



Universidad Politécnica de Madrid
Universidad Complutense de Madrid

**RNAE**: Redes neuronales y aprendizaje estadístico

Máster Inter-Universitario en Tratamiento Estadístico-Computacional de la Información

---

# Diseño e implementación de un sistema para detección de intrusiones en entornos IoT

---

***Realizado por:***
Herencia López-Menchero, Andrés

***Profesor :***
Zufiria Zatarain, Pedro J.

2 de marzo de 2024

# Índice

# 1. Introducción

## 1.1. Contexto y problema

En la era de los datos, la seguridad y la privacidad se han convertido en preocupaciones relevantes debido a la creciente sofisticación y aumento de los ciberataques. En este contexto, uno de los elementos más afectados por este tipo de ataques son los dispositivos IoT (*Internet of Things*), sobre los cuales se han investigado numerosas formas de securizarlos. Así, el presente trabajo pretende dar solución al problema de detección de intrusiones en dispositivos IoT.

**Palabras clave: IoT; CNN; PCA; ciberataques; DNS.**

## 1.2. Objetivos

El objetivo de este trabajo es implementar una arquitectura de aprendizaje estadístico para la detección de intrusiones en entornos IoT sobre el protocolo del Sistema de Nombres de Dominio (DNS). La arquitectura debe ser computacionalmente eficiente para poder ser implementada en dispositivos de reducidas prestaciones.

## 1.3. Recursos

Se emplea Python como herramienta de software sobre máquinas virtuales en la nube. Desde el punto de vista matemático y estadístico, se utilizan las redes neuronales convolucionales (CNN) para la clasificación y detección de intrusiones.

En cuanto a los datos, se usa el conjunto `IoTID20` de 2018, que incluye información etiquetada sobre ataques a dispositivos IoT.[1]. Incluye datos del tráfico de varios dispositivos IoT, como ordenadores portátiles, teléfonos inteligentes, cámaras EZVIZ y altavoces SKT NUGU, recopilados para simular escenarios en tiempo real para la investigación en materia de detección de intrusiones.

Cuenta con 86 columnas y 625.783 filas, etiquetados con varios grados de granularidad: La etiqueta o `Label`, variable binaria, indica si ha habido o no un ataque; La categoría o `Cat`, variable categórica multi-clase, indica el tipo de ataque; y la Sub-categoría o `Sub_Cat`, variable categórica multi-clase, indica el ciber-ataque concreto. En la Tabla 1 se indican los ataques con su respectivas etiquetas y descripción.

El presente trabajo tiene como objetivo la predicción de ataque, no su clasificación. Por tanto, se usará la variable `Label`.

*NOTA: Parte de la sección 2.1 se ha extraído, re-escrito y adaptado de un trabajo realizado para la asignatura de Técnicas de Reconocimiento de Patrones.*

[1] `https://drive.google.com/drive/folders/1SmLczDUiCfcFb0HzdYM1wUuMoZobsUhp`

| Sub_Cat | Cat | Label | Descripción |
|---|---|---|---|
| Mirai-Ackflooding | Mirai | Anomaly | Ataque que sobrecarga el sistema con paquetes ACK. |
| Mirai-Hostbruteforceg | Mirai | Anomaly | Intento de acceso por fuerza bruta a hosts. |
| Mirai-UDP Flooding | Mirai | Anomaly | Inundación del sistema con paquetes UDP. |
| Mirai-HTTP Flooding | Mirai | Anomaly | Ataque de inundación HTTP para agotar recursos. |
| DoS-Synflooding | DoS | Anomaly | Ataque de inundación SYN para desbordar conexiones. |
| Scan Port OS | Scan | Anomaly | Escaneo de puertos para identificar sistemas operativos. |
| Normal | Normal | Normal | Tráfico legítimo y normal. |
| Scan Hostport | Scan | Anomaly | Escaneo de puertos del host buscando vulnerabilidades. |
| MITM ARP Spoofing | MITM | Anomaly | Suplantación ARP para interceptar comunicaciones. |

Cuadro 1: Descripción de los tipos de ataques presentes en el conjunto de datos IoTID20.

# 2. Redes Neuronales Convolucionales (CNN)

## 2.1. Contexto

### 2.1.1. Concepto de red neuronal

Las redes neuronales son modelos computacionales usados en aprendizaje automático. Toda red neuronal está formada por los siguientes elementos: las capas, las neuronas, y las conexiones.

Las tres capas por las que están compuestas las redes neuronales son: (i) la capa de entrada (*input layer*); (ii) las capas ocultas (*hidden layers*), opcionales; y (iii) la capa de salida (*output layer*) [12]. Las neuronas son la mínima unidad funcional de la red, y actúan como los nodos de la misma. Las conexiones son las encargadas de unir a las neuronas, en función de un peso específico. La manera en la que se establecen estas conexiones esta determinada por la tipología concreta de la red. Véase la Figura 1.

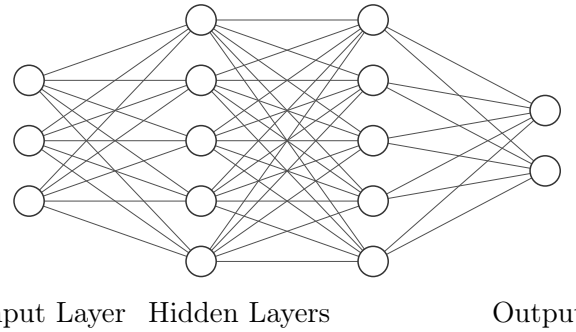


Figura 1: Arquitectura de una red neuronal básica. Fuente: Realizada en NN-SVG.

### 2.1.2. Perceptrón simple

El perceptrón simple es la arquitectura básica de red neuronal. Esta arquitectura cuenta con una sola neurona cuyas entradas quedan definidas por el vector $\mathbf{x} = (x_1, \ldots, x_N)$. Cada entrada lleva asociada un peso específico, dado por el vector de pesos $\mathbf{w} = (w_1, \ldots, w_N)$. Véase la Figura 2.

Esta arquitectura está formada por una sola neurona (define a la vez la capa de entrada y salida). La neurona calcula la suma ponderada del vector de entrada con el vector de pesos. La salida se procesa a través de una función de activación (no lineal), que transforma la entrada a una salida binaria [7].

Las funciones de activación son aplicaciones que transforman la salida, que generalmente son no lineales, continuas, monótonas y tienen un rango acotado. Pueden ser de muchos tipos, siendo comunes las funciones sigmoideas, i.e., una familia funcional de la forma $f(x) = \sigma(x) = \frac{1}{1+e^{-\alpha x}}$ con $\alpha \in \mathbb{R}$, la función hiperboloica, $f(x) = \tanh(x)$, la función signo

$$sgn(x) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x > 0 \end{cases}$$

o la función rectificada uniforme (o $ReLU$, por sus siglas en inglés), $f(x) = \max(0, x)$, entre otras. Originalmente, se usaba era la función signo. [11]

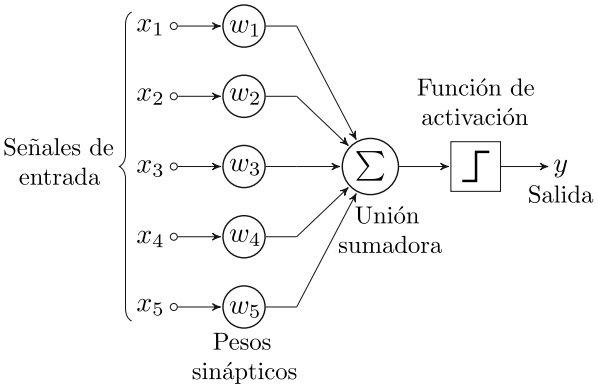


Figura 2: Arquitectura del perceptrón simple o unicapa. [13]

La salida se computa de la siguiente forma:

$$y = sgn(\sum_{i=1}^{N}(x_i w_i) + b) \tag{1}$$

donde $f$ es la función de activación y $b$ es el término de sesgo, introducido por la unión sumadora.

Esta arquitectura fue desarrollada para resolver problemas de clasificación binaria. Se busca encontrar aquellos pesos que den la salida etiquetada o esperada $\mathbf{y^d} = \{y_1^d, \ldots, y_n^d\}$, en un proceso conocido como entrenamiento. En este proceso, se utiliza la siguiente regla de aprendizaje, regida por un sistema dinámico en tiempo discreto: [6]

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \eta(y_i^d - y_i)\mathbf{x_i} \tag{2}$$

siendo $y_i$ la $i$-ésima salida asociada a $x_i$. El factor o tasa de aprendizaje (*learning rate*) se denota como $\eta$, y controla el ritmo al que se actualizan los pesos. Al ser usados en sistemas de clasificación binaria, la salida solo podrá ser 0 ó 1: si $y_i$ es distinta a $y_i^d$, $w_i$ se actualizará; si $y_i = y_i^d$, $w_i$ se mantendrá constante. El sistema converge cuando ya no hay actualización de pesos, $\mathbf{w_{t+1}} = \mathbf{w_t}$. [5]

El problema que tiene esta arquitectura es que solo puede ser usada para problemas linealmente separables. Por ello, se ideó una nueva arquitectura basada en el perceptrón simple que añade nuevas capas, constituyendo el perceptrón multicapa. Esta arquitectura sí puede lidiar con estos problemas al introducir más elementos no lineales. [10]

### 2.1.3. Perceptrón multicapa

En el Perceptrón Multicapa o MLP (*Multi-Layer Perceptron*) la salida de cada neurona en una capa se calcula mediante la suma ponderada de las salidas de las neuronas en la capa anterior, seguida de la aplicación de una función de activación. Este proceso se repite para cada capa. La Figura 1 es el esquema básico de MLP.

Este modelo se entrena siguiendo la regla de retropropagación (*back-propagation*), una generalización del método LMS (*Least-Mean-Square*). El objetivo ya no es clasificar bien todos los puntos (lo que limita su uso a problemas linealmente separables), si no minimizar el error a la salida del sistema. Esto lo hace en base a una función de coste dada por el error empírico básico (PMC) (aunque extrapolable a otras métricas de error):

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - PMC(\mathbf{w}, \mathbf{x_i}))^2 \tag{3}$$

Así, el problema de minimización será encontrar aquel argumento que minimice la función de coste:

$$\mathbf{w}^* = \arg\min E(\mathbf{w})$$

Esta arquitectura se entrena siguiendo un esquema iterativo basado en descenso del gradiente (*gradient descent*) [4]. Es decir, busca en aquella dirección donde pueda haber un mínimo local. Así, el vector de pesos se actualiza siguiendo la expresión:

$$\mathbf{w_{t+1}} = \mathbf{w_t} - \eta \frac{\partial E}{\partial \mathbf{w}}|_{w_t} \tag{4}$$

Cuando la red se computa hacia atrás, este término se obtiene eficazmente. Dado que el perceptrón usa esta regla para su entrenamiento, este esquema también recibe el nombre de Red Neuronal Prealimentada o FNN (*Feed-forward Neural Network*). Su salida se obtiene de la forma siguiente:

Supóngase una FNN de $L$ capas. Se define $a^{(l)}$ como el término de activación de la capa $l$, $a^{(l)} = f^{(l)}(W^{(l)}a^{(l-1)} + b^{(l)})$ donde el primer término es $a^{(1)} = f^{(1)}(W^{(1)}\mathbf{x} + b^{(1)})$ (i.e., la expresión obtenida en la ecuación 1 pero generalizado para todas las neuronas de la primera capa).

La salida en la última capa se expresa como:

$$y = f(W^{(L)}a^{(L)} + b^{(L)}) \tag{5}$$

Equivalentemente:

$$y = f^{(L)}(W^{(L)} \cdot f^{(L-1)}(W^{(L-1)} \dots W^{(1)} \cdot \mathbf{x} + b^{(1)}) + b^{(2)}) + \dots + b^{(L-1)}) + b^{(L)}) \tag{6}$$

donde $f^{(l)}$ es la función de activación de la capa $l$, $W^{(l)} = \{\mathbf{w_1^{(l)}}, \mathbf{w_2^{(l)}}, \dots \mathbf{w_M^{(l)}}\}$ es el conjunto de los vectores de pesos de la capa $l$ para las $M$ neuronas y $b^{(l)} = (b_1^{(l)}, b_2^{(l)}, \dots, b_M^{(l)})$ es el vector de sesgo de la capa $l$.

Las FNN son una arquitectura generalizable a muchos casos de uso y tiene buenos resultados en problemas de regresión y clasificación. Sin embargo, presenta la limitación de que no puede captar información sobre relaciones entre los vectores de entrada. Esto hace que les sea difícil trabajar sobre conjuntos de datos espaciales, temporales o en general, con correlación. Para poner solución a este problema, se desarrolla una nueva arquitectura que puede hacer frente a problemas donde presenten datos de entrada con esta estructura. Esta arquitectura recibe el nombre de red neuronal convolucional o CNN (*Convolutional Neural Network*).

## 2.2. Arquitectura

La arquitectura de estas redes se caracteriza por la inclusión de dos nuevos tipos de capas: las capas convolucionales y las capas de agrupación (*pooling*).

### 2.2.1. Capas convolucionales. Convolución.

Sea la matriz de entrada $I \in \mathbb{R}^{m \times n}$ y $F \in \mathbb{R}^{p \times q}$ un filtro (*filter*) o kérnel con $p < m$ y $q < n$. Se define la operación convolución $C(\cdot)$ para el elemento $(a, b)$ [2] de la matriz de entrada como:

$$C(a,b) = \sum_{i=1}^{m} \sum_{j=1}^{n} I(a+i, b+j) \cdot F(i,j) \tag{7}$$

Aplicado sobre una capa en una red neuronal, se añade el termino de sesgo y la función de activación, siendo la expresión como sigue:

$$C^{(l)}(a,b) = f^{(l)} \left( \sum_{i=i}^{m} \sum_{j=1}^{n} I^{(l)}(a+i, b+j) \cdot F^{(l)}(i,j) + b^{(l)} \right) \tag{8}$$

La salida de las capas convolucionales define un mapa de características.

La intuición de esta operación es sencilla. Obsérvese la Figura 3, donde se muestra este proceso para el primer elemento de la primera fila de la matriz de salida. Para el resto de elementos de la primera fila, bastaría con "desplazar" la *local receptive field* (una sub-matriz de la matriz de entrada del mismo tamaño que el filtro) a las sucesivas columnas multiplicando por el mismo *filter*. Proceso análogo para el resto de filas.

La ventaja de esta operación es que permite captar, no solo dependencias o relaciones entre los elementos de las posiciones próximas, si no también transformaciones en los datos, según cómo se defina el filtro. Es por ello por lo que estas redes son muy populares en aplicaciones de reconocimiento y transformación de imágenes, y visión por computador. [8]

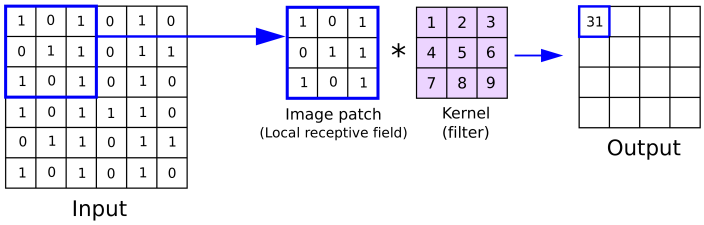


Figura 3: Representación de la operación convolución. Fuente: Wikidocs.

### 2.2.2. Capas de agrupación. *Pooling.*

Las capas de *pooling* en una CNN se utilizan para reducir la dimensionalidad de los mapas de características, decrementando así el número de parámetros a aprender y el costo computacional durante el entrenamiento. Se pueden aplicar varias técnicas de agrupación, siendo las más comunes sobre la media y sobre el máximo (*average pooling* y *max pooling*, respectivamente). La Figura 4 da una idea intuitiva de este proceso.

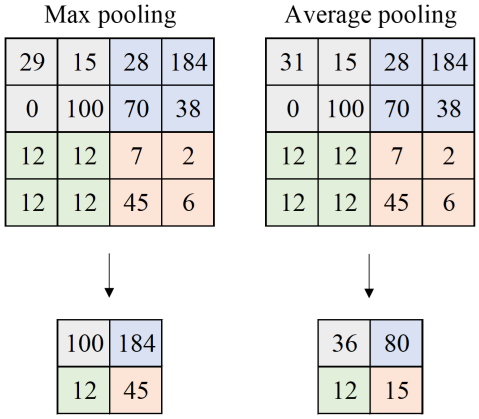


Figura 4: *Max versus Average Pooling.* Fuente: elaboración propia.

La ventaja del uso de esta capa está en la reducción de la dimensionalidad, que permite la extracción de patrones complejos fácil y eficazmente. Es por ello por lo que no es solo útil para datos de vídeo e imágenes, si no también para problemas de clasificación sobre aquellos conjuntos de datos de gran tamaño, sobre las que se quieran extraer sus características fundamentales. [1]

### 2.2.3. Entrenamiento

Para el entrenamiento de estas redes se hace uso nuevamente del método de descenso del gradiente con retropropagación [14]. Aunque en esta arquitectura, los pesos son los valores de los elementos en cada fila y columna de los filtros.

### 2.2.4. Obtención de la salida

Una vez que se han aplicado las capas convolucionales y de *pooling*, se requiere de transformar el mapa de características a un conjunto unidimensional para que pueda ser procesado por la neurona de salida (la que realiza la clasificación). A tal fin, se usa una operación de aplanado (*flatten*). [3]

Una vez se han convertido el conjunto de datos a un vector, se procesa a través de capas densas (*Dense*, dadas por una DNN o *Dense Neural Network*), que no son más que las capas *vanilla* (*fully-connected*) de las redes FNN. [9]

Finalmente, la capa de salida devuelve el resultado de la clasificación. En la Figura 5 se ve un ejemplo de una arquitectura CNN.

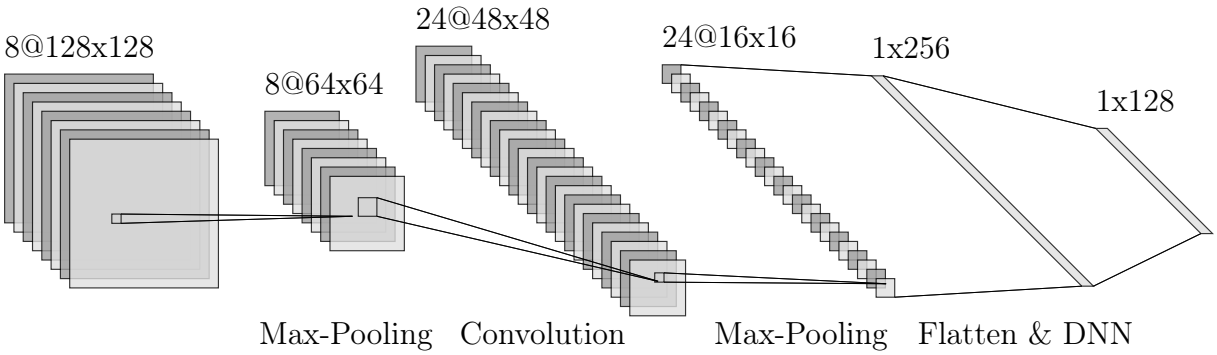


Figura 5: Diagrama de una red neuronal convolucional. Fuente: Realizada en NN-SVG.

La razón por la que se usa esta arquitectura como base para resolver el problema explicado en la sección 1.1 es por su capacidad de explotar la correlación entre las características de los datos de IoT, lo que permite trabajar con un menor número de parámetros en comparación con otros modelos de aprendizaje profundo. Esto reduce el poder computacional requerido y mejora el proceso de aprendizaje, lo cual es esencial para mejorar el rendimiento de los sistemas de detección de intrusiones (IDS) existentes y extender la detección a subcategorías de ataques maliciosos en redes de IoT.

# 3. Metodología

## 3.1. Pre-procesamiento de datos

### 3.1.1. Limpieza de datos

Se sustituyen los valores infinitos en algunos registros de datos. Estos ocurren debidos a ciber-ataques. Para preservar su orden sobre la variable donde se ubican, se sustituyen por un valor alto (`+1e12`), aunque no es infinito. Este paso será importante para la sección 3.1.3. Puesto que no se presentan valores faltantes o atípicos, no se realiza ninguna otra tarea de limpieza de datos.

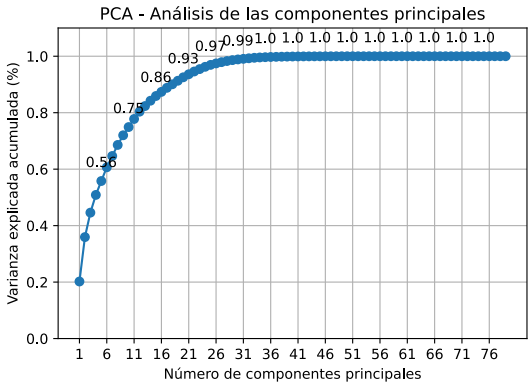


Figura 6: Varianza explicada acumulada por el proceso de PCA.

### 3.1.2. Balanceo

Sobre el conjunto de datos completo, se presenta un 6.4 % de casos donde no hay presencia de ataques de intrusión. Esto implica que el *dataset* está desbalanceado. Por tanto, es necesario llevar a cabo un proceso de compensación o balanceo de datos.

Para llevar a cabo este proceso de balanceo se realiza una estrategia de inframuestreo, muestreo sobre la clase mayoritaria (`Anomaly`) para igualar a la clase minoritaria (`Normal`). Además, se igualan los registros de cada clase de la subcategoría `Sub_Cat` dentro de la etiqueta `Anomaly`. Así, se tienen cerca de 80.000 registros balanceados.

### 3.1.3. Transformación y selección de características

El *dataset* tiene un gran número de características (86 columnas). De este conjunto se eliminan cuatro variables, correspondientes a identificadores que no aportan información relevante: `['Flow_ID', 'Src_IP', 'Dst_IP', 'Timestamp']` (identificador del flujo de información, IP de la máquina origen, IP de la máquina destino e instante de tiempo en el que se ha registrado el flujo, respectivamente).

Para el resto de variables, se puede hacer un proceso de selección de características. Aunque la CNN pueda extraer aquellas características y patrones más relevantes, es conveniente realizar una reducción de su número antes de procesarlas para aumentar la eficiencia computacional en el proceso de entrenamiento.

Para ello, se pueden seguir varias estrategias, de las que se destacan tres: (i) eliminación manual de variables; (ii) regresión por pasos (*stepwise regression*); o (iii) PCA (*Principal Component Analysis*). Es esta última estrategia la que se utilizará, ya que estandariza los datos y agrupa las características sobre componentes, lo que permitirá a la CNN captar más eficientemente patrones y dependencias que ayuden a la tarea de predicción.

#### Aplicación del PCA

El PCA es un método de reducción de dimensionalidad que agrupa las variables en componentes sobre un espacio vectorial de menor dimensión, según la información que comparten las componentes.

Para llevar a cabo este método, primeramente se realiza una estandarización de los datos, y posteriormente se ajusta un modelo de PCA al conjunto de características. Se elige el número de características principales como aquel número de componentes 95 % de la varianza acumulada (obsérvese la Figura 6). En este caso, se ha reducido el conjunto de datos de 83 características a tan solo 23 componentes (60 columnas menos sobre el dataset).

## 3.2. Entrenamiento de la CNN

### 3.2.1. Definición, creación e instanciación del modelo

El modelo se construye sobre la librería `Keras` y `Tensorflow` en Python en base a las siguientes capas:

- Una capa convolucional de una dimensión (`Conv1D`). En esta capa, se pueden tunear el número de filtros (`filters`), el tamaño del filtro (`kernel_size`), y la función de activación (`activation`). Se usa una dimensión ya que las componentes tienen forma de vector, no matriz.
- Una capa de agrupación por máximos (`MaxPooling1D`). En esta capa, se puede tunear el tamaño de la agrupación (`pool_size`).
- Una capa de aplanado (`Flatten`), para agrupar todos los filtros sobre un vector unidimensional.
- Una red densa (`Dense`), de arquitectura *fully-connected*. En esta capa, se puede tunear la función de activación y las neuronas (`dense_units`)

Las funciones de activación que se usarán son la ReLU (`activation = relu`), para las capas convolucionales, debido a sus propiedades de no saturación del gradiente, y su eficiencia computacional; y la sigmoide para la capa de salida, aunque se podría usar cualquiera de salida binaria.

### 3.2.2. Conjuntos de prueba y entrenamiento. Validación cruzada (*Cross-validation*)

Para el entrenamiento de la CNN, se divide el conjunto de datos en prueba y entrenamiento, usando validación cruzada. La validación cruzada consiste en dividir el conjunto de entrenamiento en $k$ particiones o bloques (*folds*). Posteriormente, se entrena el modelo varias veces, utilizando en cada ocasión un bloque diferente como conjunto de prueba y el resto de los bloques combinados como conjunto de entrenamiento. Este proceso tiene como objetivo reducir el sesgo asociado a la elección de los conjuntos de prueba y entrenamiento. En este caso, se selecciona que el 80 % de los datos sean de entrenamiento y el 20 % de prueba.

Obsérvese el ejemplo ilustrativo de la Figura 7.

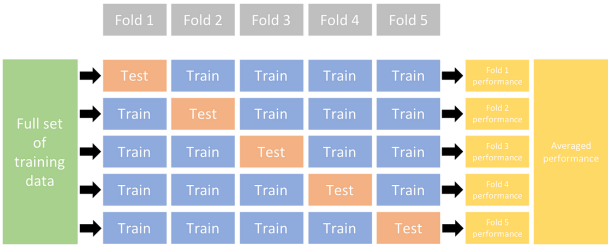


Figura 7: Proceso de validación cruzada para 5 estratos (división 80 %-20 %).

### 3.2.3. Elección de hiperparámetros

Para el resto de hiperparámetros, se realiza un proceso de búsqueda conocido como *Grid Search*. Este proceso involucra definir en una malla todos los valores que se quiere probar de hiper-parámetros e instanciar al modelo probando todas las posibles combinaciones posibles.

Después, se selecciona aquel modelo que mejor rendimiento ha obtenido con respecto a una métrica. En este caso, se selecciona como métrica la precisión de validación o `val_accuracy`, a la que se le aplica el proceso de validación cruzada ya explicado. La precisión de validación mide la capacidad del modelo de predecir correctamente las etiquetas de un conjunto de datos de validación que no se utilizó durante el entrenamiento.

$$\text{Validation accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

donde $TP$, $TN$, $FP$ y $FN$ es el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos sobre el conjunto de entrenamiento, respectivamente.

## 3.3. Evaluación del modelo

Para evaluar el rendimiento del modelo se usan las siguientes métricas:

- *Accuracy*: número de predicciones correctas entre el total de predicciones.
- *Precision*: proporción de verdaderos positivos (TP) entre todos los casos que el modelo ha predicho como positivos, tanto falsos como verdaderos (TP + TN). Precision $= \frac{TP}{TP+TN}$.
- *Recall*: tasa de verdaderos positivos sobre verdaderos positivos y falsos negativos Recall $= \frac{TP}{TP+FN}$.
- *F1-Score*: medida que combina la precisión y el recall (sensibilidad) en un solo valor, proporcionando un balance entre ambos. F1-Score $= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

## 3.4. Arquitectura completa

Finalmente, en la Figura 8 se ilustra el flujo de trabajo completo para llevar a cabo la metodología ya descrita.

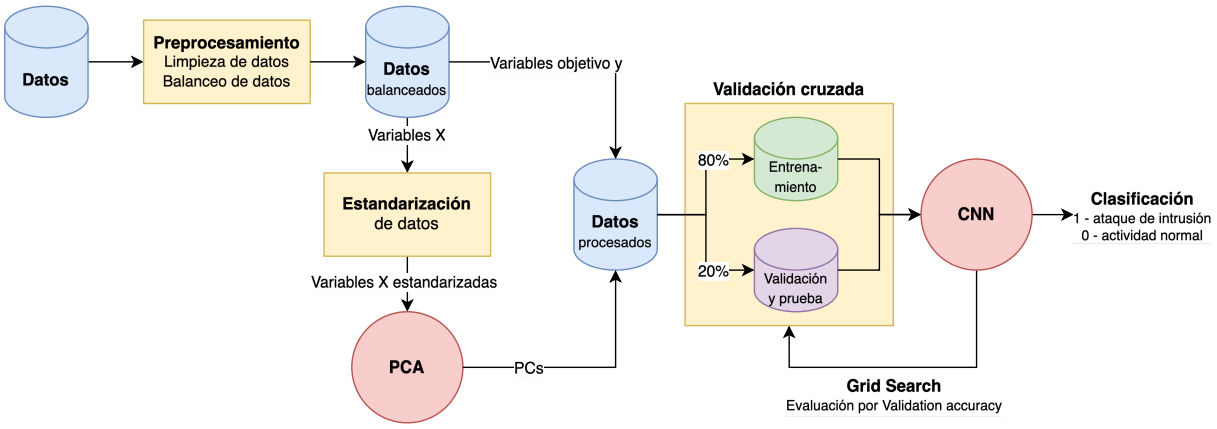


Figura 8: Flujo de trabajo completo.

# 4. Resultados

Tras realizar toda la metodología anterior, se encuentra que el mejor modelo está dado por los siguientes hiperparámetros: `{'filters': 128, 'kernel_size': 8, 'pool_size': 8, 'dense_units': 100}`.

Se aplica el modelo y se observan el siguiente rendimiento sobre las métricas descritas en la sección 3.3. Las métricas de clasificación, superiores en todos los casos al 95 % en todas las métricas, demuestra la bondad de ajuste del modelo en esta tarea, así como en predicción. Por lo que se puede concluir que la metodología de análisis de datos así como de entrenamiento del modelo es adecuada.

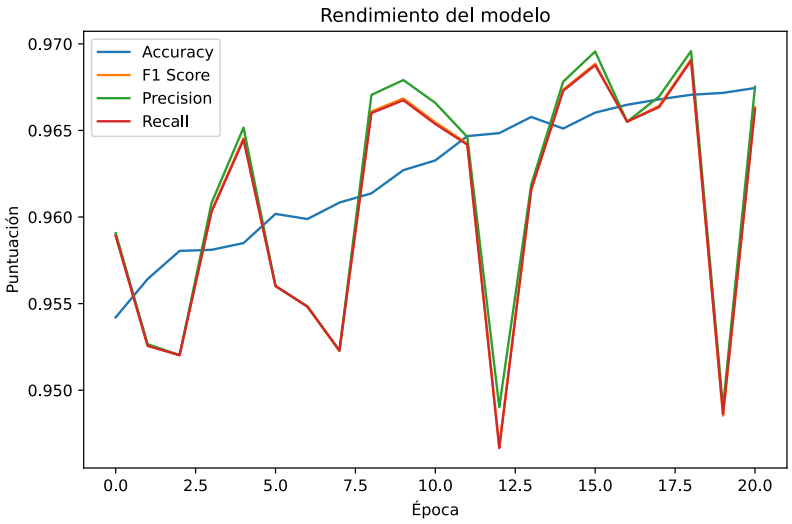


Figura 9: Evaluación de las métricas sobre las épocas para el mejor modelo.

Código del proyecto: GitHub

# Referencias


[1] Nadeem Akhtar y U Ragavendran. "Interpretation of intelligence in CNN-pooling processes: a methodological survey". En: *Neural computing and applications* 32.3 (2020), págs. 879-898.

[2] Saad Albawi, Tareq Abed Mohammed y Saad Al-Zawi. "Understanding of a convolutional neural network". En: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, págs. 1-6.

[3] Muhammad Shoaib Ali. *Flattening CNN layers for Neural Network and basic concepts*. `https://medium.com/@muhammadshoaibali/flattening-cnn-layers-for-neural-network-694a232eda6a`. Accessed: 1 de marzo de 2024. 2022.

[4] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". En: *Neurocomputing* 5.4-5 (1993), págs. 185-189.

[5] Marcus Frean. "A"thermal"perceptron learning rule". En: *Neural Computation* 4.6 (1992), págs. 946-957.

[6] S Haykin. "Neural Networks and Learning Machines, edit Prentice Hall". En: *New Jersey, USA* (2008).

[7] Laveen N Kanal. "Perceptron". En: *Encyclopedia of Computer Science*. ACM Digital Library, 2003, págs. 1383-1385.

[8] Zewen Li et al. "A survey of convolutional neural networks: analysis, applications, and prospects". En: *IEEE transactions on neural networks and learning systems* (2021).

[9] Keiron O'Shea y Ryan Nash. "An introduction to convolutional neural networks". En: *arXiv preprint arXiv:1511.08458* (2015).

[10] Marius-Constantin Popescu et al. "Multilayer perceptron and neural networks". En: *WSEAS Transactions on Circuits and Systems* 8.7 (2009), págs. 579-588.

[11] Sagar Sharma, Simone Sharma y Anidhya Athaiya. "Activation functions in neural networks". En: *Towards Data Sci* 6.12 (2017), págs. 310-316.

[12] Sun-Chong Wang y Sun-Chong Wang. "Artificial neural network". En: *Interdisciplinary computing in java programming* (2003), págs. 81-100.

[13] Colaboradores de Wikipedia. *https://es.wikipedia.org/wiki/Perceptr%C3%B3n — Wikipedia, The Free Encyclopedia*. Online; Accedido el 10 de febrero de 2024. 2024. URL: `https://es.wikipedia.org/wiki/Perceptr%5C%C3%5C%B3n`.

[14] Zhifei Zhang. "Derivation of backpropagation in convolutional neural network (cnn)". En: *University of Tennessee, Knoxville, TN* 22 (2016), pág. 23.