ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

EVALUACIÓN Y APLICACIÓN DE DIFERENTES ESTRUCTURAS DE REDES NEURONALES PARA GRAFOS IMPLEMENTACIÓN DE ENSAMBLES DE MODELOS GNN

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO MATEMÁTICO

ANDRÉS FRANCISCO ORTIZ NICOLA

andres.ortiz01@epn.edu.ec

DIRECTOR: MIGUEL ALFONSO FLORES SÁNCHEZ

miguel.flores@epn.edu.ec

DMQ, JULIO 2024

CERTIFICACIONES

Yo, ANDRÉS FRANCISCO ORTIZ NICOLA, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Andrés Francisco Ortiz Nicola

Certifico que el presente trabajo de integración curricular fue desarrollado por Andrés Francisco Ortiz Nicola, bajo mi supervisión.

Miguel Alfonso Flores Sánchez
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el(los) producto(s) resultante(s) del mismo, es(son) público(s) y estará(n) a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Andrés Francisco Ortiz Nicola

Miguel Alfonso Flores Sánchez

RESUMEN

Este estudio se centra en el desarrollo y la comparación de modelos de ensamble de redes neuronales de grafos (GNN) para la predicción del tráfico urbano, evaluando su capacidad para capturar dependencias espaciales y temporales. La predicción precisa del tráfico es fundamental para que gobiernos y empresas privadas puedan implementar medidas como la planificación de rutas, la expansión del transporte público y la mejora de servicios de navegación. Para este propósito, se utilizó una base de datos de tráfico ampliamente reconocida, en la cual se modeló la red de carreteras como un grafo. Cada nodo del grafo representa una intersección o segmento de carretera y contiene atributos como la velocidad, detectada a través de sensores instalados a lo largo de las vías.

Se implementaron varios modelos de GNN, incluyendo redes de convolución y redes de atención que, al ser combinadas con modelos que permitan el estudio de datos secuenciales como las redes recurrentes, permiten aprovechar de mejor manera las características no solo espaciales, sino también temporales de los datos. Además, se exploraron estrategias de ensamble de estos modelos para mejorar la precisión de las predicciones, empleando técnicas como la potenciación del gradiente y el promedio de predicciones de diferentes modelos base.

Los resultados obtenidos se evaluaron utilizando métricas como el error cuadrático medio (MSE) y el error absoluto medio (MAE), mostrando que los modelos de ensamble proporcionan mejoras significativas en el poder predictivo de los modelos de redes neuronales de grafos.

Palabras clave: Predicción del tráfico, redes neuronales de grafos (GNN), redes recurrentes, modelos de ensamble, error cuadrado medio, error absoluto medio.

ABSTRACT

This study focuses on the development and comparison of ensemble models of graph neural networks (GNN) for urban traffic prediction, evaluating their ability to capture spatial and temporal dependencies. Accurate traffic prediction is crucial for governments and private companies to implement measures such as route planning, public transport expansion, and navigation service improvements. For this purpose, a widely recognized traffic dataset was used, in which the road network was modeled as a graph. Each node in the graph represents an intersection or road segment and contains attributes such as speed, detected through sensors installed along the roads.

Several GNN models were implemented, including convolutional networks and attention networks, which, when combined with models that allow the study of sequential data, such as recurrent networks, better leverage both the spatial and temporal characteristics of the data. Additionally, ensemble strategies for these models were explored to improve prediction accuracy, employing techniques such as gradient boosting and averaging predictions from different base models.

The results were evaluated using metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE), showing that ensemble models provide significant improvements in the predictive power of graph neural network models.

Keywords: Traffic prediction, graph neural networks (GNN), recurrent networks, ensemble models, mean squared error, mean absolute error.

Índice general

Índice de figuras

Capítulo 1

Descripción del componente desarrollado

1.1. Motivación

En el campo del aprendizaje automático y el análisis de datos, las Redes Neuronales de Grafos (GNN) han demostrado ser una herramienta poderosa para abordar problemas complejos que involucran datos estructurados en forma de grafos. Sin embargo, cada arquitectura de GNN tiene sus propias fortalezas y limitaciones.

El desarrollo de un ensamble de modelos GNN tiene el potencial de mejorar significativamente la precisión y robustez de las predicciones al combinar las capacidades de múltiples arquitecturas de GNN explotando lo mejor de cada modelo y mitigando sus debilidades individuales. Al integrar las predicciones de varios modelos, un ensamble puede proporcionar una visión más completa y confiable de los datos, lo cual es crucial para aplicaciones críticas como la predicción de arcos, la clasificación de nodos, la detección de datos atípicos y muchas otras tareas que son muy comunes en este tipo de problemas.

1.2. Antecedentes

Tradicionalmente, los sistemas de predicción se dividen en dos categorías principales:

1. Métodos Estadísticos
2. Modelos de Aprendizaje Automático

En los métodos estadísticos tradicionales para la predicción de tráfico, se emplean modelos como el análisis de series temporales, la regresión lineal y los modelos ARIMA, que se basan en suposiciones sobre la distribución y la dependencia de los datos históricos. Estos enfoques tienden a ser efectivos para identificar patrones a corto plazo en datos bien estructurados y con pocas variables. Sin embargo, tienen limitaciones al enfrentar datos más complejos o de alta dimensionalidad, como los que se encuentran en sistemas de tráfico urbano. Por estas razones, se ha decidido utilizar estrategias de aprendizaje automático, específicamente redes neuronales, que son más adecuadas para capturar estas relaciones complejas y manejar grandes volúmenes de datos.

Las Redes Neuronales de Grafos (GNN) generan representaciones vectoriales para cada nodo en un grafo, las mismas que acumulan de manera iterativa las características de los nodos vecinos. Esta técnica permite capturar las relaciones espaciales entre los nodos, ya sea mediante conexiones directas o distancias cortas en términos de saltos o peso total de los arcos.

Al igual que otras redes neuronales, las GNN emplean una función de pérdida diferenciable y parametrizan la agregación de las características de los nodos vecinos. Esto facilita la extracción de características de alto nivel que permiten inferir el estado del grafo en diferentes momentos.

Las GNN tienen aplicaciones en diversos campos, como redes sociales, redes de interacción entre proteínas y grafos de conocimiento. Estas aplicaciones incluyen la clasificación de grafos, la inferencia de nodos o arcos dentro de una red existente y la regresión de características en nodos o arcos.

Wu et al. (2018) [33] proporcionan un análisis exhaustivo de las aplicaciones de las redes neuronales de grafos. Estos autores clasifican a las GNN en cuatro grupos:

- **Redes de grafos recurrentes:** Asumen que un nodo de un grafo comparte su información con el resto del grafo hasta llegar a una

estabilidad mediante arquitecturas recurrentes.

- **Redes de grafos convolucionales:** Formalizan la idea de la convolución en la estructura de grafos, agregando las características de cada nodo con las de sus vecinos y repitiendo este proceso en múltiples capas.
- **Autoencoders de grafos:** Son estructuras de redes que se utilizan codificar la información de grafos en un espacio latente y luego reconstruir los datos a partir de esta información codificada.
- **Redes de grafos espacio-temporales:** Buscan determinar patrones en grafos que presentan una evolución en el tiempo.

Además, discuten sobre el distinto tipo de grafos que se consideran para aplicar este tipo de redes neuronales y también mencionan que las predicciones que pueden realizarse con estos modelos pueden ser a nivel de nodos, para realizar clasificación o regresión de alguna variable; a nivel de arcos, para predecir la existencia o ausencia de un arco y su peso; y a nivel de grafos, para predecir las características de un grafo entero.

El desarrollo de la investigación en aprendizaje profundo durante los últimos años ha llevado a que más investigadores apliquen redes neuronales profundas para la predicción de tráfico [35]. Por ejemplo, en el trabajo de Huang et al. [8] se emplea una Red de Creencia Profunda (DBN) para la predicción del flujo de tráfico de manera no supervisada; por otro lado, Jia et al. [10] propusieron un modelo híbrido de DBN y un Perceptrón Multicapa (MLP) con el objetivo de predecir la velocidad. Aunque estos y varios enfoques más han obtenido buenos resultados, se centran principalmente en modelar una sola secuencia temporal, lo que limita su capacidad para considerar las dependencias espaciales en las redes de tráfico [37].

Para extraer la dependencia espacial de los datos de tráfico, las investigaciones introducen Redes Neuronales Convolucionales (CNN) en las tareas de predicción de tráfico. Ma et al. [16] propusieron un método que trata a una red de carreteras como imágenes dónde se pueden identificar secciones con tráfico y utilizar redes neuronales convolucionales para aprender las características espaciales. Wang et al. [32] implementaron

un mecanismo de corrección de errores en sus modelos de CNN para abordar los desafíos predictivos causados por eventos de tráfico imprevistos.

Sin embargo, las CNN tradicionales están limitadas a procesar únicamente estructuras espaciales en forma de rejilla, como las imágenes. Los datos frecuentemente se recopilan en estructuras más complejas, como los grafos, y es por esto que Bronstein et. al [1] proponen el Aprendizaje Profundo Geométrico (GDL). Las Redes Convolucionales de Grafos (GCN), presentadas por primera vez para una tarea semi-supervisada de clasificación [12] son uno de sus desarrollos que generalizan las CNN a dominios de grafos. Para problemas relacionados con datos de tráfico, las GCN son ampliamente adoptadas al tratar las redes de carreteras como grafos en los que existen relaciones espaciales importantes para explicar el comportamiento del tráfico [15, 36, 14].

Li et al. [15] propusieron un modelo híbrido basado en GCN que captura la dependencia espacial con caminatas aleatorias en la red de tráfico y la dependencia temporal con LSTM. Yu et al. [36] propusieron Redes Convolucionales Espacio-Temporales de Grafos (STGCN) que emplean estructuras convolucionales tanto para la componente espacial como para la temporal. Actualmente, los enfoques basados en GCN están entre las técnicas más avanzadas en la investigación de predicción de tráfico [14].

Diversos modelos se han desarrollado para la predicción del tráfico, abordando el problema desde diferentes perspectivas. Una línea emergente en esta área de investigación es la utilización de imágenes satelitales para obtener mediciones, en lugar de depender únicamente de datos tabulares obtenidos a partir de sensores. Sin embargo, en este trabajo nos centraremos en evaluar el rendimiento de algunos de estos modelos en la predicción del tráfico y en estudiar las posibles mejoras que se puedan obtener en las predicciones al utilizar ensambles de los mismos.

1.3. Objetivo general

Desarrollar un ensamble de modelos GNN que combine las predicciones de múltiples arquitecturas de GNN.

1.4. Objetivos específicos

1. Elegir las arquitecturas GNN que se utilizarán como modelos base en el ensamble.
2. Implementar y entrenar distintas técnicas de modelos de ensamble.
3. Evaluar y comparar los resultados obtenidos por el modelo de ensamble con los de los modelos individuales.

1.5. Alcance

La predicción precisa del tráfico es crucial para que tanto gobiernos como empresas privadas implementen políticas que promuevan un desarrollo urbano ordenado. Estas incluyen la optimización de rutas, expansión del transporte público y mejoras en la gestión del tráfico, elementos clave para los sistemas de transporte inteligente (ITS). Una correcta evaluación del tráfico facilita un control vehicular eficiente y mejora la operatividad de las redes viales.

Uno de los retos principales es prever dinámicas como el flujo o velocidad en autopistas, lo cual puede modelarse mediante grafos donde los sensores de tráfico se representan como nodos, y sus conexiones reflejan la proximidad física y funcional. Las características del tráfico, como velocidad y densidad, son atributos de estos nodos. Las GNN, al modelar estas redes, capturan efectivamente tanto la variabilidad temporal como las interacciones complejas del tráfico, las cuales son más direccionalmente significativas que meramente euclidianas.

Este estudio implementa y compara varios modelos ensamble de GNN para evaluar su eficacia en predecir el tráfico urbano, utilizando una base de datos reconocida. Esta comparativa busca establecer la aplicabilidad y eficiencia de las GNN en contextos urbanos complejos, considerando cómo manejan las dependencias espaciales y temporales en la predicción del tráfico.

Capítulo 2

Marco teórico

2.1. Red Neuronal

Como se detalla en el libro “Deep Learning” de Goodfellow, Bengio y Courville [5], las redes neuronales profundas son modelos compuestos por múltiples capas que, a su vez, están formadas por neuronas que reciben datos de entrada y calculan un valor de salida.

Las capas de una red se pueden categorizar en tres clases:

1. **Entrada**: Recibe los datos iniciales, que pueden ser imágenes, texto, señales u otros tipos de información. Estos datos se transforman en vectores de entrada y se transmiten a través de las conexiones de la red hasta las otras neuronas.

2. **Procesamiento**: El procesamiento se realiza de manera secuencial a través de las capas de la red. Las neuronas de cada capa ejecutan una operación matemática sobre los datos de salida de la capa anterior, conocida como función de activación. Esta función puede ser lineal o no lineal y determina si la neurona se activa y produce una salida. Durante el procesamiento, las conexiones entre neuronas tienen pesos asociados que indican la relevancia de cada conexión en el resultado final. Estos pesos se ajustan durante el entrenamiento de la red para mejorar su rendimiento en la tarea específica para la

que fue diseñada.

3. **Salida**: La información procesada en las capas ocultas se transmite a la capa de salida, donde se produce el resultado final de la red neuronal que puede ser una clasificación, una regresión u otro tipo de respuesta, dependiendo de la tarea que la red esté realizando. El valor obtenido en esta capa es fundamental para el entrenamiento de la red; pues, los pesos de las conexiones se ajustarán mediante algoritmos de aprendizaje, como el descenso del gradiente que requieren del cálculo de una función de pérdida, que calcula el error generado entre la salida de la red y el valor esperado.

En la figura 2.1 se puede ver el funcionamiento de una red neuronal

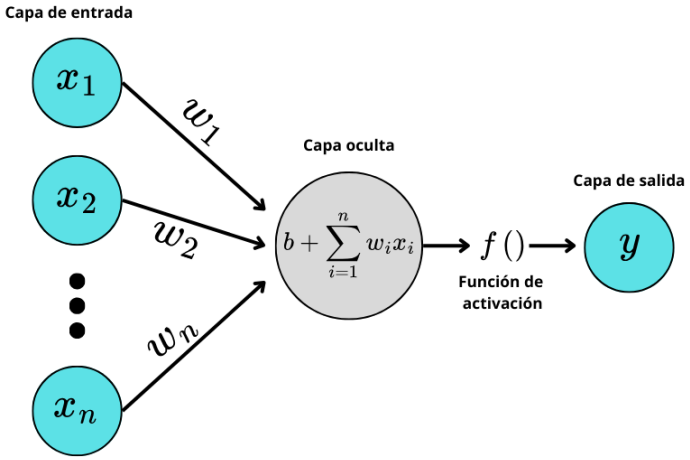


Figura 2.1: Ejemplo de red neuronal con una neurona en la capa oculta y una neurona en la capa de salida

Antes del entrenamiento de redes neuronales, es muy importante establecer los hiperparámetros de la red, estos son aquellos parámetros que no serán entrenados durante la optimización, sino que son fijados por la persona que implementa los modelos. Según Geron [6], en la arquitectura típica de las redes neuronales más sencillas, los hiperparámetros que se consideran son:

- **El número de capas**: Determina la profundidad de la red. Puede incluir una capa de entrada, varias capas ocultas y una capa de salida.

- **El número de neuronas por cada capa**: Este es el número de unidades en cada capa de la red neuronal.
- **Las funciones de activación de cada capa**: Las funciones de activación determinan la salida de una neurona dada una entrada. Ejemplos comunes incluyen la función sigmoide, ReLU y tanh.
- **La función de pérdida**: La función de pérdida cuantifica la diferencia entre la salida de la red y el valor esperado.
- **La tasa de aprendizaje**: La tasa de aprendizaje es un hiperparámetro que controla el ajuste de los pesos de la red con respecto al gradiente de la función de pérdida. Determina qué tan grandes son los pasos de ajuste en cada iteración.
- **El número de épocas**: El número de épocas se refiere al número de veces que el algoritmo de entrenamiento verá el conjunto completo de datos. Una época completa significa que todos los datos han sido usados una vez para actualizar los pesos. Para modelos complejos, un número de épocas muy alto puede significar el sobreajuste a los datos de entrenamiento por lo que se suele entrenar estos modelos hasta el punto en el que el error en el conjunto de prueba deje de disminuir.
- **El tamaño del lote**: El tamaño del lote (también conocido como batch) es el número de muestras que se procesan antes de actualizar los pesos de la red. El uso de lotes puede acelerar el entrenamiento y estabilizar la actualización de los pesos.
- **El optimizador**: Es el algoritmo que se usa para la actualización de los pesos de la red. Ejemplos de optimizadores pueden ser el descenso del gradiente estocástico o modificaciones del mismo como la propagación de raíz cuadrática media (RMSProp) o la estimación de momento adaptativo (ADAM) [11].

2.1.1. Red neuronal recurrente (RNN), LSTM y GRU

Las redes neuronales recurrentes (RNN) son una clase de redes neuronales artificiales diseñadas específicamente para trabajar con datos se-

cuenciales o series temporales [26].

A diferencia de las redes neuronales tradicionales, las RNN tienen la capacidad de recordar información de entradas anteriores, lo que les permite identificar patrones en el orden de los datos; mientras que las redes neuronales profundas tradicionales tratan las entradas y salidas de manera independiente. [26].

El entrenamiento de una red neuronal recurrente (RNN) es intensivo en tiempo y memoria debido a la necesidad de procesar cada paso temporal. Este proceso se simplifica "desenrollando" la red en tantas etapas como pasos temporales tenga la secuencia de entrenamiento, tratándola como una red de propagación hacia delante (feed-forward) donde los inputs para cada momento en el tiempo contendrán los valores en los instantes anteriores. Cada una de estas capas usa los mismos pesos para acelerar el entrenamiento [28].

Sin embargo, al aumentar la longitud de la secuencia temporal, se incrementa el número de operaciones, lo que puede llevar al problema de desvanecimiento del gradiente. Para solucionar esto, se incorporan capas de tipo LSTM o GRU [3], que ofrecen distintos mecanismos para evitar que los gradientes se diluyan a lo largo de las capas [28].

Red Neuronal Recurrente Long Short Term Memory

Las LSTM (Long Short-Term Memory) son un tipo especializado de redes neuronales recurrentes diseñadas para manejar dependencias a largo plazo en secuencias de datos sin sufrir del problema del desvanecimiento del gradiente, fueron propuestas inicialmente por Hochreiter y Schmidhuber [7] pero han sufrido algunos cambios a lo largo del tiempo. La implementación que nosotros utilizamos se basa en el trabajo de Sak et al. [25]. Los elementos clave para su funcionamiento se describe a continuación:

- **Celda de memoria:** Almacena información a lo largo del tiempo.
- **Compuertas:**
    - Compuerta de entrada: Controla la nueva información se añade.

- Compuerta de olvido: Decide qué información se elimina.
- Compuerta de salida: Determina qué información de la celda se utiliza como salida.

- **Flujo de información:** Las compuertas permiten que la red aprenda a mantener información relevante por largos períodos y a descartar información irrelevante.

El proceso de transferencia de datos en LSTM es similar al de las RNN estándar. Sin embargo, la forma en que la información se propaga es distinta. Cuando la información pasa por la LSTM, una operación decide qué información procesar y cuál descartar. Este mecanismo principal involucra células y puertas. El estado celular actúa como una vía para transferir información, funcionando como una memoria [7].

Como se describe en la documentación de la implementación de una capa LSTM en el paquete Pytorch [21], para cada elemento de la secuencia de entrada, se calculan las siguientes funciones:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

donde h_t representa el estado oculto en el tiempo t, c_t es el estado de la celda en el tiempo t, x_t es la entrada en el tiempo t, y h_{t-1} es el estado oculto de la capa en el tiempo $t-1$ o el estado oculto inicial en el tiempo 0. Las variables i_t, f_t, g_t, o_t corresponden a las puertas de entrada, olvido, celda y salida, respectivamente. La función σ es la sigmoide y $\odot$ denota el producto elemento a elemento. Los términos W y b son las matrices de pesos y los vectores de sesgo asociados a cada paso. En una LSTM de múltiples capas, la entrada $x_t^{(l)}$ de la capa l-ésima ($l \geq 2$) corresponde al estado oculto $h_t^{(l-1)}$ de la capa anterior.

Los hiperparámetros que debemos considerar al crear este tipo de re-

des son muy similares a los de las redes tradicionales, pero se debe aumentar el tamaño de la secuencia que será la entrada de la red. Aunque esto no es completamente necesario de forma teórica, en la práctica se debe fijar este valor por motivos de eficiencia computacional [26].

Red de unidad recurrente cerrada

La variante de Red Neuronal Recurrente conocida como Gated Recurrent Unit (GRU), propuestas por [2] es similar a las redes LSTM en su objetivo de resolver los problemas de las RNN tradicionales. En lugar de utilizar el estado de la celda para regular la información, las GRU emplean estados ocultos y disponen de dos puertas en lugar de tres: una puerta de restablecimiento y una puerta de actualización. Estas puertas, al igual que en las LSTM, controlan la cantidad y el tipo de información que debe ser retenida, sin embargo la red GRU no tiene cedas de memoria separada [3].

Siguiendo la documentación de Pytorch para la implementación de esta red neuronal [20], para cada elemento de la secuencia de entrada se calculan las siguientes funciones:

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})) \\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{t-1}
\end{aligned}
$$

donde h_t representa el estado oculto en el tiempo t, x_t es la entrada en el tiempo t, h_{t-1} es el estado oculto de la capa en el tiempo $t-1$ o el estado oculto inicial en el tiempo 0, y r_t, z_t, n_t son las puertas de reinicio, actualización y nueva, respectivamente. La función σ denota la función sigmoide y $\odot$ representa el producto elemento a elemento.

2.1.2. Redes neuronales de grafos (GNN), GCN y GAT

Las redes neuronales de grafos (GNN, por sus siglas en inglés) son una clase de modelos de aprendizaje automático diseñados específicamente

para trabajar con datos que tienen una estructura de grafo. A diferencia de los datos tabulares o secuenciales, los datos de grafos consisten en nodos (o vértices) y enlaces (o aristas) que describen relaciones entre estos nodos. Ejemplos de estos datos incluyen redes sociales, sistemas de recomendación, mapas de carreteras y moléculas químicas, donde las interacciones entre entidades son tan importantes como las propias entidades.

Las GNN se han desarrollado para aprovechar esta estructura intrínseca de los datos, permitiendo la propagación de información a través de las conexiones del grafo. Esto se realiza mediante un proceso de agregación de mensajes y actualización de estados, en el cual cada nodo recopila información de sus nodos vecinos y actualiza su propia representación en función de esta información [22].

Red neuronal convolucional (CNN)

Las redes neuronales convolucionales son un tipo de red neuronal diseñada especialmente para procesar datos con estructura de cuadrícula, como imágenes. Las capas de convolución se caracterizan por aplicar filtros (kernels) a la entrada para detectar características locales, de modo que, en el caso de imágenes, cada píxel se actualiza con la información de los píxeles de su alrededor. Esta idea luego se generalizó con el fin de aplicar convoluciones a estructuras de grafos [13].

Red neuronal convolucional de grafos (GCN)

Este tipo de red es una variante más general de las redes neuronales convolucionales, diseñada para manejar datos estructurados ya no solo como imágenes, sino en forma de grafos [12].

Recordemos que un grafo está compuesto por nodos, aristas, características de los nodos y características de las aristas. Los componentes clave para un modelo GCN son:

- Matriz de adyacencia **A**: donde A_{ij} representa la conexión o no entre los nodos i y j.

- Matriz de características **X**: donde X_i representa el vector de características del nodo i.

La operación de convolución en grafos se define en términos de los nodos y sus vecinos, y básicamente busca extraer y utilizar la información de los nodos conectados [12]. Los pasos a seguir son:

1. **Normalización de la matriz de adyacencia:** Este paso es crucial para estabilizar el entrenamiento y mantener la escala de las características.

   $$\hat{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

   donde $\hat{A}$ es la matriz de adyacencia normalizada y D es una matriz diagonal donde $D_{ii} = \sum_j A_{ij}$.

2. **Agregación de vecinos:** Se agrega a cada nodo la información de sus vecinos utilizando la matriz de características:

   $$H^{(l+1)} = \sigma\left((\hat{A} + I)H^{(l)}W^{(l)}\right)$$

   donde $H^{(l)}$ es la matriz de características en la capa l (con $H^{(0)} = X$), $W^{(l)}$ son los pesos de la capa l, σ es una función de activación y $\hat{A}+I$ es la matriz de adyacencia con la identidad añadida para incluir las características del propio nodo.

Una vez que la información del grafo ha sido modificada por la GNN, lo más usual es aplicar una capa de salida para obtener predicciones a partir de este grafo transformado [12].

En este caso, nuevamente, los hiperparámetros son similares a los que se utilizan para las redes tradicionales; sin embargo, en este caso vale la pena recalcar que al usar solo una capa GCN en una red neuronal, cada nodo se actualizará con la información de sus vecinos próximos, al usar dos capas se utilizará la información también de los vecinos de sus vecinos y así sucesivamente.

Red neuronal de atención de grafos (GAT)

Presentadas por Petar Velickovic et al. [31], estas redes siguen una estrategia de auto-atención [30], para calcular los coeficientes que se usarán para realizar la convolución de grafos. A continuación se muestran los pasos a seguir para la actualización de las características al pasar por esta red:

1. Primero introducimos la capa GAT. La entrada de esta capa es un conjunto de características de nodos, digamos $h_l = \{h_1^l, h_2^l, \ldots, h_N^l\}$, $h_i^l \in \mathbb{R}^F$, donde N es el número de nodos y F es el número de características de cada nodo.

2. Para transformar las características de entrada, se utiliza una matriz de pesos compartida, $W \in \mathbb{R}^{F' \times F}$ y de esta manera proyectar la entra a otro espacio de características de dimensión F'.

3. El siguiente paso es definir un mecanismo de atención y de este modo calcular el coeficiente de atención de los nodos y sus vecinos:

$$e_{ij} = a(W\vec{h_i}, W\vec{h_j}), \quad a : \mathbb{R}^{2F'} \to \mathbb{R},$$

   donde $a(\cdot, \cdot)$ es el mecanismo de autoatención, e_{ij} es el coeficiente de atención calculado. Como observación solo se toman en cuenta los vecinos directos del nodo.[31]

4. A estos resultados se les suele aplicar una función de activación LeakyRelu y; finalmente, se utiliza una función softmax para normalizar los coeficientes de atención en una forma fácilmente comparable y se obtiene el coeficiente de atención normalizado.

$$\alpha_{ij} = softmax(LeakyReLU(e_{ij})).$$

   Con estos coeficientes, se realiza la convolución para la actualización de características:

$$h_i^{l+1} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij} W^l h_j^l\right)$$

donde $N(i)$ corresponde al conjunto de nodos adyacentes que son inmediatos del nodo i.

En la Figura 2.2 se puede ver el diagrama de un mecanismo de atención empleado al nodo i con el nodo j.

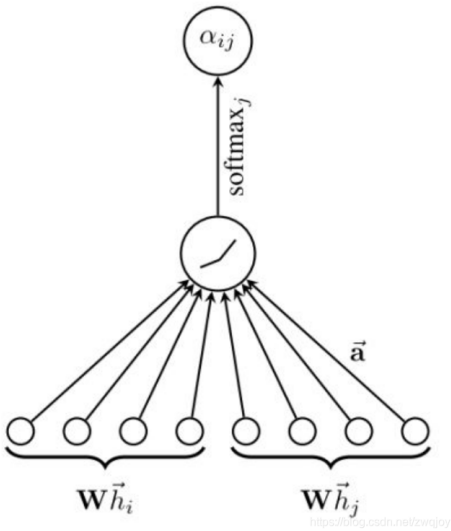


Figura 2.2: Mecanismo de atención, parametrizado por un vector de pesos $a \in \mathbb{R}^{2F'}$, aplicando la función de activación LeakyReLU[31]

Este procedimiento se puede generalizar aún más al usar múltiples "cabezas de atención" [30],[34], es decir, realizando el cálculo de múltiples coeficientes de atención independientes para luego agregar los resultados de cada cabeza mediante promedio o concatenación [31] como se puede observar en la figura 2.3. Aquí, diferentes estilos de flechas y colores denotan cálculos de atención independientes y las características agregadas de cada cabeza se concatenan o promedian para obtener $\vec{h_1'}$.

Evidentemente, en este tipo de redes también se debe elegir como hiperparámetros la cantidad de veces que se repite la capa, el número de cabezas a utilizar, y si la operación de agregación de las cabezas será de concatenación o de promedio (esto dependerá también de la dimensión que esperemos obtener al final de la capa).

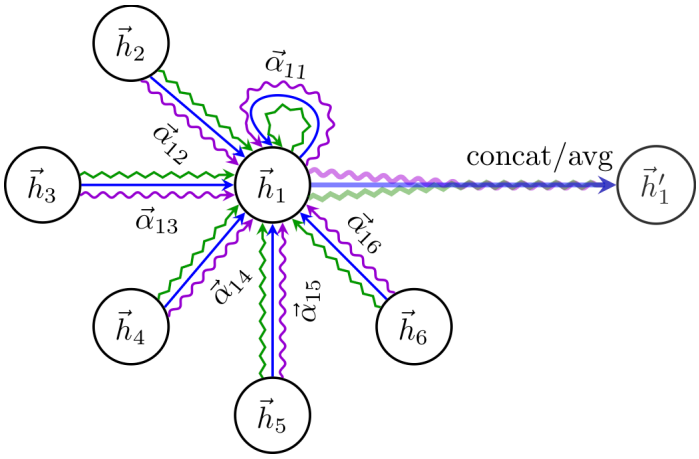


Figura 2.3: Ilustración de la atención multicabeza (con $K = 3$ cabezas) para el nodo 1 sobre sus vecinos [31]

2.2. Aprendizaje del tráfico usando GNN

En el campo de la predicción de tráfico, las redes espacio-temporales han surgido como una herramienta poderosa que combina la capacidad de modelar la estructura compleja de los datos de tráfico, representados como grafos, con la capacidad de capturar dependencias temporales a través de secuencias de datos. Estos modelos integran la flexibilidad de las redes neuronales convencionales con la capacidad de manejar datos dinámicos y relaciones espaciales inherentes a sistemas como el transporte urbano y la gestión de carreteras [15].

Las redes espacio-temporales utilizan una arquitectura híbrida (espacio-temporal) que incorpora elementos de redes de grafos y redes de procesamiento de secuencias. En este enfoque, los nodos del grafo representan ubicaciones geográficas, como intersecciones o segmentos de carreteras, y los bordes del grafo capturan las relaciones y conexiones entre estas ubicaciones. A través de capas de convolución o atención sobre el grafo, estas redes pueden capturar patrones espaciales y estructurales complejos que influyen en el flujo de tráfico; además, mediante mecanismos recurrentes se pueden procesar los datos secuenciales para modelas las fluctuaciones en el comportamiento del tráfico a lo largo del tiempo [24].

En los últimos años, se han desarrollado diversos enfoques para la predicción del tráfico utilizando modelos espacio-temporales. Uno de los

trabajos pioneros en el área es el desarrollado por Li et al. (2017), quienes introdujeron la Red de Convolución de Grafos Dirigidos (DCRNN, por sus siglas en inglés) [15]. Este modelo combina Redes Neuronales Recurrentes (RNN) con Convoluciones de Grafos para capturar tanto las dependencias temporales como las espaciales en los datos de tráfico. Otro de los modelos que se han implementado es el modelo Spatio-Temporal Graph Convolutional Network (ST-GCN) propuesto por Yu et al. (2017) [36].

La DCRNN y ST-GCN se ha destacado por su capacidad para manejar datos irregulares y no estructurados, comunes en redes de tráfico urbanas, mejorando significativamente la precisión de las predicciones en comparación con modelos tradicionales.

2.3. Modelos ensamble

El objetivo de los métodos de ensamble es combinar varios modelos para mejorar la precisión de predicción en problemas de aprendizaje que implican una variable objetivo numérica o categórica. Esta técnica se fundamenta en la premisa de que diferentes modelos pueden capturar aspectos diversos y complementarios de los datos, y al integrar sus predicciones, se puede obtener una estimación más robusta y precisa del valor objetivo [17].

2.3.1. Ensamble de promedio

La forma más sencilla de implementar un modelo de ensamble para una tarea de regresión es simplemente considerar las predicciones realizadas por los distintos modelos que forman parte del ensamble y promediarlas [6]. Para que esta tarea brinde mejores resultados que cada uno de los modelos individuales es necesario que los modelos que se usaron para el ensamble no presenten fuertes correlaciones, ya que de otra forma, todos brindarían resultados muy similares para datos de entrada parecidos y el promedio no sería mejor que las predicciones individuales.

2.3.2. Pasting y Bagging

El Pasting y el Bagging (abreviatura de *Bootstrap Aggregation*) son técnicas que se usan para obtener modelos más diversos; es decir, que tengan menos correlación. La forma en la que estos métodos funcionan es entrenando modelos iguales o, que se encuentran correlacionados, en subconjuntos de los datos distintos de modo que cada uno pueda aprender características distintas de los datos. En el caso del bagging, la selección de estos subconjuntos se la realiza tomando muestras con reemplazamiento mientras que para el pasting, estas muestras son sin reemplazamiento.

Una vez que estos modelos han sido entrenados en sus respectivos conjuntos de datos, sus predicciones son agregadas usando la moda en el caso de tareas de clasificación, o el promedio en caso de regresión [6].

2.3.3. Boosting

El boosting engloba distintas técnicas, pero la idea general es entrenar varios modelos de forma secuencial con el objetivo de que cada uno intente corregir los errores del modelo anterior.

Una de las estrategias más populares de boosting es conocida como "Gradient Boosting" o "Potenciación del gradiente", y consiste en entrenar un primer modelo para predecir los datos originales, y luego una serie de modelos que intentan predecir los errores residuales obtenidos por el modelo anterior. De esta manera, la predicción final del modelo de ensamble será la suma de todas las predicciones de estos modelos.

2.3.4. Stacking

La idea de esta estrategia es similar a la del ensamble por promedio, pero, en lugar de usar una función arbitraria para agregar las distintas predicciones (como el caso de la función promedio), se entrena un modelo que aprenda como combinar dichos resultados de la mejor forma, este modelo puede ser desde algo tan sencillo como una regresión lineal (que se encargaría de ponderar los resultados obtenidos de cada modelo),

hasta otra red neuronal [6].

2.4. Métricas de evaluación de Modelos de Regresión

La evaluación precisa del rendimiento es fundamental en el aprendizaje automático, especialmente para los modelos de regresión que predicen valores continuos. Este segmento explora meticulosamente tres métricas esenciales: el Error Cuadrático Medio (MSE), el Error Cuadrático Medio de la Raíz (RMSE), y el Error Absoluto Medio (MAE), cada una con sus propias características y aplicaciones específicas para medir la precisión de un modelo de regresión [22].

2.4.1. Error Cuadrático Medio (MSE)

El Error Cuadrático Medio (MSE) es una métrica de regresión que mide el promedio de los cuadrados de los errores. Es decir, calcula la media de las diferencias al cuadrado entre los valores reales y las predicciones. Esto implica una penalización más severa para los errores grandes, haciéndolo sensible a outliers en los datos [22].

$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Aquí, n es el número de observaciones, y_i es el valor observado y $\hat{y}_i$ es el valor predicho por el modelo.

2.4.2. Raíz del Error Cuadrático Medio (RMSE)

El RMSE es la raíz cuadrada del MSE, que ajusta las unidades de medida a la escala original de los datos, facilitando su interpretación. Proporciona una estimación más intuitiva de la magnitud de los errores de predicción [22].

$$\textbf{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Aquí, y_i y $\hat{y}_i$ mantienen sus significados como el valor observado y el valor predicho, respectivamente.

2.4.3. Error Absoluto Medio (MAE)

El MAE mide el promedio de las diferencias absolutas entre los valores observados y los predichos tratando a todos los errores por igual (a diferencia del MSE y RMSE) y brindando una métrica que se interpreta en la misma escala que los datos. [22].

$$\textbf{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Nuevamente, y_i representa el valor observado y $\hat{y}_i$ el valor predicho.

Cada una de estas métricas ofrece una visión valiosa sobre diferentes aspectos de los errores de predicción y se seleccionan en función de las características específicas de los datos y los requisitos de la aplicación.

Capítulo 3

Metodología

Este estudio investiga el desarrollo e implementación de modelos de ensamblaje basados en redes neuronales de grafos (GNN), integrando técnicas avanzadas para el análisis de datos con dependencias espaciales y temporales. La metodología aprovecha las características intrínsecas de los datos de tráfico, que presentan estas dependencias por su naturaleza.

Los pasos realizados para este trabajo son los siguientes:

1. **Selección de base de datos y preprocesamiento**
2. **Configuración de los modelos de redes neuronales de grafos**
3. **Implementación de los ensambles de modelos**
4. **Entrenamiento y evaluación**
5. **Análisis y visualización de resultados:**

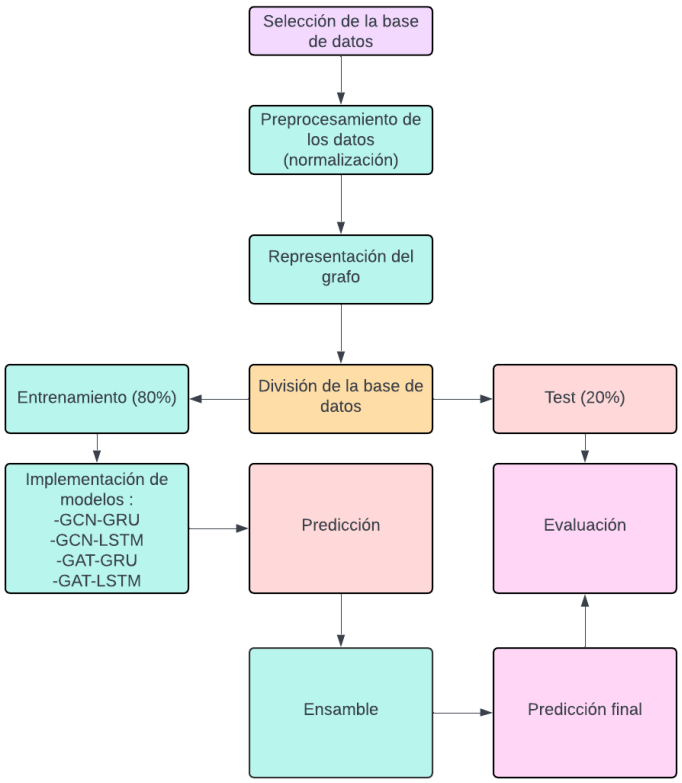


Figura 3.1: Diagrama de la metodología

Cada uno de estos pasos es detallado a profundidad a continuación.

3.1. Selección de la base de datos, preprocesamiento de los datos y representación de grafo

La base de datos que se seleccionó para la implementación de los modelos es la base conocida como METR-LA, la cual recopila información de 207 sensores de tráfico en las autopistas del condado de Los Ángeles, y cuenta con datos de 4 meses (marzo a junio de 2012). Las lecturas de velocidad del tráfico se encuentran agregadas en intervalos de 5 minutos, resultando en un total de 34,272 periodos de tiempo registrados y estas lecturas están normalizadas [15].

Se construyó un grafo de sensores basándose en las distancias de la red vial entre pares de sensores siguiendo la metodología de Li et al.

[15]. Los datos preprocesados se encuentran disponibles en la librería Pytorch-Geometric-Temporal [23].

Los datos fueron separados en un 80 % para entrenamiento y un 20 % para validación. Esta distribución se detalla en la tabla 3.1.

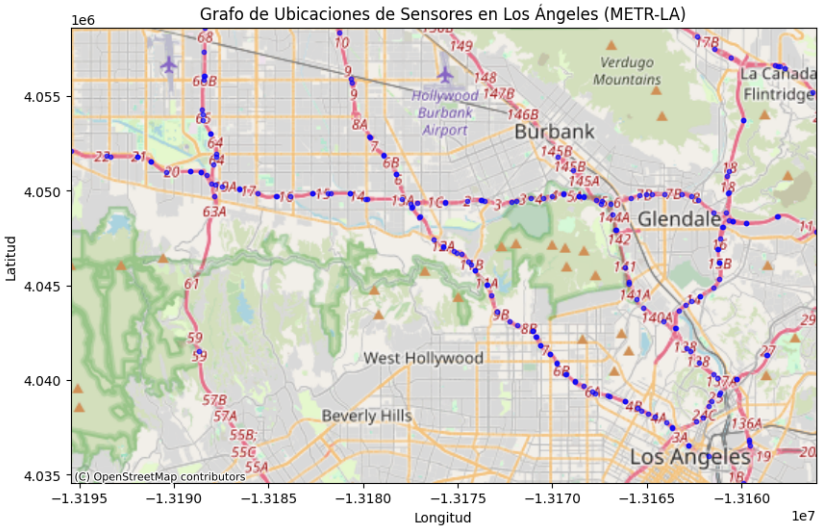


Figura 3.2: Ubicación de los sensores sobre el mapa de Los Ángeles

En este punto, se deben fijar dos hiperparámetros que son fundamentales para nuestro modelo: El largo de la secuencia de entrada y la cantidad de puntos en el futuro que se quieren predecir. Para este trabajo, se optó por fijar la secuencia de entrada en 12, y predecir solo la siguiente observación en el tiempo.

Al utilizar los datos de la librería Pytorch-Geometric-Temporal, se descargan varios objetos que se describen a continuación:

- Matriz de adyacencia (A): Una única matriz que determina la estructura del grafo; es decir, nos indica las conexiones entre los distintos sensores de tráfico.
- Características (X): Las características de los datos de tráfico que serán las entradas para nuestros modelos. En este caso, estas características contienen valores de la velocidad registrada en cada uno de los 207 sensores de velocidad a lo largo de 12 puntos en el tiempo.

- Objetivos (y): Son los valores objetivo para nuestro modelo. En este caso, son las mediciones en los 207 sensores para el siguiente instante en el tiempo.

Tipo	Porcentaje	Nº
Entrenamiento	80 %	27408
Validación	20 %	6852
Total	100 %	34260

Cuadro 3.1: División de la base de datos.

3.2. Implementación de modelos de Redes Neuronales de Grafos

Para el tratamiento de la componente espacial de los datos se seleccionaron los modelos de grafos "Graph Attention Networks (GAT)" [31] y "Graph Convoultional Networks (GCN)" [12]. Por otro lado, para la componente temporal se eligieron los modelos "Long Short Term Memory (LSTM)" [7] y "Gated Recurrent Unit (GRU)" [2].

De esta manera, se obtuvieron cuatro modelos base que resultan de las combinaciones de modelos:

Espacial	Temporal	Paper
GCN	GRU	T-GCN: A Temporal Graph ConvolutionalNetwork for Traffic Prediction [38]
GCN	LSTM	Long-Term Recurrent Convolutional Network (LRCN) [29]
GAT	GRU	Modificación de: Spatial-Temporal Graph Attention Networks: A Deep Learning Approach for Traffic Forecasting [37]
GAT	LSTM	Modificación de: Spatial-Temporal Graph Attention Networks: A Deep Learning Approach for Traffic Forecasting [37]

Cuadro 3.2: Modelos implementados

Cada uno de estos modelos sigue la misma estructura: Cada uno de los grafos de la secuencia es modificado por la capa de grafos correspondiente (GCN o GAT), y luego estos grafos con las características actualizadas pasan a la capa secuencial (LSTM o GRU). Finalmente, los resultados pasan por una capa lineal que resultará en la predicción final.

El diagrama base de los modelos implementados se puede ver en la Figura 3.3. En este gráfico los elementos $X_i, i = 1, ..., n$ representan los grafos de entrada, los $W_i, i = 1, ..., n$, los grafos transformados por la red de grafos, Z_{n+1} es el grafo predicho por la red recurrente y $\hat{Y}_{n+1}$ es la predicción final.

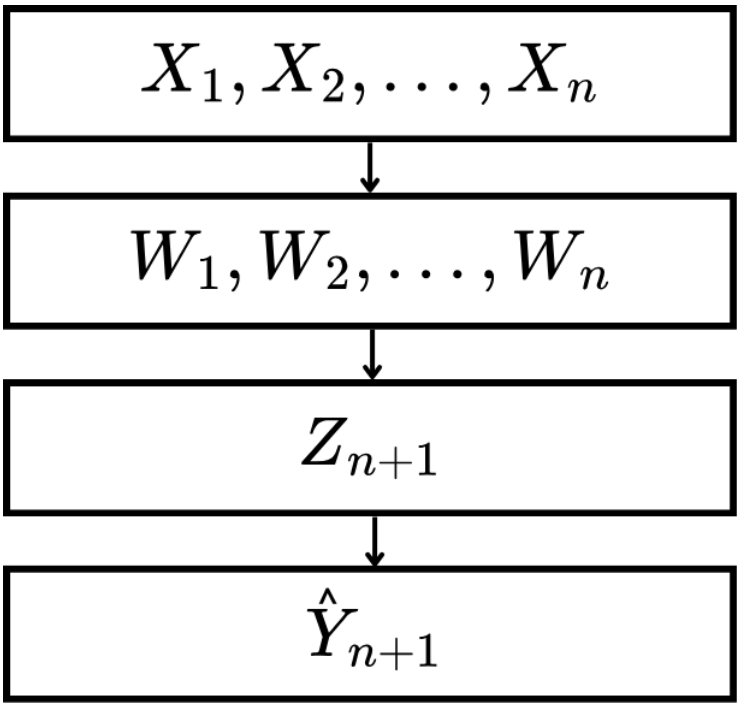


Figura 3.3: Diagrama de los modelos implementados.

3.2.1. Herramientas y Bibliotecas Utilizadas

Para todos los entrenamientos de los modelos, así como la implementación de los ensambles se utilizó la plataforma "Google Colab", que brinda un entorno alojado para ejecutar código de Python [19] y ofrece acceso gratuito a GPUs que son de mucha utilidad para entrenar modelos de redes neuronales. En esta plataforma, la versión gratuita cuenta con 12 GB de RAM, acceso a un GPU Tesla T4 con cerca de 15 GB de memoria.

Los modelos fueron implementados en la bilbioteca Pytorch [18], que brinda muchas herramientas para el manejo de datos, la creación de capas como LSTM y GRU, etc. También se hizo uso del paquete Pytorch-Geometric [4] para la implementación de las capas de grafos.

Para uno de los métodos de Stacking se usó la librería statsmodels [27] para realizar un modelo de regresión lineal. Finalmente, para la visualización de resultados se utilizó la librería Matplotlib [9].

3.2.2. Descripción de los modelos espacio-temporales

Los modelos híbridos que se exploran combinan técnicas avanzadas de procesamiento gráfico y temporal para predecir la velocidad del tráfico en cada nodo. Estos modelos comparten una serie de componentes comunes estructurados de la siguiente manera:

- **Entrada:** El grafo, con valores de la velocidad de cada nodo sirve como la única entrada para los modelos.
- **Capa de atención (solo para GAT):** Esta capa ajusta la influencia de las conexiones entre los nodos, permitiendo un modelado más fino de las relaciones espaciales. Esta capa se implementó con una sola cabeza de atención.
- **Transformación inicial:** Dependiendo del modelo, solo una capa Convolucional de Grafos (GCN) o una capa de Atención de Grafos (GAT) procesa la velocidad, transformándola en una representación con una dimensión oculta. En este caso, esta dimensión oculta tiene un tamaño de 2; es decir, se puede interpretar como si después de la transformación, cada nodo tiene 2 variables en lugar de solo una.
- **Unidad Recurrente:** Una unidad GRU o LSTM procesa las secuencias de datos temporales, capturando patrones a lo largo del tiempo.
- **Capas recurrentes:** Dos capas se utilizan para manejar la complejidad temporal y aprender dependencias a largo plazo. En estas capas el tamaño de la dimensión oculta se vuelve a duplicar, por lo que se puede interpretar como si cada nodo tiene ahora 4 variables que lo caracterizan.

- **Salida:** A partir de la salida de la capa secuencial (de tamaño 4 x 207; pues cada nodo tendrá 4 características), el modelo genera una salida de tamaño 207 a partir de una capa completamente conectada, donde cada valor representa la velocidad registrada en cada nodo en el siguiente instante de tiempo.

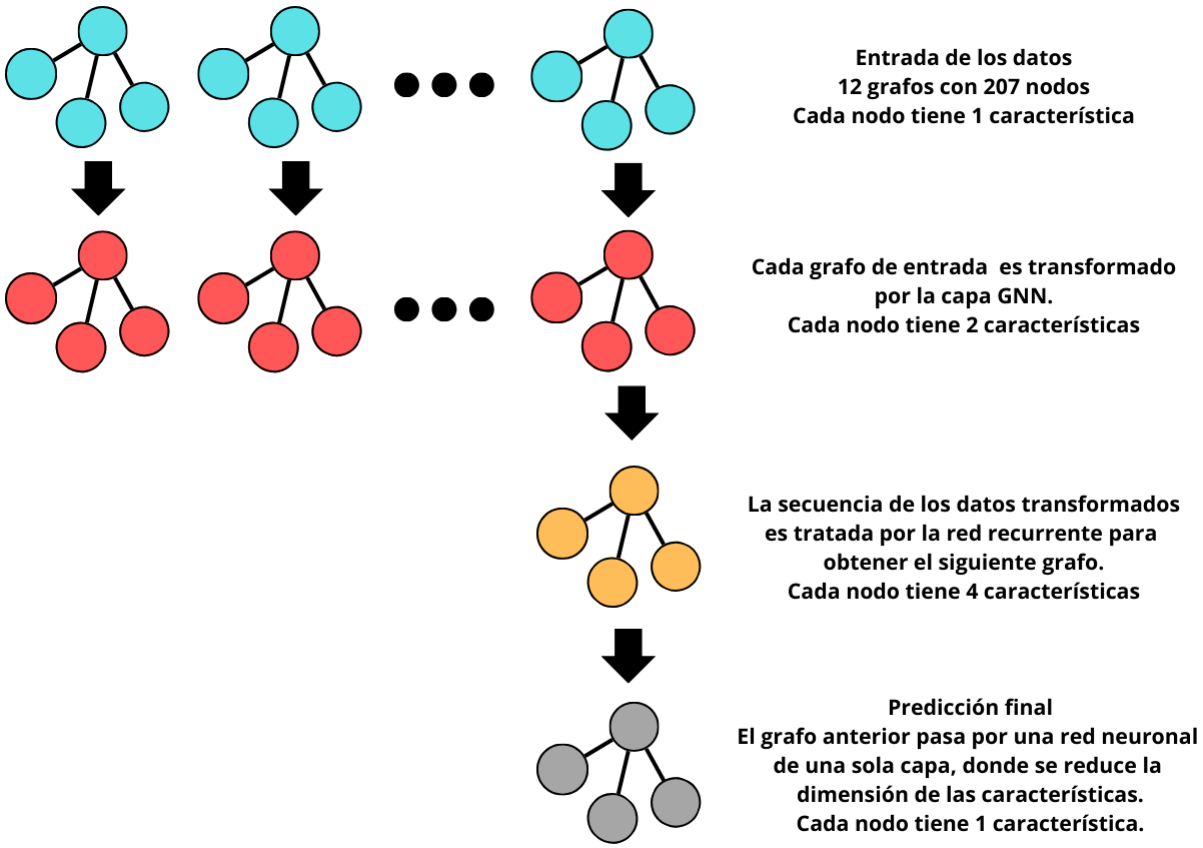


Figura 3.4: Esquema general de los modelos híbridos implementados.

3.3. Metodología de Modelos Ensamble

En este estudio, se implementaron modelos ensamble utilizando combinaciones de modelos espacio-temporales.

Los resultados de los modelos y configuraciones de ensamble fueron analizados y comparados para identificar las configuraciones más efectivas. Las visualizaciones ayudaron a interpretar el comportamiento de los modelos y a entender cómo las técnicas de ensamble contribuyen a mejorar las predicciones de tráfico.

A continuación, se describen las combinaciones específicas y la metodología utilizada para su implementación.

3.3.1. Modelos Individuales

Inicialmente, se implementaron y se entrenaron modelos esapcio temporales individuales, donde cada uno consta de una combinación de una red para capturar características espaciales —una Red Convolucional de Grafos (GCN) o una Red de Atención de Grafos (GAT)— y una unidad para analizar características temporales (una Unidad Recurrente LSTM o GRU). Estos modelos individuales incluyen:

- GAT + LSTM
- GCN + LSTM
- GAT + GRU
- GCN + GRU

Cada uno de estos modelos fue entrenado utilizando el optimizador ADAM [11], con una tasa de aprendizaje de $0{,}0001$. El tamaño de batch que se utilizó fue de 256. Además, para evitar el sobreajuste de los modelos a los datos de entrenamiento, cada modelo se entrenó hasta el punto en que el error en el conjunto de prueba dejó de disminuir. Así, los modelos GAT + GRU y GCN + GRU fueron entrenados durante 16 épocas, GAT + LSTM fue entrenado durante 13 épocas y GCN + LSTM, durante 14 épocas.

3.3.2. Procedimiento de Cálculo de la Media

Para cada combinación de modelos, el cálculo de la media se realizó siguiendo estos pasos:

1. Ejecutar cada uno de los modelos base involucrados de manera independiente.
2. Seleccionar todas las posibles combinaciones de los cuatro modelos base.

3. Recoger las predicciones de la velocidad del tráfico de cada modelo para cada combinación.

4. Calcular el promedio de las predicciones de los modelos para obtener la predicción final del modelo ensamble; es decir, el resultado predicho para cada sensor, será el promedio de las predicciones de cada uno de los modelos que conforman el ensamble en dicho sensor.

Para este modelo, los valores $\hat{y}$ se calculan como $\frac{1}{n}\sum_{j=1}^{n} x_{ji}$ con $i = 1, 2, 3, 4$, y $n =$ número de nodos.

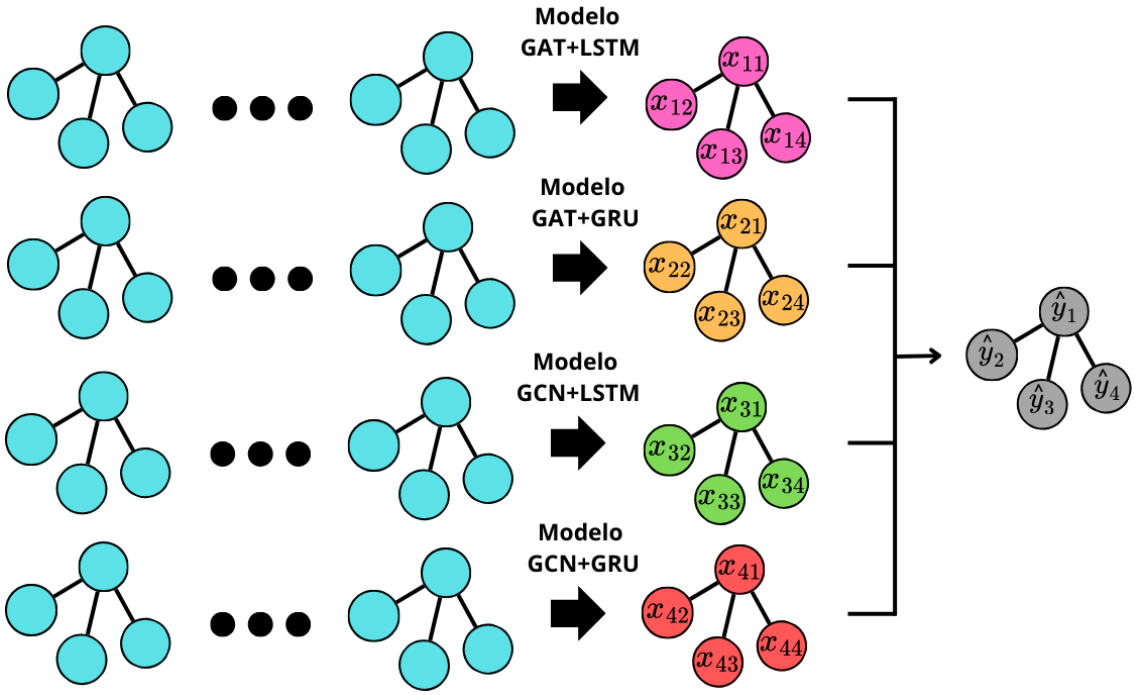


Figura 3.5: Esquema del modelo de ensamble del promedio de todos los modelos base.

3.3.3. Pasting

Con el motivo de reducir la correlación entre modelos, se aplicó la técnica de Pasting, que implica entrenar cada modelo en subconjuntos aleatorios del conjunto de datos de entrenamiento, seleccionados sin reemplazo. En este caso, el tamaño de los subconjuntos que se tomó fue del 66 % del total de la base de entrenamiento. Esto permite que cada modelo

aprenda de una porción distinta del conjunto total, intentando aumentar la diversidad de los modelos sin cambiar el tamaño del conjunto de entrenamiento. Una vez que los modelos han sido entrenados en sus respectivos conjuntos de datos, se lleva a cabo el mismo procedimiento que para el ensamble del promedio.

3.3.4. Bagging

Para realizar el bagging de los modelos, el procedimiento es similar al del pasting, con la única diferencia de que los subconjuntos que se toman son muestreados con reemplazo, lo que implica que una observación puede repetirse en el conjunto de datos en el que se entrena cada modelo. Para realizar este procedimiento también se tomaron subconjuntos con un tamaño igual al 66 % del tamaño del conjunto de entrenamiento.

En este esquema, la predicción final se calcula de la misma manera que para el modelo ensamble del promedio.

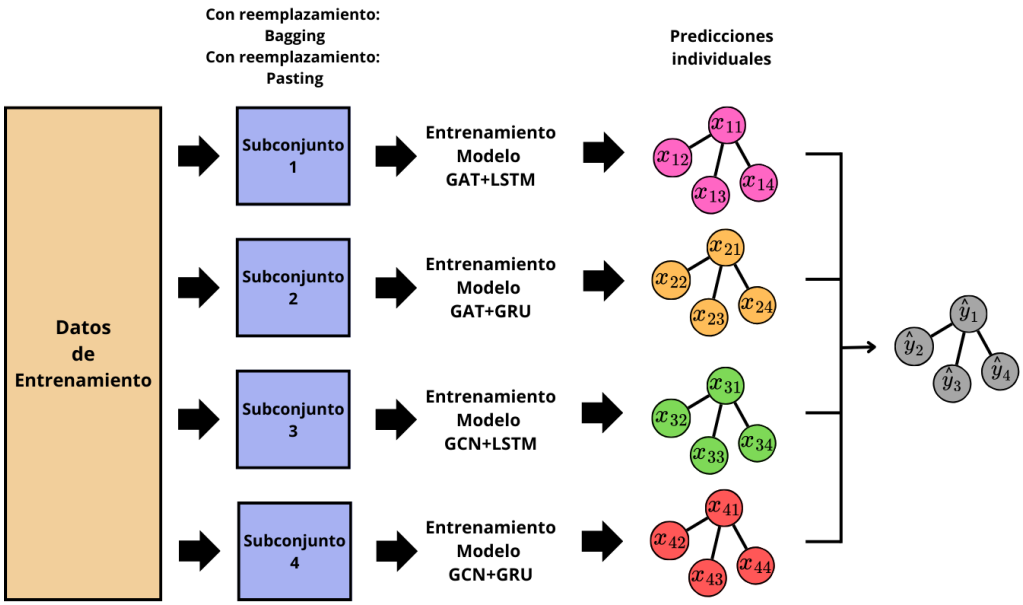


Figura 3.6: Esquema de los modelos de Bagging y Pasting que usan todos los modelos base.

3.3.5. Boosting

La estrategia de boosting que se implementó fue la de Gradient Boosting (Potenciación del gradiente). Así, se identificaron los errores de cada

uno de los modelos entrenados en el conjunto de entrenamiento. Posteriormente, se utilizaron cada uno de los 4 modelos base para intentar ajustar los errores obtenidos. De esta manera, como para cada uno de los 4 modelos originales, se utilizaron 4 modelos para ajustar sus errores, se terminó entrenando 16 modelos de boosting.

Para el entrenamiento de estos modelos también se utilizó el optimizador ADAM y una tasa de aprendizaje del $0{,}0001$. El tamaño del batch que se utilizó fue de 256. La cantidad de épocas por las que se entrenó cada uno de los 16 modelos que intentan predecir los errores de los modelos originales se encuentran en la tabla A.5.

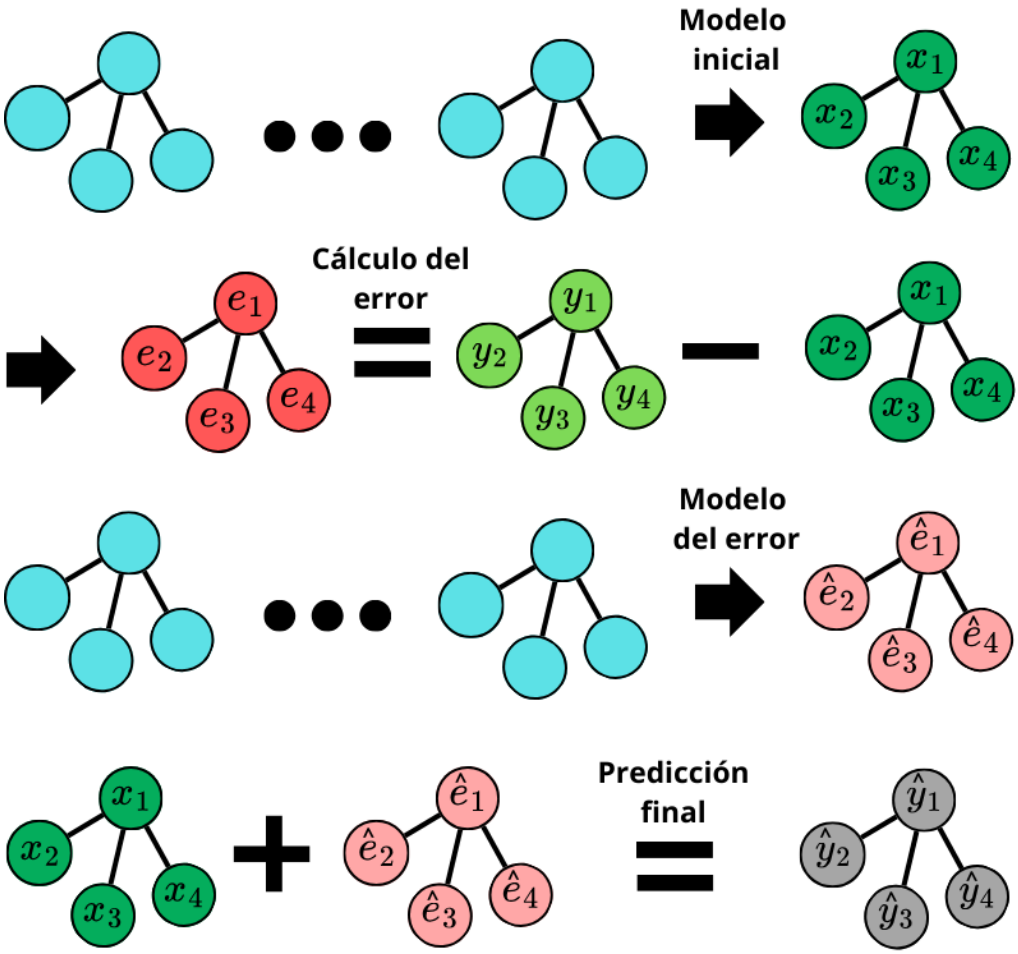


Figura 3.7: Esquema del modelo ensamble de boosting

3.3.6. Stacking

Se entrenaron en total dos modelos de stacking, los mismos que se describen a continuación:

- **Regresión lineal:** La primera forma en la que se combinaron las predicciones de cada uno de los 4 modelos fue mediante una regresión lineal. De esta manera, el valor predicho de la velocidad para cada uno de los sensores en el resultado final, estará dado por una combinación lineal de las predicciones de los 4 modelos base en el sensor respectivo; así, para este ensamble se asignarían distintos pesos a los resultados de cada modelo de forma similar a una votación ponderada.

  El esquema que tendría este modelo es similar al del modelo del promedio de resultados, con la única diferencia de que la predicción final ya no se calcula como $y_i = \frac{1}{n}\sum_{j=1}^{n} x_{ji}$ con $i = 1, 2, 3, 4$ y $n =$ número de nodos, sino que se calcula como $y_i = a_0 + \sum_{j=1}^{n} a_j x_{ji}$ con $i = 1, 2, 3, 4$, donde los coeficientes a_i para $i = 0, ..., 4$ son calculados mediante la regresión lineal.

- **Red neuronal:** El segundo modelo de stacking que se implementó es una red neuronal de 4 capas que recibe los 207 valores predichos por cada uno de los modelos base; es decir, un total de 828 valores, e intenta predecir los valores reales de los 207 sensores. Esta red neuronal se entrenó, nuevamente, con el optimizador ADAM y una tasa de aprendizaje de $0{,}0001$, con un tamaño de batch de 512 y durante 16 épocas. En este caso, se optó por incrementar el tamaño de batch para estabilizar el entrenamiento y que la función de pérdida disminuya de manera más uniforme.

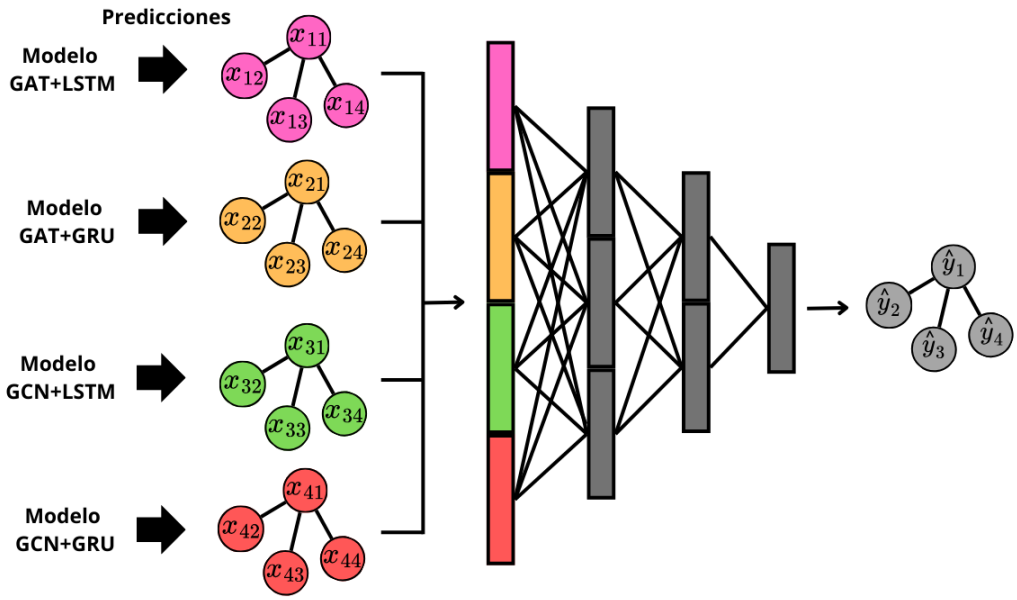


Figura 3.8: Esquema del modelo de stacking que usa una red neuronal

3.3.7. Entrenamiento y Evaluación de Modelos

Para todos los entrenamientos de redes neuronales se utilizó como función de pérdida al error cuadrático medio (MSE) con el objetivo de penalizar de mayor manera los errores de predicción más grandes.

Debido a las restricciones que presenta Google Colab respecto al uso de sus GPUs de uso gratuito, no se realizó una búsqueda de hiperparámetros óptimos mediante una búsqueda exhaustiva; pues esta implicaría el entrenamiento de los modelos en múltiples ocasiones para probar distintas combinaciones de hiperparámetros. De esta manera, se utilizó un enfoque empírico y basado en trabajos previos para elegir las configuraciones apropiadas de los modelos y, así, se logró una convergencia adecuada.

Capítulo 4

Resultados, conclusiones y recomendaciones

4.1. Resultados

En la presente sección se exponen los resultados obtenidos para los modelos espacio-temporales implementados: GCN-GRU, GCN-LSTM, GAT-GRU y GAT-LSTM. El código fuente del proyecto está disponible públicamente en GitHub: github.com/OrtizNicola/traffic-forecasting-GNN.

Para comparar los modelos de ensamblaje con los modelos originales y verificar si el rendimiento mejora al combinar los resultados de los modelos base, se presenta la siguiente tabla. En ella se muestran las métricas obtenidas por cada uno de los cuatro modelos base:

Modelo	**MSE**	**RMSE**	**MAE**
GAT + LSTM	91.76	9.57	5.01
GAT + GRU	81.16	9.00	5.14
GCN + LSTM	87.22	9.33	5.16
GCN + GRU	78.03	8.83	4.74

Cuadro 4.1: Resultados de los modelos base

A continuación, se presentarán los mejores resultados obtenidos por cada una de las estrategias de ensamble. Luego, se presenta una sección de discusión de resultados donde se realizará un análisis a más profundidad de lo obtenido. En anexos se puede encontrar una tabla completa con todos los modelos de ensamble que se implementaron.

4.1.1. Cálculo de la Media

Se presentan dos tablas, la primera muestra las métricas obtenidas por las tres mejores combinaciones de modelos respecto al error cuadrático medio (MSE) y a su raíz (RMSE); luego, se presenta la tabla con las tres mejores combinaciones respecto al error absoluto medio (MAE). Las tablas tienen como objetivo destacar el rendimiento de estas combinaciones de modelos en la tarea de predicción.

Promedio entre los modelos	**MSE**	**RMSE**	**MAE**
GAT-LSTM, GAT-GRU, GCN-GRU	76.67	8,76	4,64
GAT-GRU, GCN-GRU	76,99	8,77	4,80
GAT-LSTM, GAT-GRU, GCN-LSTM, GCN-GRU	77,55	8,81	4,65

Cuadro 4.2: Tabla de comparación de los 3 mejores modelos obtenidos respecto al RMSE y al MSE

Promedio entre los modelos	**MSE**	**RMSE**	**MAE**
GAT-LSTM, GCN-GRU	79,66	8,93	4,62
GAT-LSTM, GAT-GRU, GCN-GRU	76,67	8,76	4,64
GAT-LSTM, GCN-LSTM, GCN-GRU	79.87	8,94	4,64

Cuadro 4.3: Tabla de comparación de los 3 mejores modelos obtenidos respecto al MAE

La combinación de modelos con el valor más bajo del MSE y RMSE, "Promedio: GAT-LSTM, GAT-GRU, GCN-GRU", muestra un MSE de $76,67$, un RMSE de $8,76$ y un MAE de $4,64$.

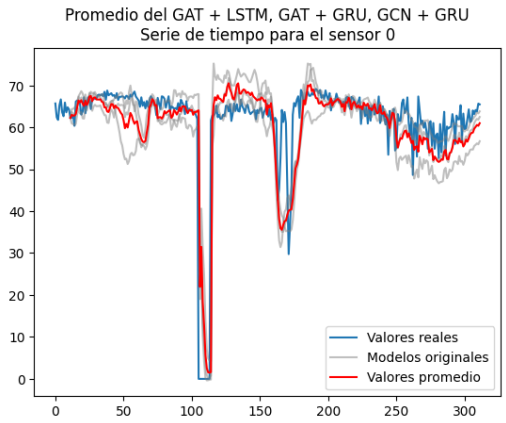
La combinación de modelos con el valor más bajo del MAE es el modelo "Promedio: GAT-LSTM, GCN-GRU", muestra un MSE de $79,66$, un RMSE de $8,93$ y un MAE de $4,62$.

En la Figura 4.1 se presentan dos gráficos de la serie de tiempo que se forma al graficar los valores de la velocidad para uno de los sensores que conforman al grafo, destacando los mejores modelos obtenidos según las métricas de error cuadrático medio (MSE) y error absoluto medio (MAE). Estos gráficos permiten visualizar y comparar cómo los modelos predicen los valores en función del tiempo en relación con los valores

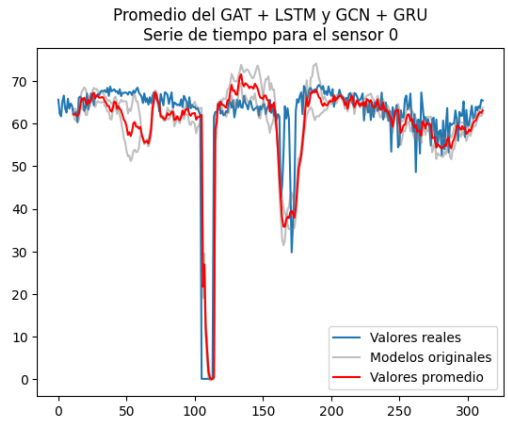
reales. Además, nos permite comparar visualmente la mejora que tienen estos modelos respecto a los modelos base con los que se hicieron los ensambles.

El gráfico (4.1a) muestra el modelo con el mejor MSE obtenido. En este caso, se promedian los modelos GAT + LSTM, GAT + GRU y GCN + GRU. La línea azul representa los valores reales del sensor 0, mientras que la línea roja muestra los valores predichos por el ensamble. La línea gris representa los valores predichos por los modelos que forman parte del ensamble. Este gráfico destaca cómo las predicciones se ajustan estrechamente a los valores reales, minimizando el error cuadrático medio.

Por otro lado, el gráfico (4.1b) presenta el modelo con el mejor MAE obtenido, promediando los modelos GAT + LSTM y GCN + GRU. Es importante notar que la diferencia principal entre estos modelos es que aquel con el mejor MSE, comete menos errores grandes que aquel que presenta un mejor MAE; pues, como se discutió en la presentación de estas métricas en el capítulo 2, el MAE valora a todos los errores por igual mientras que el MSE brinda más importancia a los errores mayores.



(a) Modelo con mejor MSE obtenido.



(b) Modelo con mejor MAE obtenido.

Figura 4.1: Serie de tiempo para el sensor 0 con los mejores modelos obtenidos según la métrica MSE y MAE.

En conjunto, estos gráficos proporcionan una comparación visual clara del desempeño de los modelos en términos de MSE y MAE, permitiendo identificar cuál combinación de modelos ofrece las predicciones más precisas y consistentes para el sensor 0. La representación gráfica facilita la comprensión de la efectividad de los modelos en la captura de las diná-

micas del tráfico urbano, destacando las ventajas de promediar múltiples modelos para mejorar la precisión de las predicciones.

4.1.2. Pasting

Se presenta dos tabla que compara los tres mejores modelos obtenidos mediante la técnica de pasting, una respecto al MSE y RMSE, y la otra respecto al MAE.

Promedio entre los modelos con Pasting	**MSE**	**RMSE**	**MAE**
GCN-LSTM, GCN-GRU	86.09	9.28	4.95
GCN-LSTM, GAT-GRU, GCN-GRU	87.10	9.33	4.99
GCN-LSTM, GAT-GRU	87.65	9.36	5.06

Cuadro 4.4: Tabla de comparación de los 3 mejores modelos obtenidos por pasting en base a los resultados del RMSE y MSE

Promedio entre los modelos con Pasting	**MSE**	**RMSE**	**MAE**
GAT-LSTM, GCN-LSTM, GCN-GRU	89,70	9.47	4.89
GAT-LSTM, GCN-LSTM, GAT-GRU, GCN-GRU	89.41	9.46	4.93
GCN-LSTM, GCN-GRU	86.09	9.28	4.95

Cuadro 4.5: Tabla de comparación de los 3 mejores modelos obtenidos por pasting en base a los resultados del MAE

La combinación de modelos con el valor más bajo del MSE y RMSE, “Pasting: GCN-LSTM, GCN-GRU”, muestra un MSE de 86.09, un RMSE de 9.28 y un MAE de 4.95.

La combinación de modelos con el valor más bajo del MAE es el modelo “Pasting: GAT-LSTM, GCN-GRU”, muestra un MSE de 89, un RMSE de 9.47 y un MAE de 4.95.

4.1.3. Bagging

Estos son los resultados que se obtuvieron para los mejores modelos de Bagging; nuevamente, la primera tabla muestra los mejores modelos respecto al RMSE y al MSE, y la segunda los mejores respecto al MAE.

Promedio entre los modelos con Bagging	**MSE**	**RMSE**	**MAE**
GAT-GRU, GCN-GRU	77.01	8.78	4.89
GAT-LSTM, GAT-GRU, GCN-GRU	78.77	8.88	4.79
GCN-LSTM, GAT-GRU, GCN-GRU	79.06	8.89	4.95

Cuadro 4.6: Tabla de comparación de los 3 mejores modelos obtenidos por bagging en base a los resultados del RMSE y MSE

Promedio entre los modelos con Bagging	**MSE**	**RMSE**	**MAE**
GAT-LSTM, GAT-GRU, GCN-GRU	78.77	8.88	4.79
GAT-LSTM, GAT-GRU	81.06	9.00	4.83
GAT-LSTM, GCN-LSTM, GAT-GRU, GCN-GRU	80.28	8.96	4.87

Cuadro 4.7: Tabla de comparación de los 3 mejores modelos obtenidos por bagging en base a los resultados del MAE

La combinación de modelos con el valor más bajo del MSE y RMSE, "Bagging: GAT-GRU, GCN-GRU", muestra un MSE de 77.01, un RMSE de 8.78 y un MAE de 4.89.

La combinación de modelos con el valor más bajo del MAE es el modelo "Bagging: GAT-LSTM, GAT-GRU, GCN-GRU", muestra un MSE de 78.77, un RMSE de 8.88 y un MAE de 4.79.

4.1.4. Boosting

A continuación, los resultados obtenidos por los tres mejores modelos que hacen uso de la técnica de "Gradient Boosting":

Modelos de Boosting	**MSE**	**RMSE**	**MAE**
Boosting de GCN-GRU con GCN-GRU	68.27	8.26	4.54
Boosting de GAT-GRU con GCN-GRU	71.55	8.46	4.77
Boosting de GCN-GRU con GAT-LSTM	73.83	8.59	4.78

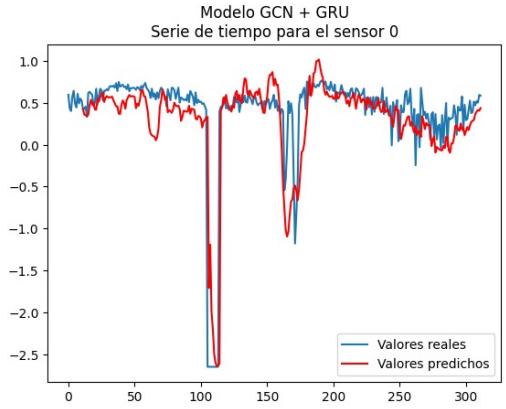
Cuadro 4.8: Tabla de comparación de los 3 mejores modelos obtenidos por boosting en base a los resultados del RMSE y MSE

Modelos de Boosting	**MSE**	**RMSE**	**MAE**
Boosting de GCN-GRU con GCN-GRU	68.27	8.26	4.54
Boosting de GAT-LSTM con GCN-GRU	74.81	8.65	4.66
Boosting de GCN-GRU con GCN-LSTM	74.80	8.65	4.74

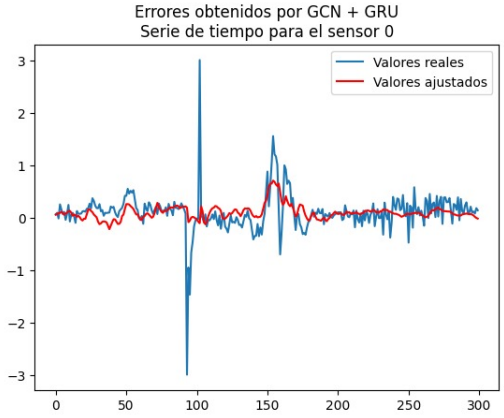
Cuadro 4.9: Tabla de comparación de los 3 mejores modelos obtenidos por bagging en base a los resultados del MAE

En este caso, el modelo que tiene el mejor MSE y RMSE es el mismo que aquel que presenta el menor MAE. Para el modelo de boosting del GCN-GRU consigo mismo, se ajustaron los errores del modelo base con el mismo modelo, y al sumar estos resultados se obtienen las predicciones finales que cuentan con un MSE de 68,27, un RMSE de 8,26 y un MAE de 4,54.

En la Figura 4.2 se muestran los valores ajustados por el modelo base, y los valores de los errores ajustados por segundo modelo. Finalmente, en la figura 4.3 se muestra la predicción final de este modelo, donde se aprecia que varios de los puntos los errores se han corregido.



(a) Serie de tiempo real vs Serie de tiempo predicha por el modelo GCN-GRU.



(b) Errores obtenidos por la serie de tiempo predicha por el modelo GCN-GRU vs Errores ajustados.

Figura 4.2: Series de tiempo sobre el sensor 0.

En la Figura 4.2, se observa la serie de tiempo real (línea azul) comparada con la serie de tiempo predicha (línea roja) utilizando el modelo combinado Boosting: GCN-GRU. La técnica de boosting toma las predicciones de varios modelos base y las combina para mejorar la precisión

global. Este proceso ayuda a reducir los errores de predicción al enfocarse en las áreas donde los modelos individuales tuvieron dificultades.

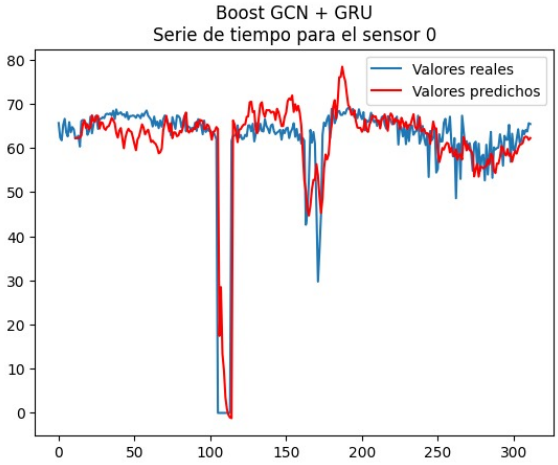


Figura 4.3: Serie de tiempo obtenida con el modelo de boosting GCN-GRU corregido por GCN-GRU sobre el sensor 0.

El gráfico muestra que el modelo GCN-GRU, al corregir sus errores por otro modelo GCN-GRU sigue de cerca la serie temporal real, con las líneas azul y roja superponiéndose en gran medida. Las desviaciones se han minimizado significativamente en comparación con los modelos individuales, indicando una mejora en la precisión gracias al proceso de boosting.

Es relevante señalar que se podría continuar entrenando modelos para corregir los errores de los modelos anteriores, y la cantidad de iteraciones de este proceso es otro hiperparámetro a considerar. Sin embargo, en este caso, se decidió realizar solo una iteración de este proceso. Esto se debe a que los errores de los modelos de boosting no mostraron una disminución significativa durante el entrenamiento, como se observa en la figura 4.4. Por lo tanto, continuar entrenando más modelos no mejoraría de manera significativa el rendimiento del modelo final.

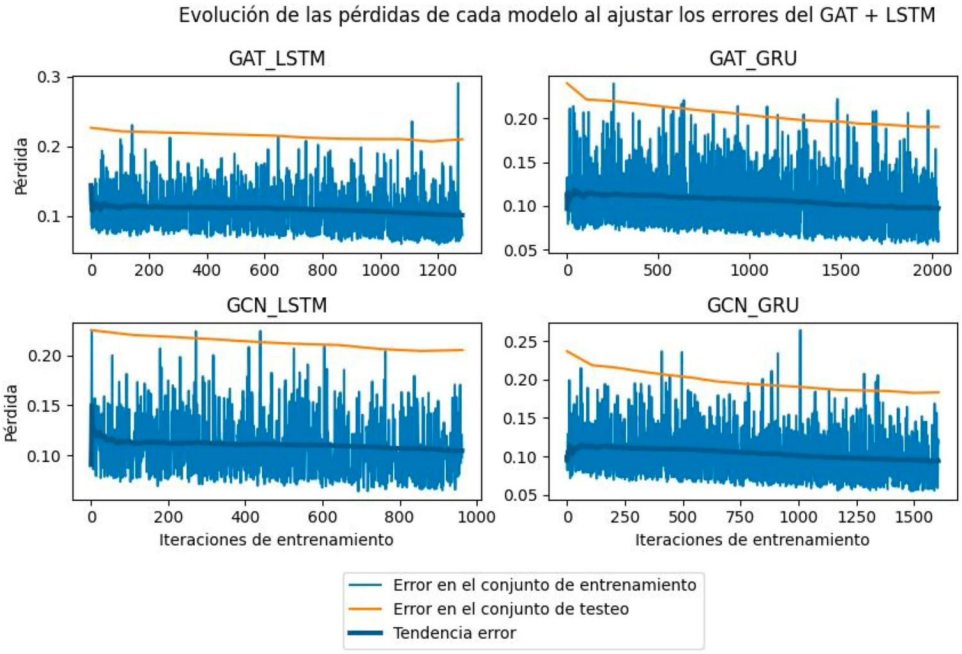


Figura 4.4: Evolución de los errores obtenidos por los modelos al intentar ajustar los errores del modelo GAT-LSTM.

4.1.5. Stacking

Para el modelo de regresión lineal, los coeficientes que se encontraron para realizar el ensamble se detallan en la siguiente tabla:

	Coef	**std err**	**t**	**P**$>\|t\|$
Constante	0.0056	0.000	41.963	0.000
Predicción de GAT-LSTM	0.0540	0.001	54.154	0.000
Predicción de GAT-GRU	0.3793	0.001	299.817	0.000
Predicción de GCN-LSTM	0.4035	0.001	331.844	0.000
Predicción de GCN-GRU	0.1708	0.001	122.599	0.000

Cuadro 4.10: Resumen de los resultados obtenidos para la regresión lineal

Estos resultados indican que, en el conjunto de entrenamiento, los resultados reales se pueden explicar mediante una combinación lineal de los resultados obtenidos por cada uno de los cuatro modelos base; pues, todos los coeficientes de la regresión son significativos.

En el Cuadro 4.11 se presenta la comparación de los resultados obtenidos para los dos modelos de stacking que se probaron:

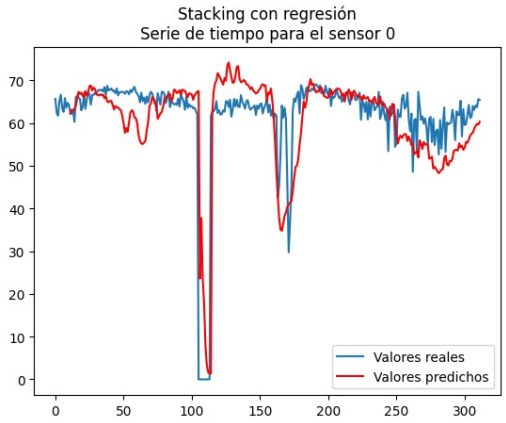
Modelos de Stacking	MSE	RMSE	MAE
Regresión lineal	78.11	8.84	4.74
Red neuronal	82.76	9.10	4.92

Cuadro 4.11: Tabla de comparación de los modelos obtenidos por stacking en base a los resultados de MSE, RMSE y MAE

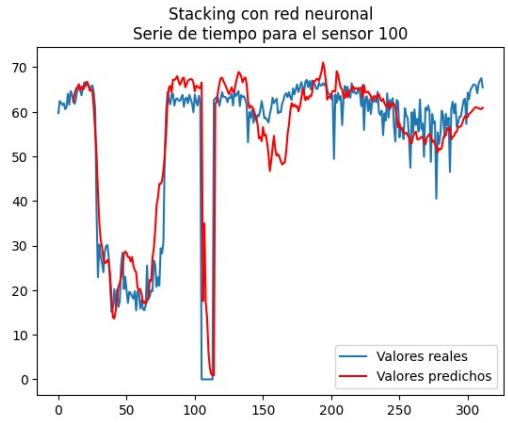
La primera fila de la tabla muestra los resultados del modelo de stacking con regresión lineal, con un MSE de 78.11, un RMSE de 8.84 y un MAE de 4.74.

La segunda fila presenta los resultados del modelo de stacking con red neuronal, con un MSE de 82.76, un RMSE de 9.10 y un MAE de 4.92.

En la Figura 4.5 se presentan dos gráficos que ofrecen una vista detallada del rendimiento de estos modelos en la predicción de la serie de tiempo para dos de los sensores.



(a) Serie de tiempo real vs Serie de tiempo predicha por el modelo stacking con regresión.



(b) Serie de tiempo real vs Serie de tiempo predicha por el modelo stacking con red neuronal.

Figura 4.5: Series de tiempo sobre el sensor 100.

4.2. Discusión de resultados

Las métricas obtenidas por el mejor de los modelos originales (GCN-GRU) fueron de 78.04 para el MSE, 8.83 para el RMSE y 4.74 para el MAE. Es importante destacar que mediante los ensambles se obtuvieron 14 modelos que obtuvieron mejores resultados en cuanto a los valores

del MSE y del RMSE. Por otro lado, se obtuvieron 8 modelos con mejores valores de MAE que el modelo GCN-GRU. Estos resultados son positivos ya que indican que mediante el ensamble de los modelos se puede obtener mejores resultados que en los modelos originales.

Es importante notar que la tabla completa con todas las métricas obtenidas por cada uno de los modelos se encuentra en anexos.

En cuanto a los modelos de promedios, hubo 5 modelos que obtuvieron un mejor MAE que el mejor de los modelos base que se muestran a continuación:

- GAT-LSTM + GCN-GRU
- GAT-LSTM + GAT-GRU + GCN-GRU
- GAT-LSTM + GCN-LSTM + GCN-GRU
- GAT-LSTM + GAT-GRU + GCN-LSTM + GCN-GRU
- GAT-LSTM + GAT-GRU + GCN-LSTM

Notamos que en todas estas combinaciones se encuentra presente el modelo GAT-LSTM y en todas, excepto una, aparece el modelo GCN-GRU. Esto tiene sentido; pues, estos 2 son los modelos que obtuvieron los mejores resultados en la métrica del MAE.

Por otro lado, siguiendo con los modelos de los promedios, 4 de estos modelos mejoraron al mejor modelo base en cuanto al MSE y RMSE. Estas combinaciones fueron:

- GAT-LSTM + GAT-GRU + GCN-GRU
- GAT-GRU + GCN-GRU
- GAT-LSTM + GAT-GRU + GCN-LSTM + GCN-GRU
- GAT-GRU + GCN-LSTM + GCN-GRU

En este caso, los modelos GCN-GRU y GAT-GRU aparecen en todas las combinaciones. Esto es entendible ya que ambos modelos base son los que obtuvieron los mejores resultados en el MSE y RMSE.

Respecto a los modelos de Pasting, los resultados obtenidos por todas las combinaciones que se probaron obtuvieron resultados peores que el mejor de los modelos base. Esto puede deberse al hecho de que, al no entrenar los modelos con tantos datos, ninguno pudo identificar patrones significativos en los datos de entrenamiento y, gracias a esto, sus resultados no fueron favorables.

Para los modelos de Bagging se obtuvieron resultados mejores que en el Pasting; sin embargo, respecto al RMSE y MSE solo una de las combinaciones que se probaron fue superior al modelo GCN-GRU; y respecto al MAE, ninguno de los modelos de bagging pudo obtener mejores resultados.

El modelo de bagging que venció GCN-GRU en MSE fue el del promedio de los modelos GAT-GRU y GCN-GRU y este obtuvo resultados prácticamente idénticos al del modelo del promedio de estos mismos modelos pero entrenados usando toda la base de datos.

De nuevo, los resultados desfavorables de este tipo de modelos podría deberse a que gracias a la reducida cantidad de datos de entrenamiento, fue difícil para las redes neuronales descubrir patrones significativos en los datos.

Los modelos de boosting fueron los que obtuvieron los mejores resultados de todos los ensambles. Así, 9 modelos en total obtuvieron mejores RMSE y MSE que el modelo GCN-GRU, mientras que 3 modelos obtuvieron mejores resultados en el MAE.

Estos son algunos de los modelos que obtuvieron mejores resultados en el MSE y RMSE:

- GCN-GRU potenciado con GCN-GRU
- GAT-GRU potenciado con GCN-GRU
- GCN-GRU potenciado con GAT-LSTM
- GCN-GRU potenciado con GCN-LSTM

Como se puede apreciar, el modelo GCN-GRU es una pieza fundamental para que este tipo de modelos presente un buen rendimiento y sucede lo mismo cuando hablamos de los modelos que obtuvieron un mejor MAE.

Es importante mencionar que el mejor modelo de todos los que se probaron es justamente uno de boosting; pues el modelo GCN-GRU potenciado por sí mismo, es decir, por otro GCN-GRU, obtuvo un RMSE de 8.26 (6.5 % mejor que el modelo GCN-GRU original) y un MAE de 4.54 (4.2 % mejor que el modelo original). El motivo de esta mejora se puede observar en la gráfica 4.2b, donde se aprecia que el GCN-GRU ofrece buenos resultados al predecir los errores residuales. Es importante considerar que, al realizar esta técnica, se debe tener cuidado con el sobreajuste a la base de datos de entrenamiento ya que si se predicen de manera perfecta los errores estos resultados podrían no generalizarse a la base de testeo y es por esta razón que muchos de los modelos de boosting se entrenaron por pocas épocas.

Por último, notamos que ninguno de los modelos de stacking pudo vencer al mejor modelo base en ninguna de las métricas consideradas. El motivo de esto es que durante el entrenamiento, no se pudo encontrar una relación que modele la unión de los 4 modelos base de una forma que sea generalizable al conjunto de prueba. Esta es la razón por la cual los modelos del promedio simple fueron más efectivos que los de stacking, ya que ensamblan los modelos base de una forma más generalizable.

En conjunto, este estudio confirma que las técnicas de ensamble son muy efectivas para mejorar la precisión y consistencia de las predicciones de tráfico urbano. Al combinar múltiples modelos, se logra capturar mejor las variaciones y patrones en los datos, proporcionando así soluciones más precisas y fiables. Estas conclusiones son esenciales para el desarrollo de sistemas de predicción de tráfico más eficientes, contribuyendo a una mejor gestión y planificación del tráfico urbano.

4.3. Conclusiones y recomendaciones

4.3.1. Conclusiones

- En este estudio, se han desarrollado varios modelos de ensamble de modelos de redes neuronales de grafos (GNN) que combinan las predicciones de múltiples arquitecturas distintas para la predicción del tráfico urbano y se obtuvo resultados superiores al de los modelos

individuales.

- Para la implementación de los ensambles, se utilizaron los modelos base: GAT-LSTM, GAT-GRU, GCN-LSTM y GCN-GRU, que demostraron tener buenas capacidades predictivas por separado. Además, se notó que el modelo más relevante para los ensambles fue el GCN-GRU, pues en aquellos modelos que este era utilizado se obtenían los mejores resultados.
- Los modelos de Stacking, Pasting y Bagging que se implementaron en este trabajo no brindaron resultados tan buenos como las estrategias del promedio simple y de boosting.
- El mejor modelo que se implementó fue el de boosting del GCN-GRU consigo mismo, que logró resultados mejores que el modelo GCN-GRU original en las métricas del MAE y RMSE en un 4.2% y un 6.5% respectivamente.

4.3.2. Recomendaciones

- Se recomienda la adopción de técnicas de ensambles para mejorar la predicción del tráfico urbano haciendo uso de redes neuronales de grafos.
- A pesar de que las arquitecturas de redes neuronales exploradas ofrecen buenos resultados, también se recomienda probar las técnicas de ensamble estudiadas en este trabajo con arquitecturas más diversas y complejas de redes espacio-temporales para intentar conseguir resultados aún mejores.
- Dada la efectividad de las técnicas de ensamble, se recomienda implementar estos modelos en sistemas de tráfico en tiempo real para mejorar la gestión y planificación del tráfico urbano. Esto podría contribuir a una mejor toma de decisiones y una mayor eficiencia en la gestión del tráfico.
- Se sugiere realizar investigaciones futuras para evaluar el rendimiento de los modelos de ensamble en diferentes contextos y con diferentes conjuntos de datos de tráfico.

- Para los modelos de Boosting, en este trabajo solo se crearon modelos para ajustar los errores de los modelos originales; sin embargo, es recomendable realizar este procedimiento de manera recursiva para crear varios modelos más que intenten ajustar los errores del modelo anterior y de esta manera mejorar los resultados.
- En los modelos de stacking, en concreto en el modelo de regresión, se ajustó una función que intenta mezclar las predicciones de los 4 modelos base de igual manera para cada sensor. Una mejor forma de abordar este problema podría ser ajustar distintos modelos para distintos sensores de modo que las predicciones sean más exactas.
- Debido a la complejidad computacional de los modelos implementados, no se ha podido explorar a profundidad las diferentes configuraciones de los hiperparámetros como el número de capas utilizadas, el número de cabezas de atención (en el caso del GAT) y el tamaño de las dimensiones ocultar; por esto, se recomienda utilizar unidades de cómputo con más capacidad de modo que se puedan implementar más modelos con más complejidad. Asimismo se recomienda probar los modelos al intentar predecir más puntos en el futuro en lugar de solo uno.

Capítulo A

Anexos

A.1. Scores obtenidos por cada modelo base

Modelos	Métricas		
	MSE	RMSE	MAE
GCN+GRU	78.037	8.834	4.742
GAT+GRU	81.161	9.009	5.149
GCN+LSTM	87.225	9.339	5.165
GAT+LSTM	91.766	9.579	5.015

Cuadro A.1: Tabla de comparación de modelos en base a los resultados de error MSE, RMSE y MAE

A.2. Scores obtenidos por los ensambles

Scores obtenidos por todos los modelos implementados.

A.2.1. Modelos Promedio

Modelos que conformaron cada promedio	**MSE**	**RMSE**	**MAE**
GAT+LSTM, GAT+GRU, GCN+GRU	76.667	8.756	4.641
GAT+GRU, GCN+GRU)	76.989	8.774	4.799
GAT+LSTM, GCN+LSTM, GAT+GRU, GCN+GRU	77.547	8.806	4.652
GCN+LSTM, GAT+GRU, GCN+GRU	77.646	8.811	4.758
GAT+LSTM, GAT+GRU	78.594	8.865	4.765
GAT+LSTM, GCN+LSTM, GAT+GRU	79.116	8.895	4.740
GCN+LSTM, GCN+GRU	79.588	8.921	4.755
GAT+LSTM, GCN+GRU	79.659	8.925	4.617
GCN+LSTM, GAT+GRU	79.731	8.929	4.924
GAT+LSTM, GCN+LSTM, GCN+GRU	79.873	8.937	4.644
GAT+LSTM, GCN+LSTM	84.723	9.205	4.822

Cuadro A.2: Resultados Promedio de Modelos

A.2.2. Modelos Pasting

Modelos que conformaron cada Pasting	**MSE**	**RMSE**	**MAE**
GCN+LSTM, GCN+GRU	86.092	9.278	4.951
GCN+LSTM, GAT+GRU, GCN+GRU	87.096	9.332	4.990
GCN+LSTM, GAT+GRU	87.651	9.362	5.061
GAT+LSTM, GCN+LSTM, GAT+GRU, GCN+GRU	89.411	9.455	4.932
GAT+LSTM, GCN+LSTM, GCN+GRU	89.695	9.470	4.893
GAT+GRU, GCN+GRU	90.791	9.528	5.148
GAT+LSTM, GCN+LSTM, GAT+GRU	90.967	9.537	4.973
GAT+LSTM, GAT+GRU, GCN+GRU	92.942	9.640	5.032
GAT+LSTM, GCN+LSTM	93.159	9.651	4.980
GAT+LSTM, GCN+GRU	95.031	9.748	5.007
GAT+LSTM, GAT+GRU	97.494	9.873	5.154

Cuadro A.3: Resultados de modelos Pasting

A.2.3. Modelos Bagging

Modelos que conformaron cada Bagging	**MSE**	**RMSE**	**MAE**
GAT+GRU, GCN+GRU	77.0065	8.775	4.885
GAT+LSTM, GAT+GRU, GCN+GRU	78.772	8.875	4.787
GCN+LSTM, GAT+GRU, GCN+GRU	79.064	8.891	4.953
GAT+LSTM, GCN+LSTM, GAT+GRU, GCN+GRU	80.278	8.959	4.870
GAT+LSTM, GAT+GRU	81.057	9.003	4.832
GCN+LSTM, GAT+GRU	81.295	9.016	5.056
GAT+LSTM, GCN+LSTM, GAT+GRU	82.223	9.067	4.916
GCN+LSTM, GCN+GRU	83.601	9.143	5.154
GAT+LSTM, GCN+GRU	83.603	9.143	4.923
GAT+LSTM, GCN+LSTM, GCN+GRU	84.347	9.184	4.996
GAT+LSTM, GCN+LSTM	89.931	9.483	5.153

Cuadro A.4: Resultados de modelos Bagging

A.2.4. Modelos Boosting

En la siguiente tabla se muestra también la cantidad de épocas por las que se entrenaron cada uno de los modelos que ajustaron los errores.

Modelo base - Modelo errores	MSE	RMSE	MAE	Époc.
GCN+GRU - GCN+GRU	68.267	8.262	4.542	16
GAT+GRU - GCN+GRU	71.545	8.458	4.774	6
GCN+GRU - GAT+LSTM	73.831	8.592	4.778	8
GCN+GRU - GCN+LSTM	74.795	8.648	4.736	7
GAT+LSTM - GCN+GRU	74.805	8.649	4.655	9
GCN+LSTM - GCN+GRU	75.391	8.683	4.964	8
GCN+GRU - GAT+GRU	76.609	8.753	4.831	6
GAT+LSTM - GAT+GRU	77.720	8.816	4.771	12
GAT+GRU - GCN+LSTM	77.936	8.828	4.973	3
GAT+GRU - GAT+LSTM	78.488	8.859	4.986	9
GCN+LSTM - GAT+GRU	79.297	8.905	5.015	15
GAT+GRU - GAT+GRU	79.959	8.942	5.053	21
GCN+LSTM - GAT+LSTM	83.425	9.134	5.204	9
GAT+LSTM - GCN+LSTM	83.955	9.163	5.065	19
GCN+LSTM - GCN+LSTM	84.876	9.213	5.127	22
GAT+LSTM - GAT+LSTM	85.685	9.257	5.099	15

Cuadro A.5: Resultados de modelos Boosting

A.2.5. Modelos de Stacking

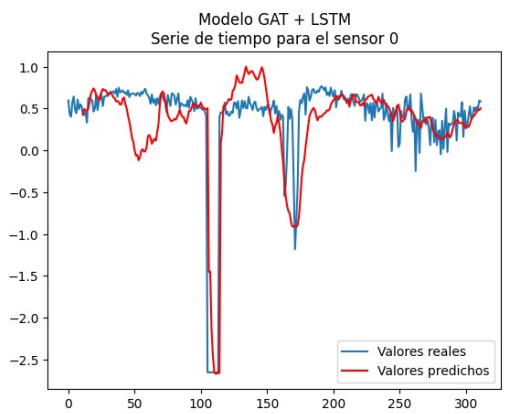
Modelos de stacking	MSE	RMSE	MAE
Stacking con Regresión Lineal	78.108	8.838	4.743
Stacking con Red Neuronal	82.755	9.097	4.915

Cuadro A.6: Resultados de modelos Stacking

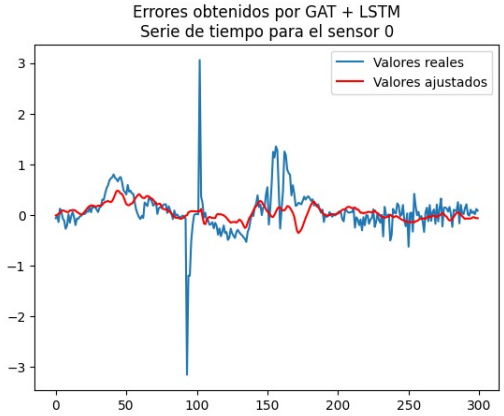
A.2.6. Visualizaciones de los resultados obtenidos Boosting

A continuación se visualizan los resultados obtenidos al ajustar los errores de cada uno de los modelos base usando el mismo modelo.

A.2.7. Al ajustar los errores del modelo GAT-LSTM



(a) Serie de tiempo real vs Serie de tiempo predicha por el modelo GAT-LSTM.



(b) Errores obtenidos por la serie de tiempo predicha por el modelo GAT-LSTM vs Errores ajustados.

Figura A.1: Series de tiempo sobre el sensor 0.

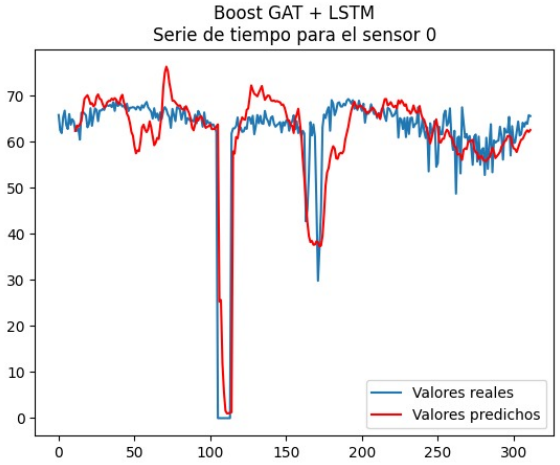
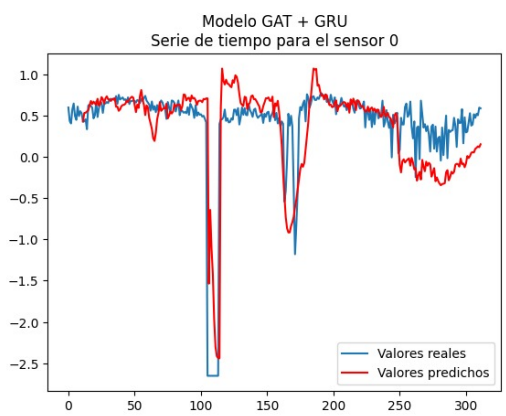
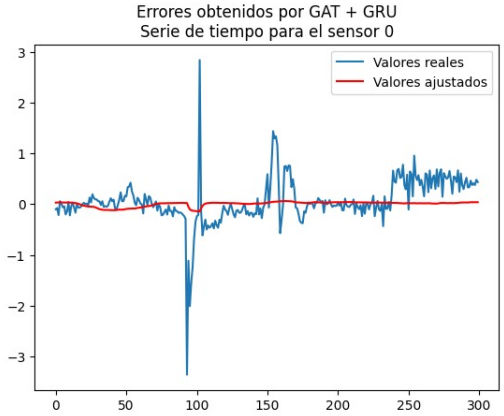


Figura A.2: Serie de tiempo obtenida con el modelo Boosting: GAT-LSTM sobre el sensor 0.

A.2.8. Al ajustar los errores del modelo GAT-GRU



(a) Serie de tiempo real vs Serie de tiempo predicha por el modelo GAT-GRU.



(b) Errores obtenidos por la serie de tiempo predicha por el modelo GAT-GRU vs Errores ajustados.

Figura A.3: Series de tiempo sobre el sensor 0.

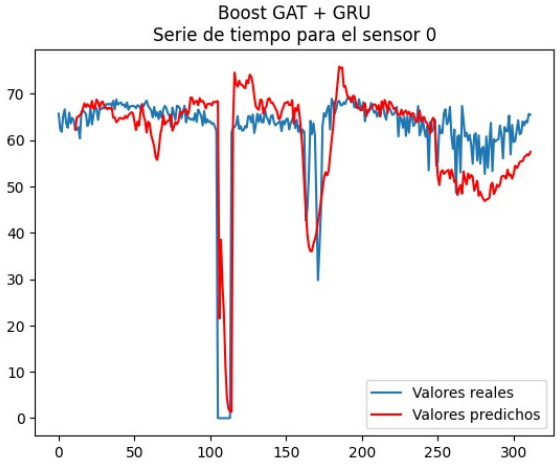
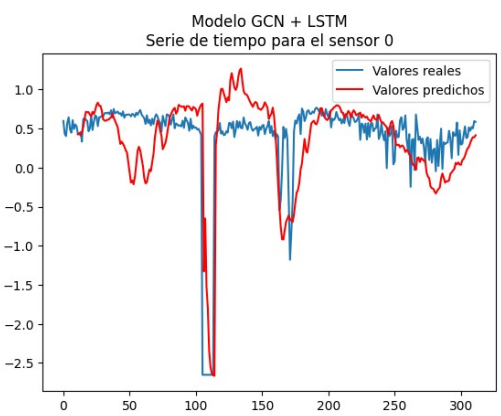
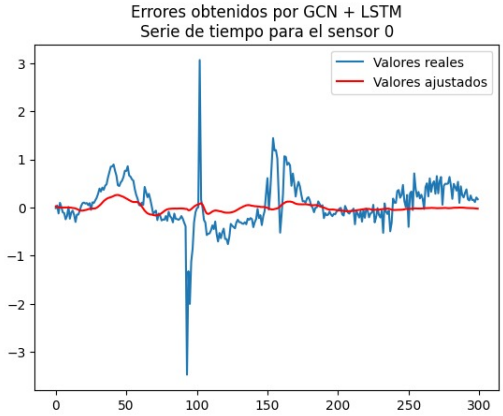


Figura A.4: Serie de tiempo obtenida con el modelo Boosting: GAT-GRU sobre el sensor 0.

A.2.9. Al ajustar los errores del modelo GCN-LSTM



(a) Serie de tiempo real vs Serie de tiempo predicha por el modelo GCN-LSTM.



(b) Errores obtenidos por la serie de tiempo predicha por el modelo GCN-LSTM vs Errores ajustados.

Figura A.5: Series de tiempo sobre el sensor 0.

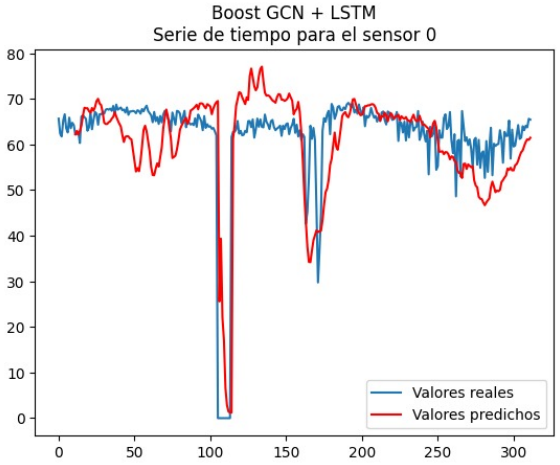


Figura A.6: Serie de tiempo obtenida con el modelo Boosting: GCN-LSTM sobre el sensor 0.

Referencias bibliográficas

[1] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[2] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

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

[6] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Wenhao Huang, Guojie Song, Haikun Hong, and Kunqing Xie. Deep architecture for traffic flow prediction: Deep belief networks with multitask learning. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2191–2201, 2014.

[9] John D. Hunter. Matplotlib: A 2d graphics environment, 2007.

[10] Yuhan Jia, Jianping Wu, and Yiman Du. Traffic speed prediction using deep learning method. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*, pages 1217–1222. IEEE, 2016.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Aprendizaje profundo. *Nature*, 521(7553):436–444, May 2015.

[14] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019.

[15] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[16] Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. *sensors*, 17(4):818, 2017.

[17] Joao Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa. Ensemble approaches for regression: A survey. *Acm computing surveys (csur)*, 45(1):1–40, 2012.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning

library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] Python Software Foundation. *Python Language Reference, version 3.10*, 2023. Accessed: 2024-07-13.

[20] PyTorch. Gru — pytorch 1.10.0 documentation. https://pytorch.org/docs/stable/generated/torch.nn.GRU.html. Accessed: insert date here.

[21] PyTorch. Lstm — pytorch 1.10.0 documentation. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html. Accessed: insert date here.

[22] S. Raschka, Y. Liu, V. Mirjalili, and D. Dzhulgakov. *Machine Learning with PyTorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python*. Expert insight. Packt Publishing, 2022.

[23] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzman Lopez, Nicolas Collignon, et al. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 4564–4573, 2021.

[24] Zahraa Al Sahili and Mariette Awad. Spatio-temporal graph neural networks: A survey. *arXiv preprint arXiv:2301.10569*, 2023.

[25] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[26] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

[27] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[28] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[29] Muhammad Sajib Uzzaman, Chandan Debnath, Md Ashraf Uddin, Md Manowarul Islam, Md Alamin Talukder, and Shamima Parvez. Lrcn based human activity recognition from video data. *SSRN Electronic Journal*, 2022.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5998–6008, 2017.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[32] Jingyuan Wang, Qian Gu, Junjie Wu, Guannan Liu, and Zhang Xiong. Traffic speed prediction and congestion source exploration: A deep learning method. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 499–508. IEEE, 2016.

[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[34] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Evaluación empírica de activaciones rectificadas en redes convolucionales, 2015. [En línea]. Disponible: https://arxiv.org/abs/1505.00853.

[35] Xueyan Yin, Genze Wu, Jinze Wei, Yanming Shen, Heng Qi, and Baocai Yin. Deep learning on traffic prediction: Methods, analysis, and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4927–4943, 2021.

[36] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[37] Chenhan Zhang, JQ James, and Yi Liu. Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting. *Ieee Access*, 7:166246–166256, 2019.

[38] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, September 2020.