Segmentación automática de series temporales con algoritmos de aprendizaje no supervisado con aplicación a la calidad de aire en Madrid

Alejandro Luque Cerpa

Segmentación automática de series temporales con algoritmos de aprendizaje no supervisado con aplicación a la calidad de aire en Madrid

Alejandro Luque Cerpa

Memoria presentada como parte de los requisitos para la obtención del título de Máster Universitario en Lógica, Computación e Inteligencia Artificial por la Universidad de Sevilla.

Tutorizada por

Prof. Tutor Miguel Ángel Gutiérrez Naranjo
Prof. Tutor Miguel Cárdenas Montes

Agradecimientos

Muchas gracias a mis dos tutores: Miguel Ángel Gutiérrez Naranjo y Miguel Cárdenas Montes, por haberme guiado, acompañado y ayudado en la elaboración de este Trabajo de Fin de Máster, así como por la pasión mostrada a la hora de trabajar sobre este tema. Este trabajo le debe mucho al interés mostrado por ambos.

Muchas gracias, de nuevo, a mis padres por soportar otro Trabajo de Fin de Estudios más y por apoyarme todo este tiempo.

Agradecimientos también a los Renacentistas, por seguir haciendo que nuestras vidas se crucen una y otra vez.

A Rafael Leal, de nuevo, por acompañarme en esta aventura desde el principio y por los intereses compartidos respecto al tema.

A Gadea Méndez, por seguir elevándonos.

Índice general

Abstract

The main objective of this paper is to model some time series: the Madrid air pollution data. Nowadays, Deep Learning systems are becoming really important for time series modelling. There are different neural networks dedicated to this task, such as convolutional neural networks or recurrent neural networks. This kind of models manage to learn long time dependencies between the data, something really useful for time series analysis. Some of these models are introduced in this paper, as well as some statistical models like ARIMA. After studying and comparing them, one special network is chosen: U-Time. U-Time is "a fully feed-forward deep learning approach to physiological time series segmentation developed for the analysis of sleep data" [Per+19]. It is possible to modify U-Time and to train that variant with the air pollution data. After that, we get a model that maps sequential inputs to labels previously defined related to the pollution levels.

Resumen

El objetivo principal de este trabajo es modelizar un ejemplo concreto de serie temporal: los datos de contaminación del aire de Madrid. Actualmente, los sistemas de Aprendizaje Profundo han demostrado ser herramientas de gran importancia a la hora de modelizar series temporales. Hay diferentes tipos de redes neuronales dedicadas a esta tarea, como pueden ser las redes neuronales convolucionales y las redes neuronales recurrentes. Estos modelos consiguen aprender dependencias temporales a muy largo plazo entre los datos, algo que es verdaderamente útil en cuanto al análisis de series temporales. En este trabajo se introducen algunos de estos modelos, así como algunos modelos estadísticos como ARIMA. Tras estudiarlos y compararlos, se elige una red neuronal en concreto: U-Time. U-Time es "un sistema completamente prealimentado que constituye una aproximación desde el Aprendizaje Profundo a la segmentación de series temporales, desarrollado con el objetivo de analizar datos relacionados con las fases del sueño" [Per+19]. Es posible modificar U-Time y entrenar dicha variante con los datos de contaminación del aire. Después del entrenamiento, se obtiene un modelo que asigna a secuencias de entrada distintas etiquetas previamente definidas relacionadas con niveles de contaminación.

1 Introducción

En ocasiones, por la naturaleza de los problemas que se estudian, se trabaja con series de datos que mantienen dependencias temporales a muy largo plazo. Ser capaces de determinar estas estacionalidades de forma automática sin señalar previamente cuál es el periodo de dichas estacionalidades otorga una gran ventaja a la hora de entender los datos y a la hora de predecir su comportamiento.

En este trabajo se va a analizar una serie de datos con información sobre la contaminación del aire de Madrid por dióxido de nitrógeno (NO_2). Se va a intentar desarrollar un modelo que permita predecir, dados unos datos de un periodo de tiempo, cuál es el nivel de peligro por contaminación en un periodo próximo.

En este primer capítulo se verá qué son las series temporales y cómo se ha enfocado su análisis desde la estadística. Se hablará también del Aprendizaje Profundo y de herramientas que se han desarrollado dentro de este ámbito. También se comentarán algunos artículos en los que se ha comparado el uso de herramientas estadísticas con el uso de modelos de Aprendizaje Profundo.

El segundo capítulo se centra en una red en neuronal ya desarrollada por otros investigadores que se han dedicado a la clasificación de señales de estudios polisomnográficos: U-Time. En este capítulo se va a explicar cómo es su estructura y funcionamiento, así como los resultados que han obtenido los desarrolladores.

En el tercer capítulo se desarrolla la investigación realizada a partir de U-Time, y de cómo se ha creado una variante de esta para intentar predecir los niveles de contaminación por NO_2, usando datos tanto horarios como diarios.

1.1 Series temporales

A lo largo del tiempo se han utilizado herramientas estadísticas para estudiar el comportamiento de multitud de datos experimentales de las diferentes ramas de la ciencia, como pueden ser la física, la química o la biología, y también de las ciencias sociales, como la economía o la psicología. Conforme se avanzaba en esta dirección se han encontrado algunas limitaciones en los métodos de los que se disponía, como en el caso de aquellos datos que cuentan con una fuerte correlación entre datos que están próximos en el tiempo. Gran parte de los métodos tradicionales de la estadística se basan en asumir que las observaciones que se realizan son independientes e idénticamente distribuidas, por lo que no podrían ser utilizadas en el caso mencionado. El análisis de datos que mantienen una cierta dependencia temporal se conoce como análisis de series temporales.

Las series temporales adquieren gran importancia en multitud de disciplinas. En economía se pueden encontrar a la hora de estudiar las ganancias de una empresa (Figura 1.1) o los niveles de desempleo de un país. En epidemiología se pueden encontrar a la hora de estudiar la expansión y mortalidad de un virus en un determinado periodo temporal. En meteorología y en sismología aparecen continuamente, y su análisis puede permitir realizar predicciones sobre la situación del clima en un cierto periodo de tiempo o sobre cuándo tendrá lugar el siguiente terremoto. En este caso se va a estudiar una serie temporal: la contaminación por NO_2 en el aire de Madrid (Figura 1.2).

Se tiene pues que el objetivo principal del análisis de series temporales es desarrollar modelos matemáticos que permitan describir detalladamente el comportamiento de dichas series y, en determinadas ocasiones, realizar predicciones utilizando dichos modelos. La estacionalidad de las series toma un papel relevante aquí, por lo que las herramientas que se usan centran parte de su esfuerzo en determinar cómo influye. En el siguiente apartado se verán algunos de los métodos más utilizados a la hora de estudiar series temporales teniendo en cuenta la estacionalidad: los modelos ARIMA.

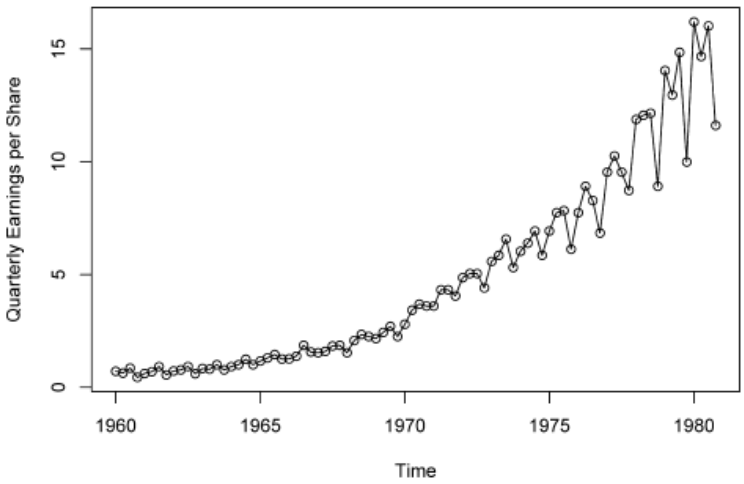


Figura 1.1: Ganancias cuatrimestrales por acción de Johnson&Johnson entre 1960-I y 1980-IV. Nótese que aunque es evidente la tendencia ascendente, también se ve una cierta correlación cuatrimestral entre los datos: en el cuarto cuatrimestre de cada año las ganancias caen un poco. Imagen extraída de [SS00].

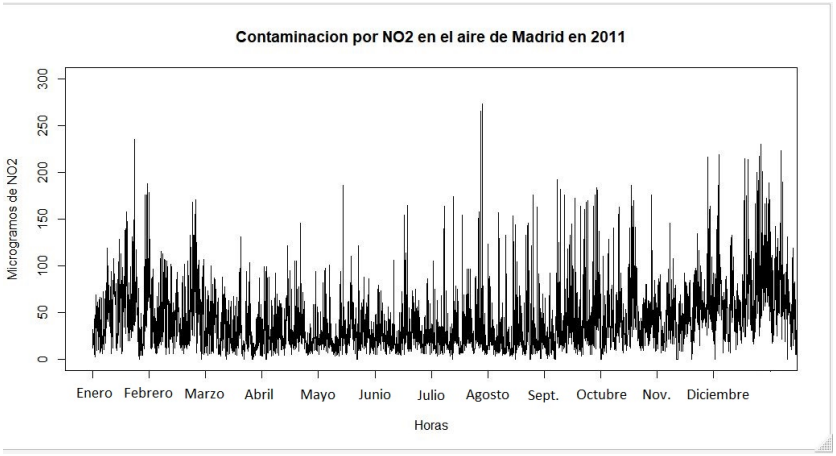


Figura 1.2: Datos horarios de la concentración de NO_2 en 2011 en la estación de monitorización de Plaza del Carmen (Madrid). Imagen propia.

1.1.1 Análisis de Series Temporales: Modelos ARIMA

En este apartado se van a esbozar tres modelos: el modelo AR, el modelo ARMA y el modelo ARIMA. Sea una serie temporal $x_0, x_1, x_2, ..., x_{t-1}, x_t$, siendo t la variable que indica la posición en el tiempo. La idea general que engloba a los modelos ARIMA es que se puede expresar cualquier valor x_t de la serie en función de los p valores anteriores a este: $x_{t-p}, x_{t-p+1}, ..., x_{t-1}$, donde p determina la cantidad de valores que se tienen que observar en el pasado para poder predecir el valor x_t. Nótese que la estacionalidad indica el valor p correspondiente.

El objetivo de este apartado no es realizar un desarrollo exhaustivo de estos modelos, sino esbozarlos a grandes rasgos para poder ver la diferencia de enfoque entre ellos y las herramientas que se han desarrollado en el ámbito del Aprendizaje Profundo. Para modelizar series temporales, se utilizan secuencias $\{X_t\}\ t \geq 0$ de variables aleatorias.

Definición 1.1 (Serie temporal estacionaria). *Sea $\{Y_t\}\ t \geq 0$ una secuencia de variables aleatorias. Se dice que es estacionaria si:*

- *La función del valor medio μ_t no depende del tiempo, es decir, $\mu_t = E(Y_t) = \mu$ constante para todo t.*
- *La función $\gamma(s,t)$, definida como $\gamma(s,t) = cov(Y_s, Y_t)$, cumple que $\gamma(t, t+k) = \gamma(0,k)\ \forall t \geq 0\ \forall k \geq 0$. Es decir, la covarianza de dos variables aleatorias de la secuencia viene definida en función de la diferencia de las posiciones de una y otra en la secuencia.*

Se dice que una serie temporal es estacionaria si lo es la secuencia de variables aleatorias que la modeliza. Se pueden definir los modelos AR, ARMA y ARIMA, además de un modelo extra que necesitamos definir: el modelo MA.

Definición 1.2 (Modelo AR). *Un modelo autorregresivo (Autoregresive model) de orden p, denotado AR(p), es de la forma:*

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + \omega_t \tag{1.1}$$

donde $x_1, x_2, ..., x_t$ es una serie temporal estacionaria, $\phi_1, \phi_2, ..., \phi_p$ son constantes, $\phi_p \neq 0$ y ω_t es ruido blanco gaussiano con media 0 y varianza σ^2_ω. Aquí se asume que la media μ de x_t es 0. Si no es así, entonces:

$$x_t = \alpha + \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + \omega_t \tag{1.2}$$

donde $\alpha = \mu(1 - \phi_1 - \phi_2 - ... - \phi_p)$.

Con esta definición se tiene pues un modelo que expresa cada dato de la serie temporal x_t en función de los p valores anteriores mediante relaciones lineales. A veces, sin embargo, se tiene que la dependencia no está tanto en los valores anteriores como en el ruido blanco que introducen. Se introduce pues el modelo de medias móviles.

Definición 1.3 (Modelo de Medias Móviles). *Un modelo de medias móviles (Moving Average Model) de orden q, denotado MA(q), es de la forma:*

$$x_t = \omega_t + \theta_1 \omega_{t-1} + \theta_2 \omega_{t-2} + ... + \theta_q \omega_{t-q} \tag{1.3}$$

donde los $\theta_1, \theta_2, ..., \theta_q$ son parámetros, $\theta_q \neq 0$ y los ω_τ son una serie de ruidos blancos de media 0 y varianza σ^2_ω.

El paso natural ahora es combinar los modelos que hemos visto, el autorregresivo (AR) y el de medias móviles (MA) en un único modelo. Este modelo es el modelo ARMA.

Definición 1.4 (Modelo Autorregresivo de Media Móvil (ARMA)). *Un modelo autorregresivo de media móvil (Autoregresive Moving Average) ARMA(p, q) es un modelo con p términos autorregresivos y q términos de media móvil de la forma:*

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + \omega_t + \theta_1 \omega_{t-1} + \theta_2 \omega_{t-2} + ... + \theta_q \omega_{t-q} \tag{1.4}$$

donde $x_1, x_2, ..., x_t$ es una serie temporal estacionaria, $\phi_p \neq 0, \theta_q \neq 0$ y y ω_t es una serie de ruidos blancos de media 0 y varianza $\sigma^2_\omega > 0$. De nuevo, si la media de x_t no es 0, entonces:

$$x_t = \alpha + \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + \omega_t + \theta_1 \omega_{t-1} + \theta_2 \omega_{t-2} + ... + \theta_q \omega_{t-q} \quad (1.5)$$

con $\alpha = \mu(1 - \phi_1 - \phi_2 - ... - \phi_p)$.

En este caso, se dice que p y q son, respectivamente, el orden autorregresivo y el orden de medias móviles.

Nótese que los modelos ARMA generalizan los modelos AR, ya que ARMA$(p, 0)$ = AR(p) $\forall p \in \mathbb{N}$, y también generalizan los modelos MA, ya que ARMA$(0, q)$ = MA(q) $\forall q \in \mathbb{N}$.

Hasta ahora se ha supuesto que la serie temporal es estacionaria, pero esto no siempre es cierto. Sin embargo, en algunas ocasiones se tiene que, si se resta a un valor de la serie el valor anterior, entonces se obtiene una nueva serie que sí es estacionaria. En otras ocasiones, habrá que realizar estas restas más de una vez. Se obtienen de esta forma los modelos ARIMA.

Definición 1.5 (Modelo ARIMA). *Un modelo ARIMA(p, d, q) (Autoregresive Integrated Moving Average) es un modelo que se obtiene de diferenciar d veces la serie temporal hasta que pasa a ser estacionaria, y después obtener un modelo ARMA(p, q) de esta nueva serie estacionaria.*

Nótese que si $d = 0$, entonces ARIMA$(p, 0, q)$ = ARMA(p, q), por lo que ARIMA generaliza tanto a ARMA, como a AR (ARIMA$(p, 0, 0)$ = AR(p)), como a MA (ARIMA$(0, 0, q)$ = MA(q).

Existen ciertos métodos para obtener el mínimo valor d para convertir la serie en estacionaria, pero no se desarrollarán estos modelos en mayor profundidad.

Se ha terminado por tanto con los modelos ARIMA, y también pues con el análisis estadístico. En la siguiente sección hablaremos del Aprendizaje Profundo y de herramientas que se pueden utilizar para analizar series temporales.

1.2 Aprendizaje Profundo

Hoy en día, la Inteligencia Artificial es un campo en constante desarrollo con muchísimas aplicaciones prácticas e investigaciones en desarrollo. Aunque ya había ciertos avances antes, el origen de la Inteligencia Artificial como disciplina se remonta a la Conferencia de Darmouth de 1956, en la que varios matemáticos y científicos debatieron sobre diversos temas, como el uso de lógica simbólica, sistemas enfocados en dominios limitados (versión primitiva de Sistemas Expertos), y diferencias entre sistemas deductivos e inductivos. Desde entonces la Inteligencia Artificial ha crecido enormemente. Se utilizan herramientas de Inteligencia Artificial en distintas tareas, como puede ser automatizar tareas rutinarias, reconocer imágenes y voces o realizar diagnósticos en medicina.

En un principio la Inteligencia Artificial se dedicó a problemas que eran difíciles intelectualmente hablando para los humanos pero relativamente simples para los ordenadores: aquellos que pueden ser descritos como una lista de reglas matemáticas formales. Un ejemplo es el sistema Deep Blue de IBM, capaz de derrotar al campeón mundial de ajedrez Garry Kasparov en 1997 [Tom03]. Actualmente se sigue avanzando en esta línea. En 2017, un sistema capaz de jugar a Go llamado AlphaGo consiguió vencer al campeón mundial Ke Jie en un enfrentamiento de tres partidas, algo que no se esperaba que se lograse, dado que el Go es un juego mucho más complejo que el ajedrez.

Sin embargo, el verdadero desafío en el que se ha centrado la Inteligencia Artificial ha sido el de resolver problemas que son sencillos para los humanos pero difíciles de describir formalmente, como puede ser reconocer caras en imágenes o reconocer palabras en una conversación hablada. Estas tareas requieren de una gran cantidad de conocimiento sobre el mundo en el que estamos, y a la vez este conocimiento es subjetivo y ni siquiera es retenido como conocimiento formal, sino que es algo intuitivo. Es por esta razón por la que se han dedicado esfuerzos en lograr que los sistemas de Inteligencia Artificial sean capaces de adquirir conocimiento propio buscando patrones en los datos que obtenga. Esta capacidad de adquirir conocimiento se conoce como Aprendizaje Automático.

El Aprendizaje Automático conlleva un gran problema: el problema de la representación. Cuando se quiere realizar un diagnóstico en medicina no se le en-

trega al sistema la historia completa del paciente, sino que se seleccionan algunos rasgos relevantes y se introducen en el sistema. La elección de estos rasgos puede ser compleja si no se sabe qué rasgos son o no relevantes a la hora de representar un elemento del campo que se estudia. En el caso de reconocer coches en imágenes, se tiene el problema de que la representación del coche varía dependiendo del ángulo desde el que lo veamos. En un ángulo determinado se podrían ver dos ruedas, mientras que en otro podría no verse ninguna.

El Aprendizaje Profundo resuelve este problema creando redes con múltiples capas. Cada una de estas capas extrae rasgos más complejos que la capa anterior basándose en lo que la anterior ha detectado. De esta forma los modelos de Aprendizaje Profundo son capaces de extraer rasgos abstractos de formas más simples (Figura 1.3).

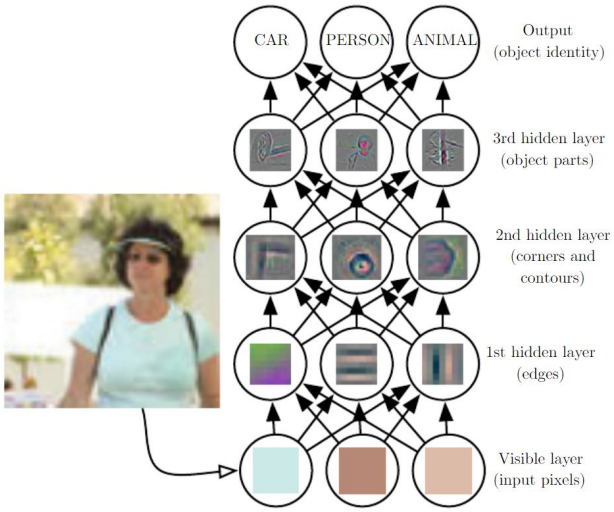


Figura 1.3: Esquema de modelo de Aprendizaje Profundo que detecta entidades en una imagen. Cada capa del modelo está conectada con la capa superior. Como se puede ver, la capa inferior extrae valores de los píxeles de la imagen, la siguiente es capaz de detectar líneas usando esos píxeles, la siguiente detecta esquinas y contornos usando las líneas, la siguiente detecta partes de objetos usando las esquinas y contornos, y la última devuelve la probabilidad de haber detectado cada entidad en base a las partes de objetos que ha detectado. Imagen extraída de [GBC16]

En esta sección se van a introducir algunos modelos de Aprendizaje Profundo y sus diferentes estructuras, propiedades y funciones. En concreto veremos Autocodificadores, Redes Neuronales Recurrentes y Redes Neuronales Convolucionales. Los autocodificadores están centrados en extraer características reduciendo la dimensionalidad de los datos y luego volviendo a reconstruirlos a partir de dichas características. Las Redes Neuronales Recurrentes están centradas en localizar dependencias a largo plazo, lo que las hace ideales para analizar datos con fuertes correlaciones temporales. Las Redes Neuronales Convolucionales están centradas en localizar características en entornos locales de datos estructurados en forma de cuadrícula.

1.2.1 Autocodificadores

Un *autocodificador* es una red neuronal que se suele utilizar en aprendizaje no supervisado, y cuya estructura se compone de dos partes principales: un *codificador* (*encoder*) y un *decodificador* (*decoder*). El *codificador* se compone de una sucesión de capas que reducen la dimensionalidad de los datos que pasan de una capa a otra según avanzan por la red, y el *decodificador* es lo contrario: se compone de una sucesión de capas que aumentan la dimensionalidad de los datos que pasan de una capa a otra según avanzan por la red (Figura 1.4). Al reducir la dimensionalidad de los datos y luego aumentarla, el autocodificador se ve obligado a extraer las características más representativas de los datos para luego reconstruirlos. Normalmente los autocodificadores se entrenan utilizando como etiquetas los mismos datos de entrada que reciben, y su objetivo es extraer la información más relevante de los datos e intentar reconstruirlos luego.

Hay que señalar que el vector que queda entre el *codificador* y el *decodificador* se denomina *vector latente* y tiene muchísima importancia. Es en este vector latente donde se concentra la información que encierran los datos de entrada, y puede dar alguna idea de qué está aprendiendo el autocodificador si se representa de forma adecuada (Figura 1.5). Los vectores latentes para cada entrada del autocodificador se encuentran en un espacio determinado, al cual llamaremos *espacio latente*.

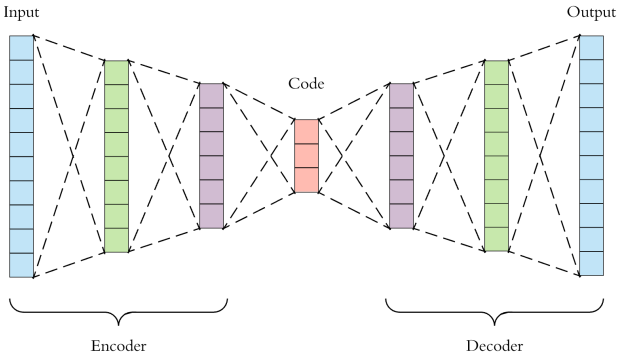


Figura 1.4: Ejemplo de autocodificador. Nótese que la dimensionalidad de los datos en la primera y la última capa es mucho mayor que en la capa que queda entre el *codificador* y el *decodificador*. Imagen extraída de [Cap].

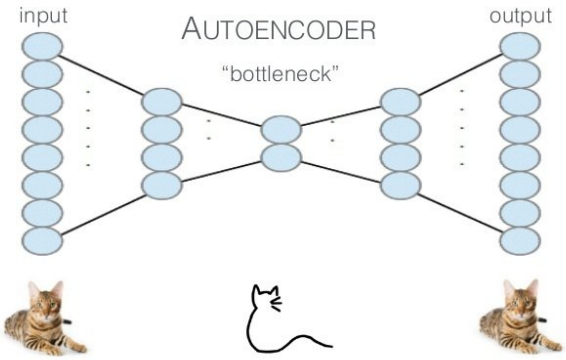
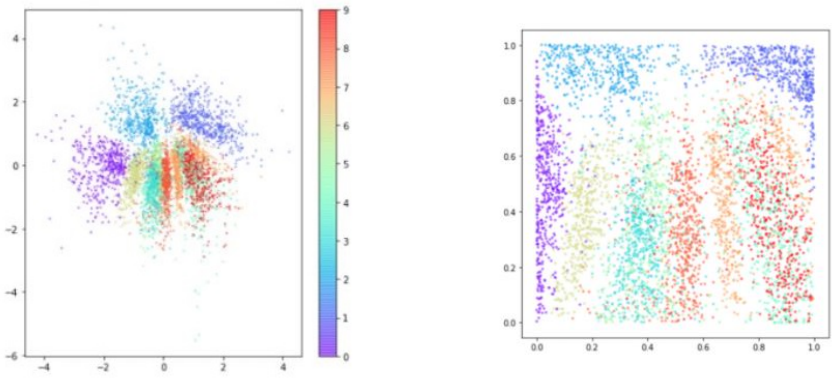


Figura 1.5: Representación del vector latente. En esta representación, tras entrenar el autocodificador con imágenes de gatos, el vector latente encierra la información que ha aprendido sobre los gatos: tienen dos orejas picudas, varios bigotes, la cabeza mucho más pequeña que el cuerpo y tienen una cola bastante larga. No presta atención por ejemplo al color ni a los ojos. Si no supiéramos qué es un gato, este vector latente daría información útil para determinarlo. Imagen extraída de [Cap].

Los autocodificadores pueden dar lugar a algunos problemas. El codificador devuelve para cada entrada un vector latente, y dado que el entrenamiento se realiza mediante retropropagación, pueden tener lugar efectos inesperados (Figura 1.6a). Puede ocurrir, por ejemplo, que imágenes que consideramos similares desde el punto de vista humano den lugar a puntos alejados en el espacio latente. Otros de los efectos que pueden darse es la presencia de grandes espacios vacíos entre los puntos o la falta de simetría . Existe otro tipo de autocodificador que soluciona estos problemas. En lugar de mapear cada entrada sobre un punto del espacio latente, estos autocodificadores las mapean a una distribución normal multivariante alrededor de un punto del espacio latente. Estos autocodificadores se denominan *autocodificadores variacionales* (Figura 1.6b).



(a) Espacio latente obtenido de un autocodificador normal.

(b) Espacio latente obtenido de un autocodificador variacional

Figura 1.6: Espacio latente de autocodificadores entrenados para reconocer dígitos utilizando la base de datos MNIST. Cada imagen de MNIST está formada por 28x28 números. Para estos autocodificadores, cada vector latente es un vector de dos dimensiones. Como se puede ver, los puntos del espacio latente están mucho mejor distribuidos en la Figura 1.6b que en la Figura 1.6a. En ocasiones se obtendrán mejores resultados si se utilizan *autocodificadores variacionales*. Imagen extraída de [GS20].

1.2.2 Redes Neuronales Recurrentes

Las *Redes Neuronales Recurrentes*, conocidas también como *Recurrent Neural Networks* o *RNNs* por su terminología en inglés, son una familia de redes neuronales que están especializadas por su estructura en el procesamiento de secuencias de datos. Reciben como datos de entrada una secuencia de vectores $x^{(1)}, x^{(2)}, ..., x^{(\tau)}$. Las RNNs son conocidas por ser capaces de procesar secuencias de longitud variable, algo que no hacen otros modelos de Aprendizaje Profundo.

Por simplicidad, se asume que las RNNs que se usan a partir de ahora operan sobre una secuencia de vectores $x^{(t)}$ con el parámetro tiempo t variando entre 1 y τ. Hay que aclarar que este parametro no tiene por qué corresponderse con unidades de tiempo reales, sino que simplemente deben representar orden. En la práctica, normalmente se trabajará con subsecuencias de tamaño variable de dicha secuencia, pero aquí se omitirán los índices de las subsecuencias para simplificar la notación.

Para representar la estructura de los distintos tipos de RNNs que se estudian en este apartado, utilizaremos grafos computacionales.

Grafos computacionales

Un *grafo computacional* es un grafo dirigido en el que los nodos representan variables, y las aristas representan operaciones. Si una variable y se obtiene aplicando una operación a una variable x, entonces existe una arista dirigida que va de x a y. Estas variables pueden ser números, vectores, matrices u otros elementos.

Si se tiene la fórmula clásica de un sistema dinámico:

$$s^{(t)} = f\left(s^{(t-1)}; \theta\right) \tag{1.6}$$

donde $s^{(t)}$ es el estado del sistema en el instante t y θ es un vector de parámetros que no evolucionan en el tiempo, entonces se tiene que el grafo computacional correspondiente al sistema es el de la Figura 1.7.

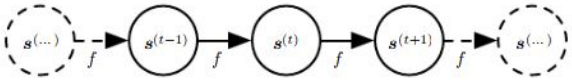


Figura 1.7: Grafo computacional correspondiente a la ecuación 1.6. En la imagen se ve cómo de cada estado se obtiene el siguiente aplicando la operación f sobre él. No se incluye θ por ser el mismo para cada operación. Imagen extraída de [GBC16].

Aunque la ecuación es recurrente por hacer referencia la definición de s en el instante t a la misma definición en el instante $t-1$, el grafo de la Figura 1.7 no parece recoger esta recurrencia. Se dice que estos grafos están *desdoblados* (o *unfolded*, por su término en inglés). La recurrencia puede recogerse usando otros grafos.

Se considera ahora un sistema dinámico que se ve influido por un vector externo al sistema $x^{(t)}$:

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}; \theta\right) \tag{1.7}$$

donde $h^{(t)}$ es el estado del sistema.

Se puede representar entonces la ecuación 1.7 como se muestra en la Figura 1.8.

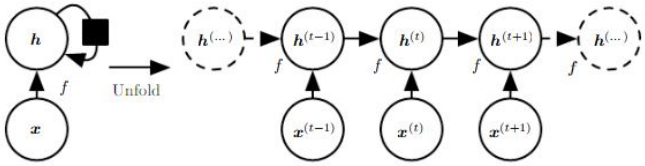


Figura 1.8: Grafos computacionales correspondientes a la ecuación 1.7. La red procesa la información que se obtiene en cada instante del vector x junto a la información del estado h. En la imagen de la izquierda se ve la versión del grafo sin desdoblar. En ella, el cuadrado negro representa el paso de un instante de tiempo. En la imagen de la derecha se ve el mismo grafo computacional desdoblado. Imagen extraída de [GBC16].

Diseños de Redes Neuronales Recurrentes

Hay varias formas de diseñar RNNs. Las más comunes son:

- Redes recurrentes que producen una salida en cada instante y que tienen conexiones recurrentes entre unidades ocultas, como la que se ilustra en la Figura 1.9.
- Redes recurrentes que producen una salida en cada instante y que tienen conexiones recurrentes únicamente de la salida en un instante a las unidades ocultas en el instante siguiente, como la que se ilustra en la Figura 1.10.
- Redes recurrentes con conexiones recurrentes entre unidades ocultas que procesan una secuencia completa antes de producir una salida, como la que se ilustra en la Figura 1.11.

El ejemplo que más usaremos a lo largo de esta sección sera el primero, referenciado en la Figura 1.9.

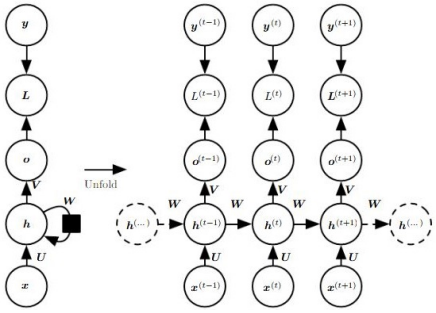


Figura 1.9: Grafos computacionales correspondientes a una RNN que produce una salida en cada instante. Se ilustra el cálculo del error de entrenamiento de una RNN que recibe como entrada una secuencia x y devuelve una secuencia de salida o. Una función L indica el error entre la salida o y el objetivo correspondiente y. La red tiene conexiones de los nodos de entrada a los nodos ocultos parametrizadas mediante una matriz de pesos U, conexiones entre nodos ocultos parametrizados por una matriz de pesos W y conexiones de los nodos ocultos a los de salida parametrizados por una matriz de pesos V. Imagen extraída de [GBC16].

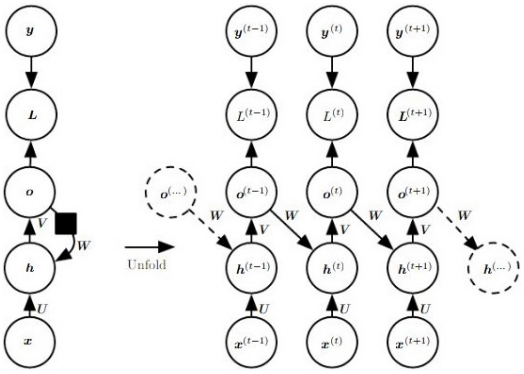


Figura 1.10: Grafos computacionales de una RNN similar a la de la Figura 1.9. La gran diferencia es que ahora la recurrencia se da en la conexión entre la salida y la capa oculta. Esta RNN es menos potente que la anterior, ya que a no ser que la salida sea dimensionalmente enorme y muy rica en información, no se transmite tanta información de la salida a la capa oculta como de la capa oculta a sí misma entre iteraciones. Sin embargo, al poder ser entrenada en paralelo utilizando como valores de salida los valores y de entrenamiento, su entrenamiento es mucho más rápido. Imagen extraída de [GBC16].

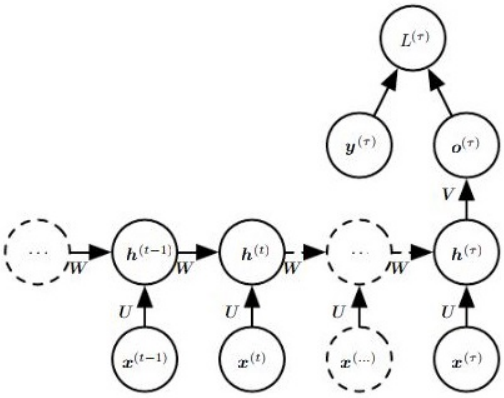


Figura 1.11: Grafo computacional correspondiente a una RNN con una única salida al final de la secuencia. Este tipo de redes suele utilizarse para resumir una secuencia y dar una representación de tamaño fijo que pueda usarse para posterior procesamiento. Imagen extraída de [GBC16].

Hay otras arquitecturas de Redes Neuronales Recurrentes especializadas en problemas más concretos. En problemas en los que el contexto es importante, como puede ser el reconocimiento del habla, se suelen utilizar las llamadas RNN bidireccionales. En problemas en los que el tamaño de la salida puede variar en función de la entrada, como en traducción automática, se suelen utilizar RNNs *codificador-decodificador*. En problemas en los que es importante recordar información obtenida de datos que se introdujeron mucho antes, como también ocurre en traducción automática, se utilizan RNNs de tipo *Long Short-Term Memory* (LSTM). Se explicarán a continuación estos tres tipos de RNNs.

Redes Neuronales Recurrentes Bidireccionales

Hasta ahora, las RNNs que se han visto asumen un cierto orden en la estructura de los datos de entrada: cada dato en un instante t se ve influenciado por los datos anteriores, pero no por los siguientes. ¿Qué ocurre si se quieren estudiar secuencias en los que los datos se ven influidos tanto por los anteriores como por los siguientes?

Un problema de este tipo puede ser el de entender una frase. Para comprender el significado de una frase generalmente es necesario escuchar todas las palabras antes de responder, especialmente en idiomas como el alemán, en los que el verbo suele ocupar la última posición.

Este problema puede resolverse con las llamadas RNNs bidireccionales. Estas redes combinan una RNN que se mueve hacia delante en el tiempo con otra que se mueve hacia atrás, de modo que ambas influyen en la salida (Figura 1.12).

Esta idea podría extenderse a imágenes, combinando cuatro RNNs de modo que cada una se desplace en una dirección: arriba, abajo, izquierda y derecha. Se tiene que las RNNs de este estilo son más costosas de entrenar que las redes convolucionales, pero permiten asociar conceptos que pueden estar demasiado lejos en la imagen como para que una red convolucional los asocie.

A pesar de sus ventajas, las RNNs bidireccionales también tienen una limitación muy importante: es necesario conocer por completo la secuencia para poder procesarla. Si se trabaja con un sensor que lee datos continuamente, estas redes

no podrían utilizarse a no ser que se decidiera fragmentar la secuencia de datos en trozos de una longitud determinada.

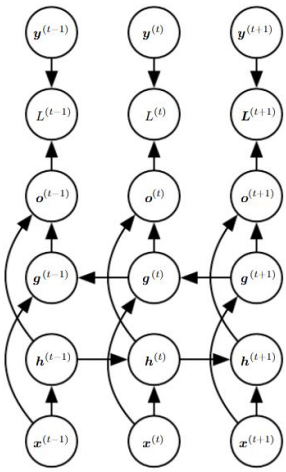


Figura 1.12: Grafo computacional correspondiente a una RNN bidireccional. La capa oculta h propaga información hacia delante en el tiempo (hacia la derecha), mientras que la capa oculta g la propaga hacia atrás (hacia la izquierda). En cada momento t, la salida $o^{(t)}$ se ve influenciada tanto por $h^{(t)}$ como por $g^{(t)}$. Imagen extraída de [GBC16].

RNNs *encoder-decoder*

Hasta ahora, se ha trabajado con RNN que devolvían un dato de salida por cada dato de entrada. Esto puede ser un problema a la hora de enfrentarse a ciertos desafíos, como puede ser traducir una frase a otro idioma (no tienen por qué tener el mismo número de palabras) o responder a preguntas (una pregunta podría constar de veinte palabras y responderse con un simple "No.").

Teniendo esto en mente, se crean las Redes Neuronales Recurrentes *codificador-decodificador*, también llamadas *secuencia a secuencia* (*sequence-to-sequence*). Estas redes se componen de una parte que "codifica" los datos de entrada en un *contexto* $\boldsymbol{C}$, y posteriormente "decodifica" este contexto $\boldsymbol{C}$ utilizando una segunda parte, de forma similar a como hacía el autocodificador (Figura 1.13). En el

contexto de las series temporales, se podría introducir en una RNN de este tipo un fragmento de una serie temporal de una longitud determinada y obtener como dato de salida una secuencia del tamaño que se elija sin necesidad de obtener un dato de salida por cada dato de entrada. Aunque no se usará directamente esta red, sí forma parte del fundamento de este trabajo.

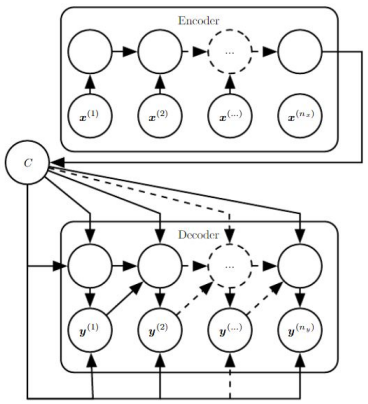


Figura 1.13: Grafo computacional correspondiente a una RNN *codificador-decodificador*. Para generar una secuencia de salida y dada una secuencia de entrada x, primero hay que introducir la secuencia de entrada x en el codificador, obtener el contexto C, y después introducir el contexto C en el decodificador para obtener la secuencia de salida y. Imagen extraída de [GBC16].

Long Short-Term Memory

Las Long Short-Term Memory (LSTM) son Redes Neuronales Recurrentes que pertenecen a una familia especial: las RNNs con puertas (*gated RNNs*). Estas RNNs se llaman así porque se basan en unidades que están controladas por una especie de "puertas" como las puertas lógicas de los circuitos electrónicos. Cada "célula" de una LSTM incluye tres puertas: una puerta de entrada que controla si va a leerse o no el dato de entrada (*input gate*), una puerta de olvido que va a controlar si se olvida o no lo aprendido por la célula (*forget gate*) y una puerta de salida que controla si se va a devolver o no una salida en cada instante (*output gate*) (Figura 1.14).

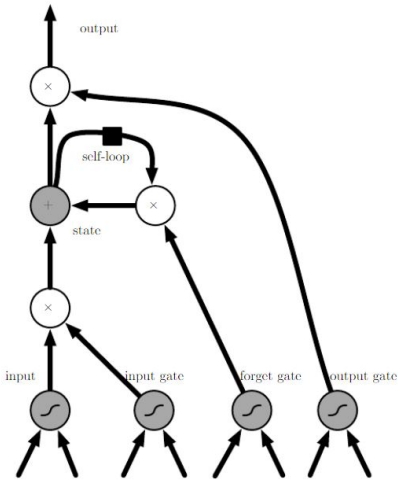


Figura 1.14: Grafo computacional correspondiente a una LSTM. Las células están conectadas de forma recurrente unas con otras y reemplazan a los nodos ocultos que hemos visto hasta ahora. Como podemos ver en el esquema, las tres puertas descritas controlan el flujo de datos tanto de entrada, como de estado de la propia célula, como de salida. Imagen extraída de [GBC16].

Una ventaja que aportan las LSTM frente a otras RNNs es que solucionan el problema de desaparición del gradiente (*vanishing gradient problem*). Este problema se refiere al hecho de que, cuando se dan dependencias a largo plazo, las derivadas que se necesitan para calcular los gradientes van acercándose cada vez más a cero según avanzan por la red. Este problema se resuelve al utilizar las puertas de olvido, ya que la información no tiene por qué perderse a largo plazo si las células LSTM almacenan esta información y la propagan usando las puertas de olvido. Este problema también puede resolverse para otros modelos siguiendo ciertas estrategias, como puede ser la eliminación de algunas conexiones entre capas. En [GBC16] puede encontrarse más información sobre el tema.

1.2.3 Redes Neuronales Convolucionales

Las redes neuronales convolucionales, también conocidas como *convolutional neural networks* o *CNN* por su terminología en inglés, son un tipo de redes neuronales que procesan datos cuya estructura es la de una cuadrícula. Los ejemplos más corrientes de datos sobre los que se trabaja con CNN son imágenes y vídeos, que pueden representarse como una cuadrícula de píxeles de dos y tres dimensiones respectivamente, y las series temporales, que pueden representarse como una cuadrícula de una dimensión. Las CNNs deben su nombre a una operación matemática: la convolución. La convolución es un tipo de operación lineal que definiremos a continuación.

En esta sección se justificará el uso de CNN exponiendo las mejoras que llevan a la práctica frente a las redes neuronales tradicionales. Se verán también otras operaciones que se suelen utilizar en CNNs, como son la agrupación (*pooling*) o el relleno (*padding*). Por último, se expondrán ejemplos de distintas arquitecturas de CNN que se aplican en diferentes problemas.

Convolución

Como se ha dicho al principio de esta sección, la convolución es una operación lineal especializada. Es una operación que se define sobre funciones como sigue:

Definición 1.6 (Convolución). *Dadas dos funciones* $f : \mathbb{R} \longrightarrow \mathbb{R}$ *y* $g : \mathbb{R} \longrightarrow \mathbb{R}$*, la convolución de* f *y* g*, denotada por* $f * g$*, es otra función* $(f * g) : \mathbb{R} \longrightarrow \mathbb{R}$ *definida como:*

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(y)g(x-y)dy \tag{1.8}$$

Siguiendo la terminología común, al primer argumento se le llama ***entrada****, y al segundo* ***núcleo*** *de la convolución. El caso discreto se define de forma similar:*

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) \tag{1.9}$$

En el Aprendizaje Profundo sólo se utiliza el caso discreto. Normalmente los núcleos son funciones que toman valores nulos en todo $\mathbb{Z}$ excepto en un conjunto finito de valores, llamado *soporte* de f y denotado por $Sop(f)$. Además, en la práctica se utiliza una variante de la convolución llamada *correlación cruzada*:

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) = \sum_{m=-\infty}^{\infty} f(m)g(n+m) \tag{1.10}$$

En el caso de las convoluciones en dos dimensiones (Figura 1.15), podemos escribir su fórmula como:

$$(f * g)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m,n)g(i+m, j+n) \tag{1.11}$$

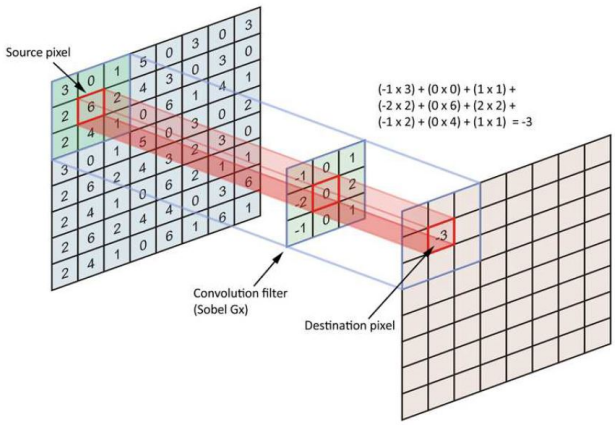


Figura 1.15: Ejemplo de convolución en 2D. La matriz de la izquierda se corresponde con una imagen que toma el papel de entrada de la convolución, la matriz del centro toma el papel de núcleo de la convolución y en la matriz de la derecha se obtiene el resultado de la convolución. Imagen extraída de [GS20].

Motivación de las Redes Neuronales Convolucionales

Las convoluciones cubren tres ideas que pueden ayudar a mejorar un sistema de aprendizaje automático: conectividad dispersa (*sparse connectivity*), parámetros compartidos (*parameter sharing*) y representaciones equivariantes (*equivalent representations*).

- **Conectividad dispersa:** Las redes neuronales tradicionales normalmente conectan cada unidad de entrada con cada unidad de salida, haciendo que la cantidad de parámetros necesaria para entrenar la red sea inmensa para redes muy profundas. Las redes convolucionales, sin embargo, trabajan con núcleos más pequeños que las entradas, de modo que cada unidad de entrada está conectada a una pequeña cantidad de unidades de salida. Esto hace que la cantidad de parámetros necesaria sea menor y que se requieran por tanto menos operaciones a la hora de computar la salida.
- **Parámetros compartidos:** Al usar el mismo núcleo de convolución en toda la entrada se tiene que, en lugar de usar cada parámetro una única vez como en las redes neuronales tradicionales, se utiliza cada parámetro del núcleo de convolución múltiples veces. Esto hace que solamente haya que almacenar los parámetros de dicho núcleo, reduciendo aún más la cantidad de parámetros que hay que almacenar.
- **Representaciones equivariantes:** Como consecuencia de usar parámetros compartidos, se tiene que se consigue equivarianza de representación. Esto significa que, si se aplica una función que modifica la entrada antes de aplicar la convolución, se obtiene el mismo resultado que si se aplica primero la convolución y después la función. Por poner un ejemplo, si se realiza una traslación en la entrada, entonces el resultado de aplicar la convolución coincide con el resultado de realizar una traslación sobre la convolución de la entrada original.
  Como consecuencia, una característica como la oreja de un gato será reconocida independientemente de la posición en la que aparezca en la imagen.

Operaciones características de la CNN

A la hora de construir Redes Neuronales Convolucionales es conveniente utilizar otras operaciones además de la convolución. Como ejemplos de estas operaciones se tienen el relleno (*padding*), la agrupación (*pooling*), el paso (*stride*) y la normalización por lotes (*batch normalization*).

- **Relleno:** Dado que los núcleos de las convoluciones suelen ser matrices, para realizar una convolución es necesario tomar como entrada de la convolución una matriz centrada en un dato. Esto hace que sea imposible realizar convoluciones sobre los elementos que estén en el borde de la estructura de los datos. Para evitar esto, lo que se hace es rellenar el borde con ceros hasta que sea posible construir matrices centradas en cada uno de los datos (Figura 1.16).

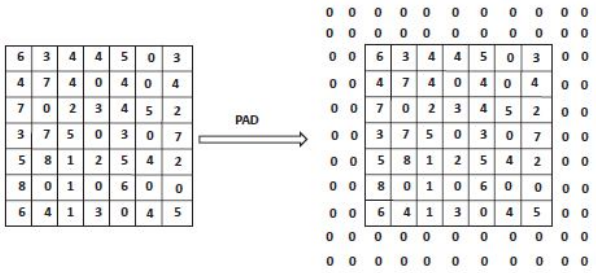


Figura 1.16: Ejemplo de relleno para un núcleo de 5x5. Al añadir dos filas y columnas de ceros en los bordes, ahora se puede tomar cada elemento de la entrada como centro a la hora de realizar la convolución. Imagen extraída de [GS20].

- **Agrupación:** Una función de agrupación es una función estadística que modifica la salida de la red en una zona determinada realizando una especie de resumen de las salidas de su entorno. Algunos ejemplos de funciones de agrupación son la función que obtiene el valor máximo de un entorno o la que obtiene el valor medio. El objetivo de estas funciones es reducir progresivamente el tamaño de la representación y lograr así que la representación sea invariante a pequeñas traslaciones. También es posible que sea invariante a giros si se usan filtros suficientes (Figura 1.17).

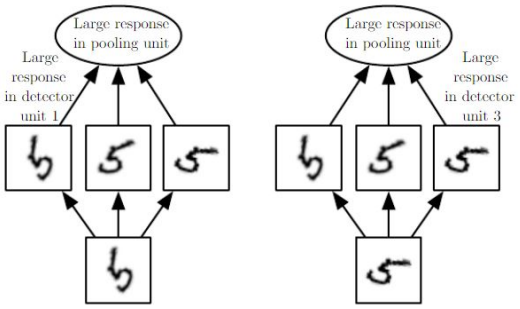


Figura 1.17: Ejemplo de agrupación. En la imagen se tiene el esquema de una CNN con tres filtros y una unidad de agrupación. Si alguno de los tres filtros se activa, entonces la unidad de agrupación lo detecta. En la imagen de la izquierda un filtro detecta un 5 girado en sentido antihorario y la unidad de agrupación se activa. En la imagen de la derecha, un filtro detecta un 5 girado en sentido horario y la unidad de agrupación se activa. Imagen extraída de [GBC16].

- **Paso:** En ocasiones se suele incluir un parámetro de paso (*stride*) que indica el salto que realiza el núcleo a la hora de desplazarse. Cuando el paso es mayor que 1, entonces la operación de convolución deja de aplicarse para todos los elementos de la entrada, y por tanto se produce una reducción de la dimensionalidad. En ocasiones puede ser muy útil usar un paso mayor que 1, ya que implica una reducción en la cantidad de cálculos que deben realizarse sin que se pierda tanta información (Figura 1.18).

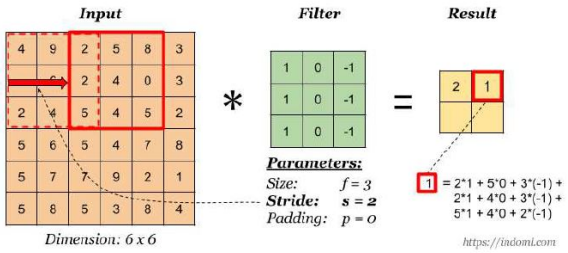


Figura 1.18: Ejemplo de paso. En la imagen vemos una matriz de entrada de 6x6 y un núcleo de 3x3. En lugar de aplicarse tantas convoluciones como se pueden utilizando el núcleo, se decide utilizar un paso de valor 2 para que el núcleo se aplique la mitad de veces en cada dirección. Se obtiene así una matriz de 2x2 en lugar de una matriz de 4x4. Imagen extraída de [Pri].

- **Normalización por lotes:** Para realizar descenso por el gradiente es importante normalizar los datos, ya que los pesos de cada capa dependen directamente de los valores de entrada. En ocasiones puede ocurrir que los pesos crezcan demasiado. Este suceso se denomina problema de explosión del gradiente (*exploding gradient problem*). Para evitarlo, lo que se hace es normalizar la salida de cada capa por cada lote de datos que introduzcamos. Este problema se trata en mayor profundidad en [IS15].

Las CNN que se utilizan normalmente se componen de varias sucesiones de convoluciones a las que se aplican rellenos, agrupaciones, pasos y normalizaciones en cada capa.

Ejemplos de arquitecturas de CNN

Algunos ejemplos de redes convolucionales son los siguientes:

- **Lenet.** Está reconocida como la primera CNN, y se compone de dos bloques de convolución y agrupación seguidas de dos capas completamente conectadas. Se utilizó para clasificar dígitos, y su creador publicó junto a la red el conjunto de datos MNIST (Figura 1.19).

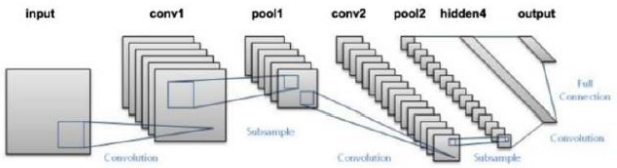


Figura 1.19: Arquitectura de Lenet. Nótese que cada capa de convolución (conv) y agrupación (pool) incluye varios núcleos. Imagen extraída de [GS20].

- **VGG16 y VGG19.** Estas redes se utilizaron para mostrar que redes más simples pero más profundas obtenían mejores resultados que otras redes más complejas de menor profundidad (Figura 1.20). Se llaman así porque una tiene 16 capas y la otra 19. Ambas usan núcleos de 3x3 y agrupaciones de 2x2.

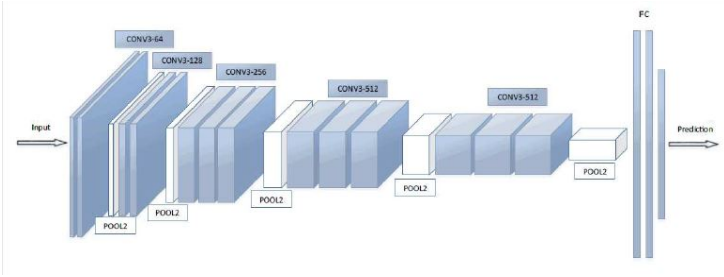


Figura 1.20: Arquitectura de VGG16. Nótese que se tienen 13 capas de convoluciones y 3 capas completamente conectadas entre sí. Imagen extraída de [GS20].

- **U-Net.** Esta red sirve de inspiración para la red U-Time que se estudia en el capítulo 2. Su objetivo era segmentar imágenes biomédicas y fue desarrollada por el Departamento de Ciencias de la Computación de la Universidad de Freiburg, Alemania (Figura 1.21). Su estructura es la de un autocodificador como el que estudiamos en la sección 1.2.1 que se construye combinando capas convolucionales y agrupaciones para construir un codificador, y capas convolucionales y capas *up-sample* que rellenan con ceros las matrices para aumentar su dimensionalidad y construir así un decodificador.

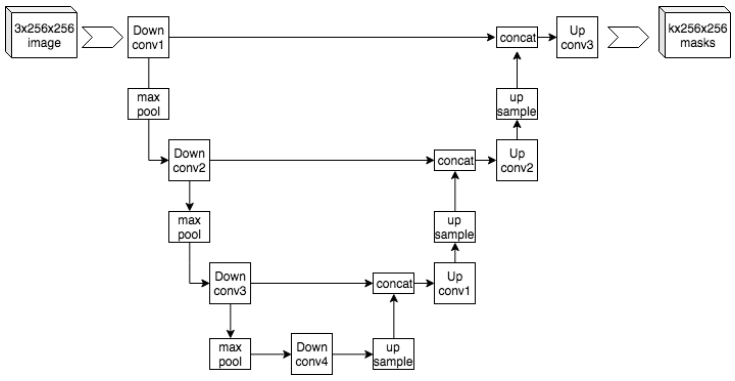


Figura 1.21: Arquitectura de U-Net. Su nombre se debe a que su estructura puede representarse como una U, en la que el brazo izquierdo lo compone el codificador y el derecho el decodificador. Imagen extraída de [GS20].

1.3 Predicciones sobre el comportamiento de Series Temporales utilizando Análisis de Series Temporales y Aprendizaje Profundo

Se han visto pues herramientas estadísticas (modelos ARIMA) y herramientas del Aprendizaje Profundo (LSTM y otras RNNs, Redes Convolucionales, etc.) que se han utilizado para trabajar sobre series temporales, pero, ¿supone una gran diferencia trabajar con unas u otras? Se comentarán a continuación algunos artículos sobre el tema.

- **Time-series analysis with neural networks and ARIMA-neural network hybrids.** En este artículo de 2003 [HN03], los autores han desarrollado modelos híbridos entre ARIMA y redes neuronales que incorporan retrasos y los han aplicado sobre datos económicos de EEUU. Para construir estos modelos híbridos, han utilizado ARIMA para determinar el tamaño de las ventanas que van a utilizar las redes neuronales, y después han intentado aprender los datos con dichas redes. Los resultados con dichas redes híbridas son algo mejores que utilizando únicamente ARIMA (Figura 1.22).

Time series	Error measure	ARIMA	Recurrent BP	Hybrid BP	Recurrent RBF	Hybrid RBF
US retail sales	RMSE	12.24	20.93	**9.94**	18.01	12.96
	MAD	10.42	16.52	**7.12**	15.57	12.05
Currency component of money supply	RMSE	8.38	5.48	**5.26**	8.85	8.53
	MAD	7.80	4.64	**4.30**	7.17	7.07
Reserves in banking system	RMSE	6.95	5.86	**5.16**	6.21	6.19
	MAD	3.77	4.32	3.91	3.78	**3.71**
Travellers' cheques	RMSE	0.22	0.16	0.14	0.17	**0.13**
	MAD	0.18	0.13	**0.11**	0.13	**0.11**
Treasury deposits at Federal Reserve	RMSE	**0.63**	0.94	0.68	0.83	0.73
	MAD	**0.43**	0.89	0.47	0.65	0.44

Figura 1.22: Comparación entre la raíz del error cuadrático medio (RMSE) y la desviación media absoluta (MAD) obtenidos por ARIMA y modelos híbridos aplicados a datos económicos de EEUU. Imagen extraída de [HN03].

- **A Comparison of ARIMA and LSTM in Forecasting Time Series.** En este artículo de 2018 [STS18], los autores han aplicado ARIMA y LSTM sobre datos financieros para ver qué modelos obtenían un mayor rendimiento. Las conclusiones que se extraen son que las LSTM superan en rendimiento a los ARIMA en aproximadamente un 85 % de media a la hora de trabajar sobre estos datos (Figura 1.23).

Stock	RMSE		% Reduction in RMSE
	ARIMA	LSTM	
N225	766.45	105.315	-86.259
IXIC	135.607	22.211	-83.621
HSI	1,306.954	141.686	-89.159
GSPC	55.3	7.814	-85.869
DJI-Monthly	516.979	77.643	84.981
DJI-Weekly	287.6	30.61	-89.356
Average	511.481	64.213	-87.445
MC	0.81	0.801	-1.111
HO	0.522	0.43	-17.624
ER	1.286	0.251	-80.482
FB	0.478	0.397	-16.945
MS	30.231	3.17	-89.514
TR	2.672	0.569	-78.705
Average	5.999	0.936	-84.394

Figura 1.23: Tabla comparativa entre LSTM y ARIMA aplicados a datos financieros que van de 1985 a 2018. Imagen extraída de [STS18].

- **Carbon futures price forecasting based with ARIMA-CNN-LSTM model.** En este artículo de 2019 [Ji+19] se intenta predecir el precio de futuros del carbón utilizando un modelo híbrido de ARIMA, CNN y LSTM. El modelo híbrido supera ligeramente a ARIMA (en torno a un 1 %) y a LSTM (en torno a un 5 %), y con gran diferencia a CNN (en torno a 19 % en RMSE y a 32 % en porcentaje de error medio absoluto (MAPE). (cuadro 1.1).

	CNN	LSTM	ARIMA	ARIMA-CNN-LSTM
RMSE	0.8616	0.7251	0.7015	**0.6940**
MAPE	0.0623	0.0456	0.0423	**0.0421**

Cuadro 1.1: Tabla comparativa entre los resultados obtenidos por ARIMA, LSTM, CNN y ARIMA-CNN-LSTM aplicados a los futuros del carbón de 2008 a 2019. Tabla extraída de [Ji+19].

2 U-Time

U-Time [Per+19] es una variante de un autocodificador ya existente, U-Net [RFB15]. U-Net nació con el objetivo de segmentar imágenes biomédicas, y U-Time sigue la misma filosofía en cuanto a segmentar imágenes. La diferencia fundamental entre U-Net y U-Time es que U-Time utiliza como entradas imágenes construidas a partir de señales de estudios polisomnográficos. Estas señales no son más que series temporales, por lo que a priori podrían construirse imágenes con cualquier otra serie temporal y utilizar la misma estructura para trabajar con ellas. Es por esta razón por la que se ha decidido estudiar U-Time en profundidad y utilizarla para analizar series temporales distintas a las de los estudios polisomnográficos.

En esta sección se va a ver por tanto el problema que trata de resolver U-Time, su funcionamiento, su estructura y los resultados obtenidos.

2.1 Introducción al problema real

Cuando duerme, el cerebro humano pasa por distintas fases que pueden distinguirse según la actividad cerebral y corporal. Estas fases se llaman *fases del sueño* (*sleep stages*), y pueden determinarse midiendo la actividad neuronal del córtex cerebral (a través de electroencefalografías, EEG), los movimientos de los ojos (a través de electrooculografías, EOG) y la actividad de músculos faciales (a través de electromiografías, EMG) en un estudio polisomnográfico (PSG). Conocer estas fases puede ser útil en medicina para ayudar a la detección de desórdenes relacionados con el sueño. Un ejemplo de PSG es el de la Figura 2.1.

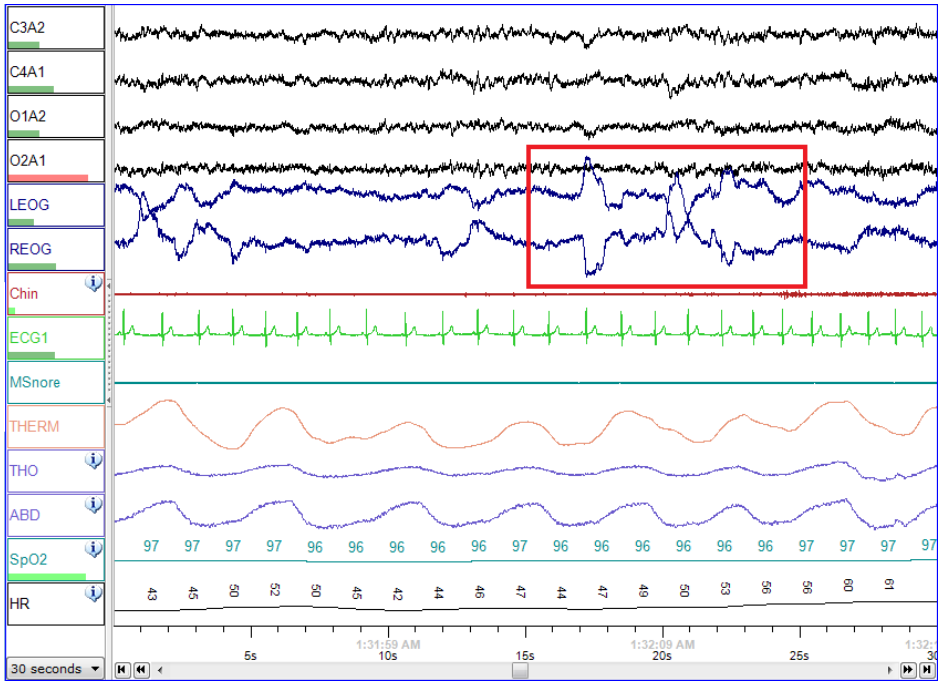


Figura 2.1: Ejemplo de un estudio polisomnográfico (PSG). La encefalografía (EEG) se compone de las 4 primeras señales. La señal EOG está dividida en dos, una para cada ojo, LEOG y REOG. Un ejemplo de EMG es la señal que mide el movimiento de la barbilla (Chin). Es importante señalar que en cada momento tenemos múltiples señales, y que estas son muy ruidosas. Imagen extraída de [Wik].

Para clasificar las señales PSG, normalmente éstas se segmentan en fragmentos de 30 segundos y después se clasifican dichos fragmentos a mano. Nótese que fragmentar señales de varios canales que duran entre 8 y 24 horas en segmentos de 30 segundos y después clasificarlos es una tarea difícil y tediosa, por lo que conviene enormemente acelerar este proceso sin perder precisión.

Afortunadamente, se ha avanzado bastante en la automatización de este proceso. Entre todas las herramientas que se han desarrollado, el Aprendizaje Profundo ha tomado un papel especialmente relevante en el análisis de series temporales fisiológicas y ya ha sido aplicada a las fases del sueño previamente. Tal y como se comentaba en el Capítulo 1, aunque las Redes Neuronales Recurren-

tes (RNN) parecen ser bastante prometedoras en el análisis de series temporales, normalmente son difíciles de optimizar y ajustar en la práctica, y se ha probado que pueden ser sustituidas por sistemas prealimentados (*feed-forward*) sin perder apenas precisión [BKK18; CW17; Vas+17]. Ahora mismo, los mejores resultados se han obtenido de redes formadas por combinaciones de capas convolucionales y recurrentes [Sup+17]. En la Figura 2.2 se incluye una de las estructuras utilizadas.

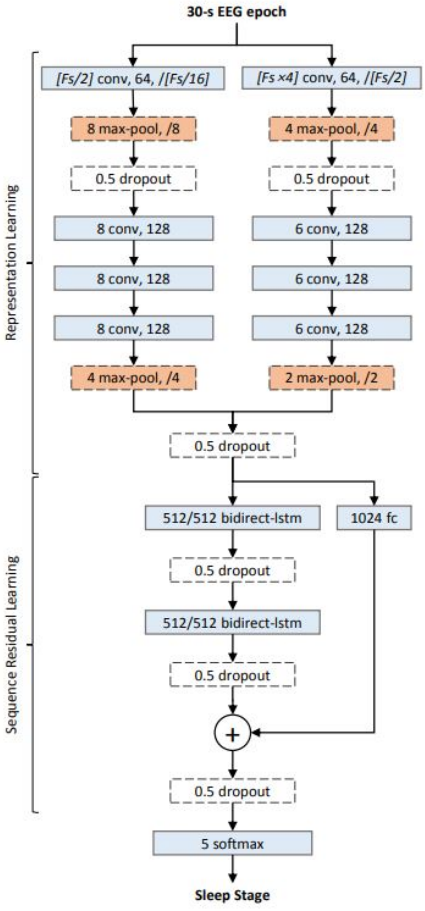


Figura 2.2: Estructura de la red que se utiliza en [Sup+17]. La parte superior se compone de capas convolucionales y la inferior de LSTM (capas recurrentes) bidireccionales. Imagen extraída de [Sup+17].

2.2 Funcionamiento de U-Time

U-Time se compone de dos partes principales: un autocodificador y un clasificador de segmentos. U-Time asigna a cada punto de la señal una clase tras atravesar el autocodificador, y después agrupa estas clases utilizando el clasificador de segmentos. Se incluye un ejemplo ilustrativo en la Figura 2.3.

Sea $x \in \mathbb{R}^{\tau S x C}$ una señal fisiológica con C canales que está muestreada a una tasa S durante τ segundos. Sea e la frecuencia a la que queremos segmentar la señal x. Teniendo x y e, se tiene que el objetivo es por tanto asignar a x un total de $\lfloor \tau \cdot e \rfloor$ etiquetas, las cuales se deciden considerando $i = S/e$ puntos de la señal. Como se dijo en el apartado anterior, normalmente se seleccionan intervalos de 30 segundos ($e = 1/30$ Hz).

La entrada x de U-Time debe estar compuesta por T segmentos consecutivos de longitud fija i. U-Time, por su estructura, predecirá las T etiquetas (una por cada segmento) a la vez, lo que significa que U-Time procesa realmente señales unidimensionales de longitud $t = Ti$ por cada canal. Técnicamente, en realidad lo que se hace es asignar a cada segmento una probabilidad por cada una de las K clases. U-Time devuelve para cada segmento la clase que tenga mayor probabilidad.

En otras palabras, una vez que se decida cuál va a ser la longitud de los segmentos de la señal a etiquetar ($1/e$), entonces se va a obtener una etiqueta por cada $1/e$ segundos. Si se obtienen S puntos de la señal por segundo, entonces cada segmento tiene $i = S/e$ puntos. Si se quiere etiquetar T segmentos, entonces se procesan señales de longitud $t = Ti$ y la red devuelve T valores seleccionados entre las K clases: la clase más probable para cada segmento. Las clases son W (despierto), N1, N2, N3 (tres fases no-REM) y R (fase REM), por lo que $K = 5$.

Es importante señalar que e es variable, y que la única razón para usar $e = 1/30$ es que es el valor que se suele utilizar. En el caso extremo en que $S = e$, entonces se intentaría asignar una etiqueta a cada punto de la señal, aunque esto puede convertirse en un problema inabordable por la cantidad de ruido que tienen la señales. También hay que señalar que, tras pasar el autocodificador, cada punto de la señal tendrá una clase asignada. Esto da gran flexibilidad a la

red, ya que si en algún momento se decidiera cambiar e teniendo la red entrenada, entonces sólo habría que modificar el clasificador que se encuentra después del autocodificador (Figura 2.3).

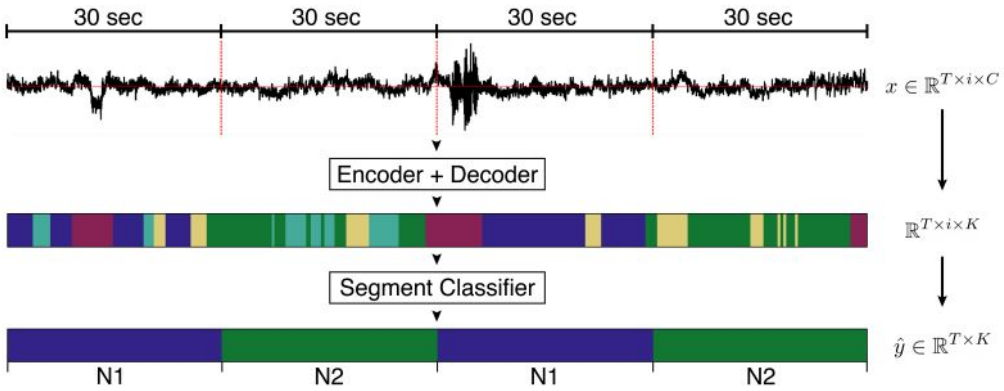


Figura 2.3: Esquema del funcionamiento de U-Time. Primero, se segmenta la señal en trozos de $1/e$ segundos. Después, se introducen en el autocodificador y se obtiene una etiqueta para cada punto de la señal. La salida del autocodificador se introduce en el clasificador de segmentos, el cual selecciona la etiqueta que aparece más veces en cada segmento y la asigna al segmento. Si la tasa e cambiara, sólo habría que modificar el tamaño de la entrada del clasificador para hacer los segmentos más grandes o más pequeños. Imagen extraída de [Per+19].

2.3 Estructura de U-Time

La estructura de U-Time se compone de un autocodificador y un clasificador de segmentos. El autocodificador está formado por un codificador de cuatro bloques y un decodificador de otros cuatro bloques siguiendo la estructura en U de U-Net [RFB15]. Cada uno de los bloques del codificador incluye varias capas convolucionales, normalizaciones por lotes y agrupaciones, mientras que los bloques del decodificador incluyen capas convolucionales, capas *up-sample* y normalizaciones por lotes (Figura 2.4). Se va a desarrollar cada uno de esos bloques a continuación:

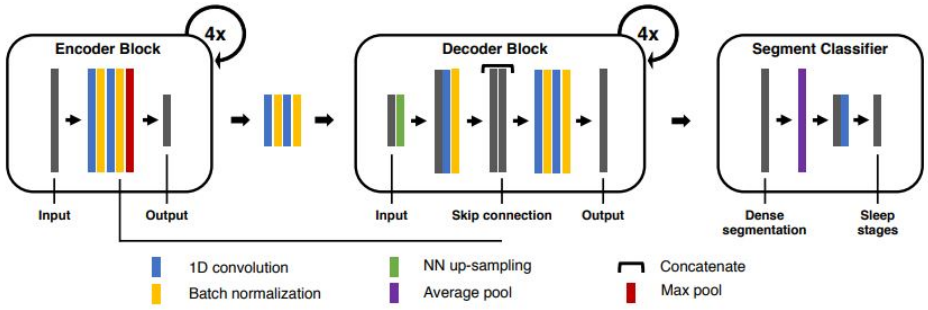


Figura 2.4: Esquema de la estructura de U-Time. Se tiene el codificador primero, compuesto por 4 bloques convolucionales. A continuación, se tienen dos convoluciones y dos normalizaciones. Después, los 4 bloques del decodificador. Por último, se encuentra el clasificador de segmentos. Imagen extraída de [Per+19].

Codificador: El codificador está compuesto por cuatro bloques convolucionales (capas 3 a 14 de la Figura 2.5). Cada vez que se utilice una convolución, se tendrá cuidado en preservar la dimensionalidad rellenando con ceros (*zero padding*) tal y como se explicó en la sección 1.2.3. En cada bloque se aplicarán dos convoluciones consecutivas con núcleos de tamaño 5x1 dilatados para que tengan anchura 9, tal y como se aconseja en [YK15]. Después de estas dos convoluciones, se aplica una normalización por lotes y después se aplica una agrupación seleccionando el máximo. Esta agrupación divide los datos en porciones de una longitud dada, y extrae el máximo de cada porción, reduciendo cada porción a un único número. Esas longitudes son 10, 8, 6 y 4 respectivamente. Una vez completados los 4 bloques, se aplican dos convoluciones más a los datos. Nótese que los núcleos son filas y no matrices cuadradas como las que se suelen encontrar en las redes convolucionales clásicas. Esto tiene cierto sentido, ya que sería extraño que pudiendo elegir la anchura de los segmentos, afectasen de igual forma los valores que están justo antes o después en la serie temporal que los que están en la misma posición en otro segmento.

Al aplicar las agrupaciones se va reduciendo la dimensión de los datos. En total, tras pasar las reducciones de factores 10, 8, 6 y 4, se reduce la cantidad de datos en un factor de 1920. Esto hace que se cumpla la filosofía básica del autocodificador: reducir la cantidad de datos que salen del codificador para forzar la

extracción de características de los datos, y forzar por tanto el aprendizaje. Además, agiliza la computación, y debido a la dilatación de los núcleos se aumenta la cantidad de datos que influyen en un determinado punto del vector latente.

Decodificador: El decodificador está compuesto también por cuatro bloques convolucionales traspuestos (capas 17 a 35 de la Figura 2.5). El adjetivo "traspuesto" señala que van a usarse bloques convolucionales, pero que estos bloques restauran los datos de entrada a la dimensión que tenían antes de pasar por los bloques del codificador. Cada bloque del decodificador se compone de una capa que realiza primero un aumento de la dimensión de los datos de entrada seleccionando el vecino más cercano para los datos que se generan (*nearest-neighbour up-sampling*) y después se aplican convoluciones con tamaños de núcleo 4, 6, 8 y 10 respectivamente. Por último, despúes de cada capa convolucional se aplica una normalización por lotes. Todo este proceso lo que hace es restaurar los datos a su dimensión original, volviéndo a aumentar la dimensión de los datos en un factor de 1920.

Una vez pasados los cuatro bloques, se aplica una última convolución punto a punto de K núcleos (uno por cada clase, de tamaño 1) que transforma cada punto de la señal introducida en K valores, cada uno representando la probabilidad de que dicho punto pertenezca a la fase del sueño K.

Se podría pensar que, teniendo una red que clasifica las señales punto a punto, podríamos empezar a entrenarla ya. Sin embargo, como se dijo previamente, lo que se clasifican son segmentos de la red (normalmente de 30 segundos, y a 100Hz eso se corresponde con 3000 puntos). Por tanto, aún hay que añadir el clasificador de segmentos justo detrás del autocodificador para poder empezar a entrenar la red. Se utilizarán como etiquetas las clasificaciones de los segmentos.

Clasificador de segmentos: Si los segmentos que hay que clasificar cuentan con i datos, entonces el clasificador de segmentos (capas 36 a 39 de la Figura 2.5) extrae la media de las i predicciones (*mean pooling*) de cada segmento para cada clase, y después aplica una convolución punto a punto (tamaño de núcleo 1). Esto obliga a transformar los datos obtenidos del autocodificador (dimensión t x K) en etiquetas para cada segmento (dimensión T x K). Se obtienen así los datos de dimensión TxK y se puede por tanto entrenar la red (Figura 2.6).

ID	Layer Type	Output dim	Kernel	Filters	Activation	Pad
1	Input	$35 \times 3000 \times 1$	-	-	-	-
2	Reshape	105000×1	-	-	-	-
3	Convolution $\rightarrow$ BN	105000×16	5	16	ReLU	same
4	Convolution $\rightarrow$ BN	105000×16	5	16	ReLU	same
5	Max Pool	10500×16	10	-	-	valid
6	Convolution $\rightarrow$ BN	10500×32	5	32	ReLU	same
7	Convolution $\rightarrow$ BN	10500×32	5	32	ReLU	same
8	Max Pool	1312×32	8	-	-	valid
9	Convolution $\rightarrow$ BN	1312×64	5	64	ReLU	same
10	Convolution $\rightarrow$ BN	1312×64	5	64	ReLU	same
11	Max Pool	218×64	6	-	-	valid
12	Convolution $\rightarrow$ BN	218×128	5	128	ReLU	same
13	Convolution $\rightarrow$ BN	218×128	5	128	ReLU	same
14	Max Pool	54×128	4	-	-	valid
15	Convolution $\rightarrow$ BN	54×256	5	256	ReLU	same
16	Convolution $\rightarrow$ BN	54×256	5	256	ReLU	same
17	Up-sample	216×256	4	-	-	-
18	Convolution $\rightarrow$ BN	216×128	4	128	ReLU	same
19	Crop & Concat(13, 18)	216×256	-	-	-	-
20	Convolution $\rightarrow$ BN	216×128	5	128	ReLU	same
21	Convolution $\rightarrow$ BN	216×128	5	128	ReLU	same
22	Up-sample	1296×128	6	-	-	-
23	Convolution $\rightarrow$ BN	1296×64	6	64	ReLU	same
24	Crop & Concat(10, 23)	1296×128	-	-	-	-
25	Convolution $\rightarrow$ BN	1296×64	5	64	ReLU	same
26	Convolution $\rightarrow$ BN	1296×64	5	64	ReLU	same
27	Up-sample	10368×64	8	-	-	-
28	Convolution $\rightarrow$ BN	10368×32	8	32	ReLU	same
29	Crop & Concat(7, 28)	10368×64	-	-	-	-
30	Convolution $\rightarrow$ BN	10368×32	5	32	ReLU	same
31	Convolution $\rightarrow$ BN	10368×32	5	32	ReLU	same
32	Up-sample	103680×32	10	-	-	-
33	Convolution $\rightarrow$ BN	103680×16	10	16	ReLU	same
34	Crop & Concat(4, 33)	103680×32	-	-	-	-
36	Convolution $\rightarrow$ BN	103680×16	5	16	ReLU	same
35	Convolution $\rightarrow$ BN	103680×16	5	16	ReLU	same
36	Convolution	103680×5	1	5	TanH	same
37	Zero padding	105000×5	-	-	-	-
38	Reshape	$35 \times 3000 \times 5$	-	-	-	-
38	Average Pooling	35×5	-	-	-	valid
39	Convolution	35×5	1	5	Softmax	same

Figura 2.5: Estructura de U-Time. Las capas de la 3 a la 35 se corrresponden con el autocodificador, y de la 36 a la 39 con el clasificador de segmentos. Dentro del autocodificador, podemos ver los 4 bloques convolucionales correspondientes al codificador (de la 3 a la 14) y los 4 bloques correspondientes al decodificador (de la 17 a la 35). Imagen extraída de [Per+19].

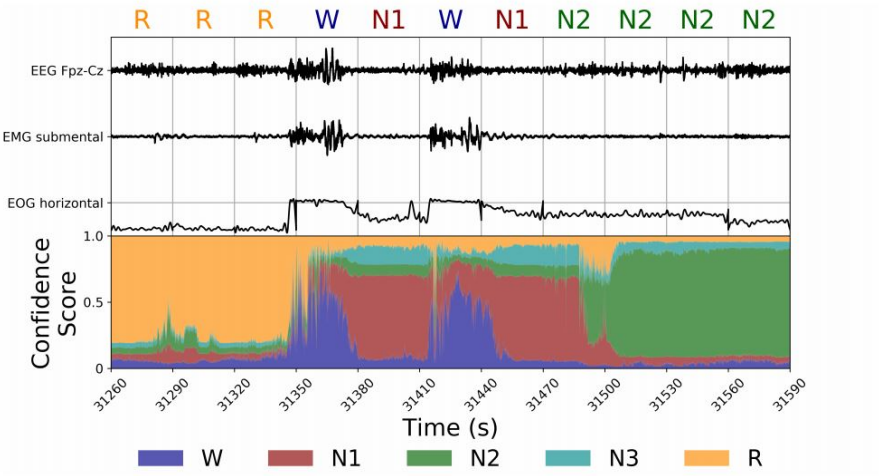


Figura 2.6: Resultados obtenidos del autocodificador tras introducir una señal de tres canales. El autocodificador asigna a cada punto de la señal un valor entre 0 y 1 por cada clase. Estos valores pueden interpretarse como una puntuación que indica la seguridad con la que el autocodificador clasifica cada punto en las distintas clases. Se ve cómo los valores con mayor seguridad en cada segmento (representados por colores) se corresponden con los valores de verdad (sobre los tres canales) en prácticamente todos los casos. Imagen extraída de [Per+19].

Adelantando parte de lo que se tratará después, se van a sustituir las señales de los estudios PSG por datos sobre la concentración del NO_2 en el aire de Madrid en diversas estaciones de medida, las etiquetas relativas a las fases del sueño por etiquetas relativas a la contaminación del aire (contaminación leve, grave, etc.) y se van a obtener de esta forma valores que nos digan con qué confianza apuesta la red por un nivel de contaminación u otro.

Antes de llegar a esa parte, sin embargo, hay que pasar al apartado de evaluación y experimentos realizados por los investigadores que desarrollaron U-Time.

2.4 U-Time: Reproducibilidad de los resultados

Hay que mencionar que el preprocesamiento de los datos es muy importante por las características del conjunto de datos que se estudia en este artículo. En el contexto de ser esto un Trabajo de Fin de Máster relacionado con la Inteligencia Artificial y el Aprendizaje Profundo, habrá apartados de ese preprocesamiento que interesen (¿cómo se balancean las clases si están desbalanceadas?) y habrá otros que no sean de interés en absoluto. Es por esto por lo que en esta sección se omiten algunos detalles relativos al análisis de las fases del sueño y la sección se centra sen los puntos interesantes a la hora de trabajar con herramientas de Aprendizaje Profundo.

Los experimentos realizados con U-Time se han realizado sobre diferentes conjuntos de datos sobre fases del sueño sin modificar la arquitectura ni hiperparámetros en el cambio de unos a otros.

Preprocesamiento: Se ha tomado una longitud de 30 segundos por segmento siguiendo el estándar de la *American Academy of Sleep Medicine*. Se han muestreado las señales con una tasa de $S = 100$Hz. Se han descartado segmentos etiquetados de forma extraña, como *en movimiento* o *sin determinar*, quedándose para todos los datos únicamente con las cinco etiquetas ya comentadas en la sección 2.2: $\{W, N1, N2, N3, R\}$. Además, las señales EEG se han normalizado y escalado para tener media 0 e IQR 1.

En algunas ocasiones además se han obtenido valores extremos al principio y al final de los estudios PSG. Estos valores están asociados a los momentos en que los sujetos de estudio se ponen los electrodos, o cuando entran o salen de la cama. Si algún segmento incluye algún valor por encima de 20 veces el IQR, entonces los valores de dicho segmento se establecen como cero. Esto sólo se aplica si el segmento se ha etiquetado como Despierto (W). Esta decisión viene derivada de la necesidad de mantener consistencia temporal entre segmentos vecinos: si se desechan los segmentos en los que el sujeto está despierto por tener valores extremos, ¿cómo va a aprender la red cuáles son los segmentos en los que el sujeto está despierto? ¿Cómo va a aprender cuál es la fase del sueño que viene justo cuando deja de estar despierto?

Este tipo de decisiones podría interesar más adelante si, entre los datos sobre la contaminación del aire de Madrid, se incluyeran datos extraños relacionados por ejemplo con el mantenimiento de las estaciones. Si supiéramos que en una determinada fecha los instrumentos de medida han dejado de funcionar, se podría descartar por completo un segmento que únicamente incluyera ceros, o se podría rellenar con la contaminación media de ese mes para no romper la serie.

Optimización: Se ha utilizado la siguiente función coste:

$$\mathcal{L}\left(y, \hat{y}\right) = 1 - \frac{2}{K} \frac{\sum_k^K \sum_n^N y_{kn} \hat{y}_{kn}}{\sum_k^K \sum_n^N y_{kn} + \hat{y}_{kn}} \tag{2.1}$$

Esta función coste se introduce en [Sud+17]. En dicho artículo se indica que es especialmente útil en casos en los que las clases estén muy desbalanceadas (como es nuestro caso). Además, para evitar aún más el desbalanceo, se han tomado batches de tamaño 12 siguiendo el siguiente esquema:

1. Se toma una clase al azar de forma uniforme de entre las cinco clases.
2. Se selecciona un segmento de la señal aleatoriamente que se corresponda con dicha clase.
3. Se desplaza el segmento a una posición aleatoria en la ventana de anchura $T = 35$ segmentos de un miembro del lote.

Este esquema, aunque distribuye de forma más uniforme los segmentos en el lote, no realiza verdaderamente un buen balanceo de los segmentos porque los otros 34 de la ventana aún están sujetos a estar desbalanceados, sin embargo lo mejora un poco.

2.4.1 Resultados

U-Time se ha aplicado a 7 conjuntos de datos de PSG distintos. En la Figura 2.7 se encuentra la medida F1 obtenida para distintos conjuntos de datos y distintas redes neuronales. En ella se puede ver que incluso sin modificar la estructura de U-Time para hacerla más específica a cada conjunto de datos, se logra un mejor

rendimiento para conjuntos de datos tanto grandes (Physionet-18) como pequeños (Sleep-EDF-39). También se obtienen mejores resultados para poblaciones sanas (Sleep-EDF-153) y enfermas (DCSM). En general, para todos los conjuntos de datos, se tiene que U-Time ha obtenido resultados mejores o iguales que otros métodos conocidos que se han utilizado previamente. En especial, U-Time se desenvuelve de forma similar o mejor que la estructura CNN-LSTM que vimos en la Figura 2.2.

		Eval		Global F1 scores					
Dataset	Model	Records	CV	W	N1	N2	N3	REM	mean
S-EDF-39	*U-Time*	39	20	0.87	0.52	0.86	0.84	0.84	0.79
	CNN-LSTM[1]	39	20	0.85	0.47	0.86	0.85	0.82	0.77
	VGGNet[2]	39	20	0.81	0.47	0.85	0.83	0.82	0.76
	CNN[3]	39	20	0.77	0.41	0.87	0.86	0.82	0.75
	Autoenc.[4]	39	20	0.72	0.47	0.85	0.84	0.81	0.74
S-EDF-153	*U-Time*	153	10	0.92	0.51	0.84	0.75	0.80	0.76
	CNN-LSTM	153	10	0.91	0.47	0.81	0.69	0.79	0.73
Physio-18	*U-Time*	994	5	0.83	0.59	0.83	0.79	0.84	0.77
	CNN-LSTM	994	5	0.82	0.58	0.83	0.78	0.85	0.77
DCSM	*U-Time*	255	5	0.97	0.49	0.84	0.83	0.82	0.79
	CNN-LSTM	255	5	0.96	0.39	0.82	0.80	0.82	0.76
ISRUC	*U-Time*	99	10	0.87	0.55	0.79	0.87	0.78	0.77
	CNN-LSTM	99	10	0.84	0.46	0.70	0.83	0.72	0.71
	Human obs.	99	-	0.92	0.54	0.80	0.85	0.90	0.80
CAP	*U-Time*	101	5	0.78	0.29	0.76	0.80	0.76	0.68
	CNN[5]	104	5	0.77	0.35	0.76	0.78	0.76	0.68
	CNN-LSTM	101	5	0.77	0.28	0.69	0.77	0.75	0.65
SVUH-UCD	*U-Time*	25	25	0.75	0.51	0.79	0.86	0.73	0.73

Figura 2.7: Comparación del rendimiento por clases obtenido por distintos modelos sobre los 7 conjuntos de datos analizados. Nótese que en general U-Time obtiene resultados iguales o mejores que los otros modelos salvo en el caso de los observadores humanos en el conjunto de datos ISRUC. En este caso la diferencia es mínima entre ambos es mínima, y sin embargo el tiempo que tarda un humano en clasificar estas señales es infinitamente superior al que tarda U-Time. Imagen extraída de [Per+19].

2.5 Lecciones aprendidas y conocimientos adquiridos

De este capítulo se han extraído algunas lecciones valiosas:

- Cuando se trabaja sobre datos muy ruidosos, puede ser conveniente utilizar autocodificadores que sean capaces de descartar el ruido y extraer las características relevantes.
- Puede ser conveniente utilizar un autocodificador compuesto por capas convolucionales a la hora de analizar series temporales en lugar de Redes Neuronales Recurrentes, Redes Neuronales Convolucionales u otros métodos.
- Si en el autocodificador se usan capas convolucionales, entonces cada dato de entrada influye en un entorno más grande cuanto más profundo sea el autocodificador, lo que puede ayudar a conservar dependencias a largo plazo.
- Si se clasifican segmentos de series temporales de distintas anchuras, entonces es suficiente con entrenar la red una única vez y después modificar el clasificador de segmentos del final para realizar una agrupación que seleccione la etiqueta que más veces aparece en cada segmento.

Estas lecciones indican un camino a seguir a la hora de trabajar con otras series temporales. Si se modifica U-Time lo suficiente como para poder trabajar sobre la serie temporal de la contaminación del aire de Madrid, entonces se dispondrá de un modelo capaz de descartar el ruido de la serie, extraer las características relevantes y clasificar cada fragmento de modo que indique el nivel de contaminación correspondiente. En el siguiente capítulo se verán los resultados obtenidos de seguir este camino.

3 Investigación a partir de U-Time

En este tercer capítulo se presentan dos variantes de U-Time. Estas variantes se construyen modificando la profundidad de U-Time y los factores en que el autocodificador reduce y aumenta la dimensionalidad de los datos. Utilizando estas variantes se analizan los datos de contaminación por NO_2 en el aire de Madrid. La idea es construir imágenes con los datos de la serie temporal utilizando una fila por cada estación de medida. Después se entrenarán las variantes de U-Time con una parte de estos datos y se comprobará su rendimiento con el resto.

3.1 Descripción de los datos

En este trabajo se han utilizado los datos horarios y diarios de la calidad del aire de Madrid entre los años 2011 y 2020. Los datos se encuentran a disposición del público en el Portal de datos abiertos del Ayuntamiento de Madrid [Mada]. Aunque este trabajo se centra en la concentración de dióxido de nitrógeno (NO_2) estos datos recogen la concentración de otros gases, como dióxido de azufre (SO_2) o monóxido de carbono (CO).

El NO_2 es un contaminante con gran impacto en la salud de las personas, pudiendo ocasionar problemas como disminución de la capacidad pulmonar, asma o bronquitis aguda. La mayor parte tiene su origen en los motores de combustión, especialmente los de diésel, al oxidarse el monóxido de nitrógeno (NO) que expulsan. Legalmente, el límite anual de NO_2 es de $40 \mu g/m^3$, mientras que el límite horario es de $200 \mu g/m^3$, aunque no debe superarse más de 18 veces al año.

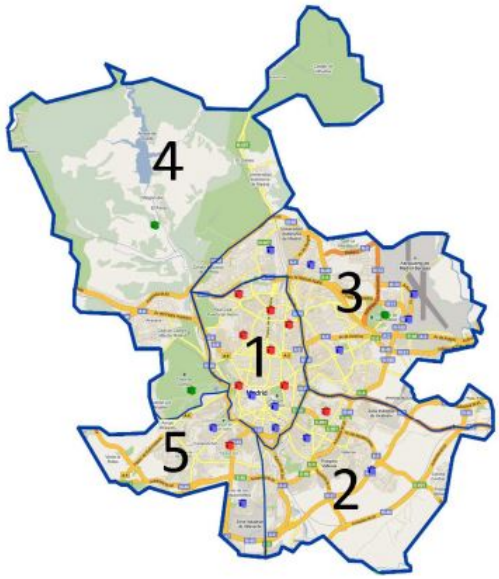


Figura 3.1: Mapa de Madrid repartido en las cinco zonas que se indican en el protocolo mencionado. Los símbolos rojos indican las estaciones de tráfico, los azules las de fondo y las verdes, las suburbanas. Imagen extraída de [Madb].

Aunque de las 24 estaciones que hay recogiendo datos todas recogen datos sobre el NO_2, en este trabajo se estudiarán los datos recogidos por 12 de ellas. Siguiendo la distribución que se incluye en el Protocolo de actuación para episodios de contaminación por dióxido de nitrógeno [Madb], estas 12 estaciones están repartidas en cinco zonas diferentes, ilustradas en la Figura 3.1. Se incluye la distribución de las 12 estaciones en estas cinco zonas y sus códigos de identificación, así como el tipo de estación:

- **Zona 1: Interior de la M30.**
    1. Escuelas de Aguirre (08), estación de tráfico.
    2. Barrio del Pilar (39), estación de tráfico.
    3. Plaza del Carmen (35), estación de fondo.
    4. Retiro (49), estación de fondo.

- **Zona 2: Sudeste.** Área delimitada por Avda. de Andalucía, Calle 30, la autovía M-23 continuando por la R-3 y hasta el límite del término municipal de Madrid.
    1. Ensanche de Vallecas (54), estación de fondo.
- **Zona 3: Noreste.** Área delimitada por la autovía M-23 y la continuación de la R-3, Calle 30 hasta la M-40 en la zona oeste y desde allí limita al norte con la M-40 hasta el límite del término municipal de Madrid. Esta zona incluye parte del Aeropuerto de Barajas.
    1. Arturo Soria (16), estación de fondo.
    2. Barajas (27), estación de fondo.
    3. Parque Juan Carlos I (59), estación suburbana.
- **Zona 4: Noroeste.** Área delimitada por el contorno del límite del municipio de Madrid por el norte, la M-40 norte, Calle 30 hasta la A-5 y el límite del municipio.
    1. El Pardo (58), estación suburbana.
    2. Casa de Campo (24), estación suburbana.
- **Zona 5: Sudoeste.** Área delimitada por el contorno sur de la Casa de Campo, Calle 30, Avda. de Andalucía y el término municipal de Madrid.
    1. Fernández Ladreda (56), estación de tráfico.
    2. Farolillo (18), estación de fondo.

3.2 Datos diarios

De [Mada] se extraen los datos diarios de concentración de NO_2 para las doce estaciones nombradas en el apartado anterior entre el 01/01/2011 y el 31/03/2020 (Figura 3.2). Las limitaciones legales actuales son las mencionadas en la sección anterior: $40\mu g/m^3$ anuales, y un límite horario de $200\mu g/m^3$ que no debe superarse más de 18 veces al año. Se tiene por tanto que no existe limitación diaria alguna salvo por la obligatoriedad de respetar el límite de $40\mu g/m^3$ anuales. Será necesario por tanto proponer algunos límites diarios que permitan etiquetar los datos diarios. En la siguiente sección se verá cómo se van a crear estos límites.

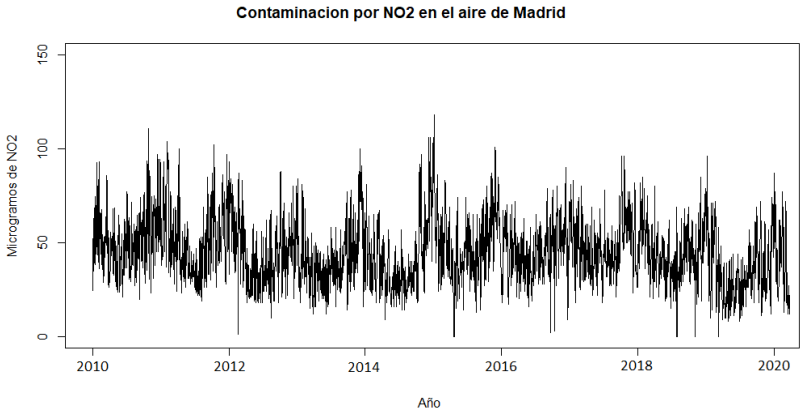


Figura 3.2: Datos diarios de la concentración de NO_2 entre 01/01/2011 y 31/03/2020 en la estación de monitorización de Plaza del Carmen (Madrid). A pesar del ruido puede observarse una cierta estacionalidad de los datos, reflejada en los picos que hay en torno al inicio de cada año. Imagen propia.

3.2.1 Niveles de alerta

A pesar de que no existen limitaciones diarias de concentración de NO_2, desde el Ayuntamiento de Madrid [Madb] se han establecido tres niveles de actuación en función de las concentraciones horarias de NO_2 que se registren:

- ***Preaviso*:** Cuando **dos** estaciones de una misma zona superan simultáneamente **180** $\mu g/m^3$ durante **dos** horas consecutivas, o **tres** estaciones de la red de vigilancia superan, simultáneamente, 180 $\mu g/m^3$ durante **tres** horas consecutivas.
- ***Aviso*:** Cuando **dos** estaciones de una misma zona superan simultáneamente **200** $\mu g/m^3$ durante **dos** horas consecutivas, o **tres** estaciones de la red de vigilancia superan, simultáneamente, **200** $\mu g/m^3$ durante **tres** horas consecutivas.
- ***Alerta*:** Cuando **tres** estaciones cualesquiera de una misma zona (o dos de la zona 4) superan, simultáneamente, **400** $\mu g/m^3$ durante **tres** horas consecutivas.

Estos escenarios pueden servir de guía para tener en cuenta la evolución de los datos diarios. Dado que los datos diarios están muy por debajo de los 200 $\mu g/m^3$, se tomarán como valores límites los percentiles anuales a la hora de establecer los umbrales. Además, dado que se utilizan estos percentiles y es poco probable que varias estaciones alcancen dichos valores simultáneamente, se ha decidido relajar esta condición a que una única estación alcance esos valores durante un periodo de tiempo determinado. Se han establecido por tanto los siguientes escenarios :

- ***Preaviso* (diario):** Cuando una estación supera el percentil 75 anual durante tres días consecutivos.
- ***Aviso* (diario):** Cuando una estación supera el percentil 90 anual durante dos días consecutivos.
- ***Alerta* (diario):** Cuando una estación supera el percentil 95 anual un día.

Teniendo estos escenarios, se puede definir por tanto una forma de etiquetar los datos de los que disponemos. Para un periodo de tiempo dado tendremos cuatro posibles etiquetas: *Sin riesgo* (si los datos no encajan en los escenarios anteriores), *Preaviso*, *Aviso* y *Alerta*. En el siguiente apartado se verá cómo se ha entrenado la red teniendo esto en cuenta.

3.2.2 Entrenamiento y resultados

Por la estructura de U-Time, hay que elegir el tamaño que tendrán las imágenes que serán introducidas en la red. Dado que tenemos doce estaciones, lo apropiado es que la altura de las imágenes sea 12, pero ¿qué hacer con la anchura? Se ha decidido en primera instancia utilizar imágenes que recojan semanas, es decir, que la anchura sea 7.

Uno de los grandes inconvenientes que surgen es que carecemos de datos. Para cada estación contamos únicamente con 3.743 datos. Si se reparten en semanas, apenas quedan 500 imágenes: insuficientes para entrenar una red neuronal. Para suplir esta falta de datos y obligar a la red a aprender el componente temporal de los datos, se utilizarán ventanas deslizantes para construir las imágenes. Esto

es, si la primera imagen va del 1 al 8 de enero de 2011, la segunda irá del 2 al 9, y así sucesivamente.

Otro inconveniente es que U-Time reduce la dimensionalidad de los datos en un factor de 1920 tras atravesar los cuatro bloques convolucionales del codificador. Las imágenes son de 12x7 (84 datos por imagen), por lo que esto es inviable. Se va a modificar la cantidad de bloques convolucionales y el factor en que reduce cada bloque la dimensionalidad de los datos para que la reducción sea acorde a la entrada. Con este objetivo en mente, se ha decidido que el codificador se componga de tres bloques convolucionales que reduzcan la dimensionalidad de los datos en un factor de 3, 2 y 2 respectivamente. Tras esto, la dimensionalidad de las imágenes se verá reducida en un factor de 12 en el vector latente. Análogamente, el decodificador se compone de tres bloques convolucionales que aumentan la dimensionalidad de los datos en un factor de 2, 2 y 3 respectivamente. No se ha modificado ningún hiperparámetro más de U-Time.

Para etiquetar las imágenes, se calculará el percentil 75, 90 y 95 anual correspondiente y se determinará si la imagen se encuentra en *Preaviso*, *Aviso*, *Alerta* o *Sin riesgo*. Una vez determinadas las imágenes de entrada y sus etiquetas, se extrae un conjunto de entrenamiento (80 %) y uno de validación (20 %) que se utilizará para comprobar si la red ha aprendido correctamente o no a etiquetar los datos. Este reparto de los datos se realiza al azar. La red se ha entrenado durante 600 épocas utilizando un tamaño de lote de 8.

El rendimiento obtenido tras analizar los datos diarios ha inferior al esperado. La red sobreajusta los niveles de *Aviso* y *Alerta* a las opciones de los extremos: o bien no hay riesgo o bien los datos se corresponden con el nivel de alerta (Figura 3.3). Esto se debe probablemente a que no se cuenta con suficientes datos como para detectar las dependencias temporales inherentes a los datos. También es posible que el vector latente sea demasiado pequeño como para que la red sea capaz de extraer en el vector latente las características necesarias para predecir el nivel de contaminación. Se podría probar a entrenar una variante que disponga de menos bloques de convolución en el codificador y el decodificador, pero entonces se perdería profundidad en el autocodificador. No se realizarán más experimentos con los datos diarios y se centrarán los esfuerzos en los datos horarios, ya que se dispone mayor cantidad de estos y tiene más sentido teniendo en cuenta que los

niveles de alerta se definen para datos horarios.

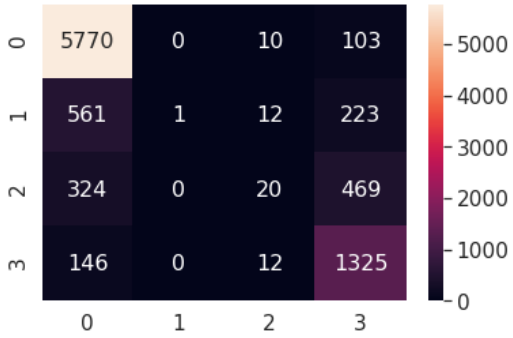


Figura 3.3: Mapa de calor de los niveles de alerta para imágenes de 7 días que se solapan en intervalos de 6 días. Los valores reales se corresponden con las filas, mientras que las predicciones se corresponden con las columnas. Se puede ver que la red concentra sus predicciones en los niveles *Sin riesgo* (0) y *Alerta* (3), sin ser capaz de distinguir los valores de *Preaviso* (1) y *Aviso* (2). Hay que señalar también que, pese a esto, la red es capaz de detectar con gran precisión las imágenes correspondientes a los niveles de Sin riesgo y *Alerta*: 98 % y 89 % respectivamente. Imagen propia.

3.3 Datos horarios

En esta sección se van a utilizar los datos horarios sobre la concentración de NO_2 en las doce estaciones que hemos visto antes a lo largo de los años 2018 y 2019. Esto nos da un total de 17.520 datos para cada estación, por lo que no será necesario solapar imágenes utilizando ventanas deslizantes como en el caso de los datos diarios al tener una gran cantidad de estos. Sin embargo, es adecuado realizar experimentos con solape entre las imágenes y sin él para ver cómo se comporta la red.

3.3.1 Niveles de alerta

Al igual que en el caso de los datos diarios, no se van a utilizar los niveles de actuación del Protocolo de actuación para episodios de contaminación por dióxido de nitrógeno [Madb] por haber muy pocas situaciones en las que se dan situaciones de *Alerta* o de *Aviso*, por no decir ninguna. Esto haría que las clases estén especialmente desbalanceadas, y que por tanto la red no pudiera aprender. Se ha realizado la comprobación por si acaso y se han obtenido los resultados esperados: la red devuelve siempre la etiqueta *Sin riesgo*.

Es por esta razón por la que se ha decidido utilizar de nuevo unos umbrales al margen del Protocolo:

- ***Preaviso* (horario):** Cuando una estación supera el percentil 75 anual de contaminación durante tres horas consecutivas.
- ***Aviso* (horario):** Cuando una estación supera el percentil 90 anual durante dos horas consecutivas.
- ***Alerta* (horario):** Cuando una estación supera el percentil 95 anual una hora.

3.3.2 Entrenamiento y resultados

A la hora de construir las imágenes, las filas se corresponderán con las estaciones al igual que en el caso de los datos diarios. Las imágenes contarán por tanto con 12 filas, es decir, altura 12. Hay que elegir sin embargo la anchura: ¿cuántas horas se tomarán para cada imagen? ¿Se solaparán unas imágenes con otras? Por otra parte, ¿cuál va a ser el objetivo de la red? Se podría utilizar para clasificar las imágenes introducidas de modo que la red se vea obligada a determinar el nivel de alerta sin conocer siquiera los percentiles 75, 90 y 95 anuales, pero esto no aportaría información sobre el futuro, ya que habría que esperar a tener una imagen completa antes de poder decir si ha habido algún nivel de alerta o no. Otra opción es asignar a cada imagen el nivel de alerta de la imagen siguiente. Si la red entrena correctamente, entonces será capaz de predecir el nivel de alerta que habrá en el futuro.

Se han realizado varios experimentos en esta línea:

- **Imágenes de anchura 6 sin solape:**
  - Se busca determinar el nivel de alerta de la imagen introducida.
  - Se busca predecir el nivel de alerta de la imagen siguiente.
- **Imágenes de anchura 24 con solape:**
  - Solape de 23 horas. Se busca determinar el nivel de alerta de la imagen introducida.
  - Solape de 23 horas. Se busca predecir el nivel de alerta de la imagen siguiente.
  - Solape de 18 horas. Se busca determinar el nivel de alerta de la imagen introducida.
  - Solape de 18 horas. Se busca predecir el nivel de alerta de la imagen siguiente.
  - Solape de 12 horas. Se busca determinar el nivel de alerta de la imagen introducida.
  - Solape de 12 horas. Se busca predecir el nivel de alerta de la imagen siguiente.
  - Solape de 6 horas. Se busca determinar el nivel de alerta de la imagen introducida.
  - Solape de 6 horas. Se busca predecir el nivel de alerta de la imagen siguiente.

Estos experimentos darán una idea de la potencia de la red a la hora de realizar predicciones a futuro sobre las imágenes a medida que disminuye el solape de una imagen con la siguiente. Por otra parte, también dará una idea de su capacidad a la hora de determinar el nivel de riesgo de la situación actual.

Experimento 1: Anchura 6 sin solape. Determinar nivel de alerta de la imagen introducida.

Al no tener solape unas imágenes con otras, se tiene que la cantidad de imágenes es menor de la que podría ser en un principio, y esto influye en los resultados (Figura 3.4). Se tiene que la red aprende la clase 0 (*Sin riesgo*) prácticamente sin problemas y rara vez la confunde, así como la clase 3 (*Alerta*) . Sin embargo, no consigue aprender correctamente la clase 1 (*Preaviso*) y la clasifica o bien como 0 o bien como 2 (*Aviso*). Tampoco aprende correctamente la clase 2 (*Aviso*) y la clasifica como 3 (*Alerta*) en numerosas ocasiones. Aunque tiene algunos fallos, estos son fallos razonables, puesto que los umbrales de las clases 2 y 3 están muy cerca (percentiles 90 y 95). Es importante señalar que la precisión para la clase 0 (*Sin riesgo*) y la clase 3 (*Alerta*) es altísima en ambos casos, de un 99.5 % y un 88 % respectivamente. Además, la precisión para la clase 2 es de un 61 %, y la cantidad de falsos negativos para las clases 2 (*Aviso*) y 3 (*Alerta*) es ínfima.

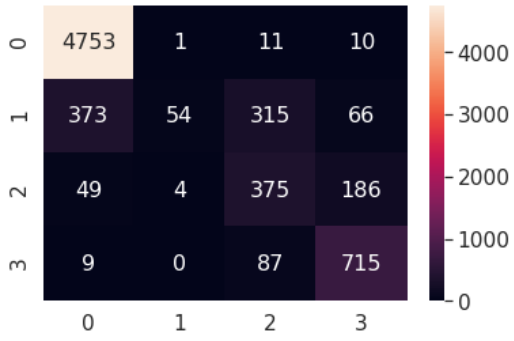


Figura 3.4: Mapa de calor de los niveles de alerta para imágenes de 6 horas que no se solapan. Las filas son los valores reales, y las columnas son las predicciones. Es relevante la gran precisión que tiene la red a la hora de clasificar imágenes correspondientes a la clase 0 (*Sin riesgo*) y la clase 3 (*Alerta*), y que cuando se equivoca en la clase 2 (*Aviso*) lo hace exagerando el nivel de contaminación y devolviendo la clase 3 (*Alerta*) normalmente. Imagen propia.

Experimento 2: Anchura 6 sin solape. Predecir el nivel de alerta de la imagen siguiente

A la hora de intentar predecir los niveles de alerta de la siguiente imagen, el rendimiento de la red ha sido muy inferior a lo esperado para las clases 1 (*Preaviso*) y 2 (*Aviso*), y no ha demostrado aprender correctamente la clase 3 (*Alerta*) (Figura 3.5).

- En torno al 75 % de los valores de la clase 1 son clasificados como clase 0, y prácticamente el resto son clasificados como clase 3.
- En torno al 50 % de los valores de la clase 2 son clasificados como clase 0, y en torno a la otra mitad como clase 3.
- En torno al 66 % de los valores de la clase 3 se clasifican correctamente. Prácticamente el resto se clasifica como clase 0.

Esto significa que la red ha dividido las imágenes entre aquellas para las que dice que el nivel de contaminación de la próxima imagen no supone ninguna alerta y aquellas para las que dice que el nivel de contaminación de la imagen siguiente es nivel de *Alerta*. Esto hace que se tengan bastantes falsos negativos para todos los niveles de alerta, por lo que no es fiable a la hora de realizar predicciones a futuro.

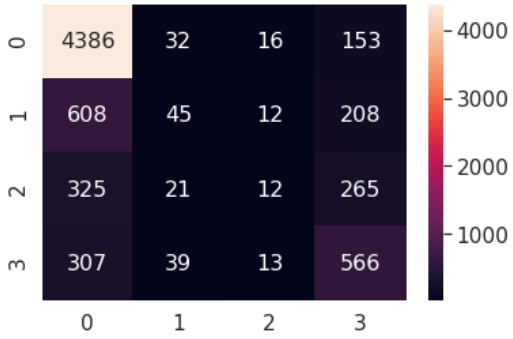


Figura 3.5: Mapa de calor de las predicciones realizadas sobre los niveles de alerta para imágenes de 6 horas que no se solapan. Se ven muchísimos falsos negativos, por lo que este modelo no es fiable en este caso. Las filas son los valores reales, y las columnas son las predicciones. Imagen propia.

Experimento 3: Anchura 24 con solape de 23 horas. Determinar nivel de alerta de la imagen introducida.

En este experimento, al utilizar un solape tan grande, se dispone de muchas más imágenes que en los dos experimentos anteriores. Al disponer de muchas más imágenes y teniendo cada una de ellas más datos, la red aprende mucho mejor cada clase, y cuando se equivoca lo hace normalmente con las clases contiguas (Figura 3.6). Esto significa que, cuando se equivoca clasificando una imagen de la clase 1, la clasifica o bien como la clase 0 o bien como la clase 2, pero raramente (en 12 casos sobre un total de 7305 imágenes) la clasifica asignándole la clase 3. Lo mismo ocurre con la clase 2.

Es importante señalar que la cantidad de falsos negativos (en el sentido de clasificar las imágenes como *Sin riesgo* cuando no es así), para los niveles de aviso y alerta es ínfima. Esto es especialmente importante por la naturaleza del problema que se estudia: determinar si la contaminación es o no relevante para una imagen dada.

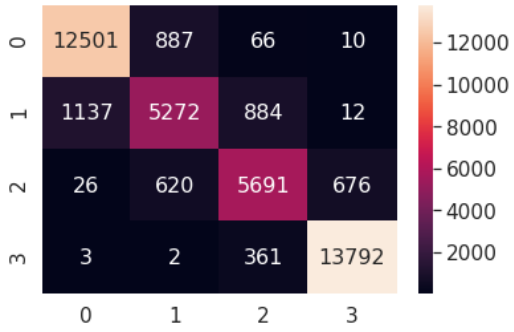


Figura 3.6: Mapa de calor de los niveles de alerta para imágenes de 24 horas que se solapan 23 horas. Las filas son los valores reales, y las columnas son las predicciones. Es relevante la gran precisión con la que cuenta la red a la hora de clasificar las 4 clases. También es importante señalar que, para cada clase, la cantidad de imágenes que la red clasifica de forma incorrecta decrece casi en un orden de magnitud conforme se la clase etiquetada se aleja de la clase real. Imagen propia.

Experimento 4: Anchura 24 con solape de 23 horas. Predecir el nivel de alerta de la imagen siguiente

De nuevo, al igual que ocurría en el Experimento 3, se tiene que en general la red aprende correctamente, y cuando se equivoca lo hace de forma razonable, confundiendo cada clase por las que están contiguas. Este comportamiento era de esperar, puesto que el solape es enorme (de 23 sobre 24 horas, casi un 96 %), por lo que los resultados deberían ser similares a los del Experimento 3 (Figura 3.7).

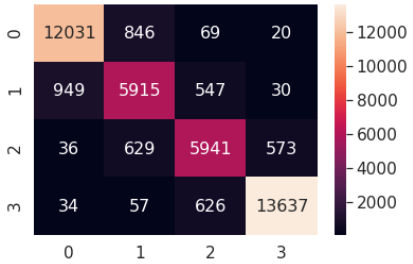


Figura 3.7: Mapa de calor de las predicciones realizadas sobre los niveles de alerta para imágenes de 24 horas que se solapan 23 horas. Las filas son los valores reales, y las columnas son las predicciones. Los resultados son prácticamente idénticos a los obtenidos en el Experimento 3 (Figura 3.6). Imagen propia.

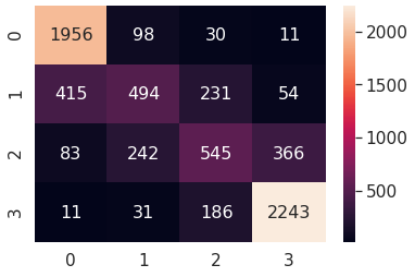


Figura 3.8: Mapa de calor de los niveles de alerta para imágenes de 24 horas con solape de 18 horas. Las filas son los valores reales, y las columnas son las predicciones. La precisión para las clases 0 y 3 es casi 1, mientras que para las clases 1 y 2 es casi de un 50 %. Se mantiene el efecto del Experimento 3: cuando la red se equivoca, asigna clases contiguas a la real. Imagen propia.

Experimento 5: Anchura 24 con solape de 18 horas. Determinar nivel de alerta de la imagen introducida.

En este experimento se cuenta con un solape inferior al solape del Experimento 3, pero aún así sigue siendo de un 75 %. Los resultados que se obtienen son similares a los del Experimento 3, aunque la precisión decae enormemente para las clases intermedias (Figura 3.8). Las clases 0 (*Sin riesgo*) y 3 (*Alerta*) se aprenden sin problemas, pero la red duda a la hora de clasificar imágenes de las clases 1 (*Preaviso*) y 2 (*Aviso*), acertando en torno al 50 % de los casos para cada una. Eso si, cuando la red se equivoca, asigna la entrada a una de las clases contiguas a la clase real, tal y como sucedía en el Experimento 3.

Al igual que en el Experimento 3, es importante señalar que la cantidad de falsos negativos para los niveles de *Aviso* y *Alerta* es ínfima. De nuevo, esto es especialmente importante por la naturaleza del problema que se estudia: determinar si la contaminación es o no relevante para una imagen dada.

Experimento 6: Anchura 24 con solape de 18 horas. Predecir el nivel de alerta de la imagen siguiente

En este caso sucede algo similar a lo que ocurre en el Experimento 5. Se mantiene el hecho de que la precisión para las clases 0 y 3 sea bastante grande. Además, cuando la red se equivoca, generalmente asigna clases contiguas a la clase real. Este efecto se pierde un poco porque han aparecido bastantes falsos negativos. La precisión para la clase 2 ha disminuido aún más que en el Experimento 5 (Figura 3.9).

Experimento 7: Anchura 24 con solape de 12 horas. Determinar nivel de alerta de la imagen introducida.

Sorprendentemente, en este experimento la red fracasa a la hora de intentar aprender la clase 2 (*Aviso*) (Figura 3.10). Las clases 0 (*Sin riesgo*) y 3 (*Alerta*) se aprenden de nuevo sin problemas, pero la red duda a la hora de clasificar imágenes de la clase 1 (*Preaviso*), acertando en torno al 50 % de los casos. De forma

similar a como ocurría en los Experimentos 3 y 5, cuando la red se equivoca, asigna la entrada a una de las clases contiguas a la clase real en el caso de la clase 2, pero para la clase 1 simplemente devuelve falsos negativos en la mitad de los casos.

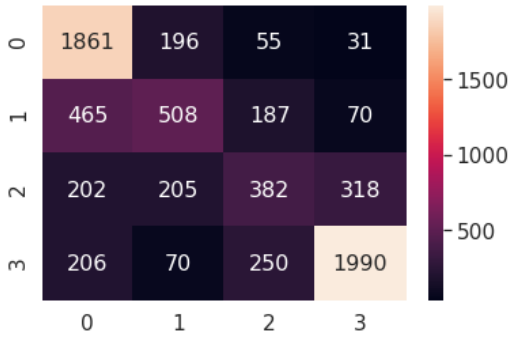


Figura 3.9: Mapa de calor de las predicciones realizadas sobre los niveles de alerta para imágenes de 24 horas que se solapan 18 horas. Las filas son los valores reales, y las columnas son las predicciones. Los resultados son prácticamente idénticos a los obtenidos en el Experimento 4 (Figura 3.8) salvo por los falsos negativos que han aparecido. Estos pueden deberse quizás a la aparición de sobreajuste. Imagen propia.

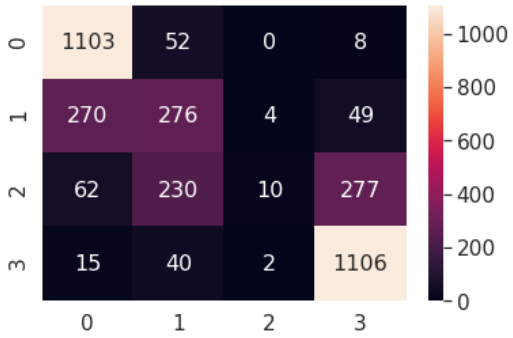


Figura 3.10: Mapa de calor de los niveles de alerta para imágenes de 24 horas con solape de 12 horas. Las filas son los valores reales, y las columnas son las predicciones. La precisión para las clases 0 y 3 es casi 1, mientras que para la clase 1 es casi de un 50 % y para la clase 2 es casi 0. Aparecen bastantes falsos negativos en la clase 1. Imagen propia.

Experimento 8: Anchura 24 con solape de 12 horas. Predecir el nivel de alerta de la imagen siguiente

En este experimento se empieza a observar una tendencia que seguirá acentuándose conforme se reduzca el solape de una imagen con la siguiente hasta llegar al caso en el que el solape sea nulo (Experimento 2). Se tiene que la red empieza a clasificar cada vez más todas las imágenes o bien como clase 0 (*Sin riesgo*) o bien como clase 3 (*Alerta*), haciendo que se desvanezcan las clases 1 y 2 (Figura 3.11). Se tiene que la precisión es buena para las clases 0 y 3, pero también existen algunos falsos negativos y falsos positivos.

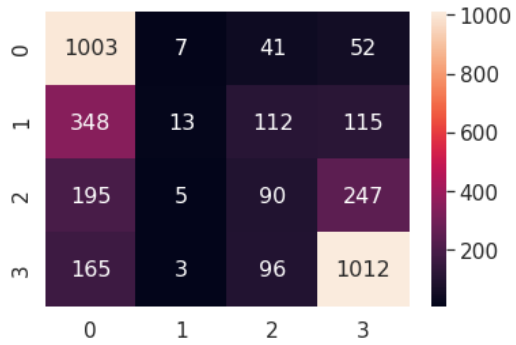


Figura 3.11: Mapa de calor de las predicciones realizadas sobre los niveles de alerta para imágenes de 24 horas que se solapan 12 horas. Las filas son los valores reales, y las columnas son las predicciones. Se ve claramente cómo las clases 1 y 2 se desvanecen y las imágenes se concentran en las clases 0 y 3. Imagen propia.

Experimento 9: Anchura 24 con solape de 6 horas. Determinar nivel de alerta de la imagen introducida.

En este caso se vuelven a obtener resultados similares a los del Experimento 5 (Figura 3.12). Las clases 0 (*Sin riesgo*) y 3 (*Alerta*) se aprenden de nuevo sin problemas, pero la red duda a la hora de clasificar imágenes de las clases 1 (*Preaviso*) y 2 (*Aviso*), acertando en torno al 50 % y al 35 % de los casos respectivamente. De forma similar a como ocurría en los Experimentos 3 y 5, cuando la red se equivoca, asigna la entrada a una de las clases contiguas a la clase real.

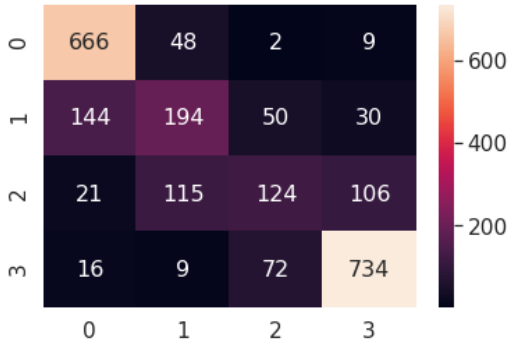


Figura 3.12: Mapa de calor de los niveles de alerta para imágenes de 24 horas con solape de 6 horas. Las filas son los valores reales, y las columnas son las predicciones. La precisión para las clases 0 y 3 es casi 1, mientras que para la clase 1 es casi de un 50 % y para la clase 2 de casi un 35 %. Los resultados son similares a los del Experimento 5. Imagen propia.

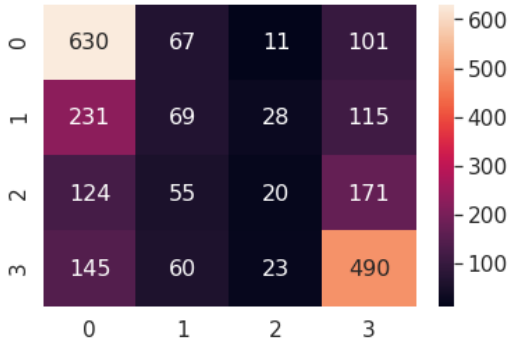


Figura 3.13: Mapa de calor de las predicciones realizadas sobre los niveles de alerta para imágenes de 24 horas que se solapan 6 horas. Las filas son los valores reales, y las columnas son las predicciones. Se ve claramente cómo las clases 1 y 2 se desvanecen, repartiéndose en las clases 0 y 3, al igual que ocurría en el Experimento 8. Imagen propia.

Experimento 10: Anchura 24 con solape de 6 horas. Predecir el nivel de alerta de la imagen siguiente

En este caso el rendimiento obtenido es muy bajo (Figura 3.13). Como se indicó en el Experimento 8, la red concentra aún más las imágenes en las clases 0 (*Sin riesgo*) y 3 (*Alerta*). Disminuye bastante la precisión, alcanzando valores de en torno al 75 % y el 66 % respectivamente para dichas clases, y no es capaz de aprender las clases 1 (*Preaviso*) y 2 (*Aviso*) . Esta red será de poca utilidad a la hora de realizar predicciones.

3.3.3 Lecciones extraídas

Mientras que ha sido prácticamente imposible utilizar los datos diarios, se han extraído algunas lecciones de los experimentos realizados sobre los datos horarios:

- Cuando el solape es escaso, las predicciones a futuro son poco fiables.
- A la hora de determinar el nivel de alerta de una imagen, la red se equivoca poco independientemente del solape, y cuando se equivoca lo hace normalmente asignando clases contiguas.
- En los casos en los que el solape es considerable, la cantidad de falsos negativos suele ser muy pequeña frente a los aciertos. Esto significa que la precisión es muy alta, por lo que el sistema raramente clasificará una imagen como *Sin riesgo* cuando su clase verdadera sea *Aviso* o *Alerta*.

En concreto, la red del Experimento 6 podría utilizarse para predecir el nivel de alerta en el que se encontrará Madrid seis horas después de los datos actuales. Predecir el nivel de contaminación de Madrid seis horas antes no supone una gran ventaja a la hora de tener que tomar decisiones como puede ser imponer restricciones al tráfico de la ciudad, pero puede servir como punto de partida sobre el que trabajar para intentar mejorar los resultados obtenidos.

4 Conclusiones

El objetivo principal de este trabajo es eminentemente práctico: construir un modelo capaz de clasificar correctamente fragmentos de una serie temporal concreta. Dicha serie temporal es la contaminación del aire de Madrid por NO_2 entre los años 2011 y 2020, cuyos datos se obtienen de diversas estaciones de medida.

Para alcanzar este objetivo se han considerado diversos modelos tanto estadísticos como de Aprendizaje Profundo aptos para trabajar con series temporales: ARIMA, Redes Neuronales Convolucionales, Redes Neuronales Recurrentes, Autocodificadores, etc. Estos modelos tienen en común la capacidad de detectar dependencias a largo plazo y estacionalidades, y tenerlas en cuenta a la hora de realizar predicciones sobre los datos. Se han considerado además comparativas de diversos artículos relacionados con series temporales para ver qué modelos podían obtener mejores resultados de entre los mencionados.

Una vez realizado este estudio, se ha analizado en profundidad U-Time: un modelo cuya estructura se basa en un autocodificador compuesto de bloques convolucionales, y un clasificador de segmentos. Aunque U-Time estaba destinada a usarse para trabajar con señales de estudios polisomnográficos, ha sido posible modificar la red lo suficiente como para dedicarla a los datos de la contaminación del aire de Madrid.

Se han considerado pues dos series temporales distintas: los datos diarios y horarios de contaminación. Los resultados obtenidos de los datos diarios han sido bastante malos, por lo que se ha decidido descartarlos y trabajar con los datos horarios. Con los datos horarios se han realizado diversos experimentos considerando fragmentos que se solapaban y fragmentos que no lo hacían, variando el

tamaño del solape en aquellos que se solapaban para ver cómo se comportaba el modelo y modificando las etiquetas en función de si queríamos clasificar el fragmento en cuestión o si queríamos predecir la etiqueta del fragmento siguiente.

Utilizando esta variante de U-Time ha sido posible construir un modelo capaz de predecir el nivel de contaminación del aire de Madrid con una previsión de seis horas y con una precisión de un 87 % para la clase *Sin riesgo*, 41 % para la clase *Preaviso*, 35 % para la clase *Aviso* y 79 % para la clase *Alerta*. Hay que tener en cuenta que los datos son reales y se ven influenciados por múltiples factores, por lo que se puede considerar que los resultados han sido positivos, especialmente los relativos a las clases *Sin riesgo* y *Alerta*. Dado que la precisión es alta para las clases *Sin riesgo* y *Alerta*, se tiene que el modelo no suele asegurar que un nivel de contaminación elevado no conlleva riesgo o viceversa.

A nivel de Trabajo de Fin de Máster, se puede concluir que se han cumplido los objetivos de que se esperaban. En primer lugar, se ha realizado un estudio teórico exhaustivo de la materia relacionada con el problema. En segundo lugar, se ha elegido un modelo existente y se ha estudiado en profundidad para entender su funcionamiento. Por último, se ha modificado dicho modelo y se ha aplicado al problema considerado.

5 Trabajo Futuro

A la hora de trabajar sobre los datos horarios de la contaminación del aire de Madrid se han construido imágenes de 12 filas (una por cada estación de medida) y 6 o 24 columnas (una por cada hora). Al existir más estaciones de medida, se podría ampliar el tamaño de las imágenes añadiendo una fila más por cada estación de medida que quiera considerarse. Además, si en lugar de trabajar con los datos horarios de 2018 y 2019 se trabajase con todos los datos desde 2011, se dispondría de datos suficientes como para construir imágenes con una mayor cantidad de columnas. Al ampliar el tamaño de las imágenes se podría aumentar la profundidad de la red un poco más y/o aumentar el tamaño del vector latente. De esta forma, el modelo debería obtener mejores resultados, siendo capaz de realizar predicciones con aún más antelación.

Podría ser interesante modificar la estructura de U-Time un poco más haciendo que el autocodificador fuera variacional. Como se vio en el Capítulo 1, los autocodificadores variacionales generalmente obtienen algunas ventajas a la hora de distribuir los puntos del vector latente, lo cual puede traducirse en un aumento del rendimiento del modelo.

Puede ser interesante también trabajar con otras series temporales y ver si se obtienen resultados similares a los descritos en este trabajo. Es fácil modificar la profundidad de la red y la estructura de los datos de entrada, por lo que puede ser interesante utilizarla en otros problemas relacionados y ver cómo se comporta.

En relación a este mismo problema, sería interesante trabajar con científicos dedicados a la calidad del aire para establecer unos niveles de contaminación realistas y útiles. Los umbrales seleccionados a partir de percentiles pueden ser útiles, pero quizás no sean la mejor opción en un escenario real.

Bibliografía

[SS00] Robert H. Shumway y David S. Stoffer. *Time Series Analysis and Its Applications*. Springer, 2000.

[HN03] James Hansen y Ray Nelson. "Time-series analysis with neural networks and ARIMA-neural network hybrids". En: *J. Exp. Theor. Artif. Intell.* 15 (jul. de 2003), págs. 315-330. DOI: 10.1080/0952813031000116488.

[Tom03] J. Tomayko. "Behind Deep Blue: Building the Computer that Defeated the World Chess Champion (review)". En: *Technology and Culture* 44 (ene. de 2003), págs. 634-635. DOI: 10.1353/tech.2003.0140.

[IS15] Sergey Ioffe y Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". En: (feb. de 2015).

[RFB15] Olaf Ronneberger, Philipp Fischer y Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". En: vol. 9351. Oct. de 2015, págs. 234-241. ISBN: 978-3-319-24573-7. DOI: 10.1007/978-3-319-24574-4_28.

[YK15] Fisher Yu y Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". En: (nov. de 2015).

[GBC16] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[CW17] Qiming Chen y Ren Wu. "CNN Is All You Need". En: (dic. de 2017).

[Sud+17] Carole Sudre y col. "Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations". En: sep. de 2017, págs. 240-248. ISBN: 978-3-319-67557-2. DOI: 10.1007/978-3-319-67558-9_28.

[Sup+17] Akara Supratak y col. "DeepSleepNet: a Model for Automatic Sleep Stage Scoring based on Raw Single-Channel EEG". En: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* PP (mar. de 2017). DOI: 10.1109/TNSRE.2017.2721116.

[Vas+17] Ashish Vaswani y col. "Attention Is All You Need". En: (jun. de 2017).

[BKK18] Shaojie Bai, J. Kolter y Vladlen Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". En: (mar. de 2018).

[STS18] Sima Siami Namini, Neda Tavakoli y Akbar Siami Namin. "A Comparison of ARIMA and LSTM in Forecasting Time Series". En: dic. de 2018, págs. 1394-1401. DOI: 10.1109/ICMLA.2018.00227.

[Ji+19] Lei Ji y col. "Carbon futures price forecasting based with ARIMA-CNN-LSTM model". En: *Procedia Computer Science* 162 (ene. de 2019), págs. 33-38. DOI: 10.1016/j.procs.2019.11.254.

[Per+19] Mathias Perslev y col. "U-Time: A Fully Convolutional Network for Time Series Segmentation Applied to Sleep Staging". En: (oct. de 2019).

[GS20] M.Ángel Gutiérrez Naranjo y David Solís Martín. *Apuntes de Aprendizaje Profundo: Autoencoders*. 2020.

[Cap] Fernando Sánchez Caparrini. *Variational AutoEncoder*. URL: http://www.cs.us.es/~fsancho/?e=232. (acceso: 24.07.2020).

[Mada] Ayuntamiento de Madrid. *Portal de datos abiertos del Ayuntamiento de Madrid*. URL: https://datos.madrid.es/portal/site/egob. (acceso: 24.07.2020).

[Madb] Ayuntamiento de Madrid. *Protocolo de actuación para episodios de contaminación por dióxido de nitrógeno*. URL: https://www.madrid.es/UnidadesDescentralizadas/Sostenibilidad/CalidadAire/Ficheros/ProtocoloNO2AprobFinal_201809.pdf. (acceso: 23.08.2020).

[Pri] Benny Prijono. *Student Notes: Convolutional Neural Networks (CNN) Introduction*. URL: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/. (acceso: 23.08.2020).

[Wik] Wikipedia. *Polysomnography*. URL: https://en.wikipedia.org/wiki/Polysomnography. (acceso: 24.07.2020).