Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Detección de Fenómenos Meteorológicos Adversos en las Islas Canarias vía Autoencoder

Adverse Weather Phenomena detection in the Canary Islands via Autoencoder

Raúl Rodríguez Torres

La Laguna, 30 de junio de 2021

D. **Dagoberto Castellanos Nieves**, con N.I.F. 79.234.766-L profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"Detección de Fenómenos Meteorológicos Adversos en las Islas Canarias vía Autoencoder."

ha sido realizada bajo su dirección por D. **Raúl Rodríguez Torres**, con N.I.F. 51.152.072-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de junio de 2021

Agradecimientos

Agradecer a la Universidad de La Laguna, al profesorado y, en especial, al profesor D. Dagoberto Castellano Nieves la formación que me han procurado durante estos años, además de la ayuda que me han otorgado para facilitarme la elaboración de este Trabajo de Fin de Grado. Agradecer a mi familia, en especial a mi abuela, padres y hermano, toda la confianza depositada, su formación y cariño durante tantos años en los que hemos celebrado juntos las victorias y superado las dificultades. A todas esas amistades, tanto nuevas como antiguas, con las que siempre he podido contar. En especial a Hernán Daniel, Amensay Ginés y Erik Ramos, quienes a día de hoy siguen ahí para lo bueno y lo malo. Agradecer a mi mejor amiga y pareja su apoyo incondicional durante tantos años, en los que ni una sola vez ha dudado y ha estado a mi lado para lograr que viera la luz al final del camino.

Licencia

Resumen

Los fenómenos meteorológicos extremos son un factor de riesgo que han incrementado en intensidad, frecuencia y gravedad. Algunos de estos eventos, como los incendios e inundaciones repentinas, pueden afectar de forma negativa a la salud y a la economía de la población en una región, país o continente. En muchas ocasiones los fenómenos no pueden ser predichos dado que, si analizamos grandes regiones o secciones, los pequeños cambios que generan valores atípicos suelen ser menos notorios. No obstante, estas anomalías podrían ser muy importantes para comprender cambios futuros en el patrón climático.

Por ello, mediante este trabajo se propone la construcción de modelos de aprendizaje profundo para la detección de valores atípicos en diversos conjuntos de datos meteorológicos de las Islas Canarias empleando la técnica no supervisada denominada Autoencoder.

Palabras clave: autoencoder; LSTM; GRU; codificador automático profundo; detección de anomalías; fenómenos meteorológicos adversos

Abstract

Exteme weather phenomenas are a risk factor that has increased in intensity, frequency and severity. Some of these events, such as fires and flash floods, can negatively affect the health and economy of the population in a region, country or continent. On many occasions, the phenomena cannot be predictable since, if we analyze vast regions or sections, the small changes that generate outliers tend to be less noticeable. However, these anomalies could be very significant in understanding future changes in the weather pattern.

Therefore, this project proposes the development of deep learning models for detecting outliers in various sets of meteorological data from the Canary Islands using the unsupervised technique called Autoencoder.

Keywords: autoencoder; LSTM; GRU; deep autoencoder; anomaly detection; adverse weather phenomena;

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Introducción

Los fenómenos meteorológicos extremos son un factor de riesgo que han incrementado en intensidad, frecuencia y gravedad. Algunos ejemplos destacables temporalmente cercanos, durante el año 2020, son la ola de calor marina que sofocó a cerca del 80 % de los océanos, los ocho ciclones tropicales que afectaron Vietnam en solo 5 semanas, el récord de calor en Canadá con una marca de 48.9 °C en Penrith y los diversos incendios en EE. UU. que consumieron aproximadamente 2.6 millones de hectáreas [36].

Algunos de estos eventos, como los incendios e inundaciones repentinas, pueden afectar de forma negativa a la salud y a la economía de la población en una región, país o continente.

En muchas ocasiones los fenómenos no pueden ser predichos dado que, si analizamos grandes regiones o secciones, los pequeños cambios que generan valores atípicos suelen ser menos notorios. No obstante, estas anomalías u outliers podrían ser muy importantes para comprender cambios futuros en el patrón climático. Definimos un outlier [1] como un valor que se espera que esté representado en un conjunto de datos y que puede ser causado por un error arbitrario o por un error sistemático relacionado con cómo son representados los datos. Mientras que definimos una anomalía como un outlier u otro valor que no se espera que exista.

Estos cambios pueden dar lugar a lo que se conoce como *Adverse Weather Phenomena (AWP)* [6], que representa a todo evento atmosférico capaz de producir, directa o indirectamente, daños a las personas o daños materiales de consideración. En sentido menos restrictivo, también puede considerarse como tal cualquier fenómeno susceptible de alterar la actividad humana de forma significativa en un ámbito espacial determinado.

Los datos para la realización de este estudio se comprenden geográficamente en las Islas Canarias: Tenerife, La Palma, El Hierro, La Gomera, Gran Canaria, Fuerteventura y Lanzarote. La elección de este territorio viene determinada por los diversos factores que contribuyen a aumentar los efectos de aquellas amenazas provocadas por los *AWP*. Algunos de estos factores son: la orografía compartimentada y montañosa, los elevados desniveles y la población concentrada en determinados

núcleos que coinciden generalmente con cuencas hidrográficas.

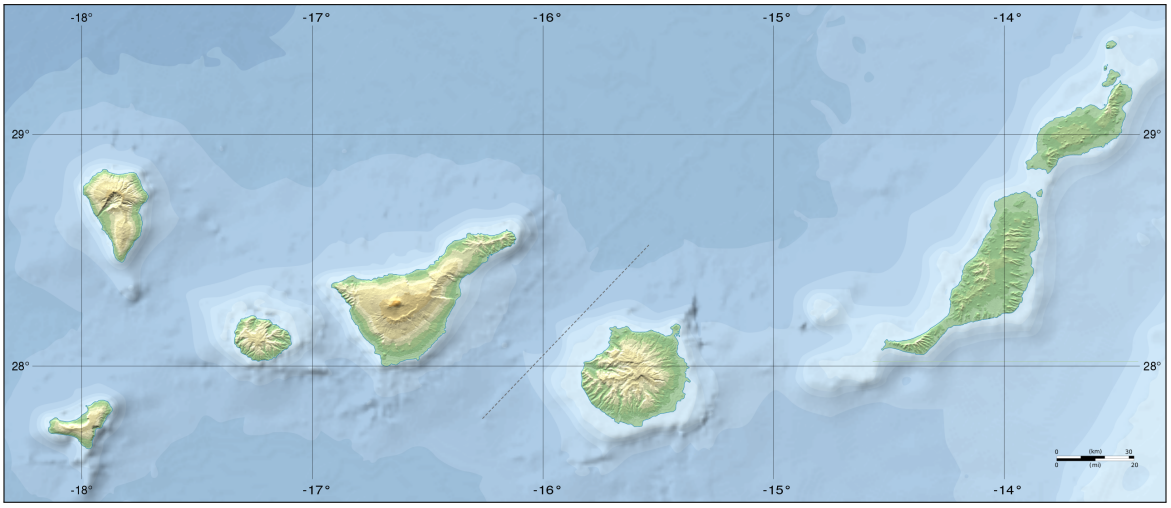


Figura 1.1: Mapa topográfico de las Islas Canarias [27].

1.2. Antecendentes y estado actual del tema

Durante las últimas dos décadas se han producido nuevos desarrollos tecnológicos destinados a crear sistemas de inteligencia artificial que pueden resolver una amplia variedad de tareas como el reconocimiento de voz, procesamiento del lenguaje natural o el reconocimiento de imágenes [7]. En un primer momento la idea era generar aplicaciones que imitaran la forma de pensar de los seres humanos en tiempo real, lo que llevó a una primera aproximación denominada *Perceptron*. Perceptron podía aprender funciones simples para ajustar datos lineales, aunque posteriormente se demostró que tenía ciertas limitaciones, como la incapacidad de aprender funciones simples semejantes a la operación *XOR*. Sin embargo, esto condujo a la idea de comenzar a aplicar diferentes capas a la red y usar la *Delta rule* en su entrenamiento, dando lugar a redes neuronales más complejas que incluyen múltiples capas y capas ocultas. Hoy en día estas arquitecturas son conocidas como *Deep Learning*.

La principal ventaja del Deep Learning es su fácil configuración, la versatilidad, el desarrollo corto y el alto rendimiento. Se han dado múltiples aplicaciones que conducen a las conocidas *Deep Learning Networks*, que han progresado en gran medida en términos de precisión. Estas son aplicadas en muchos campos o áreas de conocimiento para procesado del lenguaje natural [11], reconocimiento de imágenes complejas [43], detección de fraudes [35], detección de anomalías [37], detección de diagnósticos y tratamientos de enfermedades [38], reconocimiento visual [12], coloración de imágenes en blanco y negro [34] y una infinidad de usos adicionales.

Para realizar estas tareas existen diversos tipos de arquitecturas neuronales, entre las que se encuentran los denominados *Autonencoders*. Estos resultan de gran utilidad para la generación de imágenes, detección de opiniones fraudulentas, ge-

neración de sistemas de recomendación y extracción de características importantes en los datos.

Nos centraremos en su aplicación para la extracción de determinadas características y clasificación de los datos meteorológicos seleccionados, con el objetivo de proporcionar una referencia viable que sirva de apoyo para realizar predicciones de posibles *AWP*.

1.3. Objetivo

El objetivo principal del proyecto es identificar las anomalías climatológicas en las Islas Canarias o Fenómenos Meteorológicos Adversos (*Adverse Weather Phenomenas*), apoyándonos en las tecnologías de *Deep Learning* y *Deep Autoencoder* con la intención de generar diversos modelos capaces de identificar esta problemática.

Capítulo 2

Tecnologías y metodología propuesta

En el capítulo anterior realizamos una breve introducción al tema de nuestro proyecto, acompañado de sus objetivos, antecendentes y estado actual en el que nos encontramos. Como hemos podido observar, existen múltiples aplicaciones basadas en *Deep Learning* haciendo uso de diversas arquitecturas para la resolución de una amplia variedad de problemas. Además, hemos comprobado que el desarrollo de este proyecto puede aplicarse para resolver problemas que se ciernen sobre la sociedad.

En este capítulo presentaremos, entre otros, la detección de anomalías, el funcionamiento de la arquitectura autoencoder, el Deep Learning y la metodología que se empleará durante todo el desarrollo.

2.1. Detección de anomalías

Una anomalía es un resultado o valor que se desvía de lo esperado y siempre depende de la situación en la que se encuentre [5]. La detección de anomalías es un proceso que tiene como objetivo identificar casos inusuales dentro de datos aparentemente comparables. Se trata de una herramienta importante para detectar fraudes, intrusiones en la red y otros eventos extraños que pueden tener gran importancia pero son difíciles de encontrar.

Las anomalías pertenecen, generalmente, a una de las siguientes categorías [24]:

- Anomalías puntuales: una sola instancia de datos es anómala si está demasiado lejos del resto.
- Anomalías contextuales: la anomalía es específica del contexto. Este tipo de anomalía es común en los datos de series temporales.
- Anomalías colectivas: si algunos objetos vinculados se pueden observar frente a otros como anomalía. El objeto individual no puede ser anómalo en este caso, solo una colección de estos.

La detección de estos valores es un concepto sencillo pero realmente desafiante. La definición de regiones normales es una tarea difícil debido a que, en muchos

casos, los límites entre las anomalías y los datos normales no son precisos. En este caso las observaciones normales pueden ser representadas como anomalías y viceversa. Lo que actualmente se considera normal puede no serlo en un futuro y la mayoría de enfoques empleados en un campo para la detección de anomalías pueden no ser válidos para otro. De esta manera, los hoy considerados datos meteorológicos anómalos en unos años podrían ser claramente normales, debido a diversos factores como el cambio climático.

Existen tres estilos de detección de anomalías [3]:

- Detección de anomalías de forma supervisada: es una técnica en la que los datos de entrenamiento tienen etiquetas tanto para anomalías como para datos normales. Básicamente, se le indica al modelo durante el proceso de entrenamiento si un registro es una anomalía o no.
- Detección de anomalías de forma no supervisada: implica entrenar el modelo con datos sin etiquetar. Después del proceso de entrenamiento, se espera que el modelo sepa qué datos son normales y cuales son anómalos dentro del conjunto.
- Detección de anomalías de forma semi-supervisada: implica etiquetar parcialmente el conjunto de datos de entrenamiento. En el contexto de la detección de anomalías, este puede ser un caso en el que solo se etiqueten los datos normales.

Para el desarrollo de este proyecto hemos decidido emplear un estilo de detección de anomalías no supervisado basado en una técnica denominada *Autoencoder*.

2.2. Autoencoder en la detección de FMA

Los autoencoders [4] son redes neuronales que poseen la habilidad de descubrir representaciones de baja dimensionalidad de datos con alta dimensionalidad. Se basan en la compresión de la entrada, reduciendo su dimensionalidad, para posteriormente reconstruir la salida a partir de esta representación. (Fig 2.1).

Los autoencoders constan de 3 partes principales:

- Encoder: reduce la dimensionalidad partiendo de un conjunto de datos de alta dimensionalidad. Su objetivo es reducirla preservando la información que realmente es relevante para el aprendizaje y descartando aquella que es irrelevante.
- Decoder: expande un conjunto de datos de baja dimensionalidad en un conjunto de alta dimensionalidad. Aprende a coger los datos procedentes del encoder y a reconstruir los valores de entrada a partir de estos.
- Función de pérdida: conocida también como pérdida de reconstrucción, es el error medido entre la entrada y la salida. Esta función penaliza a la red

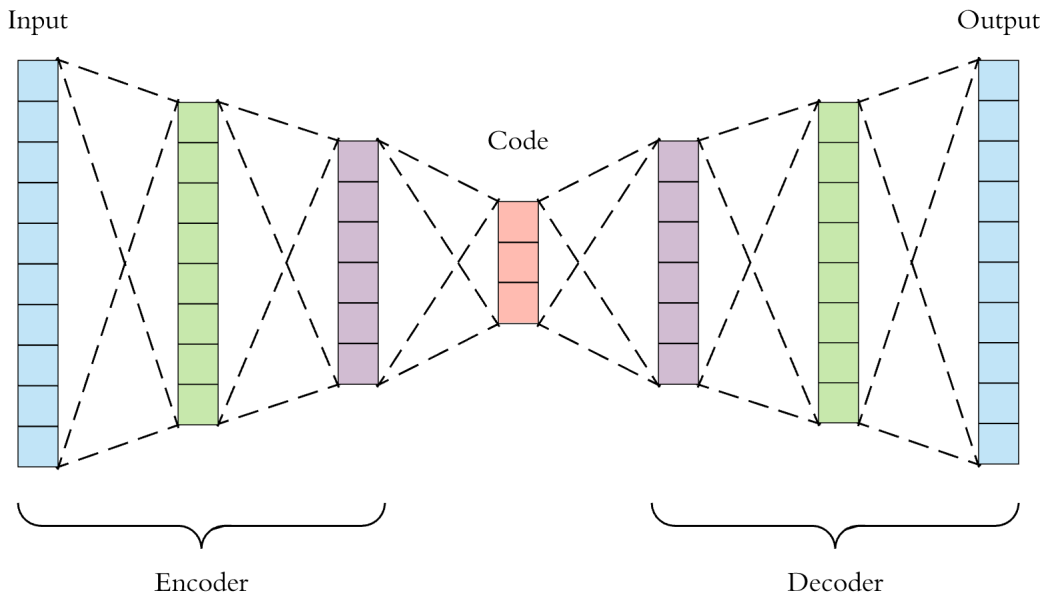


Figura 2.1: Arquitectura de un Autoencoder [13].

por crear salidas diferentes de la entrada. Normalmente, para clasificaciones binarias se emplea el *Mean Squared Error* o la *Cross Entropy*. Este error es el que nos permite diferenciar si una entrada se encuentra dentro del rango normal o fuera de este. Para ello se emplea un término conocido como threshold o umbral, que hace referencia al valor mínimo del error de reconstrucción a partir del cual clasificaremos registros como anómalos.

Dado el funcionamiento de este tipo de algoritmos, es importante tener en cuenta ciertas consideraciones [13]: (1) Los autoencoders solo son capaces de comprimir adecuadamente datos similares a los que ha empleado para su entrenamiento. (2) Se trata de una técnica no supervisada, por lo que no necesita de etiquetado de la variable objetivo para su entrenamiento. No obstante, se autosupervisan porque generan sus propias etiquetas a partir de los datos de entrenamiento. (3) La salida del autoencoder no será exactamente la misma que la entrada, será una representación cercana pero degradada.

En las soluciones basadas en autoencoders se destacan los siguientes hiperparámetros:

1. Code size: número de nodos en la capa central. Se trata de la capa más pequeña del conjunto.

2. Número de capas: número de capas sin contar las capas de entrada y salida. Tanto el encoder como el decoder poseen el mismo número de capas.

3. Número de nodos por capa: número de nodos que irán trabajando en cada capa. Como se puede observar en la figura 2.1, los nodos del encoder van reduciendose hasta llegar a la capa central, donde comenzamos a reconstruir

la entrada con un decoder que replica el encoder en número de capas y nodos.

4. Función de pérdida: empleamos el *Mean Squared Error* o la *Cross Entropy*, tal y como comentamos previamente.

Atendiendo a nuestras necesidades de proyecto, haremos uso de los *Deep Autoencoders*. Estos poseen un conjunto de capas adicionales a las capas básicas de entrada, salida y code a las que se denomina capas ocultas. Como su principal cambio es el número de capas se les puede considerar como autoencoders multicapa.

2.3. Deep Learning

El aprendizaje profundo o Deep Learning es un subcampo especial del Machine Learning (ML) basado en redes neuronales artificiales. Estas redes se basan en la estructura y funcionalidad del cerebro humano, en el proceso iterativo y progresivo de aprendizaje a través de la experimentación. Su núcleo se encuentra formado por una serie de capas que contienen un conjunto de unidades individuales, denominadas neuronas, que se encuentran interconectadas y que transmiten determinadas señales desde la entrada hasta generar una salida. Estas capas forman diversos niveles jerárquicos que dividen el proceso de aprendizaje. En el nivel inicial, la red aprende algo simple y luego envía esta información al siguiente nivel. El siguiente nivel toma esta información sencilla, la combina, compone una información más compleja, se la pasa al siguiente nivel y así sucesivamente.

El aprendizaje de estos algoritmos puede seguir tres posibles esquemas [26][33]:

- Aprendizaje supervisado: el algoritmo aprende sobre el conjunto de datos de entrenamiento realizando predicciones iterativas sobre los datos y ajustando la respuesta correcta. Este tipo de algoritmos requiere de intervención humana para etiquetar los datos de manera adecuada, es decir, debe estar presente la variable objetivo de la predicción. Los problemas de clasificación binaria mediante regresión logística son un buen ejemplo de este tipo de técnica.

- Aprendizaje no supervisado: estos algoritmos funcionan por sí mismos para descubrir la estructura inherente de los datos sin etiquetar, es decir, que la variable objetivo no se encuentra presente en los datos de entrenamiento. No obstante, se debe tener en cuenta que sigue requiriendo de intervención humana para validar las variables de salida.

- Aprendizaje semi-supervisado: estos algoritmos funcionan empleando una gran cantidad de datos no etiquetados y una pequeña cantidad de datos etiquetados.

Independientemente de la técnica de aprendizaje escogida, estos algoritmos requieren de conjuntos de datos grandes para su entrenamiento. Este aspecto, junto a la complejidad del algoritmo, produce uno de los mayores problemas del Deep Learning, la potencia de computo necesaria. Una capa más grande proporciona mayor flexibilidad y es capaz de modelar funciones más complejas, pero requiere de un mayor número de datos de entrenamiento.

Puesto que existen numerosas aplicaciones o áreas de aplicación para el aprendizaje profundo, existen múltiples tipos de redes neuronales con arquitecturas y características diferentes. En este caso nos centraremos en el uso de redes neuronales recurrentes para la resolución de nuestra problemática.

2.4. Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN) son un tipo de red neuronal que forman una estructura en cadena en la que la salida de cada capa sirve de entrada para la siguiente. Este proceso se realiza de forma recursiva hasta producir la salida final. Se basan en aprender patrones de las secuencias al depender cada salida de los resultados del paso previo. Pueden ser definidas como redes con una memoria interna que permite guardar información relevante sobre la entrada recibida, lo que proporciona la capacidad de que sean muy precisas. En nuestro caso proporciona una gran precisión al tener acceso al histórico climatológico. Se aplican en numerosos campos para la resolución de problemas como detección de fraudes [8], resumen de textos [10], traducción automática [29] y detección de anomalías [14].

Estas redes suelen seguir una estructura como la que se muestra en la figura 2.2 donde para cada timestep t, la función de activación $a^{<t>}$ y la salida $y^{<t>}$ se expresan mediante las siguientes ecuaciones [9]

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{2.1}$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \tag{2.2}$$

donde $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ son coeficientes que se comparten temporalmente y g_1, g_2 funciones de activación.

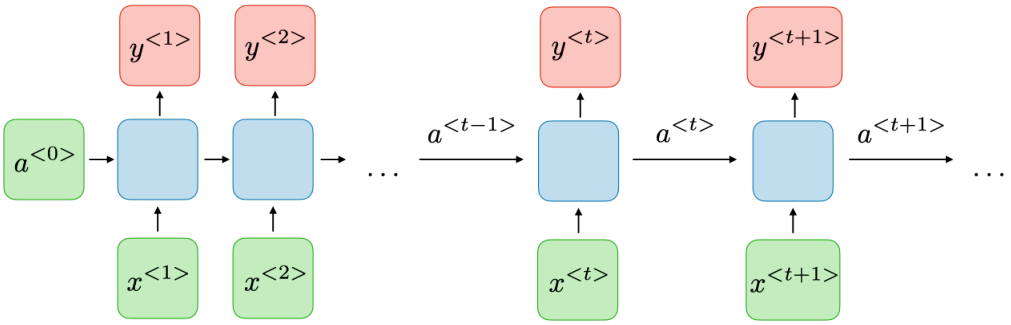


Figura 2.2: Arquitectura típica de una RNN [9].

Este tipo de redes poseen cinco variantes:

- Una entrada para una salida (One To One): usado en redes neuronales tradicionales donde $T_x = T_y = 1$.
- Una entrada para múltiples salidas (One To Many): usado en generación de música donde $T_x = 1, T_y > 1$.
- Múltiples entradas para una salida (Many To One): usado en clasificación de sentimientos donde $T_x > 1, T_y = 1$.
- Múltiples entradas para múltiples salidas (Many To Many): usado en clasificación de entidades nombradas donde $T_x = T_y$ o en traducciones donde $T_x \neq T_y$.

Las RNN se pueden ver afectadas por dos problemas denominados *gradiente explosivo* y *desvanecimiento de gradiente*. Un error de gradiente es la dirección y magnitud calculadas durante el entrenamiento de una red neuronal que se utiliza para actualizar los pesos de esta en la dirección y cantidad correctas. En este tipo de redes, los gradiente de error pueden acumularse durante una actualización y dar como resultado gradientes muy grandes. Estos, a su vez, dan como resultado grandes actualizaciones de los pesos que provocan inestabilidad en la red. Esto es lo que se conoce como un gradiente explosivo. Esta circunstancia puede provocar que la red no pueda aprender de los datos de entrenamiento y, en el mejor de los casos, que no pueda aprender sobre largas secuencias de entrada. La explosión se produce mediante un crecimiento exponencial al multiplicar repetidamente los gradientes a través de las capas de la red que tienen valores superiores a 1,0. Por otro lado, se puede producir el caso contrario y es que el gradiente baje tan rápido que dificulte el aprendizaje de algunas dependencias en largas secuencias.

Nombre	**Ecuación**
Gradiente Explosivo	$\|\|\frac{\partial h_i}{\partial h_{i-1}}\|\|_2 > 1$
Desvanecimiento de Gradiente	$\|\|\frac{\partial h_i}{\partial h_{i-1}}\|\|_2 < 1$

Tabla 2.1: Problemas de gradiente en RNN.

Para solventar esta problemática aparecen las capas *GRU (Gated Recurrent Unit)* y *LSTM (Long Short-Term Memory)*, que proporcionan mecanismos internos llamados *gates* que permiten regular el flujo de información. Estas compuertas pueden aprender que datos en la secuencia es importante preservar o cuales debe descartar, permitiendo pasar únicamente información relevante a lo largo de la secuencia para realizar las predicciones [31] [30]. (Fig. 2.3).

La diferencia clave entre ambas reside en que GRU posee dos compuertas, *reset* y *update*, mientras que LSTM posee tres, *input*, *output* y *forget*. Normalmente, es preferible GRU para conjuntos de datos pequeños, dada su menor complejidad, y LSTM para conjuntos de datos más grandes. Además, la reducción de complejidad de GRU hace que tienda a ser más eficiente computacionalmente [31].

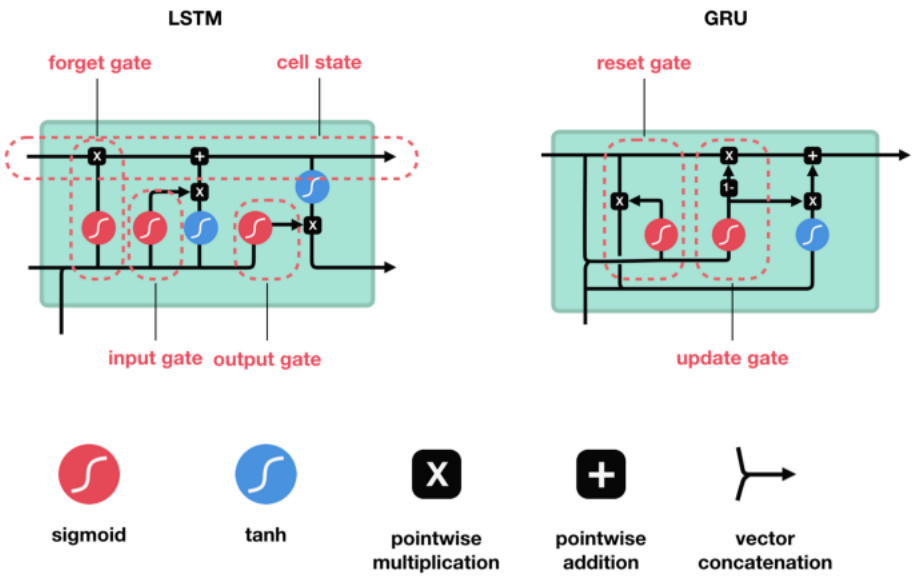


Figura 2.3: Arquitectura de las capas GRU y LSTM [31].

Si exponemos las principales ventajas e inconvenientes de emplear redes neuronales recurrentes, podríamos destacar las siguientes:

Ventajas	**Desventajas**
Pueden modelar secuencias de datos en las que cada muestra puede asumirse que depende de la anterior	Problemas de gradiente explosivo y desvanecimiento de gradiente
Pueden ser usadas incluso con capas convolucionales para extender la efectividad del pixel vecino	Entrenamiento difícil
Pueden procesar entradas de cualquier tamaño	Cómputo lento en muchos casos
Incrementar el tamaño de la entrada no aumenta el tamaño del modelo	Difícil acceso a información antigua

Tabla 2.2: Ventajas y desventajas de emplear redes neuronales recurrentes.

2.5. Tecnologías empleadas

Como ya se definió previamente en los objetivos, nuestro proyecto consiste en la identificación de fenómenos meteorológicos adversos en el archipiélago canario apoyándonos en el uso de Deep Autoencoders. Para ello, hemos considerado la utilización de una serie de tecnologías que reflejamos a continuación.

Como lenguaje de programación para el desarrollo de los modelos utilizaremos *Python 3* por su amplia selección de bibliotecas y marcos, su simplicidad y el apoyo que recibe al contar con una gran cantidad de recursos y documentación de alta calidad. Como entorno de desarrollo hemos optado por *JupyterLab* [20] por su flexibilidad a la hora de poder ejecutar fragmentos de código a elección, por su interfaz gráfica usable y accesible y por su rendimiento. Un cuaderno Jupyter realizado en el proyecto se encuentra recogido en el Apéndice C.

Para la construcción de los modelos nuestra elección ha sido emplear *Keras*, *Tensorflow* y *Scikit-learn*. Keras [41] es una librería de código abierto escrita en

Python para redes neuronales capaz de ejecutarse sobre Tensorflow [40], una biblioteca de código abierto desarrollada por Google para aprendizaje automático a través de un rango de tareas. Keras nos proporciona los elementos necesarios para contruir la estructura del modelo (capas, regularizadores, etc) y Tensorflow el entorno para el desarrollo de estos modelos de múltiples capas. Por otro lado, Scikit-learn [25] es una librería de código abierto escrita en Python para aprendizaje automático que nos proporciona las métricas necesarias para la evaluación de estos modelos.

Como librerías para el manejo y análisis de datos emplearemos *Pandas* y *NumPy*, una biblioteca que da soporte para crear vectores y matrices grandes multidimensionales.

Por último, para el preprocesado de datos hemos decidido emplear R [42] por su facilidad de uso, flexibilidad y amplia variedad de librerías para el procesamiento de datos y estudio estadístico. Las librerías empleadas se encuetran recogidas en la tabla 2.3. Como entorno de desarrollo para este lenguaje emplearemos RStudio [22], un entorno desarrollado por la misma compañía que cubre con todas las necesidades.

Librería	**Descripción**
dplyr	Manipulación y operaciones con data frames
readxl	Extracción y exportación de datos en excel
readr	Lectura de datos rectangulares como csv y excel
corrplot	Visualización de matrices de correlación e intervalos de confianza

Tabla 2.3: Librerías empleadas en R.

2.6. Metodología

La metodología que proponemos recoge elementos aceptados en la ciencia de datos. La implementación de los algoritmos se realizó de forma exploratoria e iterativa, pasando de técnicas simples a técnicas más sofisticadas. Los fases principales que destacamos serían: (1) Preprocesamiento de datos (extracción de características, imputación, selección y escalado); (2) Creación y compilación de modelos (LSTMs y GRUs); (3) Entrenamiento de los modelos, (4) Evaluación de los modelos; (5) Validación del error de reconstrucción; (6) Selección del umbral; (7) Detección y predicción de anomalías. (Fig. 2.4).

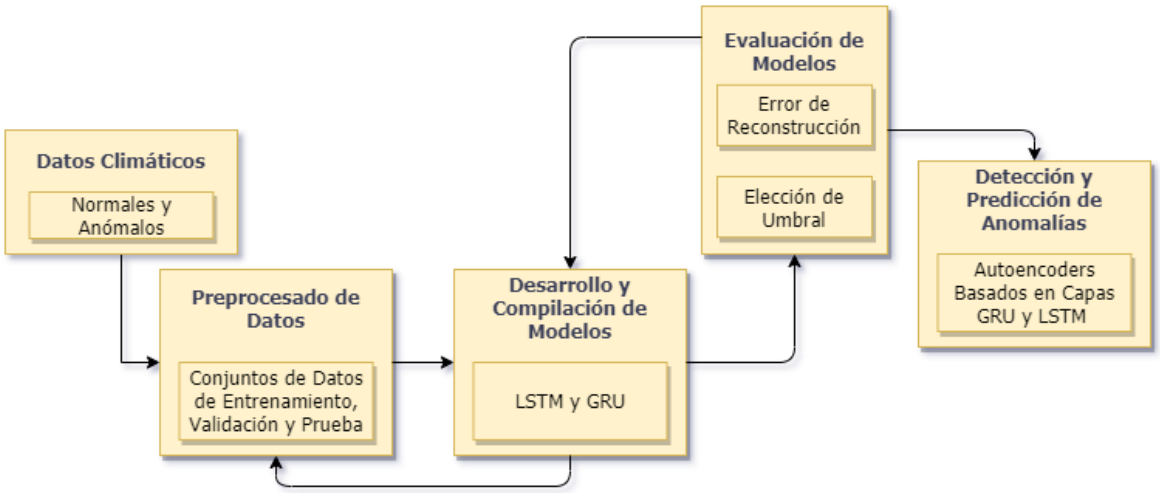


Figura 2.4: Metodología general aplicada.

Estas fases pueden ser subdivididas en una serie de tareas clave:

- **Elección de los conjuntos de datos:** recolección y análisis de los datos.
- **Preprocesado de los datos:** manipulación de los datos con el objetivo de obtener registros de calidad para el entrenamiento de los modelos.
- **Modelado de los datos:** modificación de la forma de los datos. Este paso es necesario para las capas LSTM, cuya entrada viene dada en tres campos: samples, timesteps y features. Estos campos hacen referencia al número de muestras, el número de pasos previos a recordar y el número de características disponibles respectivamente.
- **Generación del modelo a partir de la línea base:** establecimiento del modelo de partida siguiendo la línea base para obtener los primeros resultados.
- **Entrenamiento:** establecimiento de los parámetros de entrenamiento y ejecución del mismo.
- **Evaluación del modelo:** análisis de los resultados obtenidos durante el entrenamiento.
- **Selección del umbral:** establecimiento del punto de corte a emplear para el filtrado de datos en función del error de reconstrucción.
- **Ajuste de hiperparámetros:** ajuste de parámetros de entrenamiento con el objetivo de mejorar los resultados.
- **Modificación del modelo:** si los resultados obtenidos son muy deficientes, es recomendable modificar directamente el modelo. Si aun así este continua siendo ineficiente, es aconsejable volver al procesamiento de datos.

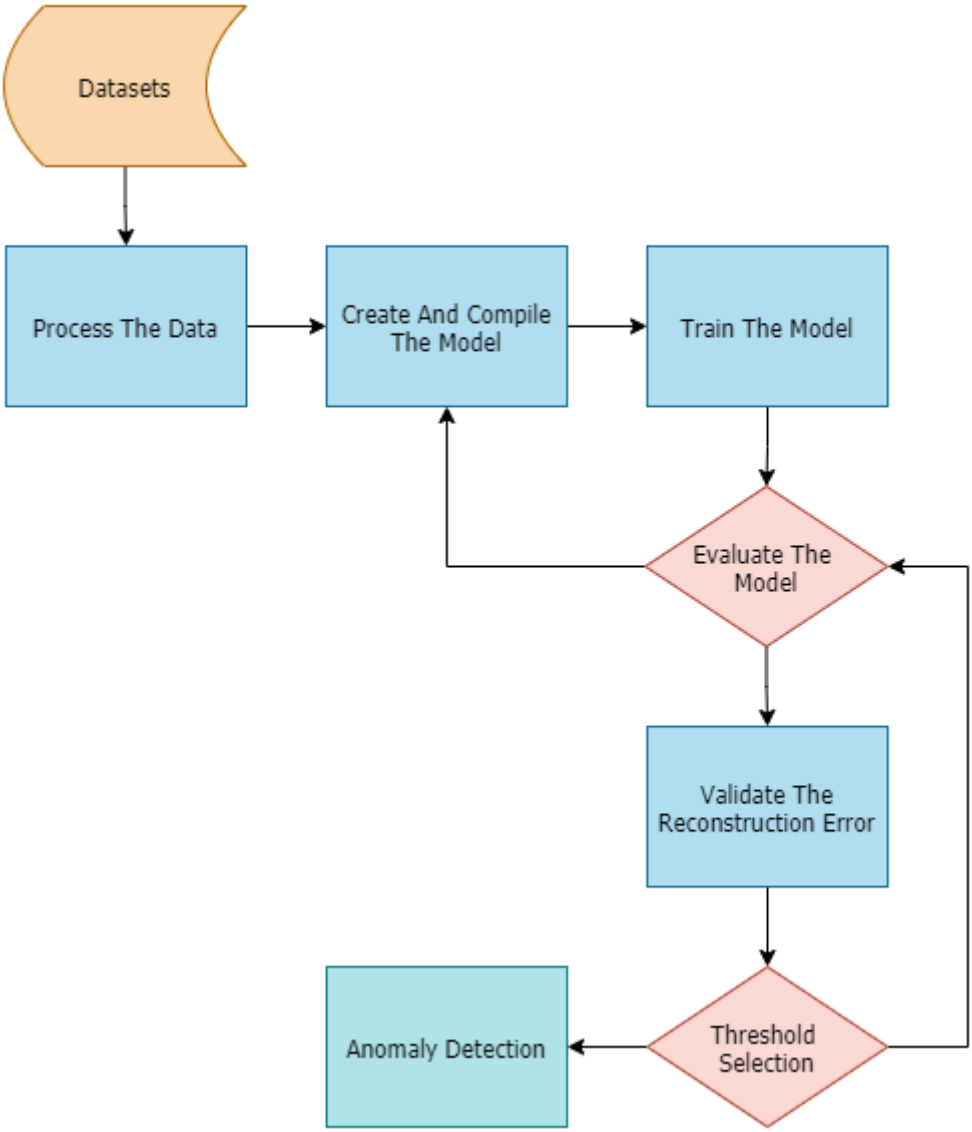


Figura 2.5: Flujo de trabajo propuesto para la obtención de resultados.

Durante este capítulo se ha realizado una introducción a los diversos conceptos aplicados en nuestro proyecto. En los siguientes tópicos se profundizará, siguiendo la metodología propuesta, en cada una de las etapas para cumplir los objetivos.

Capítulo 3

Elección y procesado de datos climatológicos de Canarias

En este capítulo se expone la recuperación y elección de los datos, los criterios empleados y las diversas técnicas aplicadas para su procesamiento.

3.1. Selección de datos

Los conjuntos de datos empleados en nuestro proyecto provienen de la NOAA (National Oceanic and Atmospheric Administration) [28]. Se trata de una organización americana responsable de describir y predecir los cambios en el medio ambiente mediante la investigación de los océanos, la atmósfera, el espacio y el sol. Su principal función es entender y predecir los cambios en el ambiente de nuestro planeta, tanto a nivel atmosférico como oceánico.

Dada la naturaleza del proyecto se requiere de conjuntos de datos grandes para el entrenamiento de las redes neuronales. Por lo tanto, se recogieron los datos comprendidos en los siguientes rangos anuales. (Tabla 3.1).

Datasets (D_n)	**Isla**	**Años**
D_1	Lanzarote	1978 to 2018
D_2	Fuerteventura	1974 to 2019
D_3	Gran Canaria	1974 to 2019
D_4	La Gomera	2004 to 2019
D_5	El Hierro	1974 to 2019
D_6	La Palma	1974 to 2019
D_7	Tenerife	1974 to 2019

Tabla 3.1: Conjuntos de datos recogidos por isla y serie temporal de la NOAA.

La recopilación de estos datos es realizada durante años por diversas estaciones meteorológicas ubicadas en puntos concretos de las islas. Encontramos un total de siete estaciones, una por isla, con las siguientes referencias y coordenadas. (Tabla 3.2).

USAF	Estación	Lat	Lon	Elev (m)
600010	El Hierro	27.815	-17.887	+0031.4
600050	La Palma	28.626	-17.756	+0032.6
600070	La Gomera	28.033	-17.217	+0219.0
600150	Tenerife Norte	28.483	-16.342	+0631.9
600300	Gran Canaria	27.932	-15.387	+0023.8
600350	Fuerteventura	28.453	-13.864	+0025.3
600400	Lanzarote	28.945	-13.605	+0014.3

Tabla 3.2: Principales estaciones meteorológicas en las Islas Canarias.

Toda esta información es útil relacionarla con los parámetros globales de las islas, con lo que podemos obtener datos de interés como la altitud de la estación respecto a la altitud y área de la isla o la complejidad orográfica. (Tabla 3.3)

Isla	Altitud (m)	Área (km^2)	Altitud/Área $(10^{-3}/km)$	Orografía Compleja
Lanzarote	670	790.5	0.85	Media
Fuerteventura	807	1633.3	0.49	Baja
Gran Canaria	1426	1529.8	0.93	Media
La Gomera	1487	359.1	4.14	Alta
El Hierro	1500	290.5	5.16	Alta
La Palma	2423	690.2	3.51	Alta
Tenerife	3718	2007.8	1.85	Alta

Tabla 3.3: Islas por altitud y orografía.

La orografía es un factor de gran importancia que nos ayuda a comprender la tendencia de ciertos valores. Por ejemplo, una zona con una orografía compleja presenta, normalmente, zonas montañosas de gran altitud que dan lugar a lo que se denomina *precipitación orográfica* [15]. La precipitación orográfica es aquella generada por el ascenso de una masa de aire forzada al encontrar un obstáculo en el relieve topográfico. Por lo tanto, la orografía compleja puede provocar diferencias en los niveles de precipitación y otros eventos meteorológicos tanto entre islas como entre diversas zonas de la misma.

3.2. Exploración de datos

Antes de comenzar con el preprocesamiento de datos se debe realizar una exploración de estos. En esta sección mostraremos las características de los conjuntos de valores junto a su extensión y método de recolección.

3.2.1. Características de los datos

Cada conjunto de datos posee un número variable de características o atributos que hacen referencia a valores como la fecha y hora del registro, la precipitación de lluvia en el instante de tiempo, el viento, etc. De forma general, hace referencia a los valores que la tecnología presente es capaz de recoger.

Puesto que las estaciones meteorológicas pueden variar de unas a otras, incluso la misma estación debido a actualizaciones a lo largo de los años, destacamos los siguientes campos que son una base importante y siempre presente en los diferentes conjuntos. (Tabla 3.4).

Campo	Nombre	Descripción	Unidad
time	Time	El tiempo de una observación en un punto geográfico	UTC
temp	Temperature	Temperatura del aire	$°C$
wd	Wind Direction	El ángulo, medido en el sentido de las agujas del reloj, entre el norte verdadero y la dirección desde la que sopla el viento	$Grados(°)$
ws	Wind Speed	La tasa de viaje horizontal del aire más allá de un punto fijo	$metros/segundo$
atmos_pres	Atmospheric Pressure	La presión atmosférica en el punto de observación	hPa
dew_point	Dew Point	La temperatura a la que se debe enfriar una determinada porción de aire a presión y contenido de vapor de agua constantes para que se produzca la saturación	°C
rh	Relative Humidity	Relación adimensional, expresada en porcentaje, de la cantidad de humedad atmosférica presente en relación con la cantidad que estaría presente si el aire estuviera saturado	%
ceil_hgt	Ceiling Height	La altura sobre el nivel del suelo (AGL) de la nube más baja o la capa de fenómenos oscurecedores en alto con 5/8 o más cielo cubierto, que puede ser predominantemente opaca, o la visibilidad vertical en una obstrucción basada en la superficie	$metros$
visibility	Visibility	La distancia horizontal a la que se puede ver e identificar un objeto	$metros$
aa1_1	Liquid Precipitation Period Quantity	Cantidad de tiempo durante el cual se midió la precipitación de líquido	$Horas$
aa1_2	Liquid Precipitation Depth Dimension	La profundidad de precipitación de líquidos que se mide en el momento de una observación	$milimetros$
aa1_3	Liquid Precipitation Condition Code	Código que denota si una dimensión de profundidad de precipitación de líquido era un valor de traza	$Entero$ [17]
aa1_4	Liquid Precipitacion Quality Code	Código que denota un estado de calidad de los datos de precipitación de líquidos notificados	$Entero$ [18]

Tabla 3.4: Atributos destacables de los conjuntos de datos [16].

3.2.2. Método de recolección y extensión de los conjuntos de datos

Los diferentes conjuntos de datos están formados por múltiples registros que han sido recogidos utilizando un método temporal que divide cada día en diversas entradas. Estas son recogidas a determinadas horas del día, por lo que un mismo día posee un número de entradas superior a uno que recoge la información en ese momento concreto.

Debido a esta forma de recolección, uno de los puntos claves del procesamiento de datos es unificar los múltiples valores diarios para obtener una única entrada por día.

Isla	Número de Mediciones
Lanzarote	489.455
Fuerteventura	451.628
Gran Canaria	677.963
La Gomera	38.689
El Hierro	253.836
La Palma	373.063
Tenerife	498.712

Tabla 3.5: Número de registros iniciales por isla.

3.3. Preprocesamiento de datos

Tras el estudio de los diferentes conjuntos de datos es hora de procesarlos para valorar la calidad y la cantidad de aquellos con los cuales trabajaremos. La elección de datos y su estudio preliminar fue realizado en el capítulo anterior. Para la obtención de los conjuntos de datos finales dividiremos el proceso en las siguientes etapas.

3.3.1. Reducción de dimensionalidad

Los datos recogidos presentan una dimensionalidad muy amplia, es decir, presentan una gran cantidad de características o atributos. Por ello, es importante reducir la dimensionalidad con el objetivo de preservar aquellos de los que poseemos suficientes valores no nulos que serán realmente útiles para nuestras predicciones. Con esto no solo facilitamos nuestro trabajo para manejar los datos, sino que también facilitamos el aprendizaje y el cómputo de los algoritmos simplificándolos en gran medida.

Los campos preservados hacen referencia a los previamente destacados en la tabla 3.4.

3.3.2. Limpieza y transformación de datos

Este proceso permitirá identificar datos incoherentes o incompletos para aplicar una serie de técnicas con las que podremos modificarlos, eliminarlos o formatearlos.

En primer lugar, realizamos un ajuste de aquellos datos que poseen fallos de medición o cuyos valores se encuentran fuera del rango natural. Estos desajustes se pueden dar, por ejemplo, por fallos en el propio hardware de la estación meteorológica. En los conjuntos de datos nos encontramos con fallos en las precipitaciones de tipo `aa1_2`, por lo que convertimos los valores iguales o superiores a 999.9 a 0. (Listado 3.1).

```
Data$aa1_2[Data$aa1_2 >= 999.9] <- 0.0
```

Listado 3.1: Modificación de valor para las precipitaciones con fallos de medición. Sustitución de valores superiores a 999.9 por 0.0.

Posteriormente detectamos que la mayoría de conjuntos de datos poseen valores reflejados como *INF* o *-INF*. Estos valores no aportan nada para el posterior análisis, por lo que los sustituimos por el valor *NA*. Este valor hace referencia a lo que se denomina un *Missing Value*, un valor no presente en el campo. Se trata de una forma útil de sustituir un dato puesto que no afecta a futuros cálculos que podamos realizar sobre el conjunto de valores. (Listado 3.2).

```
Data_sin_INF$temp[!is.finite(Data_sin_INF$temp)] <- NA
Data_sin_INF$wd[!is.finite(Data_sin_INF$wd)] <- NA
Data_sin_INF$ws[!is.finite(Data_sin_INF$ws)] <- NA
Data_sin_INF$atmos_pres[!is.finite(Data_sin_INF$atmos_pres)] <- NA
Data_sin_INF$dew_point[!is.finite(Data_sin_INF$dew_point)]<- NA
Data_sin_INF$rh[!is.finite(Data_sin_INF$rh)]<- NA
Data_sin_INF$ceil_hgt[!is.finite(Data_sin_INF$ceil_hgt)]<- NA
Data_sin_INF$visibility[!is.finite(Data_sin_INF$visibility)]<- NA
Data_sin_INF$aa1_1[!is.finite(Data_sin_INF$aa1_1)]<- NA
Data_sin_INF$aa1_2[!is.finite(Data_sin_INF$aa1_2)]<- NA
Data_sin_INF$aa1_3[!is.finite(Data_sin_INF$aa1_3)]<- NA
Data_sin_INF$aa1_4[!is.finite(Data_sin_INF$aa1_4)]<- NA
```

Listado 3.2: Sustitución de valores INF y -INF en los diferentes predictores de los datos por NA.

Como ya comentamos previamente las entradas se encuentran recogidas por fecha y hora, por lo que tenemos múltiples entradas para el mismo día. Nuestro objetivo es unificar todas estas entradas para poder trabajar el conjunto como registros diarios únicos. Para ello realizamos la división del campo date, en año, mes y día, y realizamos la agrupación y sumarización de atributos aplicando en cada caso que se realice la media de todos los valores de un mismo día. Con esto lo que conseguimos es obtener la unificación de registros diarios y que, para cada entrada, el valor de sus campos sea la media de todos los valores recogidos durante el transcurso del mismo.

Aprovechando este último proceso, realizamos la adición de nuevos campos con información extraida del conjunto. Ciertos campos, como la temperatura y la presión atmosférica, poseen información que se puede extraer y que resulta de interés para los modelos como son sus valores mínimo, máximo y medio. (Listado 3.3).

```
Data_1 <- dplyr::mutate(Data_sin_INF, year = lubridate::year(time),
                        month = lubridate::month(time), day = lubridate::day(time)
) %>%
  dplyr::group_by(year, month, day) %>%
  dplyr::summarize(temp_max = max(temp, na.rm = TRUE),
    temp_avg = mean(temp, na.rm = TRUE),
    temp_min = min(temp, na.rm = TRUE),
    prec = sum(aa1_2, na.rm = TRUE),
    wd = mean(wd, na.rm = TRUE),
    ws = max(ws, na.rm = TRUE),
    atmos_pres_avg = mean(atmos_pres, na.rm = TRUE),
    atmos_pres_min = min(atmos_pres, na.rm = TRUE),
    atmos_pres_max = max(atmos_pres, na.rm = TRUE),
```

```
    rh = mean(rh, na.rm = TRUE),
    ceil_hgt = mean(ceil_hgt, na.rm = TRUE),
    visibility = mean(visibility, na.rm = TRUE)
  )
```

Listado 3.3: Copia de los datos en un nuevo conjunto. Este conjunto posee la fecha dividida en componentes para poder ser posteriormente agrupada. A partir de ahí se realiza la unificación de datos atendiendo a los valores de la fecha empleando, para cada predictor, que su valor sea la media de la fecha objetivo. Además, se añaden nuevos campos basados en la media, el máximo y el mínimo de un campo padre.

Con los campos de año, mes y día sustituimos la fecha inicial para darle el formato que más nos convenga. En nuestro caso decidimos generarla con formato *YYYY-MM-DD*. (Listado 3.4).

```
Data_1$date <- as.Date(paste(Data_1$year, Data_1$month,
                             Data_1$day, sep = "-"))
Data_1 <- Data_1 %>% select(date, everything())
```

Listado 3.4: Generación de un campo date con el formato de fecha seleccionado. Se establece este nuevo predictor en la cabeza del conjunto de datos.

A continuación realizamos un último ajuste de valores que se encuentren fuera del rango normal. En este caso valores atípicos en la lluvia, donde establecemos los valores superiores a 650 a 0.0. (Listado 3.5).

```
Data_1$prec[Data_1$prec >= 650] <- 0.0
```

Listado 3.5: Sustitución de los valores de precipitación superiores a 650 por 0.0.

Para concluir con la limpieza y transformación de datos necesitamos sustituir los *Missing Values* de nuestros conjuntos. No obstante, puesto que hemos realizado múltiples acciones que requieren de operaciones matemáticas, es muy probable que se hallan generado de nuevo valores *INF* o *-INF*. Por consiguiente, repetimos el procedimiento para la eliminación de estos y realizamos a posteriori la sustitución de todos los valores *NA*. Para esta sustitución aplicamos la técnica aplicada en el resto de campos, aplicar la media de los valores del mismo.

3.3.3. Almacenamiento y análisis de datos preprocesados

Con la limpieza y transformación de datos previa hemos obtenido unos datos limpios y formateados para servir de entrada en los modelos. Estos deben ser guardados y analizados para detectar posibles problemas, como la alta correlación de variables o campos de poca relevancia, que podrían ocasionar exceso de cómputo en los modelos con información redundante o de poco valor.

En primer lugar realizamos el guardado de los modelos como un fichero *csv*, un tipo de documento para representar datos en forma de tabla en las que las columnas se separan por comas y las filas por saltos de línea. (Listado 3.6).

```
Modelo <- Data_1
write.csv2(Modelo, file = paste("PROCESSED_DATASETS/",
            Output_Name, "_Processed.csv", sep = ""),
            row.names = TRUE)
```

Listado 3.6: Guardado del conjunto de datos procesado en un fichero csv.

Con los datos almacenados podemos realizar un análisis de la correlación de variables con el objetivo de identificar, con los conocimientos que poseemos de cada característica, aquellas que estan altamente relacionadas y pueden ser eliminadas del conjunto.

Este proceso se simplifica empleando un gráfico como la figura 3.1, donde podemos apreciar mediante colores en la matriz si existe una relación directa entre las variables.

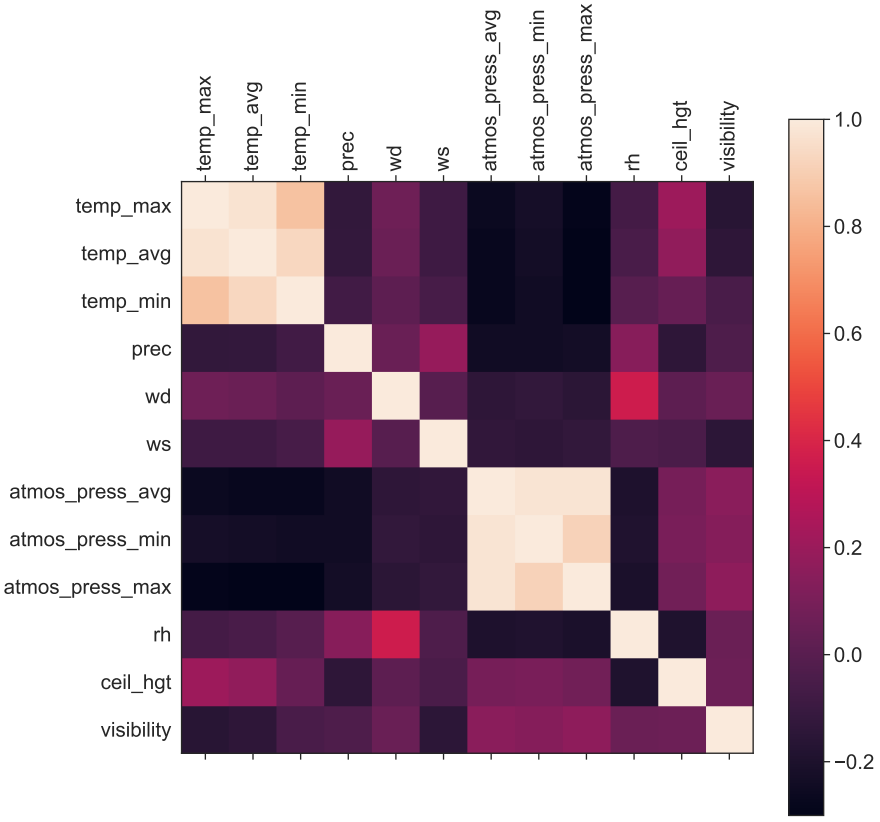


Figura 3.1: Matriz de colores para la correlación de variables en el conjunto de datos de La Gomera.

Vemos una correlación alta positiva entre los campos max, min y avg de las variables, lo cual es natural puesto que estamos referenciando a los mismos datos de maneras distintas. Por lo tanto, suprimimos los campos max y min para conservar únicamente la media.

Además, realizamos la eliminación de los campos *visibility* y *ceiling height*, puesto que ninguno va a suponer un impacto en las predicciones y simplificamos así el trabajo de los modelos a priori.

3.3.4. Etiquetado de datos

En esta sección procedemos al etiquetado de los datos. Para ello añadimos una nueva característica, FMA, de tipo booleana que representa como valor negativo un clima normal y como valor positivo un fenómeno meteorológico adverso.

Recordemos que Autoencoder es una técnica no supervisada, por lo que no se le facilitan datos etiquetados. Este campo no será proporcionado al modelo en su entrenamiento y servirá únicamente para la posterior validación de las predicciones.

Para realizar este etiquetado se recogieron todos los campos del conjunto de datos y otros adicionales, entres los que se incluye la propia etiqueta, que tras una serie de fórmulas producen la salida. Estos campos adicionales y sus fórmulas representadas como pseudocódigo se encuentran recogidos en la tabla 3.6.

ID	Campo	Descripción	Pseudocódigo
F1	T_MAX_Y	Temperatura Máxima Nivel Amarillo	if(temp_max >= 34) then 1 else 0
F2	T_MAX_O	Temperatura Máxima Nivel Naranja	if(temp_max >= 37) then 1 else 0
F3	T_MAX_R	Temperatura Máxima Nivel Rojo	if(temp_max >= 40) then 1 else 0
F4	T_MIN_Y	Temperatura Mínima Nivel Amarillo	if(temp_min <= -1) then 1 else 0
F5	T_MIN_O	Temperatura Mínima Nivel Naranja	if(temp_min <= -4) then 1 else 0
F6	T_MIN_R	Temperatura Mínima Nivel Rojo	if(temp_min <= -8) then 1 else 0
F7	Vientos_Y	Racha Máxima Nivel Amarillo	if(ws >= 19.44) then 1 else 0
F8	Vientos_O	Racha Máxima Nivel Naranja	if(ws >= 25) then 1 else 0
F9	Vientos_R	Racha Máxima Nivel Rojo	if(ws >= 36.1) then 1 else 0
F10	Prec_12_Y	Precipitación en 12 horas Nivel Amarillo	if(prec >= 60) then 1 else 0
F11	Prec_12_O	Precipitación en 12 horas Nivel Naranja	if(prec >= 100) then 1 else 0
F12	Prec_12_R	Precipitación en 12 horas Nivel Rojo	if(prec >= 180) then 1 else 0
F13	Prec_1_Y	Precipitación en 1 hora Nivel Amarillo	if(prec >= 15) then 1 else 0
F14	Prec_1_O	Precipitación en 1 hora Nivel Naranja	if(prec >= 30) then 1 else 0
F15	Prec_1_R	Precipitación en 1 hora Nivel Rojo	if(prec >= 60) then 1 else 0
F16	FMA	Etiqueta de clasificación de FMAs	SUM(F1:F15) == 0 then 0 else 1

Tabla 3.6: Campos y operaciones para el etiquetado de datos.

Los rangos empleados para determinar las salidas provienen de los umbrales establecidos para los diferentes niveles de alerta en el archipiélago canario por la *AEMET* [19].

Capítulo 4

Experimentos, modelos y resultados

Tras la búsqueda, elección y procesamiento de los diferentes conjuntos de datos necesarios para el desarrollo de los modelos, pasamos a la parte de experimentación y resultados. En esta sección se mostrarán los diferentes modelos, hiperparámetros y resultados obtenidos.

4.1. Hiperparámetros de los modelos

Existen una gran variedad de parámetros que son establecidos por el desarrollador como paso previo al entrenamiento. Estos son conocidos como hiperparámetros y hacen referencia a un conjunto de variables ajustables que permiten controlar el proceso de entrenamiento de un modelo. Nosotros, vamos a presentarlos para futuras aplicaciones de la siguiente manera. (Tabla 4.1).

Nombre	**Descripción**
Techniques	Tipo de capa RNN empleada.
Epochs	Número de épocas de entrenamiento. Se trata de un hiperparámetro de descenso de gradiente que controla el número de pases completos a través del conjunto de datos de entrenamiento.
Activation	Función de activación empleada. Esta es aplicada a la salida de un nodo para limitar o consolidar su valor.
Dropout Rate	Porcentage de neuronas abandonadas u olvidadas de forma aleatoria durante el entrenamiento de la red neuronal. Es una técnica de regularización para reducir el sobreajuste.
Learning Rate	Parámetro de ajuste en un algoritmo de optimización que determina el tamaño del paso en cada iteración mientras se mueve hacia un mínimo de una función de pérdida.
Optimizer	Algoritmo o método empleado para cambiar los atributos de la red neuronal, como los pesos y la tasa de aprendizaje, con el fin de reducir las pérdidas.
Hidden Units	Número de capas ocultas y los nodos que se encuentran en cada una de ellas.
Batch	Número de muestras en las que se dividen las épocas. Se trata de un hiperparámetro de descenso de gradiente que controla la cantidad de muestras de entrenamiento para trabajar antes de que se actualicen los parámetros internos del modelo.
Loss	Función de pérdida a minimizar.

Tabla 4.1: Hiperparámetros. Nomenclatura establecida y descripción.

4.2. Línea base

Dado que nuestro desarrollo pretende demostrar que los autoencoders son una técnica acertada para la predicción de anomalías climáticas, proponemos un modelo base de baja complejidad. Sobre este se realizarán entrenamientos y validaciones siguiendo la metodología y métricas propuestas.

El modelo, tanto para GRU como para LSTM, está basado en cuatro capas intermedias con nodos de 32 - 4 - 4 - 32 respectivamente y empleando como función de activación *ReLU* (Rectified Linear Unit) [2]. Esta es una de las más utilizadas para las capas ocultas y se define matemáticamente como $y = max(0, x)$, por lo que cuando se calcula la salida de un nodo escoge el valor máximo entre 0 y el valor obtenido.

```
def autoencoder_model_GRU(X):
    inputs = Input(shape = (X.shape[1], X.shape[2]))
    L1 = GRU(32, activation = 'relu', return_sequences = True)(inputs)
    L2 = GRU(4, activation = 'relu', return_sequences = False)(L1)
    L3 = RepeatVector(X.shape[1])(L2)
    L4 = GRU(4, activation = 'relu', return_sequences = True)(L3)
    L5 = GRU(32, activation = 'relu', return_sequences = True)(L4)
    output = TimeDistributed(Dense(X.shape[2]))(L5)
    model = Model(inputs = inputs, outputs = output)
    return model
```

Listado 4.1: Modelo utilizado como línea base empleando Keras y Tensorflow.

4.3. Métricas empleadas

En este apartado hablaremos sobre las métricas empleadas que nos permiten evaluar la calidad de los modelos.

La evaluación de un algoritmo de Deep Learning es una parte esencial de cualquier proyecto de este ámbito. Un modelo puede devolver resultados extraordinarios cuando se evalúa con una métrica, por ejemplo *Precision* y *Recall*, pero dar resultados deficientes cuando se emplea en comparación con otras como *Logarithmic Loss* o similares. Por ello, la elección de unas métricas apropiadas es fundamental ya que condiciona la eficacia de los modelos en la obtención de resultados.

Dado que intentamos determinar cuando un clima es normal o adverso en las Islas Canarias, nos encontramos ante un problema de clasificación binaria. Por lo que podemos establecer métricas dentro de un amplio abanico. Tras evaluar todas las posibilidades nos hemos decantado por las que se muestran a continuación.

- **Confusion Matrix:** la matriz de confusión describe el rendimiento para modelos de clasificación, siendo una herramienta que permite la visualización del desempeño de un algoritmo de aprendizaje profundo. Para un problema de clasificación binaria como el nuestro, se trataría de una matriz 2x2 donde las columnas hacen referencia a los valores predichos por el modelo y las filas a los resultados reales. Para su interpretación, los resultados son dados como “Positivos” o “Negativos”, aunque en nuestro caso hacemos referencia a climas normales o fenómenos meteorológicos adversos. (Fig. 4.1). Parte de su importancia reside en que, a partir de ella, se extraen nuevas métricas como las que comentaremos posteriormente en este tópico.

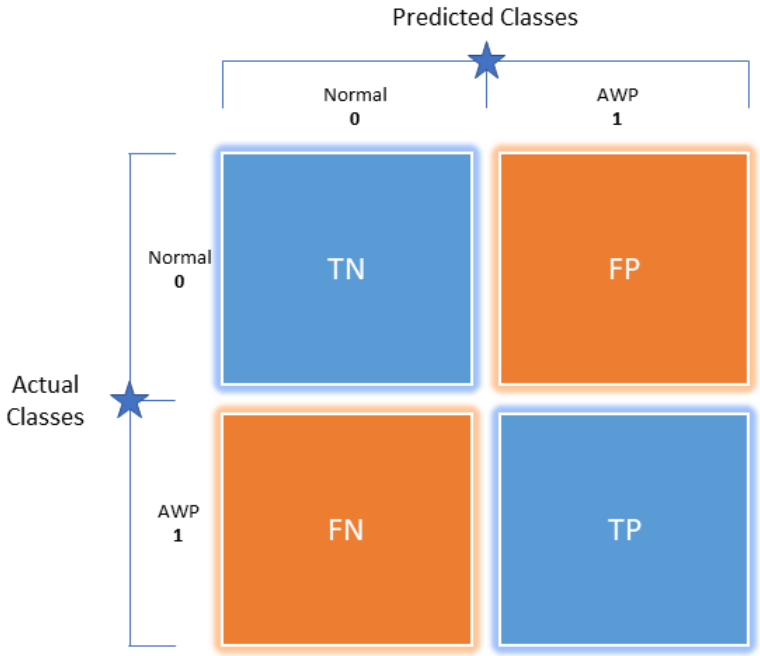


Figura 4.1: Matriz de confusión 2x2 para la clasificación de climas normales o AWPs.

Esta métrica posee los siguientes términos asociados:

- **TN (True Negative):** el valor predicho y el valor actual son negativos. En nuestro caso, el número de veces en las que el clima era normal y nuestro modelo predijo que, efectivamente, era normal.
- **FP (False Positive):** el valor predicho es positivo y el valor actual negativo. En nuestro contexto, el número de veces en las que el clima era normal y nuestro modelo predijo que se estaba produciendo un FMA.
- **FN (False Negative):** el valor predicho es negativo y el valor actual positivo. En nuestro caso, el número de veces en las que se estaba produciendo un FMA y nuestro modelo predijo que se trataba de un clima normal.
- **TP (True Positive):** el valor predicho y el valor actual son positivos. En nuestro contexto, el número de veces en las que se estaba produciendo un FMA y nuestro modelo predijo que, efectivamente, era un FMA.

- **Precision:** esta métrica mide la precisión del modelo al clasificar una muestra como positiva. Para nosotros, es la probabilidad de que una entrada sea etiquetada como normal de entre todas las entradas (normales y anómalas). Es decir, cuantas son correctas de todas las predicciones normales. Se calcula como la relación entre el número de muestras positivas clasificadas correctamente y el número total de muestras clasificadas como positivas (correctas

o incorrectas).

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

Por lo tanto, si el modelo realiza muchas clasificaciones positivas incorrectas o pocas clasificaciones positivas correctas, tendremos una precisión baja. En caso contrario, será alta.

- **True Positive Rate o Recall:** mide la capacidad del modelo para detectar muestras positivas, por lo que cuanto mayor sea, más muestras positivas son detectadas. En nuestro caso, hace referencia a la probabilidad de que una entrada sea etiquetada como normal de todas las entradas clasificadas. Esto quiere decir, cuantas de todas las entradas normales del conjunto de datos fueron correctamente predichas. Se calcula como la relación entre el número de muestras positivas correctamente clasificadas y el número total de muestras positivas.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

Esta métrica solo se preocupa por como se clasifican las muestras positivas, siendo independiente de como lo hacen las negativas.

Ambas medidas, *Precision* y *Recall*, se representan mediante valores comprendidos entre cero y uno. Además, la obtención de resultados altos en una suele suponer la obtención de resultados bajos en la otra. Por ello, es bastante útil emplear una métrica que combina ambas denominada *F-Score*.

- **F-Score:** realiza la combinación de la *Precision* y el *Recall*. Normalmente cuanto mayor sea su valor mejor se comporta el modelo. Su cálculo parte de una fórmula base donde el valor de β define la importancia que se le da a una métrica, *Precision* o *Recall*, respecto a la otra.

$$F_{beta} = (1 + \beta^2)\frac{precision * recall}{\beta^2 * precision + recall} \tag{4.3}$$

Los valores comprendidos en el rango $0 \leq \beta < 1$ proporcionan mayor peso a la *Precision* frente al *Recall* y aquellos > 1 otorgan la prioridad al *Recall*. En nuestro caso emplearemos la métrica *F1-score*, que establece $\beta = 1$, puesto que queremos generar un balance entre ellas.

- **False Positive Rate (FPR):** se refiere a la frecuencia con la que el modelo se equivoca cuando los valores reales son negativos. Se calcula como la relación entre el número de falsos positivos y el número total de eventos negativos reales.

$$FPR = \frac{FP}{FP + TN} \tag{4.4}$$

- **Receiver Operating Characteristic Curve (ROC):** se trata de una gráfica que relaciona las métricas *True Positive Rate* y *False Positive Rate* frente a múltiples thresholds o umbrales. Bajar el umbral de clasificación cataloga más

elementos como positivos, aumentando así tanto los falsos positivos como los verdaderos positivos.

- **Area Under Curve (AUC):** es el área comprendida bajo la *ROC Curve*. Se trata de un valor numérico entre 0 y 1 que puede ser interpretado como la probabilidad de que el modelo clasifique un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio. En ocasiones se describe también como AUROC.

Para obtener estas métricas emplearemos la librería de software libre denominada *Scikit-learn* [25]. Como ya comentamos anteriormente, se trata de una biblioteca para aprendizaje automático desarrollada en Python que incluye múltiples algoritmos de clasificación y regresión entre otros.

4.4. Experimentos

En esta sección mostraremos el entorno empleado, tanto software como hardware, los modelos propuestos y los resultados obtenidos tanto de forma numérica como gráfica. En definitiva, este apartado representa el seguimiento de la metodología previamente comentada, es decir, las decisiones tomadas y ejecutadas.

4.4.1. Código de las soluciones propuestas

Puesto que se trata de un proyecto con múltiples algoritmos, los cuales no pueden ser recogidos en su totalidad ni ejecutados en el presente documento, se decidió agruparlos en un repositorio Github [21]. Su uso permite mantener un histórico del desarrollo del proyecto en cada una de las fases. El repositorio recibe el nombre de *TFG_AWP_DETECTION_VIA_AUTOENCODER*, al que se puede acceder mediante el siguiente enlace [32].

4.4.2. Entorno

Todos los modelos fueron desarrollados, entrenados y evaluados sobre un ordenador personal con las siguientes características. (Tabla 4.2).

CPU	GPU	RAM	SO	Almacenamiento
Intel Core i5-7600K 3.8GHz	Asus GeForce GTX 1050 Ti 4GB GDDR5	12 GB 3000Mhz CL15	Windows 10 Education	Kingston A400 SSD 120GB

Tabla 4.2: Ordenador personal empleado.

Para la parte software empleamos las siguientes versiones. (Tabla 4.3).

Nombre	Tipo	Versión
Jupyter Lab	IDE	6.2.0
Python	Lenguaje de Programación	3.7.9
Keras	Librería	2.4.3
Tensorflow	Librería	2.4.1
NumPy	Librería	1.19.5
Pandas	Librería	1.2.2
Scikit-learn	Librería	0.24.1

Tabla 4.3: Software empleado.

4.4.3. Modelos propuestos

Tras aplicar la metodología indicada en la figura 2.5 en las diferentes islas del archipiélago canario obtuvimos los modelos finales. Estos modelos representan el fruto de todo el estudio y desarrollo del proyecto, los cuales nos proporcionan una serie de resultados que serán mostrados y analizados posteriormente.

Dadas las diversas arquitecturas disponibles sobre RNN no queriamos proporcionar una única solución. Por lo tanto, nos hemos decantado por realizar un desarrollo sobre LSTM (Long short-term memory) y GRU (Gated recurrent unit). De esta manera abarcamos diferentes soluciones y podemos comparar, de forma directa, los costes de computación y los diversos ajustes necesarios para proporcionar soluciones aproximadas.

Siguiendo la definición de los diversos campos reflejada en el tópico Hiperparámetros de los modelos, podemos representar los modelos obtenidos mediante las tablas 4.4 y 4.5.

Datasets	Model	Epochs	Activation	Dropout rate	Learning rate	Optimizer	Hidden Units	Batch	Loss
Lanzarote	M_{GRU1}	204	relu	L1 = 1e-5, L2 = 1e-4	0.001	adamax	[16-4-4-16]	128	mae
Fuerteventura	M_{GRU2}	152	relu	L2 = 0.01, 0.2	0.001	adamax	[16-4-4-16]	128	mae
Gran Canaria	M_{GRU3}	148	relu		0.001	adamax	[16-4-4-16]	32	mae
La Gomera	M_{GRU4}	300	relu		0.001	adamax	[32-4-4-32]	32	mae
El Hierro	M_{GRU5}	281	relu	0.2	0.001	adamax	[32-4-4-32]	48	mae
La Palma	M_{GRU6}	173	relu		0.001	adamax	[32-4-4-32]	32	mae
Tenerife	M_{GRU7}	161	relu	L2 = 0.01, 0.2	0.001	adamax	[64-32-4-4-32-64]	128	mae

Tabla 4.4: Modelos GRU obtenidos para la detección de anomalías en las Islas Canarias.

Datasets	Model	Epochs	Activation	Dropout rate	Learning rate	Optimizer	Hidden units	Batch	Loss
Lanzarote	M_{LSTM1}	300	relu	L1 = 1e-5, L2 = 1e-4, 0.2	0.001	adamax	[16-4-4-16]	128	mae
Fuerteventura	M_{LSTM2}	55	relu	L2 = 0.01, 0.2	0.001	adamax	[16-4-4-16]	128	mae
Gran Canaria	M_{LSTM3}	183	relu	L2 = 0.01, 0.2	0.001	adamax	[16-4-4-16]	32	mae
La Gomera	M_{LSTM4}	300	relu		0.001	adamax	[32-4-4-32]	32	mae
El Hierro	M_{LSTM5}	281	relu	0.2	0.001	adamax	[32-4-4-32]	48	mae
La Palma	M_{LSTM6}	288	relu	L2 = 0.01	0.001	adamax	[32-4-4-32]	32	mae
Tenerife	M_{LSTM7}	264	relu	L2 = 0.01, 0.2	0.001	adamax	[64-32-4-4-32-64]	128	mae

Tabla 4.5: Modelos LSTM obtenidos para la detección de anomalías en las Islas Canarias.

Como ejemplificación del proceso realizado para la construcción de modelos, exponemos los pasos realizados para la isla de *El Hierro*.

Comenzamos dividiendo el conjunto de datos en grupo de entrenamiento y grupo de evaluación, siguiendo una relación 80/20 respectivamente. Tras esta primera división, se procede a la obtención del grupo de validación. Para ello dividimos el conjunto de entrenamiento siguiendo la relación anterior, un 80 % para entrenamiento y un 20 % para validación. (Tabla 4.6).

Conjunto de datos	**Número de registros**
Original	16.735
Evaluación	3.347
Entrenamiento	10.544
Validación	2.636

Tabla 4.6: Registros por conjunto de datos de El Hierro.

A continuación realizamos la normalización de datos, de esta manera estos adquieren la forma adecuada para ser empleados en capas LSTM. Estas capas emplean un tensor de forma (Número de muestras) x (Pasos de tiempo) x (Número de características). (Listado 4.2).

```
x_train = x_train.reshape(x_train.shape[0], 1, x_train.shape[1])
x_test = x_test.reshape(x_test.shape[0], 1, x_test.shape[1])
x_validation = x_validation.reshape(x_validation.shape[0], 1,
                                    x_validation.shape[1])
```

Listado 4.2: Modelado de los datos de entrenamiento, prueba y validación para adaptarse al tensor de forma de las capas LSTM.

Con los datos normalizados podemos establecer el modelo base, entrenarlo y evaluar los resultados obtenidos con el objetivo mejorarlo modificando los hiperparámetros o el modelo en caso de resultados realmente deficientes.

Tras la realización de este procedimiento presentamos los siguientes modelos. (Listado 4.3, Listado 4.4).

```
def autoencoder_model_GRU(X):
    inputs = Input(shape = (X.shape[1], X.shape[2]))

    L1 = GRU(32, activation = 'relu', return_sequences = True)(inputs)
    L2 = Dropout(0.2)(L1)
    L3 = GRU(4, activation = 'relu', return_sequences = False)(L2)
    L4 = RepeatVector(X.shape[1])(L3)
    L5 = GRU(4, activation = 'relu', return_sequences = True)(L4)
    L6 = GRU(32, activation = 'relu', return_sequences = True)(L5)

    outputs = TimeDistributed(Dense(X.shape[2]))(L6)

    model = Model(inputs = inputs, outputs = outputs)
    return model
```

Listado 4.3: Modelo de El Hierro con capas GRU.

```
def autoencoder_model_LSTM(X):
    inputs = Input(shape = (X.shape[1], X.shape[2]))

    L1 = LSTM(32, activation = 'relu', return_sequences = True)(inputs)
    L2 = Dropout(0.2)(L1)
    L3 = LSTM(4, activation = 'relu', return_sequences = False)(L2)
    L4 = RepeatVector(X.shape[1])(L3)
    L5 = LSTM(4, activation = 'relu', return_sequences = True)(L4)
    L6 = LSTM(32, activation = 'relu', return_sequences = True)(L5)

    outputs = TimeDistributed(Dense(X.shape[2]))(L6)

    model = Model(inputs = inputs, outputs = outputs)
    return model
```

Listado 4.4: Modelo de El Hierro con capas LSTM.

Estos emplean la configuración de compilación y sistema de entrenamiento reflejada en el listado 4.5, donde se establece un método para prevenir el overfitting o sobreajuste. Este función recibe el nombre de *early_stopping_callback* y parte del método *EarlyStopping* de Keras [23]. Esta callback permite detener el entrenamiento del modelo cuando un parámetro no se modifica en un número determinado de epochs. En este caso valoramos un total de 50 epochs monitorizando la pérdida de validación.

```
model.compile(optimizer = 'adamax', loss = 'mae',
            metrics = ['mae', 'mse', 'mape', 'msle', 'cosine_proximity'])

dataframe_train = DataFrame()
dataframe_validation = DataFrame()

early_stopping_callback = EarlyStopping(monitor='val_loss',
                            mode='min', verbose=1, patience=50)

for i in range(1):
    print('Fit model on training data...')
    start = time.time()
    epochs = 300
    batch_size = 48
    history = model.fit(x_train, x_train, epochs = epochs,
                        batch_size = batch_size,
                        validation_data = (x_validation, x_validation),
                        callbacks=[early_stopping_callback],
                        verbose = 0).history
    end = time.time()

    dataframe_train[str(i)] = history['loss']
    dataframe_validation[str(i)] = history['val_loss']

    print('Time to training model:', end - start)
```

Listado 4.5: Compilación y entrenamiento para modelo GRU y LSTM de El Hierro.

4.5. Resultados

En esta sección mostraremos los resultados de los modelos en las islas. Nos apoyaremos en una serie de tablas y representaciones gráficas para su análisis.

Tras el entrenamiento de los modelos propuestos obtenemos una serie de gráficos que nos permiten evaluar el rendimiento y detectar tendencias. Además, nos permiten establecer el valor de corte para la clasificación binaria. Puesto que se trata de un amplio abanico de gráficos, hemos tomado la decisión de presentar una selección de estos empleando gráficos unificados y simples.

El conjunto de gráficas pertenecientes a este tópico, incluidas las que se mostrarán a continuación, se encuentran recogidas en el Apéndice A y el Apéndice B.

- **Model Loss:** representa la pérdida, tanto en entrenamiento como en validación, en función del número de épocas procesadas. En la figura 4.2 podemos apreciar como todos los modelos con capas GRU demuestran un ajuste correcto. Esto se debe a que las curvas de entrenamiento y validación se encuentran en los mismos rangos para el error de reconstrucción según avanzan las épocas. Podemos verlo en mejor detalle en la figura 4.3, donde se refleja el resultado para el modelo de El Hierro con capas LSTM.

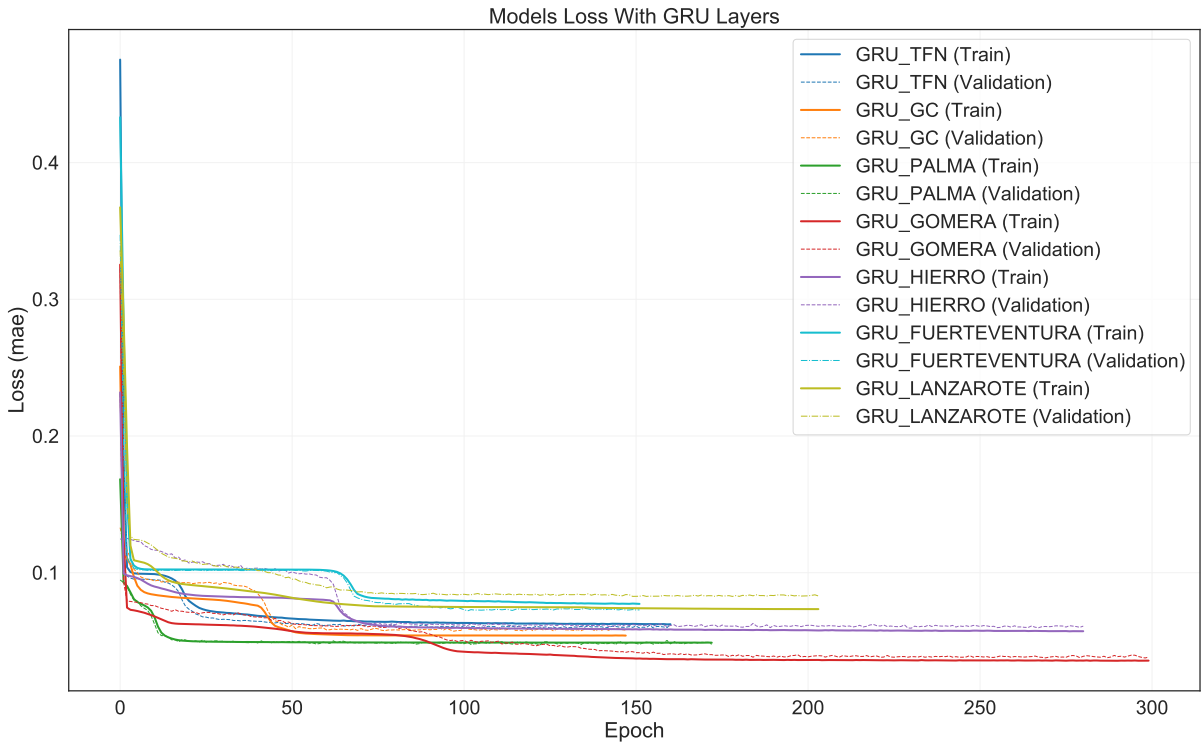


Figura 4.2: Gráfica de pérdida de los modelos con capas GRU.

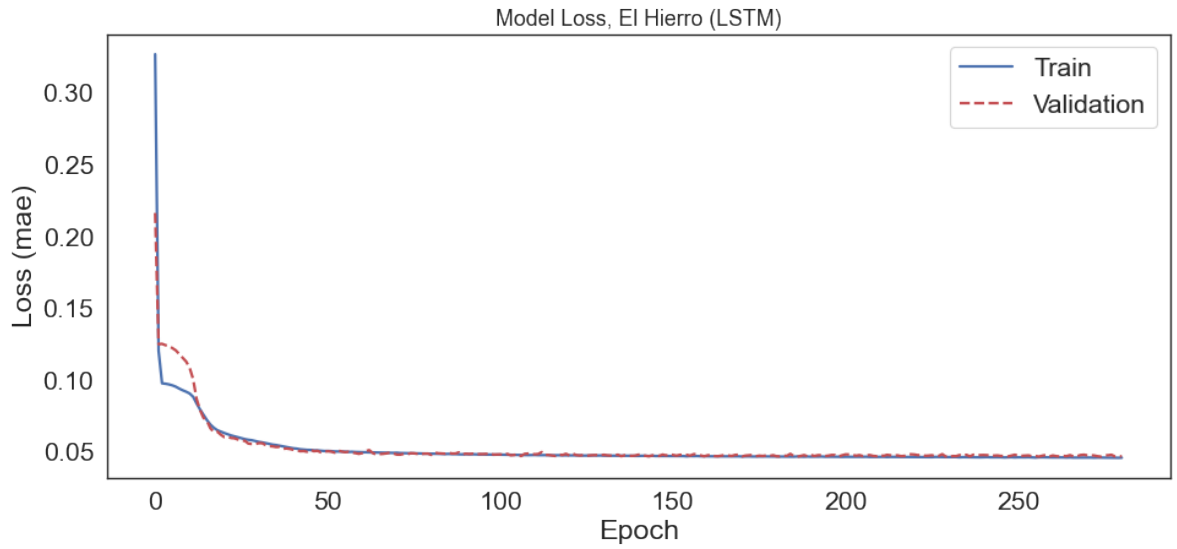


Figura 4.3: Gráfica de pérdida de El Hierro con capas LSTM.

- **Precision And Recall For Different Threshold Values:** representa la *Precision* y el *Recall* (eje Y) en función de posibles valores para el threshold (eje X). Es empleada para la selección del umbral del error de reconstrucción en cada uno de los modelos. En la figura 4.4 podemos ver la agrupación para todos los modelos con capas LSTM. En la figura 4.5 apreciamos el gráfico para La Palma con capas GRU donde se aplicó como umbral el punto de intersección entre la Precision y el Recall.

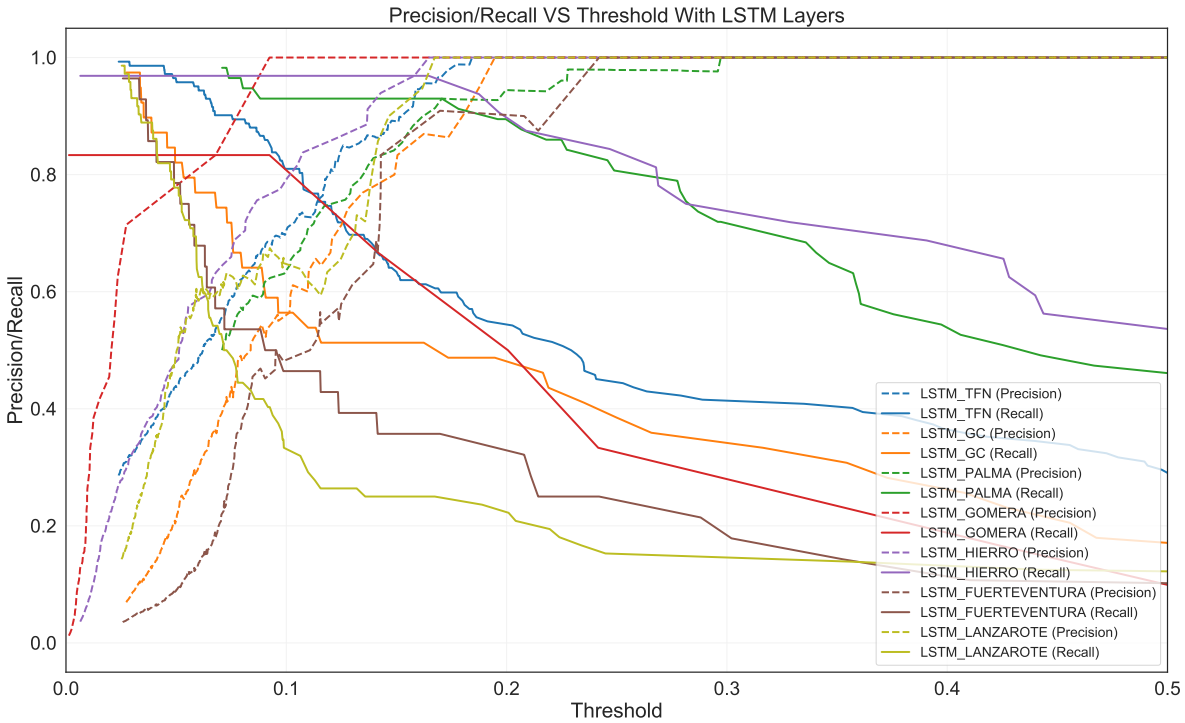


Figura 4.4: Gráfica de Precision/Recall frente al threshold para los modelos con capas LSTM.

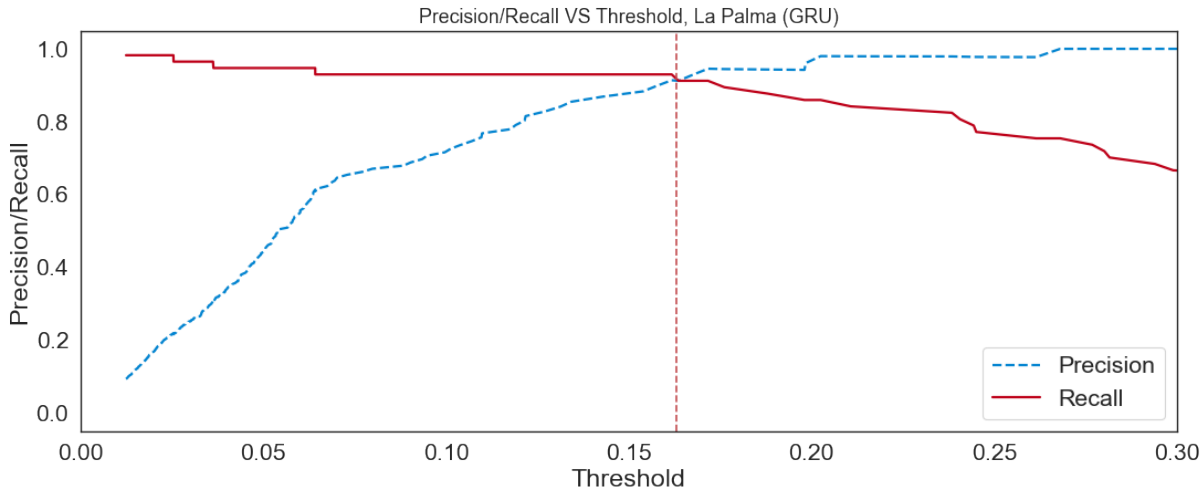


Figura 4.5: Gráfica de Precision/Recall frente al threshold de La Palma con capas GRU.

- **ROC Curve:** permite evaluar el *True Positive Rate* y el *False Positive Rate* frente a múltiples thresholds. Además, nos proporciona información sobre el *AUC*, el área bajo esta curva. En ella se muestra la eficacia para la clasificación binaria ante la que nos encontramos en nuestra problemática. En la figura 4.6 reflejamos la agrupación para todos los modelos con capas GRU. En la figura 4.7 encontramos el gráfico para el modelo de Tenerife con capas LSTM. Este demuestra una eficacia elevada. No obstante, al tratarse de clases desbalanceadas, como explicaremos más adelante, llega a ser una medida relativa.

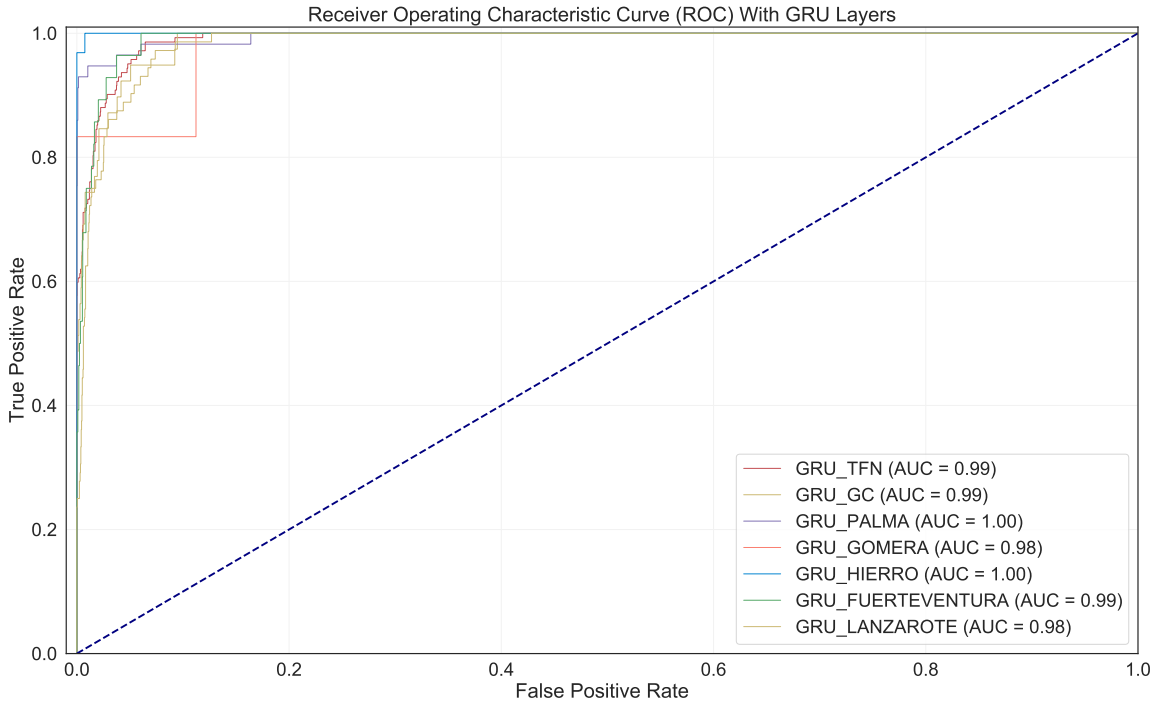


Figura 4.6: ROC Curve para los modelos con capas GRU.

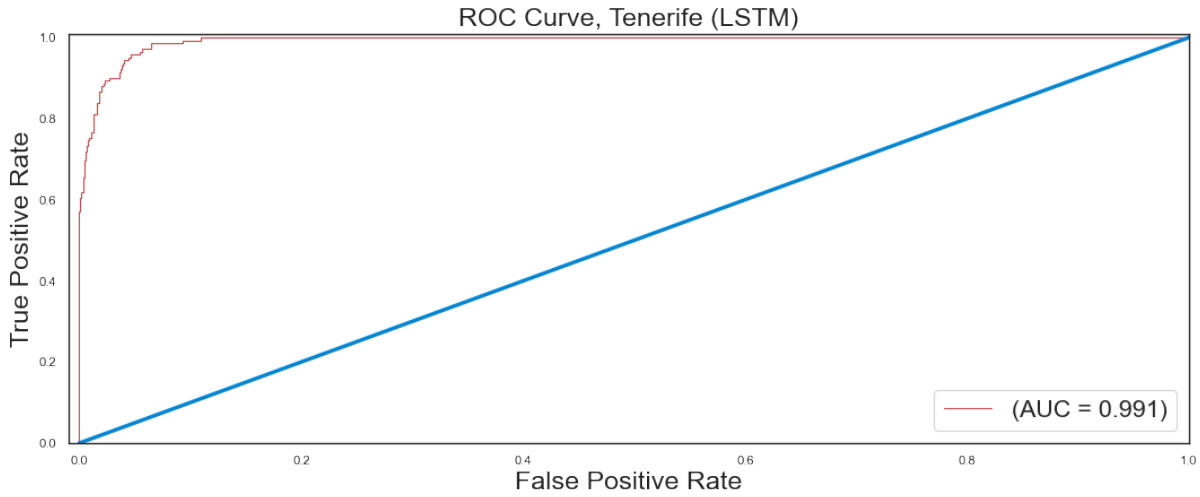


Figura 4.7: ROC Curve de Tenerife con capas LSTM.

- **Recall VS Precision:** nos permite representar la tendencia de la *Precision* y el *Recall* según estos crecen. En la figura 4.8 encontramos la agrupación para los modelos con capas LSTM. En la figura 4.9 reflejamos el resultado para el modelo de La Palma con capas GRU. Muestra como la Precision se mantiene a 1 hasta que se alcanza aproximadamente 0.75 de Recall, donde se produce un descenso de forma escalonada. Demuestra que obtiene una alta tasa de Precision y Recall de forma simultánea.

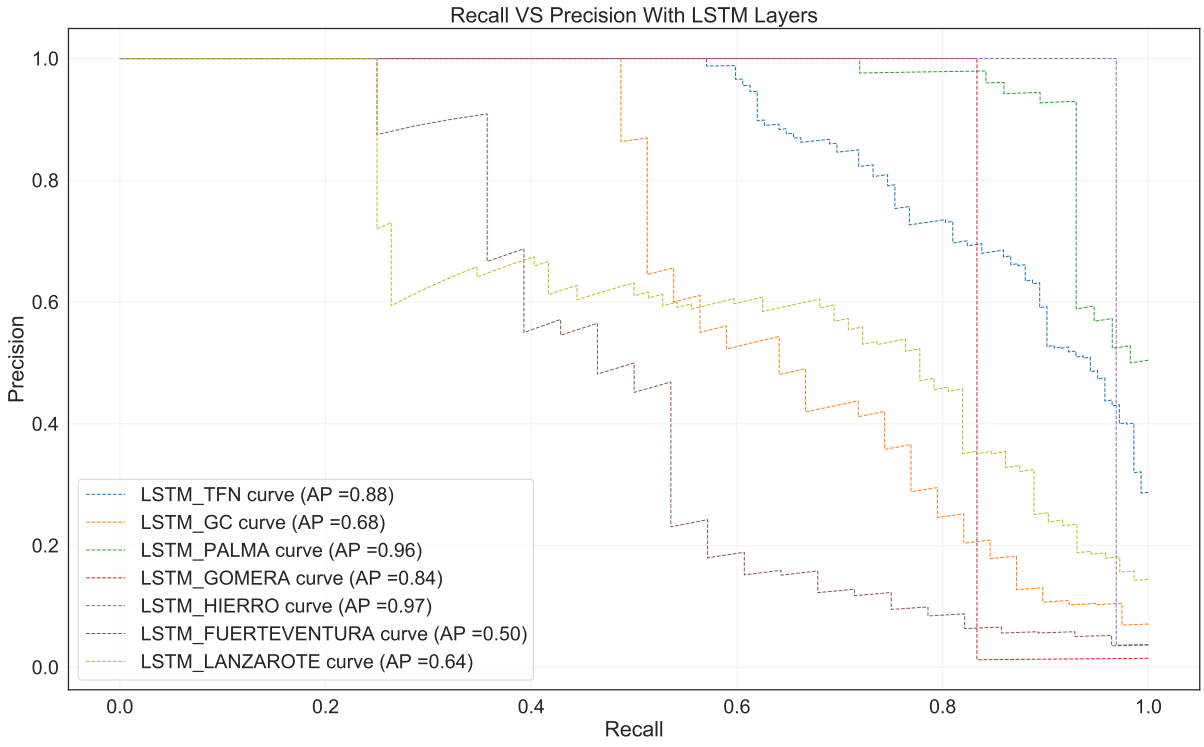


Figura 4.8: Gráfica Recall VS Precision para los modelos con capas LSTM.

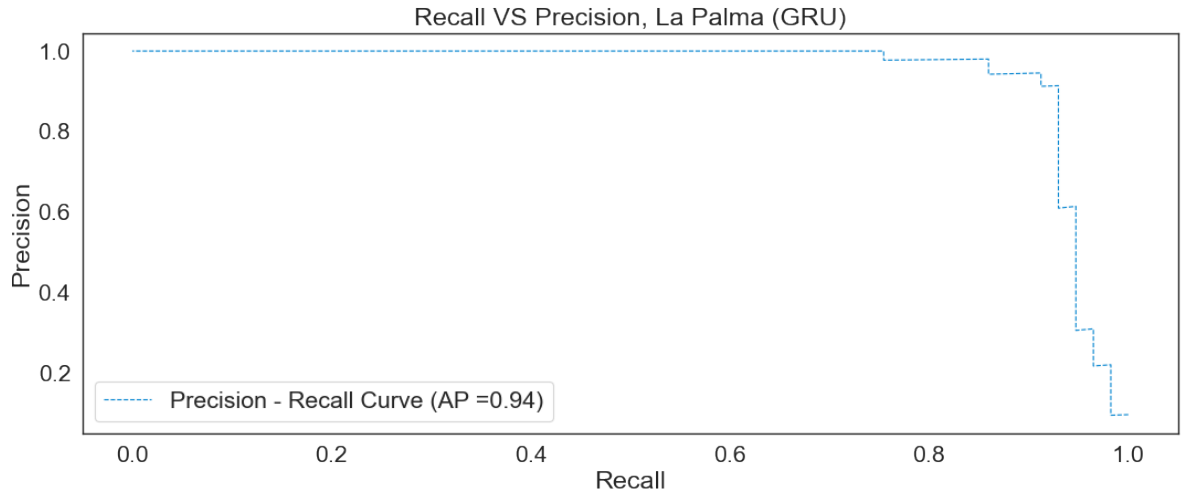


Figura 4.9: Gráfica Recall VS Precision de La Palma con capas GRU.

4.5.1. Thresholds

Para la labor de clasificación de nuestros modelos debemos determinar los thresholds o umbrales del error de reconstrucción. Estos umbrales establecen que las entradas con error de reconstrucción inferior serán clasificadas como normales y las que posean uno superior como FMAs.

Los thresholds escogidos para el estudio son el valor máximo (Max), la intersección entre la *Precision* y el *Recall* ($Precision \equiv Recall$) y el sumatorio de la desviación típica y la media ($\sigma + \overline{x}$). (Tabla 4.7).

Islas	**Modelos**	**Threshold**		
		$\boldsymbol{Max}$	$\boldsymbol{\sigma + \overline{x}}$	$\boldsymbol{Precision \equiv Recall}$
Lanzarote	$LSTM_1$	1.000072	0.155364	0.062439
	GRU_1	0.999754	0.160354	0.062504
Fuerteventura	$LSTM_2$	1.000607	0.208837	0.095293
	GRU_2	1.000227	0.155609	0.072572
Gran Canaria	$LSTM_3$	1.000244	0.169497	0.101431
	GRU_3	0.999811	0.137928	0.088365
La Gomera	$LSTM_4$	0.998243	0.073697	0.067892
	GRU_4	0.998229	0.099252	0.072965
El Hierro	$LSTM_5$	1.000211	0.106936	0.158080
	GRU_5	1.000195	0.125746	0.164424
La Palma	$LSTM_6$	1.000329	0.170279	0.170590
	GRU_6	0.999669	0.130324	0.163635
Tenerife	$LSTM_7$	1.000158	0.160977	0.114679
	GRU_7	0.994148	0.114744	0.112484

Tabla 4.7: Thresholds obtenidos para cada modelo siguiendo el valor máximo (Max), la intersección entre la Precision y el Recall ($Precision \equiv Recall$) y el sumatorio de la desviación típica y la media ($\sigma + \overline{x}$).

4.5.2. Resultados de las métricas

Teniendo en cuenta los thresholds obtenidos en la tabla 4.7, podemos reflejar los resultados empleando tres métricas clave: Precision, Recall y F1-score. (Tabla 4.8).

Islas	**Modelos**	Max			$\sigma + \bar{x}$			$Precision \equiv Recall$
		Precision	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Precision & Recall & F1-score**
Lanzarote	$LSTM_1$	0.99	0.54	0.57	0.94	0.62	0.69	0.79
	GRU_1	0.99	0.54	0.57	0.96	0.62	0.68	0.81
Fuerteventura	$LSTM_2$	1.00	0.55	0.59	0.95	0.66	0.74	0.75
	GRU_2	1.00	0.55	0.59	0.95	0.68	0.75	0.77
Gran Canaria	$LSTM_3$	0.99	0.55	0.59	0.93	0.76	0.82	0.78
	GRU_3	0.99	0.55	0.59	0.90	0.76	0.81	0.81
La Gomera	$LSTM_4$	0.50	0.50	0.50	0.92	0.92	0.92	0.92
	GRU_4	0.50	0.50	0.50	1.00	0.92	0.95	0.92
El Hierro	$LSTM_5$	1.00	0.70	0.79	0.91	0.98	0.94	0.98
	GRU_5	1.00	0.70	0.79	0.92	0.98	0.95	0.98
La Palma	$LSTM_6$	0.99	0.63	0.71	0.96	0.96	0.96	0.96
	GRU_6	0.99	0.63	0.71	0.92	0.96	0.94	0.96
Tenerife	$LSTM_7$	0.98	0.58	0.63	0.96	0.81	0.87	0.87
	GRU_7	0.98	0.58	0.64	0.90	0.80	0.84	0.86

Tabla 4.8: Métricas de la evaluación del rendimiento en la identificación de los AWP en los modelos GRU y LSTM propuestos.

Junto a estos datos es necesario representar y evaluar la matriz de confusión para cada umbral, puesto que es aquí donde podemos determinar la cantidad de aciertos o fallos que tenemos con las predicciones realizadas. (Tabla 4.9).

Islas	**Modelos**	**Matriz de Confusión**					
		Max		$\sigma + \bar{x}$		$Precision \equiv Recall$	
Lanzarote	$LSTM_1$	2923	0	2921	2	2894	29
		66	6	54	18	29	43
	GRU_1	2923	0	2922	1	2896	27
		66	6	55	17	27	45
Fuerteventura	$LSTM_2$	3258	0	3257	1	3244	14
		25	3	19	9	14	14
	GRU_2	3258	0	3257	1	3245	13
		25	3	18	10	13	15
Gran Canaria	$LSTM_3$	3316	0	3313	3	3299	17
		35	4	19	20	17	22
	GRU_3	3316	0	3311	5	3301	15
		35	4	19	20	15	24
La Gomera	$LSTM_4$	953	0	952	1	952	1
		6	0	1	5	1	5
	GRU_4	953	0	953	0	952	1
		6	0	1	5	1	5
El Hierro	$LSTM_5$	3315	0	3308	7	3314	1
		19	13	1	31	1	31
	GRU_5	3315	0	3309	6	3314	1
		19	13	1	31	1	31
La Palma	$LSTM_6$	3294	0	3289	5	3290	4
		42	15	4	53	4	53
	GRU_6	3294	0	3284	10	3289	5
		42	15	4	53	5	52
Tenerife	$LSTM_7$	3212	0	3207	5	3177	35
		119	23	55	87	35	107
	GRU_7	3212	0	3192	20	3174	38
		118	24	57	85	38	104

Tabla 4.9: Resultados de la matriz de confusión para los diferentes modelos aplicando los thresholds propuestos.

4.6. Análisis de resultados

En esta sección realizaremos un análisis profundo de los resultados obtenidos, indicando la tendencia de estos y su interpretación.

Hemos recogido un total de 7 conjuntos de datos que, tras su procesamiento, han sido empleados para el entrenamiento de los modelos previamente generados. Los modelos resultantes forman un total de 14, puesto que hemos generado dos modelos por isla aplicando capas GRU y LSTM.

El primer estudio a realizar sería una comparación directa entre las capas, donde nos centramos en sus necesidades de cómputo. (Tabla 4.10).

Islas	Modelos	Epochs	Batch Size	Tiempo (s)
Lanzarote	$LSTM_1$	300	128	680.87
	GRU_1	204	128	533.59
Fuerteventura	$LSTM_2$	55	128	130.45
	GRU_2	152	128	432.77
Gran Canaria	$LSTM_3$	183	32	1624.46
	GRU_3	148	32	1605.91
La Gomera	$LSTM_4$	300	32	325.95
	GRU_4	300	32	305.49
El Hierro	$LSTM_5$	281	48	1823.81
	GRU_5	281	48	2040.75
La Palma	$LSTM_6$	288	32	2703.58
	GRU_6	173	32	1874.88
Tenerife	$LSTM_7$	264	128	915.92
	GRU_6	161	128	603.05

Tabla 4.10: Requerimientos computacionales de los modelos.

Atendiendo al conjunto, y teniendo en cuenta que previamente establecimos una callback para evitar el sobreentrenamiento de los modelos, podemos observar que en la mayoría de casos demostramos que los modelos con capas GRU requieren de menor número de épocas y tiempo de computación. Incluso en los casos en los que el número de iteraciones coincide los tiempos son menores en este tipo de capa. (Fig. 4.10).

Esto nos da un indicador de que las capas GRU proporcionan resultados de forma más rápida y con menos entrenamiento. Podemos apreciarlo también en las tablas 4.4 y 4.5, donde comprobamos que los modelos LSTM requerían de mayor regularización. No obstante, en cuanto a la clasificación los resultados son muy similares, por lo que realmente la diferencia de capas en este caso solo proporciona variaciones de tiempo y cómputo.

Dado que previamente seleccionamos thresholds en función de tres medidas clave, mostramos las matrices de confusión para cada uno en la tabla 4.9. En ella confirmamos que los resultados no presentan grandes variaciones por el tipo de capa aplicada.

Apreciamos que determinados modelos presentan clasificaciones con una gran tasa de error con respecto al etiquetado FMA. Si estudiamos en detenimiento las gráficas *Model Loss*, que podemos encontrar en el Apéndice A.1, podemos determinar nuevos umbrales en los que los resultados sean más acertados tendiendo a

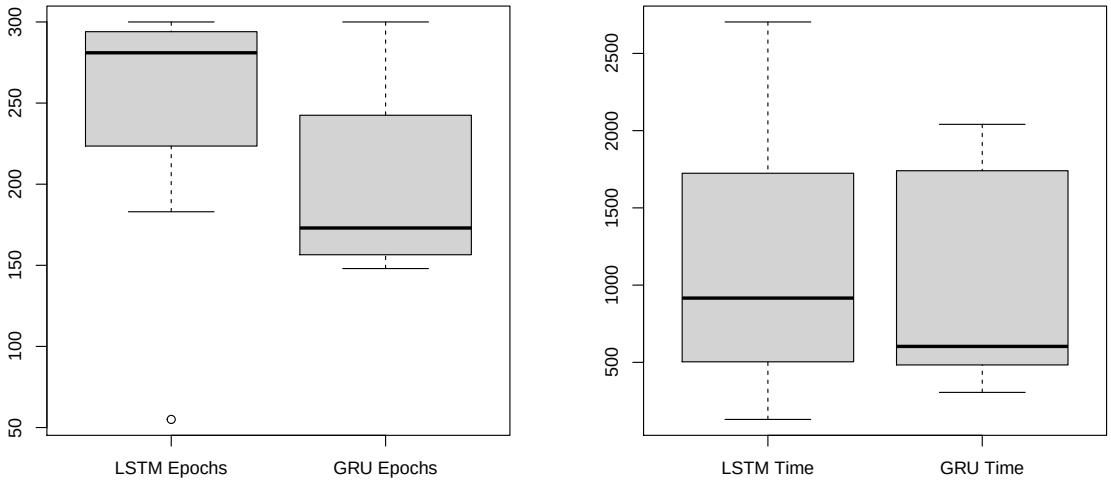


Figura 4.10: Diagramas de cajas para los tiempos y épocas de las capas GRU y LSTM en los modelos propuestos.

mejorar alguna de las clasificaciones. Puesto que estamos hablando de modelos que podrían servir para indicar a un colectivo cuando es seguro o no acudir al exterior, proponemos los siguientes umbrales donde tendemos a mejorar los aciertos en la clasificación de FMAs. (Tabla 4.11).

Islas	Modelos	Threshold	Matriz de Confusión		Precision	Recall	F1-Score	Clasificación Normal (%)	Clasificación FMA (%)
Lanzarote	$LSTM_1$	0.039	2789	134	0.66	0.92	0.72	95.42	88.89
			8	64					
	GRU_1	0.029	2648	275	0.60	0.95	0.65	90.60	98.61
			1	71					
Fuerteventura	$LSTM_2$	0.049	3019	239	0.54	0.87	0.56	92.66	82.14
			5	23					
	GRU_2	0.031	3163	95	0.61	0.95	0.67	97.08	92.86
			2	26					
Gran Canaria	$LSTM_3$	0.032	2954	362	0.55	0.93	0.56	89.08	97.44
			1	38					
	GRU_3	0.03	3136	180	0.58	0.95	0.63	94.57	94.87
			2	37					
La Gomera	$LSTM_4$	0.093	953	0	1.00	0.92	0.95	100	83.34
			1	5					
	GRU_4	0.099	953	0	1.00	0.92	0.95	100	83.34
			1	5					
El Hierro	$LSTM_5$	0.165	3315	0	1.00	0.98	0.99	100	96.88
			1	31					
	GRU_5	0.184	3315	0	1.00	0.98	0.99	100	96.88
			1	31					
La Palma	$LSTM_6$	0.171	3290	4	0.96	0.96	0.96	99.88	92.98
			4	53					
	GRU_6	0.163	3289	5	0.96	0.96	0.96	99.85	92.98
			4	53					
Tenerife	$LSTM_7$	0.044	3001	211	0.70	0.96	0.77	93.43	98.59
			2	140					
	GRU_7	0.042	2994	218	0.70	0.96	0.76	93.21	98.59
			2	140					

Tabla 4.11: Resultados para los diferentes modelos aplicando thresholds propuestos con el objetivo de mejorar los aciertos de valores FMA.

La elección de estos puntos de corte sigue basandose en tratar de encontrar un punto intermedio entre *Precision* y *Recall*, pero con la salvedad de que para mejorar los *True Negatives* debemos dar mayor peso al recall, buscando de esta manera adquirir mejores resultados en esta métrica.

Pasando al apartado gráfico podemos visualizar que los modelos poseen ciertas tendencias. En el caso general de la pérdida de los modelos, reflejados en el Apéndice B.1, apreciamos la tendencia de las curvas de entrenamiento y validación a discurrir de forma paralela y a aproximarse. Esto nos indica que el ajuste del modelo es correcto y que no se aprecia overfitting, puesto que la validación no tiende a alejarse del entrenamiento de forma ascendente.

Estos gráficos pueden tener múltiples formas y ser perfectamente válidos. Por ejemplo, tenemos casos en los que la validación se encuentra por encima del entrenamiento de forma cercana pero sin unirse, casos en los que se encuentra por debajo y casos en los que esta oscila por encima y por debajo (Fig. 4.3). Lo importante es que en ningún caso hemos obtenido una tendencia a intersectarse y alejarse, lo que nos indicaría que el modelo es ineficiente ya sea porque se encuentra sobreentrenado, porque no posee la configuración adecuada para el caso en cuestión o por problemas en los datos de entrenamiento.

Por otro lado, estudiando los gráficos de índice de puntos respecto al error de reconstrucción recogidos en el Apéndice A.5, apreciamos la tendencia de los puntos normales a mantenerse en la zona inferior y de los puntos anómalos a ascender. Empleando como ejemplo el caso de Fuerteventura con capas GRU, figura 4.11, observamos claramente esta concentración de valores normales y la tendencia de otros a superar el umbral establecido. Por tanto, este sería un ejemplo en el que tendemos a aportar mayor peso al *Recall*, sacrificando en cierta medida la *Precision*. Esta decisión provoca que se clasifiquen erroneamente algunos climas normales con el objetivo de acentuar la capacidad de obtener un mayor porcentaje de acierto en los FMA. No obstante, siempre se busca cierto balance, por lo que finalmente para este modelo en particular se obtiene, observando los datos recogidos en la tabla 4.11, un 92.86 % de acierto para la clasificación de FMAs y un 97.08 % para la clasificación de climas normales.

En esta misma tabla vemos representados los porcentajes de acierto para la clasificación de todos los modelos. Observamos que en conjunto todos poseen un rendimiento pico que oscila entre el 82.14 % y el 98.61 % para la clasificación de FMAs, y de un 89.08 % y un 100 % para la clasificación de climas normales. Existe claramente una tendencia en la que los climas normales son clasificados con mayor precisión, suceso que no es de extrañar ya que los modelos han sido entrenados con datos mayoritariamente normales. Esto provoca que los algoritmos conozcan mejor este tipo de valores y les resulte más fácil identificarlos.

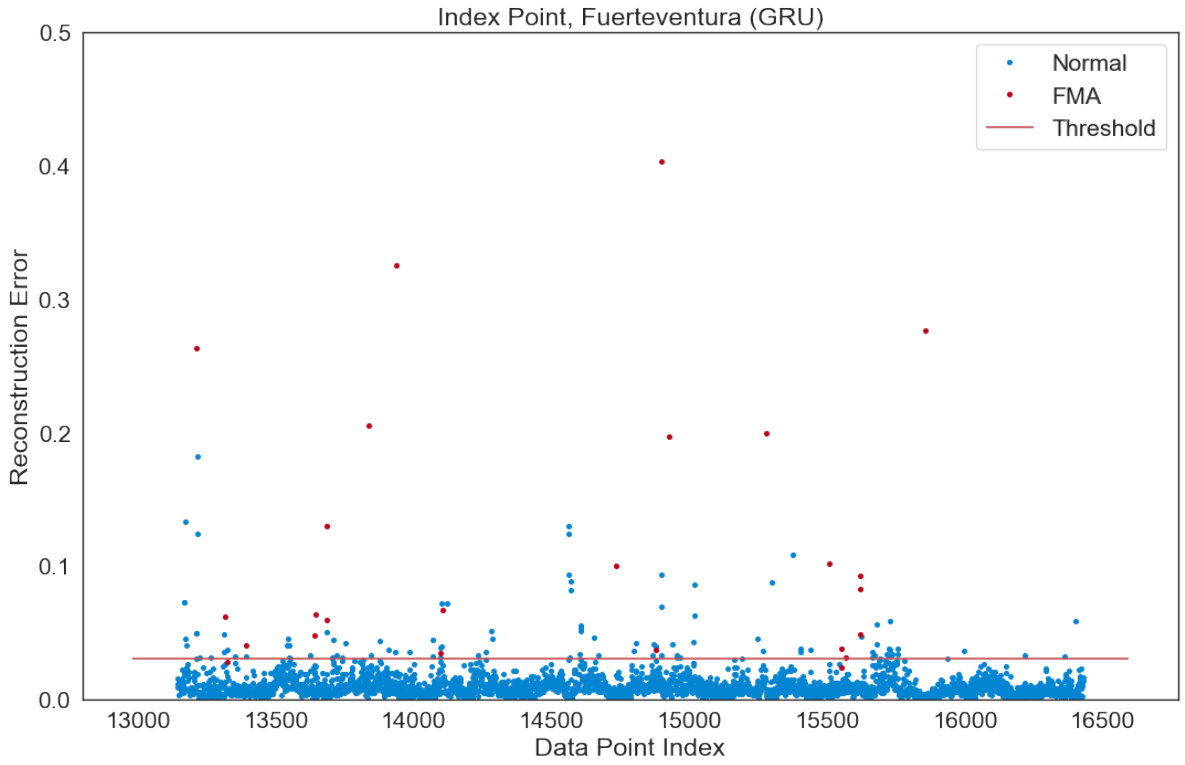


Figura 4.11: Gráfico de entradas de datos respecto al error de reconstrucción para el modelo GRU de Fuerteventura.

Existen otros casos, como es el de La Gomera, donde no se requiere de una evaluación para encontrar un punto de recall alto sacrificando parte de la precisión. Esto se debe a que, analizando el gráfico de *Precision* y *Recall* frente al threshold, las curvas de *Precision* y *Recall* se encuentran en su valor más alto en el mismo punto de corte. (Fig 4.12)

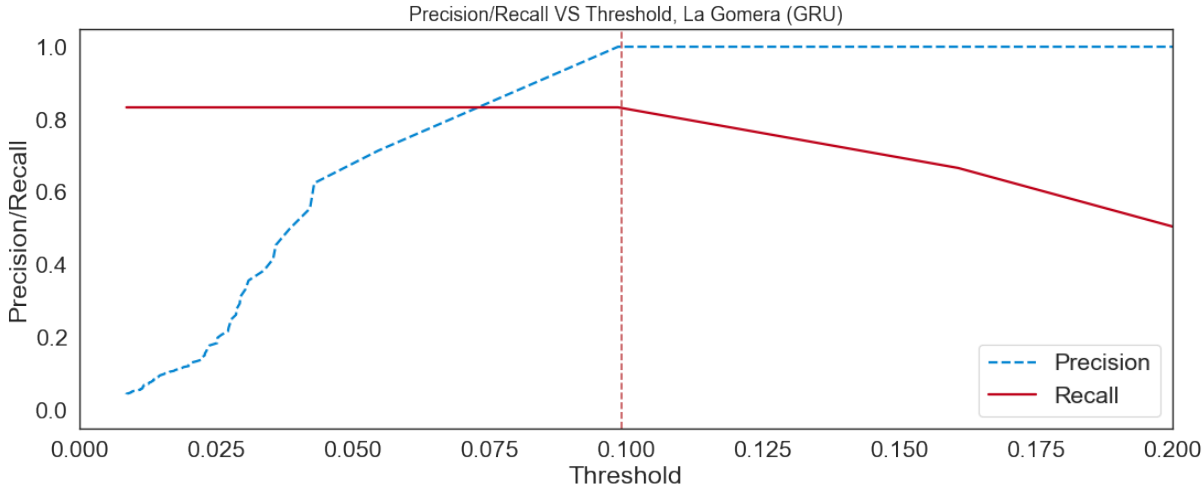


Figura 4.12: Gráfico de Precision/Recall frente al Threshold para el modelo GRU de La Gomera.

Con estos resultados podemos decir que los modelos son capaces de etiquetar, con una tasa de error variable, los datos de entrada de forma correcta detectando y

diferenciando lo que es un FMA de lo que es un clima normal. Poseen la habilidad de clasificar con una alta capacidad estas entradas y variar los thresholds para otorgar mayor peso a la métrica que nos interese priorizar. Esto permite que sean modelos flexibles y atractivos para su implementación en una amplia variedad de aplicaciones como detectores de peligro atmosférico, aplicaciones de alerta ciudadana, etc.

Capítulo 5

Conclusiones y líneas futuras

En este capítulo presentamos las conclusiones obtenidas a partir del desarrollo del proyecto basado en redes de aprendizaje profundo centrado en la técnica no supervisada denominada Autoencoder y las líneas futuras del trabajo.

5.1. Conclusiones

Al finalizar el proyecto hemos llegado a las siguientes conclusiones:

- Se desarrollaron múltiples modelos capaces de identificar aquellos fenómenos meteorológicos adversos dentro de diversos conjuntos de datos correspondientes a las Islas Canarias.
- Las métricas escogidas demostraron que los modelos propuestos proporcionan unos resultados de alta calidad.
- Se obtuvieron modelos flexibles y atractivos para su implementación en múltiples tipos de aplicaciones.
- Se comprobó que las capas *Long Short-Term Memory (LSTM)* y *Gated Recurrent Unit (GRU)* favorecen el desarrollo exitoso de modelos basados en la arquitectura *Autoencoder* para la detección de anomalías.
- Las capas GRU en comparación con las LSTM han sido más eficientes computacionalmente.
- Los autoencoders demostraron una alta capacidad para la detección de anomalías climáticas en las Islas Canarias.
- La utilización de métodos gráficos para la evaluación de resultados fue imprescindible para su correcta interpretación.
- Se obtuvo una alta tasa de *Recall* con una *Precision* elevada en cada uno de los modelos.

5.2. Líneas futuras

En esta sección se proponen algunos aspectos destacables útiles para continuar el desarrollo del proyecto:

- Atendiendo al beneficio que puede generar a nivel regional en las Islas Canarias la utilización de estos modelos, se plantea el futuro desarrollo de una aplicación que haga uso de estos para proporcionar una respuesta en función de los datos facilitados. Esta respuesta vendría dada mediante la clasificación empleada durante todo el proyecto, *Normal o AWP*. Además, su desarrollo podría servir de motivación para diversos estudios meteorológicos o implementación de nuevos modelos.
- Implementación de Keras Tuner [39], una librería que ayuda a seleccionar el conjunto óptimo de hiperparámetros para los programas en Tensorflow. El proceso de seleccionar el conjunto correcto de estos parámetros es conocido como *Hyperparameter Tuning o Hypertuning*.

Capítulo 6

Conclusions and future improvements

In this chapter, we present the conclusions of the development of this project based on Deep Learning Networks focused on the unsupervised technique called Autoencoder and future lines of work.

6.1. Conclusions

At the end of the project, we reached the following conclusions:

- The multiple developed models are able to identify those adverse meteorological phenomena within various data sets corresponding to the Canary Islands.
- The chosen metrics demonstrated that the proposed models provide high-quality results.
- There were obtained flexible and engaging models for their implementation in multiple types of applications.
- It does found that the *Long Short-Term Memory (LSTM)* and *Gated Recurrent Unit (GRU)* layers favour the successful development of models based on the *Autoencoder* architecture for anomaly detection.
- GRU layers compared to LSTM have been more computationally efficient.
- The autoencoders demonstrated a high capacity for the detection of climatic anomalies in the Canary Islands.
- Using graphic methods for the results' evaluation was essential for their correct interpretation.
- There was obtained a high *Recall* rate with high *Precision* in each of the models.

6.2. Future improvements

This section contains some helpful highlights to continue the development of the project:

- Considering the benefit that the use of these models can generate at the regional level in the Canary Islands, is proposed the future development of an application that makes use of them to provide a response based on the data provided. This answer would be given by the classification used throughout the project, *Normal or AWP*. In addition, its development could serve as motivation for various meteorological studies or the implementation of new models.
- Keras Tuner's Implementation [39]: Keras Tuner is a library that helps to select the optimal set of hyperparameters for Tensorflow programs. The process of choosing the correct group of these parameters is known as *Hyperparameter Tuning or Hypertuning*.

Capítulo 7

Presupuesto

En este capítulo se presenta una estimación sobre el coste de realización de este proyecto.

7.1. Presupuesto de recursos humanos

En este apartado se muestra de forma desglosada el coste de los recursos humanos que ha sido invertido en el desarrollo.

Tareas	***Horas Empleadas***	***Coste Bruto***	***Coste Total Bruto***
Búsqueda y valoración de documentación relacionada con el ámbito del proyecto	80	20€	1.600€
Búsqueda y evaluación de los conjuntos de datos del archipiélago canario	12	20€	240€
Preprocesamiento y etiquetado de los datos	20	20€	400€
Elaboración y entrenamiento de los modelos	134	20€	2.680€
Evaluación de los modelos	24	20€	480€
Documentación	32	20€	640€
Total:	***302***		***6.040€***

Tabla 7.1: Presupuesto de los recursos humanos.

7.2. Presupuesto material

En este apartado se calculan los costes de los distintos componentes necesarios para la elaboración del proyecto.

Elemento	***Coste total***
Ordenador Personal	650€
Librerías	0€
IDE / Editor	0€
Total:	***650€***

Tabla 7.2: Presupuesto en materiales.

7.3. Presupuesto final

El presupuesto final del proyecto, teniendo en cuenta todos los costes previamente mencionados, quedaría de la siguiente manera.

Elemento	***Coste total***
Coste de RRHH	6.040€
Coste de materiales	650€
Total:	***6.690€***

Tabla 7.3: Presupuesto final.

Apéndice A

Gráficas simples de modelos propuestos

A.1. Gráficas simples *Model Loss*

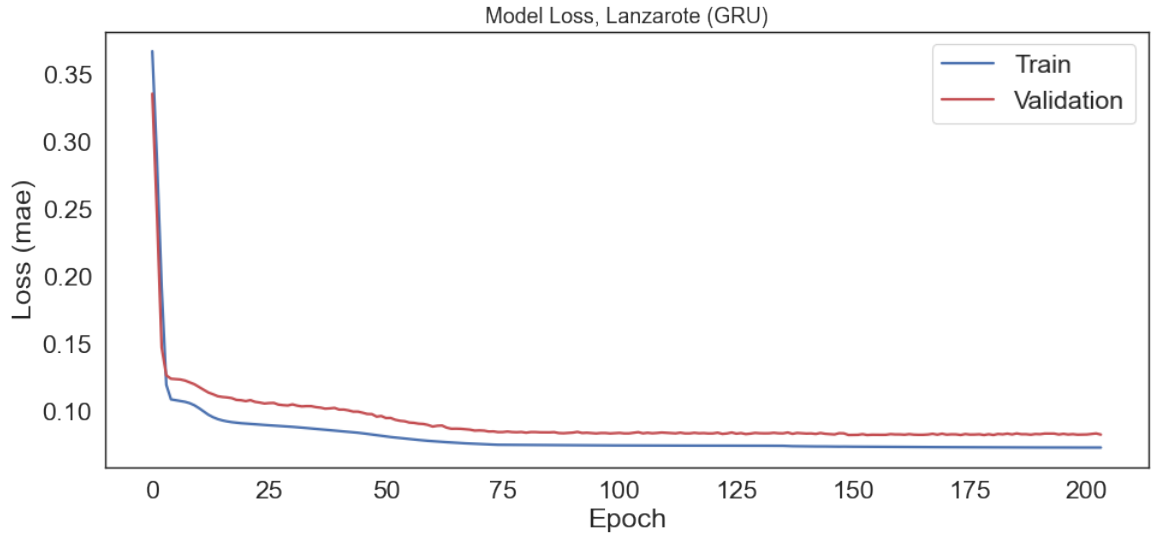


Figura A.1: Gráfico *Model Loss* para la isla de Lanzarote con capas Gated Recurrent Unit (GRU).

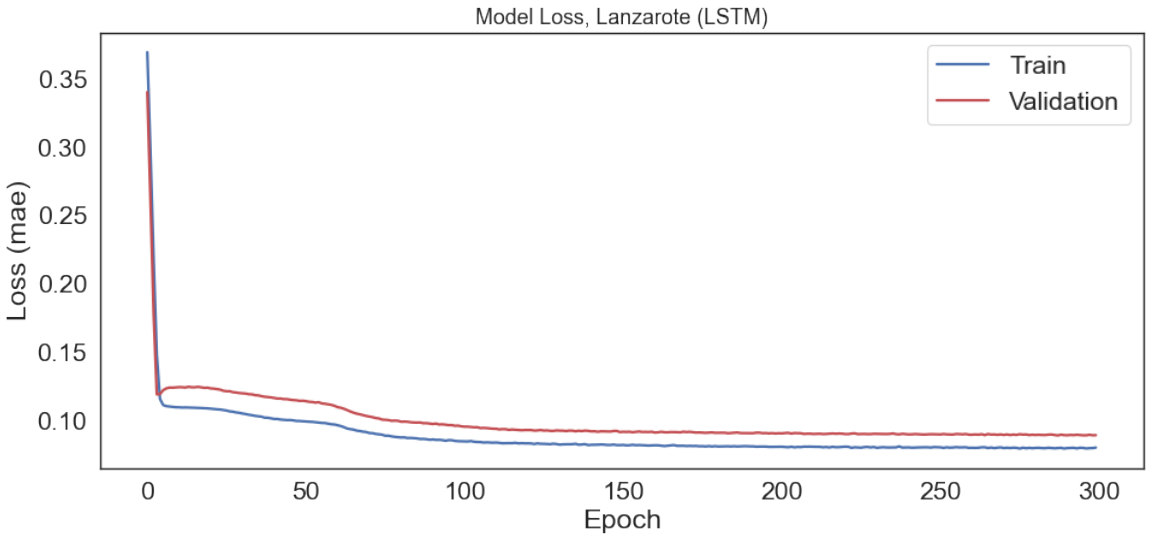


Figura A.2: Gráfico *Model Loss* para la isla de Lanzarote con capas Long Short-Term Memory (LSTM).

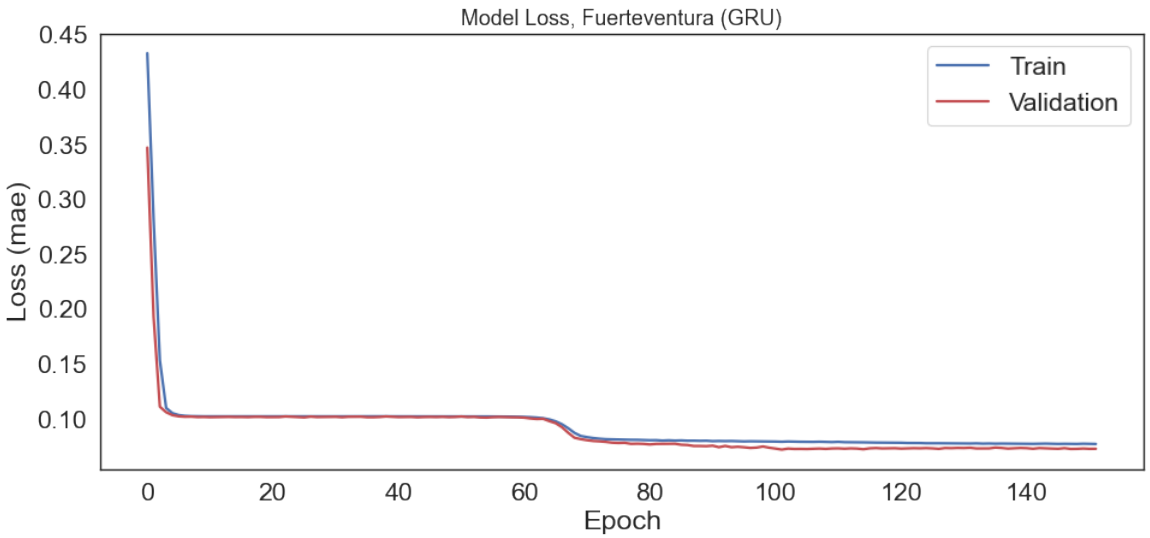


Figura A.3: Gráfico *Model Loss* para la isla de Fuerteventura con capas Gated Recurrent Unit (GRU).

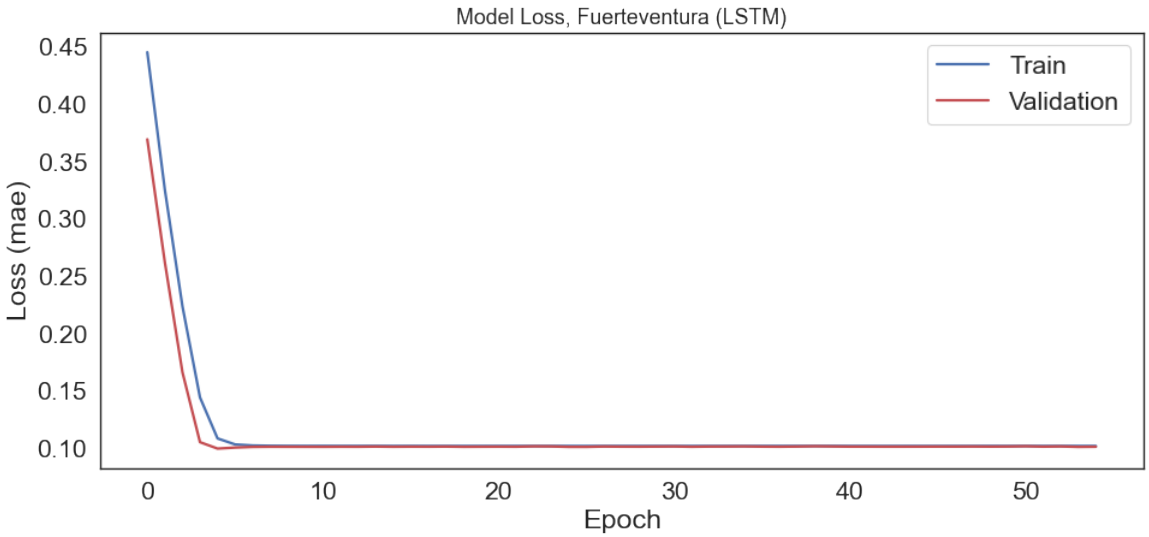


Figura A.4: Gráfico *Model Loss* para la isla de Fuerteventura con capas Long Short-Term Memory (LSTM).

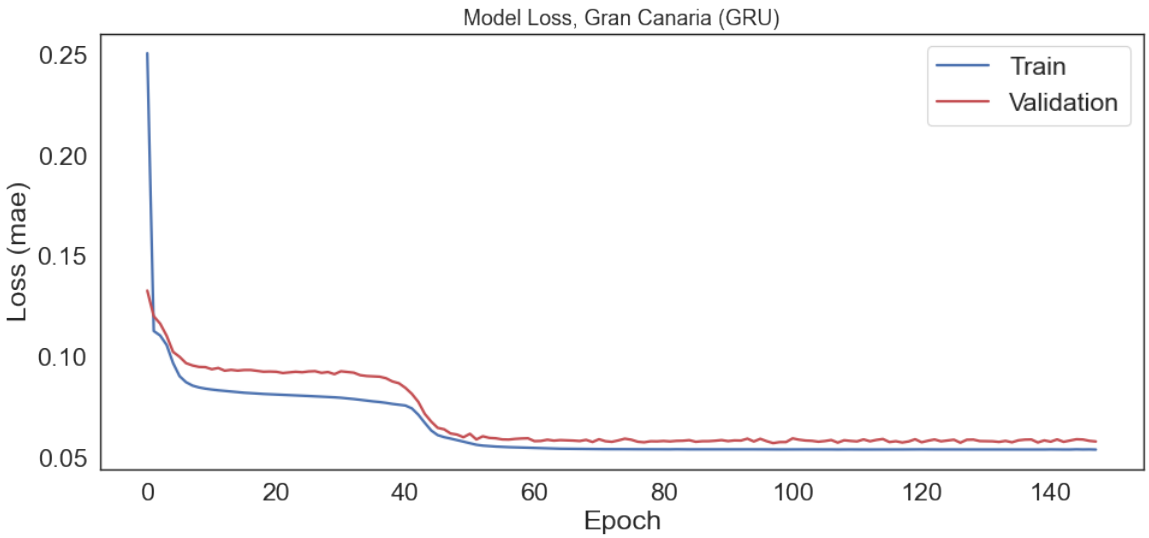


Figura A.5: Gráfico *Model Loss* para la isla de Gran Canaria con capas Gated Recurrent Unit (GRU).

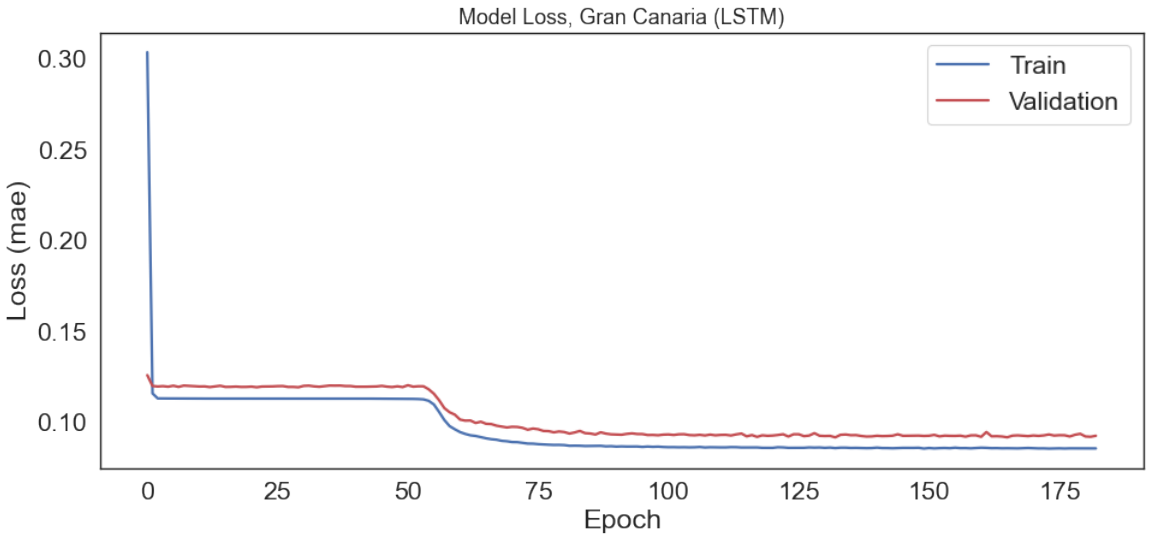


Figura A.6: Gráfico *Model Loss* para la isla de Gran Canaria con capas Long Short-Term Memory (LSTM).

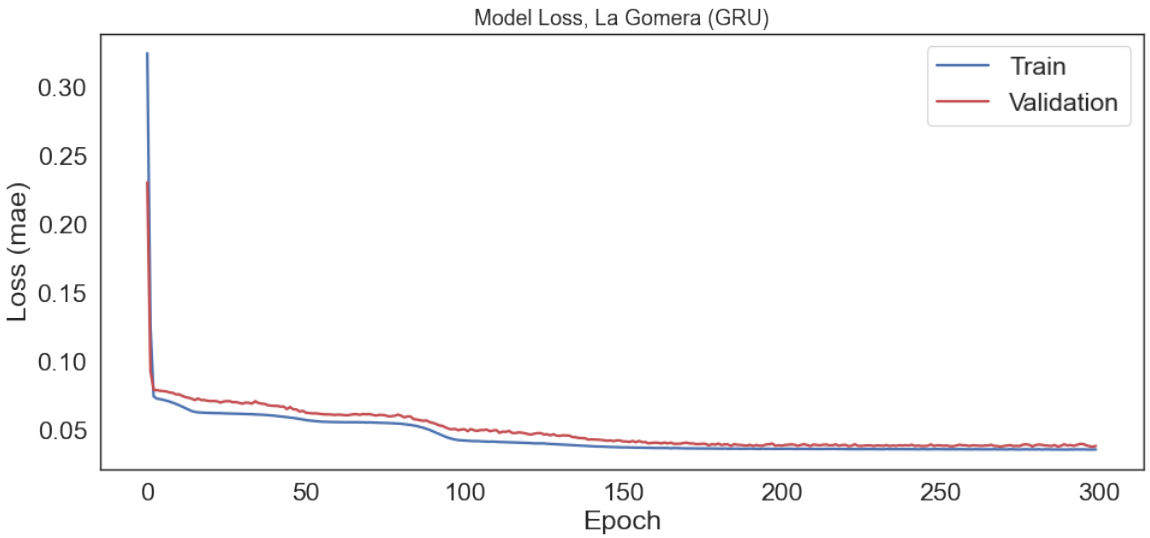


Figura A.7: Gráfico *Model Loss* para la isla de La Gomera con capas Gated Recurrent Unit (GRU).

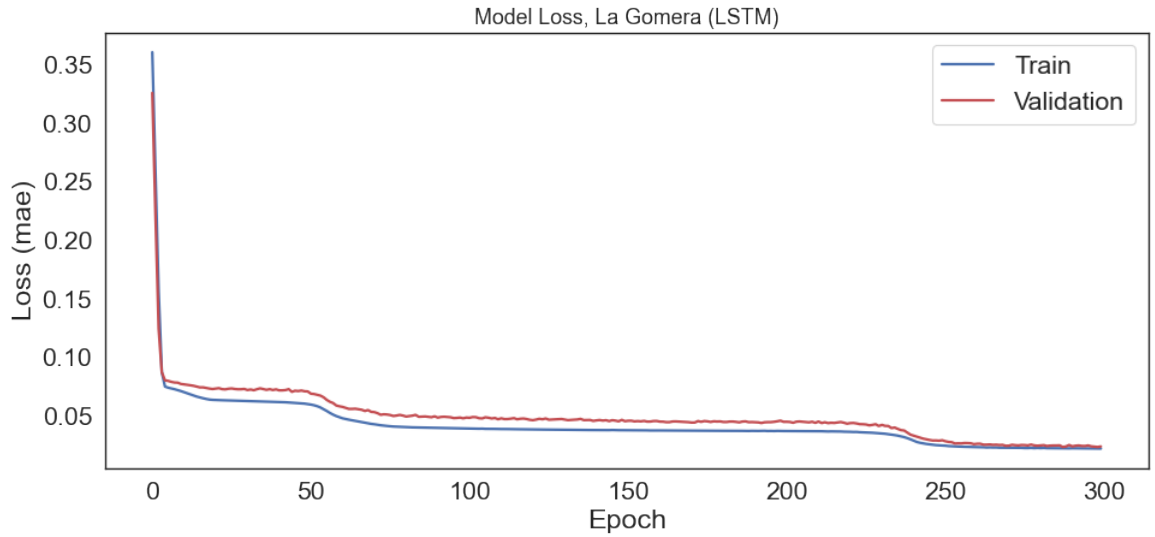


Figura A.8: Gráfico *Model Loss* para la isla de La Gomera con capas Long Short-Term Memory (LSTM).

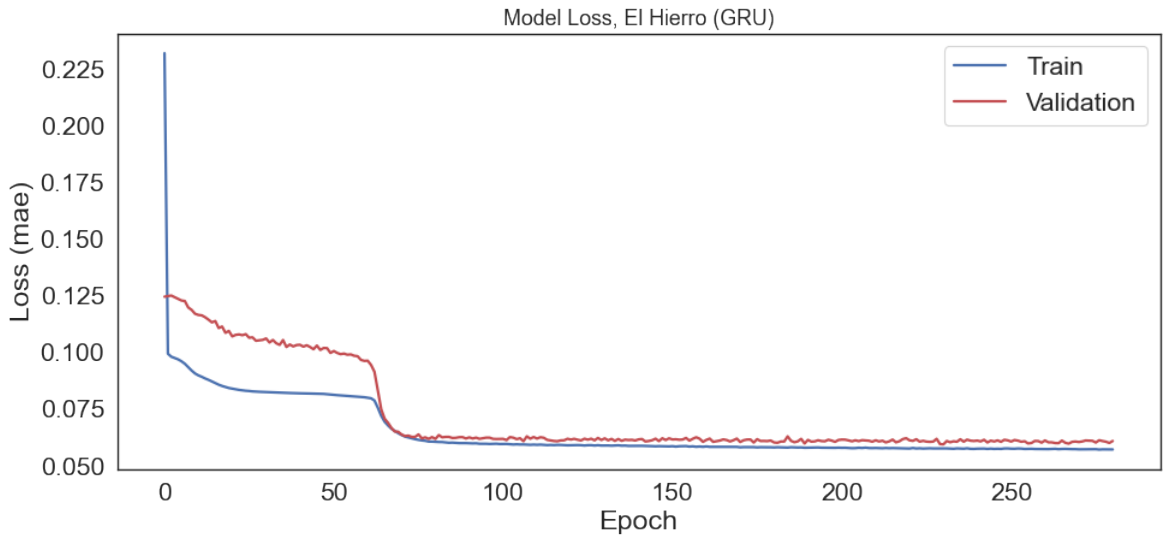


Figura A.9: Gráfico *Model Loss* para la isla de El Hierro con capas Gated Recurrent Unit (GRU).

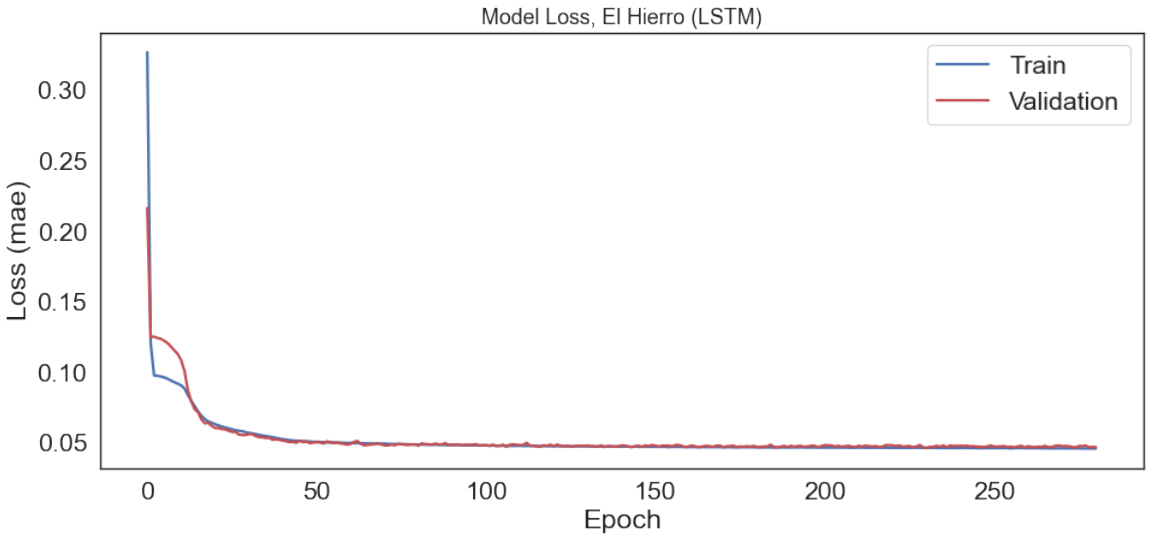


Figura A.10: Gráfico *Model Loss* para la isla de El Hierro con capas Long Short-Term Memory (LSTM).

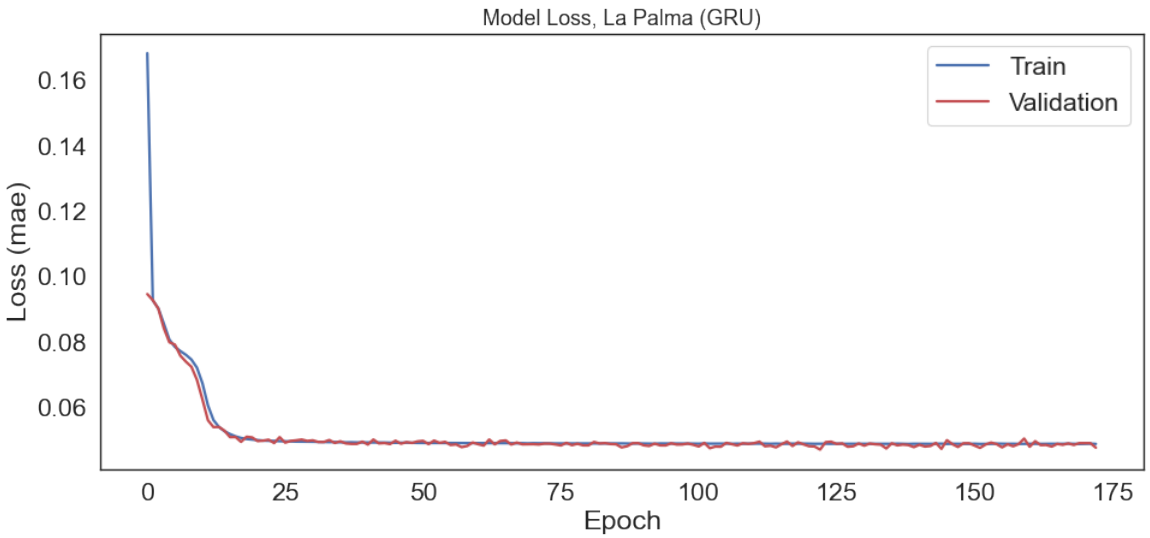


Figura A.11: Gráfico *Model Loss* para la isla de La Palma con capas Gated Recurrent Unit (GRU).

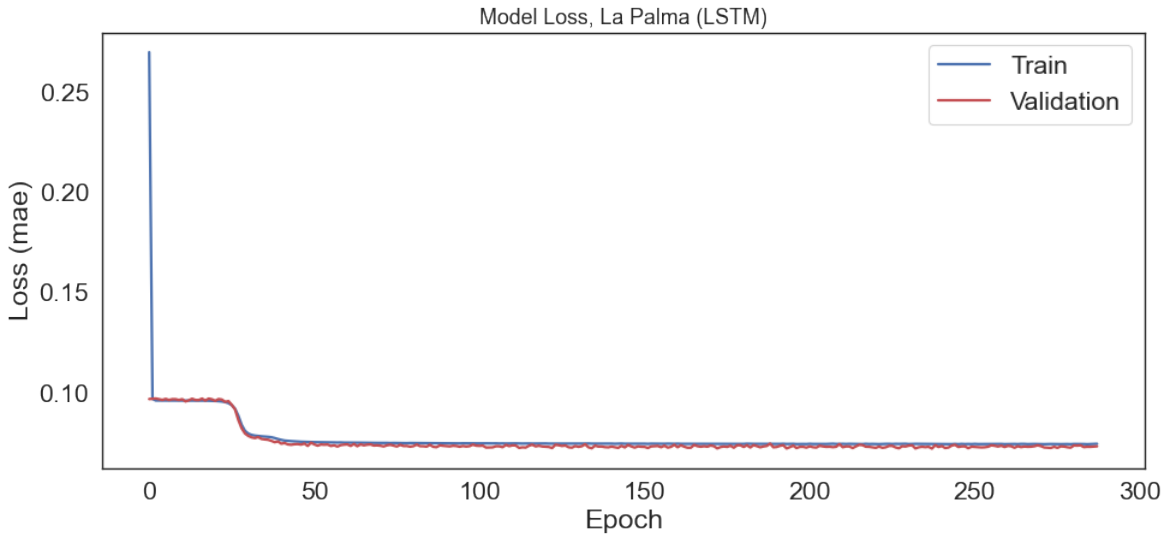


Figura A.12: Gráfico *Model Loss* para la isla de La Palma con capas Long Short-Term Memory (LSTM).

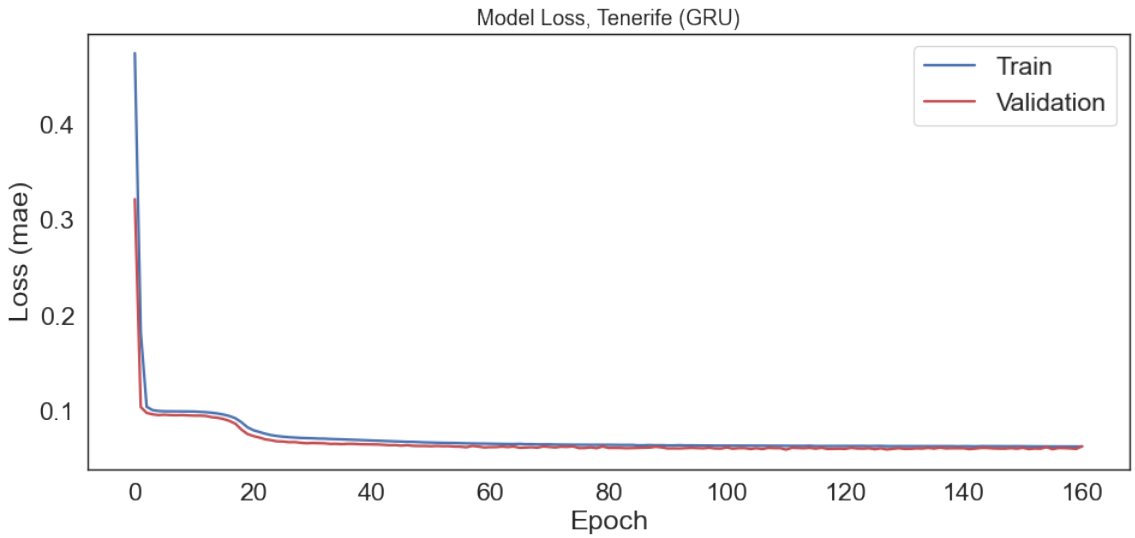


Figura A.13: Gráfico *Model Loss* para la isla de Tenerife con capas Gated Recurrent Unit (GRU).

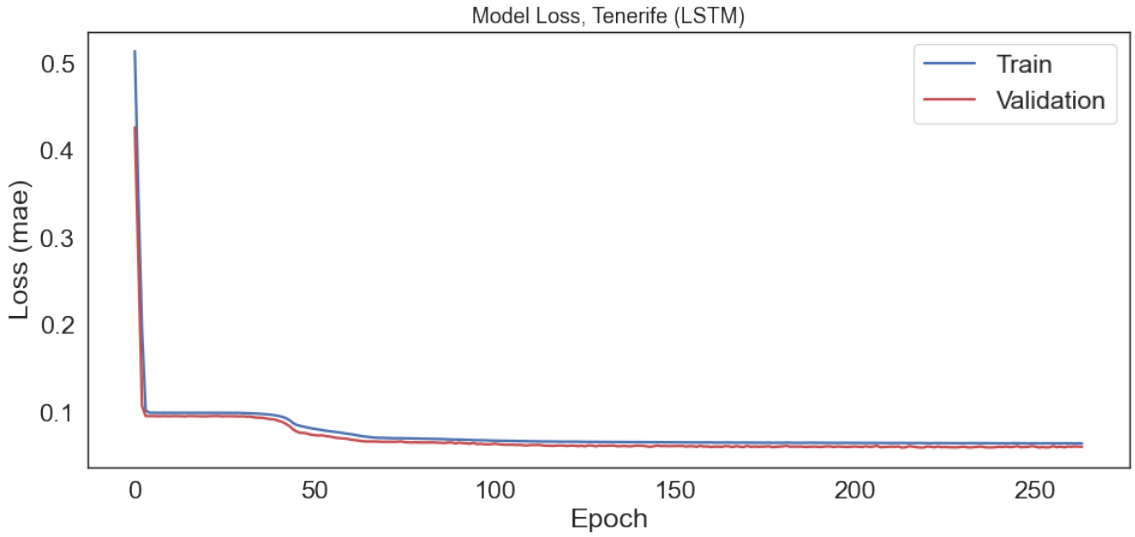


Figura A.14: Gráfico *Model Loss* para la isla de Tenerife con capas Long Short-Term Memory (LSTM).

A.2. Gráficas simples *Precision/Recall VS Threshold*

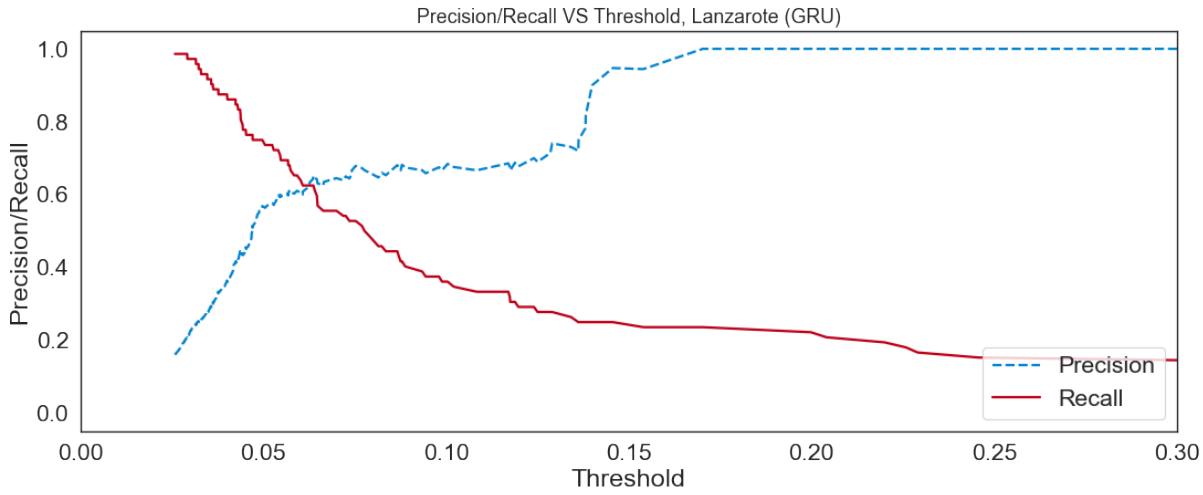


Figura A.15: Gráfico *Precision/Recall VS Threshold* para la isla de Lanzarote con capas Gated Recurrent Unit (GRU).

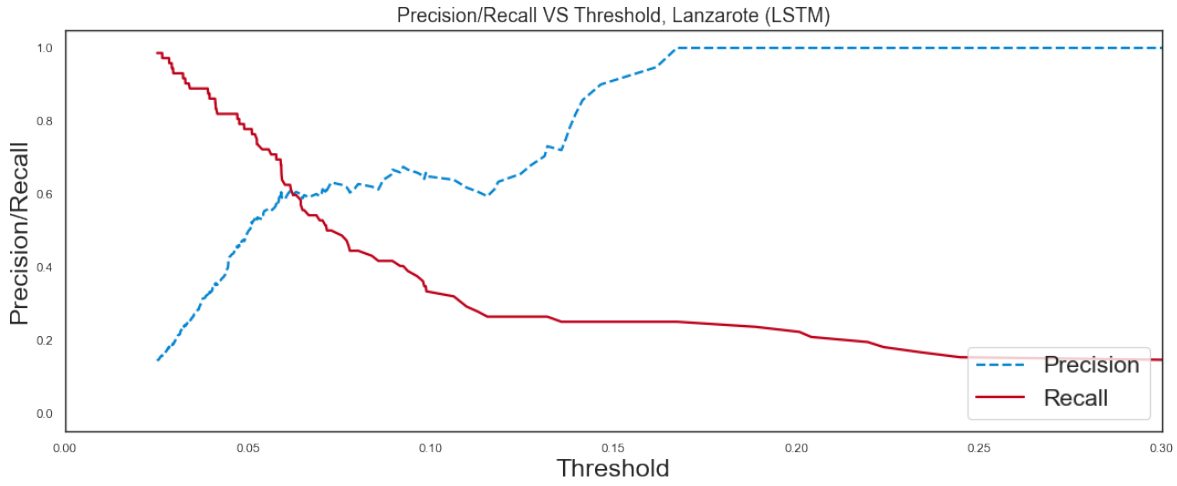


Figura A.16: Gráfico *Precision/Recall VS Threshold* para la isla de Lanzarote con capas Long Short-Term Memory (LSTM).

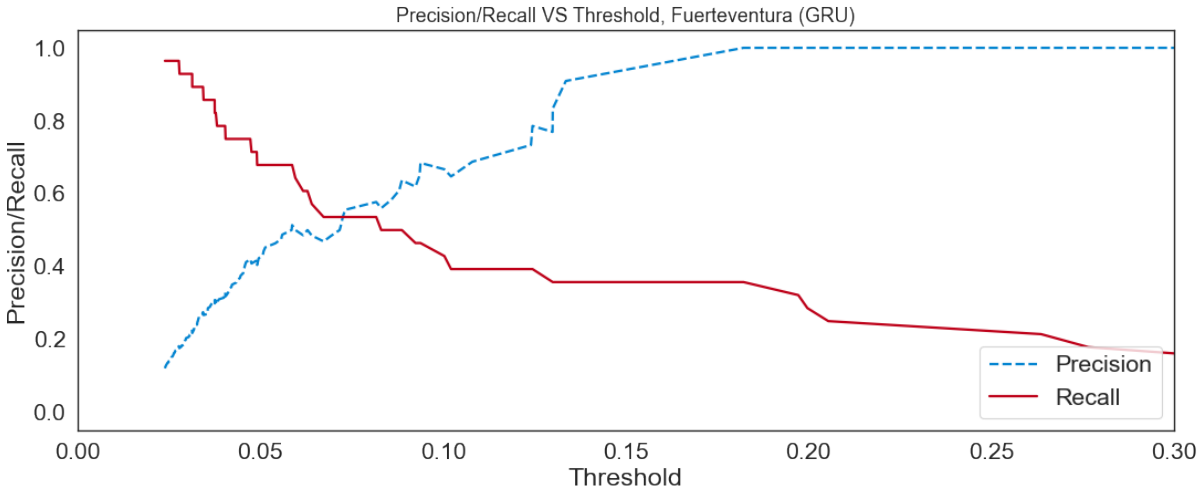


Figura A.17: Gráfico *Precision/Recall VS Threshold* para la isla de Fuerteventura con capas Gated Recurrent Unit (GRU).

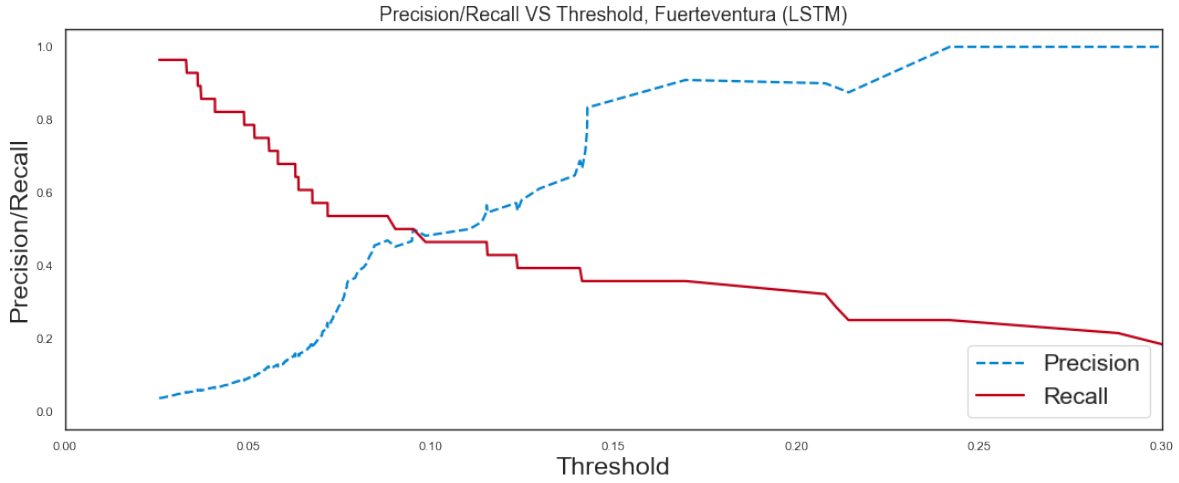


Figura A.18: Gráfico *Precision/Recall VS Threshold* para la isla de Fuerteventura con capas Long Short-Term Memory (LSTM).

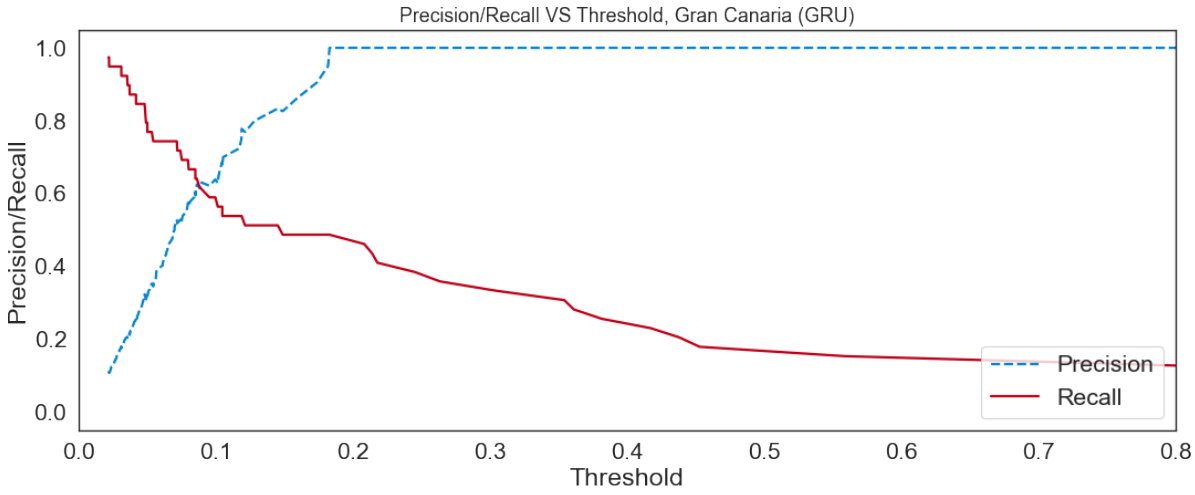


Figura A.19: Gráfico *Precision/Recall VS Threshold* para la isla de Gran Canaria con capas Gated Recurrent Unit (GRU).

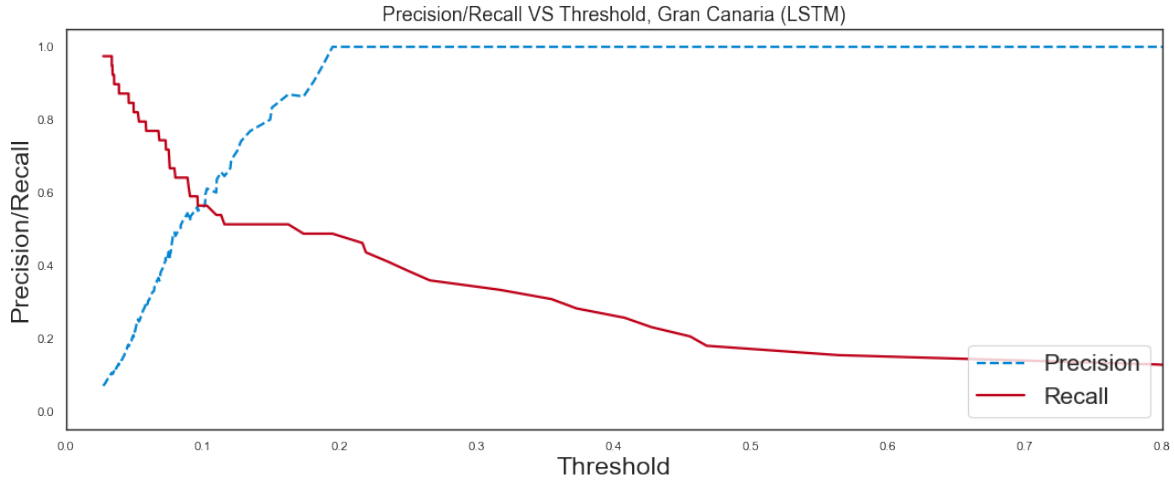


Figura A.20: Gráfico *Precision/Recall VS Threshold* para la isla de Gran Canaria con capas Long Short-Term Memory (LSTM).

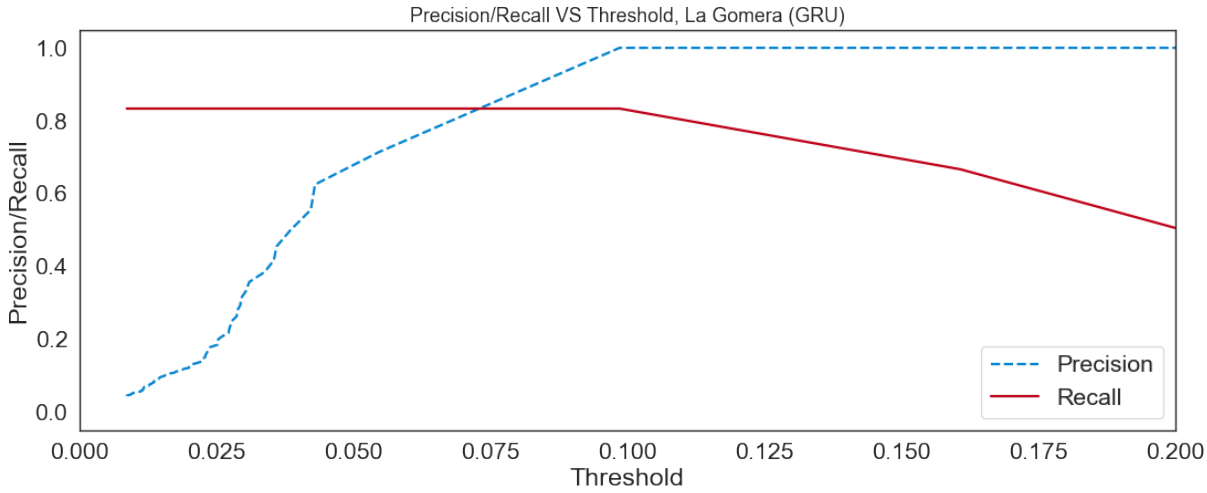


Figura A.21: Gráfico *Precision/Recall VS Threshold* para la isla de La Gomera con capas Gated Recurrent Unit (GRU).

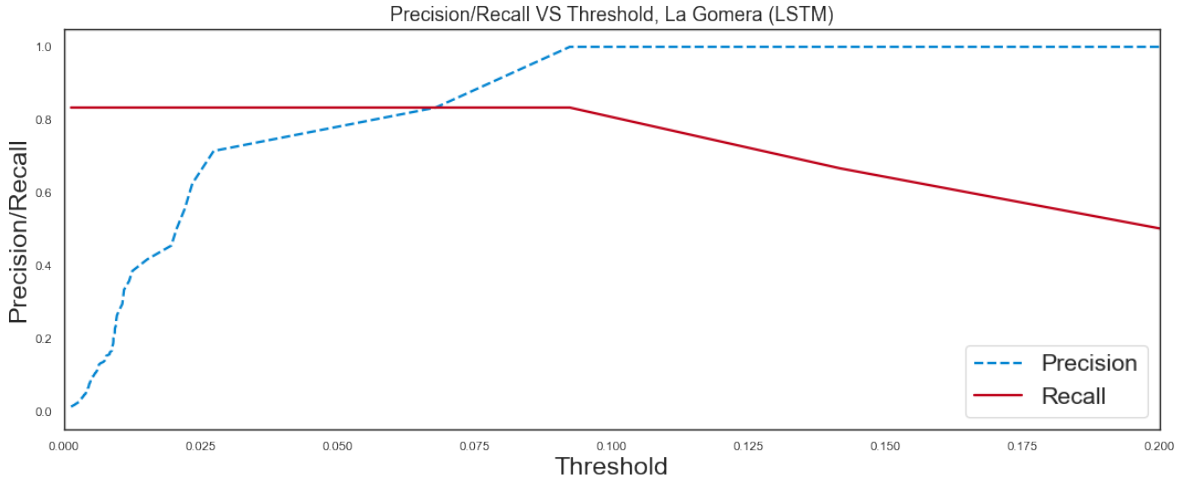


Figura A.22: Gráfico *Precision/Recall VS Threshold* para la isla de La Gomera con capas Long Short-Term Memory (LSTM).

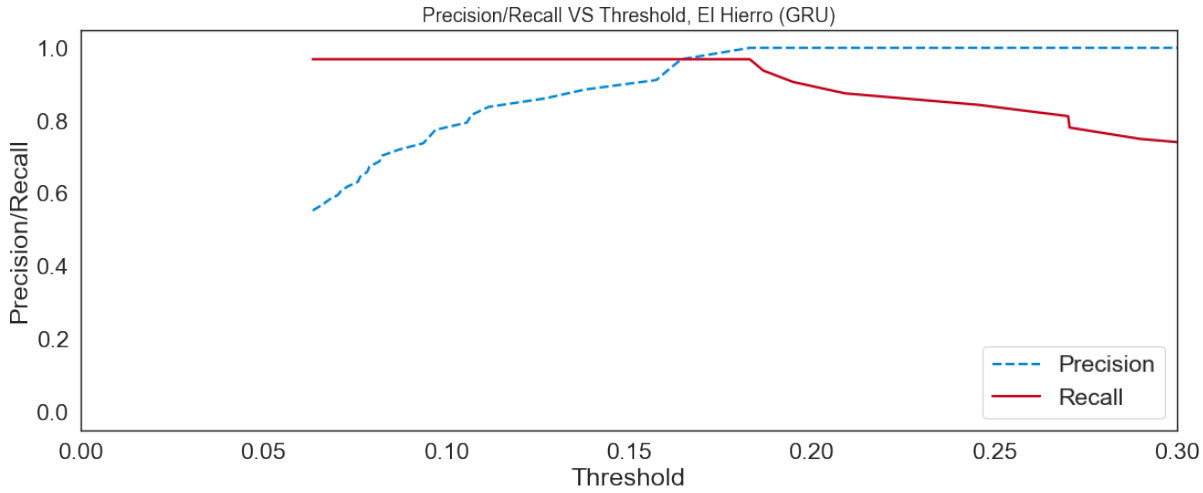


Figura A.23: Gráfico *Precision/Recall VS Threshold* para la isla de El Hierro con capas Gated Recurrent Unit (GRU).

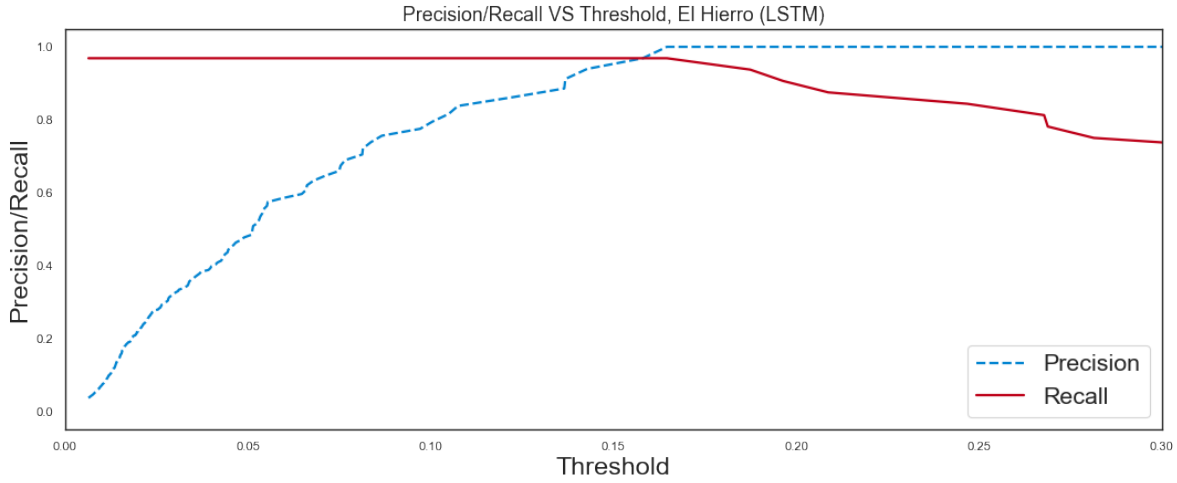


Figura A.24: Gráfico *Precision/Recall VS Threshold* para la isla de El Hierro con capas Long Short-Term Memory (LSTM).

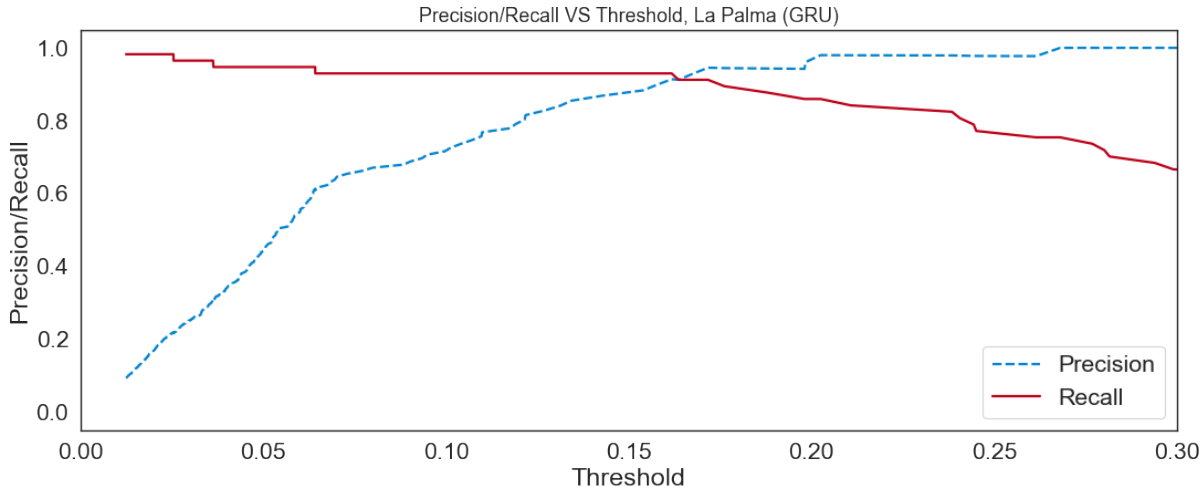


Figura A.25: Gráfico *Precision/Recall VS Threshold* para la isla de La Palma con capas Gated Recurrent Unit (GRU).

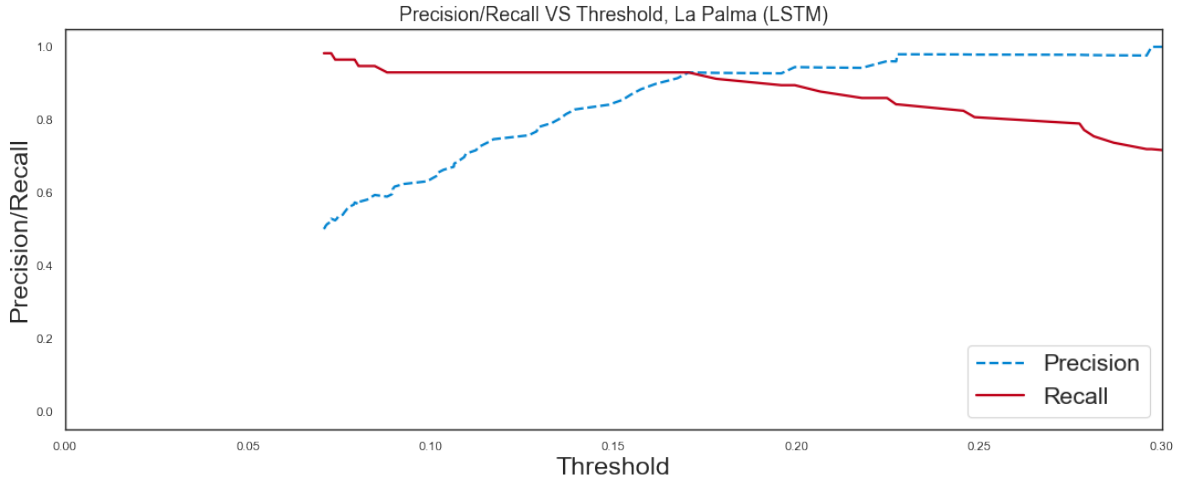


Figura A.26: Gráfico *Precision/Recall VS Threshold* para la isla de La Palma con capas Long Short-Term Memory (LSTM).

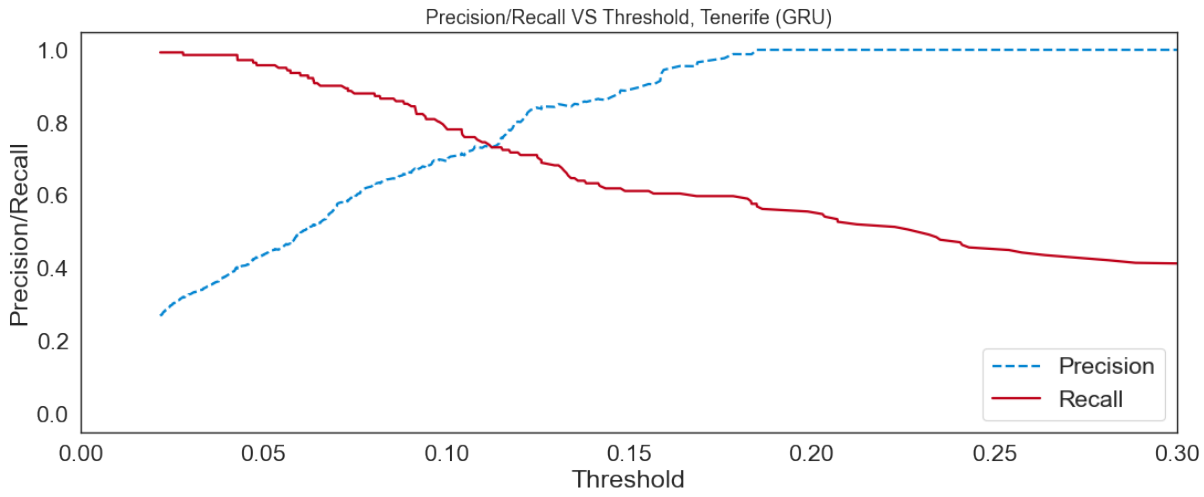


Figura A.27: Gráfico *Precision/Recall VS Threshold* para la isla de Tenerife con capas Gated Recurrent Unit (GRU).

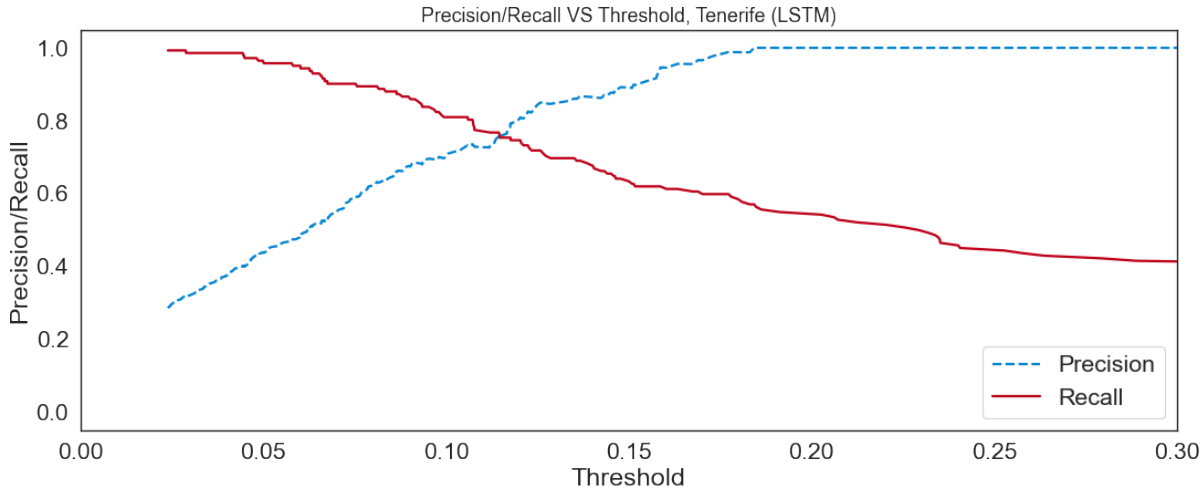


Figura A.28: Gráfico *Precision/Recall VS Threshold* para la isla de Tenerife con capas Long Short-Term Memory (LSTM).

A.3. Gráficas simples *Receiver Operating Characteristic Curve (ROC)*

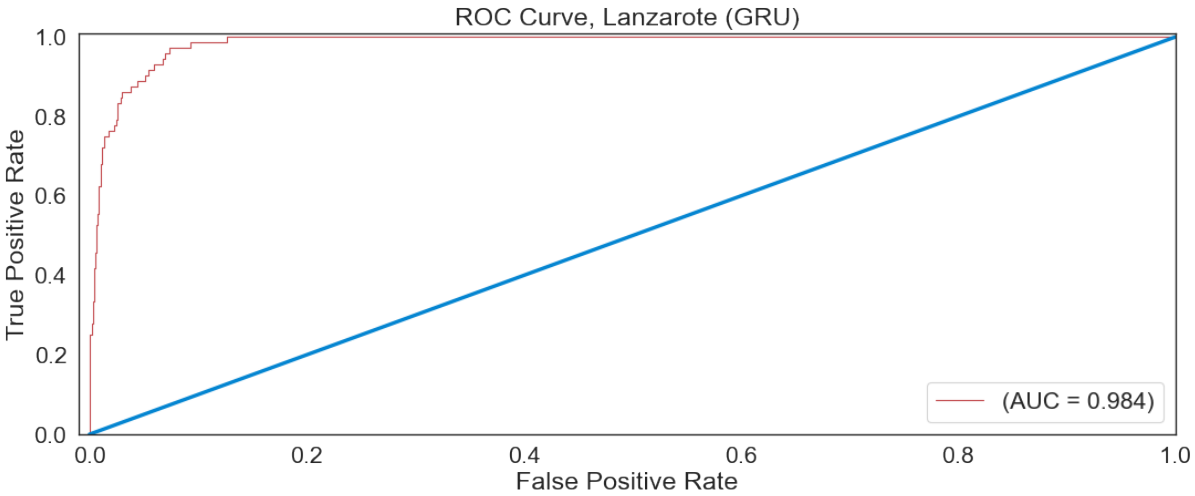


Figura A.29: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Lanzarote con capas Gated Recurrent Unit (GRU).

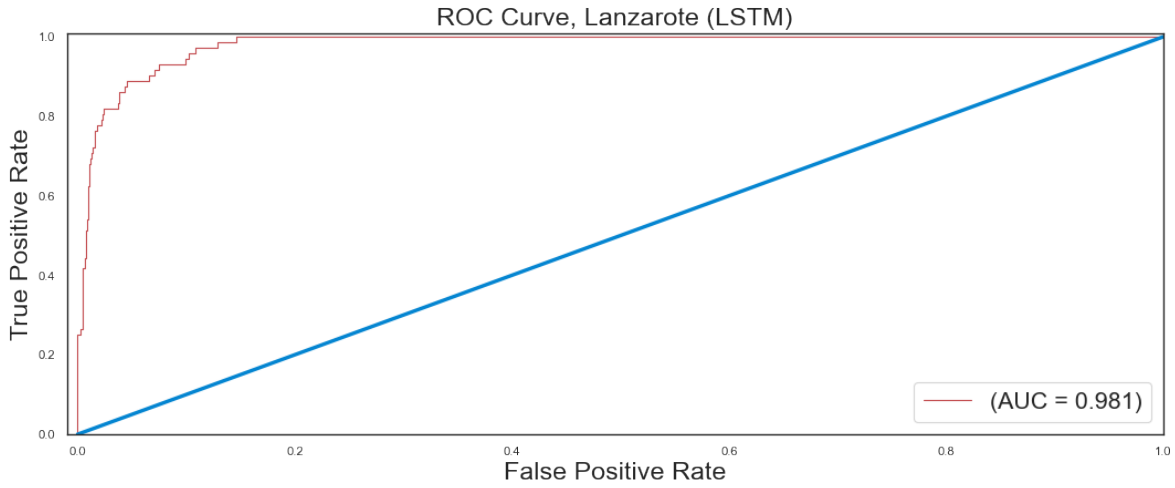


Figura A.30: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Lanzarote con capas Long Short-Term Memory (LSTM).

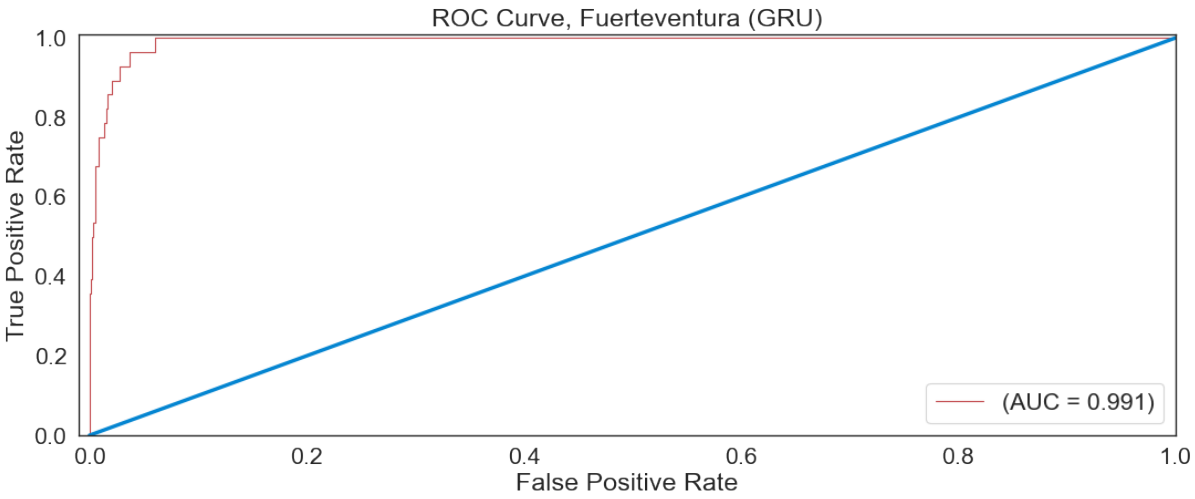


Figura A.31: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Fuerteventura con capas Gated Recurrent Unit (GRU).

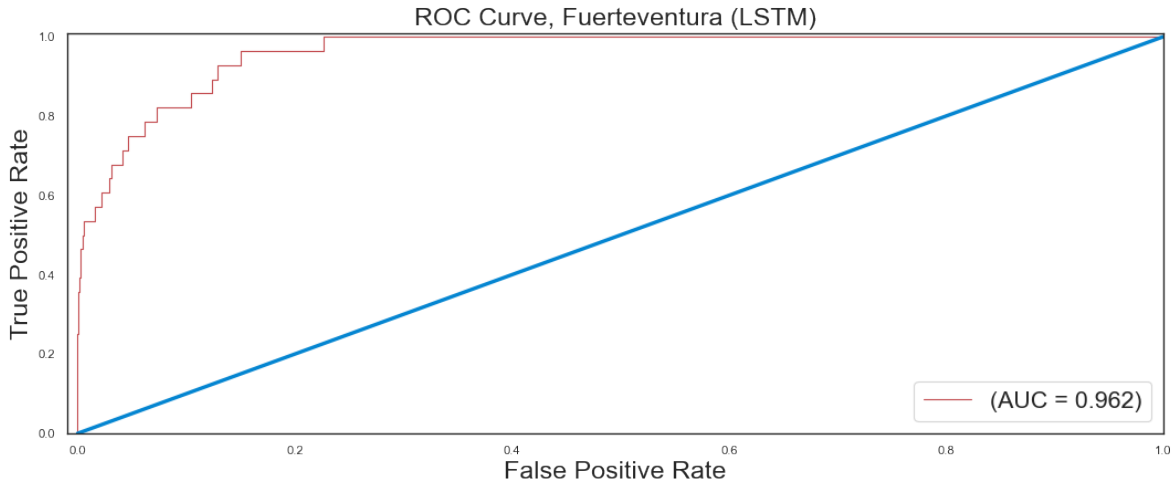


Figura A.32: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Fuerteventura con capas Long Short-Term Memory (LSTM).

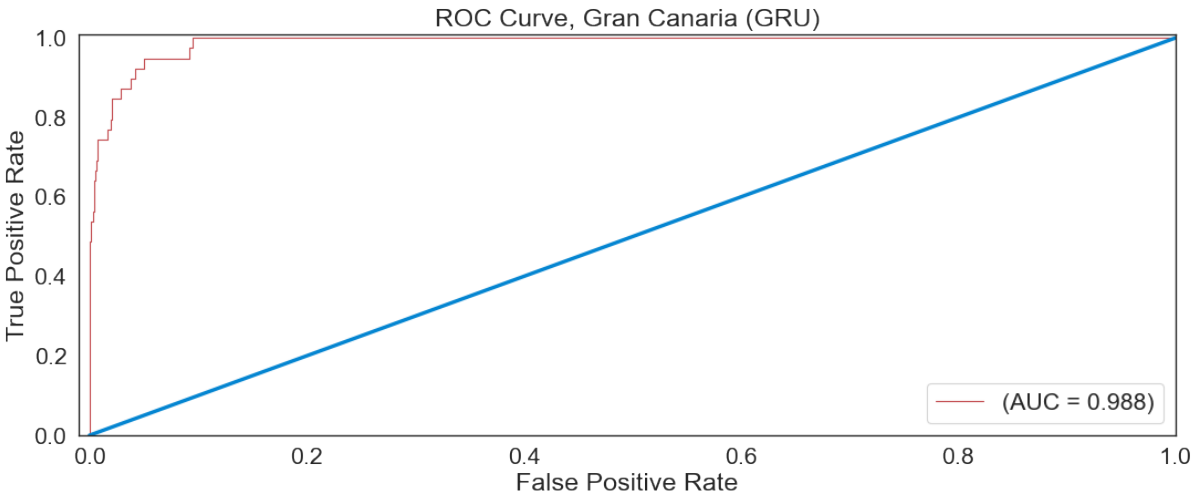


Figura A.33: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Gran Canaria con capas Gated Recurrent Unit (GRU).

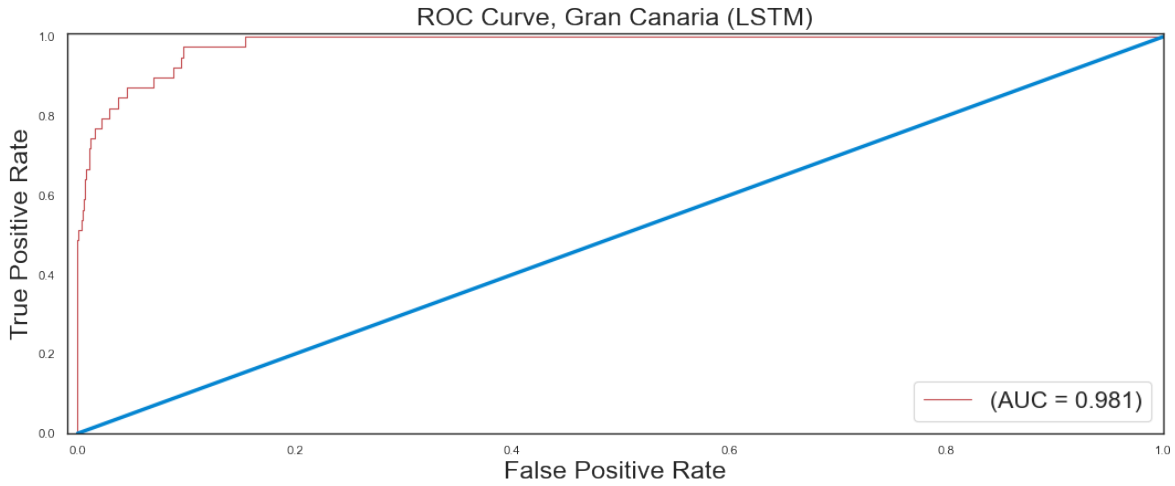


Figura A.34: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Gran Canaria con capas Long Short-Term Memory (LSTM).

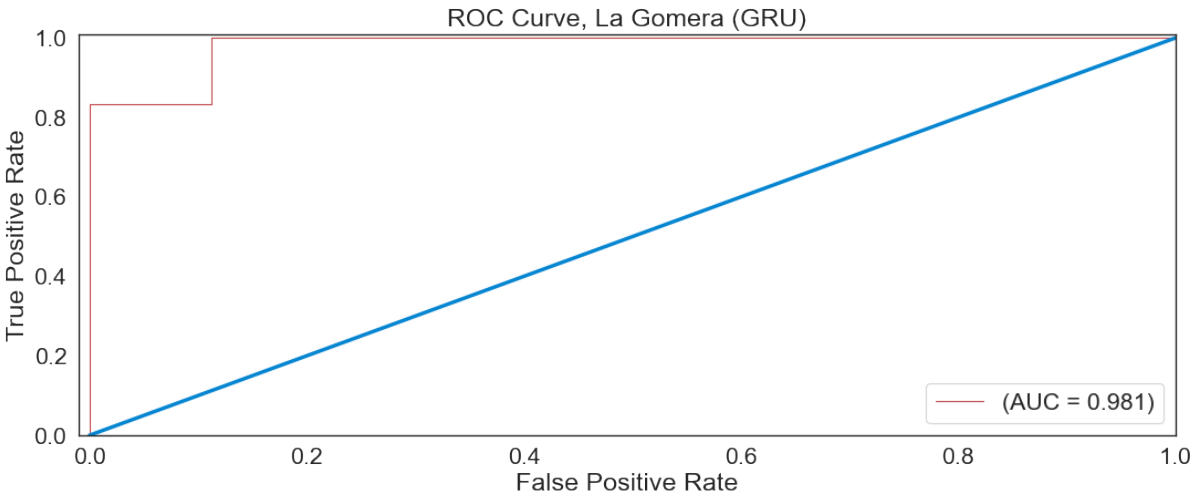


Figura A.35: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de La Gomera con capas Gated Recurrent Unit (GRU).

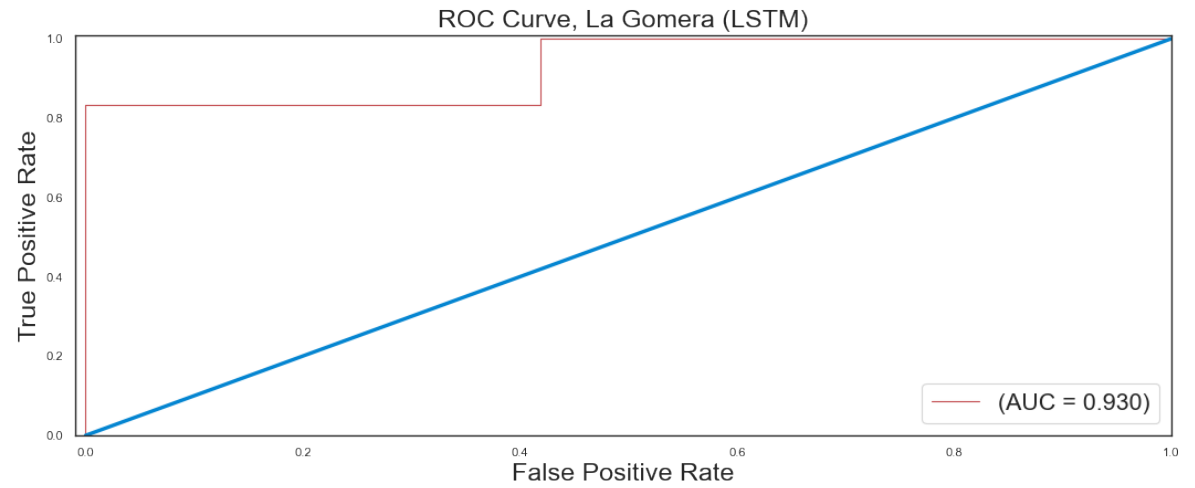


Figura A.36: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de La Gomera con capas Long Short-Term Memory (LSTM).

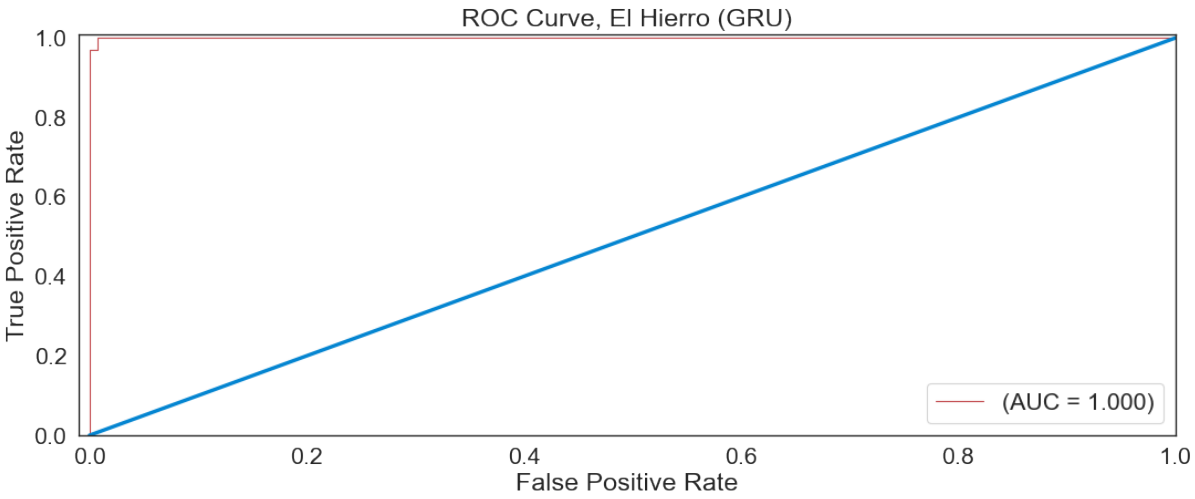


Figura A.37: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de El Hierro con capas Gated Recurrent Unit (GRU).

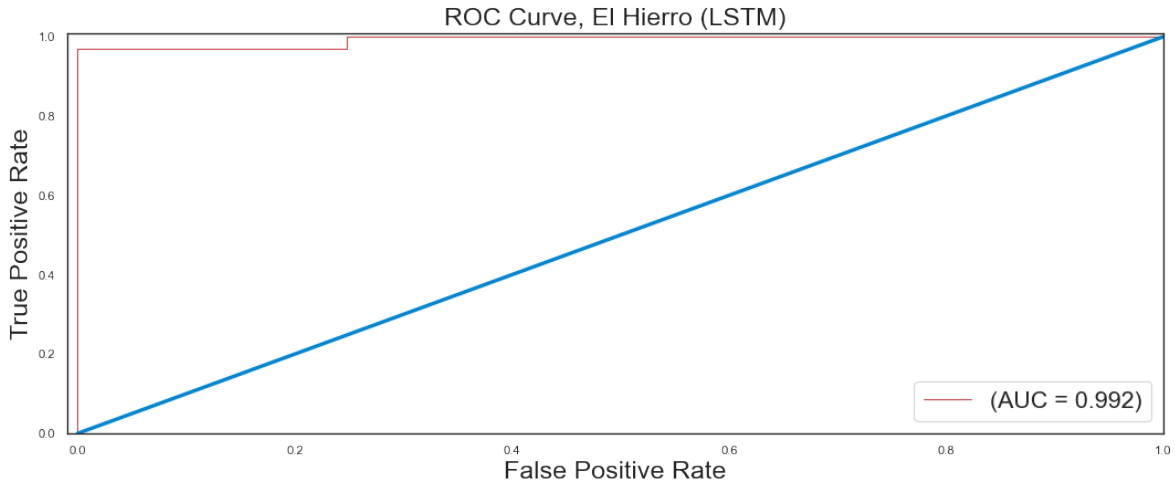


Figura A.38: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de El Hierro con capas Long Short-Term Memory (LSTM).

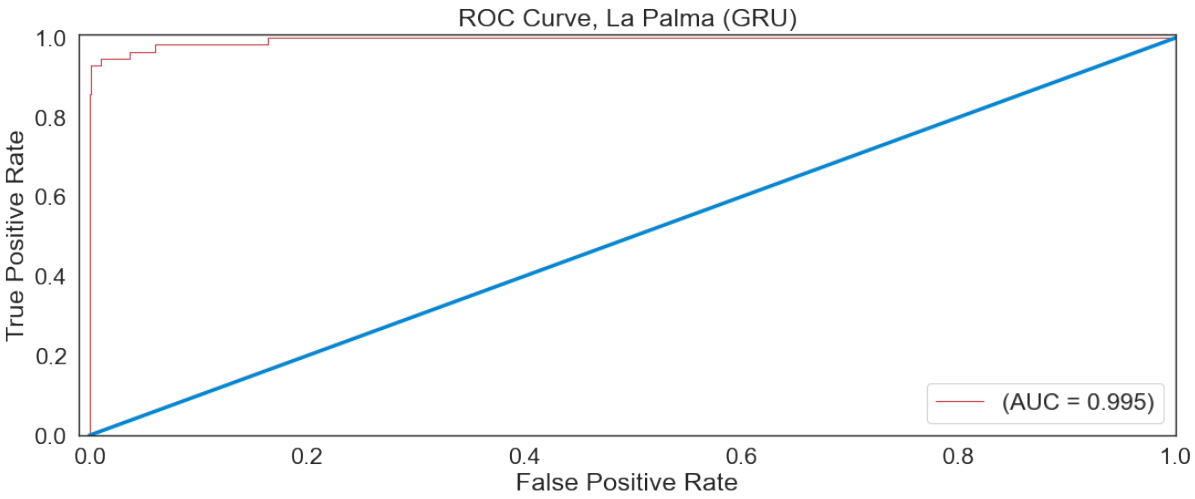


Figura A.39: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de La Palma con capas Gated Recurrent Unit (GRU).

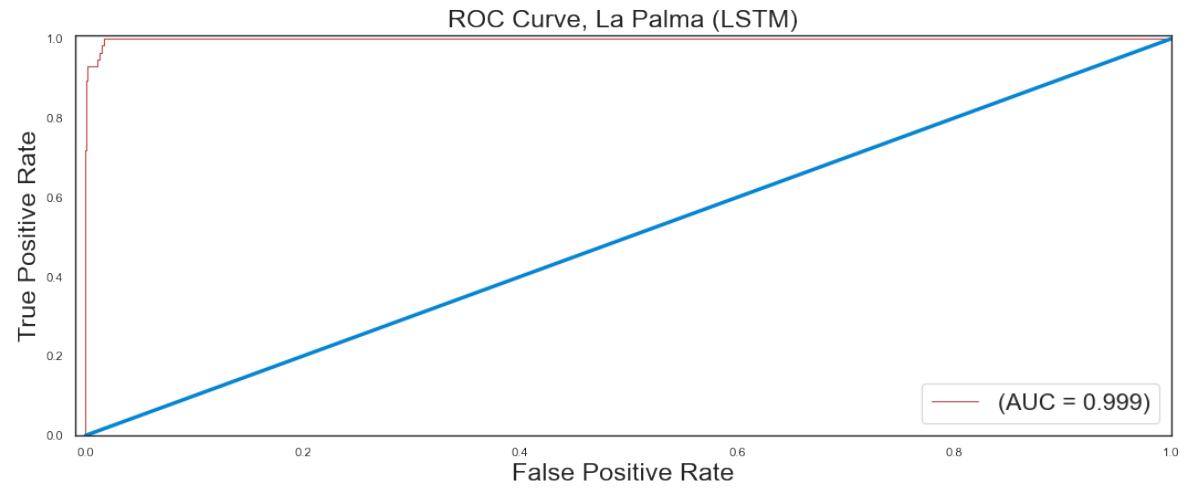


Figura A.40: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de La Palma con capas Long Short-Term Memory (LSTM).

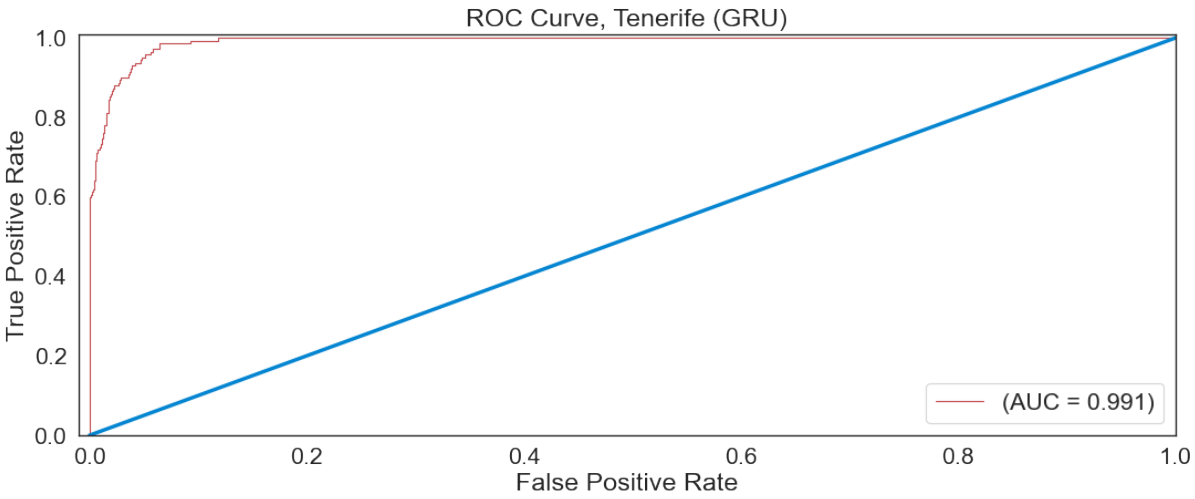


Figura A.41: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Tenerife con capas Gated Recurrent Unit (GRU).

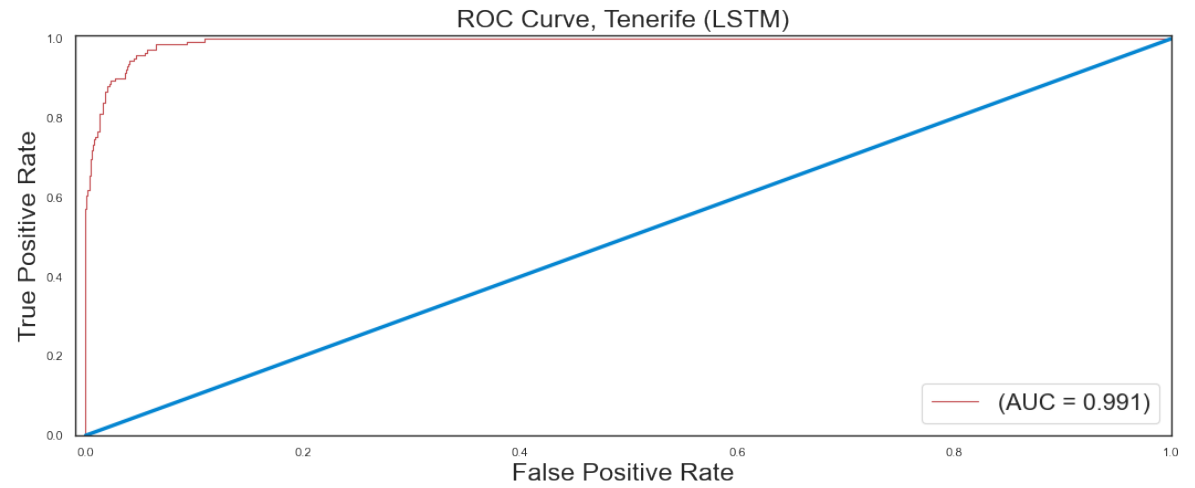


Figura A.42: Gráfico *Receiver Operating Characteristic Curve (ROC)* para la isla de Tenerife con capas Long Short-Term Memory (LSTM).

A.4. Gráficas simples *Precision/Recall*

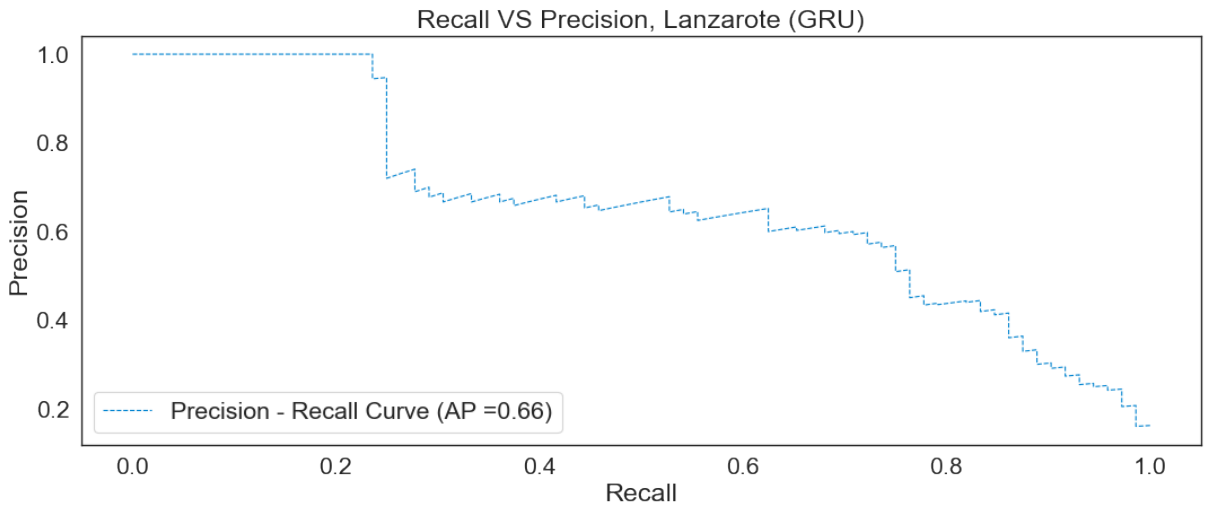


Figura A.43: Gráfico *Precision/Recall* para la isla de Lanzarote con capas Gated Recurrent Unit (GRU).

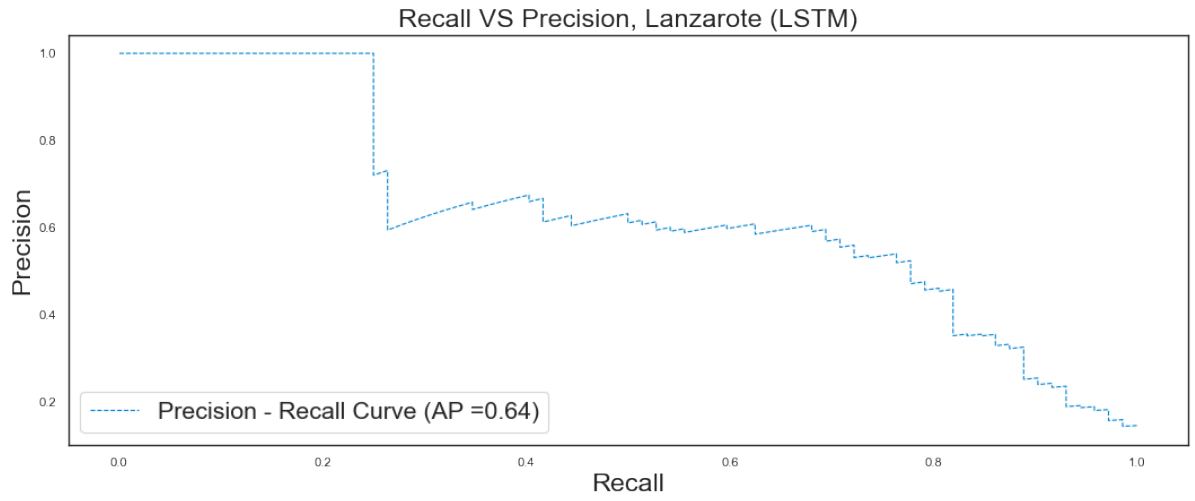


Figura A.44: Gráfico *Precision/Recall* para la isla de Lanzarote con capas Long Short-Term Memory (LSTM).

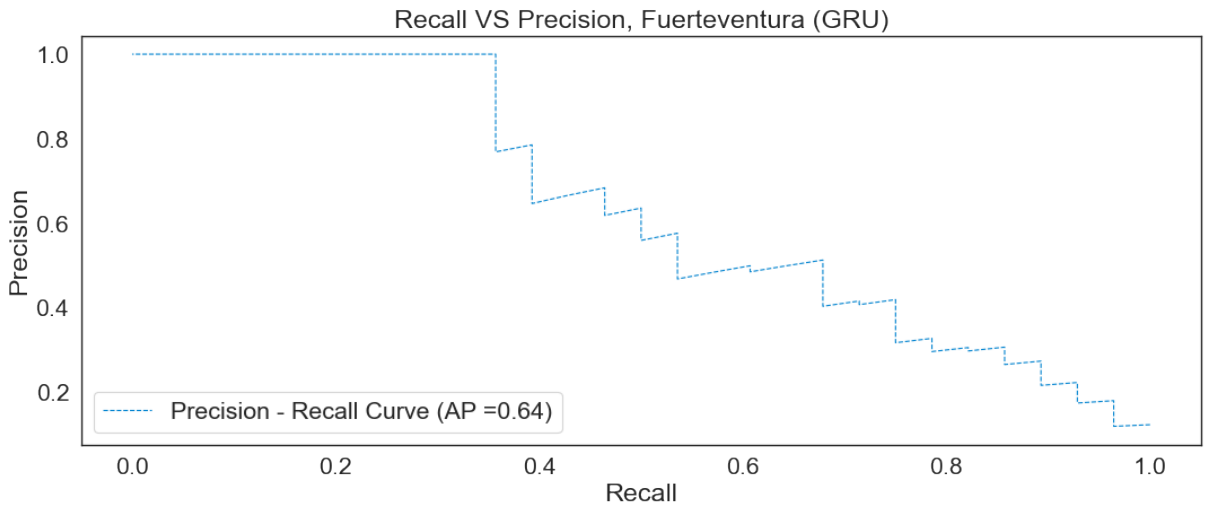


Figura A.45: Gráfico *Precision/Recall* para la isla de Fuerteventura con capas Gated Recurrent Unit (GRU).

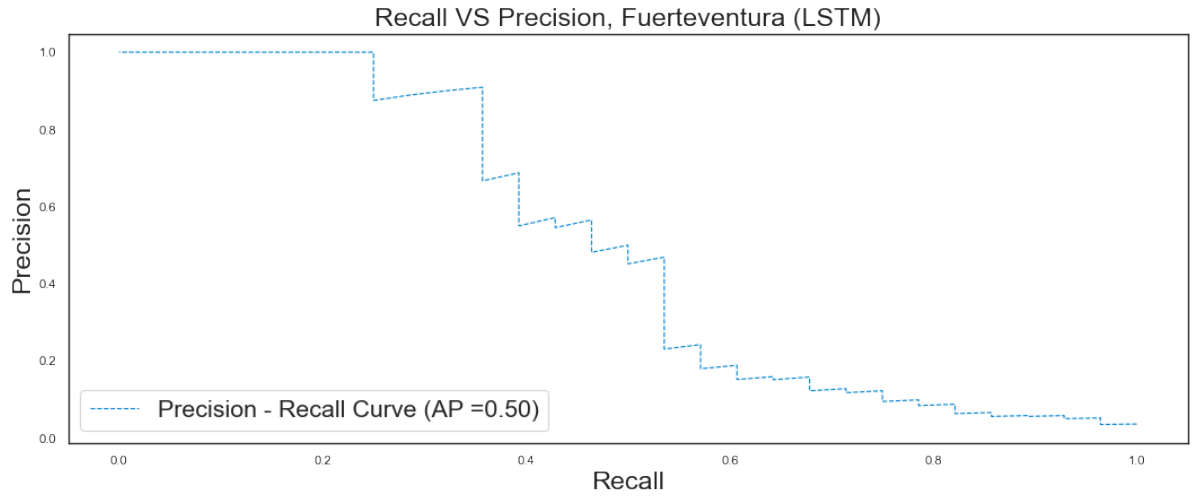


Figura A.46: Gráfico *Precision/Recall* para la isla de Fuerteventura con capas Long Short-Term Memory (LSTM).

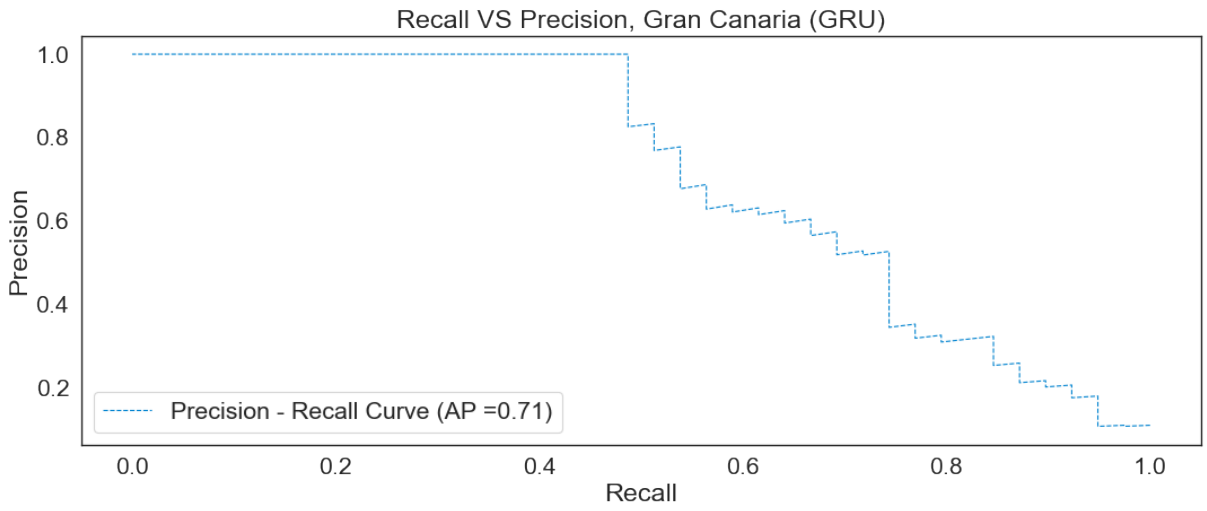


Figura A.47: Gráfico *Precision/Recall* para la isla de Gran Canaria con capas Gated Recurrent Unit (GRU).

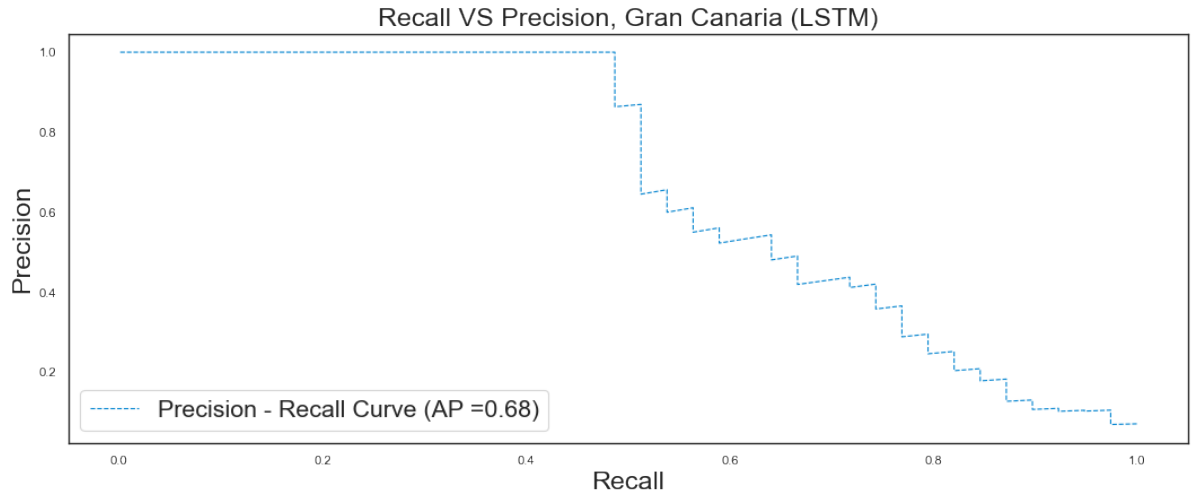


Figura A.48: Gráfico *Precision/Recall* para la isla de Gran Canaria con capas Long Short-Term Memory (LSTM).

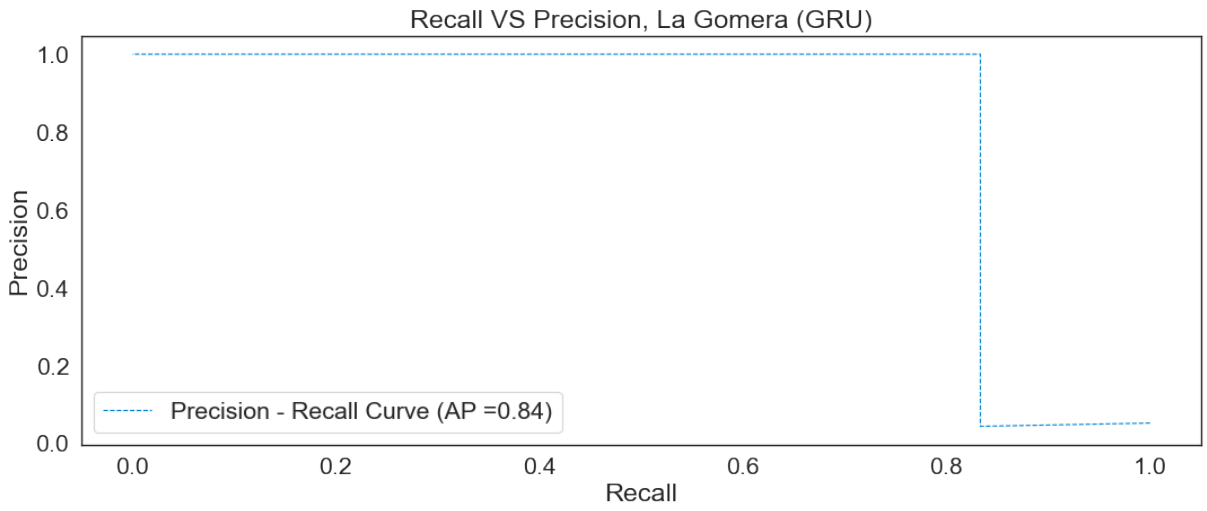


Figura A.49: Gráfico *Precision/Recall* para la isla de La Gomera con capas Gated Recurrent Unit (GRU).

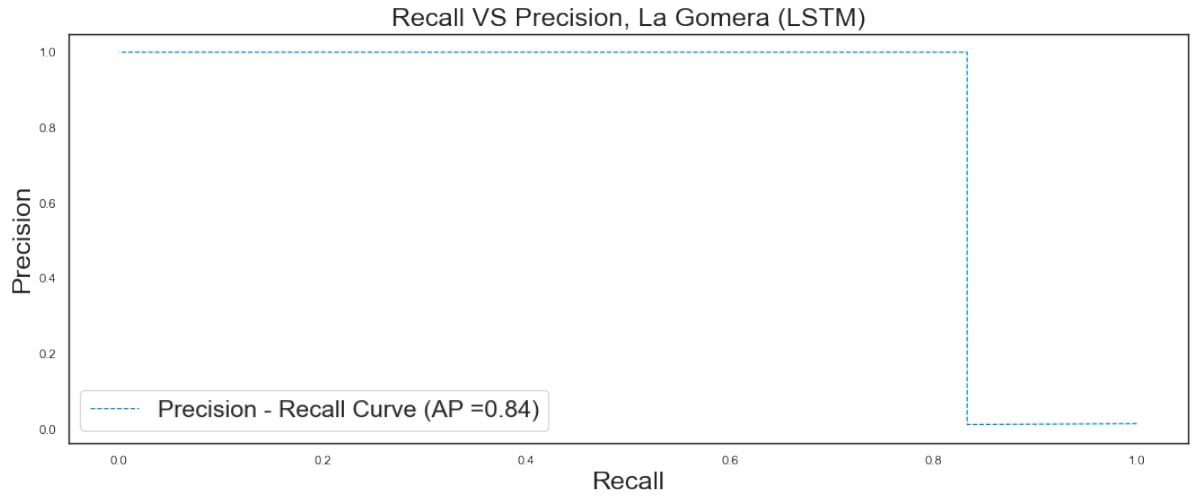


Figura A.50: Gráfico *Precision/Recall* para la isla de La Gomera con capas Long Short-Term Memory (LSTM).

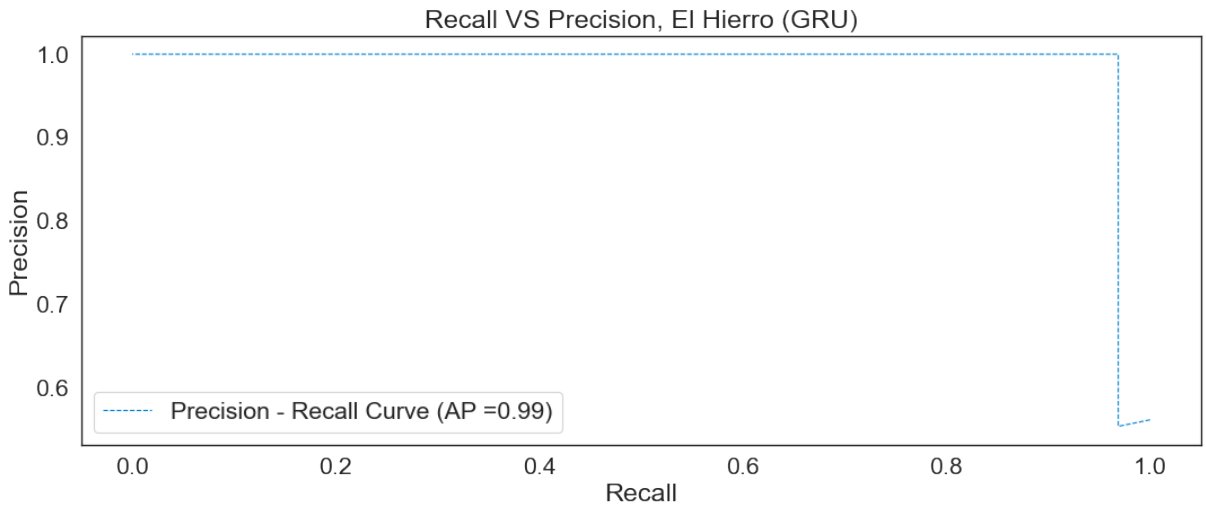


Figura A.51: Gráfico *Precision/Recall* para la isla de El Hierro con capas Gated Recurrent Unit (GRU).

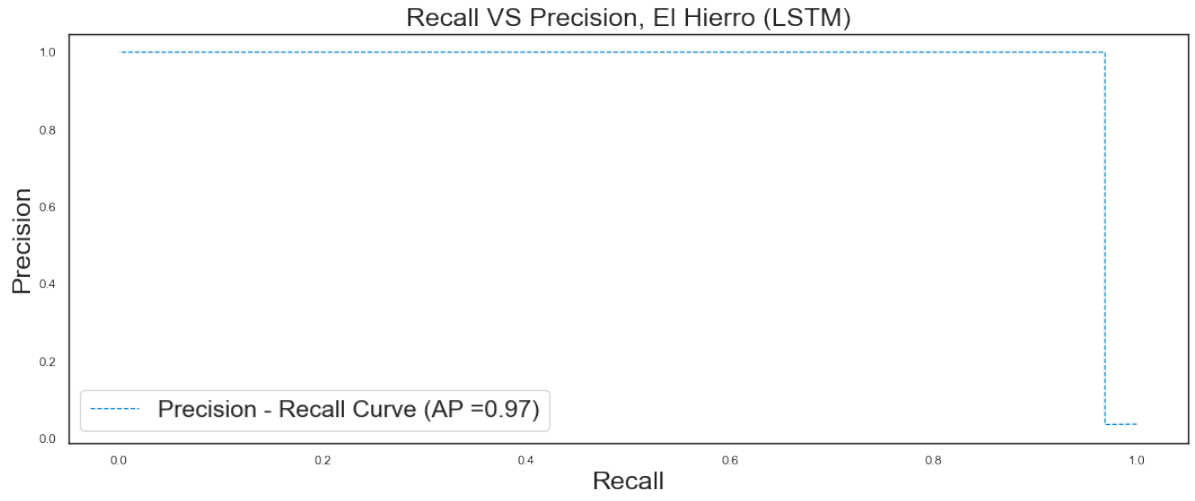


Figura A.52: Gráfico *Precision/Recall* para la isla de El Hierro con capas Long Short-Term Memory (LSTM).

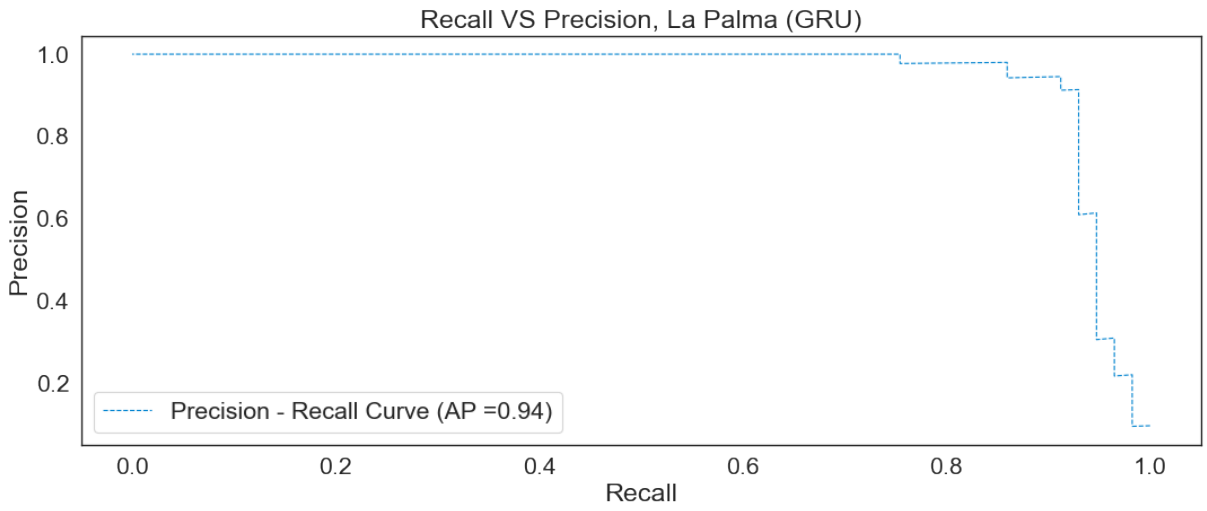


Figura A.53: Gráfico *Precision/Recall* para la isla de La Palma con capas Gated Recurrent Unit (GRU).

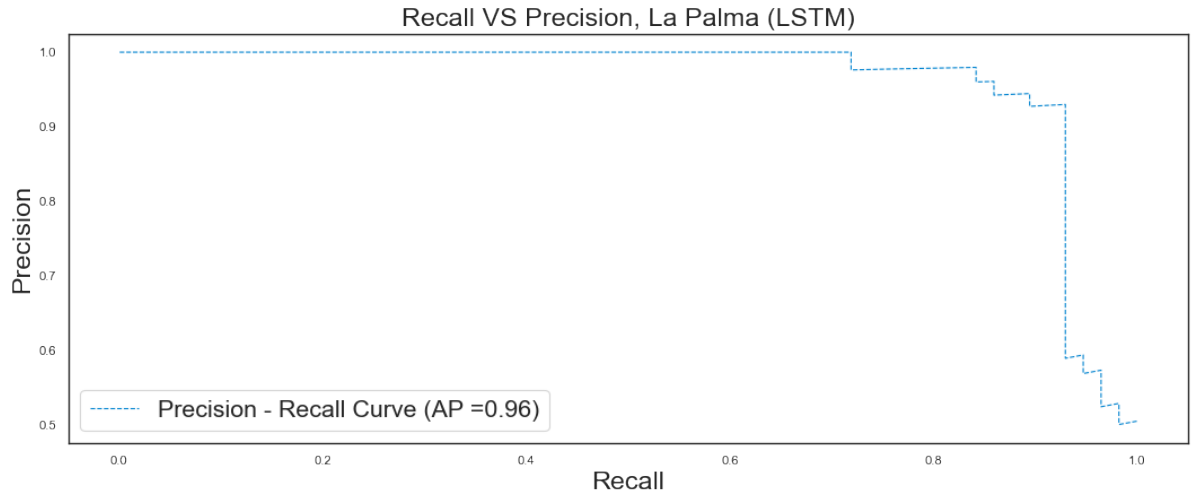


Figura A.54: Gráfico *Precision/Recall* para la isla de La Palma con capas Long Short-Term Memory (LSTM).

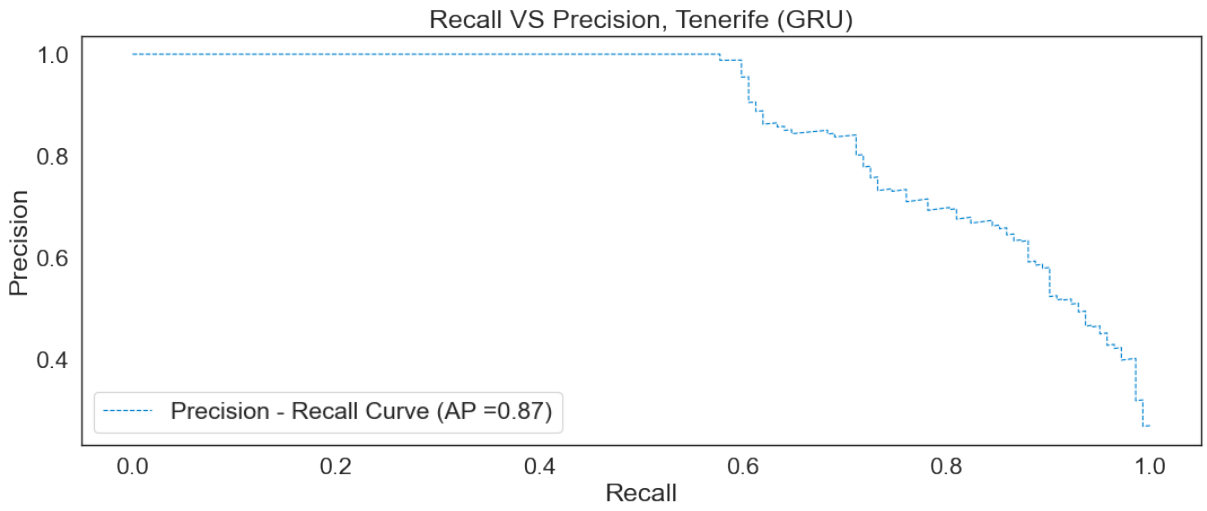


Figura A.55: Gráfico *Precision/Recall* para la isla de Tenerife con capas Gated Recurrent Unit (GRU).

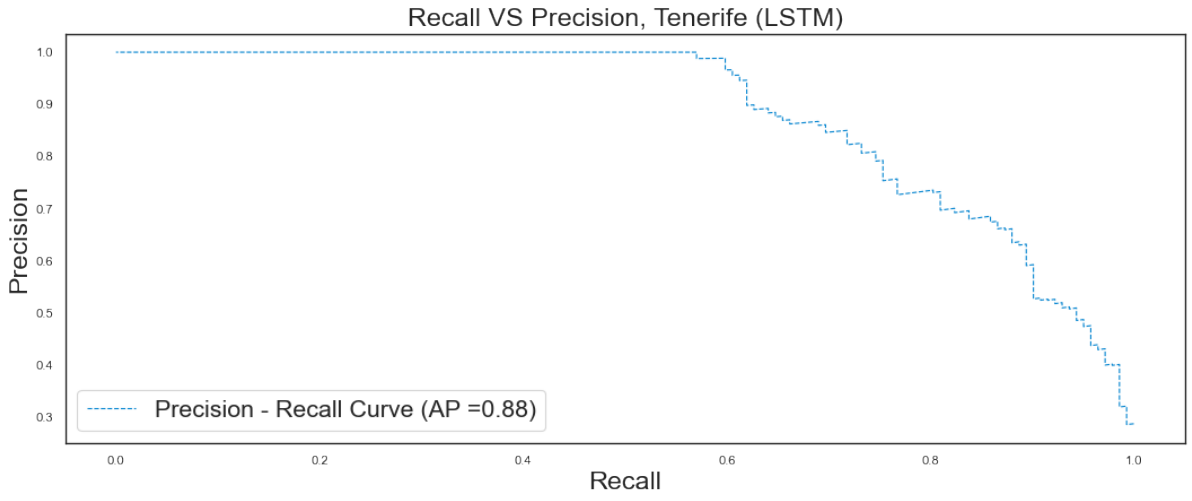


Figura A.56: Gráfico *Precision/Recall* para la isla de Tenerife con capas Long Short-Term Memory (LSTM).

A.5. Gráficas simples *Index Point*

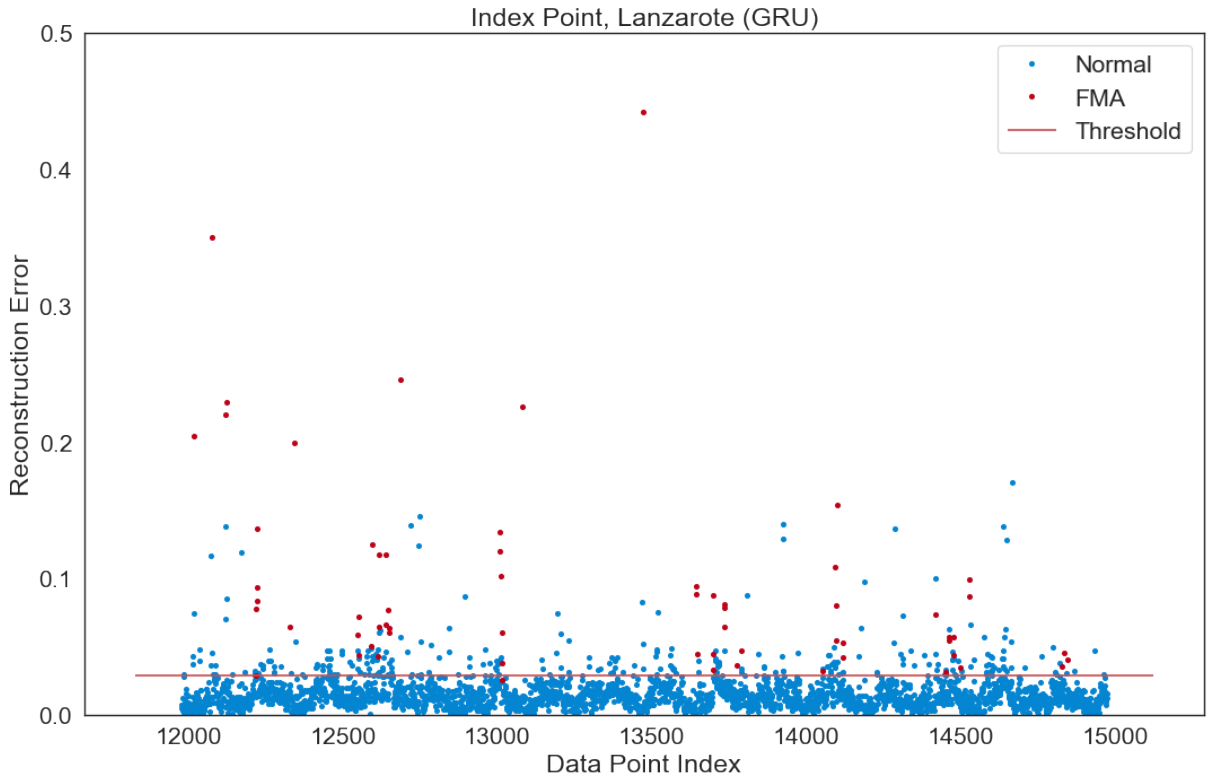


Figura A.57: Gráfico *Index Point* para la isla de Lanzarote con capas Gated Recurrent Unit (GRU).

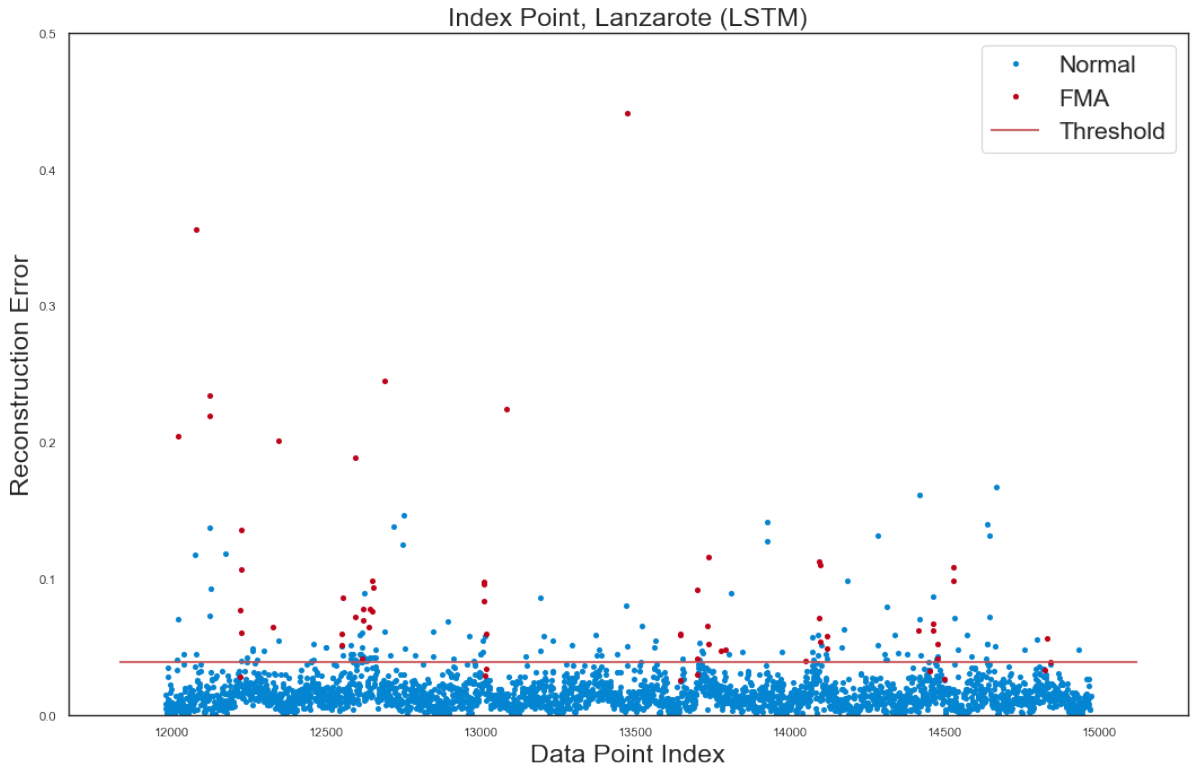


Figura A.58: Gráfico *Index Point* para la isla de Lanzarote con capas Long Short-Term Memory (LSTM).

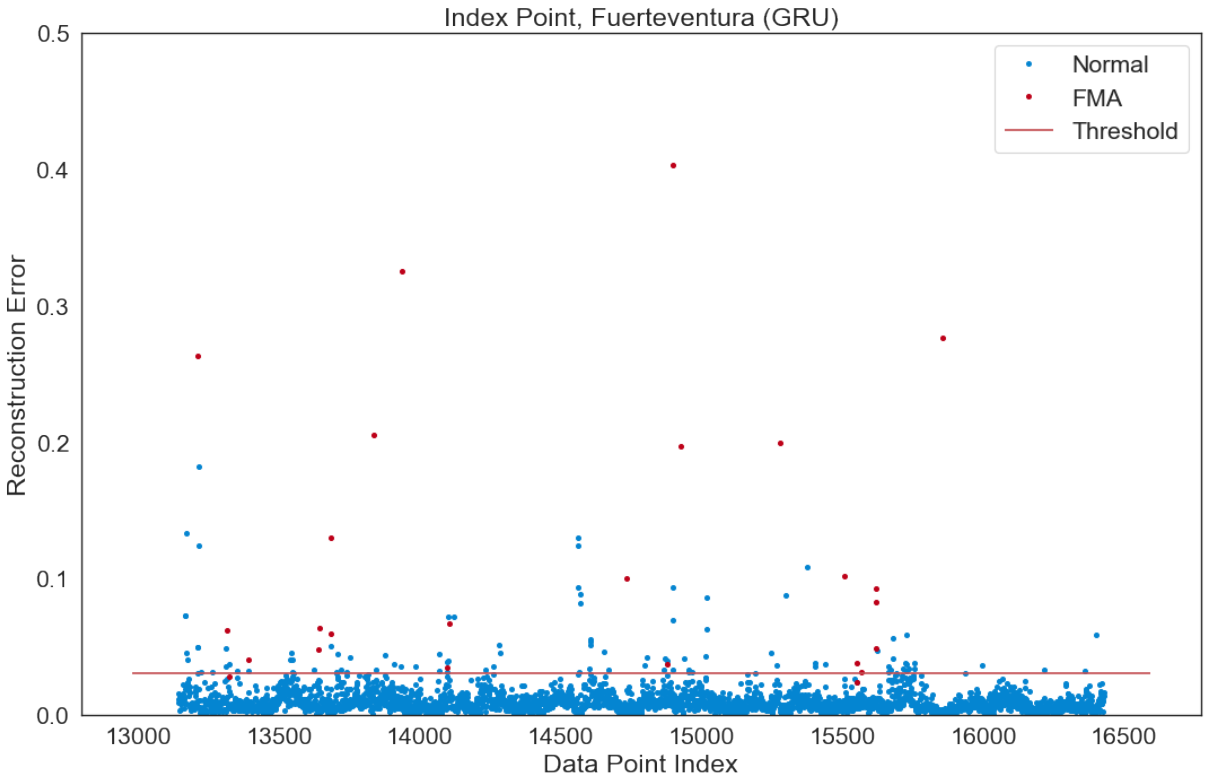


Figura A.59: Gráfico *Index Point* para la isla de Fuerteventura con capas Gated Recurrent Unit (GRU).

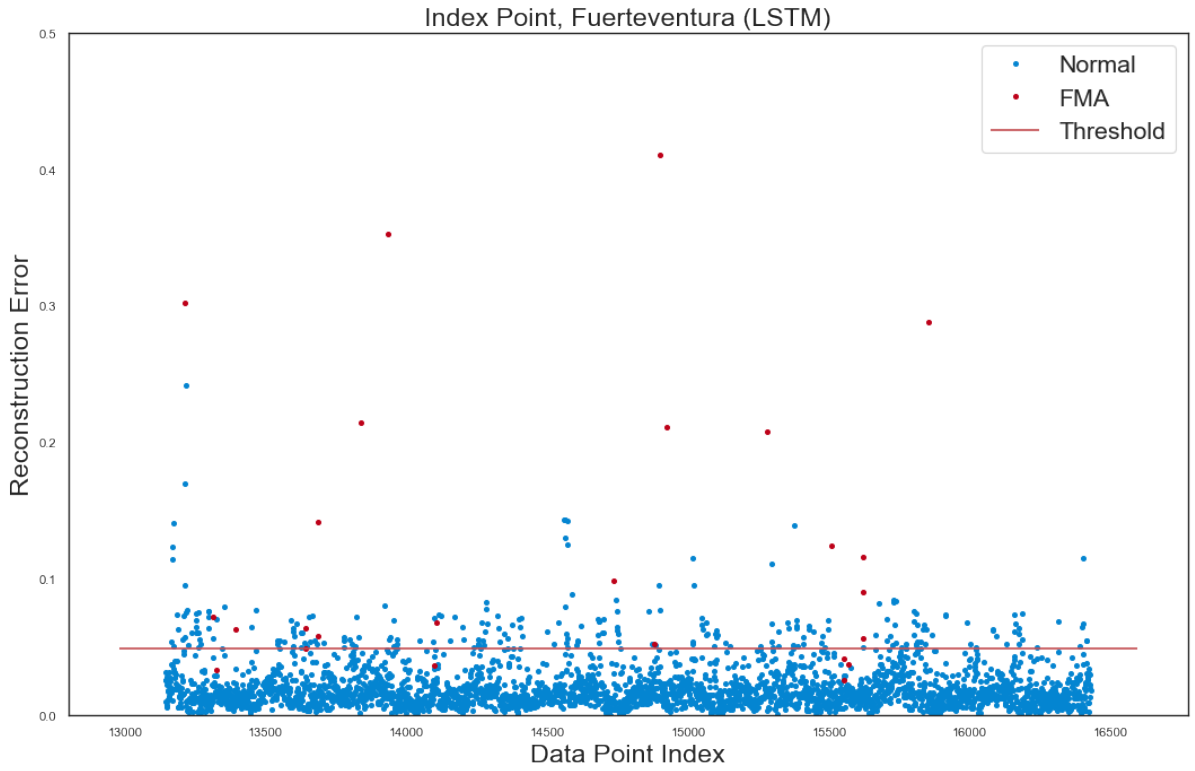


Figura A.60: Gráfico *Index Point* para la isla de Fuerteventura con capas Long Short-Term Memory (LSTM).

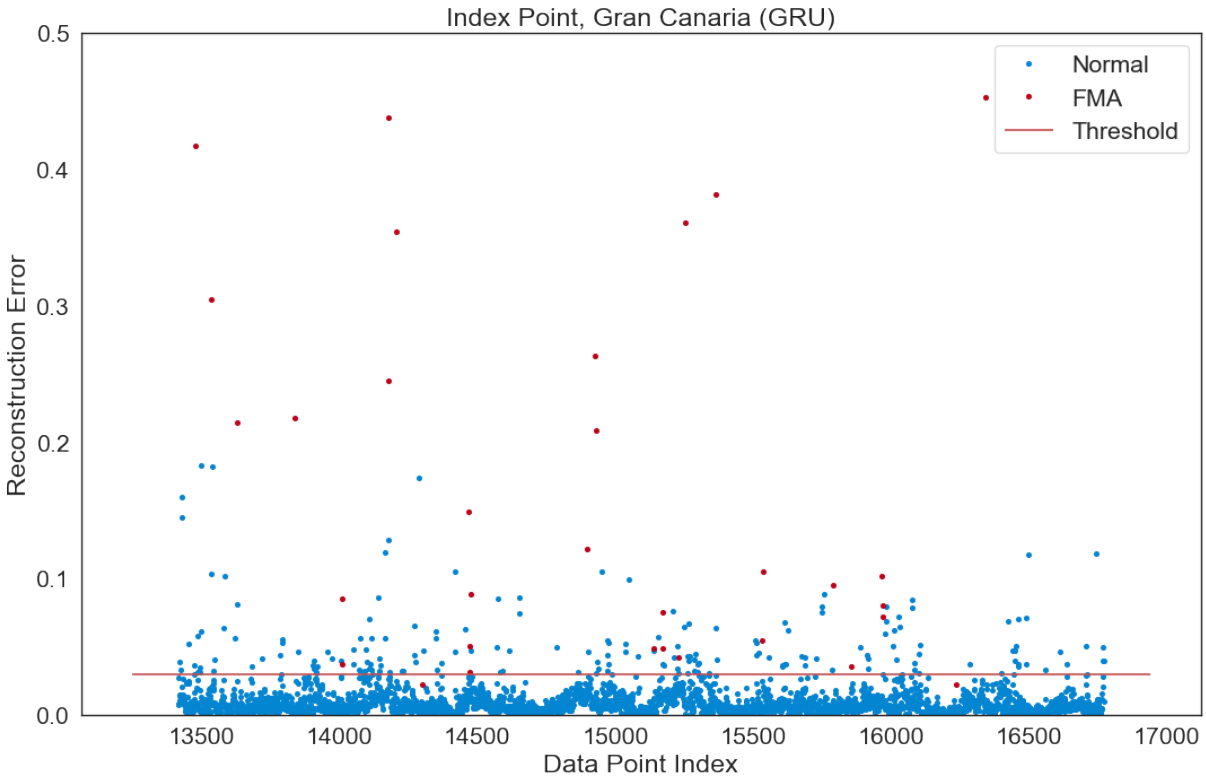


Figura A.61: Gráfico *Index Point* para la isla de Gran Canaria con capas Gated Recurrent Unit (GRU).

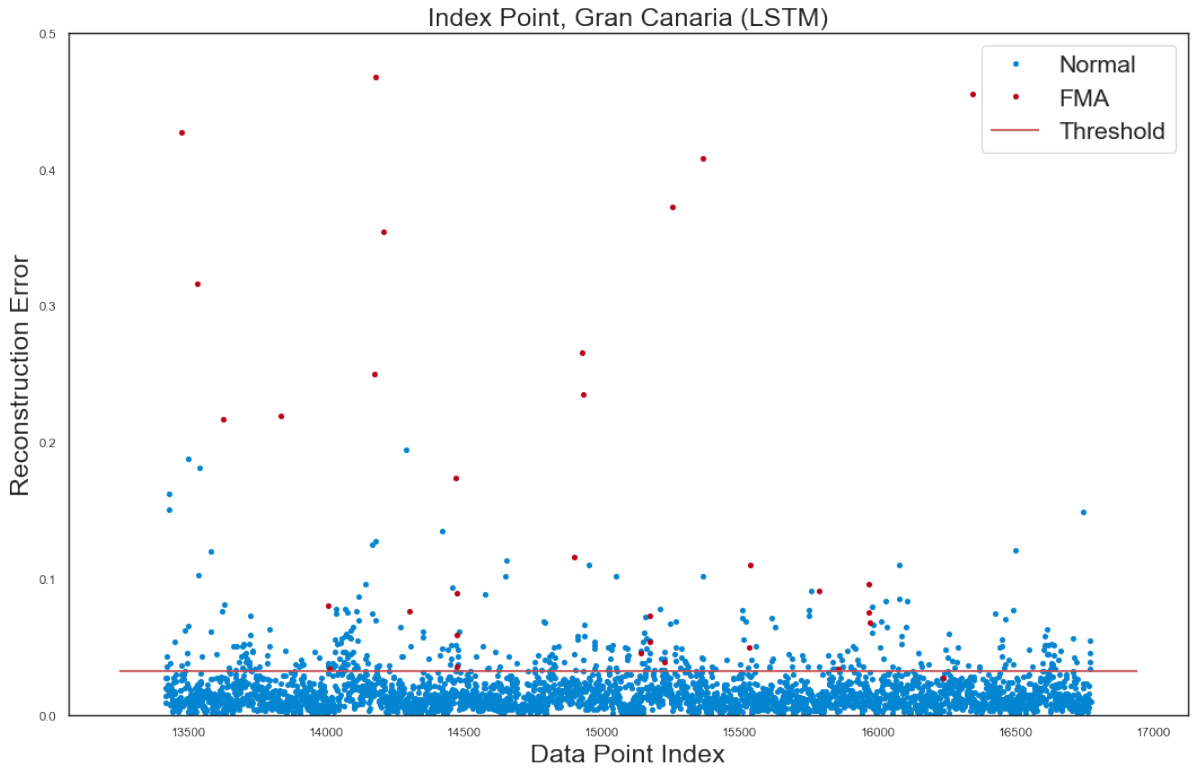


Figura A.62: Gráfico *Index Point* para la isla de Gran Canaria con capas Long Short-Term Memory (LSTM).

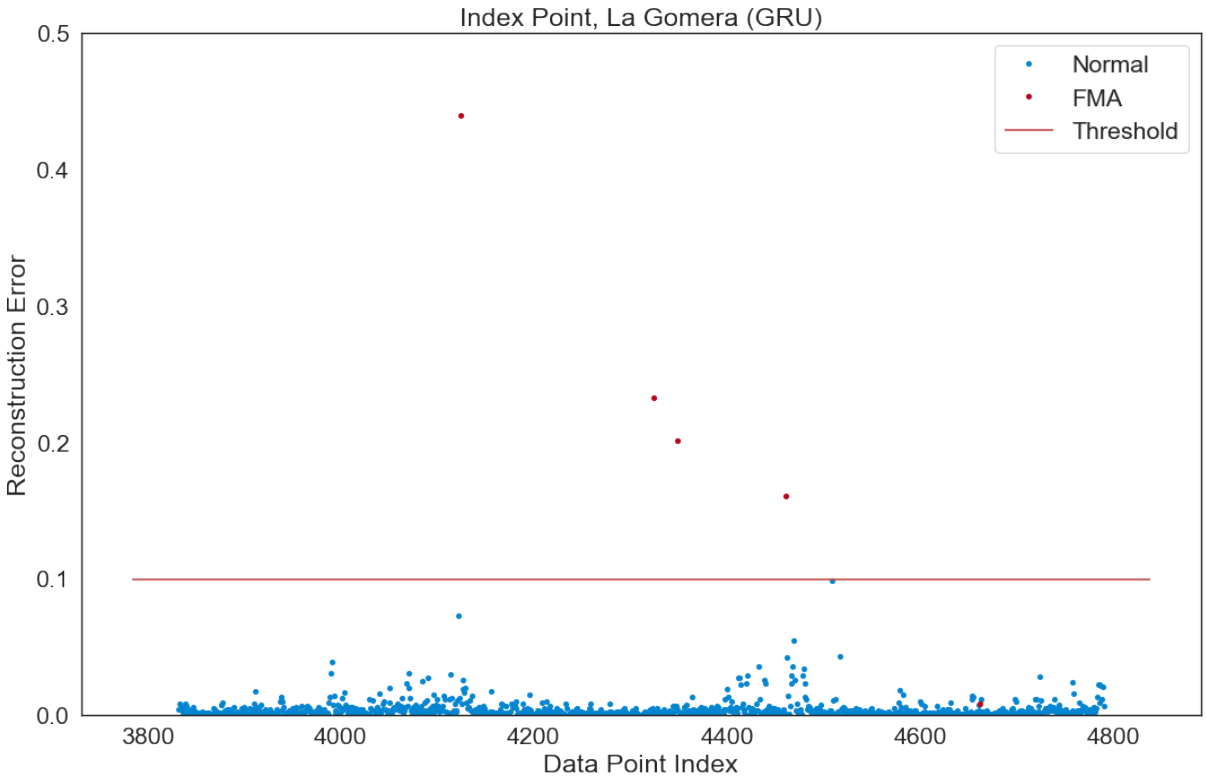


Figura A.63: Gráfico *Index Point* para la isla de La Gomera con capas Gated Recurrent Unit (GRU).

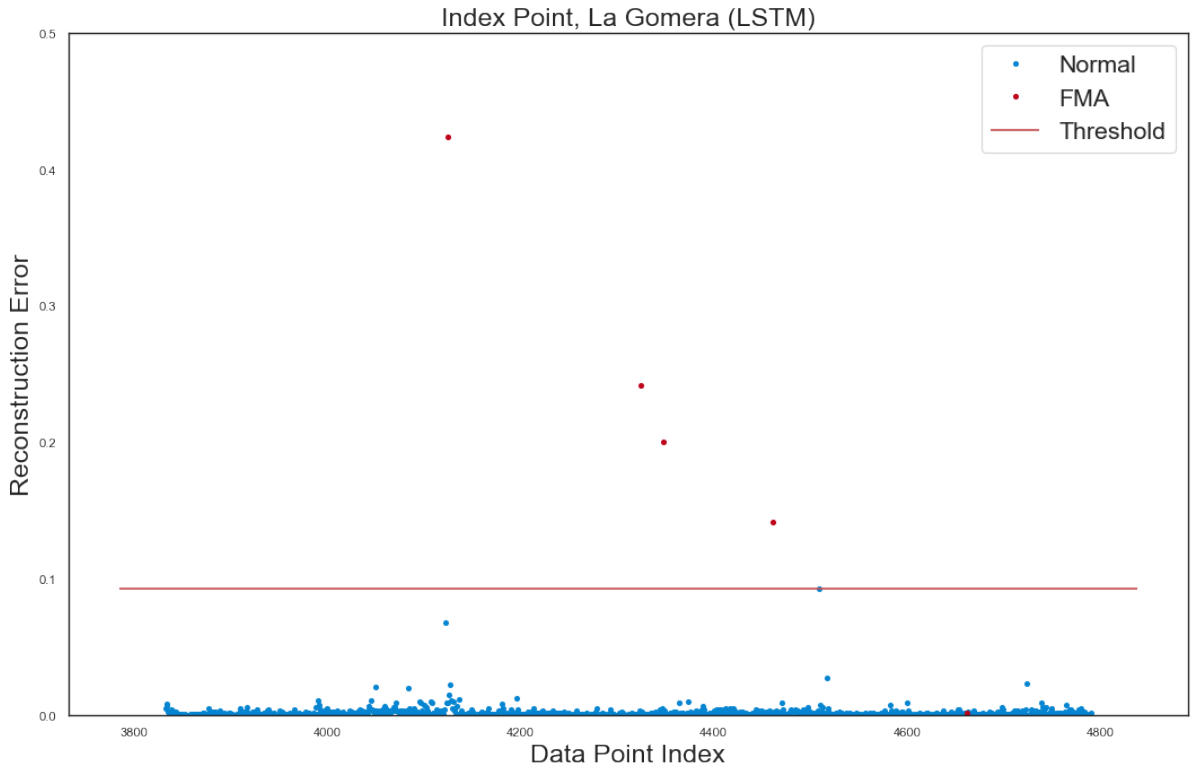


Figura A.64: Gráfico *Index Point* para la isla de La Gomera con capas Long Short-Term Memory (LSTM).

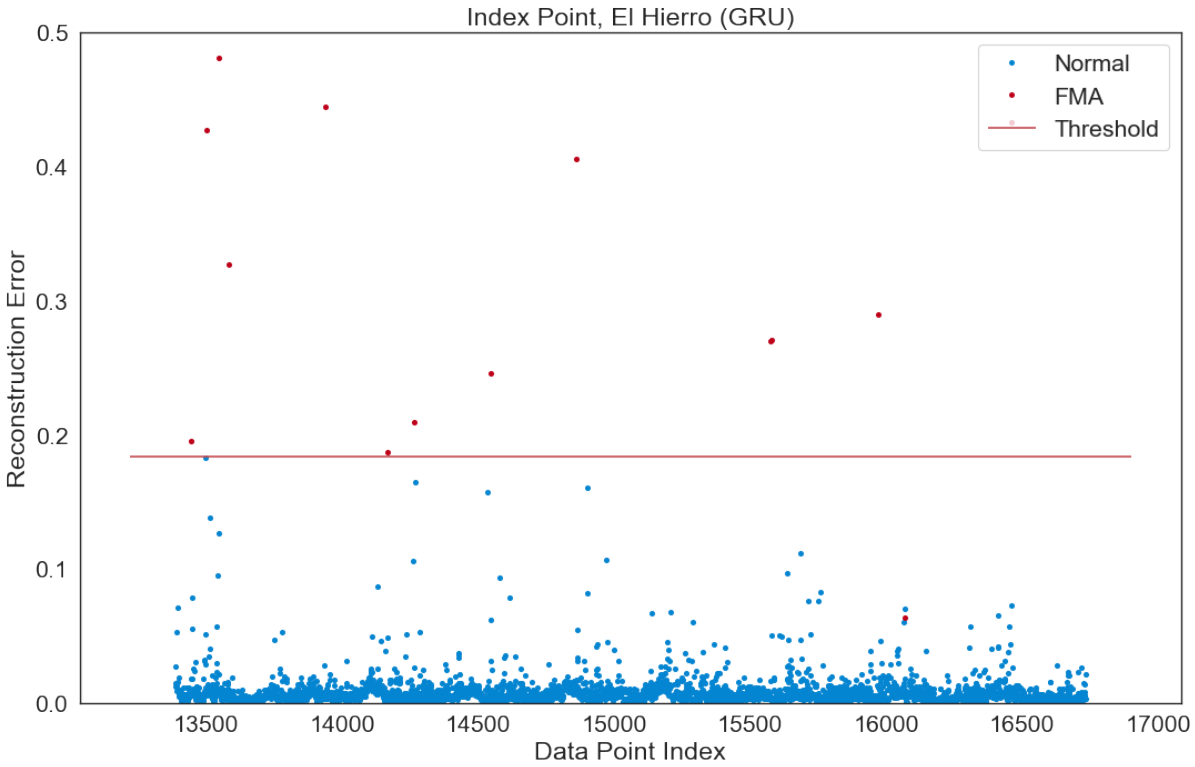


Figura A.65: Gráfico *Index Point* para la isla de El Hierro con capas Gated Recurrent Unit (GRU).

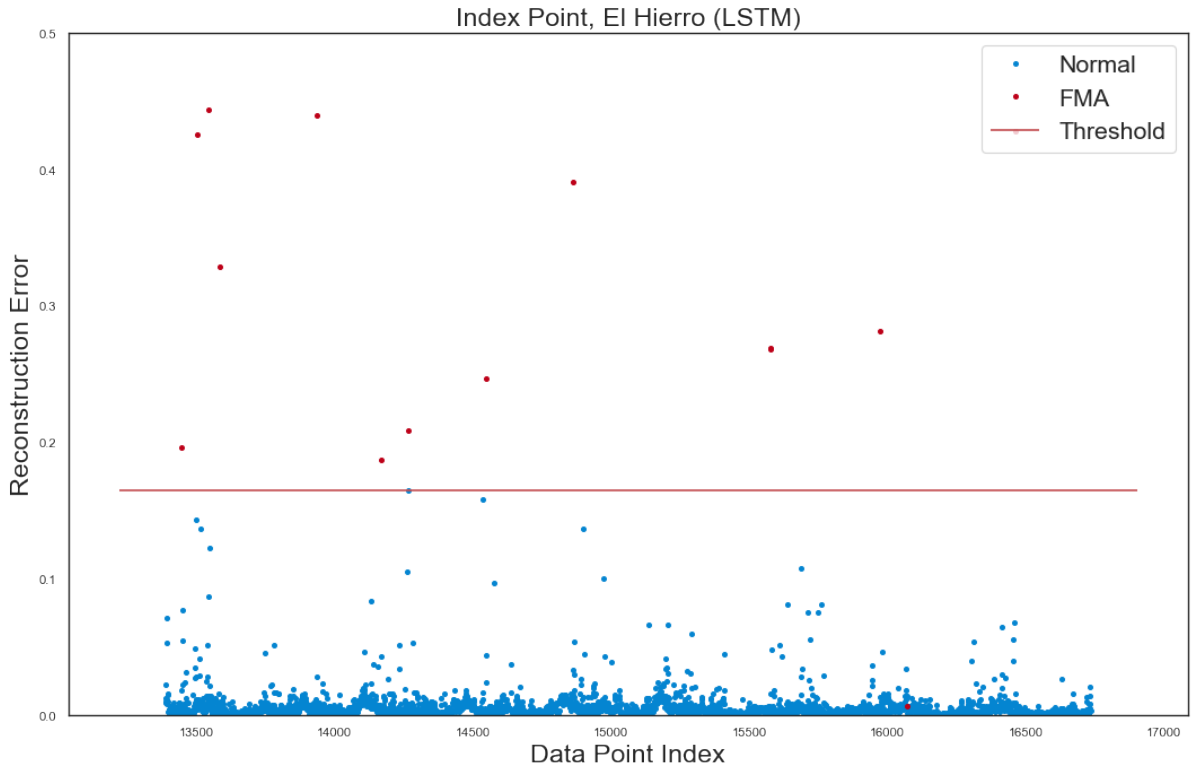


Figura A.66: Gráfico *Index Point* para la isla de El Hierro con capas Long Short-Term Memory (LSTM).

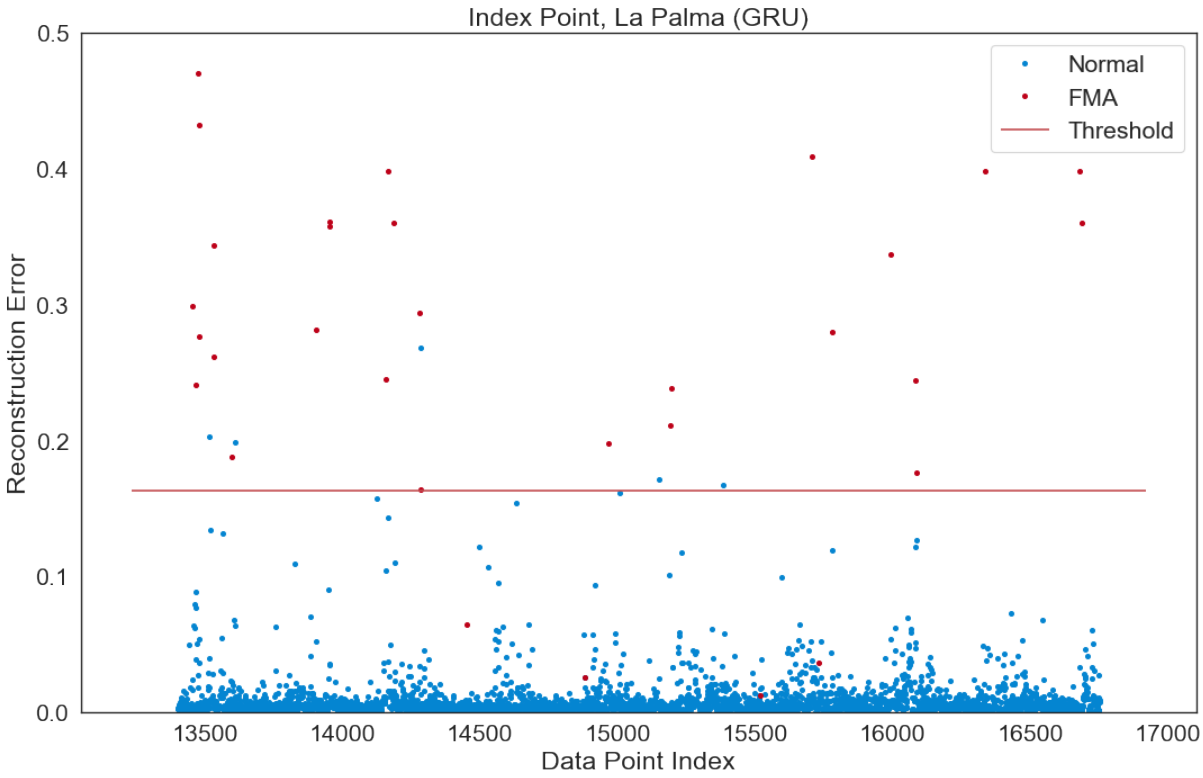


Figura A.67: Gráfico *Index Point* para la isla de La Palma con capas Gated Recurrent Unit (GRU).

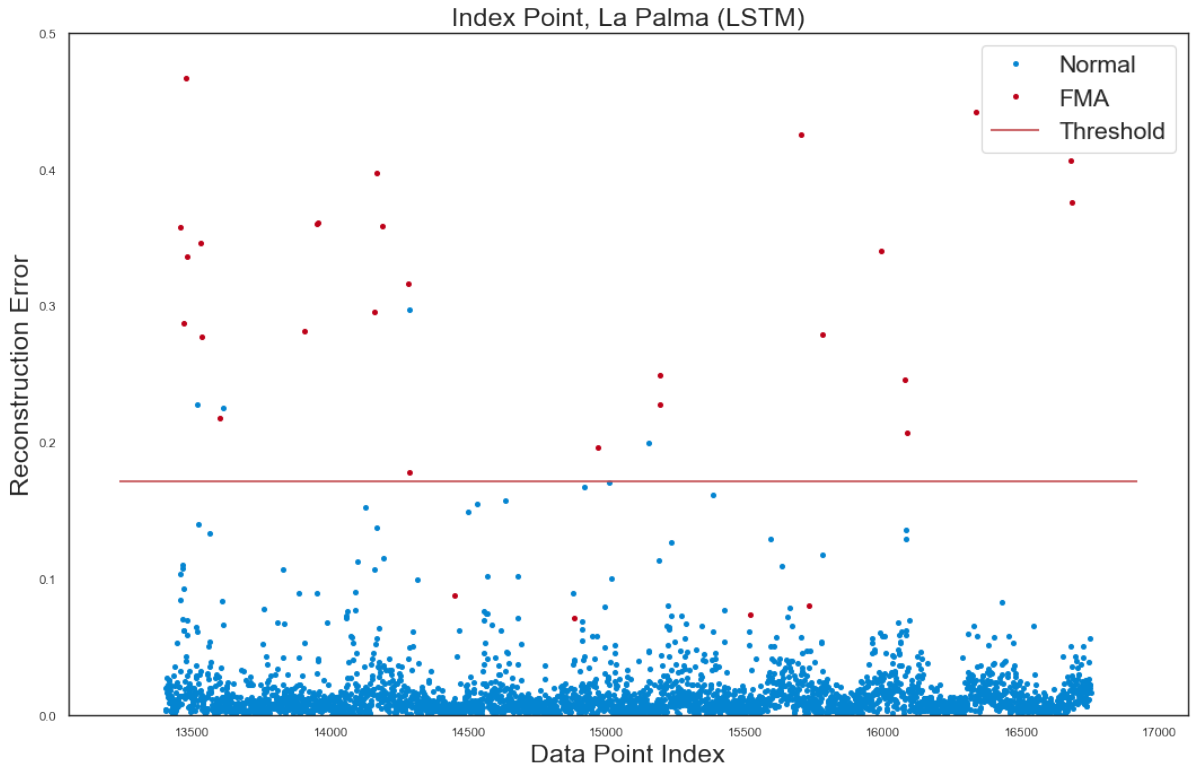


Figura A.68: Gráfico *Index Point* para la isla de La Palma con capas Long Short-Term Memory (LSTM).

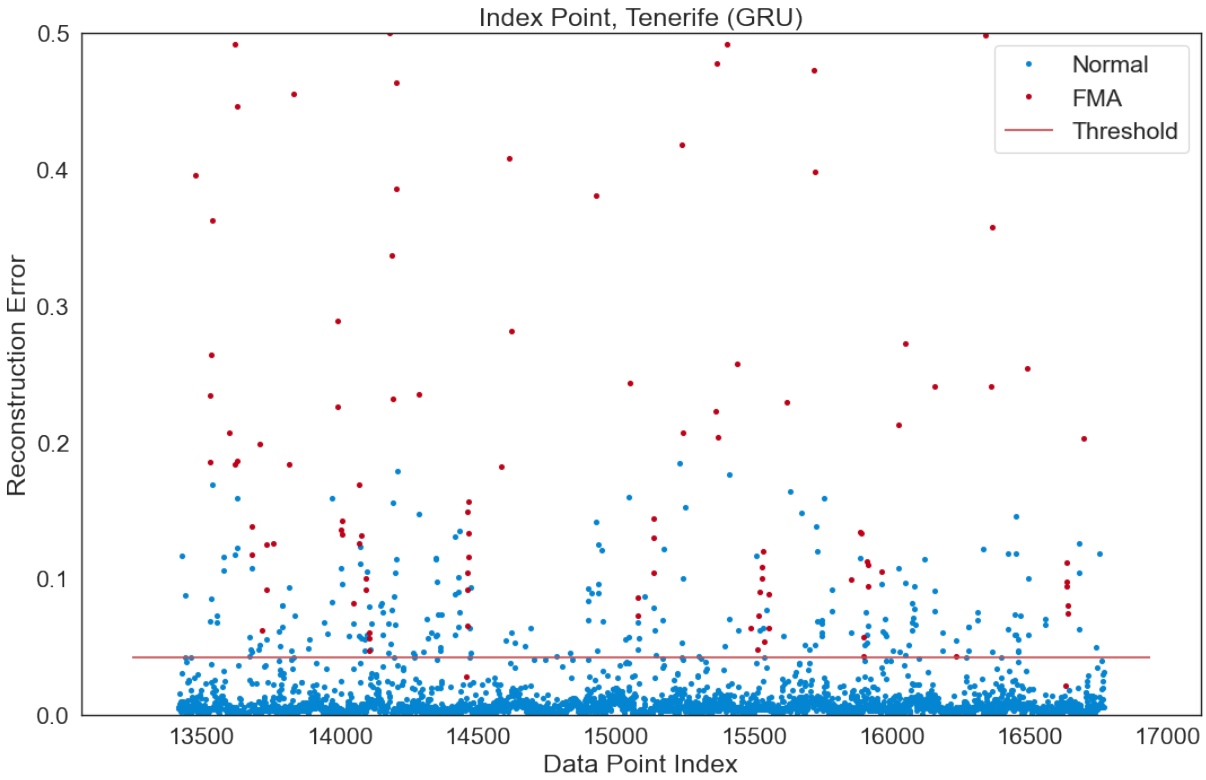


Figura A.69: Gráfico *Index Point* para la isla de Tenerife con capas Gated Recurrent Unit (GRU).

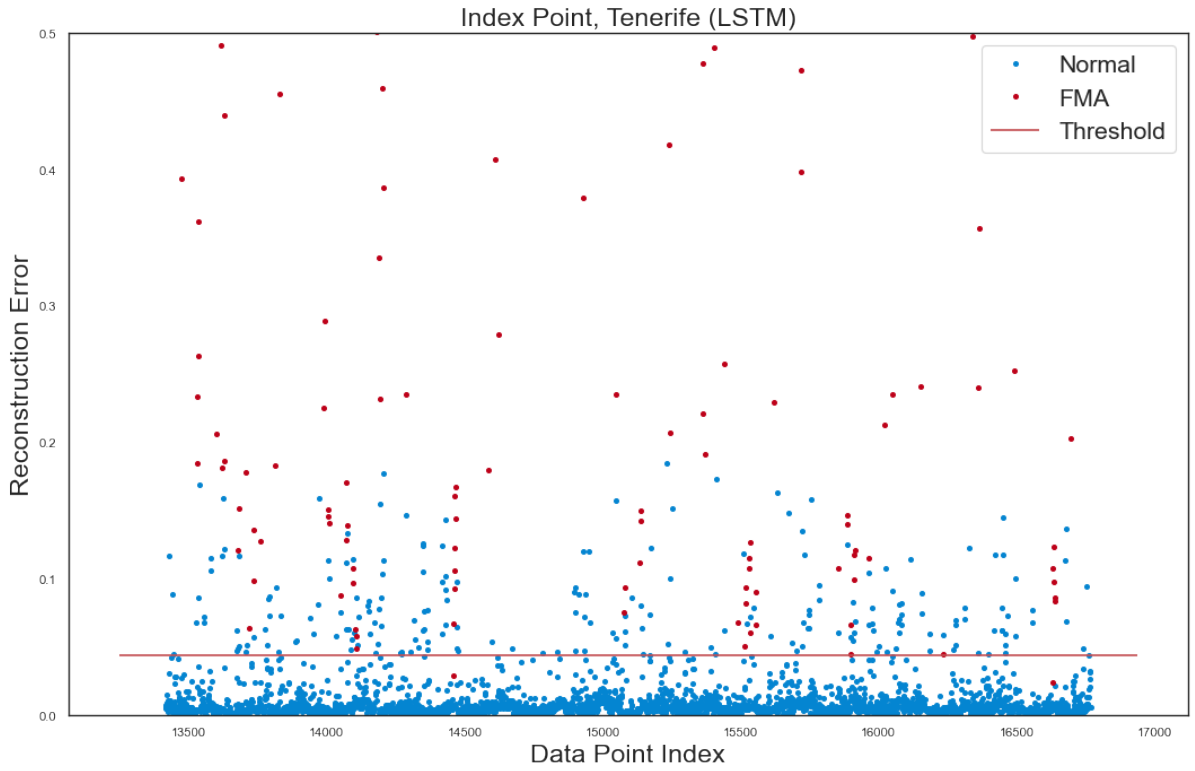


Figura A.70: Gráfico *Index Point* para la isla de Tenerife con capas Long Short-Term Memory (LSTM).

Apéndice B

Gráficas unificadas de modelos propuestos

B.1. Gráficas unificadas *Model Loss*

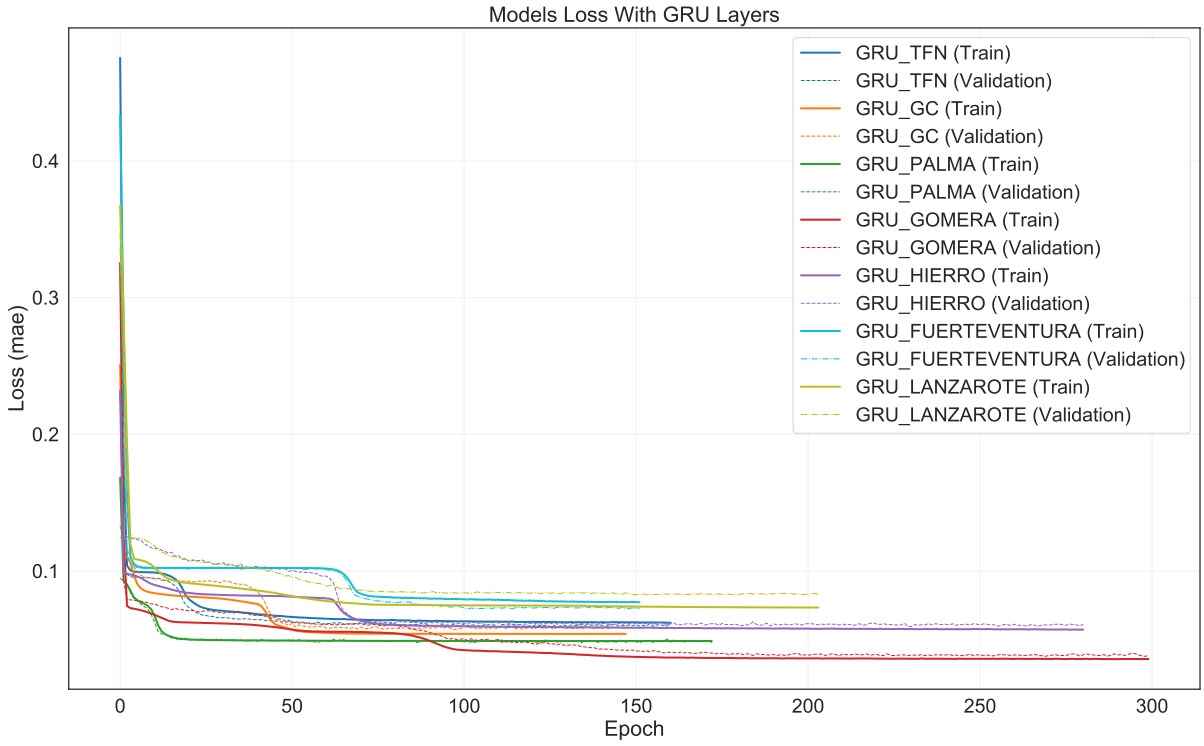


Figura B.1: Gráfico *Model Loss* de todos los modelos con capas Gated Recurrent Unit (GRU).

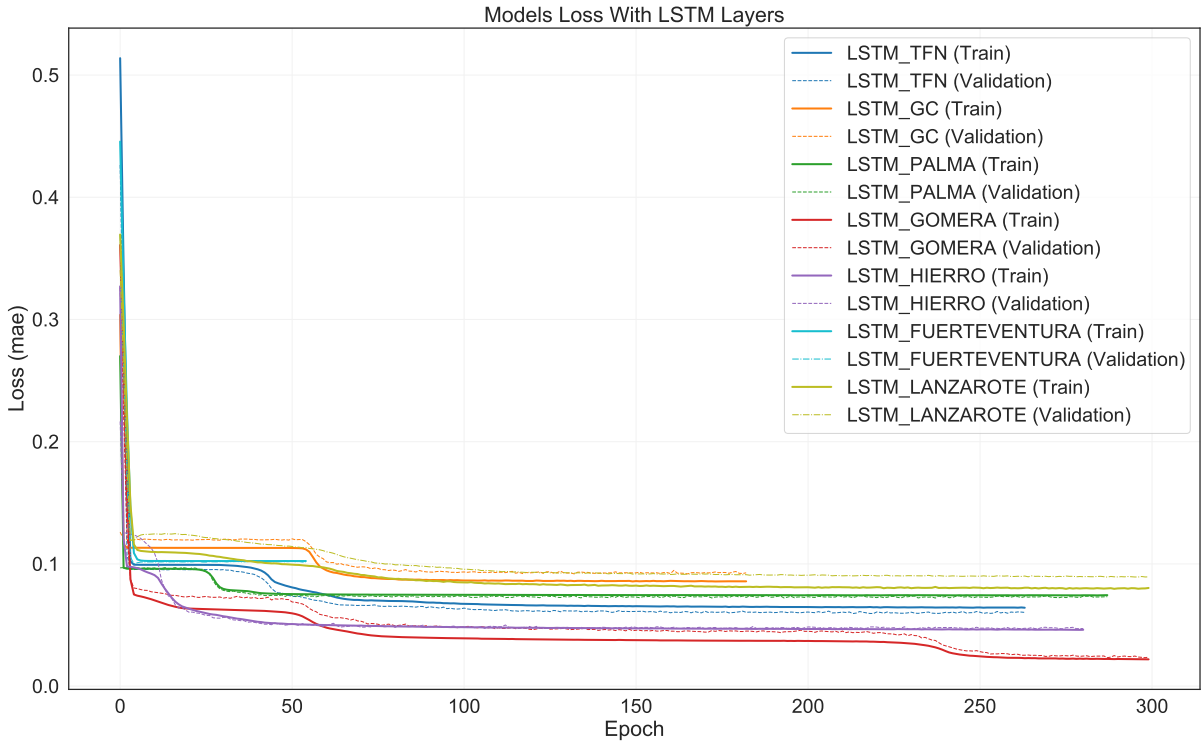


Figura B.2: Gráfico *Model Loss* de todos los modelos con capas Long Short-Term Memory (LSTM).

B.2. Gráficas unificadas *Precision/Recall*

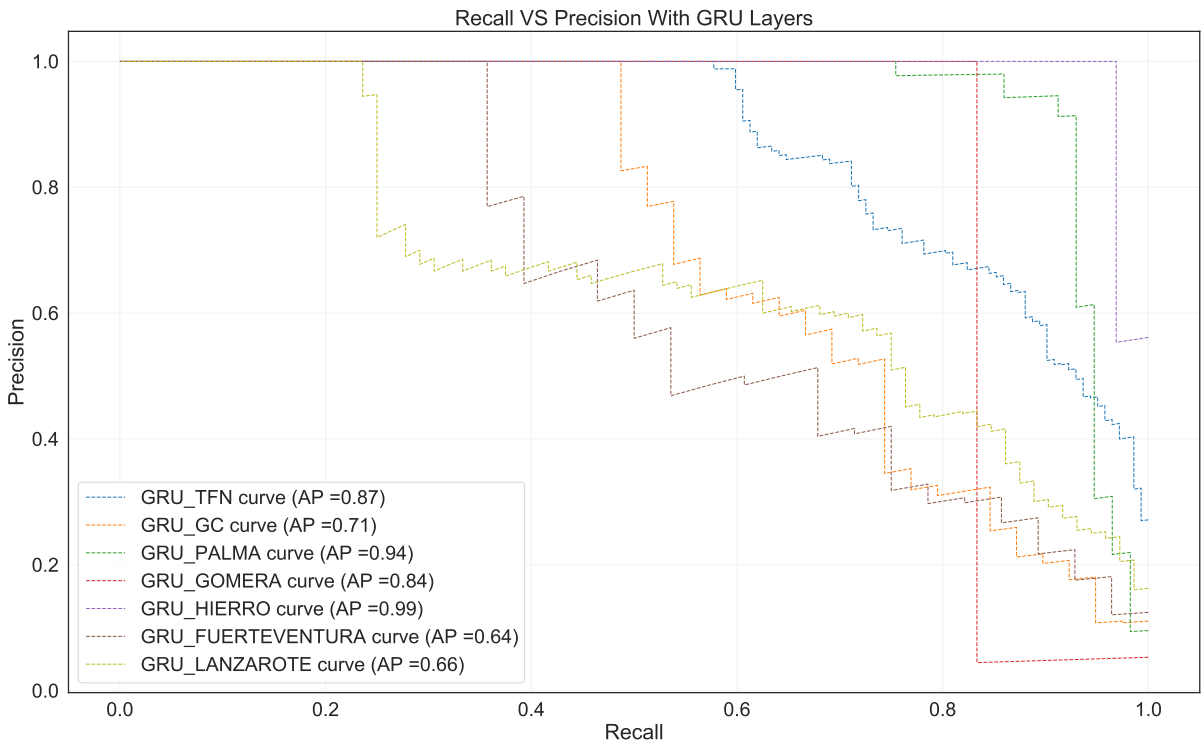


Figura B.3: Gráfico *Precision/Recall* de todos los modelos con capas Gated Recurrent Unit (GRU).

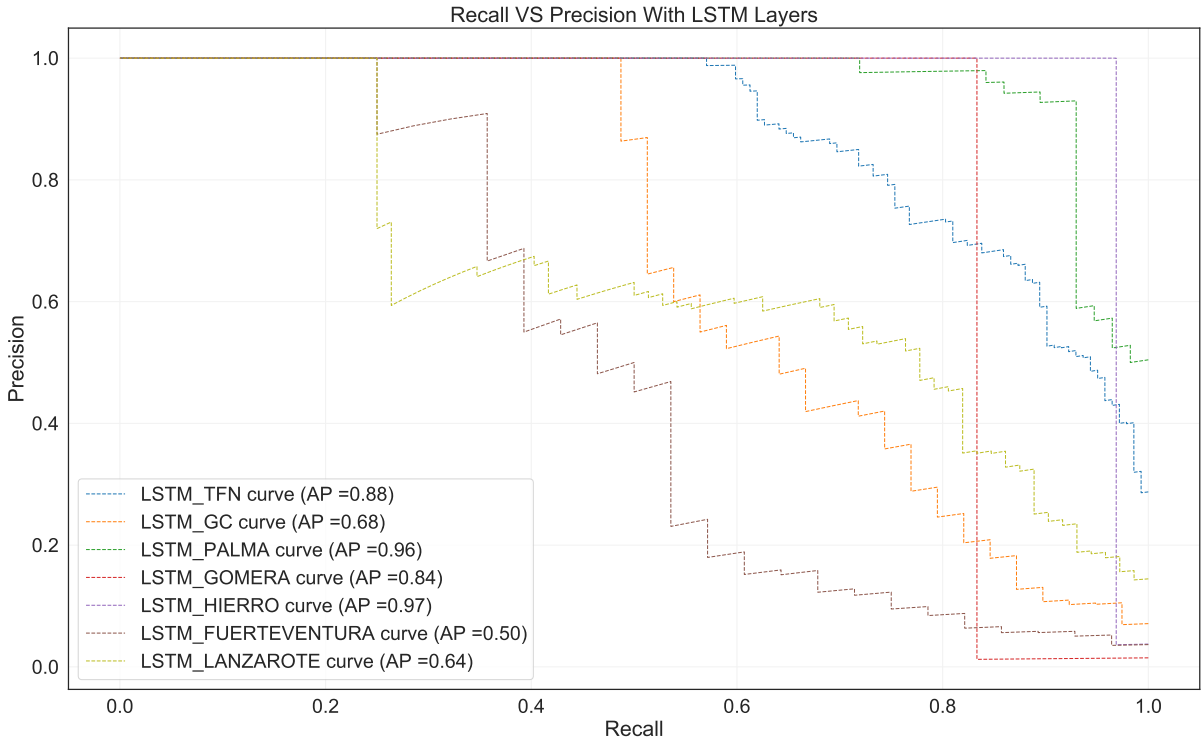


Figura B.4: Gráfico *Precision/Recall* de todos los modelos con capas Long Short-Term Memory (LSTM).

B.3. Gráficas unificadas *Receiver Operating Characteristic Curve (ROC)*

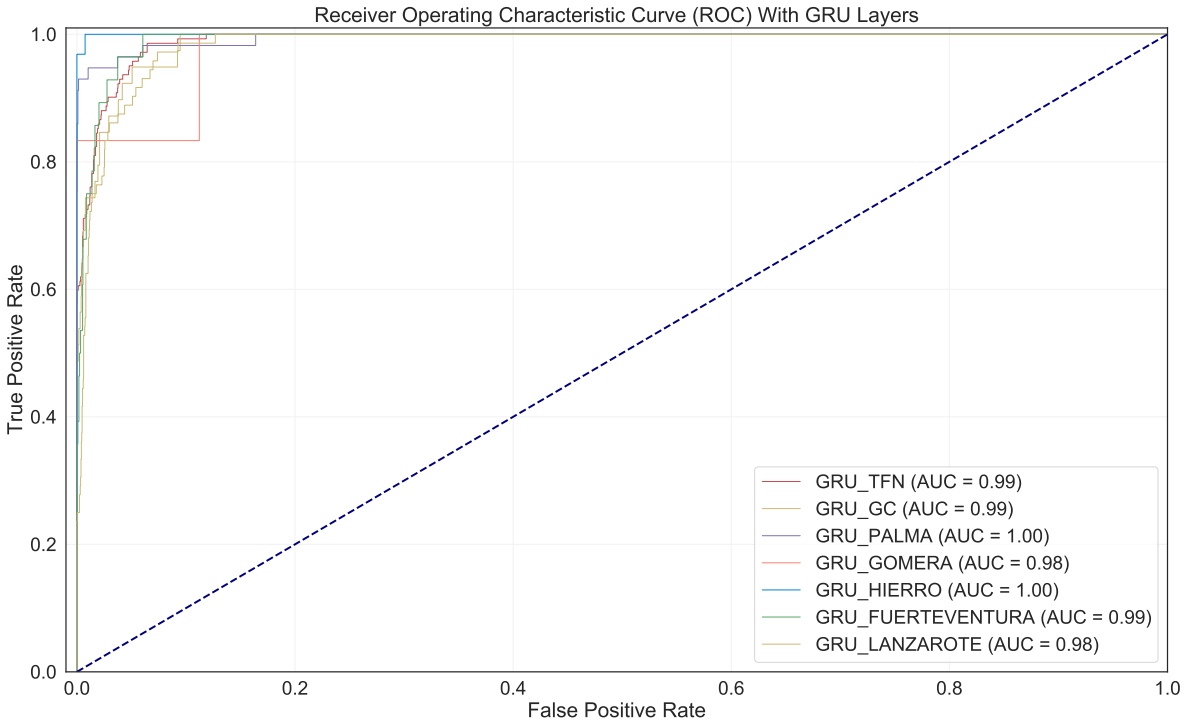


Figura B.5: Gráfico *Receiver Operating Characteristic Curve (ROC)* de todos los modelos con capas Gated Recurrent Unit (GRU).

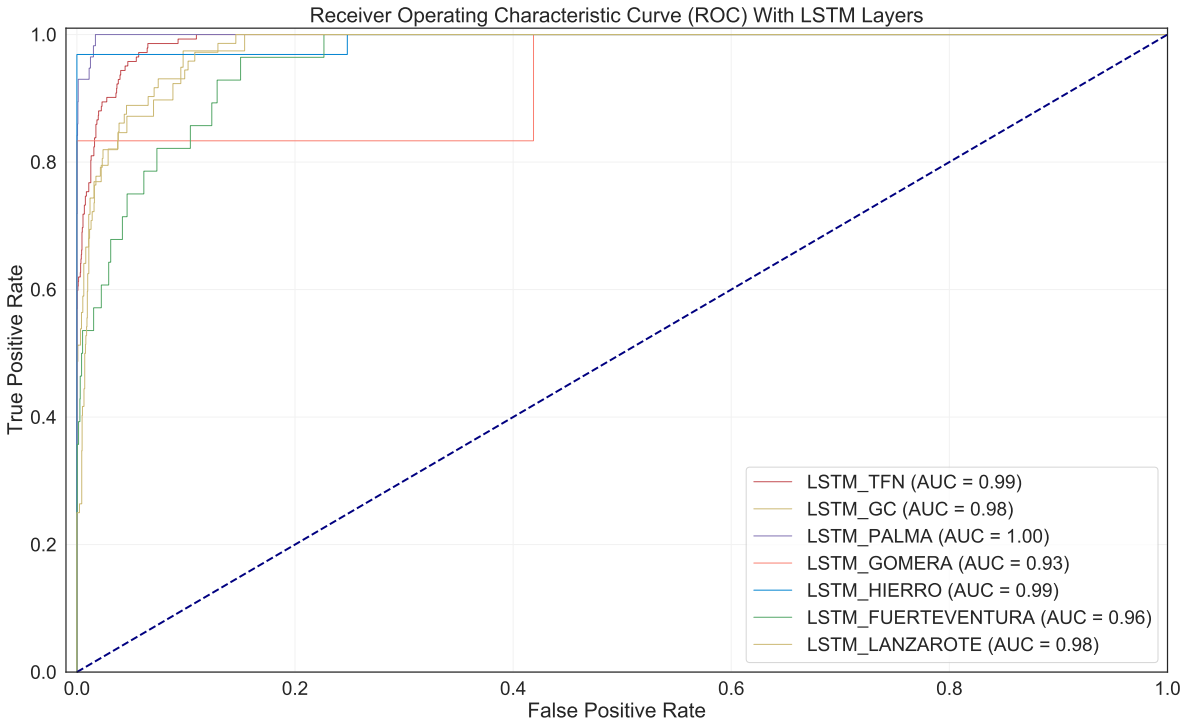


Figura B.6: Gráfico *Receiver Operating Characteristic Curve (ROC)* de todos los modelos con capas Long Short-Term Memory (LSTM).

B.4. Gráficas unificadas *Precision/Recall VS Threshold*

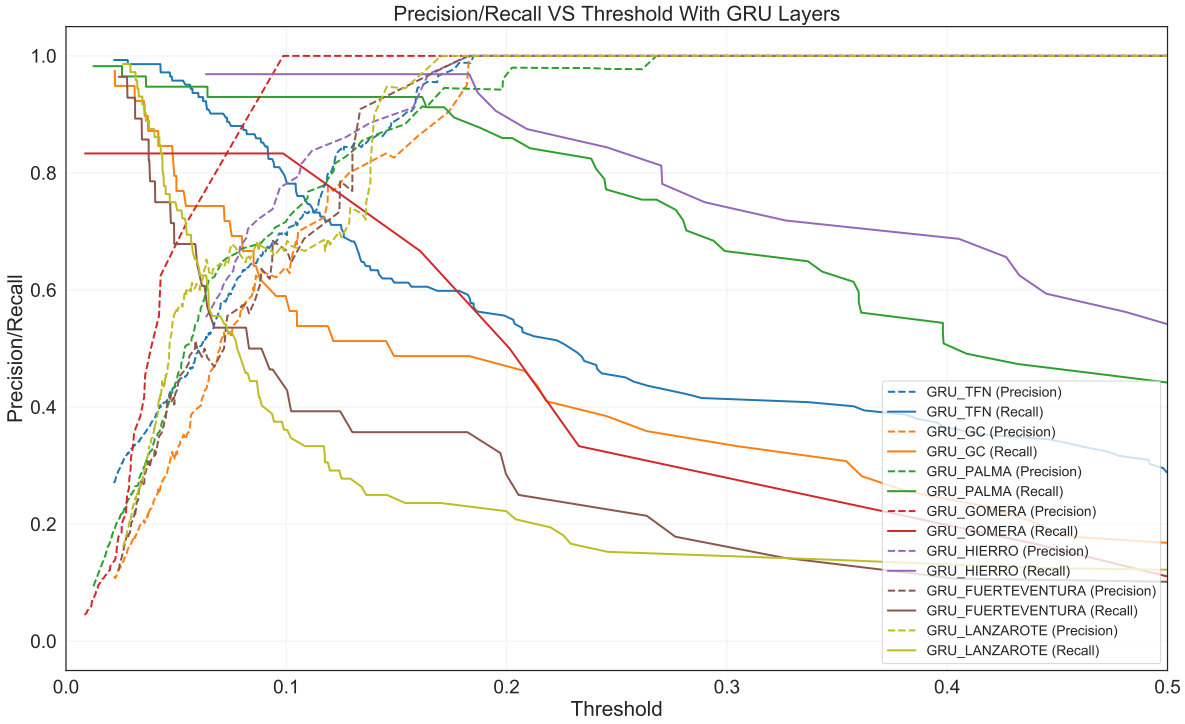


Figura B.7: Gráfico *Precision/Recall VS Threshold* de todos los modelos con capas Gated Recurrent Unit (GRU).

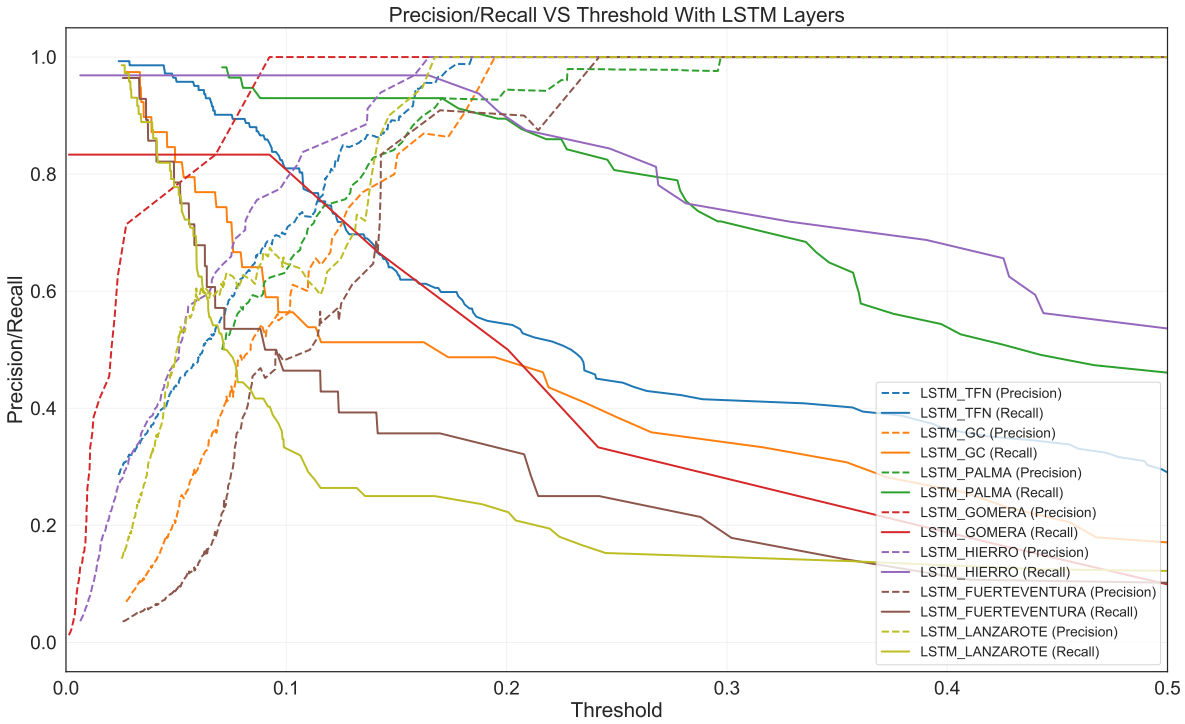


Figura B.8: Gráfico *Precision/Recall VS Threshold* de todos los modelos con capas Long Short-Term Memory (LSTM).

Apéndice C

Cuaderno de El Hierro con red GRU

Modelo_Hierro

June 24, 2021

0.1 Import Libraries

[1]:
```
import os
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
import time
import joblib
import logging
import math
import matplotlib.pyplot as plt
import pickle
%matplotlib inline

from pandas import DataFrame
from sklearn.preprocessing import MinMaxScaler
from numpy.random import seed
from keras.callbacks import EarlyStopping
from keras.layers import Input, Dropout, Dense, LSTM, GRU, RNN,␣
↪TimeDistributed, RepeatVector
from keras.models import Model
from keras import regularizers
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import confusion_matrix, precision_recall_curve,␣
↪cohen_kappa_score, fbeta_score
from sklearn.metrics import recall_score, classification_report, auc,␣
↪roc_curve, log_loss
from pylab import rcParams
```

[2]:
```
sns.set(color_codes = True)
col_list = ['cerulean', 'scarlet']
sns.set(style = 'white', font_scale = 1.75, palette = sns.
↪xkcd_palette(col_list))
rcParams['figure.figsize'] = 14, 8.7
```

[3]:
```
tf.random.set_seed(123)
seed(123)
```

[4]:
```
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

[5]:
```
LABELS = ['Normal', 'FMA']
```

0.2 GPU

[6]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs", len(logical_gpus), "Logical GPUs")
    except RuntimeError as error:
        print(error)
else:
    print('No GPUs detected')
```

```
1 Physical GPUs 1 Logical GPUs
```

0.3 Load Datas

[7]:
```
data = pd.read_csv('Dataset_Hierro_Labeled.csv', sep = ';')
```

[8]:
```
data
```

[8]:

	temp_max	temp_avg	temp_min	prec	wd	ws	atmos_pres_avg	\
0	17.0	16.60	15.0	0.0	165.00	6.1	1018.95	
1	19.0	18.80	18.0	0.0	18.00	5.1	1018.95	
2	20.0	19.43	18.0	0.0	168.33	7.7	1018.95	
3	20.0	19.00	17.0	0.0	174.00	4.1	1018.95	
4	21.0	20.50	19.0	0.0	145.14	0.0	1018.95	
…	…	…	…	…	…	…	…	
16730	24.0	21.08	17.0	0.0	198.89	7.2	1019.31	
16731	25.0	22.02	20.0	0.0	174.46	6.2	1019.19	
16732	24.0	21.70	19.0	0.0	180.37	7.7	1019.94	
16733	24.2	21.74	18.0	0.0	179.64	6.2	1023.14	
16734	23.0	19.68	17.7	0.0	204.81	6.7	1024.94	

	atmos_pres_min	atmos_pres_max	rh	ceil_hgt	visibility	FMA
0	1017.947023	1019.979342	77.44	5054.0	11200.00	0
1	1017.947023	1019.979342	63.06	13560.0	11200.00	0
2	1017.947023	1019.979342	70.86	22000.0	11200.00	0
3	1017.947023	1019.979342	70.45	22000.0	11200.00	0
4	1017.947023	1019.979342	77.88	22000.0	11200.00	0
…	…	…	…	…	…	…
16730	1018.400000	1020.100000	60.43	12737.0	11636.38	0

```
16731     1018.100000     1020.200000  52.43   22000.0    12454.90    0
16732     1018.900000     1021.500000  57.62   22000.0    13927.45    0
16733     1021.800000     1024.800000  48.05   22000.0    13469.19    0
16734     1024.000000     1026.100000  74.59   17728.0    12738.04    0

[16735 rows x 13 columns]
```

[9]:
```
train = data[0:math.trunc(len(data) * 0.8)]
test = data[math.trunc(len(data) * 0.8):len(data)]
```

[10]:
```
print("Dataset shape:", data.shape)
print("Train shape:", train.shape)
print("Test shape:", test.shape)
print("Train + Test entries:", train.shape[0] + test.shape[0])
```

```
Dataset shape: (16735, 13)
Train shape: (13388, 13)
Test shape: (3347, 13)
Train + Test entries: 16735
```

0.4 FMA Filter

[11]:
```
train.describe()
```

[11]:

	temp_max	temp_avg	temp_min	prec	wd	\
count	13388.000000	13388.000000	13388.000000	13388.000000	13388.000000	
mean	22.706507	21.277218	19.432035	0.936846	133.976209	
std	2.489230	2.270146	2.453652	11.404823	104.289585	
min	12.800000	12.800000	-8.000000	0.000000	10.000000	
25%	21.000000	19.380000	18.000000	0.000000	37.690000	
50%	23.000000	21.180000	19.000000	0.000000	106.670000	
75%	24.600000	23.100000	21.000000	0.000000	202.310000	
max	40.000000	32.780000	28.400000	538.000000	360.000000	

	ws	atmos_pres_avg	atmos_pres_min	atmos_pres_max	\
count	13388.000000	13388.000000	13388.000000	13388.000000	
mean	8.611766	1018.860937	1017.896474	1019.814573	
std	2.980291	3.620019	3.795174	3.766826	
min	0.000000	974.700000	929.600000	997.100000	
25%	6.700000	1016.900000	1015.900000	1017.800000	
50%	8.200000	1018.950000	1017.947023	1019.800000	
75%	10.300000	1020.830000	1020.000000	1021.800000	
max	46.800000	1035.050000	1033.000000	1060.000000	

	rh	ceil_hgt	visibility	FMA
count	13388.000000	13388.000000	13388.000000	13388.000000
mean	72.674537	16570.565804	12481.929626	0.015536
std	7.935268	6323.007601	3053.236602	0.123677

```
min       25.840000    200.000000    800.000000      0.000000
25%       67.750000  12555.560000  10600.000000      0.000000
50%       72.830000  18964.290000  11055.855000      0.000000
75%       77.980000  22000.000000  13940.000000      0.000000
max       97.600000  22000.000000  30000.000000      1.000000
```

[12]:
```
train = train[train['FMA'] == 0]
```

[13]:
```
train.shape
```

[13]:
```
(13180, 13)
```

[14]:
```
split = math.trunc(train.shape[0] * 0.2)
validation = train[-split:]
train = train[:-split]
```

[15]:
```
print("Validation shape:", validation.shape)
print("Train shape:", train.shape)
```

```
Validation shape: (2636, 13)
Train shape: (10544, 13)
```

[16]:
```
train_fma = train.iloc[:, -1]
validation_fma = validation.iloc[:, -1]
test_fma = test.iloc[:, -1]
```

[17]:
```
train_fma.columns = ['train_fma']
validation_fma.columns = ['validation_fma']
test_fma.columns = ['test_fma']
```

[18]:
```
test_fma.head()
```

[18]:
```
13388    0
13389    0
13390    0
13391    0
13392    0
Name: FMA, dtype: int64
```

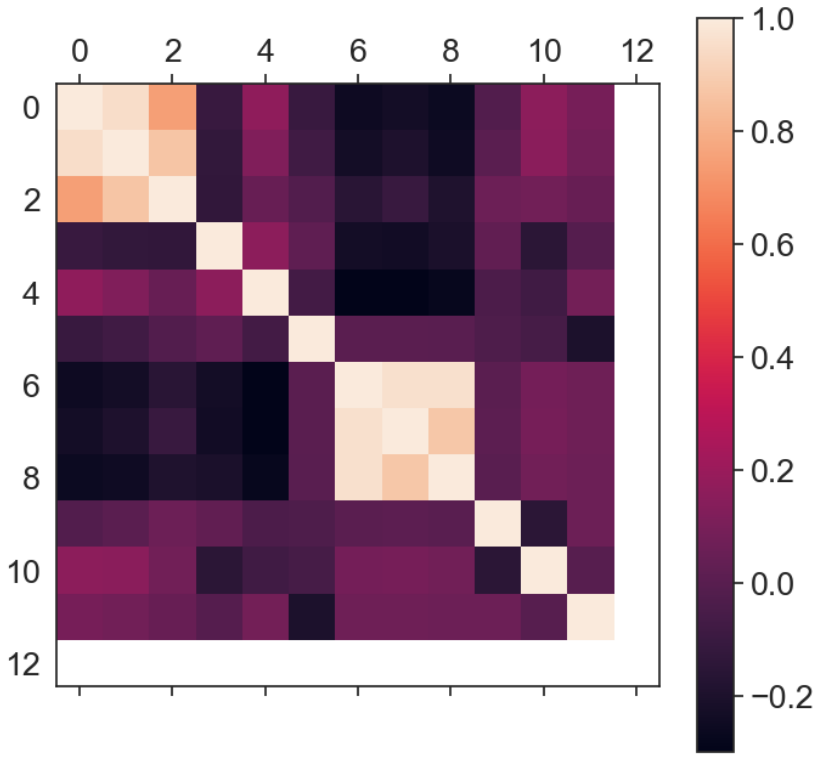
0.5 Correlation

[19]:
```
train.corr()
```

[19]:
```
                temp_max  temp_avg  temp_min      prec        wd        ws  \
temp_max        1.000000  0.949516  0.747638 -0.100844  0.170304 -0.102377
temp_avg        0.949516  1.000000  0.869429 -0.124874  0.127590 -0.079188
temp_min        0.747638  0.869429  1.000000 -0.130120  0.051205 -0.014435
prec           -0.100844 -0.124874 -0.130120  1.000000  0.161278  0.022470
```

```
wd              0.170304  0.127590  0.051205  0.161278  1.000000 -0.069004
ws             -0.102377 -0.079188 -0.014435  0.022470 -0.069004  1.000000
atmos_pres_avg -0.250040 -0.230673 -0.153937 -0.229134 -0.298460  0.008912
atmos_pres_min -0.227529 -0.197003 -0.105437 -0.235078 -0.294981  0.010274
atmos_pres_max -0.255335 -0.244950 -0.189453 -0.202962 -0.275311  0.003840
rh             -0.014657  0.008368  0.065450  0.027937 -0.036636 -0.033786
ceil_hgt        0.159599  0.153718  0.077008 -0.147485 -0.077427 -0.058980
visibility      0.094221  0.080790  0.051059 -0.007038  0.082084 -0.198670
FMA                  NaN       NaN       NaN       NaN       NaN       NaN

                atmos_pres_avg  atmos_pres_min  atmos_pres_max        rh  \
temp_max             -0.250040       -0.227529       -0.255335 -0.014657
temp_avg             -0.230673       -0.197003       -0.244950  0.008368
temp_min             -0.153937       -0.105437       -0.189453  0.065450
prec                 -0.229134       -0.235078       -0.202962  0.027937
wd                   -0.298460       -0.294981       -0.275311 -0.036636
ws                    0.008912        0.010274        0.003840 -0.033786
atmos_pres_avg        1.000000        0.963162        0.963505  0.010238
atmos_pres_min        0.963162        1.000000        0.875830  0.015856
atmos_pres_max        0.963505        0.875830        1.000000  0.004584
rh                    0.010238        0.015856        0.004584  1.000000
ceil_hgt              0.090694        0.092199        0.080323 -0.149465
visibility            0.070115        0.067297        0.065218  0.064525
FMA                        NaN             NaN             NaN       NaN

                ceil_hgt  visibility  FMA
temp_max        0.159599    0.094221  NaN
temp_avg        0.153718    0.080790  NaN
temp_min        0.077008    0.051059  NaN
prec           -0.147485   -0.007038  NaN
wd             -0.077427    0.082084  NaN
ws             -0.058980   -0.198670  NaN
atmos_pres_avg  0.090694    0.070115  NaN
atmos_pres_min  0.092199    0.067297  NaN
atmos_pres_max  0.080323    0.065218  NaN
rh             -0.149465    0.064525  NaN
ceil_hgt        1.000000   -0.002422  NaN
visibility     -0.002422    1.000000  NaN
FMA                  NaN         NaN  NaN
```

[20]:
```
figure = plt.figure(figsize = (8,8), dpi = 100)
ax = figure.add_subplot(111)
cax = ax.matshow(train.corr(), interpolation = 'nearest')
figure.colorbar(cax)
plt.show()
```



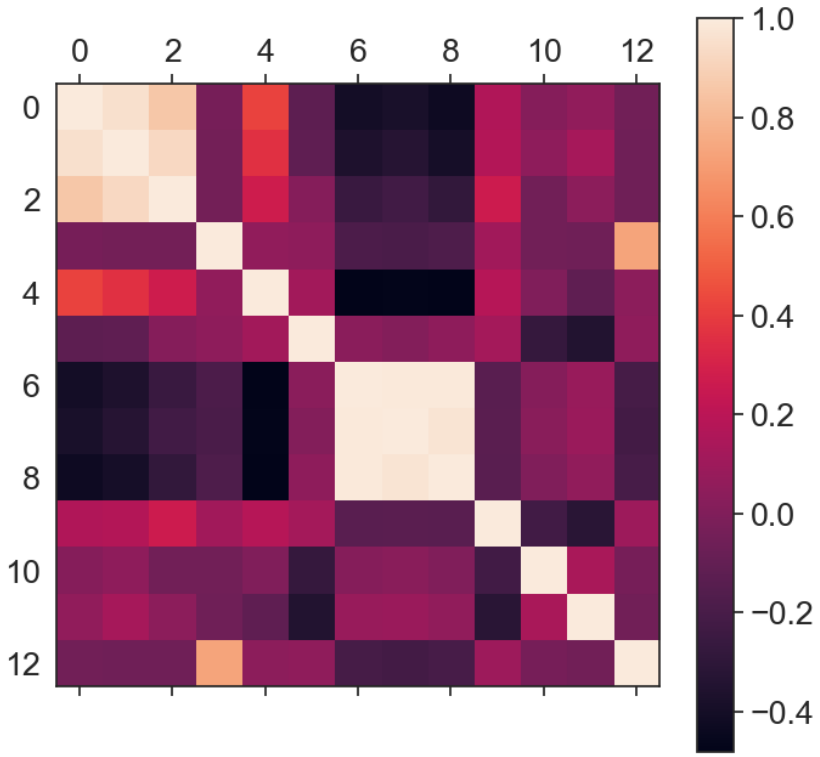
[21]:
```
test.corr()
```

[21]:

	temp_max	temp_avg	temp_min	prec	wd	ws
temp_max	1.000000	0.956674	0.858016	-0.031627	0.418320	-0.119923
temp_avg	0.956674	1.000000	0.928298	-0.043761	0.362853	-0.115150
temp_min	0.858016	0.928298	1.000000	-0.042249	0.272543	0.019969
prec	-0.031627	-0.043761	-0.042249	1.000000	0.062059	0.050729
wd	0.418320	0.362853	0.272543	0.062059	1.000000	0.117928
ws	-0.119923	-0.115150	0.019969	0.050729	0.117928	1.000000
atmos_pres_avg	-0.405454	-0.361046	-0.254483	-0.183748	-0.480057	0.035410
atmos_pres_min	-0.378885	-0.328584	-0.225194	-0.189480	-0.472257	0.015425
atmos_pres_max	-0.424749	-0.389916	-0.281422	-0.177417	-0.481434	0.050063
rh	0.160932	0.178106	0.270641	0.113001	0.185797	0.121566
ceil_hgt	0.019734	0.050818	-0.047724	-0.051967	0.001156	-0.268319
visibility	0.060125	0.130664	0.042781	-0.054016	-0.114197	-0.347454

```
FMA             -0.050312 -0.058648 -0.055440  0.730834  0.043366  0.055182

                 atmos_pres_avg  atmos_pres_min  atmos_pres_max        rh  \
temp_max              -0.405454       -0.378885       -0.424749  0.160932
temp_avg              -0.361046       -0.328584       -0.389916  0.178106
temp_min              -0.254483       -0.225194       -0.281422  0.270641
prec                  -0.183748       -0.189480       -0.177417  0.113001
wd                    -0.480057       -0.472257       -0.481434  0.185797
ws                     0.035410        0.015425        0.050063  0.121566
atmos_pres_avg         1.000000        0.992728        0.991995 -0.138440
atmos_pres_min         0.992728        1.000000        0.974066 -0.131562
atmos_pres_max         0.991995        0.974066        1.000000 -0.137261
rh                    -0.138440       -0.131562       -0.137261  1.000000
ceil_hgt               0.018126        0.032905        0.003763 -0.223856
visibility             0.080715        0.091226        0.058178 -0.317063
FMA                   -0.207435       -0.216938       -0.199413  0.102303

                 ceil_hgt  visibility       FMA
temp_max         0.019734    0.060125 -0.050312
temp_avg         0.050818    0.130664 -0.058648
temp_min        -0.047724    0.042781 -0.055440
prec            -0.051967   -0.054016  0.730834
wd               0.001156   -0.114197  0.043366
ws              -0.268319   -0.347454  0.055182
atmos_pres_avg   0.018126    0.080715 -0.207435
atmos_pres_min   0.032905    0.091226 -0.216938
atmos_pres_max   0.003763    0.058178 -0.199413
rh              -0.223856   -0.317063  0.102303
ceil_hgt         1.000000    0.143032 -0.033159
visibility       0.143032    1.000000 -0.050588
FMA             -0.033159   -0.050588  1.000000
```

[22]:
```
figure = plt.figure(figsize = (8,8), dpi = 100)
ax = figure.add_subplot(111)
cax = ax.matshow(test.corr(), interpolation = 'nearest')
figure.colorbar(cax)
plt.show()
```



0.6 Update Datas

[23]:
```
train = train.drop(['temp_max', 'temp_min', 'atmos_pres_min', 'atmos_pres_max',␣
↪'ceil_hgt', 'visibility', 'FMA'], axis = 1)
```

[24]:
```
test = test.drop(['temp_max', 'temp_min', 'atmos_pres_min', 'atmos_pres_max',␣
↪'ceil_hgt', 'visibility', 'FMA'], axis = 1)
```

[25]:
```
validation = validation.drop(['temp_max', 'temp_min', 'atmos_pres_min',␣
↪'atmos_pres_max', 'ceil_hgt', 'visibility', 'FMA'], axis = 1)
```

[26]:
```
print('Training shape:', train.shape)
print('Test shape:', test.shape)
print('Validation shape:', validation.shape)
```

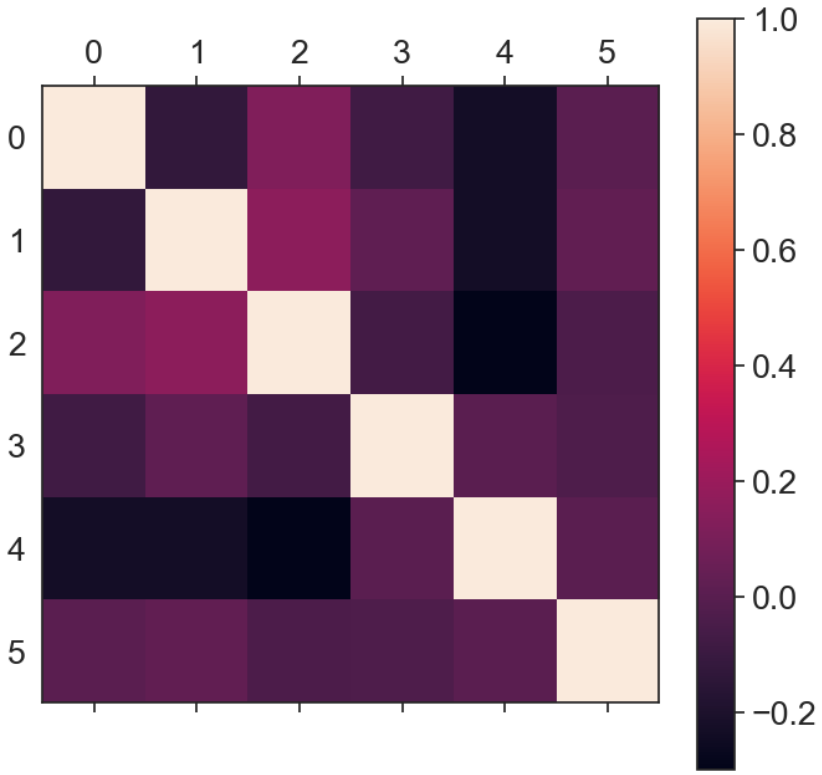
```
Training shape: (10544, 6)
```

```
Test shape: (3347, 6)
Validation shape: (2636, 6)
```

[27]:
```
train.corr()
```

[27]:
```
                 temp_avg      prec        wd        ws  atmos_pres_avg  \
temp_avg         1.000000 -0.124874  0.127590 -0.079188       -0.230673
prec            -0.124874  1.000000  0.161278  0.022470       -0.229134
wd               0.127590  0.161278  1.000000 -0.069004       -0.298460
ws              -0.079188  0.022470 -0.069004  1.000000        0.008912
atmos_pres_avg -0.230673 -0.229134 -0.298460  0.008912        1.000000
rh               0.008368  0.027937 -0.036636 -0.033786        0.010238

                       rh
temp_avg         0.008368
prec             0.027937
wd              -0.036636
ws              -0.033786
atmos_pres_avg   0.010238
rh               1.000000
```

[28]:
```
figure = plt.figure(figsize = (8,8), dpi = 100)
ax = figure.add_subplot(111)
cax = ax.matshow(train.corr(), interpolation = 'nearest')
figure.colorbar(cax)
plt.show()
```



[29]:
```
test.corr()
```

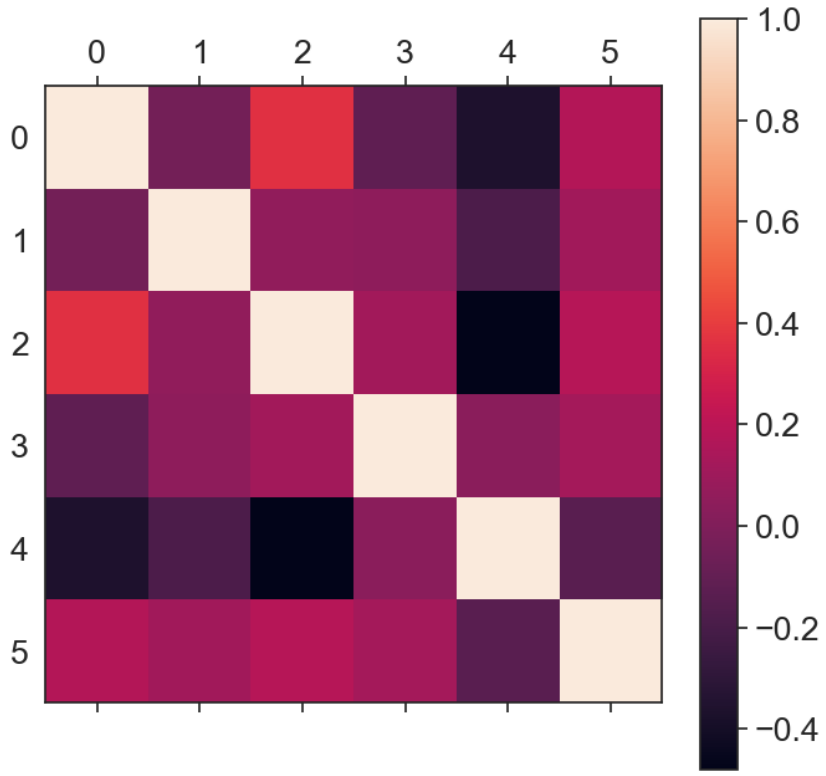
[29]:

	temp_avg	prec	wd	ws	atmos_pres_avg	\
temp_avg	1.000000	-0.043761	0.362853	-0.115150	-0.361046	
prec	-0.043761	1.000000	0.062059	0.050729	-0.183748	
wd	0.362853	0.062059	1.000000	0.117928	-0.480057	
ws	-0.115150	0.050729	0.117928	1.000000	0.035410	
atmos_pres_avg	-0.361046	-0.183748	-0.480057	0.035410	1.000000	
rh	0.178106	0.113001	0.185797	0.121566	-0.138440	

	rh
temp_avg	0.178106
prec	0.113001
wd	0.185797

```
ws              0.121566
atmos_pres_avg -0.138440
rh              1.000000
```

[30]:
```
figure = plt.figure(figsize = (8,8), dpi = 100)
ax = figure.add_subplot(111)
cax = ax.matshow(test.corr(), interpolation = 'nearest')
figure.colorbar(cax)
plt.show()
```



0.7 Data Normalization

[31]:
```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(train)
x_test = scaler.transform(test)
x_validation = scaler.transform(validation)
scaler_filename = 'scaler_data_Hierro'
joblib.dump(scaler, scaler_filename)
```

[31]:
```
['scaler_data_Hierro']
```

[32]:
```
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
print('x_validation shape:', x_validation.shape)
```

```
x_train shape: (10544, 6)
x_test shape: (3347, 6)
x_validation shape: (2636, 6)
```

[33]:
```
x_train = x_train.reshape(x_train.shape[0], 1, x_train.shape[1])
x_test = x_test.reshape(x_test.shape[0], 1, x_test.shape[1])
x_validation = x_validation.reshape(x_validation.shape[0], 1, x_validation.
 ↪shape[1])
```

[34]:
```
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
print('x_validation shape:', x_validation.shape)
```

```
x_train shape: (10544, 1, 6)
x_test shape: (3347, 1, 6)
x_validation shape: (2636, 1, 6)
```

0.8 Model

[35]:
```
def autoencoder_model_GRU(X):
    inputs = Input(shape = (X.shape[1], X.shape[2]))

    L1 = GRU(32, activation = 'relu', return_sequences = True)(inputs)
    L2 = Dropout(0.2)(L1)
    L3 = GRU(4, activation = 'relu', return_sequences = False)(L2)
    L4 = RepeatVector(X.shape[1])(L3)
    L5 = GRU(4, activation = 'relu', return_sequences = True)(L4)
    L6 = GRU(32, activation = 'relu', return_sequences = True)(L5)

    outputs = TimeDistributed(Dense(X.shape[2]))(L6)

    model = Model(inputs = inputs, outputs = outputs)
    return model
```

[36]:
```
model = autoencoder_model_GRU(x_train)
model.compile(optimizer = 'adamax', loss = 'mae', metrics = ['mae', 'mse',␣
↪'mape', 'msle', 'cosine_proximity'])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1, 6)]            0
_________________________________________________________________
gru (GRU)                    (None, 1, 32)             3840
_________________________________________________________________
dropout (Dropout)            (None, 1, 32)             0
_________________________________________________________________
gru_1 (GRU)                  (None, 4)                 456
_________________________________________________________________
repeat_vector (RepeatVector) (None, 1, 4)              0
_________________________________________________________________
gru_2 (GRU)                  (None, 1, 4)              120
_________________________________________________________________
gru_3 (GRU)                  (None, 1, 32)             3648
_________________________________________________________________
time_distributed (TimeDistri (None, 1, 6)              198
=================================================================
Total params: 8,262
Trainable params: 8,262
Non-trainable params: 0
_________________________________________________________________
```

0.9 Training

[37]:
```
dataframe_train = DataFrame()
dataframe_validation = DataFrame()

early_stopping_callback = EarlyStopping(monitor='val_loss', mode='min',␣
↪verbose=1, patience=50)

for i in range(1):
    print('Fit model on training data...')
    start = time.time()

    epochs = 300
    batch_size = 48
```

```
    history = model.fit(x_train, x_train, epochs = epochs, batch_size =␣
↪batch_size, validation_data = (x_validation, x_validation),␣
↪callbacks=[early_stopping_callback], verbose = 0).history

    end = time.time()

    dataframe_train[str(i)] = history['loss']
    dataframe_validation[str(i)] = history['val_loss']

    print('Time to training model:', end - start)
```

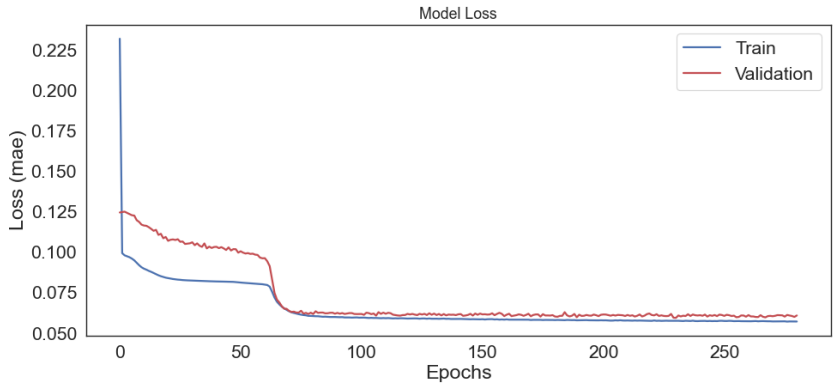
```
Fit model on training data…
Epoch 00281: early stopping
Time to training model: 2100.978576898575
```

0.10 Training Results

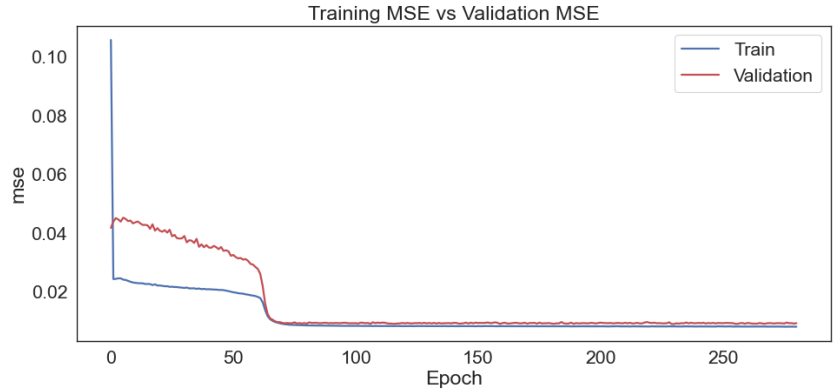
[38]:
```
loss = model.evaluate(x_train, x_train, verbose = 0)
for name, value in zip(model.metrics_names, loss):
    print(name, value)
```

```
loss 0.055625371634960175
mae 0.055625371634960175
mse 0.00801103189587593
mape 183126.0625
msle 0.0037872642278671265
cosine_proximity 0.9876352548599243
```

[39]:
```
figure, ax = plt.subplots(figsize = (14, 6), dpi = 80)
ax.plot(history['loss'], 'b', label = 'Train', linewidth = 2)
ax.plot(history['val_loss'], 'r', label = 'Validation', linewidth = 2)
ax.set_title('Model Loss', fontsize = 16)
ax.set_ylabel('Loss (mae)')
ax.set_xlabel('Epochs')
ax.legend(loc = 'upper right')
plt.show()
```

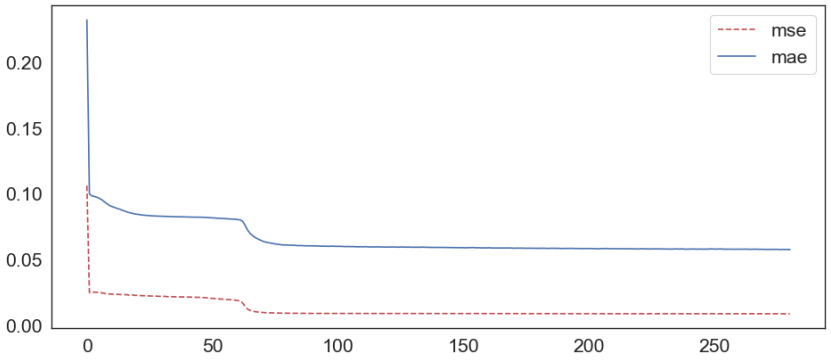


[40]:
```
fig, ax = plt.subplots(figsize = (14, 6), dpi = 80)
ax.plot(history['mse'], 'b', label = 'Train', linewidth = 2)
ax.plot(history['val_mse'], 'r', label = 'Validation', linewidth = 2)
ax.set_title('Training MSE vs Validation MSE')
ax.set_ylabel('mse')
ax.set_xlabel('Epoch')
ax.legend(loc = 'upper right')
plt.show()
```



[41]:
```
fig, ax = plt.subplots(figsize = (14, 6), dpi = 80)
ax.plot(history['mse'], 'r--', label = 'mse')
```

```
ax.plot(history['mae'], 'b', label = 'mae')
ax.legend(loc = 'upper right')
plt.show()
```



0.10.1 Distribution of Loss Function

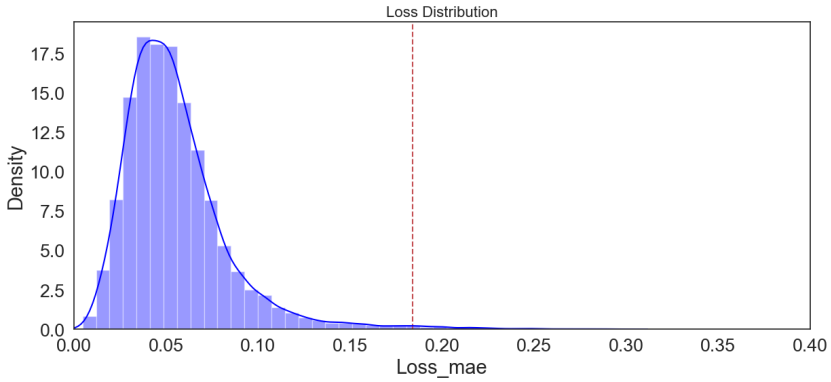
[42]:
```
x_pred = model.predict(x_train)
x_pred = x_pred.reshape(x_pred.shape[0], x_pred.shape[2])
x_pred = pd.DataFrame(x_pred, columns = train.columns)
x_pred.index = train.index
```

[43]:
```
scored = pd.DataFrame(index = train.index)
Xtrain = x_train.reshape(x_train.shape[0], x_train.shape[2])
scored['Loss_mae'] = np.mean(np.abs(x_pred - Xtrain), axis = 1)
```

[44]:
```
plt.figure(figsize = (14, 6), dpi = 100)
plt.title('Loss Distribution', fontsize = 16)
plt.axvline(0.184, ls = '--', color = 'r')
plt.annotate('Threshold = 0.184', xy = (0.06, 30), xycoords = 'data', fontsize␣
 ↪= 14, horizontalalignment = 'center', verticalalignment = 'bottom')
sns.distplot(scored['Loss_mae'], bins = 40, kde = True, color = 'blue')
plt.xlim([0.0, .4])
```

```
D:\TFG\Environment\TFG\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[44]:
```
(0.0, 0.4)
```



0.11 Test Predictions

[45]:
```
x_pred = model.predict(x_test)
x_pred = x_pred.reshape(x_pred.shape[0], x_pred.shape[2])
x_pred = pd.DataFrame(x_pred, columns = test.columns)
x_pred.index = test.index
```

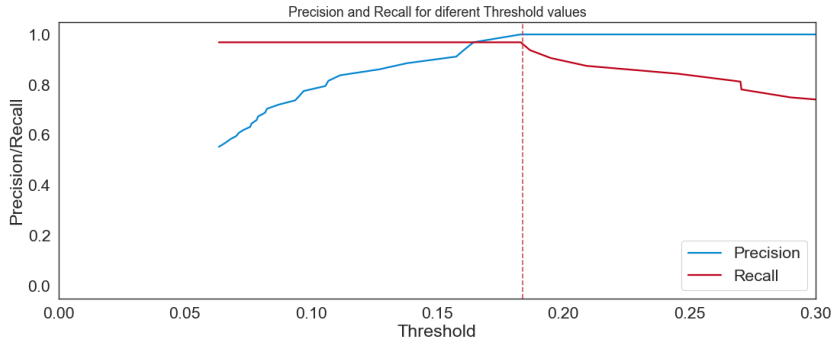
[46]:
```
error_dataframe = pd.DataFrame(index = test.index)
Xtest = x_test.reshape(x_test.shape[0], x_test.shape[2])
error_dataframe['Reconstruction_error'] = np.mean(np.power(x_pred - Xtest, 2),␣
↪axis = 1)
error_dataframe['True_class'] = test_fma
error_dataframe.head()
```

[46]:

	Reconstruction_error	True_class
13388	0.014947	0
13389	0.012112	0
13390	0.027037	0
13391	0.013097	0
13392	0.013282	0

[47]:
```
precision, recall, threshold = precision_recall_curve(error_dataframe.
↪True_class, error_dataframe.Reconstruction_error)
```

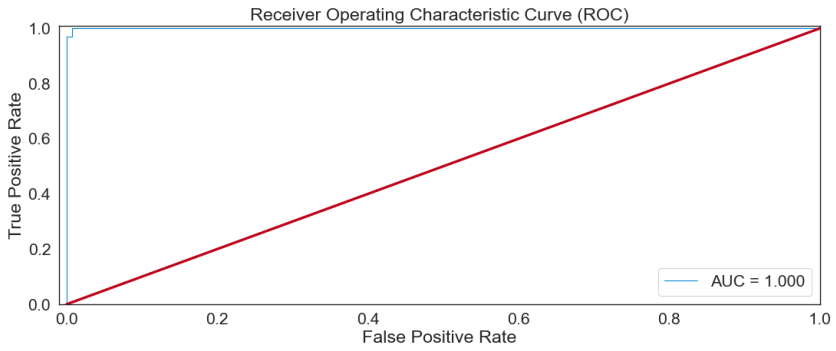
[48]:
```
fig, ax = plt.subplots(figsize = (16, 6), dpi = 80)
ax.plot(threshold, precision[1:], label = 'Precision', linewidth = 2)
ax.plot(threshold, recall[1:], label = 'Recall', linewidth = 2)
ax.set_title('Precision and Recall for diferent Threshold values', fontsize =␣
↪16)
ax.set_xlabel('Threshold')
ax.set_ylabel('Precision/Recall')
ax.set_xlim([0.0, 0.3])
ax.axvline(0.184, ls = '--', color = 'r')
ax.legend(loc = 'lower right')
plt.show()
```



0.11.1 ROC Curve Check

[49]:
```
false_pos_rate, true_pos_rate, threshold = roc_curve(error_dataframe.
↪True_class, error_dataframe.Reconstruction_error)
roc_auc = auc(false_pos_rate, true_pos_rate)
```

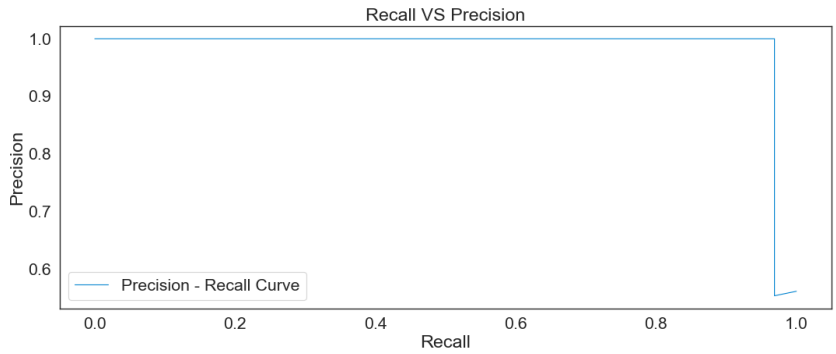
[50]:
```
fig, ax = plt.subplots(figsize = (16, 6), dpi = 80)
ax.plot(false_pos_rate, true_pos_rate, linewidth = 1, label = 'AUC = %0.3f' %␣
↪roc_auc)
ax.plot([0,1], [0,1], linewidth = 3)
ax.set_title('Receiver Operating Characteristic Curve (ROC)')
ax.set_ylabel('True Positive Rate')
ax.set_xlabel('False Positive Rate')
ax.set_xlim([-0.01, 1])
ax.set_ylim([0, 1.01])
ax.legend(loc = 'lower right')
plt.show()
```



0.11.2 Recall VS Precision Thresholding

[51]:
```
precision, recall, threshold = precision_recall_curve(error_dataframe.
↪True_class, error_dataframe.Reconstruction_error)
```

[52]:
```
fig, ax = plt.subplots(figsize = (16, 6), dpi = 80)
ax.plot(recall, precision, linewidth = 1, label = 'Precision - Recall Curve')
ax.set_title('Recall VS Precision')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
ax.legend(loc = 'lower left')
plt.show()
```



0.11.3 Reconstruction Error VS Threshold Check

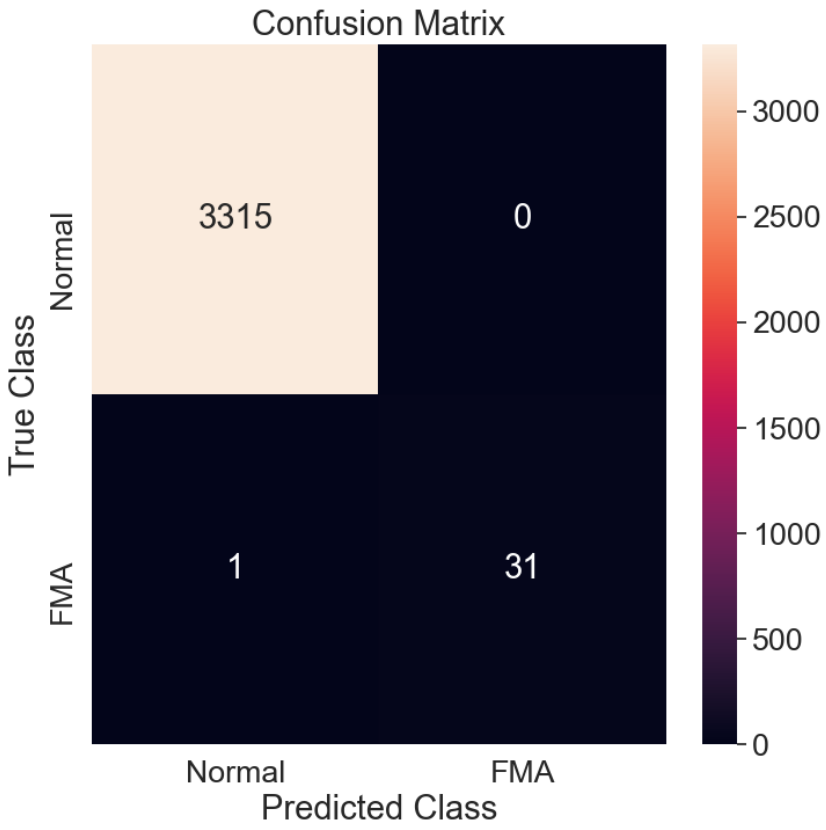
[53]:
```
threshold_fixed = 0.184
groups = error_dataframe.groupby('True_class')
```

[54]:
```
fig, ax = plt.subplots(figsize = (16, 10), dpi = 80)

for name, group in groups:
    ax.plot(group.index, group.Reconstruction_error, marker = 'o', ms = 3.5,␣
 ↪linestyle = '', label = 'FMA' if name == 1 else 'Normal')

ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors = 'r',␣
 ↪zorder = 100, label = 'Threshold')
ax.set_title('Reconstruction Error for different classes')
ax.set_ylabel('Reconstruction Error')
ax.set_xlabel('Data Point Index')
ax.set_ylim([0, 0.5])
ax.legend(loc = 'upper right')
plt.show()
```

0.11.4 Confusion Matrix

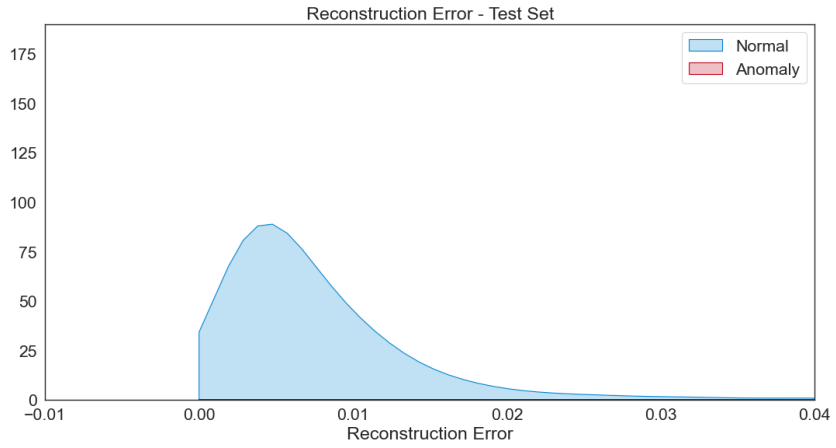
[55]:
```
pred_y = [1 if error > threshold_fixed else 0 for error in error_dataframe.
↪Reconstruction_error.values]
```

[56]:
```
matrix = confusion_matrix(error_dataframe.True_class, pred_y)
fig, ax = plt.subplots(figsize = (8, 8), dpi = 100)
sns.heatmap(matrix, xticklabels = LABELS, yticklabels = LABELS, annot = True,␣
↪fmt = 'd')
ax.set_title('Confusion Matrix')
ax.set_ylabel('True Class')
ax.set_xlabel('Predicted Class')
plt.show()
```



0.11.5 Reconstruction Error - Test Set

[57]:
```
fig, ax = plt.subplots(figsize = (16, 8), dpi = 80)
sns.kdeplot(error_dataframe.Reconstruction_error[error_dataframe.True_class ==␣
↪0], label = 'Normal', shade = True, clip = (0, 10))
sns.kdeplot(error_dataframe.Reconstruction_error[error_dataframe.True_class ==␣
↪1], label = 'Anomaly', shade = True, clip = (0, 10))
ax.set_title('Reconstruction Error - Test Set')
ax.set_xlim([-0.01, 0.04])
ax.set_ylim([0, 190])
ax.set_xlabel('Reconstruction Error')
ax.set_ylabel('')
ax.legend(loc = 'upper right')
plt.show()
```



0.12 Thresholds

0.12.1 Threshold with Max

[58]:
```
x_train_pred = model.predict(x_train)
train_loss = np.mean(np.abs(x_train_pred - x_train), axis = 1)
threshold_max = np.max(train_loss)
print("Reconstruction error threshold with max: ", threshold_max)
```

```
Reconstruction error threshold with max:  1.000195269312826
```

0.12.2 Threshold with Mean and Std

[59]:
```
x_train_pred = model.predict(x_train)
train_loss = np.mean(np.abs(x_train_pred - x_train), axis = 1)
threshold_mean_std = np.mean(train_loss) + np.std(train_loss)
print("Reconstruction error threshold with mean and std: ", threshold_mean_std)
```

```
Reconstruction error threshold with mean and std:  0.1257456349912156
```

0.12.3 Threshold Precision Recall

[60]:
```
def intersection_point():
    precision_points = np.array([[threshold[i],precision[i]] for i in␣
 ↪range(len(precision[1:]))])
    recall_points = np.array([[threshold[i],recall[i]] for i in␣
 ↪range(len(recall[1:]))])
    nrows, ncols = precision_points.shape
    dtype = {'names': ['f{}'.format(i) for i in range(ncols)], 'formats': ncols␣
 ↪* [precision_points.dtype]}
    intersection = np.intersect1d(precision_points.view(dtype), recall_points.
 ↪view(dtype))
    return intersection.view(precision_points.dtype).reshape(-1, ncols)
```

[61]:
```
threshold_precision_recall = intersection_point()[0][0]
```

[62]:
```
print(threshold_precision_recall)
```

```
0.1831234243062405
```

[63]:
```
def print_stats(ytest, ypred):
    print("Accuracy: {:.5f}, Cohen's Kappa Score: {:.5f}".format(
        accuracy_score(ytest, ypred),
        cohen_kappa_score(ytest, ypred, weights="quadratic")))
    ll = log_loss(ytest, ypred)
    print("Log Loss: {}\n".format(ll))
    print("Confusion Matrix:")
    print(confusion_matrix(ytest, ypred))
    print("Classification Report:")
    print(classification_report(ytest, ypred, target_names = LABELS))
    print("Fbeta Score(0.5):{:.3f}".format(fbeta_score(ytest, ypred,␣
 ↪average="micro", beta=0.5)))
```

[64]:
```
threshold_fixed = [threshold_max, threshold_mean_std,␣
 ↪threshold_precision_recall]

print('---------------------------------------------------------------------------')

for threshold in threshold_fixed:
```

```
    pred_y = [1 if error > threshold else 0 for error in error_dataframe.
↪Reconstruction_error.values]
    matrix = confusion_matrix(error_dataframe.True_class, pred_y)

    precision = precision_score(error_dataframe.True_class, pred_y, average =␣
↪'weighted')
    recall = recall_score(error_dataframe.True_class, pred_y, average =␣
↪'weighted')
    fbeta = fbeta_score(error_dataframe.True_class, pred_y, beta = 0.5)

    print('Result:\n')
    print('Precision Score = %.3f\nRecall Score = %.3f\nFbeta Score = %.3f\n' %␣
↪(precision, recall, fbeta))
    print('Threshold fixed = %.3f' % threshold)
    print_stats(error_dataframe.True_class, pred_y)
  ␣
↪print('------------------------------------------------------------------------------')
```

```
------------------------------------------------------------------------------
Result:

Precision Score = 0.994
Recall Score = 0.994
Fbeta Score = 0.774

Threshold fixed = 1.000
Accuracy: 0.99432, Cohen's Kappa Score: 0.57543
Log Loss: 0.1960671501354366

Confusion Matrix:
[[3315    0]
 [  19   13]]
Classification Report:
              precision    recall  f1-score   support

      Normal       0.99      1.00      1.00      3315
         FMA       1.00      0.41      0.58        32

    accuracy                           0.99      3347
   macro avg       1.00      0.70      0.79      3347
weighted avg       0.99      0.99      0.99      3347

Fbeta Score(0.5):0.994
------------------------------------------------------------------------------
Result:

Precision Score = 0.998
```

```
Recall Score = 0.998
Fbeta Score = 0.861

Threshold fixed = 0.126
Accuracy: 0.99791, Cohen's Kappa Score: 0.89750
Log Loss: 0.07223669923781276

Confusion Matrix:
[[3309    6]
 [   1   31]]
Classification Report:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00      3315
         FMA       0.84      0.97      0.90        32

    accuracy                           1.00      3347
   macro avg       0.92      0.98      0.95      3347
weighted avg       1.00      1.00      1.00      3347

Fbeta Score(0.5):0.998
--------------------------------------------------------------------------------
Result:

Precision Score = 1.000
Recall Score = 1.000
Fbeta Score = 0.994

Threshold fixed = 0.183
Accuracy: 0.99970, Cohen's Kappa Score: 0.98398
Log Loss: 0.010319323691339715

Confusion Matrix:
[[3315    0]
 [   1   31]]
Classification Report:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00      3315
         FMA       1.00      0.97      0.98        32

    accuracy                           1.00      3347
   macro avg       1.00      0.98      0.99      3347
weighted avg       1.00      1.00      1.00      3347

Fbeta Score(0.5):1.000
--------------------------------------------------------------------------------
```

Bibliografía

[1] Sridhar Alla, Suman Kalyan Adari. Outlier definition. In *Beginning Anomaly Detection Using Python-Based Deep Learning*, page 16, 2019.

[2] Sridhar Alla, Suman Kalyan Adari. Relu. In *Beginning Anomaly Detection Using Python-Based Deep Learning*, pages 102–103, 2019.

[3] Sridhar Alla, Suman Kalyan Adari. The three styles of anomaly detection. In *Beginning Anomaly Detection Using Python-Based Deep Learning*, page 19, 2019.

[4] Sridhar Alla, Suman Kalyan Adari. What are autoencoders? In *Beginning Anomaly Detection Using Python-Based Deep Learning*, pages 123–127, 2019.

[5] Sridhar Alla, Suman Kalyan Adari. What is an anomaly? In *Beginning Anomaly Detection Using Python-Based Deep Learning*, page 1, 2019.

[6] AEMET Agencia Estatal de Meteorología. Adverse weather phenomena, 2021. [Online, accessed 2021-06-18].

[7] Deekshith Shetty, Harshavardhan C A, M Jayanth Varma, Shrishail Navi, Mohammed Riyaz Ahmed. Deep learning applications. In *Diving Deep into Deep Learning: History, Evolution, Types and Applications*, pages 2840–2844, Jan 2020.

[8] Yara Alghofaili, Albatul Albattah, and Murad A. Rassam. A financial fraud detection model based on lstm deep learning technique. *Journal of Applied Security Research*, 15(4):498–516, 2020.

[9] Afshine Amidi, Shervine Amidi. Recurrent neural network cheatsheet. [Online, accessed 2021-06-20].

[10] Dima Suleiman, Arafat Awajan. Deep learning based abstractive text summarization: Approaches, datasets, evaluation measures, and challenges. *Mathematical Problems in Engineering*, 2020:29, 2020.

[11] Samuel R. Bowman, Luke Vilnis , Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, Samy Bengio. Generating sentences from a continuous space. *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 10–21, 2016.

[12] Pournami S. Chandran, N B Byju, R U Deepak, K N Nishakumari, P Devanand, and P M Sasi. Missing child identification system using deep learning and multiclass svm. In *2018 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pages 113–116, 2018.

[13] Arden Dertat. Applied deep learning - part 3: Autoencoders, 2017. [Online, accessed 2021-06-23].

[14] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery.

[15] EcuRed. Lluvia orográfica, Aug 2019. [Online, accessed 2021-06-18].

[16] NOAA National Centers for Environmental Information. *Federal Climate Complex Data Documentation For Integrated Surface Data (ISD)*. US Air Force - 14th Weather Squadron, 151 Patton Avenue Asheville, NC 28801-5001 USA, January 2018.

[17] NOAA National Centers for Environmental Information. Liquid-precipitation condition code. In *Federal Climate Complex Data Documentation For Integrated Surface Data (ISD)*, pages 13–14, 151 Patton Avenue Asheville, NC 28801-5001 USA, January 2018.

[18] NOAA National Centers for Environmental Information. Liquid-precipitation quality code. In *Federal Climate Complex Data Documentation For Integrated Surface Data (ISD)*, page 14, 151 Patton Avenue Asheville, NC 28801-5001 USA, January 2018.

[19] Mª Dolores Olmeda Gordo. Plan nacional de predicción y vigilancia de fenómenos meteorológicos adversos, Jun 2018. [Online, accessed 2021-06-19].

[20] Fernando Pérez, Brian Granger. Jupyter project, 2021. [Online, accessed 2021-06-21].

[21] GitHub Inc. Github: Where the world builds software. [Online, accessed 2021-06-23].

[22] RStudio Inc. Rstudio | open source & professional software for data science, 2021. [Online, accessed 2021-06-21].

[23] Keras. Keras api reference. callbacks api. earlystopping. [Online, accessed 2021-06-19].

[24] Sergey Kibish. A note about finding anomalies, 2018. [Online, accessed 2021-06-20].

[25] Scikit Learn. Scikit learn: Machine learning in python. [Online, accessed 2021-06-19].

[26] Amitha Mathew, Amudha Arul, and S. Sivakumari. *Deep Learning Techniques: An Overview*, pages 599–608. Springer Science+Business Media, 01 2021.

[27] Mysid. Blank topographic map of the canary islands. `https://commons.wikimedia.org/wiki/File:Blank_topographic_map_of_the_Canary_Islands.svg`, 2021. [Online, accessed 2021-06-23].

[28] National Oceanic and Atmospheric Administration. Noaa about us page, May 2021. [Online, accessed 2021-06-18].

[29] Janhavi R. Chaudhary, Ankit C. Patel. Bilingual machine translation using rnn based deep learning. *International Journal of Scientific Research in Science, Engineering and Technology*, 4:1480–4, 2018.

[30] Hemanth Pedamallu. Rnn vs gru vs lstm, 2020. [Online, accessed 2021-06-23].

[31] Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation, 2018. [Online, accessed 2021-06-23].

[32] Raúl Rodríguez Torres. Tfg_awp_detection_via_autoencoder. `https://github.com/raulrodrigueztorres/TFG_AWP_DETECTION_VIA_AUTOENCODER`, 2021. [Online, accessed 2021-06-23].

[33] Brij Rokad. Machine learning approaches and its applications, 2019. [Online, accessed 2021-06-23].

[34] Adrian Rosebrock. Black and white image colorization with opencv and deep learning. [Online, accessed 2021-06-19].

[35] Abhimanyu Roy, Jingyi Sun, Robert Mahoney, Loreto Alonzi, Stephen Adams, and Peter Beling. Deep learning detecting fraud in credit card transactions. In *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pages 129–134, 2018.

[36] Alejandro Sepúlveda. Los eventos meteorológicos que marcaron al mundo en 2020, Dec 2020. [Online, accessed 2021-06-18].

[37] Harpreet Singh, Emily M. Hand, and Kostas Alexis. Anomalous motion detection on highway using deep learning. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1901–1905, 2020.

[38] Devansh Srivastav, Akansha Bajpai, and Prakash Srivastava. Improved classification for pneumonia detection using transfer learning with gan based synthetic image augmentation. In *2021 11th International Conference on*

Cloud Computing, Data Science Engineering (Confluence), pages 433–437, 2021.

[39] Google Brain Team. Introduction to the keras tuner, 2021. [Online, accessed 2021-06-24].

[40] Google Brain Team. Tensorflow, 2021. [Online, accessed 2021-06-21].

[41] Keras Team. Keras: the python deep learning api, 2021. [Online, accessed 2021-06-21].

[42] R Core Team. R documentation. [Online, accessed 2021-06-18].

[43] Wang Weihong and Tu Jiaoyang. Research on license plate recognition algorithms based on deep learning in complex environment. *IEEE Access*, 8:91661–91675, 2020.