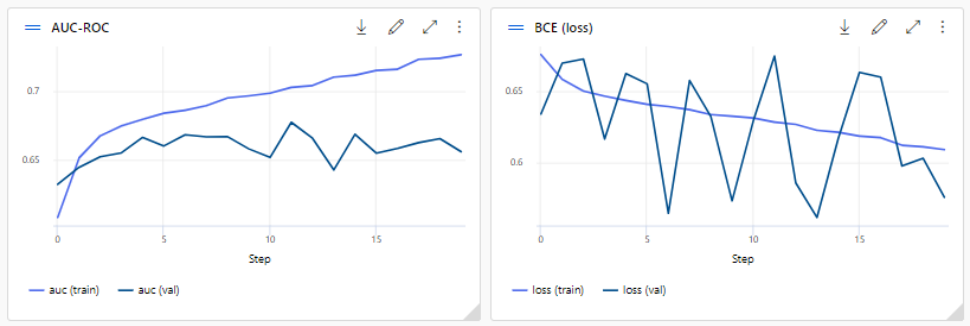

Figura 4.18: Imagen mostrando la evolución de las métricas AUC-ROC y BCE durante el entrenamiento del modelo RNN en el experimento AEC_FG_RNN_LSTM.

Respecto a la calibración de las salidas del modelo, se obtiene una ECE de 0,308 que tras el proceso de calibración se corrige hasta un valor de 0,0025. En las gráficas de la imagen 4.19 se muestran los diagramas de confiabilidad antes y después del proceso de calibración.

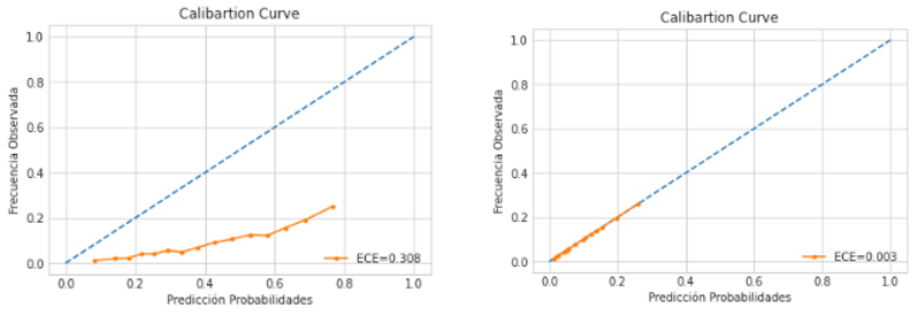

Figura 4.19: Imagen con las gráficas de confiabilidad del sistema construido por el experimento AEC_FG_RNN_LSTM, antes y después de la calibración.

En cuanto a las métricas de evaluación de rendimiento del sistema, aplicadas sobre el conjunto de test, se obtiene una AUC-PR de 0.139, una AUC-ROC de 0.694 y una BCE de 0.551. En la tabla 4.8 se reflejan estos resultados junto con el valor del resto de métricas secundarias.

	Métricas Principales			**Métricas Secundarias**		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
AEC_FG_RNN_LSTM	0.139	0.694	0.551	0.131	0.552	0.212

Tabla 4.8: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento AEC_FG_RNN_LSTM, evaluadas sobre el conjunto de datos de test.

Finalmente, en la imagen 4.20 se muestra los gráficos de las curvas ROC y PR resultantes de la evaluación del sistema sobre el conjunto de datos de test.

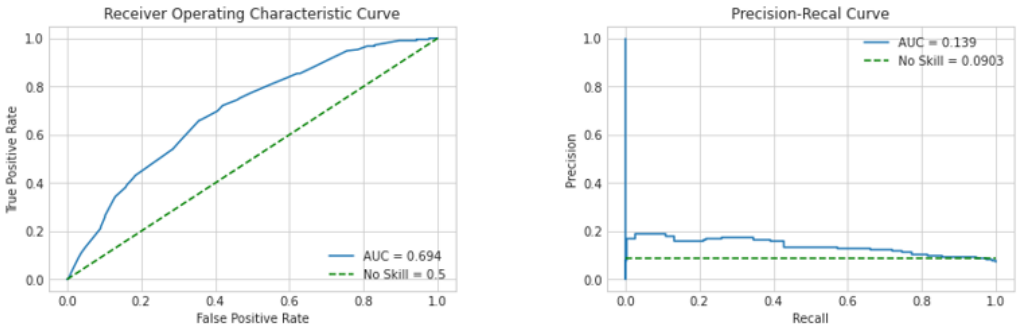

Figura 4.20: Imagen con las gráficas de las curvas ROC y PR resultantes de la evaluación sobre el conjunto de datos de test del sistema construido por el experimento AEC_FG_RNN_LSTM.

A la vista de los resultados obtenidos por el experimento, se puede concluir que la inclusión del filtro gaussiano a las imágenes en escala de grises no ha contribuido a mejorar el aprendizaje del sistema, ni tampoco a mejorar el rendimiento del mismo, si se compara con el sistema equivalente del experimento AEC_RNN_LSTM.

4.3.7. Experimento Sistema con filtro gaussiano, autoencoder convolucional y RNN basada en GRU (AEC_FG_RNN_GRU)

Las gráficas de evolución de las métricas AUC-ROC y BCE (imagen 4.21) a lo largo de las épocas del entrenamiento del modelo RNN, muestras un comportamiento muy similar al reflejado en el experimento AEC_RNN_GRU, indicando un entrenamiento sin sobreajuste: la AUC-ROC de entrenamiento mejora lentamente a lo largo de las épocas hasta llegar a un valor de 0,70, al igual que la AUC-ROC de validación la cual asciende lentamente hasta valores medios del 0,67. Por su parte la BCE de entrenamiento se reduce lentamente hasta alcanzar un valor de 0,63, y la BCE de validación experimenta el tradicional comportamiento fluctuante.

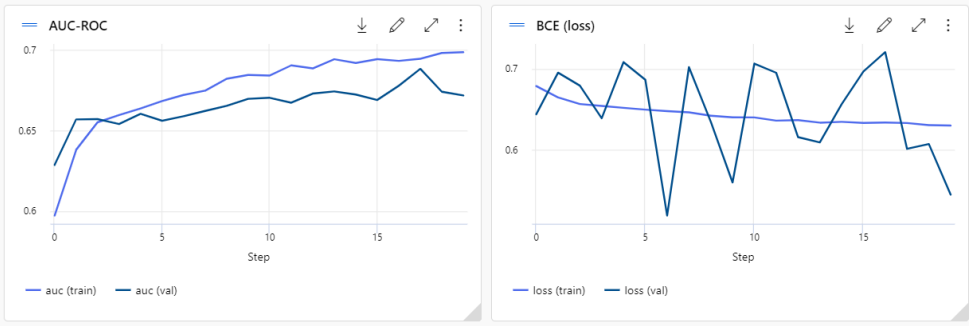

Figura 4.21: Imagen mostrando la evolución de las métricas AUC-ROC y BCE durante el entrenamiento del modelo RNN en el experimento AEC_FG_RNN_GRU.

Respecto a la calibración de las salidas del modelo, se obtiene una ECE de 0,298 que tras el

proceso de calibración se corrige hasta un valor de 0,0040. En las gráficas de la imagen 4.22 se muestran los diagramas de confiabilidad antes y después del proceso de calibración.

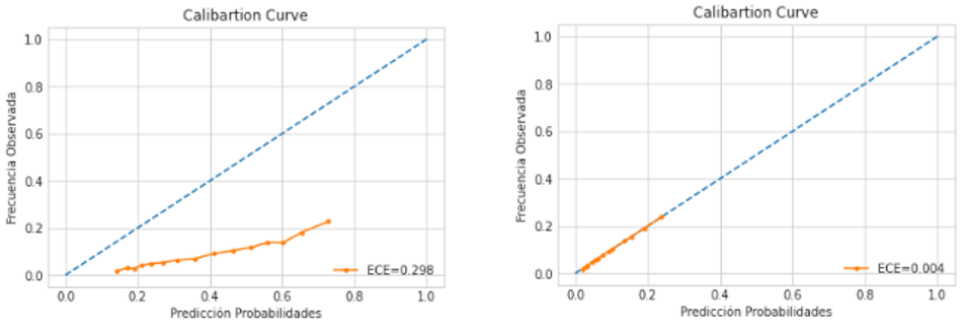

Figura 4.22: Imagen con las gráficas de confiabilidad del sistema construido por el experimento AEC_FG_RNN_GRU, antes y después de la calibración.

En cuanto a las métricas de evaluación de rendimiento del sistema, aplicadas sobre el conjunto de test, se obtiene una AUC-PR de 0,153, una AUC-ROC de 0,685 y una BCE de 0,529. En la tabla 4.9 se reflejan estos resultados junto con el valor del resto de métricas secundarias.

	Métricas Principales			**Métricas Secundarias**		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
AEC_FG_RNN_GRU	0.153	0.685	0.529	0.131	0.524	0.210

Tabla 4.9: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento AEC_FG_RNN_GRU, evaluadas sobre el conjunto de datos de test.

Finalmente, en la imagen 4.23 se muestra los gráficos de las curvas ROC y PR resultantes de la evaluación del sistema sobre el conjunto de datos de test.

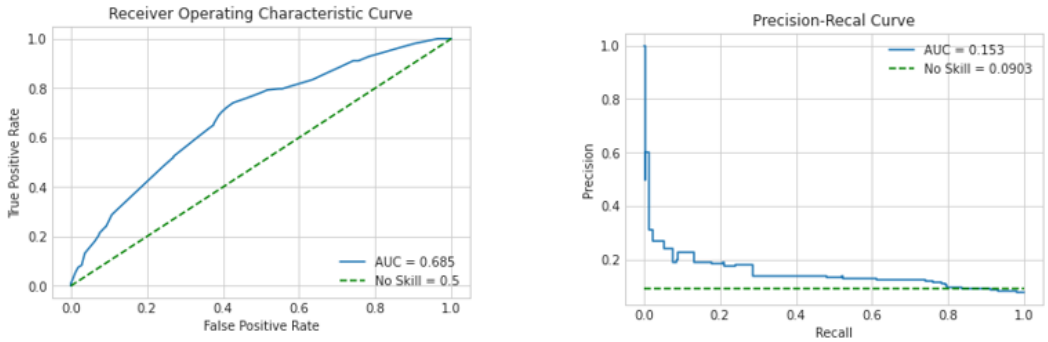

Figura 4.23: Imagen con las gráficas de las curvas ROC y PR resultantes de la evaluación sobre el conjunto de datos de test del sistema construido por el experimento AEC_FG_RNN_GRU.

A la vista de los resultados obtenidos, en comparación con los del sistema equivalente AEC_RNN_GRU, se concluye que la incorporación del filtro gaussiano a las imágenes en escala de grises ha ayudado

ligeramente a reducir el sobreajuste en el entrenamiento del modelo RNN, aunque el rendimiento del sistema es sensiblemente peor, puesto que se obtienen valores de AUC-PR de 0,153 y AUC-ROC de 0,685 en el conjunto de test, frente a una AUC-PR de 0,174 y AUC-ROC de 0,724 del experimento AEC_RNN_GRU.

4.3.8. Resumen de resultados

En la tabla 4.10 se muestra un resumen de los resultados obtenidos por los sistemas construidos en cada uno de los experimentos, en las diferentes métricas de evaluación de desempeño sobre el conjunto de datos de test. En la tabla se han resaltado en negrita los valores mejores en cada métrica.

	Métricas Principales			**Métricas Secundarias**		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
MB_RNN_LSTM	0.144	0.623	0.763	0.174	0.294	0.219
AES_RNN_LSTM	0.092	0.564	0.776	0.109	0.142	0.123
AES_RNN_GRU	0.098	0.574	0.805	0.136	0.189	0.158
AEC_RNN_LSTM	0.146	0.693	**0.504**	0.149	0.528	0.232
AEC_RNN_GRU	**0.174**	**0.724**	0.511	**0.156**	**0.580**	**0.246**
AEC_FG_RNN_LSTM	0.139	0.694	0.551	0.131	0.552	0.212
AEC_FG_RNN_GRU	0.153	0.685	0.529	0.131	0.524	0.210

Tabla 4.10: Tabla que muestra los resultados obtenidos en las métricas de desempeño por los modelos generados en los diferentes experimentos, evaluadas sobre el conjunto de datos de test.

Observando dichos resultados, se pueden extraer las siguientes conclusiones:

- El modelo base (MB_RNN_LSTM), el cual confecciona las características de cada instante de una jugada directamente del conjunto de datos original con la posición de los jugadores y del balón, ofrece un rendimiento discreto, con un AUC-PR de 0,144 y un AUC-ROC de 0,623, aunque mejor que un clasificador aleatorio o que asigne todas las instancias a una sola clase. Además durante la fase de entrenamiento se aprecia un acusado sobreajuste.

- Al incluir en el sistema un autoencoder no convolucional para extraer las representaciones latentes de las imágenes en escala de grises generadas a partir de la posición de los jugadores y del balón (experimentos AES_RNN_LSTM y AES_RNN_GRU), se obtienen modelos que presentan un sobreajuste incluso mayor durante la fase de entrenamiento, lo que hace que generalicen muy pobremente y no sólo no sean capaces de superar el rendimiento del modelo base, sino que además presentan métricas de AUC-PR y AUC-ROC sólo un poco mejores que las que tendría un clasificador básico o aleatorio (AUC-PR = 0,09 y AUC-ROC = 0,5).

- Cuando se utiliza un autoencoder convolucional capaz de extraer no sólo las relaciones lineales entre los píxeles de las imágenes en escala de grises de cada jugada, sino también las relaciones espaciales (experimentos AEC_RNN_LSTM y AEC_RNN_GRU), se obtienen rendimientos

mejores que el modelo base, llegando a una AUC-PR del 0.174 y una AUC-ROC de 0.724 en el caso del sistema AEC_RNN_GRU. Ademas, se mitigan los problemas de sobreajuste en el entrenamiento, especialmente con el modelo AEC_RNN_GRU.

- Finalmente, cuando se incluye en las imágenes en escala de grises el filtro gaussiano (experimentos AEC_FG_RNN_LSTM y AEC_FG_RNN_GRU) no se aprecia mejoría en el rendimiento de los sistemas, aunque los leves síntomas de sobreajuste durante el entrenamiento del experimento AEC_RNN_GRU, desaparecen por completo en el caso del modelo AEC_FG_RNN_GRU.

Tomando en consideración todas las observaciones anteriores, se establece como el mejor sistema el construido por el experimento AEC_RNN_GRU consistente en entrenar una RNN basada en unidades GRU sobre secuencias de representaciones latentes obtenidas mediante un autoencoder convolucional aplicado a las imágenes en escala de grises (sin filtro gaussinao) que identifican la posición de los jugadores y del balón en cada instante de la jugada de tiro a puerta.

Dicho modelo no presenta sobreajuste durante la fase de entrenamiento y obtiene un valor de AUC-PR de 0,174 y AUC-ROC de 0,724 sobre el conjunto de datos de test, los cuales superan el rendimiento del modelo base, llegando a unos valores muy aceptables para cumplir el objetivo principal marcado en el presente TFM: construir un sistema que permita predecir la probabilidad de marcar un gol que tiene un jugada de fútbol para evaluar y comparar la dificultad del remate respecto a otras jugadas.

Por último, en la tabla 4.10 se muestran los valores de las métricas de rendimiento obtenidos por los diferentes sistemas, evaluadas sobre todo el conjunto de datos disponible.

	Métricas Principales			**Métricas Secundarias**		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
MB_RNN_LSTM	0.745	0.937	0.137	0.713	0.764	0.738
AES_RNN_LSTM	0.830	0.938	0.113	0.836	0.779	0.806
AES_RNN_GRU	0.792	0.935	0.122	0.766	0.804	0.784
AEC_RNN_LSTM	0.215	0.734	0.273	0.217	0.398	0.281
AEC_RNN_GRU	0.208	0.722	0.276	0.214	0.399	0.278
AEC_FG_RNN_LSTM	0.202	0.724	0.276	0.194	0.472	0.275
AEC_FG_RNN_GRU	0.186	0.700	0.289	0.205	0.304	0.245

Tabla 4.11: Tabla que muestra los resultados obtenidos en las métricas de desempeño por los modelos generados en los diferentes experimentos, evaluadas sobre todo el conjunto de datos disponible.

Como no puede ser de otra manera, los valores de AUC-PR y AUC-ROC en los sistemas que presentan un acusado sobreajuste son engañosamente elevados, ya que la mayor parte de las instancias del conjunto de datos han sido empleadas durante el entrenamiento. Este comportamiento se aprecia en el modelo base (MB_RNN_LSTM) y en los sistemas que utilizan un autoencoder no convolucional (AES_RNN_LSTM y AES_RNN_GRU).

El resto de sistemas, salvo el AEC_FG_RNN_GRU, tienen unos valores de AUC-PR y AUC-ROC muy similares, siendo ligeramente más elevados en el caso de los sistemas que utilizan modelos RNN basados en LSTM (AEC_RNN_LSTM y AEC_FG_RNN_LSTM) puesto que presentaron un ligero sobreajuste durante el entrenamiento frente a los que utilizan RNN basadas en unidades GRU.

A destacar que el modelo AEC_RNN_GRU consigue una AUC-ROC sobre todo el conjunto de datos prácticamente idéntica a la AUC-ROC calculada sobre el conjunto de datos de test, lo que reafirma en considerarlo el mejor sistema para cumplir el objetivo principal del TFM.

4.4. Comparación del sistema respecto al algoritmo "Goal Probability" de LLFP.

Finalmente, con el objetivo de determinar si el sistema elegido (AEC_RNN_GRU) tienen un rendimiento lo suficientemente aceptable como para sustituir al actual algoritmo "Goal Probabilility" que ha desarrollado LLFP, ambos modelos se evalúan sobre un mismo conjunto de datos. Este conjunto de datos es un subconjunto del que es fuente principal del presente trabajo, descrito en la sección 4.1.1, puesto que del mismo se han retirado sólo las jugadas a balón parado (1.430 jugadas) por no ser objeto de inferencia por parte del algoritmo "Goal Probability". Tras esta reducción el conjunto de datos se compone de 26.816 jugadas de gol, de las cuales 2.461 fueron goles, mientras que 24.355 no fueron goles.

El evaluar el sistema AEC_RNN_GRU sobre este conjunto de datos, implica medir el rendimiento sobre un gran número de jugadas que fueron empleadas durante el entrenamiento, por lo que puede llevar a unos resultados inválidos si el modelo sufriera de sobreajuste. No obstante, como se ha demostrado en la sección anterior, el sistema AEC_RNN_GRU no adolece de sobreajuste al obtener una AUC-ROC sobre todo el conjunto de datos disponible prácticamente idéntica a la AUC-ROC calculada sobre el conjunto de datos de test.

Tras la evaluación se obtiene para el modelo "Goal Probability" de LLFP una AUC-PR de 0,275, una AUC-ROC de 0,731 y una BCE de 0,294, mientras que para el modelo AEC_RNN_GRU se obtiene una AUC-PR de 0,217, una AUC-ROC de 0,725 y una BCE de 0,278. Por tanto, el rendimiento de ambos modelos puede considerarse prácticamente idéntico, siendo el del "Goal Probability" ligeramente superior. En la tabla 4.12 se reflejan los resultados de ambos sistemas junto con el valor del resto de métricas secundarias.

	Métricas Principales			Métricas Secundarias		
Experimento	**AUC-PR**	**AUC-ROC**	**BCE**	**Precisión**	**Sensibilidad**	**F-Score**
AEC_RNN_GRU	0.217	0.725	0.278	0.222	0.401	0.286
"Goal Probability"	0.275	0.731	0.294	0.247	0.401	0.305

Tabla 4.12: Tabla con las métricas de evaluación de rendimiento del sistema construido por el experimento AEC_RNN_GRU, respecto las obtenidas por el modelo "Goal Probability" de LLFP, evaluadas sobre un conjunto de datos homogéneo.

Además, el modelo desarrollado en el presente TFM elimina algunas de las limitaciones actuales del modelo "Goal Probability" de LLFP (ver 1.1.2):

- El modelo "Goal Probability" no se calcula para las jugadas de balón parado (saque de esquina, faltas indirectas, etc..) puesto que, al tomar en cuenta sólo el instante del remate a puerta, no ofrece buenos resultados según indica LLFP; mientras que el desarrollado en el presente trabajo si es posible utilizarlo en dichas ocasiones al incorporar el movimiento de los jugadores y del balón en los 5 segundos previos al remate, como cualquier otra jugada de gol.

- El modelo "Goal Probability" no predice satisfactoriamente distintas tipologías de jugadas de gol, como boleas o remates de cabeza,.. ya que las variables explicativas no integran la información de la altura del balón y, por tanto, considera que todos los tiros se han realizado con el balón controlado por el jugador con sus pies. El modelo desarrollado en el presente trabajo si incorpora la altura del balón como entrada al sistema para ser tenido en cuenta por los algoritmos de aprendizaje profundo empleados.

- Puesto que el modelo "Goal Probability" sólo toma en cuenta el instante del remate a puerta, no contempla el contexto de la jugada previo al disparo que, en muchas ocasiones, condiciona la dificultad del remate y por tanto la probabilidad de marcar el gol; mientras que el sistema desarrollado en el presente trabajo tiene en cuenta los datos de posicionamiento de los jugadores y del balón en los 5 segundos anteriores al disparo y, por tanto, aprovecha la información de contexto de la jugada.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

Una de las métricas avanzadas que LaLiga de Fútbol Profesional introdujo a partir de la temporada 2021-2022 en las retransmisiones de televisión de los partidos fue el algoritmo "Goal Probability" que indica la probabilidad de que un remate a puerta acabe en gol. El modelo "Goal Probability", pese a su interpretabilidad y eficiencia computacional, tiene algunas limitaciones como, por ejemplo, que no incorpora las jugadas a balón parado, que no predice satisfactoriamente las jugadas de gol en las que sea relevante considerar la altura del balón o que no contempla el contexto de la jugada previo al remate. Estas limitaciones están asociadas al diseño de las variables explicativas del modelo, las cuales se han construido a partir de la posición de los jugadores y del balón en el momento del remate a puerta y, por tanto, no tienen cuenta la información temporal de lo acontecido en la jugada en los segundos previos al disparo.

El presente TFM ha tenido como motivación principal predecir la probabilidad de marcar un gol que tiene un jugada de fútbol, resolviendo las limitaciones que presenta el modelo "Goal Probability" de LLFP, mediante el diseño, construcción y evaluación de un sistema de predicción basado en RNNs que incorpora la información de cómo trascurre la jugada en los últimos 5 segundos anteriores al disparo a puerta y no sólo del momento del remate. Para extraer la información espacio-temporal de la jugada previa a un remate a puerta, se han generado unas representaciones bidimensionales construidas a partir de los datos de la posición (coordenadas espaciales x,y) de cada jugador y de la pelota en los 125 instantes anteriores al disparo, las cuales son asimiladas a imágenes en escala de grises; y conforman las secuencias de elementos que alimentan la RNN. Además, para utilizar las relaciones espaciales contenidas en las imágenes en escala de grises, la RNN se ha combinado con un Autoencoder Convolucional, el cual permite condensar las características subyacentes de las imágenes en forma de vector numérico de baja dimensionalidad, llamado representación latente.

Adicionalmente, puesto que el problema que el presente trabajo ha pretendido resolver es un problema supervisado de clasificación binaria (gol o no gol), en donde no se toma como resultado

la clase a partir de un umbral aplicado a la salida del modelo, sino la probabilidad de ocurrencia del evento, se transforman las salidas del modelo en probabilidades reales mediante un ejercicio de calibración de los resultados.

Con el objetivo de ir confirmando que las decisiones metodológicas tomadas son correctas, se diseña un conjunto de experimentos en los que se prueban distintas aproximaciones de manera incremental:

- Un primer experimento que entrena un modelo base RNN y que confecciona las características de cada instante de una jugada directamente del conjunto de datos original con la posición de los jugadores y el balón, sin generar representaciones bidimensionales (asimiladas a imágenes en escala de grises) ni extraer las características latentes de las mismas mediante un autoencoder convolucional.

- Un segundo grupo de experimentos que entrenan unos modelos RNN basados en unidades LSTM y GRU, que representan cada instante de un jugada mediante una imagen en escala de grises y que extraen las características latentes de las mismas mediante un autoencoder simple no convolucional.

- Y un tercer grupo de experimentos incorporando el razonamiento metodológico completo: entrenan unos modelos RNN basados en unidades LSTM y GRU, representan cada instante de un jugada mediante una imagen en escala de grises y extraen las representaciones latentes de las mismas mediante un autoencoder convolucional

Tras la ejecución de los experimentos se comprueba que la aproximación metodológica elegida es acertada, puesto que los sistemas que combinan las RNN con los autoencoder convolucionales son los que ofrecen los mejores valores en las métricas de evaluación del rendimiento (principalmente AUC-PR y AUC-ROC).

Dentro del tercer grupo de experimentos también se ha demostrado que los sistemas que contienen RNNs basadas en unidades GRU tienen mejor comportamiento que los basados en unidades LSTM; y que incorporar un filtro gaussiano a las imágenes en escala de grises con el objetivo de ampliar el número de pixeles diferentes de cero y potenciar el aprendizaje del autoencoder, no mejoran el rendimiento del sistema.

Concretamente, las principales conclusiones de los experimentos son las siguientes:

- El modelo base ofrece un rendimiento discreto con un AUC-PR de 0,144 y un AUC-ROC de 0,623, aunque mejor que un clasificador aleatorio o que asigne todas las instancias a las una sóla clase. Además durante la fase de entrenamiento se aprecia un acusado sobreajuste.

- Al incluir en el sistema un autoencoder no convolucional para extraer las representaciones latentes de las imágenes en escala de grises generadas a partir de la posición de los jugadores y el balón, se obtienen modelos que presentan un sobreajuste incluso mayor durante la fase

de entrenamiento, lo que hace que generalicen muy pobremente y, no sólo no sean capaces de superar el rendimiento del modelo base, sino que además presentan métricas de AUC-PR y AUC-ROC sólo un poco mejores que las que tendría un clasificador básico o aleatorio (AUC-PR = 0,09 y AUC-ROC = 0,5).

- Cuando se sustituye el autoencoder por uno convolucional capaz de extraer, no sólo las relaciones lineales entre los píxeles de las imágenes en escala de grises de cada jugada, sino también las relaciones espaciales, se obtienen rendimientos mejores que el modelo base, llegando a una AUC-PR del 0,174 y una AUC-ROC de 0,724 en el caso del sistema con un modelo RNN basado en unidades GRU. Ademas, con este modelo se mitigan notablemente los problemas de sobreajuste en el entrenamiento.

- Finalmente, cuando se incluye en las imágenes en escala de grises el filtro gaussiano no se aprecia mejoría en el rendimiento de los sistemas, aunque los síntomas de sobreajuste durante el entrenamiento desaparecen por completo en el caso del sistema con un modelo RNN basado en unidades GRU.

Aunque un AUC-PR del 0,174 y un AUC-ROC de 0,724 parece resultados modestos para un sistema de clasificación, hay que tener en cuenta que el hecho de marcar un gol es un evento extraño en un partido de fútbol y prácticamente imposible de modelar debido a la cantidad de factores aleatorios que intervienen en su consecución. Ademas estos valores de rendimiento son muy similares a los obtenidos por el algoritmo "Goal Probabilility" que ha desarrollado LLFP sobre conjuntos de datos homogéneos. Por tanto, a la vista de estos resultados, se considera que se ha cumplido el objetivo principal marcado en el presente TFM: construir un sistema que permita predecir la probabilidad de marcar un gol que tiene un jugada de fútbol para evaluar y comparar la dificultad del remate respecto a otras jugadas, incorporando la información del trascurso de la jugada y no sólo del momento del disparo y así resolver las limitaciones del actual algoritmo "Goal Probability" de LLFP.

El trabajo también ha permitido cumplir los objetivos secundarios o específicos:

- Revisar las aproximaciones realizadas en estos últimos años al problema de predecir la probabilidad de gol en el fútbol, resumiendo el "estado del arte" en cuanto a las soluciones propuestas.

- A partir de las fuentes de información disponibles, construir procesos y algoritmos que permitan extraer la información espacio-temporal de la jugada previa a un remate a puerta y representarla de manera que sea apropiada para alimentar sistema predictivo propuesto.

- Construcción de un conjunto de datos correctamente etiquetado para resolver el problema de aprendizaje supervisado de clasificación del que se ocupa el trabajo.

- Diseñar y entrenar modelos de aprendizaje profundo basados en RNNs.

- Evaluar los resultados de las estrategias propuestas para determinar cual es el sistema que mejor resuelve el problema planteado.

- Interpretar y calibrar los resultados del modelo para conocer el grado de cumplimiento del objetivo general propuesto.

- Hacer uso de la herramienta Azure Machine Learning y el framework de trabajo que implementa, para diseñar los flujos de trabajo, registrar los experimentos y gobernar los modelos y los conjuntos de datos.

Finalmente, a modo ilustrativo, en la figura 5.1 se muestran distintos ejemplos de jugadas de partidos, comparando los valores ofrecidos por el algoritmo "Goal Probability" de LLFP y los obtenidos mediante el sistema desarrollado en el presente trabajo.

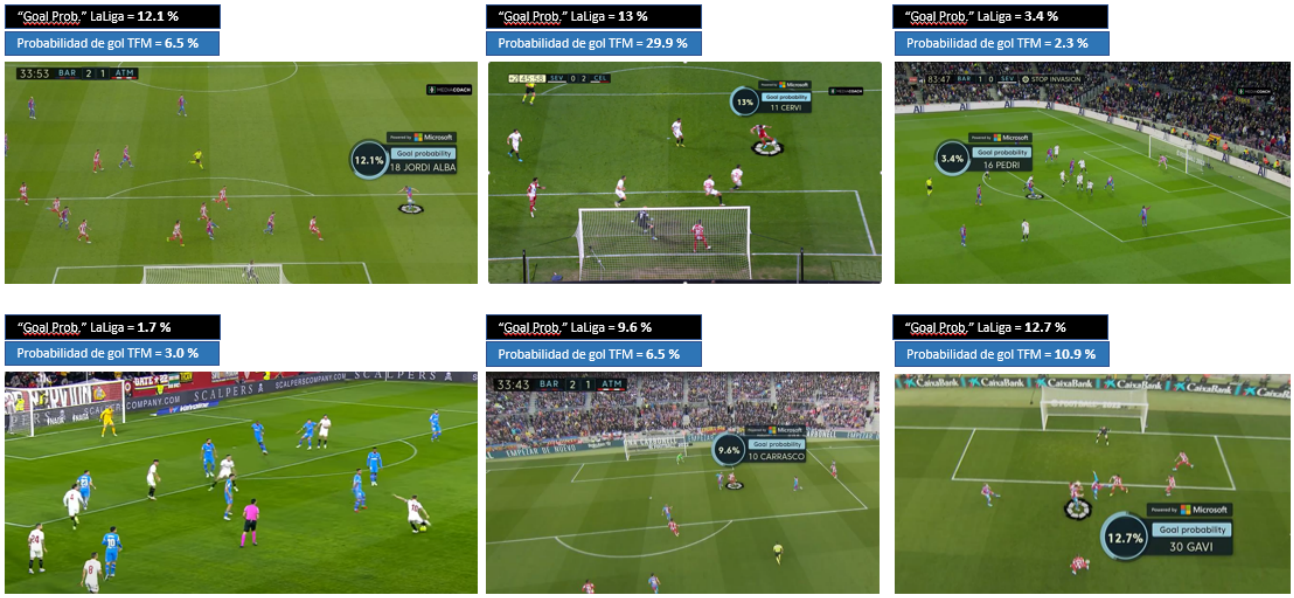

Figura 5.1: Figura que muestra distintas imágenes de disparos a puerta que acabaron el gol junto con la probabilidad ofrecida por el el algoritmo "Goal Probability" de LLFP y los valores obtenidos mediante el sistema desarrollado en el TFM. Imágenes cedidas por LLFP.

5.2. Trabajos futuros

Las limitaciones de la infraestructura computacional disponible para desarrollar el presente trabajo (ver sección 3.4.1.5) ha impedido poder realizar ciclos de entrenamiento de los modelos RNN superiores a los ejecutados en los experimentos. Si bien es cierto que las gráficas de evolución de las métricas AUC-ROR y BCE a lo largo de los ciclos de entrenamientos reflejan que el número de épocas elegido es suficiente, en el caso del experimento AEC_RNN_GRU la gráfica asociada (4.21) aconseja

seguir entrenando el modelo durante más épocas y así comprobar si el modelo sigue evolucionando sin presentar sobreajuste.

En el mismo sentido, una infraestrucutra computacional mas potente hubiera posibilitado:

- El diseño y entrenamiento de autoencoders convolucionales más complejos (con más capas y nodos)
- El diseño y entrenamiento de RNNs más complejas (con más capas y nodos)
- El diseño y entrenamiento de sistemas con una arquitectura como la reflejada en la figura 2.10, en donde, para procesar las secuencias de imágenes, en lugar utilizar el enfoque seguido por el presente TFM en donde se divide el entrenamiento del sistema en dos pasos (un primer paso para generar las representaciones latentes de las imágenes mediante un autoencoder convolucional y un segundo paso para entrenar la RNN alimentada por los vectores generados), se siga una estrategia en donde se efectúe el entrenamiento del sistema de una sóla vez, tomando lotes de secuencias de imágenes de entrada y calculando los pesos, tanto de la parte convolucional como de la parte recurrente, en el mismo proceso mediante la tradicional técnica de retropropagación del gradiente a partir de una única función de pérdida.

Otra posible línea de trabajo a desarrollar una vez completado el último punto anterior, podría ser diseñar y construir otros sistemas alternativos para resolver el problema, como los descritos en la sección 2.2.4 y reflejados en la figura 2.9:

- **Redes Convolucionales 3D** (3D Convnet): son como redes convolucionales estándar, pero con filtros convolucionales espacio-temporales, por lo que crean directamente representaciones espacio-temporales.
- **Redes de 2 Flujos** (Two-Stream Networks): se compone de dos flujos de datos, uno que procesa cada imagen mediante una CNN profunda y otro que procesa, también a través de una CNN profunda, un grupo de secuencias de flujo óptico.
- **Combinación de 3D Convnets y Two-Stream Networks** (TwoStream Inflated 3D ConvNets): bien se superpone una 3D Convnet sobre una Two-Stream Networks, o bien se construye una Two-Stream Networks procesada mediante 3D Convnets en lugar de CNN profundas.

Por otro lado, según se ha indicado en la sección 3.1, existen otros datos que son susceptibles de aportar valiosa información para determinar la dificultad de un disparo a portería y por tanto contribuir al entrenamiento de un modelo que infiera la posibilidad de que el tiro sea gol. Estas fuentes de información complementarias podrían ser: datos de la calidad del jugador (capacidades técnicas, eficacia histórica de cara a puerta, estado de forma,.. y no sólo referida al jugador que remata a puerta, sino también a los defensores o del portero), datos ambientales (información meteorológica, información sobre la sensación de presión ambiental generada por los aficionados,..), información de

la competición (posición del equipo en la clasificación, marcador del encuentro en el momento del disparo, criticidad del resultado del partido para cumplir los objetivos del equipo, ...), información sobre la dificultad del remate (velocidad del balón tanto antes como después del remate, si el remate se realiza con la pierna consonante con la catalogación del jugador, si la posición del jugador que remata no es natural sino forzada,...), etc...Una línea de mejora del trabajo presentado podría ser localizar y emplear en el entrenamiento de los modelos alguna de estas fuentes de información complementarias, para verificar si se consigue mejorar el rendimiento del sistema.

Finalmente, otra posible línea de trabajo a partir de los sistemas desarrollados en el TFM, es la construcción de un proceso de búsqueda de jugadas similares. Las RNNs mantienen ciertos estados ocultos que dependen no sólo del elemento de la secuencia tratada, sino también de los estados obtenidos a partir de los elementos de la misma secuencia pero procesados anteriormente (ver sección 2.2.2). Las unidades GRU y LSTM modulan a través de sus puertas la cantidad de información (cuanta mantener o cuanta excluir) y el tipo (información de corto o largo plazo) que se debe incluir en los estados ocultos al procesar cada elemento. Por tanto, estos estados ocultos no dejan de ser vectores numéricos que condensan la información de toda la secuencia y pueden verse como una representación latente de baja dimensionalidad de las características espacio-temporales de la secuencia completa.

Si se extrae de cada jugada los vectores ocultos ofrecidos por la RNN y se calculan las distancias entre los distintos vectores, intuitivamente se podrían identificar como similares aquellas jugadas cuyas distancias entre vectores ocultos fueran mínimas. Además, a partir de estas distancias se podrían utilizar algoritmos de aprendizaje no supervisado para establecer agrupaciones o clústeres de jugadas similares o que compartan ciertas características comunes.

Bibliografía

Anzer, G. and Bauer, P. (2021). A goal scoring probability model for shots based on synchronized positional and event data in football (soccer). *Frontiers in Sports and Active Living*, 3:624475.

Barnett, V. and Hilditch, S. (1993). The effect of an artificial pitch surface on home team performance in football (soccer). *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 156(1):39.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bialkowski, A., Lucey, P., Carr, P., Yue, Y., Sridharan, S., and Matthews, I. (2014a). Identifying team style in soccer using formations learned from spatiotemporal tracking data. In *ICDMWorkshop on Spatial and Spatiotemporal Data Mining*.

Bialkowski, A., Lucey, P., Carr, P., Yue, Y., Sridharan, S., and Matthews, I. (2014b). Large-scale analysis of soccer matches using spatiotemporal tracking data. In *2014 IEEE International Conference on Data Mining*. IEEE.

Brier, G. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3.

Bröcker, J. and Smith, L. A. (2007). Increasing the reliability of reliability diagrams. *Weather and Forecasting*, 22(3):651–661.

Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset.

Catak, Y., Sahin, K., Guney, O., and Ozkan, H. (2022). *Deep Convolutional Autoencoder Architecture for Predictive Maintenance Applications*, pages 1–4. IEEE, Safranbolu, Turkey.

Chawla, N., Bowyer, K., Hall, L., and Kegelmeyer, W. (2002). Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder-decoder approaches.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Chollet, F. (2017). *Xception: Deep Learning with Depthwise Separable Convolutions*, pages 1800–1807. IEEE, Honolulu, HI, USA.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling.

Creswell, A., Arulkumaran, K., and Bharath, A. A. (2017). On denoising autoencoders trained to minimise binary cross-entropy.

Dal Pozzolo, A., Caelen, O., and Bontempi, G. (2015). *When is Undersampling Effective in Unbalanced Classification Tasks?*, pages 200–215. Springer International Publishing.

Daskalaki, S., Kopanas, I., and Avouris, N. (2006). Evaluation of classifiers for an uneven class distribution problem. *Applied Artificial Intelligence*, 20(5):381–417.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

Donahue, J., Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Darrell, T., and Saenko, K. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2625–2634. IEEE.

Eggels, H. (2016). Expected goals in soccer: explaining match results using predictive analytics. Master's thesis, Eindhoven University of Technology.

Ensum, J., Pollard, Richard, ., and Taylor, S. (2004). Applications of logistic regression to shots at goal in association football: calculation of shot probabilities, quantification of factors and player/team. *Journal of Sports Sciences*.

Fan, Y., Qian, Y., Xie, F.-L., and Soong, F. (2014). *TTS synthesis with bidirectional LSTM based recurrent neural networks*, pages 1964–1968. ISCA.

Feichtenhofer, C., Pinz, A., and Zisserman, A. (2016). *Convolutional Two-Stream Network Fusion for Video Action Recognition*. IEEE.

Finn, C., Goodfellow, I., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

Gopalakrishnan, K. and Salem, F. M. (2020). Sentiment analysis using simplified long short-term memory recurrent neural networks.

Graves, A. (2013). Generating sequences with recurrent neural networks.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognitionwith recurrent neural networks. In *International Conference on Machine Learning*.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly, 2nd edition.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

Heuer, A., Mueller, C., and Rubner, O. (2010). Soccer: is scoring goals a predictable poissonian process? *Europhys. Lett. 89 (2010) 38007*, 89(3):38007.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hoekstra, V., Bison, P., and Eiben, G. (2012). Predicting football results with an evolutionary ensemble classifier. page 68.

Huang, J. and Ling, C. (2005). Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310.

Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study1. *Intelligent Data Analysis*, 6(5):429–449.

Jones, P., James, N., and Mellalieu, S. (2004). Possession as a performance indicator in soccer. *International Journal of Performance Analysis in Sport*, 4(1):98–102.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. *JMLR*.

Kapoor, A., Chollet, F., and Pal, S. (2022). *Deep Learning with TensorFlow and Keras*. Packt Publishing, 3rd edition.

Karlis, D. and Ntzoufras, I. (2003). Analysis of sports data by using bivariate poisson models. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 52(3):381–393.

Kerr, M. G. S. (2015). *Applying machine learning to event data in soccer*. PhD thesis, Massachusetts Institute of Technology.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Landgrebe, T., Paclik, P., and Duin, R. (2006). *Precision-recall operating characteristic (P-ROC) curves in imprecise environments*, pages 123–127. IEEE.

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2015). Ssd: Single shot multibox detector. pages 21–37.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Lucey, P., Oliver, D., Carr, P., Roth, J., and Matthews, I. (2015a). Assessing team strategy using spatiotemporal data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1–9. ACM.

Lucey, P., Oliver, D., Carr, P., Roth, J., and Matthews, I. (2015b). Quality vs quantity: Improved shot prediction in soccer using strategic features from spatiotemporal data. In *9th Annual MIT Sloan Sports Analytics Conference*.

Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. (2011). *Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction*, volume 6791 of *Lecture Notes in Computer Science*, pages 52–59. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proc. Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119.

Mollineda, R., Alejo, R., and Sotoca, J. (2007). *The class imbalance problem in pattern classication and learning*, pages 978–984.

Ng, J. Y.-H., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). Beyond short snippets: Deep networks for video classification.

Niculescu-Mizil, A. and Caruana, R. (1999). Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*, volume 7, pages 625–632. ACM Press.

Pakdaman Naeini, M., Cooper, G., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1):2901–2907.

Platt, J. (1999). *Probabilistic outputs for support vector machinesand comparison to regularized likelihood methods*, pages 61–74.

Pollard, R., Ensum, J., and Taylor, S. (2004). Estimating the probability of a shot resulting in a goal: The effects of distance, angle and space. *International Journal of Soccer and Science*.

Pollard, R. and Reep, C. (1997). Measuring the effectiveness of playing strategies at soccer. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 46(4):541–550.

Radford, A., Wu, J., Child, R., Luan, D., DarioAmodei, and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.

Rathke, A. (2017). An examination of expected goals and shot efficiency in soccer. *Journal of Human Sport and Exercise*, 12(Proc2):S514–S529.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement.

Reep, C. and Benjamin, B. (1968). Skill and chance in association football. *Journal of the Royal Statistical Society. Series A (General)*, 131(4):581.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation.

Rowlinson, A. (2020). Football shot quality. Master's thesis, Aalto University,Espoo, Finland.

Rudd, S. (2011). A framework for tactical analysis and individual offensive production assessment in soccer using markov chains.

Saerens, M., Latinne, P., and Decaestecker, C. (2002). Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. *Neural Computation*, 14(1):21–41.

Saito, T. and Rehmsmeier, M. (2015). The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3):e0118432.

Seyfioglu, M., Ozbayoglu, A., and Gurbuz, S. (2018). Deep convolutional autoencoder for radar-based classification of similar aided and unaided human activities. *IEEE Transactions on Aerospace and Electronic Systems*, 54(4):1709–1723.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting.

Simonyan, K. and Zisserman, A. (2014a). *Two-stream convolutional networks for action recognition in videos*, pages 568–576.

Simonyan, K. and Zisserman, A. (2014b). Very deep convolutional networks for large-scale image recognition.

Srivastava, N., Mansimov, E., and Salakhutdinov, R. (2015). Unsupervised learning of video representations using lstms.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). *Generating Text with Recurrent Neural Networks*.

Sutskever, I., Vinyals, O., and Le, Q. (2014). *Sequence to sequence learning with neural networks*, pages 3104–3112.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions.

Tan, M., Pang, R., and Le, Q. (2020). *EfficientDet: Scalable and Efficient Object Detection*, pages 10778–10787. IEEE, Seattle, WA, USA.

Toma, R. N., Piltan, F., and Kim, J.-M. (2021). A deep autoencoder-based convolution neural network framework for bearing fault classification in induction motors. *Sensors*, 21(24):8453.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). *Learning Spatiotemporal Features with 3D Convolutional Networks*, pages 4489–4497. IEEE.

Uijlings, J., Sande, K., Gevers, T., and Smeulders, A. (2013). Selective search for object recognition. *International Journal of Computer Vision*, 104.

Varol, G., Laptev, I., and Schmid, C. (2017). Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1510–1517.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Yang, K., Cao, Y., Zhang, Y., Fan, S., Tang, M., Aberg, D., Sadigh, B., and Zhou, F. (2021). Self-supervised learning and prediction of microstructure evolution with convolutional recurrent neural networks. *Patterns*, 2(5):100243.

Zadrozny, B. and Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. *ICML*, pages 609–616.

Anexo A

Planificación y presupuesto

A.1. Planificación

A.1.1. Planificación Inicial

Previo al inicio del trabajo se realiza una planificación estimada de esfuerzo necesario para acometer el presente TFM. Las fases y tareas asociadas a dicha planificación son las siguientes:

- **Fase de ideación**: En esta etapa se realizan las siguientes tareas,
  - Se asimila y se comprende el problema que se intenta resolver.
  - Se busca y revisa bibliografía y documentación relacionada para entender el estado del arte y enfocar el modelo de solución.
  - Se investigan arquitecturas de aprendizaje profundo adecuadas para la predicción de secuencias de imágenes.
  - Se diseña a alto nivel, tanto el enfoque general de desarrollo del TFM, como los distintos experimentos que se quieren ejecutar.
- **Fase de recopilación y preparación de datos**:
  - Se buscan los conjuntos de datos adecuados para la tarea.
  - Se realiza el análisis exploratorio de los datos para comprender mejor su naturaleza y la información y limitaciones que contienen.
  - Se limpian y preprocesan los datos para dejarlos listos como entrada de los experimentos.
  - Se dividen los conjuntos de datos para entrenamiento, validación y test.
- **Fase de diseño y selección arquitectura de los modelos**:

  - Se diseña la arquitectura base de los modelos de aprendizaje profundo que se utilizan en los experimentos, incluyendo capas, tamaños de filtros, funciones de activación, tasa de aprendizaje, funciones de optimización, etc...
  - Se eligen las métricas de evaluación apropiadas para medir el rendimiento de los modelos.
  - Se desarrolla el código de entrenamiento y evaluación de los modelos.
  - Se entrenan los modelos y se ajustan los parámetros (capas, tamaños de filtros, funciones de activación, tasa de aprendizaje, funciones de optimización,...), seleccionando aquellos con los que se obtiene un mejor rendimiento.

- **Fase de ejecución y evaluación de los experimentos**:

  - Se diseña la planificación de ejecución de los pasos de los que consta cada experimento.
  - Se realiza la ejecución de cada experimento.
  - Se evalúan los resultados obtenidos por los distintos experimentos para seleccionar el mejor sistema.
  - Se implementan gráficos o visualizaciones para explicar los resultados.
  - Se recopilan las conclusiones.

- **Fase de comparación con el algoritmo "Goal Probability"** :

  - Sobre un conjunto de datos homogéneo, se comparan los resultados de rendimiento del sistema construido en el TFM respecto a los arrojados por el modelo "Goal Probability" de LLFP.

- **Elaboración de la memoria**

  - Se diseña de la estructura de la memoria.
  - Se escribe la memoria.
  - Se prepara la presentación del trabajo y la defensa de la memoria ante el comité de evaluación.

En la tabla A.1 se muestra la estimación de esfuerzo en horas asociada a cada una de las fases del trabajo.

En cuanto a la planificación temporal de las fases y tareas, inicialmente se estimó un calendario de desarrollo que se refleja en la tabla A.2

Fase	Horas
Ideación	50 h
Recopilación y preparación de datos	50 h
Diseño y selección arquitectura de los modelos	75 h
Ejecución y evaluación de los experimentos	65 h
Comparación con el algoritmo "Goal Probability"	10 h
Elaboración de la memoria	50 h
Total	**300 h**

Tabla A.1: Tabla con la estimación inicial de horas de desarrollo asociadas a cada una de las fases en las que se descompone la elaboración del TFM.

Fase	Fecha Inicio	Fecha Fin	Duración
Ideación	21-02-2022	31-03-2022	5 semanas
Recopilación y preparación de datos	01-04-2022	30-04-2022	4 semanas
Diseño y selección arquitectura de los modelos	01-05-2022	30-06-2022	8 semanas
Ejecución y evaluación de los experimentos	01-07-2022	31-08-2022	8 semanas
Comparación con el algoritmo "Goal Probability"	01-09-2022	15-09-2022	2 semana
Elaboración de la memoria	16-09-2022	31-12-2022	14 semanas
Defensa de la memoria	15-02-2023	15-02-2023	-
Total	**21-02-2022**	**15-02-2023**	**41 semanas**

Tabla A.2: Tabla con la planificación temporal inicial de desarrollo asociada a cada una de las fases en las que se descompone la elaboración del TFM.

A.1.2. Planificación Final

Una vez iniciado el proceso de elaboración del TFM se producen desviaciones significativas respecto a la estimación de esfuerzo y la planificación inicial. Los principales motivos que justifican estas desviaciones son los siguientes:

- La planificación inicial atiende a una elaboración de TFM en donde todas las fases se ejecutan de manera secuencial, cuando la realidad es que ha sido un proceso iterativo en algunas partes de las fases. De esta manera, una vez terminada la fase "Ejecución y evaluación de los experimentos" las conclusiones obtenidas aconsejaban ajustar y rediseñar las arquitecturas de los modelos de aprendizaje profundo (fase de "Diseño y selección arquitectura de los modelos"), ajustar el preprocesamientos de datos (fase de "Recopilación y preparación de datos") e incluso la confección de nuevos experimentos (fase de "Ideación"); como, por ejemplo, la incorporación de un filtro gaussiano a la hora de generar las imágenes en escala de grises con la representación de la posición de los jugadores y del balón en cada jugada (ver sección 3.2.1.1).

- El alto volumen de información que contenían los conjuntos de datos utilizados en los diferentes experimentos y las exigencias computacionales que requiere el entrenamiento de los modelos de aprendizaje profundo empleados, impedían que la infraestructura de computación disponible

fuera capaz de procesar los pasos de los experimentos sin agotar sus recursos; concretamente la memoria RAM. Este hecho hizo que se tuvieran que diseñar unos procesos en los que se distribuyera la carga de trabajo: dividiendo el flujo de proceso de cada experimento en pasos, troceando los conjuntos de datos y paralelizando la ejecución de los trabajos (ver sección 3.4.1.6); todo ello no contemplado en la planificación inicial del TFM.

- El ajuste y selección de los parámetros de los modelos de aprendizaje profundo (capas, tamaños de filtros, funciones de activación, tasa de aprendizaje, funciones de optimización,..) implicó una duración muy superior a la inicialmente estimada debido, por un lado, al gran número de combinaciones evaluadas y, por otro, a que cada ciclo de ejecución de un experimento tardaba alrededor de 3 horas de duración de proceso.
- La elevada bibliografía que fue necesario revisar, tanto para documentar el estado del arte del problema que intenta resolver el TFM, como para justificar documentalmente las decisiones metodológicas que sustentan el diseño de los experimentos, hizo que la elaboración de la memoria llevara mucho más tiempo que el inicialmente estimado.
- La acertada revisión de la primera versión de la memoria por parte de los tutores del TFM implicó la corrección y reescritura de alguna de sus partes y el ajuste y reejecución de los experimentos.

Atendiendo a estos motivos, la valoración de esfuerzo en horas dedicadas y la duración real de la elaboración del TFM queda reflejada en la tabla A.3

Fase	Fecha Inicio	Fecha Fin	Duración	Horas
Ideación Inicial	21-02-2022	31-03-2022	5 semanas	50 h
Ideación	01-04-2022	22-12-2022	36 semanas	350 h
Recopilación y preparación de datos				
Diseño y selección arquitectura de los modelos				
Ejecución y evaluación de los experimentos				
Comparación con el algoritmo "Goal Probability"				
Elaboración de la memoria	16-01-2022	28-05-2023	18 semanas	120 h
Corrección y reescritura memoria tras revisión	15-06-2023	15-09-2023	12 semanas	35 h
Defensa de la memoria	15-10-2023	15-10-2023	-	-
Total	**21-02-2022**	**15-02-2023**	**71 semanas**	**555 h**

Tabla A.3: Tabla con la planificación, duración y horas de dedicación real asociadas a cada una de las fases en las que se descompone la elaboración del TFM.

A.2. Presupuesto

El presupuesto económico asignado a la elaboración del TFM se compone de dos factores principales: los costes de personal asociados a la valoración monetaria de las horas empleadas y el coste

de la infraestructura técnica utilizada para el desarrollo y ejecución de los experimentos y el análisis de resultados.

A.2.1. Costes de Personal

Para estimar la valoración monetaria de las horas empleadas, se utiliza la plataforma **malt.es**[1], portal especializado en poner en contacto empresas con profesionales expertos freelance para el desarrollo de proyectos y actividades de tecnología. En dicha plataforma se puede obtener que la tarifa media por día del perfil que mejor se adapta al tipo de tareas desarrollados en el presente TFM (Científico de Datos especialista en Tensorflow con menos de 2 años de experiencia) asciende a **216€**[2].

Utilizando dicha tarifa, y considerando que un día de trabajo contiene 8 horas, obtenemos que la valoración económica de las horas empleadas en la elaboración del TFM es de **14.985€**:

$$\frac{216€}{8h} 555h = 14{,}985€$$

A.2.2. Costes de Infraestructura Técnica

Según se ha indicado en el la sección 3.4.1, se emplea Azure Machine Learning como plataforma integral para el desarrollo de código, para el aprovisionamiento del entorno e infraestructura de cómputo y para la ejecución de los experimentos.

El costes de uso de dicha plataforma se descompone en los siguientes elementos:

- **Costes de uso de máquina virtuales**: Se facturan las horas de utilización de las máquinas virtuales empleadas en la elaboración del TFM (ver sección 3.4.1.5), tanto para el desarrollo de código y el análisis y exploración interactiva de datos mediante Notebooks (instancia Standard_D2s_v3), como para la ejecución de los experimentos (clúster Standard_D12_v2 y Standard_NC6 con GPUS).

  El la tabla A.4 se muestran los precios unitarios asociados al uso de las máquinas virtuales[3], junto con el tiempo de utilización empleado por máquina para desarrollar el TFM y los costes totales que se imputan a cada instancia de cómputo.

- **Costes de almacenamiento**: Azure también factura por los servicios de almacenamiento empleados, los cuales se dividen en dos conceptos:

[1] https://www.malt.es/
[2] https://www.malt.es/t/barometro-tarifas/data/data-scientist/data-scientist-tensorflow
[3] https://azure.microsoft.com/es-es/pricing/details/machine-learning/

Instancia de Computo	Precio Unitario	Horas de uso	Coste
Standard_D2s_v3	0,098€/hora	375 h	36,75€
Standard_D12_v2	0,352€/hora	38 h	13,38€
Standard_NC6	0,885€/hora	98 h	86,73€
Total		**511 h**	**136,86€**

Tabla A.4: Tabla con los precios unitarios, tiempo de uso y coste asociado a cada instancia de cómputo empleada en la elaboración del TFM.

- El almacenamiento en discos SSD de tipo P10 que se utilizan en en el entorno de trabajo de Azure Machine Learning[4]. Este servicio permite guardar los Notebooks y scripts desarrollados para el TFM y los datos temporales necesarios para el análisis y exploración interactiva de los mismos.
- El servicio de almacenamiento Azure Blob Storage[5], vinculado con el entorno de trabajo de Azure Machine Learning, que contiene el Data Storage donde se ubican los conjuntos de datos originales e intermedios necesarios para ejecutar los experimentos (ver sección 3.4.1.2).

Durante el periodo de elaboración del TFM se han incurrido en unos costes de almacenamiento por ambos conceptos de **783€**, lo que supone un coste mensual de 43,5€ a lo largo de los 18 meses de uso de Azure Machine Learning (del 01-04-2022 al 30-09-2023).

A.2.3. Costes totales

Por tanto, atendiendo a lo comentado en los apartados anteriores, el presupuesto económico asignado a la elaboración del TFM asciende a **15.904,86€**. En la tabla A.5 se muestra el resumen de los costes desglosados por los conceptos principales.

Concepto	Subconcepto	Coste
Personal	Científico de Datos TF <2 años exp.	14.985€
Infraestructura	Cómputo	136,86€
	Almacenamiento	783€
Total		**15.904,86€**

Tabla A.5: Tabla con los costes asociados a la elaboración del TFM, desglosados por los conceptos principales.

[4]https://azure.microsoft.com/es-es/pricing/details/managed-disks/
[5]https://azure.microsoft.com/es-es/pricing/details/storage/blobs/