APLICACIÓN DE LA TÉCNICA DE APRENDIZAJE AUTOMÁTICO TRANSFORMER NEURAL NETWORKS EN LA IDENTIFICACIÓN DE PATRONES DE FLUJO BIFÁSICO DE ACEITE Y AGUA EN DUCTOS VERTICALES

ERWING EDUARDO PERILLA PLATA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE INGENIERÍA MECÁNICA

INGENIERÍA MECÁNICA

BUCARAMANGA

2023

APLICACIÓN DE LA TÉCNICA DE APRENDIZAJE AUTOMÁTICO TRANSFORMER NEURAL NETWORKS EN LA IDENTIFICACIÓN DE PATRONES DE FLUJO BIFÁSICO DE ACEITE Y AGUA EN DUCTOS VERTICALES

ERWING EDUARDO PERILLA PLATA

Proyecto de grado para optar por el título de Ingeniero Mecánico

DIRECTOR

OCTAVIO ANDRÉS GONZÁLEZ ESTRADA

PhD en Ingeniería Mecánica y de Materiales

CODIRECTOR

CARLOS MAURICIO RUIZ DIAZ

Magister en Ingeniería Mecánica

MARLON MAURICIO HERNÁNDEZ CELY

PhD en Ingeniería Mecánica

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE INGENIERÍA MECÁNICA

INGENIERÍA MECÁNICA

BUCARAMANGA

2023

DEDICATORIA

Dedico este trabajo a mis padres, Leonel Perilla y Aylen Plata, por su apoyo, impulso y comprensión a lo largo de mi formación como Ingeniero Mecánico.

También dedico este trabajo a mis hermanos Andrey Perilla y Leonardo Perilla, por ser un gran ejemplo para mí y fuente de motivación en mi formación como profesional.

Por último, dedico este trabajo a mis seres queridos peludos de 4 patas, mis gatos Oliver y Tom, quienes por 15 años alegraron mi vida y ahora descansan en paz.

AGRADECIMIENTOS

Agradezco primeramente a Dios por guiarme con su luz a lo largo de mi vida, por bendecir cada instante de mi existir y principalmente, por brindarme la oportunidad de ser un profesional.

Agradezco al profesor Octavio Andrés González Estrada, director de mi proyecto de grado, por brindarme su orientación y su apoyo a lo largo de la realización de este proyecto, por sus enseñanzas y conocimiento como profesional, por su paciencia y por haberme brindado esta gran oportunidad.

Agradezco a Carlos Mauricio Ruiz Diaz, codirector de mi proyecto de grado, por brindarme las bases para la realización de este proyecto, por su guía y conocimiento en el tema de redes neuronales, por su apoyo y disposición a querer mejorar este proyecto.

Agradezco al profesor William Pinto Hernández, por ser un gran docente y modelo que seguir, por su apoyo a lo largo de mi formación como profesional, por sus consejos, por su sabiduría, por su ayuda en momentos de dificultad y por ser alguien en quien confiar.

Agradezco a Alejandro Carreño, por presentarme este proyecto de grado que desafiaba mis habilidades y me haría emprender un reto personal al aprender acerca de programación y redes neuronales.

Agradezco a mis amigos del alma Juan Pablo, Nicolas, Daniel y Juan Diego, por su amistad incondicional, por su apoyo a lo largo de mi formación como profesional y gracias a todos mis demás amigos por hacer de la universidad una gran experiencia.

Asimismo, agradezco a todos aquellos que directa o indirectamente aportaron en mi formación como profesional.

CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE ANEXOS

Pág.

GLOSARIO

D	Diámetro de la tubería
A	Área de sección transversal de la tubería
A_w	Área de sección transversal correspondiente al agua
A_o	Área de sección transversal correspondiente al aceite
Q_w	Caudal de inyección del agua
Q_o	Caudal de inyección del aceite
C_w	Fracción volumétrica correspondiente al agua
C_o	Fracción volumétrica correspondiente al aceite
J_w	Velocidad superficial correspondiente al agua
J_o	Velocidad superficial correspondiente al aceite
J	Velocidad del flujo bifásico
ε_w	Holdup del agua
ε_o	Holdup del aceite
V_w	Velocidad in situ del agua
V_o	Velocidad in situ del aceite
TNN	Transformer Neural Network
IA	Inteligencia Artificial
VP	Verdadero Positivo
VN	Verdadero Negativo
FP	Falso Positivo
FN	Falso Negativo

w_{ji}	Pesos de atención
Q	Vector Queries
K	Vector Keys
V	Vector Values
d_k	Dimensión del modelo
μ_o	Viscosidad del aceite
ρ_o	Densidad del aceite
ω_t	Pesos en el tiempo t
ω_{t+1}	Pesos en el tiempo t+1
m_t	Agregado de gradientes en el tiempo t
m_{t-1}	Agregado de gradientes en tiempo t-1
α	Tasa de aprendizaje en el tiempo t
δL	Derivada de la función de pérdida
$\delta \omega_t$	Derivada de pesos en el tiempo t
β	Parámetro de media móvil
$\beta_1 \; y \; \beta_2$	Tasas de decaimiento del promedio de gradientes
v_t	Suma de cuadrados de gradientes pasados
$\mathcal{L}_{BCE}$	Función de pérdida Binary Cross Entropy
y_i	Clase para predecir
$\widehat{y_i}$	Probabilidad predicha para la clase
c	Número de clases
n	Número de ejemplos

RESUMEN

Título: Aplicación de la técnica de aprendizaje automático transformer neural networks en la identificación de patrones de flujo bifásico de aceite y agua en ductos verticales *

Autor: Erwing Eduardo Perilla Plata **

Palabras clave: flujo bifásico, inteligencia artificial, Transformer Neural Network

Descripción: Con el alto costo en el transporte de hidrocarburos, la corrosión en tuberías y la necesidad de conocer el comportamiento de los flujos bifásicos, el crecimiento de la industria 4.0 en la industria oil & gas ha permitido la introducción de nuevas tecnologías para la solución de estas problemáticas. Dichas soluciones consisten en la introducción de modelos predictivos usando redes neuronales. Para este proyecto, a partir de diversos autores se estructuro una base de 4864 datos con información de las propiedades de flujo bifásico aceite-agua en tubería vertical. Posteriormente, se desarrolló una estructura de transformer neural network (TNN), la cual consta únicamente con la estructura del Codificador del arreglo original. A su vez, se propusieron diversas configuraciones (dentro de los parámetros que se modificaron se encuentra el número de cabezas atencionales, la función de activación, el dropout y la tasa de aprendizaje) para el modelo de TNN y así seleccionar aquel con mejores resultados. Una vez entrenada la red, se realizan predicciones con un set de datos con el que el modelo no haya interactuado y así realizar mapas de flujo con los patrones predichos por el modelo. El modelo de TNN desarrollado es capaz de predecir 9 de 10 patrones de flujo implementados en la base de datos, con una precisión máxima del 53,07%. Asimismo, de los diferentes patrones de flujo predichos se presenta una precisión promedio del 63,21% y una exactitud promedio del 86,51%.

* Trabajo de grado

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería Mecánica. Pregrado en Ingeniería Mecánica. Director: Octavio Andrés González Estrada, PhD en Ingeniería Mecánica y de Materiales. Codirectores: Carlos Mauricio Ruiz Diaz, Magister en Ingeniería Mecánica, Marlon Mauricio Hernández Cely, PhD en Ingeniería Mecánica.

ABSTRACT

Title: Application of the machine learning technique transformer neural networks in the identification of biphasic flow patterns of oil and water in vertical pipes *

Author: Erwing Eduardo Perilla Plata **

Keywords: artificial intelligence, biphasic flow, Transformer Neural Network

Description: With the high cost of transporting hydrocarbons, corrosion in pipes and the need to know the behavior of biphasic flows, the growth of industry 4.0 in the oil & gas industry has allowed the introduction of new technologies to solve these problematic. These solutions consist of the introduction of predictive models using neural networks. For this project, from various authors, a database of 4864 data was structured with information on the properties of two-phase oil-water flow in vertical pipes. Subsequently, a transformer neural network (TNN) structure was developed, which consists only of the Encoder structure of the original arrangement. In turn, various configurations were proposed (among the parameters that were modified are the number of attentional heads, the activation function, the dropout and the learning rate) for the TNN model and thus select the one with the best results. Once the network is trained, predictions are made with a data set with which the model has not interacted and thus make flow maps with the patterns predicted by the model. The developed TNN model can predict 9 out of 10 flow patterns implemented in the database, with a maximum accuracy of 53.07%. Likewise, of the different predicted flow patterns, an average precision of 63.21% and an average accuracy of 86.51% are presented.

* Degree Work

** Faculty of Physicomechanical Engineering. School of Mechanical Engineering. Undergraduate in Mechanical Engineering. Director: Octavio Andrés González Estrada, PhD in Mechanical and Materials Engineering. Co-directors: Carlos Mauricio Ruiz Diaz, Master's in mechanical engineering, Marlon Mauricio Hernández Cely, PhD PhD in Mechanical Engineering.

INTRODUCCIÓN

El transporte de flujos multifásicos es común en la industria actual, donde dichos flujos están compuestos por dos o más fases y/o diferentes estados físicos (líquido, sólido o gaseoso), este tipo de flujos se encuentran presentes en varios sectores industriales (como el agrícola, el alimentario, el farmacéutico, el petrolero, etc.). Para resaltar el alcance de los flujos multifásicos, estos se clasifican según las fases involucradas, siendo el flujo trifásico compuesto por gas, petróleo y sustancias sólidas. La subcategoría de flujos multifásicos de mayor estudio son los flujos bifásicos, los cuales pueden configurarse como gas-líquido, gas-sólido, líquido-sólido y líquido-líquido [1], [2]. La subcategoría de flujos bifásicos líquido-líquido, se encuentra presente dentro de la industria oil & gas en la producción de crudo a partir de combinaciones de agua/petróleo en tubería, para su posterior transporte a instalaciones de procesamiento o refinería [3], [4], [5].

Un factor importante de los flujos multifásicos es su configuración geométrica interna o patrón de flujo, generado a partir de la interacción de las sustancias, que a su vez depende de los caudales de las sustancias, el diámetro de la tubería y las propiedades de transporte de cada flujo [1]. Una variable importante para determinar el tipo de patrón de flujo presente en tubería es la inclinación de esta, dado que los patrones de flujo presentes en tubería horizontal son diferentes a los encontrados en tubería vertical. El estudio del comportamiento de patrones de flujo bifásico en tubería es crucial en cuestiones como la reducción de costos de bombeo para la extracción de crudos pesados (con viscosidades superiores a 10000 cP), dado que estos conforman un tercio de las reservas mundiales de hidrocarburos [6]. A su vez, un problema en la industria del petróleo es la corrosión interna y externa en tuberías, dado que un estimado de las fallas presentes en tubería

que transporta petróleo, son ocasionadas en un 80% por corrosión interna, lo cual conlleva un elevado costo su mantenimiento, siendo necesario conocer el patrón de flujo presente en la tubería para minimizar estas fallas [7].

Por otra parte, en [8] se estudian modelos predictivos para el cálculo de la fracción volumétrica de un flujo bifásico agua-aceite en la horizontal utilizando una red neuronal artificial mostrando como resultados un error absoluto medio porcentual (AAPE) de 3,01% y un coeficiente de determinación de 0. Asimismo, en [9] se menciona que con la ayuda de predicciones del flujo bifásico a partir de CFD (Computational Fluid Dynamics) implementando el software STAR-CCM+, es posible tener un control preciso del riesgo de corrosión en oleoductos. Adicionalmente, gracias al avance generado por la cuarta revolución industrial (industria 4.0), se han implementado beneficios en diversos campos con la ayuda de la inteligencia artificial (IA), la cual posee el potencial para procesar grandes cantidades de datos [10]. El rango de aplicación de dicha técnica de aprendizaje se extiende a diversos sectores industriales, como es el caso del sector oil & gas, por ello, se requieren métodos innovadores para reemplazar los métodos antiguos de monitoreo en el transporte de hidrocarburos, en el cual se registra la presencia de flujos multifásicos en distintas combinaciones de líquidos, gases y sólidos. De esta manera, la simulación de procesos que incluyen flujos multifásicos ha permitido generar soluciones a diversos problemas de ingeniería, a partir del modelamiento de los mismos, lo cual se extiende con la amplia gama de aplicaciones que garantizan un adecuado flujo de hidrocarburos al interior de tuberías [11]–[14].

El amplio dominio del aprendizaje automático ha generado diversas alternativas para lograr la solución de problemas complejos asociados al reconocimiento y clasificación de datos [14]. Dentro

de las técnicas implementadas se encuentra Transformer Neural Network (TNN), una red neuronal creada originalmente para la traducción de largas cadenas de texto [15] y a su vez muestra un gran rendimiento en modelos multilingües [16]. Asimismo, con la estructura BERT que se basa en el TNN original, se ha implementado en diversidad de aplicaciones, como lo es el análisis de texto para clasificación de texto [17] o de emociones [18]. La detección y clasificación de imágenes que son tareas recurrentes de las redes neuronales convolucionales, fueron adaptadas a TNN, en un modelo denominado VisualTransformer [19]. Por otra parte, en el estudio del ADN se han desarrollado modelos basados en TNN para la identificación de secuencias del genoma [20], [21], y a su vez para la predicción del perfil de secuencia de proteínas [22]. Además, esta técnica se ha implementado en ingeniería para la predicción de energía eólica [23], [24]. Adicionalmente, uno de los puntos fundamentales en el uso de redes neuronales, es la disposición del 80% de la base de datos para el entrenamiento del modelo, el 10% para prueba y 10% para la validación [25].

Dada la creciente implementación de IA en la industria petroquímica para controlar los procesos concernientes a fluidos multifásicos, en el presente trabajo se desarrolla un modelo inteligente basado en TNN, capaz de predecir los patrones de flujo al interior de tuberías verticales con un flujo bifásico liquido-liquido de aceite y agua. Primero, se selecciona el tipo de Transformer más conveniente para la aplicación deseada, seguidamente se procede a realizar una revisión bibliográfica concerniente a flujo bifásico agua/aceite de diversos autores. A continuación, se filtra y divide la base de datos asegurándose que el conjunto de datos implementado en el entrenamiento contenga los valores mínimos y máximos de cada parámetro implementado en las entradas del modelo. Finalmente, se implementan diferentes configuraciones de Transformer, para luego de acuerdo con el error y precisión de cada modelo entrenado seleccionar el más adecuado.

Flujo bifásico de aceite y agua

El flujo bifásico es un compuesto conformado por dos fases que fluyen simultáneamente a través de una tubería, siendo recurrentes en los sistemas industriales y a su vez en la naturaleza. Así mismo, este tipo de flujo se encuentra presente en la industria petrolera y al momento de este ser transportado por las diferentes fases de producción puede ir acompañado de otros fluidos como lo es el agua. A su vez, en este flujo se evidencia la formación de configuraciones geométricas en la tubería denominadas patrones de flujo [7]. Los flujos bifásicos logran ser caracterizados a partir de las propiedades que los describen, tales que, considerando un flujo constante se pueden considerar las fracciones volumétricas, las velocidades in situ de los fluidos y las velocidades superficiales de inyección.

Es necesario aclarar que en el presente trabajo se utiliza una nomenclatura específica para referirse a las propiedades relacionadas al agua con un subíndice *w* y con una *o* para las relacionadas al aceite.

Fracciones volumétricas

En el flujo bifásico, la fracción volumétrica *C* es la proporción correspondiente a cada una de las fases que componen el flujo, para su cálculo es necesario conocer los caudales de inyección Q_w para el agua y Q_o para el aceite, así:

$$C_w = \frac{Q_w}{Q_w + Q_o}, \qquad C_o = \frac{Q_o}{Q_w + Q_o} \tag{1}$$

Velocidades superficiales

Las velocidades superficiales correspondientes a la fase del agua J_w y aceite J_o, son obtenidas mediante la división entre el caudal de inyección de cada una y el área de la sección transversal de la tubería:

$$J_w = \frac{Q_w}{A}, \qquad J_o = \frac{Q_o}{A} \tag{2}$$

A su vez, mediante la suma de ambas velocidades es posible conocer la velocidad de la mezcla J:

$$J = J_w + J_o \tag{3}$$

Holdup o fracción volumétrica in situ

En el flujo bifásico que pasa por una tubería, cada una de las fases ocupa una fracción de la sección transversal de área A, es decir, A_w corresponde al área ocupada por el agua y A_o por el aceite, con ello se puede determinar el holdup de cada fase en el interior de la tubería.

$$\varepsilon_w = \frac{A_w}{A}, \qquad \varepsilon_o = \frac{A_o}{A} \tag{4}$$

Velocidades in situ

A diferencia de las velocidades superficiales (J_w *y* J_o), las velocidades in situ ocurren debido al deslizamiento entre las fases de la mezcla bifásica, sus cálculos se realizan a partir del caudal que fluye de cada fase a través del área que ocupa esta misma, así:

$$V_w = \frac{Q_w}{A_w}, \qquad V_o = \frac{Q_o}{A_o} \tag{5}$$

1. OBJETIVOS

1.1. OBJETIVO GENERAL

Diseñar un modelo predictivo basado en inteligencia artificial capaz de identificar patrones de flujo bifásico en ductos verticales implementando la técnica de aprendizaje automático transformer neural networks.

1.2. OBJETIVOS ESPECÍFICOS

- Estructurar una base de datos con información de velocidades superficiales, viscosidades y fracciones volumétricas de un flujo bifásico de aceite y agua para ductos verticales de distintos diámetros a partir de un análisis bibliográfico, con el fin de formar el vector de entradas del modelo inteligente.

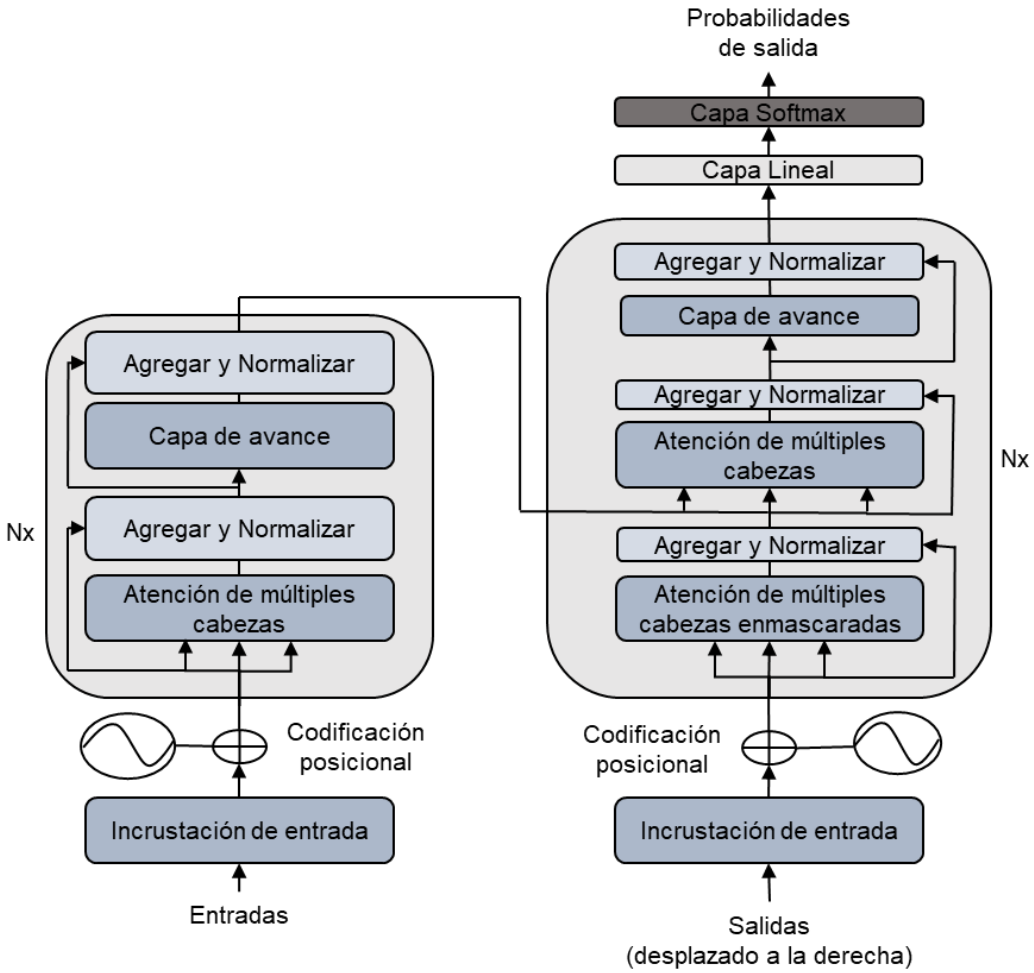
- Implementar las expresiones teóricas desarrolladas para el modelamiento matemático de flujos bifásicos en ductos verticales para determinar las zonas de transición de patrones de flujo y representarlas en gráficas 2D, utilizando el software PYTHON.

- Desarrollar una estructura de transformer neural networks que procese de manera optimizada la información contenida en la base de datos construida, con el fin de obtener un modelo inteligente capaz de predecir patrones de flujo bifásico de aceite y agua en ductos verticales, utilizando el software PYTHON.

2. MATERIALES Y MÉTODOS

La arquitectura de la TNN original se basa en el codificador-decodificador, como se aprecia en la Figura 1. Dicha arquitectura que se usa ampliamente para tareas como la traducción automática, donde una secuencia de palabras se traduce de un idioma a otro. Esta arquitectura consta de dos componentes, el *Encoder*, que convierte una secuencia de tokens de entrada en una secuencia de vectores de incrustación y el *Decoder*, que utiliza el estado oculto del codificador para generar iterativamente una secuencia de salida de fichas, una ficha a la vez.

Figura 1. Arquitectura Encoder-Decoder del Transformer.



Fuente. Modificado de [15].

2.1. TIPOS DE TRANSFORMER

La arquitectura del transformer fue diseñada para tareas de secuencia a secuencia como la traducción automática, pronto se adaptaron el Encoder y el Decoder de forma independiente, por lo cual existen tres tipos diferentes de transformer [26]:

- *Encoder-only*: Este modelo es capaz de convertir una secuencia de entrada de texto en una representación numérica adecuada para tareas como la clasificación de texto o el reconocimiento de entidades nombradas. La representación calculada para un token dado en esta arquitectura depende de la atención bidireccional, es decir, depende tanto del contexto izquierdo como del derecho del token.

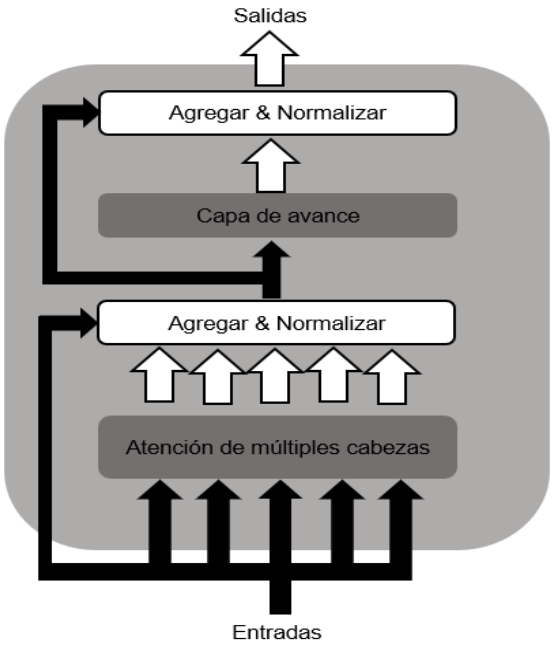
- *Decoder-only*: Este tipo de modelos es capaz de completar automáticamente una secuencia mediante la predicción iterativa de la siguiente palabra más probable. En este caso, el modelo se basa en la atención causal o autorregresiva, es decir, la representación calculada para un token en esta arquitectura depende únicamente del contexto izquierdo.

- *Encoder-Decoder*: Estos se utilizan para modelar asignaciones complejas de una secuencia de texto a otro, es decir, son adecuados para tareas de traducción automática y resumen.

Para el caso en particular de este trabajo se hará uso del modelo de *Encoder-only*, por lo cual, a continuación, se explicará a detalle la estructura del Encoder o codificador en español. La estructura del bloque del codificador se evidencia en la Figura 2, por este entra una secuencia de

embeddings y es alimentada a través de las siguientes capas: Una capa de autoatención de varios cabezales y una capa de avance.

Figura 2. Anatomía del Encoder.



Fuente. Elaboración propia.

La autoatención es un mecanismo que permite a las redes neuronales asignar a cada elemento de una secuencia una cantidad diferente de atención, es decir, un valor de pesos diferente a cada ítem. Por lo tanto, la idea principal de la autoatención es usar una secuencia completa para el cálculo de un promedio ponderado de cada incrustación. Una manera de formular esto es que dada una secuencia de tokens X_1, ..., X_n, la autoatención produce una secuencia de nuevas incrustaciones X_1', ..., X_n' donde cada X_i' es una combinación lineal de todos los X_j, como se presenta en la ecuación (1):

$$X'_i = \sum_{j=1}^{n} w_{ji} x_j \qquad (1)$$

Los coeficientes w_{ji} son los pesos de atención y están normalizados de tal manera que $\Sigma_j w_{ji} = 1$.

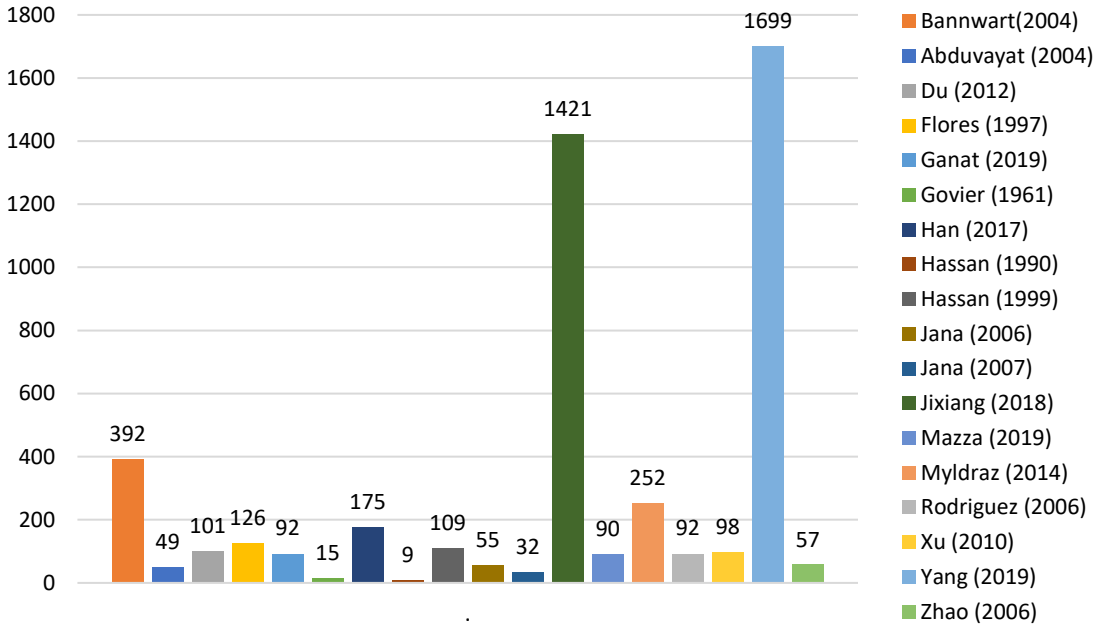
El proceso de autoatención consiste en 3 vectores Q (*queries*), K (*keys*) y V (*values*) que se originan de multiplicar cada entrada por matrices W(entrenadas durante el entrenamiento). Se realiza un producto escalar entre el vector Q y el vector K, luego dicho resultado es dividido entre la raíz cuadrada del tamaño de dimensión de los vectores de entrada originales, posteriormente será normalizado con la operación softmax y multiplicado con el vector V. Para facilidades de cómputo la información es tratada en forma matricial. En la ecuación (2) se representa el proceso de atención.

$$Atención(Q, K, V) = Z = softmax\left(\frac{Q * K^T}{\sqrt{d_k}}\right) * V \qquad (2)$$

2.2. CONJUNTO DE DATOS

A partir de la literatura referente a flujo bifásico compuesto por agua y aceite se estructuró una base de datos compuesta por 4864 datos para tubería vertical provenientes de 18 autores diferentes presentados en la Figura 3.

Figura 3. Cantidad de datos obtenidos por autor de la literatura.



Fuente. Elaboración propia

En la Tabla 1, se evidencia a detalle la información correspondiente a los datos de cada autor, donde se muestra el diámetro de la tubería que se implementó en los experimentos, la viscosidad y densidad del aceite usado, el número de patrones de flujo encontrados y la cantidad de datos extraídos. Previamente a la introducción de los datos en las diferentes fases del modelo, es necesario realizar una selección y tratamiento de los datos como se indica en el Anexo A.

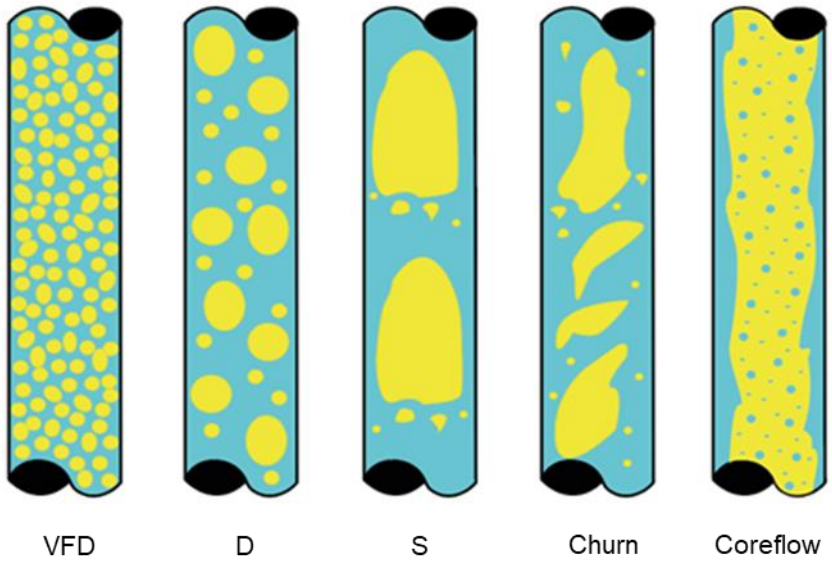
Tabla 1. Información general de la base de datos para flujo bifásico compuesto por agua y aceite en tubería vertical

Referencia	D[m]	μ_o[Pa.s]	ρ_o[kg/m^3]	N°. de patrones de flujo	N°. de datos
[27]	0,0284	0,488	925	3	392
[28]	0,1064	0,00188	800	6	49
[29]	0,02	0,011	856	3	101
[30]	0,05	0,02	850	5	126
[31]	0,04	0,035	860	4	92
[32]	0,0263	0,0201	851	2	15
[33]	0,02	0,125	801	4	175
[34]	0,127	0,001	801	2	9
[35]	0,027	0,02	801	4	109
[36]	0,0254	0,001	792	4	55
[37]	0,0254	0,001	792	5	32
[38]	0,01	0,287	1823	3	1421
[39]	0,026	0,0011	793	5	90
[40]	0,03	0,03	856	4	252
[41]	0,0284	0,5	930	1	92
[42]	0,03	0,044	860	4	98
[43]	0,02	0,09732	857	7	1699
[44]	0,04	0,0041	824	1	57

Fuente. Elaboración propia

Tomando como referencia los esquemas desarrollados por cada científico se lograron identificar 9 patrones de flujo diferentes y 1 intermedio de transición(TF), a continuación, se enuncian con sus respectivas cantidades: VFD o/w (265), VFD w/o (204), D o/w (947), D w/o (1292), S o/w (459), S w/o (656), Churn o/w (21), Churn w/o (126), Core flow (480), TF (314). En la Figura 4 se realiza una aproximación gráfica del comportamiento de los patrones de flujo más destacados.

Figura 4. Patrones de flujo bifásico en tubería vertical.



Fuente. Elaboración propia

Como se mencionó anteriormente los patrones más comunes para tubería a 90° son los siguientes:

-*Gotas dispersas (VFD)*: Este tipo de configuración se presenta cuando el caudal de una fase es menor al correspondiente a la otra fase, donde el caudal más bajo se manifiesta en forma de pequeñas gotas finas y son arrastradas por el gran caudal perteneciente a la otra fase. En este tipo de flujo, el perfil de velocidades sigue la tendencia del flujo monofásico de agua en tubería, es decir, mayor velocidad en el centro del tubo y menor conforme se aproxima a las paredes. Se reconoce por las siglas VFD, cuando la dispersión es de manera homogénea, completamente desarrollada y fina.

-*Gotas (D*): A diferencia del patrón de gotas dispersas, estas gotas son de mayor tamaño y con formas de esfera, tapón elíptico y tapón esférico.

-*Intermitente (S / Churn)*: Al aumentar el caudal menor, las gotas presentes en el flujo de gotas(D) tienden a unirse en gotas alargadas de mayor tamaño y separadas por espacios grandes pertenecientes a la otra fase del flujo. De esta manera, si el tamaño se aproxima al diámetro de la tubería y presenta un comportamiento de pistón mecánico, se denomina slug (S). Así mismo, si se presentan gotas dispersas o existe coalescencia entre los pistones de líquido consecutivos, se puede denominar churn, a su vez, esto puede incurrir en la generación de un comportamiento turbulento del flujo.

-*Anular (Core flow)*: Este patrón de flujo se configura cuando el aceite fluye en el núcleo del tubo, mientras el agua es una película que rodea el flujo central. Se manifiesta cuando el caudal de agua es mayor, a su vez, la interfase presenta un arreglo ondulatorio que depende de las velocidades de inyección de las fases. Por otra parte, en la película de agua se pueden manifestar gotas dispersas de aceite y el núcleo de aceite puede presentar distorsiones de alta frecuencia, a este flujo se le denomina core flow.

En general, cuando se evidencia la presencia de variación de un patrón de flujo a otro, se puede definir como un flujo intermedio de transición y se representa por las siglas TF. Por otra parte, la base de datos estudiada posee información relacionada a las velocidades superficiales de los fluidos involucrados y su mezcla, fracción volumétrica del agua y del aceite, diámetro de la tubería implementada, viscosidad del aceite y nombre de los patrones de flujo desarrollados.

2.3. MODELO TRANSFORMER NEURAL NETWORK

La estructuración del modelo implementado se constituye a partir del codificador implementado en TNN y una capa softmax a la salida del modelo. El vector de entradas X se conforma por las velocidades superficiales de los fluidos (aceite y agua), la suma de estas, la fracción volumétrica in situ (holdup) de ambos fluidos, la viscosidad del aceite y el diámetro de la tubería de sección transversal circular, como se presenta en la ecuación (3).

$$X = \begin{Bmatrix} J_o \\ J_w \\ J_{o+w} \\ \varepsilon_o \\ \varepsilon_w \\ D \\ \mu_o \end{Bmatrix} \tag{3}$$

El optimizador del entrenamiento seleccionado es Adam, debido a que este integra las características de dos metodologías, una es la del algoritmo de descenso de gradiente con impulso y la otra es la del algoritmo RMSP (Root Mean Square Propagation). El descenso de gradiente con impulso se utiliza para acelerar el algoritmo de descenso del gradiente tomando en cuenta el promedio ponderado exponencial de los gradientes, es decir, que al usar promedios hace que el algoritmo converja hacia los mínimos a un ritmo más rápido. Este método se define en las ecuaciones (4) y (5).

$$\omega_{t+1} = \omega_t - \alpha m_t \tag{4}$$

Donde,

$$m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta \omega_t}\right] \tag{5}$$

Siendo m_t el agregado de gradientes en el tiempo t actual. Inicialmente, m_t = 0. Además, m_{t-1} es el agregado de gradientes en el tiempo t-1, ω_t son los pesos en el tiempo t, ω_{t+1} son los pesos en el tiempo t+1, α es la tasa de aprendizaje en el tiempo t. A su vez, δL es la derivada de la función de pérdida, $\delta\omega_t$ es la derivada de pesos en el tiempo t, β = 0.9 es el parámetro de media móvil.

La propagación cuadrática media (RMSP), en lugar de tomar la suma acumulada de gradientes cuadrados toma el promedio móvil exponencial. Se define de la siguiente manera en las ecuación (6).

$$\omega_{t+1} = \omega_t - \frac{\alpha}{(v_t + \mathcal{E})^{\frac{1}{2}}}\left[\frac{\delta L}{\delta \omega_t}\right] \tag{6}$$

Donde,

$$v_t = \beta v_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta \omega_t}\right]^2 \tag{7}$$

Siendo v_t la suma de cuadrados de gradientes pasados, es decir, sum(∂L/$\partial\omega_{t-1}$). Inicialmente, v_t = 0. $\mathcal{E} = 10^{-8}$ es una pequeña constante positiva para no dividir por cero cuando $v_t \longrightarrow 0$, α = 0,001 es el parámetro de tamaño de paso (tasa de aprendizaje).

Por lo tanto, con Adam se controla la velocidad del descenso del gradiente, haciendo posible que al alcanzar el mínimo global de pasos mínimos y de pasos lo suficientemente grandes para superar los mínimos locales en el recorrido, logrando así la eficiencia. En este caso, se utiliza la ecuación (8)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta \omega_t}\right];\quad v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta \omega_t}\right]^2 \tag{8}$$

Donde, β_1=0,9 y β_2=0,999 son las tasas de decaimiento del promedio de gradientes en los dos métodos anteriores.

Dado que m_t y v_t se han iniciado como 0 de acuerdo con los métodos anteriores, se observa que tienden a estar sesgados hacia 0 debido a que β_1 y $\beta_2 \approx 1$. Adam soluciona este problema al computar con corrección de sesgo m_t y v_t . Esto a su vez, se realiza para tener un control sobre los pesos mientras se alcanza el mínimo global y así evitar oscilaciones elevadas cuando se está cerca de él, es decir, se está adaptando al descenso del gradiente con cada iteración. Las ecuaciones utilizadas son (9) y (10):

$$\widehat{m_t} = \frac{m_t}{1-{\beta_1}^t};\ \widehat{v_t} = \frac{v_t}{1-{\beta_2}^t} \tag{9}$$

Por lo tanto:

$$\omega_{t+1} = \omega_t - \widehat{m_t}\left(\frac{\alpha}{\sqrt{\widehat{v_t}} + \varepsilon}\right) \tag{10}$$

La función de pérdida implementada en el aprendizaje del modelo desarrollado es la función *Binary Crossentropy*, normalmente es implementada en problemas binarios de clasificación, pero también puede ser usada en problemas donde las variables a predecir toman valores entre 0 y 1. Esta función se define en la ecuación (11).

$$\mathcal{L}_{BCE} = -\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{c}[y_i * \log \hat{y}_i + (1 - y_i) * \log(1 - \hat{y}_i)] \quad (11)$$

Donde, y_i es la clase para predecir, $\hat{y}_i$ es la probabilidad predicha para la clase, c es el número de clases y n es el número de ejemplos.

Se realizaron variaciones con 4 funciones de activación diferentes, dentro de las cuales tenemos la función ReLu, la cual transforma los valores de entrada anulando los valores negativos y dejando los positivos intactos, esta es definida por la ecuación (12).

$$ReLu(x) = \max(0, x) = \begin{cases} 0 \text{ para } x < 0 \\ x \text{ para } x \geq 0 \end{cases} \quad (12)$$

La segunda función de activación es sigmoide, que es usada en modelos donde es necesario predecir la probabilidad como resultado, dado que toma un rango de valores entre 0 y 1. Su definición se aprecia en la ecuación (13).

$$Sigmoide(x) = \frac{1}{1 + e^{-x}} \quad (13)$$

También, se implementó la función tangente hiperbólica, la cual es similar a la continuidad de la función sigmoide, pero con salidas en un rango de -1 a 1, y se define en la ecuación (14):

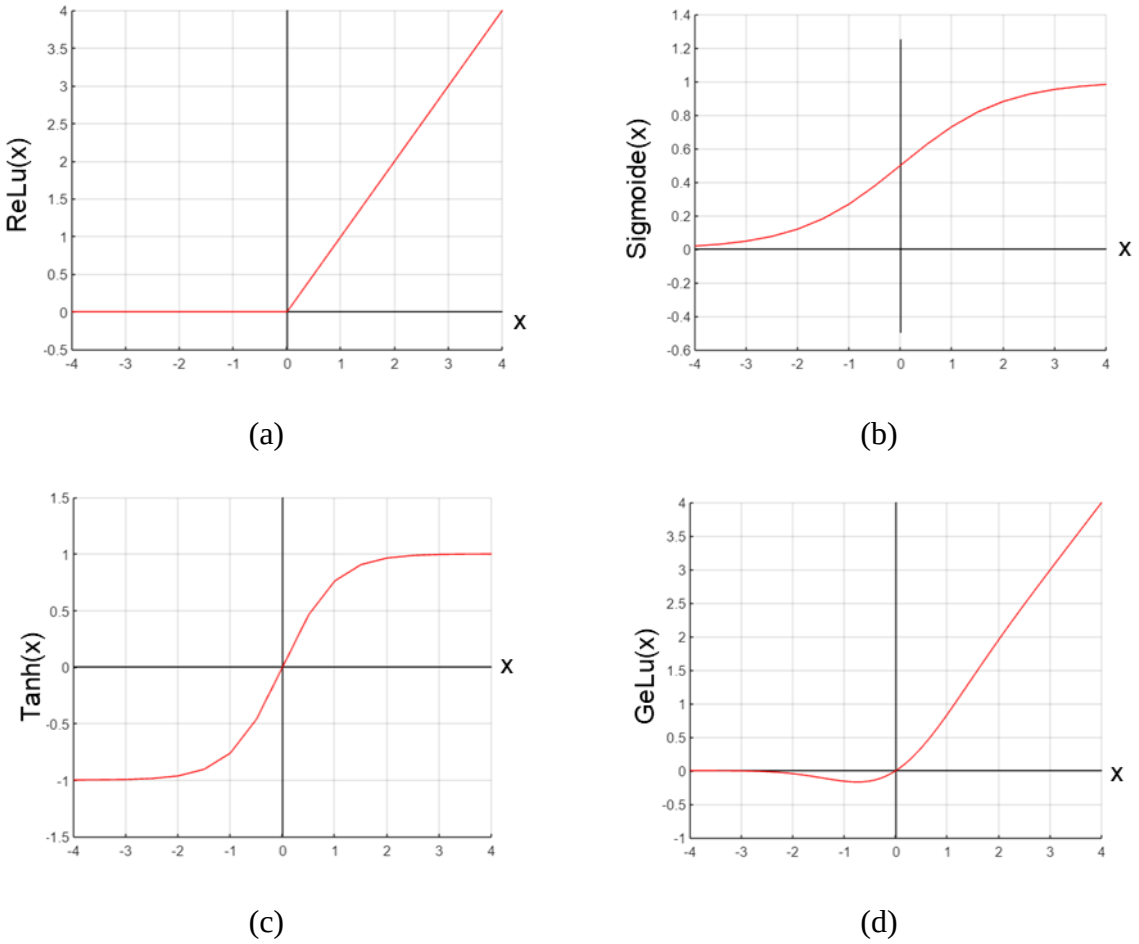
$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (14)$$

Por último, se usó la función de activación GeLu en la ecuación (15), la cual añade la no linealidad al multiplicar de firma estocástica por 0 y 1, y se asemeja a la función ReLu.

$$GeLu(x) = x * \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \quad (15)$$

En la Figura 5 se representan las funciones de activación implementadas.

Figura 5. Funciones de activación: (a) ReLu, (b) Sigmoide, (c) Tanh, (d) GeLu.



Fuente. Elaboración propia.

En el Anexo A, es posible evidenciar como se implementaron todas estas funciones en el modelo de red neuronal y a su vez, precisar el paso a paso para la elaboración de la TNN. Se estudio la variación de parámetros del modelo TNN para encontrar la configuración con el mejor desempeño. En primer lugar, se varió la tasa de aprendizaje (*learning rate*) con los siguientes valores: 0.005, 0.001 y 0.0005. Se tuvieron en cuenta los valores de 0.1, 0.3 y 0.5 para la capa correspondiente al dropout, se tomaron cantidades de 1, 2 y 4 para los cabezales de atención en la capa multihead-attention, y se implementó un único codificador. La dimensión de embedding es de 32 y la dimensión de la capa de avance es de 32. Finalmente, para cada simulación en particular se

tuvieron en cuenta 55 épocas durante el proceso de entrenamiento de la red neuronal, y un batch size de 2800. Para comparar el desempeño de las diferentes configuraciones se evalúa la precisión como el número de aciertos sobre el total de datos correspondientes a cada fase.

A su vez, se realizó una matriz de confusión a los modelos que fueron seleccionados por su desempeño, de la cual se extrajo la precisión y exactitud para cada uno de los patrones de flujo. Primero es necesario extraer los valores verdaderos positivos (VP), verdaderos negativos (VN), falsos positivos (FP) y falsos negativos (FN). La precisión se define en la ecuación (16) y la exactitud en la ecuación (17).

$$Precisión = \frac{VP}{VP + FP} \tag{16}$$

$$Exactitud = \frac{VP + VN}{VP + FN + FP + VN} \tag{17}$$

3. RESULTADOS

Se realizaron pruebas con 110 configuraciones de parámetros diferentes, de las cuales se seleccionaron 8 modelos considerando las configuraciones que generaran menor overfitting durante la fase de entrenamiento e identificaran el mayor número de patrones de flujo en la fase de validación. En la Tabla 2 se muestran los resultados obtenidos en el entrenamiento de la TNN con diferentes variaciones en los parámetros de entrenamiento. La tabla muestra las precisiones obtenidas con cada uno de los modelos en las diferentes fases (entrenamiento, prueba y validación).

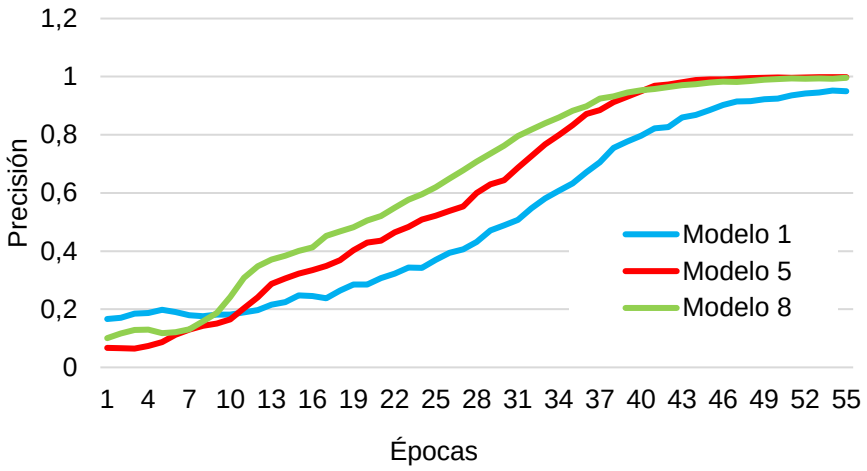
Tabla 2. Resultados de precisión en las fases de entrenamiento, prueba y validación, en diferentes modelos.

Parámetros variables	Modelo 1	Modelo 2	Modelo 3	Modelo 4	Modelo 5	Modelo 6	Modelo 7	Modelo 8
Función de activación	sigmoid	relu	sigmoid	relu	sigmoid	tanh	gelu	relu
N° cabezales de atención	2	4	4	4	1	2	2	2
Dropout	0,3	0,5	0,5	0,1	0,1	0,3	0,3	0,1
Tasa de aprendizaje	0,001	0,001	0,001	0,0005	0,001	0,001	0,001	0,001
Precisión entrenamiento (%)	99,92	99,54	99,49	99,38	99,97	99,61	99,72	99,97
Precisión prueba (%)	52,25	51,23	51,43	51,64	55,53	52,87	49,80	54,10
Precisión validación (%)	51,04	48,98	49,18	52,87	52,25	50,00	48,98	53,07

Fuente. Elaboración propia

Llevando a cabo un análisis detallado a la precisión obtenida por los modelos desarrollados para Transformer Neural Network, fue posible determinar cuáles fueron los mejores resultados durante las fases de entrenamiento y prueba, siendo estos correspondientes a los modelos 1, 5 y 8. Estos modelos se seleccionaron debido a que su precisión supera el 99,9% durante la fase de entrenamiento y el 52% durante la fase de prueba. La Figura 6 muestra la precisión de los 3 modelos seleccionados durante la fase de entrenamiento, en la cual es posible apreciar que los modelos 5 y 8 son similares en su valor máximo.

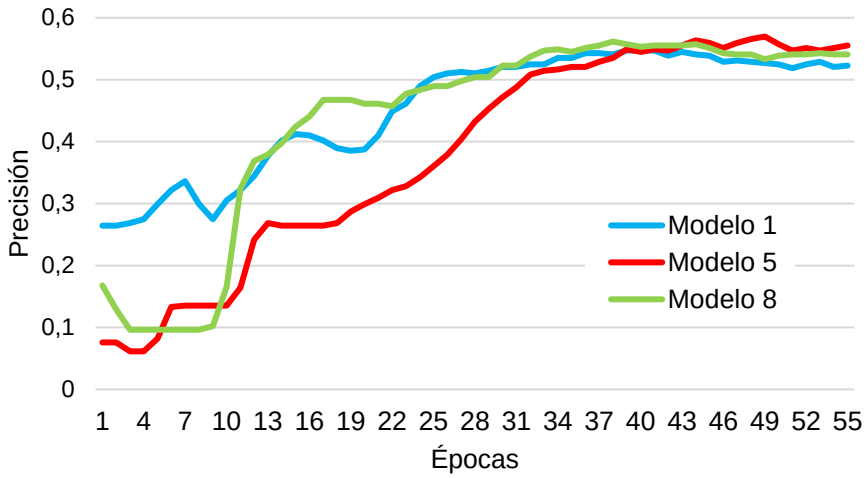
Figura 6. Comportamiento de la precisión durante la fase de entrenamiento.



Fuente. Elaboración propia

La Figura 7 muestra la precisión de los tres modelos seleccionados durante la fase de prueba. Se aprecia que el modelo 5 obtiene una mayor precisión que los demás modelos.

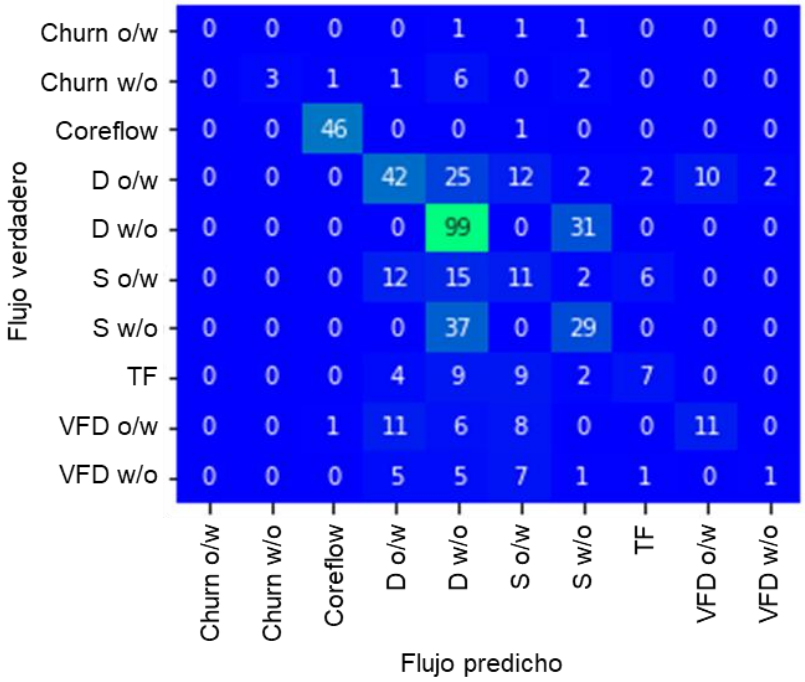
Figura 7. Comportamiento de la precisión durante la fase de prueba.



Fuente. Elaboración propia

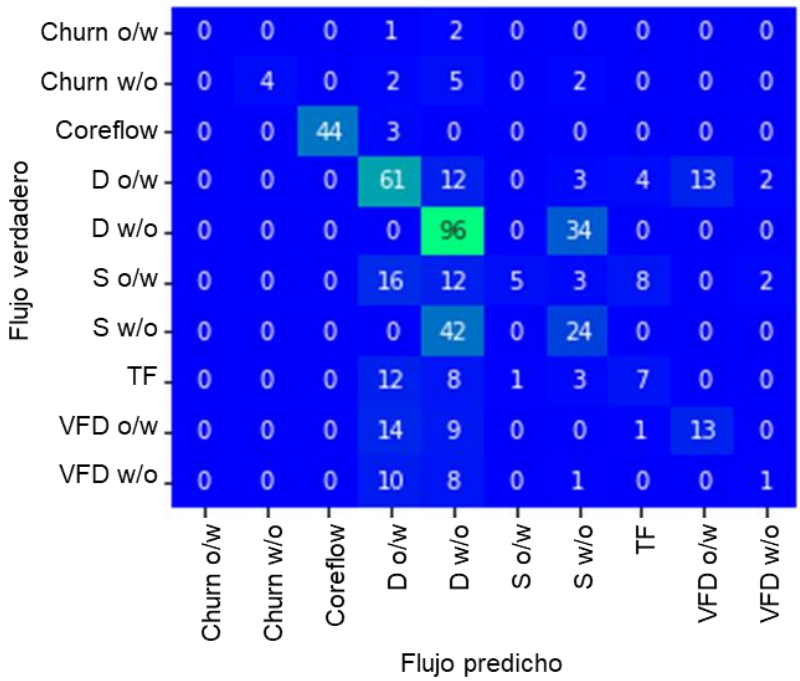
Para los modelos seleccionados se construyeron las matrices de confusión para analizar el comportamiento de las predicciones para cada uno de los patrones de flujo. En la matriz de confusión se observan los verdaderos positivos (VP) para cada categoría, es decir, la cantidad de datos que se predijeron correctamente para un patrón de flujo específico. A su vez, los demás patrones predichos correctamente serán los verdaderos negativos (VN), es decir, los otros valores en la diagonal principal. Los falsos positivos (FP) son aquellos que el modelo cree predecir correctamente, dichos valores son los que corresponden a la columna del patrón de flujo predicho y los falsos negativos, son aquellos valores correspondientes a la fila del patrón de flujo verdadero. Dichas matrices de confusión se pueden apreciar en la Figura 8, Figura 9 y Figura 10. Por ejemplo, para el modelo 1, tomando como referencia el patrón D w/o, se tiene un valor de VP = 99, sumando los demás valores de la diagonal principal se obtiene VN = 150, sumando los demás valores de la columna D w/o se tiene FP = 104, y sumando los demás valores de la fila D o/w se obtiene FN = 31.

Figura 8. Matriz de confusión para el modelo 1.



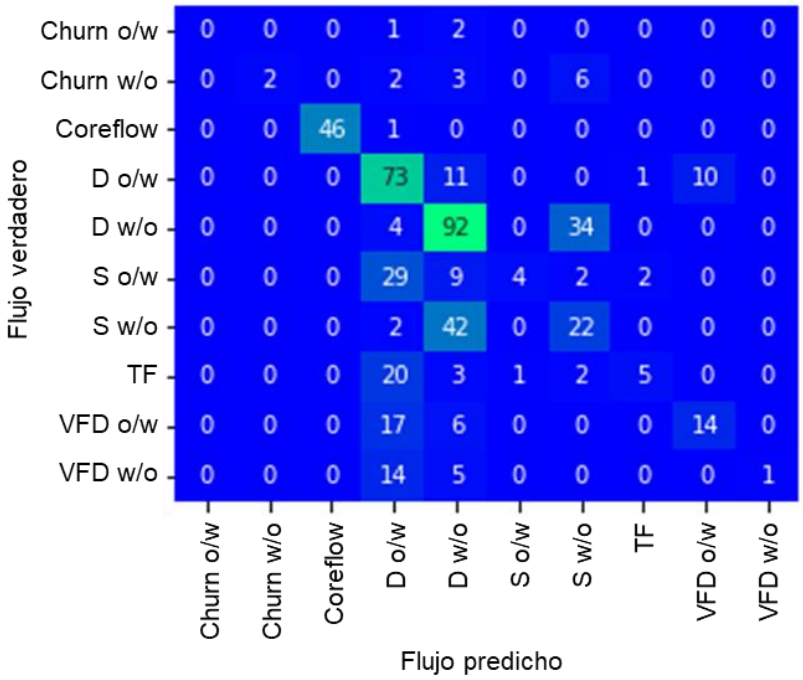
Fuente. Elaboración propia

Figura 9. Matriz de confusión para el modelo 5.



Fuente. Elaboración propia

Figura 10. Matriz de confusión para el modelo 8.



Fuente. Elaboración propia

En la Tabla 3 se muestran los valores de precisión y exactitud por tipo de flujo de los tres modelos seleccionados, dichos valores son obtenidos aplicando las ecuaciones (16) y (17). De acuerdo con la Tabla 3, se observa que el mejor desempeño corresponde al modelo 8, con promedios de precisión de 63,21% y exactitud de 86,51%.

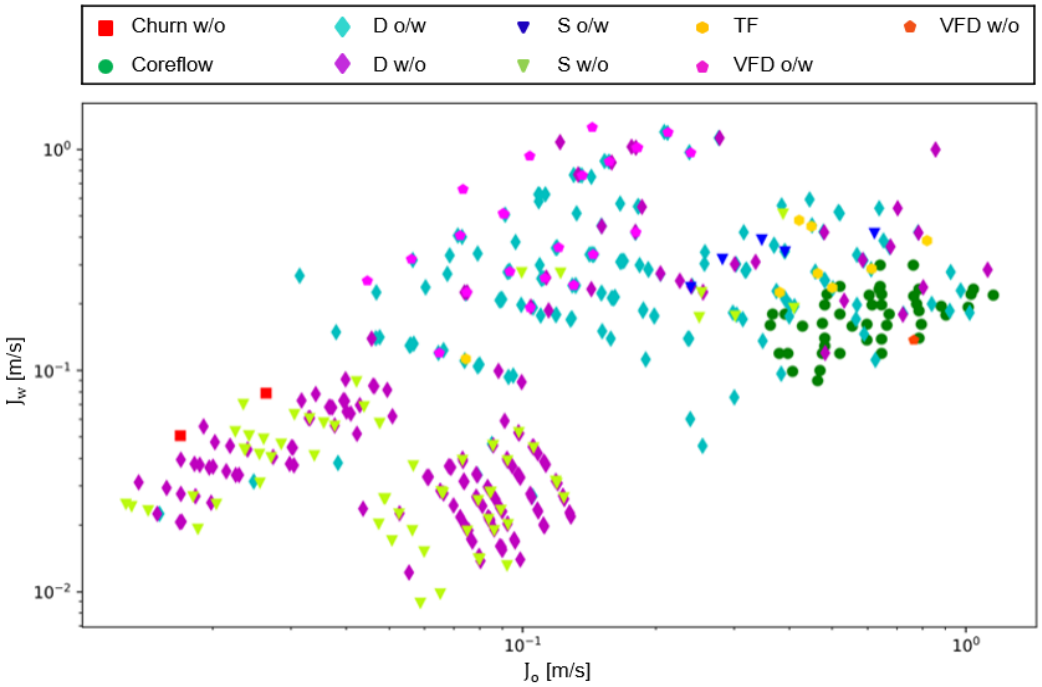
Tabla 3. Resultados de precisión y exactitud por tipo de flujo de acuerdo con las predicciones realizadas por los modelos

	Modelo 1		Modelo 5		Modelo 8	
	Precisión [%]	Exactitud [%]	Precisión [%]	Exactitud [%]	Precisión [%]	Exactitud [%]
Churn o/w	0,00	98,81	0,00	98,84	0,00	98,85
Churn w/o	100,00	96,14	100,00	96,59	100,00	95,93
Coreflow	95,83	98,81	100,00	98,84	100,00	99,62
D o/w	56,00	74,33	51,26	73,49	44,79	70,00
D w/o	48,77	64,84	49,48	65,89	53,18	68,52
S o/w	22,45	77,33	83,33	85,86	80,00	85,76
S w/o	41,43	76,15	34,29	74,34	33,33	74,64
TF	43,75	88,30	35,00	87,33	62,50	89,93
VFD o/w	52,38	87,37	50,00	87,33	58,33	88,70
VFD w/o	33,33	92,25	20,00	91,76	100,00	93,17
Promedio	49,39	85,43	52,34	86,03	63,21	86,51

Fuente. Elaboración propia

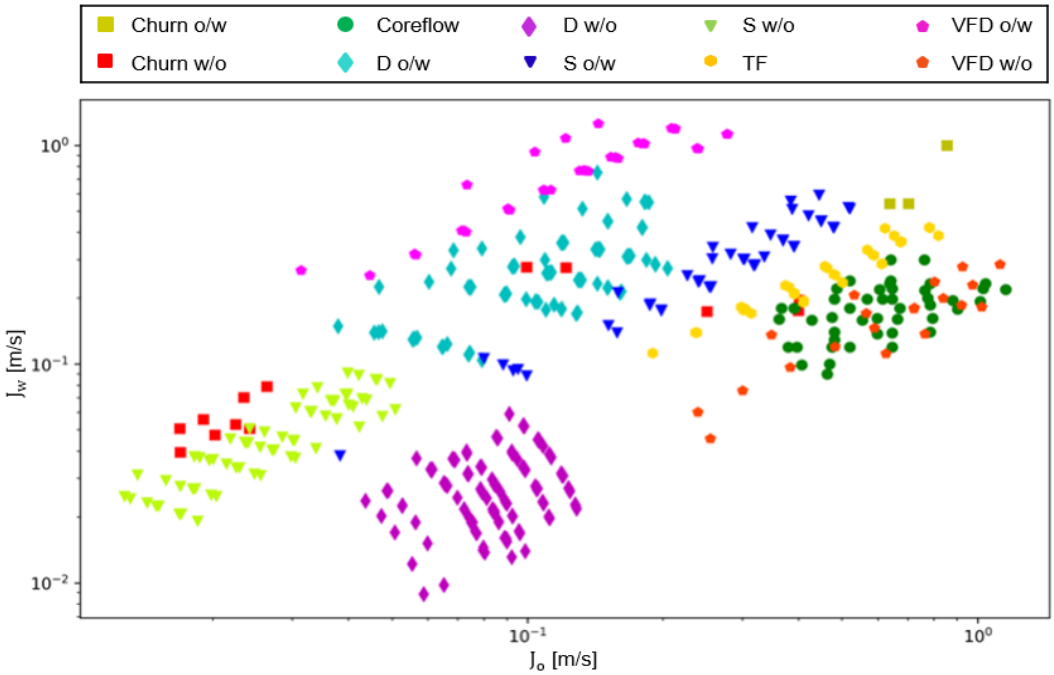
La Figura 11 muestra el mapa de flujo predicho por el modelo 8 y en la Figura 12 se puede observar el mapa de flujo correspondiente a los resultados correctos de la base de datos. En la Figura 11, es posible evidenciar que el modelo seleccionado es capaz de predecir 9 de los 10 tipos de flujo evidenciados en la Figura 12. El modelo 8 no puede predecir el patrón de flujo Churn o/w correctamente, posiblemente debido a que en la base de datos utilizada es el patrón de flujo con mejor cantidad de datos. Para Churn o/w se tienen 21 datos, de los cuales se utilizan solo 15 para el entrenamiento, esto comparado, p. ej., con el patrón D o/w con 1196 datos. El mapa de la Figura 11 muestra una clusterización semejante a los datos de referencia, mostrando mezclas en algunas zonas.

Figura 11. Mapa de flujo predicho por el modelo 8.



Fuente. Elaboración propia

Figura 12. Mapa de flujo con los resultados correctos.



Fuente. Elaboración propia

En la Figura 11, es posible evidenciar que el modelo seleccionado es capaz de predecir 9 de los 10 tipos de flujo evidenciados en la Figura 12, siendo el flujo Churn o/w el ausente en las predicciones del modelo N°8.

4. CONCLUSIONES

Se estructuró una base de datos de patrones de flujo bifásico a partir de la información reportada en la literatura por diferentes autores, para un total de 4864 experimentos, y en la cual se incluyeron 7 parámetros de interés de los flujos multifásicos, y se identifican 10 patrones de flujo distintos.

Se desarrolló una red neuronal Transformer con el software Python para la predicción de patrones de flujo bifásico. La red implementada tiene una capacidad de predicción de patrones de flujo con una precisión de entrenamiento del 99,97 % y de validación del 53,07 % para el modelo 8. Asimismo, este modelo pudo predecir 9 patrones de flujo de los 10 implementados en la fase de entrenamiento, con una precisión promedio para los diferentes patrones de flujo predichos del 63,21 % y una exactitud promedio del 86,51 %. Esto quiere decir que el modelo de TNN es capaz de predecir los diferentes patrones de flujo con una exactitud alta y una precisión media. También, es importante mencionar que la TNN no requiere demasiados recursos tecnológicos para su entrenamiento, dado que en esta fase presenta precisiones cercanas al 100%, usando un tamaño de lote considerable y poca cantidad de épocas, haciendo de la TNN una red neuronal eficiente.

Se construyeron los mapas de flujo a partir de los datos de referencia y con el modelo predictivo, evidenciando una similitud entre el gráfico para los datos predichos y el gráfico para los datos experimentales. Asimismo, se evidencia la clusterización de los datos y las zonas de transición entre patrones de flujo.

Los mejores resultados de predicción de los modelos fueron obtenidos con las funciones de activación sigmoide y relu, con una tasa de aprendizaje del 0,001. En cuanto a la cantidad de

cabezales atencionales, se evidencian buenos resultados con un único cabezal o dos. También, es preciso mencionar que un dropout del 0,1 o el 0,3 es suficiente.

Como recomendación se sugiere implementar una base de datos con mayor cantidad de información y con un equilibrio entre la cantidad de datos por tipos de flujo implementados, para que de esta manera el modelo de TNN no particularice sus predicciones hacia el patrón de flujo con mayor cantidad de datos en la fase de entrenamiento.

BIBLIOGRAFÍA

[1] E. S. Rosa, *Escoamento multifásico isotérmico: modelos de multifluidos e de mistura*. Porto Alegre, Brasil: Bookman, 2012.

[2] M. M. Hernández-Cely and C. Ruiz-Díaz, "Estudio de los fluidos aceite-agua a través del sensor basado en la permitividad eléctrica del patrón de fluido," *Rev. UIS Ing.*, vol. 19, no. 3, pp. 177–186, Apr. 2020, doi: 10.18273/revuin.v19n3-2020017.

[3] D. S. Santos, P. M. Faia, F. A. P. Garcia, and M. G. Rasteiro, "Oil/water stratified flow in a horizontal pipe: Simulated and experimental studies using EIT," *J. Pet. Sci. Eng.*, vol. 174, no. December 2018, pp. 1179–1193, 2019, doi: 10.1016/j.petrol.2018.12.002.

[4] H. Liu *et al.*, "Numerical quasi-three dimensional modeling of stratified oil-water flow in horizontal circular pipe," *Ocean Eng.*, vol. 251, p. 111172, 2022, doi: 10.1016/j.oceaneng.2022.111172.

[5] M. Obaseki, P. T. Elijah, and P. B. Alfred, "Development of model to eliminate sand trapping in horizontal fluid pipelines," *J. King Saud Univ. - Eng. Sci.*, vol. 34, no. 6, pp. 425–434, 2022, doi: 10.1016/j.jksues.2020.11.006.

[6] N. M. de Almeida Coelho *et al.*, "Energy savings on heavy oil transportation through core annular flow pattern: An experimental approach," *Int. J. Multiph. Flow*, vol. 122, p. 103127, 2020, doi: 10.1016/j.ijmultiphaseflow.2019.103127.

[7] E. Sánchez, C. Romero, S. Zeppieri, and D. González-Mendizabal, "Estudio experimental sobre patrones de flujo líquido-líquido en tuberias verticales," 2007.

[8] C. Ruiz-Díaz, M. M. Hernández-Cely, and O. A. González-Estrada, "Modelo predictivo para el cálculo de la fracción volumétrica de un flujo bifásico agua- aceite en la horizontal utilizando una red neuronal artificial," *Rev. UIS Ing.*, vol. 21, no. 2, pp. 155–164, 2022,

doi: 10.18273/revuin.v21n2-2022013.

[9] V. Sanguino, “Modelamiento en CFD de flujo bifásico (aceite-agua) en una tubería vertical,” Universidad de los Andes, 2014.

[10] J. Sossa, “El papel de la inteligencia artificial en la Industria 4.0,” in *Inteligencia artificial y datos masivos en archivos digitales sonoros y audiovisuales*, P. O. Rodríguez Reséndiz, Ed. CDMX: Univ. Nacional Autónoma de México, 2020.

[11] G. A. Valle Tamayo, F. Romero Consuegra, and M. E. Cabarcas Simancas, “Predicción de flujo multifásico en sistemas de recolección de crudo: descripción de requerimientos,” *Rev. Fuentes el Reventón Energético*, vol. 15, no. 1, pp. 87–99, Jul. 2017, doi: 10.18273/revfue.v15n1-2017008.

[12] C. M. Ruiz-Diaz, M. M. Hernández-Cely, and O. A. González-Estrada, “A Predictive Model for the Identification of the Volume Fraction in Two-Phase Flow,” *Cienc. en Desarro.*, vol. 12, no. 2, pp. 49–55, 2021.

[13] I. Jahanandish, B. Salimifard, and H. Jalalifar, “Predicting bottomhole pressure in vertical multiphase flowing wells using artificial neural networks,” *J. Pet. Sci. Eng.*, vol. 75, no. 3–4, pp. 336–342, Jan. 2011, doi: 10.1016/j.petrol.2010.11.019.

[14] F. Semeraro, A. Griffiths, and A. Cangelosi, “Human–robot collaboration and machine learning: A systematic review of recent research,” *Robot. Comput. Integr. Manuf.*, vol. 79, no. August 2022, p. 102432, 2022, doi: 10.1016/j.rcim.2022.102432.

[15] A. Vaswani *et al.*, “Attention Is All You Need,” *31st Conf. Neural Inf. Process. Syst. (NIPS 2017), Long Beach, CA, USA.*, Jun. 2017.

[16] S. M. Lakew, M. Cettolo, and M. Federico, “A comparison of transformer and recurrent neural networks on multilingual neural machine translation,” *COLING 2018 - 27th Int.*

Conf. Comput. Linguist. Proc., pp. 641–652, 2018.

[17] T. B. Brown *et al.*, “Language models are few-shot learners,” *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, pp. 1154–1156, 2020.

[18] N. Safi Samghabadi, P. Patwa, S. Pykl, P. Mukherjee, A. Das, and T. Solorio, “Aggression and Misogyny Detection using BERT: A Multi-Task Approach,” *Proc. Second Work. Trolling, Aggress. Cyberbullying*, no. May, pp. 11–16, 2020.

[19] L. Meng *et al.*, “AdaViT: Adaptive Vision Transformers for Efficient Image Recognition,” pp. 12309–12318, 2021.

[20] N. Q. K. Le and Q. T. Ho, “Deep transformers and convolutional neural network in identifying DNA N6-methyladenine sites in cross-species genomes,” *Methods*, vol. 204, no. November 2021, pp. 199–206, 2022, doi: 10.1016/j.ymeth.2021.12.004.

[21] A. Sharma, P. Jain, A. Mahgoub, Z. Zhou, K. Mahadik, and S. Chaterji, “Lerna: transformer architectures for configuring error correction tools for short- and long-read genome sequencing,” *BMC Bioinformatics*, vol. 23, no. 1, pp. 1–26, 2022, doi: 10.1186/s12859-021-04547-0.

[22] A. Behjati, F. Zare-Mirakabad, S. S. Arab, and A. Nowzari-Dalini, “Protein sequence profile prediction using ProtAlbert transformer,” *Comput. Biol. Chem.*, vol. 99, no. October 2021, p. 107717, 2022, doi: 10.1016/j.compbiolchem.2022.107717.

[23] K. Qu, G. Si, Z. Shan, X. G. Kong, and X. Yang, “Short-term forecasting for multiple wind farms based on transformer model,” *Energy Reports*, vol. 8, pp. 483–490, 2022, doi: 10.1016/j.egyr.2022.02.184.

[24] L. Wang, Y. He, L. Li, X. Liu, and Y. Zhao, “A novel approach to ultra-short-term multi-step wind power predictions based on encoder–decoder architecture in natural language

processing," *J. Clean. Prod.*, vol. 354, no. December 2021, p. 131723, 2022, doi: 10.1016/j.jclepro.2022.131723.

[25] I. Gomaa, A. Gowida, S. Elkatatny, and A. Abdulraheem, "The prediction of wellhead pressure for multiphase flow of vertical wells using artificial neural networks", doi: 10.1007/s12517-021-07099-y/Published.

[26] L. Tunstall, L. Von Werra, and T. Wolf, *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. Sebastopol, US: O'Reilly, 2022.

[27] A. C. Bannwart, O. M. H. Rodriguez, C. H. M. de Carvalho, I. S. Wang, and R. M. O. Vara, "Flow Patterns in Heavy Crude Oil-Water Flow," *J. Energy Resour. Technol.*, vol. 126, no. 3, pp. 184–189, Sep. 2004, doi: 10.1115/1.1789520.

[28] P. Abduvayt, R. Manabe, T. Watanabe, and N. Arihara, "Analisis of oil-water flow tests in horizontal, hilly-terrain, and vertical pipes," *Proc. - SPE Annu. Tech. Conf. Exhib.*, pp. 1335–1347, 2004, doi: 10.2523/90096-ms.

[29] M. Du, N.-D. Jin, Z.-K. Gao, Z.-Y. Wang, and L.-S. Zhai, "Flow pattern and water holdup measurements of vertical upward oil–water two-phase flow in small diameter pipes," *Int. J. Multiph. Flow*, vol. 41, pp. 91–105, May 2012, doi: 10.1016/j.ijmultiphaseflow.2012.01.007.

[30] J. Flores, X. Chen, and J. Brill, "Characterization of Oil-Water Flow Patterns in Vertical and Deviated Wells," in *Proceedings of SPE Annual Technical Conference and Exhibition*, Oct. 1997, vol. 1, pp. 55–64. doi: 10.2523/38810-MS.

[31] T. Ganat, S. Ridha, M. Hairir, J. Arisa, and R. Gholami, "Experimental investigation of high-viscosity oil–water flow in vertical pipes: flow patterns and pressure gradient," *J.*

Pet. Explor. Prod. Technol., vol. 9, no. 4, pp. 2911–2918, Dec. 2019, doi: 10.1007/s13202-019-0677-y.

[32] G. W. Govier, G. A. Sullivan, and R. K. Wood, "The upward vertical flow of oil-water mixtures," *Can. J. Chem. Eng.*, vol. 39, no. 2, pp. 67–75, Apr. 1961, doi: 10.1002/cjce.5450390204.

[33] Y. F. Han, N. D. Jin, L. S. Zhai, H. X. Zhang, and Y. Y. Ren, "Flow pattern and holdup phenomena of low velocity oil-water flows in a vertical upward small diameter pipe," *J. Pet. Sci. Eng.*, vol. 159, no. May, pp. 387–408, Nov. 2017, doi: 10.1016/j.petrol.2017.09.052.

[34] A. R. Hasan and C. S. Kabir, "A New Model for Two-Phase Oil/Water Flow: Production Log Interpretation nd Tubular Calculations," *SPE Prod. Eng.*, vol. 5, no. 02, pp. 193–199, May 1990, doi: 10.2118/18216-PA.

[35] A. R. Hasan and C. S. Kabir, "A simplified model for oil/water flow in vertical and deviated wellbores," *SPE Prod. Facil.*, vol. 14, no. 1, pp. 56–62, 1999, doi: 10.2118/54131-PA.

[36] A. K. Jana, G. Das, and P. K. Das, "Flow regime identification of two-phase liquid-liquid upflow through vertical pipe," *Chem. Eng. Sci.*, vol. 61, no. 5, pp. 1500–1515, 2006, doi: 10.1016/j.ces.2005.09.001.

[37] A. K. Jana, P. Ghoshal, G. Das, and P. K. Das, "An Analysis of Pressure Drop and Holdup for Liquid-Liquid Upflow through Vertical Pipes," *Chem. Eng. Technol.*, vol. 30, no. 7, pp. 920–925, Jul. 2007, doi: 10.1002/ceat.200700033.

[38] J. Guo *et al.*, "Heavy oil-water flow patterns in a small diameter vertical pipe under high temperature/pressure conditions," *J. Pet. Sci. Eng.*, vol. 171, pp. 1350–1365, Dec. 2018,

doi: 10.1016/j.petrol.2018.08.021.

[39] R. A. Mazza and F. K. Suguimoto, “Experimental investigations of kerosene-water two-phase flow in vertical pipe,” *J. Pet. Sci. Eng.*, vol. 184, p. 106580, Jan. 2020, doi: 10.1016/j.petrol.2019.106580.

[40] K. Mydlarz-Gabryk, M. Pietrzak, and L. Troniewski, “Study on oil–water two-phase upflow in vertical pipes,” *J. Pet. Sci. Eng.*, vol. 117, pp. 28–36, May 2014, doi: 10.1016/j.petrol.2014.03.007.

[41] O. M. H. Rodriguez and A. C. Bannwart, “Experimental study on interfacial waves in vertical core flow,” *J. Pet. Sci. Eng.*, vol. 54, no. 3–4, pp. 140–148, 2006, doi: 10.1016/j.petrol.2006.07.007.

[42] J. Xu, D. Li, J. Guo, and Y. Wu, “Investigations of phase inversion and frictional pressure gradients in upward and downward oil–water flow in vertical pipes,” *Int. J. Multiph. Flow*, vol. 36, no. 11–12, pp. 930–939, Nov. 2010, doi: 10.1016/j.ijmultiphaseflow.2010.08.007.

[43] Y. Yang *et al.*, “Oil-Water flow patterns, holdups and frictional pressure gradients in a vertical pipe under high temperature/pressure conditions,” *Exp. Therm. Fluid Sci.*, vol. 100, no. September 2018, pp. 271–291, Jan. 2019, doi: 10.1016/j.expthermflusci.2018.09.013.

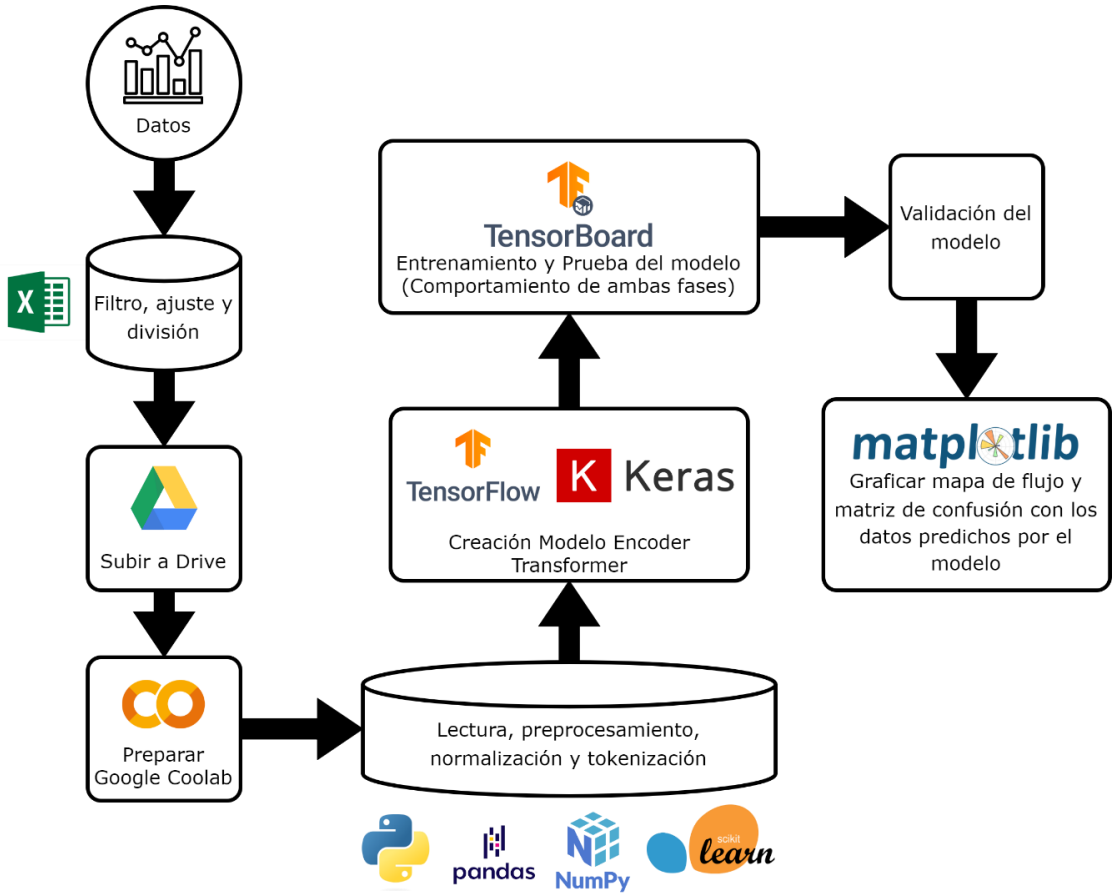
[44] D. Zhao, L. Guo, X. Hu, X. Zhang, and X. Wang, “Experimental study on local characteristics of oil-water dispersed flow in a vertical pipe,” *Int. J. Multiph. Flow*, vol. 32, no. 10–11, pp. 1254–1268, 2006, doi: 10.1016/j.ijmultiphaseflow.2006.06.004.

ANEXOS

Anexo A. Implementación del código TNN en Python

A continuación, se presenta la metodología practica desarrollada para el tratamiento de la base de datos implementada en el modelo de red neuronal en sus diferentes fases (entrenamiento, prueba y validación) y a su vez, se presenta el paso a paso del código de Python realizado para el desarrollo de este proyecto, como se evidencia de forma resumida en la Figura 13.

Figura 13. Diagrama de flujo del paso a paso del código en Python con las herramientas usadas en cada fase.



Fuente. Elaboración propia

En primer lugar, se debe filtrar la base de datos tomando los valores máximos y mínimos de cada uno de los parámetros de entrada a la red neuronal. Este proceso, se realizó de forma equitativa debido a que era necesario solo tomar el 80% de los datos correspondientes a cada tipo de flujo y con ellos formar la base de datos destinada a la fase de entrenamiento de la red neuronal.

El 20% de los datos restantes de cada flujo fue dividido en dos partes iguales para conformar los sets de datos correspondientes a la fase de prueba y la fase de validación. Es decir, que con este proceso se obtendría la siguiente división de la base de datos general: 80% Entrenamiento (3888 datos), 10% Prueba (488 datos) y 10% Validación (488 datos). A su vez, es necesario convertir a números las etiquetas de los patrones de flujo, siendo 10 patrones en total, estos se renombran de 0 a 9 dado que en el lenguaje de programación los datos comienzan a leerse a partir de 0.

A continuación, se subieron a la interfaz de Google Drive los 3 archivos Excel correspondientes a las bases de datos de las diferentes fases, esto debido a que para el presente proyecto se hizo uso de la interfaz llamada Google Colaboratory. Esta interfaz es una herramienta que nos permite hacer uso del lenguaje Python en un cuaderno Jupyter sin necesidad de instalar un programa enfocado al desarrollo del código necesario para la realización de este proyecto. Una vez dentro de Google Colaboratory nos dispondremos a preparar Google Drive para posteriormente leer en el código los archivos Excel previamente subidos.

```
from google.colab import drive
drive.mount('/gdrive')
```

Después, nos dispondremos a importar las librerías de numpy y pandas, las cuales son muy útiles para la lectura y manejo de datos.

```
import pandas as pd
import numpy as np
```

Se definieron las tres rutas correspondientes a los archivos Excel de las bases de datos y seguidamente se leyeron con ayuda del comando *pd.read_excel(ruta, header=0)* .

```
ruta1 = '/gdrive/MyDrive/2021-
TransformerNN/Documentos/DATA TRAIN 2.0.xlsx'
ruta2 = '/gdrive/MyDrive/2021-
TransformerNN/Documentos/DATA TEST 2.0.xlsx'
ruta3 = '/gdrive/MyDrive/2021-
TransformerNN/Documentos/DATA VAL 2.0.xlsx'
df_train = pd.read_excel(ruta1, header=0)
df_test = pd.read_excel(ruta2, header=0)
df_val = pd.read_excel(ruta3, header=0)
```

Luego, se concatenaron las bases de datos de manera vertical mediante *pd.concat([matriz1,matriz2], axis=0)* , esto para hacer más fácil el manejo de la base de datos a lo largo del código.

```
df = pd.concat([df_train,df_test], axis=0)
df = pd.concat([df,df_val], axis=0)
```

Se realiza una división de las entradas y salidas del modelo para cada una de las fases (X para las entradas & Y para las salidas) , para ello es necesario conocer donde delimitan los datos luego de concatenarlos. El comando *.iloc[filas,columnas]* sirve para particionar los datos en un rango especificado y *.values* para extraer los datos de la partición anterior.

```
Xtrain = df.iloc[:3888,0:7].values
Ytrain = df.iloc[:3888,7:].values
Xtest = df.iloc[3888:4376,0:7].values
Ytest = df.iloc[3888:4376,7:].values
Xval = df.iloc[4376:,0:7].values
```

```
Yval = df.iloc[4376:,7:].values
```

Seguidamente, se convierten a DataFrame las listas definidas anteriormente, esto mediante el comando *pd.DataFrame(lista)*.

```
Xtrain = pd.DataFrame(Xtrain)
Xtest = pd.DataFrame(Xtest)
Xval = pd.DataFrame(Xval)
Ytrain = pd.DataFrame(Ytrain)
Ytest = pd.DataFrame(Ytest)
Yval = pd.DataFrame(Yval)
```

Después, se concatenan únicamente los DataFrame correspondientes a los datos de entrada del modelo.

```
data_input = pd.concat([Xtrain,Xtest], axis=0)
data_input = pd.concat([data_input,Xval], axis=0)
```

Se procede a realizar una normalización de los datos de entrada, para ello es necesario importar *preprocessing* de *sklearn* y seguidamente utilizar el comando *preprocessing.MinMaxScaler(feature_range=(inicio,fin))* para configurar el rango de la normalización, seguido por el comando *min_max_scaler.fit_transformer(DataFrame)*. La normalización se realiza en un rango de 1 a 2 debido a que si se realiza de 0 a 1 es posible que el modelo de TNN tome los valores en 0 como nulos. Es necesario convertir a DataFrame debido a que el proceso de normalización convierte a los datos en una lista.

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler(feature_range = (1,
2))
df_s = min_max_scaler.fit_transform(data_input)
df_s = pd.DataFrame(df_s, columns=df_params.columns) # Numpy a p
andas
```

A continuación, se concatenan únicamente los DataFrame correspondientes a los datos de salida del modelo.

```
data_output = pd.concat([Ytrain,Ytest], axis=0)
data_output = pd.concat([data_output,Yval], axis=0)
```

Para clasificación de secuencias es necesario transformar las etiquetas de salida en una codificación en caliente (one hot encoding), es decir, una representación de variables categóricas como vectores binarios, por ello fue necesario que las etiquetas se asignasen a valores enteros (en este caso de 0 a 9). Luego, cada valor entero se representa como un vector binario que tiene todos los valores en cero excepto el índice del entero, el cual está marcado con un 1. Para ello es necesario importar *to_categorical* de *tensorflow.keras.utils*, luego se dispone a realizar la codificación en caliente mediante el comando *to_categorical(DataFrame)*. Es necesario convertir a DataFrame debido a que el proceso anterior convierte a los datos en una lista.

```
from tensorflow.keras.utils import to_categorical
y = to_categorical(data_output)
output = pd.DataFrame(y) # Numpy a pandas
output.head()
```

Después, es necesario definir una función para la creación de un diccionario el cual contendrá los datos de entrada. Con este diccionario es posible realizar posteriormente un paso importante de las redes Transformer, denominado Tokenizar. La función se definió de la siguiente manera.

```
def crear_diccionario(num_columna):
    for sec in df_s.iloc[:,num_columna]:
      if sec not in dicc_params:
        dicc_params[sec] = len(dicc_params)
    return dicc_params
```

Se define el diccionario dicc_params con un elemento en su interior, el cual pertenece al indicativo 0 que corresponde al valor 0, esto debido a que la red Transformer toma los valores 0 como nulos en el proceso de atención. Seguidamente, se aplica la función definida con anterioridad a cada una de las columnas correspondientes a los parámetros de entrada.

```
dicc_params = {0:0}
for i in range(0,7):
  dicc_input = crear_diccionario(i)
```

Es necesario convertir el DataFrame de los datos normalizados a un array, este proceso se realiza mediante el comando *np.array(DataFrame)*. Posteriormente, es necesario definir la función que realiza el proceso de tokenización, en este proceso se reemplazan los valores de la base de datos por los índices correspondientes del diccionario previamente construido. Es decir, que la base de datos de entrada estará compuesta por números enteros en lugar de decimales.

```
df_array = np.array(df_s)
def codificar_tokens(secs, dicc):
  secs_codif = []
  for sec in secs:
    sec_codif = []
    for token in sec:
      sec_codif.append(dicc[token])
    secs_codif.append(sec_codif)
  return secs_codif
encoder_input = codificar_tokens(df_array,dicc_input)
```

Luego, es necesario convertir a DataFrame la matriz de datos de entrada construida anteriormente y concatenarla de manera horizontal con los datos de salida del modelo. Este proceso se realiza con el fin de asegurarnos de que las entradas y las salidas se emparejen correctamente.

```
encoder_input = pd.DataFrame(encoder_input, columns=df_params.co
lumns)
data = pd.concat([encoder_input,output], axis=1)
```

Se requiere particionar de nuevo la base de datos, para con ello definir los valores de entrada y salida del modelo en cada una de las fases de entrenamiento, prueba y validación.

```
X_train = data.iloc[:3888,0:7].values
Y_train = data.iloc[:3888,7:].values
X_test = data.iloc[3888:4376,0:7].values
Y_test = data.iloc[3888:4376,7:].values
X_val = data.iloc[4376:,0:7].values
Y_val = data.iloc[4376:,7:].values
```

Seguidamente, es necesario importar la librería de tensorflow, importar la librería keras de tensorflow e importar la herramienta layers de tensorflow.keras, además de importar sqrt de math para hacer uso de la raíz cuadrada, estas librerías son útiles para la construcción de redes neuronales.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from math import sqrt
```

Para nuestro modelo es importante definir la codificación posicional, este proceso sirve para equipar cada parámetro de entrada con un vector de información sobre su posición en la cadena de parámetros de entrada. En otras palabras, se mejora la entrada del modelo para inyectar el orden de los parámetros. Este procedimiento fue propuesto junto a la arquitectura del Transformer, dado que en cadenas de palabras es muy útil dadas las grandes longitudes que puede manejar, este paso se definió de la siguiente manera.

$$\vec{p}_t^{(i)} = f(t)^{(i)} = \begin{cases} sen(w_t.t), si\ i = 2k \\ cos(w_t.t), si\ i = 2k+1 \end{cases}$$

Donde,

$$w_t = \frac{1}{10000^{2k/d}}$$

En este caso en particular, al ser solo 7 parámetros de entrada por cada dato, no es muy perceptible la gran utilidad de este paso. La capa de codificación posicional se define en dos funciones, así:

```
def get_angles(pos, i, d_model):
  angle_rates = 1 / np.power(10000, (2 * (i//2)) / np.float32(d_
model))
  return pos * angle_rates

def positional_encoding(position, d_model):
  angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                          np.arange(d_model)[np.newaxis, :],
                          d_model)
  # apply sin to even indices in the array; 2i
  angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
  # apply cos to odd indices in the array; 2i+1
  angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
  pos_encoding = angle_rads[np.newaxis, ...]

  return tf.cast(pos_encoding, dtype=tf.float32)
```

Seguidamente, se implementa la capa de incrustación la cual es previa a la capa de codificación posicional, por lo cual introducimos la función de posicionamiento dentro de la definición de la función de incrustación y posición. Al momento de retornar los valores de salida de la capa este sea una combinación del dato original con el vector de incrustación y el vector de posicionamiento para su posterior ingreso al codificador del Transformer. Esta capa se define de la siguiente manera:

```
class TokenAndPositionEmbedding(layers.Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        self.d_model = embed_dim
        super(TokenAndPositionEmbedding, self).__init__()
```

```
        self.embedding=layers.Embedding(input_dim=vocab_size,
                                        output_dim=embed_dim)
        self.pos_encoding=positional_encoding(maxlen,embed_dim)

    def call(self, x):
        seq_len = tf.shape(x)[1]

        # añadir embedding and position encoding.
        x = self.embedding(x)  # (batch_size, input_seq_len, d_m
odel)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x += self.pos_encoding[:, :seq_len, :]
        return x
```

Posteriormente, se implementa el bloque codificador del Transformer, el cual está compuesto por la capa de cabezales atencionales, seguida de una capa de dropout, luego una capa de normalización, continua una capa de avance, seguida de otra capa de dropout y finalmente una segunda capa de normalización. Todas estas capas se definen en una misma función para así construir la estructura que compone al bloque codificador de la TNN, dicha función se define de la siguiente forma:

```
class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, dropout):
        super(TransformerBlock, self).__init__()
        self.att = layers.MultiHeadAttention(num_heads=num_heads
,
                                             key_dim=embed_dim)
        self.ffn = keras.Sequential([
            layers.Dense(ff_dim, activation=ACTIVATION),
            layers.Dense(embed_dim),
        ])
        self.layernorm1 = layers.LayerNormalization(epsilon=1e-
6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-
6)
        self.dropout1 = layers.Dropout(dropout)
        self.dropout2 = layers.Dropout(dropout)


    def call(self, inputs, training):
```

```
        attn_output=self.att(inputs, inputs)  # self-
attention layer
        attn_output = self.dropout1(attn_output, training=traini
ng)
        out1 = self.layernorm1(inputs+attn_output)#layer norm
        ffn_output=self.ffn(out1) #feed-forward layer
        ffn_output=self.dropout2(ffn_output,training=training)
        return self.layernorm2(out1 + ffn_output) #layer norm
```

Por último, se define el Encoder el cual está compuesto por varios bloques codificadores, esto con el fin de construir correctamente la anatomía de la TNN, es preciso aclarar que para este proyecto se hizo uso de un único bloque codificador, por lo cual esta función pudo haberse ignorado. La función Encoder se encuentra definida de la siguiente manera:

```
class Encoder(tf.keras.layers.Layer):
  def __init__(self, num_layers, embed_dim, num_heads, ff_dim, d
ropout):
    super(Encoder, self).__init__()

    self.enc_layers = [TransformerBlock(embed_dim, num_heads, ff
_dim, dropout)
                       for _ in range(num_layers)]

  def call(self, x, training):
    for i in range(num_layers):
      x = self.enc_layers[i](x, training)

    return x  # (batch_size, input_seq_len, d_model)
```

A continuación, es necesario escribir el comando *rm -rf ./logs/*, el cual es necesario para borrar el cache de la interfaz TensorBoard, dicha interfaz es útil para observar el comportamiento de la perdida y precisión del modelo durante la fase de entrenamiento y prueba. También, es necesario importar datatime, luego se define el tamaño máximo de cada entrada (en este caso son 7 parámetros por entrada) y se define el tamaño total del diccionario usado en la tokenización. Se define la función de activación que usaremos en las diferentes capas de la red neuronal.

Seguidamente se precisan los demás parámetros, el número de codificadores (*num_layers=1*) y el número de cabezales atencionales (*num_heads*). También, se configura el embed_dim y el ff_dim en 32, siguiendo como base el número establecido en el texto original de 256 y dividiéndolo en la mitad hasta llegar al número 4 (256/128/64/32/16/8/4), encontrando que el 32 proporciona mejores resultados. Además, se define el dropout que será usado por las capas de dropout a lo largo del modelo.

```
import datetime
maxlen = tf.shape(X_train)[-1] #7
vocab_size_1 = vocab_size_2 = len(dicc_input) #Tamaño del diccio
nario
ACTIVATION = "relu"
num_layers = 1
num_heads_1 = 4 # Número de cabezas de atención
embed_dim_1 = 32 # Dimensión de las capas del codificador
ff_dim_1 = 32
dropout_1 = 0.1
```

Se debe definir el modelo a implementar mediante el comando *keras.Sequential()*, seguido del comando *.add(layers.Input(shape=(maxlen,)))* para la capa de inicio del modelo. Con el comando *.add(función)* podremos agregar una a una las funciones que componen el modelo de TNN con sus respectivos parámetros de funcionamiento. Posteriormente, se agrega una capa para disminuir la dimensión de los datos con el comando *layers.GlobalAveragePooling1D()*, se agrega una capa de dropout con el comando *layers.Dropout(dropout).* Al final, se agrega una capa Softmax con 10 neuronas (equivalente a las 10 categorías de flujo presentes) para medir de 0 a 1 la probabilidad de predecir el tipo de flujo correcto.

```
model_1 = keras.Sequential()
model_1.add(layers.Input(shape=(maxlen, )))
model_1.add(TokenAndPositionEmbedding(maxlen, vocab_size_1, embe
d_dim_1))
```

```
model_1.add(Encoder(num_layers,embed_dim_1, num_heads_1, ff_dim_
1, dropout_1))
model_1.add(layers.GlobalAveragePooling1D()) #...
model_1.add(layers.Dropout(dropout_1))
model_1.add(layers.Dense(10, activation='softmax'))
```

Es necesario comprimir el modelo, para este caso se implementa como optimizador “Adam”, al cual se le variará la tasa de aprendizaje, además, se implementará como función de perdida “binary crossentropy” y como métricas “accuracy”.

```
model_1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate
=0.001), loss="binary_crossentropy", metrics=["accuracy"])
```

Posteriormente, se procede a configurar la herramienta TensorBoard, para que esta se enlace directamente con el modelo y sea posible visualizar las gráficas de comportamiento durante la fase de entrenamiento y prueba.

```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d
-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=lo
g_dir, histogram_freq=1)
```

Se define el número de épocas de entrenamiento en 55, dado que este era el promedio en que se estabilizaba el comportamiento de la precisión de entrenamiento y de prueba. Luego, se procede a entrenar el modelo mediante el comando *.fit(entradas, salidas, épocas, validation_data=(entradas, salidas), callbacks=[tensorboard_callbaks], batch_size=tamaño de lote)*. El tamaño de lote fue definido como 2800, debido a que, si este se aumentaba, la precisión del modelo era menor y si se disminuía, el overfiting aumentaría considerablemente.

```
EPOCHS = 55
hist = model_1.fit(
      X_train, Y_train, epochs=EPOCHS, validation_data=(X_test,Y
_test), callbacks=[tensorboard_callback],
      batch_size = 2800)
```

Se evalúa cada una de las fases del modelo, primero la fase de entrenamiento y prueba, esto se realiza mediante el comando *.evaluate(entradas, salidas)* y posteriormente se imprimen los resultados para así conocer las precisiones alcanzadas por el modelo luego del entrenamiento.

```
acc_train = model_1.evaluate(X_train,Y_train)
acc_test = model_1.evaluate(X_test,Y_test)
print(f'model_1 - exactitud entrenamiento/prueba: {100*acc_train
[1]:.2f}%/{100*acc_test[1]:.2f}%')
```

Luego, se evalúa la fase de validación, lo cual significa que el modelo de TNN predecirá datos que no ha visto o evaluado previamente, a su vez, debemos imprimir el resultado.

```
puntaje = model_1.evaluate(X_val,Y_val,verbose=0)
print(f'Validación modelo 1: exactitud {100*puntaje[1]:.2f}%')
```

Mediante la herramienta de TensorBoard podremos observar el comportamiento a lo largo de las épocas establecidas para la precisión y la pérdida del modelo durante las fases de entrenamiento y la prueba. Por otra parte, esta herramienta puede tardar un poco en ejecutarse, pero con ella es posible realizar un suavizado de las gráficas de comportamiento y a su vez descargar los datos que componen la gráfica. Para cargar esta herramienta es necesario ejecutar los siguientes comandos.

```
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

Seguidamente, se realizan los preparativos para graficar el mapa de flujo correspondiente a la predicción realizada por el modelo, para ello solo es necesario extraer los datos pertenecientes a la

velocidad superficial del aceite y del agua de la base de datos construida para la fase de validación. Es decir, se extraen los datos de todas las filas, pero exclusivamente de las columnas 0 y 1.

```
Jo = Xval.iloc[:,0].values
Jw = Xval.iloc[:,1].values
```

Se crean dos listas por cada tipo de flujo, una para los datos de velocidad superficial del aceite y otra para la velocidad superficial del agua , en este caso tendremos 10 casos (del 0 al 9) lo cual serian 20 listas en total.

```
"caso 0"
Jo_churn_ow = []
Jw_churn_ow = []
"caso 1"
Jo_churn_wo = []
Jw_churn_wo = []
"caso 2"
Jo_coreflow = []
Jw_coreflow = []
"caso 3"
Jo_D_ow = []
Jw_D_ow = []
"caso 4"
Jo_D_wo = []
Jw_D_wo = []
"caso 5"
Jo_S_ow = []
Jw_S_ow = []
"caso 6"
Jo_S_wo = []
Jw_S_wo = []
"caso 7"
Jo_TF = []
Jw_TF = []
"caso 8"
Jo_VFD_ow = []
Jw_VFD_ow = []
"caso 9"
Jo_VFD_wo = []
Jw_VFD_wo = []
```

Se procede a realizar y registrar las predicciones realizadas por el modelo, para ello es necesario crear una lista vacía en la cual se guardarán las etiquetas de los valores predichos. Seguidamente, mediante el comando *modelo.predict(datos entrada)* se realiza la predicción y después se convierte a DataFrame mediante *pd.DataFrame()*, para luego ser convertida a array mediante el comando *np.array()*. A continuación, se convierte a número entero los resultados de la predicción dado que estos se encuentran configurados en un vector one-hot. Para ello se utiliza un siclo *for* en el cual se ejecuta el comando *.append()* para agregar el valor a la lista vacía definida previamente, seguido del comando *np.argmax()* para extraer la posición del máximo valor en el vector one-hot.

```
y_predict = []
prediccion = model_1.predict(X_val)
resultados = pd.DataFrame(prediccion)
resultados_array = np.array(resultados)
for i in range (len(X_val)):
  y_predict.append(np.argmax(resultados_array[i,:]))
```

Antes de graficar el mapa de flujo se realizaron dos gráficos adicionales, el primer grafico es la comparativa de los datos reales con los datos predichos y a su vez, muestra la cantidad de errores obtenidos durante la fase de validación, un ejemplo de grafica generada se evidencia en la **¡Error! No se encuentra el origen de la referencia.**. Esto se realiza de la siguiente manera:

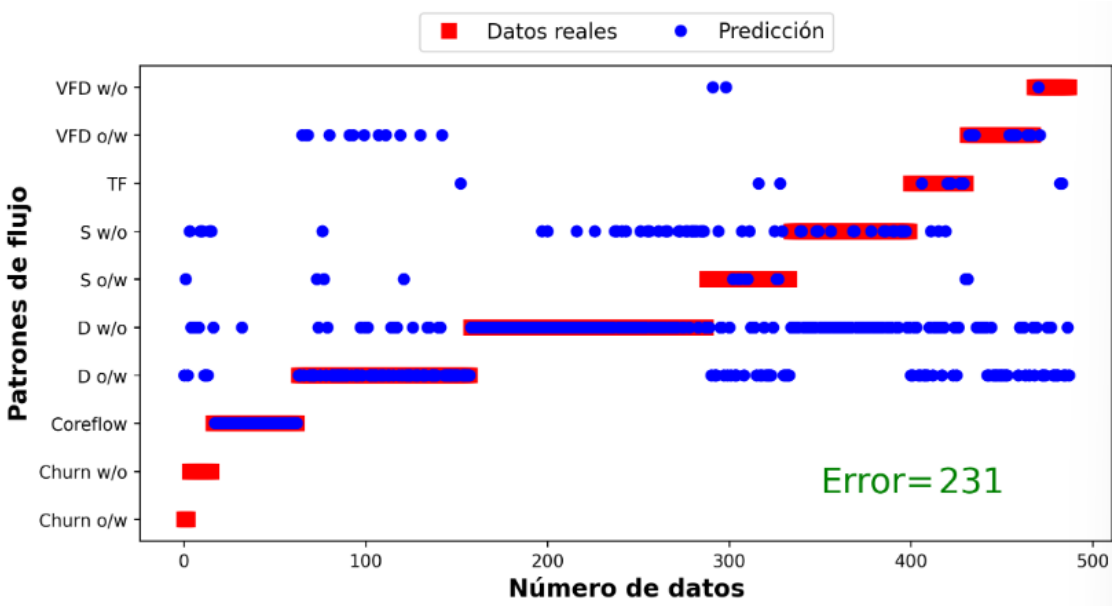
```
Resultados1=np.array(y_predict).reshape(len(X_val),)
eje_x = np.arange(0,len(Resultados1),1)
y_datos = np.array(Yval)
eje_y = y_datos.reshape(len(y_datos),)
error = 0
for i in range(len(X_val)):
  if eje_y[i] != Resultados1[i]:
    error += 1
```

A continuación, se importa la librería matplotlib.pyplot, es una herramienta muy útil para la realización de graficas en Python. Se configura el tamaño de la imagen y la fuente del texto, luego

se configura la leyenda junto a los marcadores, en este caso “rs”(r=rojo, s=cuadrado) para los datos reales y “bo”(b=azul, o=circulo). Se requiere establecer el nombre al que pertenece cada marcador y su tamaño. Se nombran y configuran los ejes del gráfico y se establece la escala (lineal, en este caso). A su vez, se instauran las etiquetas de los flujos a lo largo del eje Y. Por último, se imprime en el gráfico el texto del error calculado y se ubica la caja con la leyenda, como se aprecia en la Figura 14.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5),dpi=250)
plt.rcParams["font.family"] = "Times New Roman"
plt.plot(eje_x,eje_y,"rs", label="Datos reales",ms=8)
plt.plot(eje_x,Resultados1,"bo", label="Predicción",ms=6)
plt.xlabel("Número de datos",fontsize=14,fontweight="bold")
plt.ylabel("Patrones de flujo",fontsize=14,fontweight="bold")
plt.xscale("linear")
plt.yscale("linear")
yticks = ["Churn o/w", "Churn w/o", "Coreflow", "D o/w", "D w/o"
, "S o/w", "S w/o", "TF", "VFD o/w", "VFD w/o"]
plt.yticks(np.arange(10),yticks)
plt.text(350, 0.55, 'Error=', fontsize=20, color='green')
plt.text(415, 0.55, error, fontsize=20, color='green')
plt.legend(bbox_to_anchor=(0., 1.04, 1., 0.06), loc="center", nc
ol=2, borderaxespad=0., fontsize=12)
plt.show()
```

Figura 14. Ejemplo gráfica de Predicción vs Valores reales.

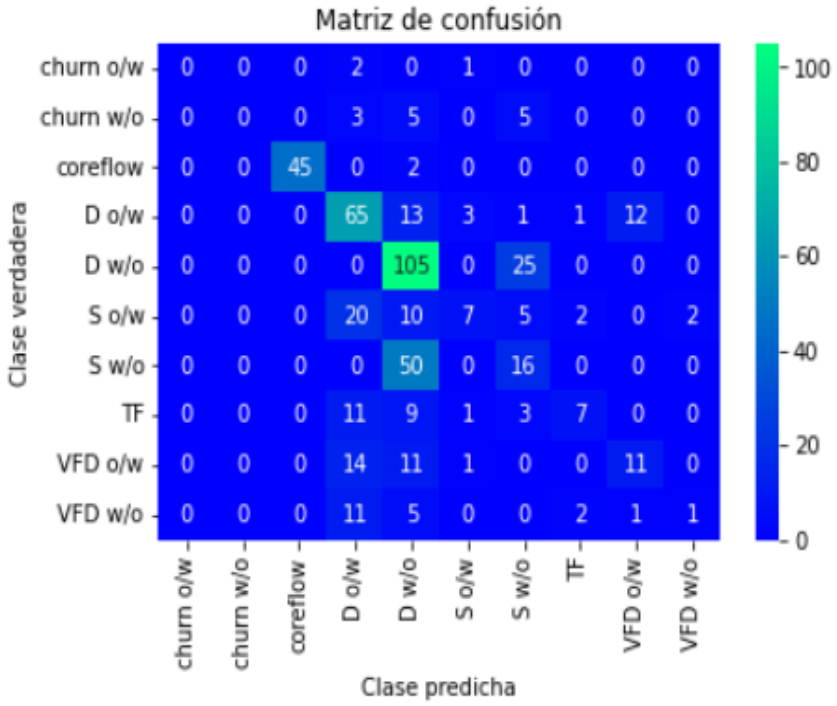


Fuente. Elaboración propia

El segundo gráfico es una matriz de confusión, la cual permite observar cómo se están realizando las predicciones y su desempeño. Un ejemplo de dicha matriz se evidencia en la **¡Error! No se encuentra el origen de la referencia.**. Para ello, es necesario importar confusion_matrix de sklearn.metrics y seaborn. Esta se realizó de la siguiente manera:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
matrix = confusion_matrix(eje_y, Resultados1)
columns = ["churn o/w","churn w/o","coreflow","D o/w","D w/o","S
 o/w","S w/o","TF","VFD o/w","VFD w/o"]
df_matrix = pd.DataFrame(matrix,index=columns,columns=columns)
graphic = sns.heatmap(df_matrix,cmap='winter',annot=True,fmt="d"
)
plt.title('Matriz de confusión')
plt.xlabel('Clase predicha')
plt.ylabel('Clase verdadera')
plt.show()
```

Figura 15. Ejemplo matriz de confusión.



Fuente. Elaboración propia

Volviendo con el mapa de flujo de los valores predichos, es necesario guardar en las diferentes listas de los patrones de flujo, los valores correspondientes a la velocidad superficial del aceite y del agua. Para ello, se realiza un for con un if y condiciones elif anidadas, esto asegura el guardado de los datos de forma correcta de acuerdo con el número entero que representa cada patrón de flujo.

```
for i in range(0,len(y_predict)):
  flujo = y_predict[i]
  if flujo==0:
    Jo_churn_ow.append(Jo[i])
    Jw_churn_ow.append(Jw[i])
  elif flujo==1:
    Jo_churn_wo.append(Jo[i])
    Jw_churn_wo.append(Jw[i])
  elif flujo==2:
    Jo_coreflow.append(Jo[i])
    Jw_coreflow.append(Jw[i])
```

```
  elif flujo==3:
    Jo_D_ow.append(Jo[i])
    Jw_D_ow.append(Jw[i])
  elif flujo==4:
    Jo_D_wo.append(Jo[i])
    Jw_D_wo.append(Jw[i])
  elif flujo==5:
    Jo_S_ow.append(Jo[i])
    Jw_S_ow.append(Jw[i])
  elif flujo==6:
    Jo_S_wo.append(Jo[i])
    Jw_S_wo.append(Jw[i])
  elif flujo==7:
    Jo_TF.append(Jo[i])
    Jw_TF.append(Jw[i])
  elif flujo==8:
    Jo_VFD_ow.append(Jo[i])
    Jw_VFD_ow.append(Jw[i])
  elif flujo==9:
    Jo_VFD_wo.append(Jo[i])
    Jw_VFD_wo.append(Jw[i])
```

Por último, al igual que en el primer gráfico, es necesario configurar el tamaño, la fuente y la leyenda correspondiente a cada patrón de flujo. Para este caso, se realiza mediante un if, debido a que, si el modelo no es capaz de predecir ningún dato con la etiqueta en cuestión, este no aparecerá en la leyenda del gráfico. A su vez, se configuran los ejes y se establece la escala (log, logarítmica en este caso). También, es necesario ubicar la caja con la leyenda del mapa de flujo.

```
plt.figure(figsize=(10.4,6),dpi=250)
plt.rcParams["font.family"] = "Times New Roman"
if len(Jo_churn_ow) != 0:
  plt.plot(Jo_churn_ow,Jw_churn_ow,"ys",label="Churn o/w",ms=6)
if len(Jo_churn_wo) != 0:
  plt.plot(Jo_churn_wo,Jw_churn_wo,"rs",label="Churn w/o",ms=6)
if len(Jo_coreflow) != 0:
  plt.plot(Jo_coreflow,Jw_coreflow,"go",label="Coreflow",ms=6)
if len(Jo_D_ow) != 0:
  plt.plot(Jo_D_ow,Jw_D_ow,"cd",label="D o/w",ms=6)
if len(Jo_D_wo) != 0:
  plt.plot(Jo_D_wo,Jw_D_wo,"md",label="D w/o",ms=6)
if len(Jo_S_ow) != 0:
```

```
  plt.plot(Jo_S_ow,Jw_S_ow,"bv",label="S o/w",ms=6)
if len(Jo_S_wo) != 0:
  plt.plot(Jo_S_wo,Jw_S_wo,"v",color="#BBF90F",label="S w/o",ms=
6)
if len(Jo_TF) != 0:
  plt.plot(Jo_TF,Jw_TF,"h",color="#FFD700",label="TF",ms=6)
if len(Jo_VFD_ow) != 0:
  plt.plot(Jo_VFD_ow,Jw_VFD_ow,"p",color="#FF00FF",label="VFD o/
w",ms=6)
if len(Jo_VFD_wo) != 0:
  plt.plot(Jo_VFD_wo,Jw_VFD_wo,"p",color="#FF4500",label="VFD w/
o",ms=6)
plt.xlabel("Jo [m/s]",fontsize=14,fontweight="bold")
plt.ylabel("Jw [m/s]",fontsize=14,fontweight="bold")
plt.xscale("log")
plt.yscale("log")
plt.legend(bbox_to_anchor=(0., 1.04, 1., 0.06), loc="lower left"
, ncol=5, mode="expand", borderaxespad=0., fontsize=12)
plt.show()
```