Aplicación de Redes Multicapa a problemas de regresión en el Dataset sobre la demanda de bicicletas compartidas en Seúl.

Isabella Herrarte López, Gabriel Jeannot Viaña, Valentina Loaiza Mejia, Andrea Saavedra Viveros

Facultad de Ingeniería, Universidad Autónoma de Occidente  
Cali, Valle del Cauca, Colombia

[Isabella.herrarte@uao.edu.co](mailto:Isabella.herrarte@uao.edu.co)  
[gabriel.jeannot@uao.edu.co](mailto:gabriel.jeannot@uao.edu.co)  
[valentina.loaiza@uao.edu.co](mailto:valentina.loaiza@uao.edu.co)  
[Andrea.saavedra@uao.edu.co](mailto:Andrea.saavedra@uao.edu.co)

**Abstract**— This paper will provide a guide to train a Multi-Layer Perceptron (MLP) on Tensorflow2-Keras in order to solve the regression problem in the Dataset on the demand for bicycle sharing in Seoul, South Korea. The report will include information on the network architecture, optimizers used, data processing, loss visualization in TensorBoard, and validation of the obtained models. Experimental validation of the trained network will be performed on an Arduino to analyze the results obtained between Google Colab (using Python) and Arduino. In addition, it will be illustrated which of all architectures has a better prediction of the data behavior.

**Keywords**—Arduino; Data Set Bikes; Keras; MLP Network; Neural Networks; Regression Problem; TensorBoard; Tensorflow.

**Resumen**—El objetivo de este documento es proporcionar una guía para entrenar un Perceptrón Multicapa (MLP) en Tensorflow2-Keras, con el fin de resolver el problema de regresión en el Dataset sobre la demanda de bicicletas compartidas en Seúl, Corea del Sur. El informe incluirá información sobre la arquitectura de la red, los optimizadores utilizados, el procesamiento de datos, la visualización de la pérdida en TensorBoard y la validación de los modelos obtenidos. La validación experimental de la red entrenada se llevará a cabo en un Arduino para analizar los resultados obtenidos entre Google Colab (utilizando Python) y Arduino. Además, se ilustrará cuál de todas las arquitecturas tiene una mejor predicción del comportamiento de los datos.

**Palabras Clave**—Arduino; Data Set Bikes; Keras; Red MLP; Redes Neuronales; Regresión; TensorBoard; Tensorflow.

1. INTRODUCCIÓN

Con la llegada de la Inteligencia Artificial, la ingeniería ha evolucionado la forma en la que aborda los desafíos sociales e industriales, razón por la cual ahora se han estado utilizando herramientas que optimizan y generan soluciones metodológicas más ágiles, seguras y poderosas. Anteriormente, los procesos complejos eran resueltos por los humanos de manera empírica o con mucho esfuerzo y preparación, mientras que, actualmente, existen muchas herramientas que agilizan el trabajo. Según NetApp, una empresa de gestión de datos y servicios en la nube, la Inteligencia Artificial (IA) es la base para imitar los procesos de inteligencia humana mediante la creación y aplicación de algoritmos en un entorno dinámico de computación [1]. Por tal motivo, en la ingeniería se ha estado usando el Machine Learning y la Inteligencia Computacional, las cuales tienen diversos contextos de aplicación o formas para realizar la IA.

El Machine Learning es una forma de Inteligencia Artificial que utiliza algoritmos para descubrir patrones recurrentes en diferentes tipos de datos, como números, palabras, imágenes y estadísticas; y que, a partir de un aprendizaje, se busca mejorar el rendimiento en la ejecución de una tarea específica [2]. Por otro lado, el uso de algoritmos en el Machine Learning presenta varios problemas, siendo los más relevantes la Regresión, la Clasificación, la Identificación de Similitudes y el Clustering. Se puede decir que la Regresión es una subtarea del Aprendizaje Supervisado en el Machine Learning que busca establecer una relación entre características y una variable objetivo continua mediante el uso de Redes Neuronales, las cuales se encuentran dispuestas en capas para que iterativamente aprendan los datos.

De acuerdo con lo anterior, el objetivo de este documento es entrenar una Red Multicapa Superficial en Tensorflow2-Keras para resolver el problema de regresión del Dataset sobre la demanda de bicicletas compartidas en Seúl; esto último con el fin de analizar el comportamiento del conjunto de datos y obtener la predicción del número de bicicletas necesarias en cada hora para proporcionar un suministro estable de bicicletas de alquiler en la capital de Corea del Sur. Por otro lado, se puede decir que este proyecto es un desafío en el curso de Redes Neuronales y Deep Learning, el cual permitirá al equipo de trabajo adquirir conocimientos experimentales sobre el uso de Redes Neuronales Multicapa en problemas de regresión. Este conocimiento puede aplicarse en la vida cotidiana para solucionar problemas como la predicción de la estatura de las personas, el pronóstico del consumo de un producto en los siguientes meses o incluso el cálculo de las importaciones de un país, entre otros.

1. MARCO TEÓRICO

El desarrollo del proyecto requiere una variedad de conocimientos y fundamentos que abarcan desde la importación de librerías para el desarrollo del Notebook en Google Collaboratory hasta la validación experimental de los modelos en Arduino. Por lo tanto, el marco teórico del proyecto se divide en dos secciones principales: el Notebook de Google Collaboratory, que incluye el archivo adjunto en el pie de página, y la implementación en Arduino, que también se presentará como un anexo al documento.

En el caso del Notebook, se especificarán las bibliotecas que se emplearán para escribir el código:

**Pandas** es un paquete de Python que facilita el trabajo con datos etiquetados o relacionales, proporcionando estructuras de datos rápidas, flexibles y expresivas. En este caso particular, se utilizará como una herramienta para cargar el conjunto de datos en formato .csv [3].

**Numpy** es una biblioteca de Python que incluye un objeto de matriz de N dimensiones, junto con funciones avanzadas como la transmisión (broadcasting) y herramientas útiles para el álgebra lineal, la transformada de Fourier y la generación de números aleatorios [4].

**Matplotlib** es una biblioteca en Python que ofrece una amplia variedad de herramientas para la creación de gráficos interactivos, animados y estáticos. Es capaz de generar figuras de alta calidad para su uso en diferentes formatos, tanto para entornos impresos como interactivos, y puede ser utilizada en diversos contextos, desde scripts hasta servidores web y kits de herramientas de interfaz gráfica de usuario [5].

**Seaborn** es una biblioteca de Python que se basa en matplotlib para la visualización de datos. Se utiliza para crear gráficos estadísticos con una interfaz de alto nivel y atractivos diseños [6].

Además de ello, para cumplir con los requisitos de la guía, se incluirán referencias teóricas relacionadas con el entrenamiento de redes neuronales con Tensorflow2-Keras, los optimizadores que se emplearán y las arquitecturas que se considerarán en el proceso.

Para la adecuación de la red neuronal, se hacen uso de ciertas **librerías** específicas en Python, tales como:

import tensorflow as tf

TensorFlow es una biblioteca de código abierto desarrollada por Google para el aprendizaje automático. Esta biblioteca se utiliza para construir y entrenar redes neuronales capaces de detectar patrones y correlaciones en datos, similares al aprendizaje y razonamiento humano [7]. TensorFlow se utiliza en una amplia variedad de aplicaciones, desde la clasificación de imágenes hasta el procesamiento del lenguaje natural y la robótica.

from tensorflow.keras.models import Sequential

Un modelo *Secuencial* se emplea cuando se requiere implementar un conjunto simple de capas neuronales donde cada capa tiene un único tensor de entrada y un único tensor de salida. Esto permite la incorporación de características específicas de entrenamiento e inferencia en el modelo, que se definen capa por capa en orden [8].

from tensorflow.keras.layers import Dense

La capa *Dense* es una de las capas más comunes y ampliamente utilizadas en las redes neuronales profundas. Se trata de una capa regularmente conectada en la que cada neurona de la capa anterior está conectada a cada neurona de la capa siguiente [9]. Esta capa también se conoce como capa totalmente conectada, ya que cada neurona de una capa está conectada con todas las neuronas de la siguiente capa. En resumen, la capa Dense es responsable de aplicar una transformación lineal a los datos de entrada y agregar una función de activación para producir la salida de la capa.

Ahora, se explicará cómo se fundamentan los optimizadores utilizados para crear redes neuronales en el Notebook de Colab, de manera similar a como se explicaron las librerías anteriores. Un optimizador es una herramienta que busca encontrar los valores de los parámetros de una red neuronal con el objetivo de reducir el error en la propagación de la información; el proceso por el cual se logra esto se llama "backpropagation", y, en este, el optimizador actualiza los valores de los parámetros de manera equitativa en la red, basándose en la tasa de aprendizaje [9]. Para sustentar teóricamente los optimizadores que se utilizarán, se hace referencia al apartado de optimizadores [10] en la página web oficial de Keras.

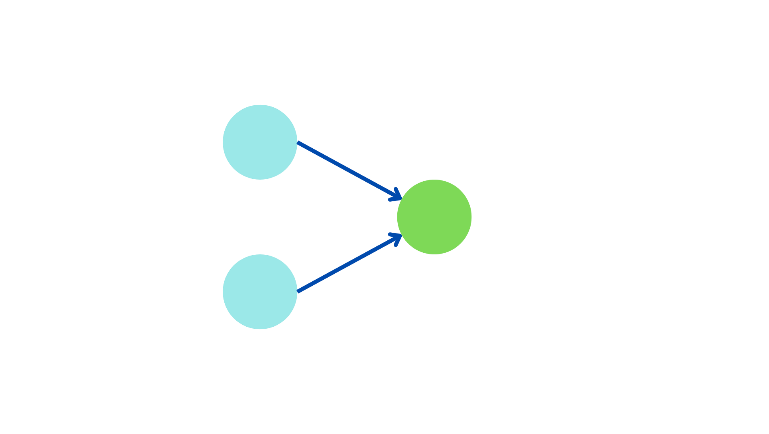
**Adam**, es un optimizador que utiliza una técnica basada en el gradiente descendente estocástico para realizar el ajuste de los parámetros en una red neuronal. Este método se basa en la estimación adaptativa de momentos de primer y segundo orden, lo que lo hace computacionalmente eficiente y requiere poca memoria. Además, es capaz de mantener su eficacia a pesar de que haya cambios en la escala diagonal de los gradientes, lo que lo hace adecuado para problemas que involucren un gran número de parámetros o datos [11].

**Adamax**, es una variante de Adam que utiliza la norma del infinito. Los valores por defecto de los parámetros se basan en los proporcionados en el documento. En algunos casos, Adamax puede ser superior a Adam, especialmente en modelos que involucren incrustaciones [12].

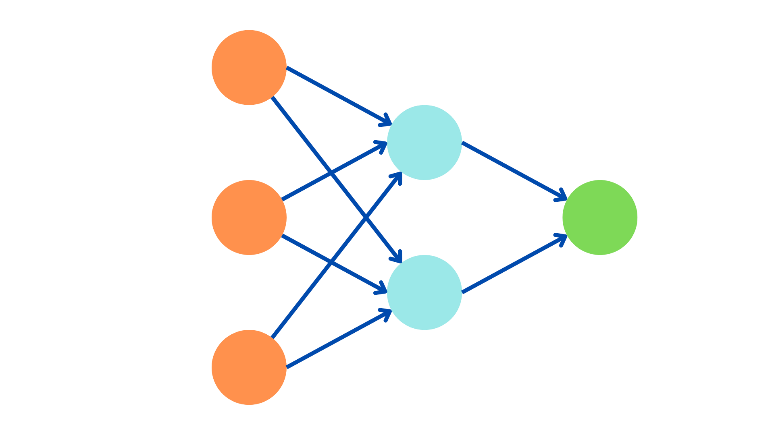
**Nadam**, es un optimizador que utiliza un algoritmo similar a Adam. En concreto, es esencialmente RMSprop con impulso, pero con la adición del impulso Nesterov, lo que se denomina Momento Acelerado Adaptativo de Nesterov [13].

Tanto los optimizadores como la morfología de la red son aspectos clave para crear diferentes modelos de redes neuronales. Según InteractiveChaos, el concepto de Arquitectura en una red neuronal no se refiere solo al número de capas o de neuronas en cada una, sino también a la conexión entre ellas, al tipo de neuronas que se utilizan y la forma en que se entrenan. En este sentido, existen diferentes tipos de neuronas, como la celda de entrada, la celda oculta y la celda de salida.

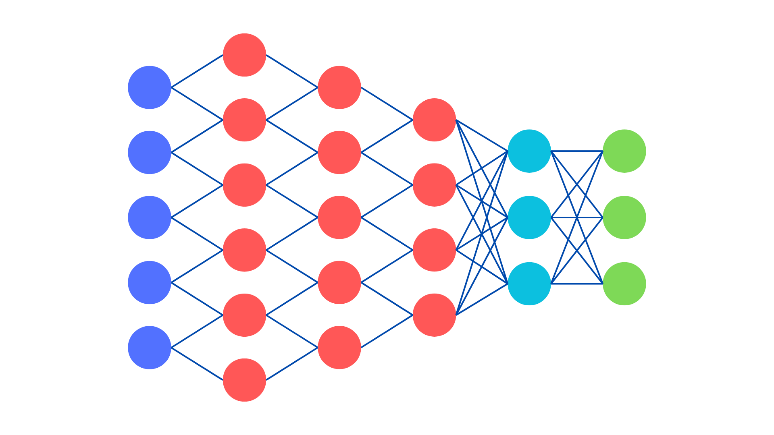
**Perceptrón**, es la arquitectura más simple posible en una red neuronal. El perceptrón consta de dos celdas de entrada y una celda de salida. Los datos de entrada llegan a las celdas de entrada y, se procesan en la celda de salida mediante una media ponderada y una función de activación, y se devuelve el valor resultante.



**Redes neuronales de alimentación directa (feed forward networks)**, son una extensión del perceptrón. Estas redes consisten en varias capas de neuronas, todas ellas conectadas entre sí de manera completa (es decir, una celda está conectada con todas las celdas de la siguiente capa). Hay al menos una capa oculta en estas redes, la información circula en una sola dirección (de izquierda a derecha), y su entrenamiento se suele realizar mediante backpropagation, que se explicará más adelante.



**Redes Neuronales Convolucionales (Convolutional Neural Networks o CNN)**, también conocidas como Redes Neuronales Convolucionales Profundas (Deep Convolutional Neural Networks o DCNN), estas se utilizan principalmente para procesamiento de imágenes. Las capas de estas redes (que no siempre están formadas por neuronas) se pueden dividir en dos bloques: el primer bloque está formado principalmente por capas convolucionales y de agrupación (pooling), y su objetivo es identificar patrones visuales, mientras que el segundo bloque se enfoca en la clasificación de los datos que se reciben.



En resumen, el equipo decidió trabajar en la implementación de una red neuronal utilizando el conjunto de datos de demanda de bicicletas compartidas de Seúl (Bike Sharing Demand Data [14]), que incluye información sobre el número de bicicletas alquiladas por hora, datos meteorológicos y días festivos.



*Ilustración 1 Alquiler de bicicletas en Seúl*

La fuente de referencia presenta las siguientes características en el conjunto de datos obtenidos:

● Características del conjunto de datos: Multivariante

● Número de instancias: 8760

● Área: Informática

● Características de los atributos: Entero, Real

● Número de atributos: 14

● Fecha de donación 2020-03-01

● Tareas asociadas: Regresión

● ¿Valores perdidos? N/A

● Número de visitas a la web: 81774

**Problemática**. En Corea del Sur, en el año 2020, se introdujeron bicicletas de alquiler en muchas ciudades urbanas para mejorar la movilidad de las personas. Por lo tanto, es crucial que estas bicicletas estén disponibles y accesibles para el público en el momento adecuado para reducir el tiempo de espera. Con el tiempo, proporcionar un suministro estable de bicicletas de alquiler se convirtió en una preocupación importante, y es por eso que se propone la idea de predecir la cantidad de bicicletas necesarias para garantizar un suministro estable de bicicletas de alquiler en la capital, Seúl.



*Ilustración 2 Alquiler de bicicletas en Seúl*

**Acercamiento e interpretación de la Data.** Se decide señalar que el conjunto de datos contiene información sobre las condiciones meteorológicas (temperatura, humedad, velocidad del viento, visibilidad, punto de rocío, radiación solar, nieve, precipitaciones) del momento en que se registraron los datos, junto con el número de bicicletas alquiladas por hora y la fecha correspondiente. Esta información se presenta en una lista que incluye:

* **Date**: year-month-day.
* **Rented Bike count**: Cantidad de bicicletas alquiladas por hora en la ciudad.
* **Hour**: Hora del día.
* **Temperature**: se refiere a la medida de la energía térmica presente en un objeto, medida en grados Celsius (°C). Es una variable física que indica la cantidad de energía térmica presente en un cuerpo [15].
* **Humidity**: se refiere al porcentaje de humedad en el aire, que es la cantidad de vapor de agua presente en relación con la cantidad máxima que podría estar presente cuando el aire está saturado [16].
* **Windspeed**: Velocidad del viento en m/s
* **Visibility**: Visibilidad, 10 m.
* **Dew Point Temperature:** se refiere a la temperatura, medida en grados Celsius (°C), a la cuál el aire debe enfriarse a presión constante para alcanzar una humedad relativa del 100%. Es decir, es la temperatura a la que el vapor de agua en el aire comienza a condensarse en forma de rocío.
* **Solar radiation:** MJ/m2, correspondiente a la energía emitida por el Sol, que se propaga en todas las direcciones a través del espacio mediante ondas electromagnéticas.
* **Rainfall:** Precipitación, en mm, es la fase del ciclo hidrológico que consiste en la caída de agua desde la atmósfera hacia la superficie terrestre.
* **Snowfall:** se refiere a la cantidad de nieve, medida en centímetros, que ha caído en un lugar específico. La nieve es un tipo de precipitación que consiste en pequeños cristales de hielo que se forman a partir de la congelación de partículas de agua en suspensión en la atmósfera. Estos cristales se agrupan y caen a la superficie terrestre en forma de copos blancos. En ciertas condiciones de temperatura, estos copos pueden acumularse y formar una capa de nieve sobre la superficie terrestre.
* **Seasons:** Winter, Spring, Summer, Autumn, las cuales se traducen como: Invierno, primavera, verano, otoño.
* **Holiday:** Vacaciones/ No vacaciones, que es un dato binario de las personas de la ciudad en el momento de usar la bicicleta.
* **Functional Day - NoFunc (Non Functional Hours), Fun (Functional hours):** el cual corresponde a las Horas funcionales y no funcionales de las personas de la ciudad.

1. *Planteamiento del enunciado*

Teniendo en cuenta la información anterior, la Guía del Mini proyecto del Curso, propone entrenar una Red MLP superficial en Tensorflow2-Keras que permita resolver el problema de regresión previamente definido donde dicha red entrenada deberá ser emulada en Arduino como plataforma de implementación final.

1. *Metodología*

Para llevar a cabo el proyecto, se seguirá un proceso que comienza con el entrenamiento de la red neuronal en Tensorflow2-Keras. Durante este proceso, se probarán al menos tres optimizadores y tres arquitecturas diferentes de red neuronal con un máximo de tres capas ocultas. Después, se evaluará el rendimiento de la red neuronal calculando el coeficiente de regresión obtenido; una vez que se haya entrenado adecuadamente, se implementará en Arduino para asegurarse de que la red funcione correctamente, esto último verificando que las entradas necesarias para generar la salida se ajusten adecuadamente, lo que debería resultar en una salida que coincida con la obtenida en Tensorflow2-Keras.

Algunos elementos que el equipo de trabajo decide llevar a cabo para desarrollar el proyecto, constan de:

1. Entendimiento del Dataset seleccionado.
2. Partición del Dataset en datos de entrenamiento y validación.
3. Normalización de las variables del Dataset.
4. Entrenamiento del modelo neuronal y la selección de su arquitectura.
5. Validación de los modelos neuronales trabajados para seleccionar el mejor.
6. Validación del mejor modelo neuronal implementado en Arduino, verificando que su comportamiento es igual a la entrada en Keras.

La solución se llevará a cabo siguiendo los mismos pasos que se presentaron en el marco teórico, utilizando dos archivos: uno correspondiente al Notebook de Google Collaboratory y otro a la implementación en Arduino. No obstante, en esta sección se incluirán algunos pasos intermedios necesarios para completar el proceso.

1. *Notebook de Google Colab*

En relación al Google Colab, se mostrarán fragmentos de código que servirán para explicar la secuencia o método utilizado para solucionar el problema.

En primer lugar, para llevar a cabo el **Entendimiento de la Data seleccionada**, se descarga el conjunto de datos sobre el recuento de bicicletas públicas alquiladas en cada hora en el sistema de alquiler de bicicletas de Seúl, junto con datos meteorológicos y sobre las vacaciones en la ciudad de Seúl. El conjunto de datos está disponible en el repositorio de UCI Machine Learning y contiene 8070 instancias con 14 atributos diferentes. Según la documentación, se trata de un problema de regresión cuyo objetivo es predecir el número de bicicletas necesarias en cada hora para proporcionar un suministro estable de bicicletas de alquiler en la capital de Corea del Sur.

Se procede a *importar las bibliotecas* necesarias para el desarrollo del mini-proyecto. Pandas se utilizará para cargar el conjunto de datos en formato .csv, numpy para manipular arreglos, matplotlib.pyplot para crear gráficas y seaborn para obtener herramientas como pairplot y heatmap para analizar la correlación entre los datos.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sn

Después, se procede a *cargar los datos del archivo* SeoulDataBike.csv utilizando la ubicación del archivo y especificando el tipo de caracteres cp1252 que se utilizan en la base de datos.

data=pd.read\_csv('/content/SeoulBikeData.csv',encoding='cp1252')

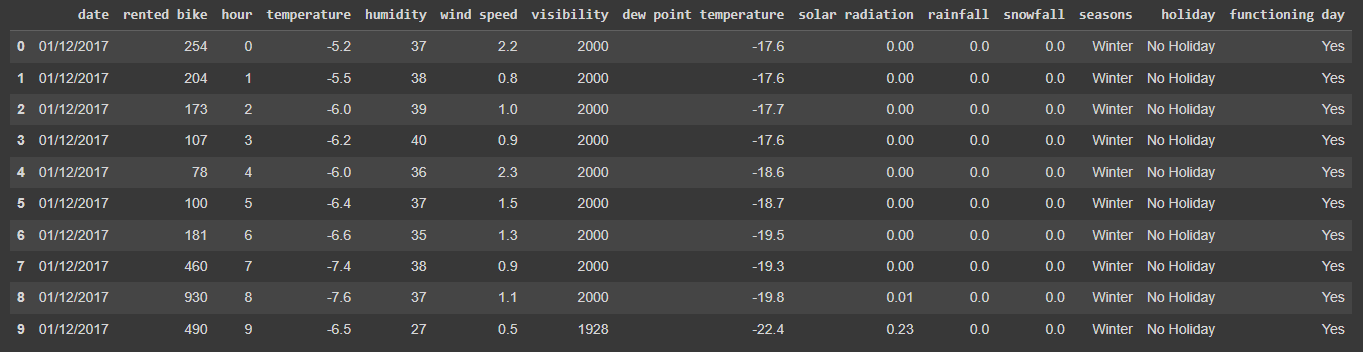
print(data.shape)

data.head(10)

Ahora, se realiza *una modificación en los nombres* de las columnas con el objetivo de facilitar el acceso a cada una de ellas, lo cual se logra eliminando ciertos detalles como la unidad y las letras mayúsculas presentes en los nombres originales

data.columns=['date','rented bike','hour','temperature','humidity','wind speed','visibility','dew point temperature','solar radiation','rainfall','snowfall', 'seasons', 'holiday', 'functioning day']

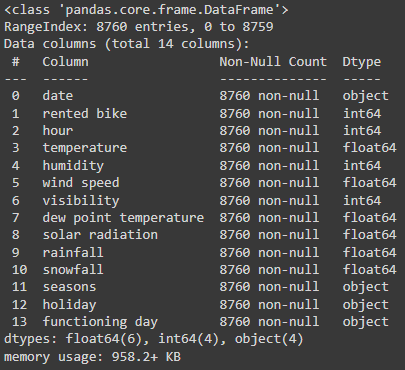
data.head(10)



*Ilustración 3 Modificación en los nombres de las columnas*

Se realiza una verificación para *determinar si existen valores nulos en los datos y en caso de existir, cuántos son*. Se concluye que no hay valores faltantes en ninguna de las columnas de la base de datos. Esto se realiza de la siguiente manera:

data.info()



*Ilustración 4 Verificación de valores nulos*

Después de verificar que *no hay datos nulos en la base de datos*, se procede a realizar un proceso de filtrado para eliminar los datos innecesarios. Se comienza eliminando las variables de tipo *Date* y luego se revisan los datos de tipo *String*.

data = data.drop(['date'], axis=1)

print(pd.value\_counts(np.array(data['functioning day'])))

print(" ")

print(pd.value\_counts(np.array(data['holiday'])))

print(" ")

print(pd.value\_counts(np.array(data['seasons'])))

Luego de realizar el proceso de filtrado, se observó que la mayoría de los datos de las variables "Holiday" (Holiday:8328 y No Holiday:432) y "Functioning Day" (Yes:8465 y No:295) se encuentran, en su mayoría, en un único valor, a diferencia de la variable "Season" (Winter: 2160, Spring: 2208, Summer: 2208 y Autumn: 2184), que presenta una distribución casi uniforme de sus datos. Por esta razón, se asigna un entero del 0 al 3 para diferenciar las diferentes estaciones: Winter: 0, Spring: 1, Summer: 2 y Autumn: 3.

data['seasons'][data['seasons'] == 'Winter'] = 0

data['seasons'][data['seasons'] == 'Spring'] = 1

data['seasons'][data['seasons'] == 'Summer'] = 2

data['seasons'][data['seasons'] == 'Autumn'] = 3



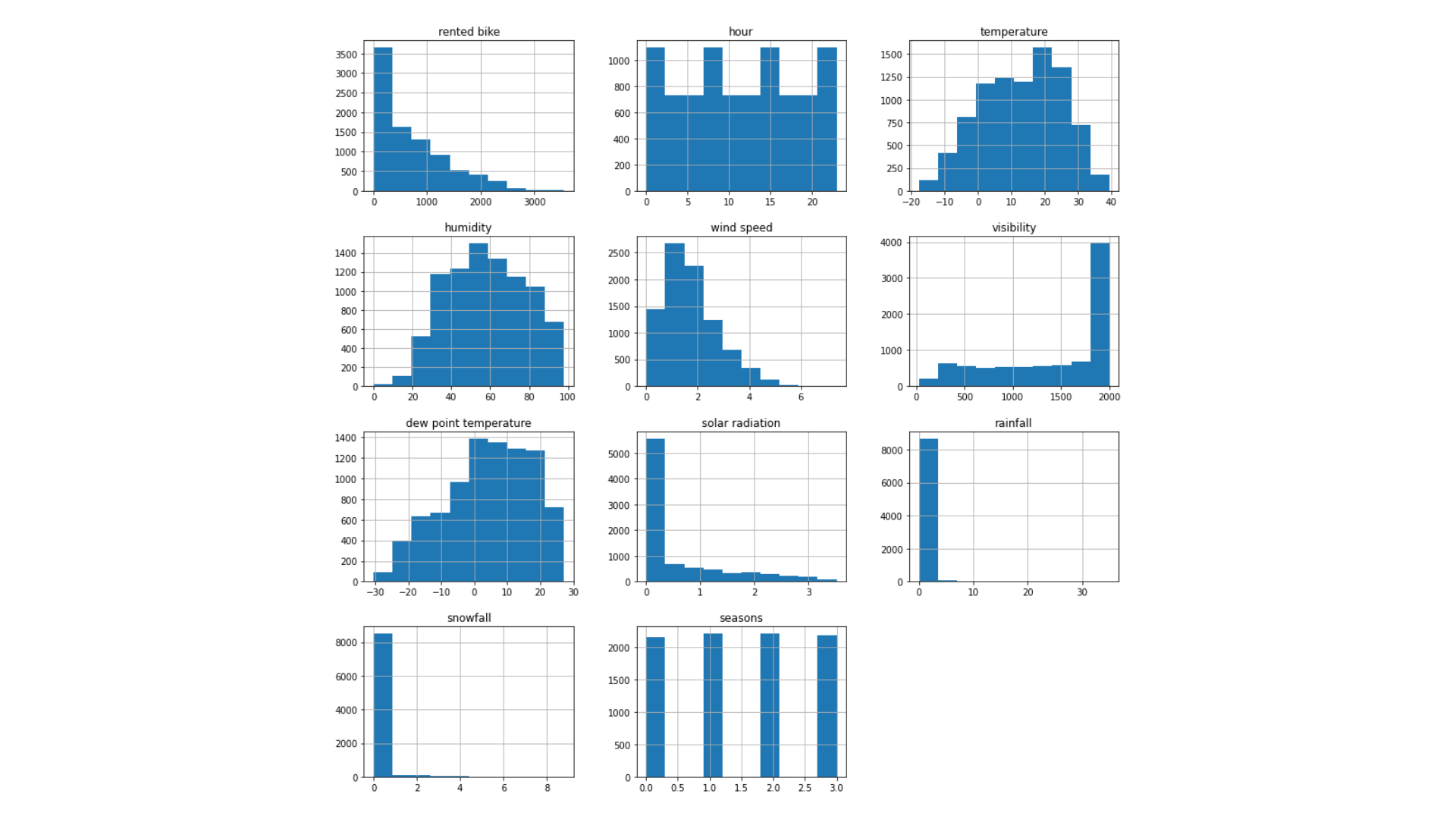
*Ilustración 5 Estaciones del año*

Teniendo en cuenta lo anterior, se decide omitir del análisis Holiday y Functioning Day, así:

data = data.drop(['holiday','functioning day'], axis=1)

Después, se crea un histograma que muestra que los atributos de Rainfall (8232 datos en 0.0) y Snowfall (8317 datos en 0.0) contienen la mayoría de sus datos en cero. Por lo tanto, se decide eliminarlos ya que no son relevantes para el análisis.

data.hist(figsize=(15,15))



*Ilustración 6 Histogramas obtenidos*

Para justificar lo mencionado anteriormente, se emplean las siguientes líneas de código con el fin de contar la cantidad de ocurrencias del valor 0 en los atributos Rainfall y Snowfall. Después de examinar el comportamiento de los datos, se utiliza la función "drop" para eliminar la información no relevante durante el proceso:

count = np.count\_nonzero(data['rainfall']==0.0)

print('rainfall = 0 ->', count)

count = np.count\_nonzero(data['snowfall']==0.0)

print('snowfall = 0 ->', count)

data = data.drop(['rainfall', 'snowfall'], axis=1)

Después de eliminar la información no relevante, se procede a *normalizar los datos*. El objetivo de este proceso es tener un rango de operación uniforme entre [-1,1], similar al que se puede obtener en MATLAB, lo que permite una mayor distribución de los datos. Para lograr esto, se utiliza una función que normaliza cada atributo de la base de datos.

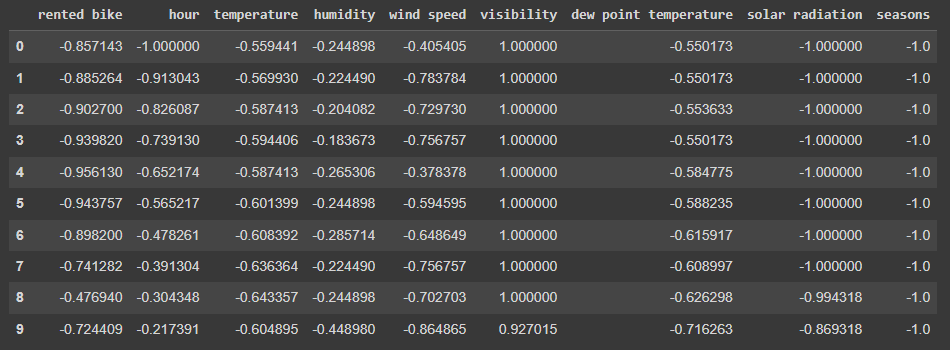
def normalizar(x,xmax,xmin,ymax,ymin):

m = (ymax-ymin)/(xmax-xmin)

b = ymin - m\*xmin

y = m\*x + b

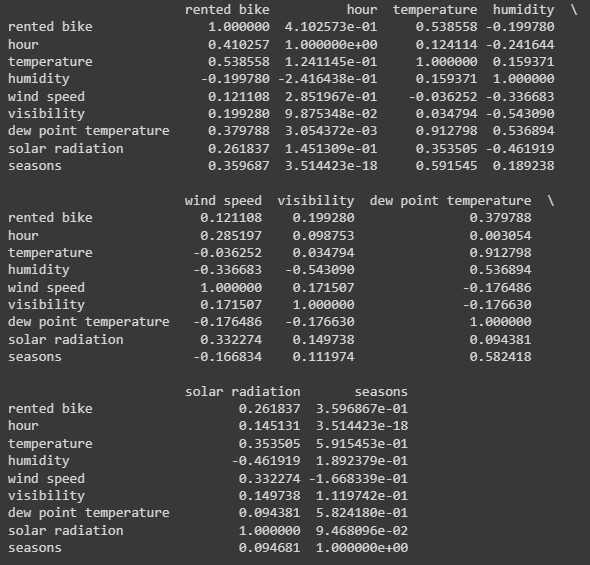
return y



*Ilustración 7 Datos normalizados*

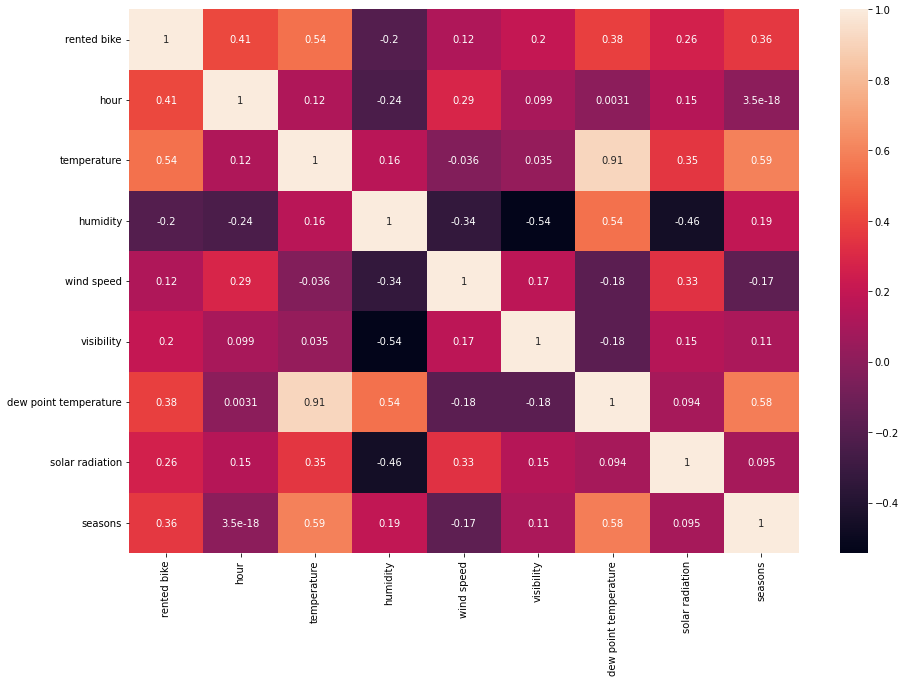
Ahora, se explica cómo se realizó el *análisis de correlación numérica* de los atributos en el conjunto de datos. Primero se utiliza la herramienta seaborn para graficar cada atributo en comparación con todos los demás y la salida, con el objetivo de analizar la correlación entre ellos. Luego se calculan los factores de correlación y se representan en un mapa de calor, donde se observa que la temperatura, la hora y la cantidad de bicicletas rentadas están directamente correlacionados. Además, se resalta que los atributos relacionados con la temperatura también tienen una alta correlación entre sí, lo cual no es ideal, ya que se busca que las entradas sean diferentes entre sí.

corr = data2.corr() [17]



*Ilustración 8 Correlación de los datos*

Con el objetivo de visualizar dicha correlación entre variables, se decide implementar un mapa de calor, de la siguiente manera:



*Ilustración 9 Mapa de calor: Correlación entre variables*

El siguiente paso en el proyecto es la creación del Modelo Neuronal y la selección de su arquitectura. Para esto, se comienza con la importación de las librerías necesarias, las cuales ya se han descrito en el marco teórico del documento. Adicionalmente, se utiliza la función "plot\_model" de Keras para convertir el modelo creado en un archivo de formato de punto.

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import plot\_model

En vista de que el objetivo de la red neuronal es predecir la cantidad de bicicletas alquiladas, se establece este valor como el objetivo o target de la situación. Para construir el conjunto de entrenamiento x\_train, se utiliza la información contenida en la base de datos data3 sin incluir el valor objetivo.

target = data2['rented bike'].values #y\_train data3 = data2.drop(['rented bike'], axis=1) print(target.shape)

print(data3.shape)

Este método permite dividir el conjunto de datos en grupos aleatorios de entrenamiento y prueba. Al especificar que los datos de validación serán el 30% del total de datos, se asegura que la red neuronal esté siendo entrenada con una cantidad suficiente de datos y que se puedan validar los resultados con un conjunto de datos independiente. [18]

from sklearn.model\_selection import train\_test\_split seed = 42

x\_train,x\_test,y\_train,y\_test = train\_test\_split(dat a3, target, test\_size=0.30, random\_state=seed)

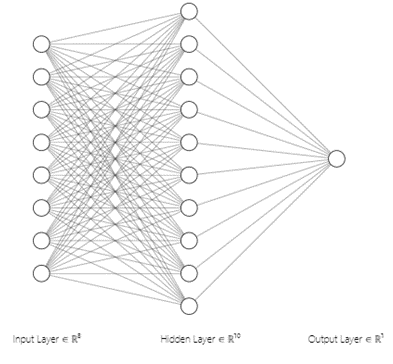
print(x\_train.shape) print(y\_train.shape) print(x\_test.shape) print(y\_test.shape)

Después de realizar la separación de datos anterior, se obtuvieron los siguientes conjuntos: x\_train con dimensiones (6132, 8), y\_train con dimensiones (6132,), x\_test con dimensiones (2628, 8) y y\_test con dimensiones (2628,). El conjunto de datos x\_test es utilizado para la evaluación y comparación de los modelos.

El equipo de trabajo decidió probar diferentes arquitecturas para el modelo de predicción. Se presentarán tres arquitecturas diferentes, variando tanto el número de neuronas en cada capa como el número de capas en la red. Se estableció un límite máximo de tres capas profundas. Además, se implementarán tres optimizadores diferentes para cada arquitectura: Adamax, Nadam y Adam.

input\_dim = x\_train.shape[1] num\_clases = 1

Considerando lo mencionado, la Primera Arquitectura consta de tres capas: una capa de entrada con 8 neuronas que representan los datos seleccionados del Dataset para el análisis, una capa oculta con 10 neuronas que utiliza la función de activación ReLu y una capa de salida con una sola neurona que proporciona información exclusivamente sobre el número de bicicletas alquiladas, cuya función de activación es lineal.



*Ilustración 10 Representación visual de la primera arquitectura.*

Uno de los objetivos del mini proyecto es incorporar al menos tres optimizadores para cada una de las tres arquitecturas propuestas. En primer lugar, se utilizará el optimizador "Adam" a través de una función denominada model\_Adam() de la siguiente manera:

def model\_Adam():

model = Sequential()

model.add(Dense(10, input\_dim = input\_dim, activation='relu'))

model.add(Dense(num\_clases, activation='linear'))

model.summary()

opt = tf.keras.optimizers.Adam(learning\_rate=0.001)

model.compile(loss = 'mse', optimizer = opt)

return model

REFERENCIAS

| [1] | NetApp, «¿Qué es la inteligencia artificial?,» [En línea]. Available: https://www.netapp.com/es/artificial-intelligence/what-is-artificial-intelligence/. |
| --- | --- |
| [2] | DataScientest, «Machine Learning: definición, funcionamiento, usos,» 13 Diciembre 2021. [En línea]. Available: https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos. |
| [3] | PyPI, «pandas 1.5.3,» [En línea]. Available: https://pypi.org/project/pandas/. |
| [4] | PyPI, «numpy 1.24.2,» [En línea]. Available: https://pypi.org/project/numpy/. |
| [5] | PyPI, «matplotlib 3.7.1,» [En línea]. Available: https://pypi.org/project/matplotlib/. |
| [6] | PyPI, «seaborn 0.12.2,» [En línea]. Available: https://pypi.org/project/seaborn/. |
| [7] | Google, «TensorFlow: Open-source machine learning,» 9 Noviembre 2015. [En línea]. Available: https://www.youtube.com/watch?v=oZikw5k\_2FM. |
| [8] | TensorFlow, «El modelo secuencial,» [En línea]. Available: https://www.tensorflow.org/guide/keras/sequential\_model?hl=es-419. |
| [9] | Jesús, «¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning?,» 7 Enero 2020. [En línea]. Available: https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/#:~:text=Entonces%2C%20lo%20que%20un%20optimizador,se%20conoce%20como%20%E2%80%9Cbackpropagation%E2%80%9D.. |
| [10] | Keras, «Optimizers,» [En línea]. Available: https://keras.io/api/optimizers/. |
| [11] | Keras, «Adam,» [En línea]. Available: https://keras.io/api/optimizers/adam/. |
| [12] | Keras, «Adamax,» [En línea]. Available: https://keras.io/api/optimizers/adamax/. |
| [13] | Keras, «Nadam,» [En línea]. Available: https://keras.io/api/optimizers/Nadam/. |
| [14] | UC Irvine Machine Learning, «Seoul Bike Sharing Demand Data Set,» [En línea]. Available: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand. |
| [15] | fisicalab, «Temperatura,» [En línea]. Available: https://www.fisicalab.com/apartado/temperatura. |
| [16] | Lebalap Academy, «Porcentaje de Humedad: Qué es y cómo afecta al F1,» [En línea]. Available: https://lebalap.academy/f1/porcentaje-de-humedad/#%3A~%3Atext%3DEl%20porcentaje%20de%20humedad%20del%2Caparecer%C3%ADan%20gotas%20de%20agua%20l%C3%ADquida. |
| [17] | pandas, «pandas.DataFrame.corr,» [En línea]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html. |
| [18] | Keras, [En línea]. Available: https://keras.io/api/optimizers/adamax/. |

1. NetApp, «¿Qué es la inteligencia artificial?,» [En línea]. Available: https://www.netapp.com/es/artificial-intelligence/what-is-artificial-intelligence/.
2. DataScientest, «Machine Learning: definición, funcionamiento, usos,» 13 Diciembre 2021. [En línea]. Available: https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos.
3. PyPI, «pandas 1.5.3,» [En línea]. Available: https://pypi.org/project/pandas/.
4. PyPI, «numpy 1.24.2,» [En línea]. Available: https://pypi.org/project/numpy/.
5. PyPI, «matplotlib 3.7.1,» [En línea]. Available: https://pypi.org/project/matplotlib/.
6. PyPI, «seaborn 0.12.2,» [En línea]. Available: https://pypi.org/project/seaborn/.
7. Google, «TensorFlow: Open-source machine learning,» 9 Noviembre 2015. [En línea]. Available: https://www.youtube.com/watch?v=oZikw5k\_2FM.
8. TensorFlow, «El modelo secuencial,» [En línea]. Available: https://www.tensorflow.org/guide/keras/sequential\_model?hl=es-419.
9. Jesús, «¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning?,» 7 Enero 2020. [En línea]. Available: https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/#:~:text=Entonces%2C%20lo%20que%20un%20optimizador,se%20conoce%20como%20%E2%80%9Cbackpropagation%E2%80%9D..
10. Keras, «Optimizers,» [En línea]. Available: https://keras.io/api/optimizers/.
11. Keras, «Adam,» [En línea]. Available: https://keras.io/api/optimizers/adam/.
12. Keras, «Adamax,» [En línea]. Available: https://keras.io/api/optimizers/adamax/.
13. Keras, «Nadam,» [En línea]. Available: https://keras.io/api/optimizers/Nadam/.
14. UC Irvine Machine Learning, «Seoul Bike Sharing Demand Data Set,» [En línea]. Available: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand.
15. fisicalab, «Temperatura,» [En línea]. Available: https://www.fisicalab.com/apartado/temperatura.
16. Lebalap Academy, «Porcentaje de Humedad: Qué es y cómo afecta al F1,» [En línea]. Available: https://lebalap.academy/f1/porcentaje-de-humedad/#%3A~%3Atext%3DEl%20porcentaje%20de%20humedad%20del%2Caparecer%C3%ADan%20gotas%20de%20agua%20l%C3%ADquida.
17. pandas, «pandas.DataFrame.corr,» [En línea]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html.
18. Keras, [En línea]. Available: https://keras.io/api/optimizers/adamax/.
19. Scikit Learn, «Sklearn.model\_selection.train\_test\_split» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html.