UNIVERSIDAD
ANDRES BELLO

UNIVERSIDAD ANDRÉS BELLO
FACULTAD DE INGENIERÍA

INGENIERÍA CIVIL INFORMÁTICA

Clasificación de estrellas variables por medio de aprendizaje autosupervisado comparando LSTM y GRU

PABLO ANDRES JARA CHIARA
ROBERTO PATRICIO MERINO LARA
Profesor guía: Billy Peralta Márquez

SANTIAGO - CHILE

Julio - 2022

Índice

1. Resumen

Teniendo en consideración el esfuerzo que se ha realizado durante el último tiempo para clasificar estrellas variables utilizando técnicas de aprendizaje automático, es que toma importancia el siguiente trabajo. Este busca aplicar nuevos algoritmos en el área para continuar simplificando la tarea, y para que de esta manera los científicos y astrónomos puedan dedicar su esfuerzo obteniendo nuevos conocimientos a partir de la automatización de esta. Se mencionan importantes conceptos como estrellas variables, recolección de datos, preprocesamiento de ellos, algoritmos de aprendizaje automático para comprender a cabalidad el trabajo que se muestra en este documento. Por lo que estos temas serán abordados en detalle cuando sea necesario y de manera más general dependiendo de la profundidad que se quiera dar al asunto tratado.

2. Abstract

Considering the effort that has been made in recent times to classify variable stars using machine learning techniques, the following work takes its importance. It seeks to apply new algorithms in the area to continue simplifying the task, and so that scientists and astronomers can devote their efforts to obtaining new knowledge from its automation. Important concepts such as variable stars, data collection, data preprocessing, machine learning algorithms will be mentioned to fully understand the work shown in this document. Therefore, these topics will be addressed in detail when necessary and in a more general way depending on the depth that's needed to the subject matter.

3. Introducción

Uno de los momentos claves para el desarrollo de la ciencia moderna es el invento del telescopio, lo que más que permitir mirar hacia el espacio y las estrellas permitió mirar a nuestra propia especie. A partir de estas observaciones y mezclando otras disciplinas se desarrolló el método científico, del que se han contabilizado notables avances que han permitido el desarrollo de la tecnología actual y el entendimiento del mundo que nos rodea. Pero una de las principales razones para mirar al cielo y ponernos en perspectiva es que desde el inicio de nuestra historia se ha mirado el cielo intentando entender qué nos rodea y cómo lo que nos rodea afecta el diario vivir. En muchas culturas diferentes, en distintos periodos de la historia hemos creado respuestas para los mismos fenómenos, pero sólo el entendimiento por medio del método científico permitió sintonizar a toda la especie detrás de una idea similar.

Persiguiendo esta misma curiosidad intrínseca del ser humano es que se han logrado avances importantes en distintas áreas: biología, física, matemáticas, química, medicina y últimamente las ciencias de la computación. Todas estas disciplinas tienen en común la búsqueda de una verdad simplemente intentando demostrar que no sabemos la respuesta a nada, y bajo esta primicia es que se pueden modificar respuestas encontradas anteriormente y mejorarlas para que nuestro modelo del mundo cada vez tenga más sentido.

La interconexión de distintas disciplinas y la cada vez más difusa línea que las separa guía el camino hacia el siguiente trabajo que mezclará las disciplina de las ciencias de la computación y la astronomía, trabajando en sinergia para avanzar en ambos campos y superar desafíos que ninguno podría haber conocido sin la existencia del otro. La astronomía aporta el estudio del universo y las enormes cantidades de datos extraídos del mismo, y la ciencia de la computación aportará la tecnología para medir y obtener información lógica a partir de los mismos.

En este trabajo específico se utilizarán herramientas de aprendizaje automático para clasificar observaciones obtenidas de una recopilación de datos que realizó la agencia espacial europea (ESA) con su telescopio espacial Gaia. Específicamente bases de datos con cientos de miles de observaciones de estrellas variables y su clasificación.

Es importante entender la diferencia entre datos e información, y es que, a pesar de la gran cantidad de datos que se obtienen de las observaciones astronómicas no toda esta es utilizable de inmediato. Se requieren técnicas de preprocesamiento y procesamiento de los mismos, para ser analizados. Y en este punto es donde toma importancia el aprendizaje automático, es posible entrenar a una máquina para que realice el trabajo que durante todo este tiempo vienen realizando los y las astrónomas.

Los caminos posibles son aumentar la cantidad de gente dedicada al trabajo de etiquetado de observaciones, o mejorar notablemente la manera en que se etiquetan las mismas, que es claramente el camino que se ha venido realizando en los últimos años. La solución propuesta tiene que ver con esto último y utilizando trabajos anteriores es que se puede realizar una aproximación más óptima con menor cantidad de datos etiquetados, disminuyendo así el principal cuello de botella a la hora de obtener información útil de toda esta cantidad de datos.

Nuestra propuesta consiste en entrenar una red neuronal no supervisada utilizando un método de parches, explicado en detalle posteriormente, y utilizar este mismo aprendizaje que no tiene costo alguno de etiquetas para posteriormente obtener mejores resultados con una muestra menor, es decir, aproximarnos a un resultado similar a un aprendizaje supervisado sin la necesidad de la costosa tarea de etiquetado.

El trabajo abordará en primera instancia el marco teórico, es decir, los trabajos en los que nos apoyamos para conseguir el conocimiento necesario para realizar la tarea propuesta. Se explicarán conceptos tanto astronómicos como de las ciencias de la computación que envuelven el trabajo y harán más fácil su comprensión e involucramiento.

Conceptos como las curvas de luz y las técnicas para entenderlas de mejor manera, la base de datos sobre la que se trabajó y en la que estarán basados los resultados, conceptos sobre redes neuronales, tanto en general como específicamente las utilizadas en este trabajo. Además de trabajos previos relacionados con la tarea de distintos autores.

Luego de comprender estos conceptos se pasará a describir el trabajo realizado, la etapa de preprocesamiento, metodología utilizada para llevar a cabo los experimentos además de la explicación en profundidad de los experimentos y el camino que siguió para acercarse a resultados relevantes.

Finalmente se explicará las conclusiones obtenidas a partir de los resultados y se comparará estos con la hipótesis inicial para evaluar el resultado en sí.

4. Marco teórico

4.1. Estrellas variables

[1] [2] Es importante además del contexto tener claro cuál será el tema de estudio. Si bien los algoritmos de aprendizaje automático se pueden aplicar a innumerables temas, este es uno de los que funciona bastante bien por la cantidad de datos recolectados en los distintos telescopios alrededor del mundo. Además de lo anterior es un tema de estudio muy importante para desarrollar en Chile, ya que el norte, específicamente las regiones de Antofagasta, Atacama y Coquimbo, gracias a sus climas áridos, los cielos están siempre limpios, convirtiéndolos en un escenario ideal para la observación y desarrollo de investigación.

Entre los observatorios más famosos del país, se encuentra ALMA (Atacama Large Milimiter Array), uno de los proyectos astronómicos más grandes del mundo. Actualmente Chile posee un 40 % de la observación astronómica mundial.

¿Cómo se detectan las variaciones de estas estrellas? Uno de los tipos de variabilidad más comunes involucra cambios en el brillo, el que corresponderá a la magnitud lumínica de la estrella. Al registrar una estrella específica por un período de tiempo prolongado es posible obtener una curva de la magnitud de la luz de la estrella en el tiempo, llamada curva de luz. La manera en que la estrella brilla, o sea, la forma que tenga su curva de luz les otorga a los astrónomos la información para inferir por qué esta estrella brilla y por ende a que clase de estrella variable corresponde. Dependiendo del tipo de estrella (que ahora se llamará clase) el período de esta variará, algunas teniendo períodos de oscilación muy largos, las que no se considerarán en estos estudios, ya que el tiempo de uso de telescopio es escaso para cada investigación y el resto de las clases nos entregan información más que suficiente para llevar a cabo los experimentos necesarios. Los picos de brillo son llamados máxima y su contraparte mínima.

Algunas métricas importantes a recolectar de las observaciones son, por ejemplo, el tipo de período de la curva de luz, su periodicidad o irregularidad. La magnitud del período de fluctuación de la estrella, además de la forma que presenta esta curva de luz, analizando simetrías,

variaciones bruscas o suaves y las máxima y mínima de cada curva de luz.

Las nubes de magallanes son dos galaxias enanas que orbitan la vía láctea, visibles sólo desde el hemisferio sur del planeta. Se ven como piezas separadas a la vía láctea a simple vista (sin la necesidad de un telescopio). La Gran Nube de Magallanes se encuentra a unos 163.000 años luz de la vía láctea y tiene un diámetro de 14.000 años luz, mientras que la Nube Pequeña de Magallanes a unos 197.000 años luz, y tiene un diámetro de 7.000 años luz, en comparación a los 100,000 años luz de diámetro de la Vía Láctea, por ejemplo.

4.2. Algunos tipos de estrellas variables

Estrellas variables pulsantes: Las estrellas variables pulsantes se hinchan y se encogen, afectando el brillo que emiten.

Las estrellas variables Cefeidas son un subgrupo de estrellas que pertenecen a este grupo. Estas pulsan en un periodo regular causado por la resonancia de su propia masa, de unos días hasta meses.

RR Lyrae: Son similares a las Cefeidas pero emiten menos brillo y tienen además períodos mucho más cortos. Tienen una relación bien entre su periodo y su luminosidad, por lo que son buenas para ser utilizadas como indicadores de distancia, al igual que las Cefeidas. Pueden variar desde un 20 % a un 500 % de su luminosidad sobre un periodo de unas cuantas horas hasta un día o más.

Delta Scuti: También son similares a las Cefeidas pero con mucho menos brillo y períodos más cortos aún que RR Lyrae. Antes eran conocidas como Cefeidas Enanas. Son velas estándares, o sea, sirven para medir el brillo de estrellas utilizando el período de la misma, de la misma forma que las Cefeidas.

Mira Variables: Son una clase de estrellas pulsantes caracterizadas por su fuerte color rojo

y períodos de pulsación mayores a 100 días. Son gigantes rojos en su etapa tardía de evolución estelar, y se convertirán en enanas blancas en unos pocos millones de años más.

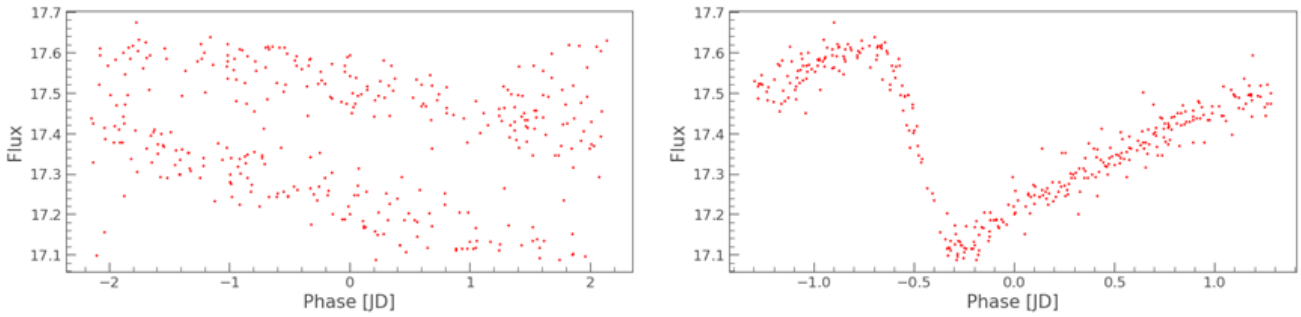
Estos son algunos de los ejemplos de estrellas que se estudiarán en este trabajo, y se espera clasificar utilizando técnicas de aprendizaje automático. Si bien dentro de los parámetros del modelo a utilizar no es necesario conocer la composición de cada estrella, o sus diferencias específicas con otras variables, sí es útil tener un conocimiento previo para analizar si los resultados obtenidos son representativos de la realidad y trabajos anteriores.

4.3. Curvas de luz

[3] En el campo de la astronomía, una curva de luz es un gráfico que refleja la intensidad de una señal de luz versus el tiempo, es decir, la variación de la magnitud lumínica captada en el tiempo. Estas curvas de luz pueden ser periódicas, como en el caso de un sistema binario eclipsado, por ejemplo un planeta orbitando una estrella que hace disminuir su magnitud lumínica con el periodo de su rotación, cefeidas variables u otras estrellas variables que se verán a continuación. Los gráficos de magnitud aparente en una estrella variable generalmente se utilizan para visualizar y analizar su comportamiento.

Algunas técnicas que se utilizan para hacer estas curvas más simples de analizar son por ejemplo el 'doblar' una curva sobre sí misma, dependiendo del periodo que esta tenga, utilizando el procesamiento de la señal. La imagen de abajo muestra el algoritmo de Lomb-Scargle Periodogram en acción, para mostrar una señal menos ruidosa que la original, y que nos puede permitir obtener información relevante para el análisis de la curva de luz. Pasamos de una señal ruidosa y sin sentido, a una que podemos interpretar más fácilmente.

La librería 'lightkurve' nos provee los recursos necesarios para procesar esta curva de luz. Funciones como *period_at_max_power* nos entrega el valor peak en el periodograma, luego se usa este valor para definir dónde se debe 'doblar' el gráfico y obtener una curva más limpia.



(a) Folded periodogram

(b) Folded at max power

Figura 1: Curvas de luz

4.4. Redes neuronales utilizadas

4.4.1. LSTM

[4] La red LTSM se diseñó para evitar los problemas de dependencia a largo plazo. Recordar las largas secuencias durante un largo período de tiempo es su forma de trabajar.

El funcionamiento es un proceso ligeramente complejo, como se puede ver en la imagen anterior en cada momento toma entrada de tres estados diferentes como el estado de entrada actual, la memoria a corto plazo de la celda anterior y, por último, la memoria a largo plazo.

Estas células utilizan las compuertas para regular la información que se debe mantener o descartar en la operación de bucle antes de pasar la información a largo y corto plazo a la siguiente celda. Podemos imaginar estas puertas como filtros que eliminan información no deseada seleccionada e irrelevante. Hay un total de tres puertas que LSTM utiliza como Input gate, Forget gate, Output gate.

Input Gate

La puerta de entrada decide qué información se almacenará en la memoria a largo plazo. Solo funciona con la información de la entrada actual y la memoria a corto plazo del paso anterior. En esta puerta, filtra la información de variables que no son útiles.

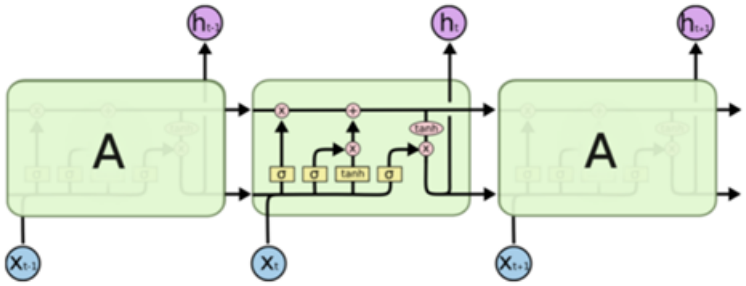


Figura 2: El módulo repetitivo en una LSTM contiene cuatro capas interactivas [5]

Forget Gate

La puerta de reinicio decide qué información de la memoria a largo plazo se conserva o se descarta y esto se hace multiplicando la memoria entrante a largo plazo por un vector de olvido generado por la entrada actual y la memoria corta entrante.

Output Gate

La puerta de salida tomará la entrada actual, la memoria a corto plazo anterior y la memoria a largo plazo recién calculada para producir nueva memoria a corto plazo que se transmitirá a la celda en el siguiente paso de tiempo. La salida del paso de tiempo actual también se puede extraer de este estado oculto.

4.4.2. GRU

[6] En resumen, GRU funciona de forma similar que la RNN la diferencia que tienen está en la operación y las puertas asociadas con cada unidad GRU. GRU incorpora dos mecanismos de operación, la puerta de actualización y la puerta de reinicio.

Update Gate

La puerta de actualización es responsable de determinar la cantidad de información previa que debe pasar a lo largo del siguiente estado. Esto es realmente útil porque el modelo puede decidir copiar toda la información del pasado y eliminar el riesgo de desaparecer el gradiente.

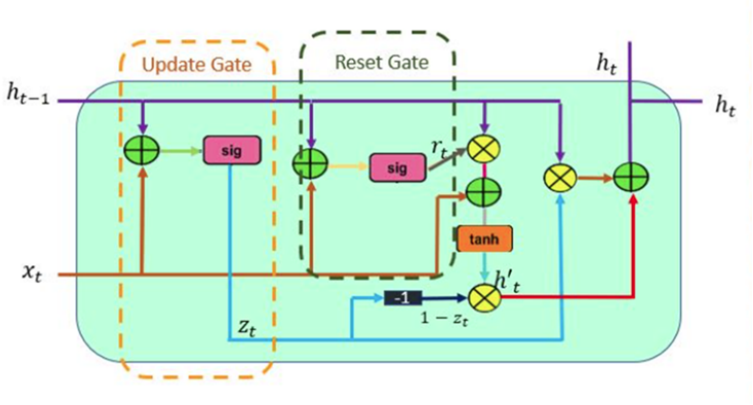


Figura 3: Estructura típica de un modelo GRU [7]

Restart Gate

La puerta de reinicio se utiliza desde el modelo para decidir cuánta de la información pasada se necesita reutilizar o no; en resumen, decide si el estado celular anterior es importante o no.

4.5. ¿Cuál es la diferencia entre GRU y LSTM?

[7] GRU posee dos puertas, LSTM tiene tres puertas.

GRU no posee ninguna memoria interna, no tienen una puerta de salida que esté presente, en LSTM la puerta de entrada y la puerta de salida están acopladas por una puerta de actualización y en GRU la puerta de reinicio se aplica directamente al estado oculto anterior.

En LSTM, la responsabilidad de la puerta de reinicio es asumida por las dos puertas, es decir, la entrada y salida.

GRU utiliza menos parámetros de entrenamiento y, por lo tanto, utiliza menos memoria y se ejecuta más rápido que LSTM, mientras que LSTM es más preciso en un conjunto de datos más grande. Uno puede elegir LSTM si se trata de secuencias grandes y la precisión se refiere, GRU se utiliza cuando tiene menos consumo de memoria y desea resultados más rápidos.

4.6. Aprendizaje autosupervisado

[8] El aprendizaje autosupervisado es un método de aprendizaje automático y consiste en aprender a desarrollar una tarea a partir de datos sin etiquetar, como lo hace el aprendizaje no supervisado, puede incluso ser considerado como un paso intermedio entre el aprendizaje no supervisado y supervisado. La red aprende en dos pasos. Primero, la tarea es resuelta basada en en pseudo etiquetas propuestas para la tarea, en este caso utilizaremos un método que se explicará más adelante llamado data2vec. Segundo, la tarea real se realiza ya sea con aprendizaje supervisado o no supervisado, en este caso utilizaremos aprendizaje supervisado en una tarea de clasificación.

Una de las características atractivas del aprendizaje autosupervisado es que el entrenamiento inicial puede realizarse con data de menor calidad donde el resultado final no es de mayor importancia, ya que no es la tarea a la que se quiere medir su rendimiento.

Se sitúa en un punto medio entre el aprendizaje supervisado y no supervisado como se mencionó anteriormente ya que toma características de ambos paradigmas. Genera su aprendizaje a partir de datos sin etiquetar como el aprendizaje no supervisado, pero la salida de este método no es la clusterización como generalmente es usado el aprendizaje no supervisado, más bien la clasificación, segmentación o regresión como cualquier modelo de aprendizaje supervisado, es decir, puede obtener lo mejor de ambos. La precisión al clasificar de una red supervisada, y usar la riqueza de datos sin etiquetar que tienen las redes no supervisadas.

4.7. Base de datos de Gaia

Gaia es un observatorio espacial de la Agencia Europea Espacial (ESA por sus siglas en inglés). Fue lanzada en 2013 y se espera que opere hasta 2025. La nave fue diseñada para la astrometría, es decir, medición de distancias, paralajes y posiciones astronómicas. El hecho de que estas mediciones sean hechas desde el mismo espacio entrega una precisión sin precedentes. Esta misión apunta a construir el catálogo tridimensional más grande y fiel que se haya creado del espacio, totalizando aproximadamente mil millones de objetos astronómicos, principalmente

estrellas, pero también planetas, cometas, asteroides y quasares entre otros.

Para estudiar con mayor precisión la posición y movimiento de los objetivos a medir, el observatorio monitorea cada objeto alrededor de 70 veces durante el periodo que dure la misión principal (2014-2019).

La base de datos de Gaia llamada DR2 (Data Release 2) comprende un catálogo de 363969 observaciones de estrellas variables, y fue obtenida a partir de los recursos dispuestos por Becker et al. Las curvas de luz en este set de datos contienen un número suficiente de clases y ejemplos para ser utilizadas como grupo de entrenamiento, pero no son representativas de la población real de estrellas ya que están sesgadas hacia objetos brillantes que cumplan cierto límite. El set de datos publicado fue construido utilizando un clasificador de etapas múltiples, basado en las características de curvas de luz además de paralajes, metalicidades y relaciones astrofísicas entre otros metadatos, por lo que una clasificación basada sólo en las curvas de luz no obtendrá un puntaje perfecto *(Becker et al. cap 4.2)*.

El set de datos está compuesto de dos elementos principales, un archivo .csv que contiene el ID del objeto observado, la clase a la que pertenece, su ubicación en memoria y el número de observaciones del mismo. Además de este archivo que será utilizado como índice se encuentra el set de datos de las curvas de luz en sí mismas que está compuesto por una serie de archivos .dat indicado en el archivo anterior.

ID	Class	Path	N
4395790042666181888	0	./LCs/4395790042666181888.dat	47
1748775099405304832	0	./LCs/1748775099405304832.dat	30
1441812897701323136	0	./LCs/1441812897701323136.dat	59
4868009430431207552	2	./LCs/4868009430431207552.dat	33
5793099859342399744	1	./LCs/5793099859342399744.dat	35

4.8. Revisión bibliográfica adicional

En esta subsección se describirán los trabajos que sirven como antecedentes para llevar a cabo la tarea. Estos varían desde trabajos relacionados con la obtención de los datos, trabajos previos realizados en el mismo tema de clasificación de estrellas variables con métodos supervisados, temas de aprendizaje automático básicos para contextualizar.

Además de algunos trabajos donde se estudian bases de datos similares a la que se utilizará ahora y el paper principal en el que se basa este trabajo relacionado con redes neuronales recurrentes para la clasificación de estrellas variables.

4.8.1. Data Mining and Machine-Learning in Time-Domain Discovery Classification

[9] El descubrimiento de patrones mediante el uso de tecnologías de clasificación en astronomía de "Time-Domain".

Uno de sus objetivos es abstraer al astrónomo de la recolección de datos mediante técnicas de ML y Data-Mining, ponderando la clasificación y el coste de cómputo (entre otras variables), dejándolo así en la interpretación de estos y al procesamiento/clasificación automático de imágenes en el dominio del aprendizaje automático.

En el PTF, se encontró que la cantidad de candidatos reales de eventos astronómicos era de uno por cada cien, por lo que utilizar escáner humano era inviable. En este caso se decidió aplicar un clasificador ML (random forest) en casos con su salida conocida (clasificado por humanos), con una docena de características, estos se puntúan entre 0 y 1, siendo 1 un evento real y 0 uno no genuino (hablando de eventos astronómicos relevantes, tomando en consideración la razón señal-ruido y otros factores). Luego de un año de muestras se crearon set de entrenamientos de candidatos reales y falsos.

La aproximación tradicional de clasificación ha sido basada en expertos humanos, pero con

datos de cientos de miles a miles de millones de estrellas y eventos, este rol debe ser cambiado necesariamente por ML y otras técnicas de Data Mining.

La clasificación en base a características presenta ventajas para utilizar modelos de ML, hace que resultados obtenidos en condiciones diferentes puedan ser comparables y además permite el ajuste de la complejidad de las características dependiendo de la calidad de la fuente de datos.

Hay dos grandes grupos de métodos a utilizarse en la clasificación de estos eventos, estos son la clasificación supervisada y la clasificación no supervisada. Sus objetivos finales son distintos, el primero presenta un modelo más certero, con menos errores. Mientras que la clasificación no supervisada muestra la distribución de las características, y la cantidad de clases encontradas. Además de lo anterior está la automatización en la búsqueda de features.

En síntesis, lo que se busca es poner a disposición de la clasificación de eventos astronómicos puntuales las herramientas de clasificación de ML y Data-Mining, sopesando sus ventajas y desventajas dependiendo del modelo y método a utilizar. Para de esta manera aumentar la cantidad de eventos interesantes encontrados sin sacrificar el tiempo de los astrónomos.

4.8.2. Automated supervised classification of variable stars

[10] La idea principal es obtener series de datos más limpios, en mayor volúmen y escalables a otros experimentos gracias a la clasificación por inteligencia artificial y reconocimiento de patrones. En este caso se van a analizar los datos de la librería de OGLE en tres locaciones específicas, siendo estos el Galactic bulge y Magellanic Clouds.
Cada clase de estrella variable a observar se clasificará con características propias que identifiquen a la clase y serán comparadas con resultados obtenidos por clasificación humana (la que será considerada correcta). Y las métricas para determinar la calidad del clasificador serán los rangos de detección positivo/negativo y verdadero/falso.

Uno de los principios en este paper es el de considerar la información de color en el clasifi-

cador y ver cómo se comporta el mismo en comparación a cuando el color no es considerado. Bayessian Network (MSBN) y Gaussian Mixture (GM) son dos de los clasificadores principales utilizados. Siguiendo una serie de pasos discriminativos se puede obtener la probabilidad de que una estrella pertenezca a una u otra clase.

Finalmente se muestra la efectividad del clasificador y se invita a la futura investigación donde el clasificador no muestra un gran rendimiento, sea por el diseño del mismo o por la propia fuente de los datos a estudiar. Y se deja en claro que con clasificadores automáticos se puede dejar abstraer al astrónomo de la recopilación y clasificación de los datos y se puede este centrar en el estudio astrofísico que explique los fenómenos encontrados.

4.8.3. An improved quasar detection method in EROS-2 and MACHO LMC data sets

[11] Este trabajo habla del uso de un clasificador boosted RF (algoritmo explicado en el paper), utilizando mejoras realizadas a los algoritmos de clasificación de ML anteriores y a las características de las curvas de luz utilizadas. Contando lo anterior se agrega eficiencia al algoritmo al aproximar y optimizar un parámetro, luego se utilizan los parámetros obtenidos como features del modelo. Se utiliza parallel computing para disminuir el tiempo de procesamiento y se espera que los quasar encontrados puedan proveer información crítica en la evolución de la galaxia, entre otros.

El modelo es entrenado combinando distintos clasificadores lo que mejora el rendimiento de este en los datasets utilizados (dependiendo de la calidad de las muestras). Se encuentra un alto porcentaje (25 %) de falsos positivos en cierta muestra, pero se presenta una posible solución para este problema.

4.8.4. FATS: Feature Analysis for Time Series

[12] En primera instancia se deja claro que es una serie temporal y como el estudio de estas con sus características pueden llevar a clasificar distintas estrellas según sus emisiones de luz en el tiempo. El trabajo que se presenta se entrega una librería que calcula rápida y eficientemente características de una curva de luz e invita a futuros usuarios e investigadores a utilizar las características que se agregan y además a agregar las propias.

Se presentan además muchas características identificables de estas curvas de luz y la manera en que se calculan.

Finalmente se proveen instrucciones para e características y se explica cómo funciona (su input/output), además explicando las salidas que tendrá la ejecución de esta.

4.8.5. Machine learning basics

[13] Para entender bien el deep learning es necesario entender conceptos básicos de machine learning. La mayoría de los algoritmos de machine learning pueden dividirse en dos categorías: algoritmos supervisados y no supervisados.
"Por aprendizaje se quiere decir que un algoritmo aprende de la experiencia E, respecto a una tarea T y una medida de rendimiento P, si su rendimiento en la tarea T, medido por P, mejora con la experiencia E". En el texto se proveen ideas intuitivas para diferentes tipos de tareas, rendimientos y experiencias.

La tarea puede dividirse en distintas categorías, por ejemplo: clasificación, regresión, transcripción, traducción de máquina, salida estructurada, salida estructurada, detección de anomalías, eliminación de ruido, entre otros.El rendimiento es específico de cada tarea, para las tareas de clasificación el rendimiento será su certeza por ejemplo. También se puede comparar la salida del algoritmo con un set de prueba conocido. Es importante saber que se quiere medir para definir un buen medidor de rendimiento.La experiencia de un algoritmo se puede clasificar en sí este es supervisado o no supervisado.

Algoritmos no supervisados: estos tienen como experiencia un gran set de datos con muchas características, el algoritmo aprende características útiles de los mismos.

Algoritmos supervisados: poseen un set de datos con sus características, pero cada muestra está asociada a una etiqueta u objetivo.

Una manera común de representar los datos es con una matriz de diseño, donde cada columna presenta una característica y cada fila, una muestra.

La regresión lineal es un algoritmo simple en machine learning, implica un problema de regresión, donde el algoritmo recibe un vector como entrada y como salida un escalar y. Esto nos entrega además una función de error que podemos utilizar para medir el rendimiento del algoritmo, este error tiende a cero cuando la predicción sea igual a los datos de prueba. Además, existe un parámetro de sesgo que puede ayudar a afinar esta predicción.

La generalización es el concepto de que nuestro algoritmo rinde bien en set de datos nuevos, que introduce también dos conceptos nuevos: underfitting y overfitting. El primero corresponde a un error no lo suficiente bajo en el set de entrenamiento, el segundo a una diferencia grande entre el error del set de entrenamiento y el de prueba.

El teorema de "no free lunch", nos habla de la especificidad de los algoritmos de machine learning. "Por cada par de algoritmos de búsqueda, hay tantos problemas en el que el primer algoritmo es mejor que el segundo como problemas en el que el segundo es mejor que el primero". Por lo que el objetivo final de la investigación en machine learning no es encontrar un algoritmo de aprendizaje universal, sino entender qué tipo de algoritmo tiene un buen rendimiento dado cierto tipo de set de datos.
Estadística Bayesiana: se consideran todos los posibles valores de un evento aleatorio. Se elige un "prior" (previo al resultado), que puede ser lo bastante amplio para reflejar el grado de incertidumbre en el valor de la probabilidad antes de observar los datos. Luego, utilizando el teorema de bayes (probabilidad condicionada), podemos recuperar el efecto de los datos sobre

nuestra “creencia” previa. Nuestro “prior” generalmente comienza como una distribución normal con alta entropía (principio de máxima entropía) y luego la observación de los datos causa que luego de la observación se pierda entropía y se concentre alrededor de valores de parámetros altamente probables.

4.8.6. Scalable End-to-End Recurrent Neural Network for Variable Star Classification

[14] Se enfatiza nuevamente en la capacidad de los algoritmos de clasificación debido a la cantidad de data a la que se enfrentan los astrónomos por la naturaleza de la disciplina. Es nombrada LSST como una survey que producirá alrededor cerca de 15 TB de data por noche que deberán ser procesados en tiempo real, lo que contrasta con muchas otras surveys que se realizan a diario, pero que se diferencian en su manera de tomar datos, objetivos científicos, ópticas, detectores, entre otros factores. Lo que aumenta el error de los clasificadores, obligando a tratar cada una de estas particularidades de forma independiente, dificultando así el uso de técnicas de ML.

Métodos no supervisados se han desarrollado para crear features sin la intervención directa de un científico, los que han mostrado ser competentes y entregan la ventaja de son óptimos para una survey específica y poseen menos sesgo que su contraparte humana.

Se nombran ejemplos de distintas ANN (y algoritmos similares como CNN y RNN) y de cómo estas han sido utilizadas en la obtención de features en surveys específicos, lo que en combinación con otros algoritmos de clasificación permiten disminuir el costo computacional a la vez que aumentan la veracidad de sus clasificaciones.

En este trabajo se propone una neural network que puede aprovechar la gran cantidad de datos que proveen la surveys actuales a la vez que reduce el pre-procesamiento necesario para realizar clasificaciones de estrellas variables, sin necesidad de computar features, puede escalar a gran cantidad de datos y no necesita reentrenar el modelo, lo que lo hace especial.

Los resultados obtenidos con este método son muy buenos, comparándose con métodos que son el estado del arte como los Random Forest usando features FATS (de las que se habló más arriba en este documento). El método presentado no utiliza magnitudes de forma explícita, sino, en su etapa de preprocesamiento los computa para cada curva de luz. Es un método que escala de forma lineal con el número de puntos en la curva de luz, además el método está implementado con aceleración por GPU o que permite procesar cientos de objetos en paralelo, haciéndolo una opción viable en la siguiente generación de surveys como LSST.

4.8.7. The OGLE Collection of Variable Stars. Classical, Type II, and Anomalous Cepheids toward the Galactic Center

[15] En este trabajo se presenta una colección de estrellas variables detectadas por OGLE (Optical Gravitational Lensing Experiment).

Las Cefeidas y estrellas RR Lyr, a veces llamadas pulsantes clásicos, se mueven en oscilaciones radiales. Las Cefeidas variables se pueden dividir en varias subclases que muestran diferentes masas, edades e historias evolutivas. Se nombran distintos tipos de Cefeidas (basadas en sus períodos) cada una en una distinta etapa de su evolución estelar y se plantea que el estado de evolución de las Cefeidas anómalas está bajo debate. En este trabajo se presenta una colección de Cefeidas de OGLE en el bulbo galáctico, donde por primera vez se presentan Cefeidas anómalas en las regiones centrales de la vía láctea. Algunas de las estrellas que anteriormente habían sido clasificadas como Cefeidas clásicas y RR Lyr, en este trabajo se clasifican como Cefeidas anómalas, debido a que cuando se clasificaron por primera vez estas estrellas, no se tenían los métodos para distinguir entre las que eran y su nueva clasificación basándose sólo en sus curvas de luz. Estos descubrimientos proveen una herramienta para poner a prueba modelos de estrellas pulsantes, así como también estudiar la estructura y la formación de estrellas en la región del bulbo galáctico.

4.8.8. Colorful Image Colorization

[16] El objetivo principal de este paper es el de producir una colorización plausible que pueda engañar a un humano, utilizando feed forward en una CNN. Utilizando el espacio de colores "CIELAB", en el que se tienen tres canales: l, a, b. L de lightness, luego a y b para cuatro colores únicos de la visión humana, estos son rojo, verde, azul y amarillo.

Predecir el color tiene la propiedad de que entrenar la data es prácticamente gratis, sólo se debe tomar el canal L como entrada y los canales ab como señal de supervisión, debido a esto es posible trabajar con grandes volúmenes de datos (nota mía: supongo que es a que no se requiere etiquetación).

Los resultados de anteriores intentos tienden a verse desaturados, posiblemente porque su función de error alienta las predicciones conservadoras, al intentar reducir el coste euclidiano entre la estimación y la verdad. En vez de eso, los autores utilizan un coste relacionado con el problema de la colorización. Debido a que la predicción de color es inherentemente multimodal, muchos objetos pueden tener distintas colorizaciones plausibles, una manzana puede ser roja, verde o amarilla, pero es improbable que sea azul o naranja, por ejemplo. Por esta razón se busca predecir una distribución de posibles colores para cada pixel, de hecho, se enfatizan los colores raros. Lo que lleva al modelo a explotar toda la diversidad de la gran cantidad de datos en los que fue entrenado. El resultado final es una coloración más vibrante y perceptivamente realista que los intentos anteriores.

Para la evaluación de la colorización, los autores innovan en cuanto al testing de los resultados, llevando a cabo un "Test de Turing de colorización", en el que les muestran a los participantes colores reales y sintéticos (producidas como resultado del modelo), y les piden identificar el falso. Donde se logra engañar a los participantes en un 32 % de las veces, se indica también que una verdad fundamental, lograría el 50 % de esta métrica. Este 32 % es significativamente mayor que los trabajos previos, lo que demuestra en muchos casos que el algoritmo está produciendo resultados cercanos al fotorrealismo.

También se explora la colorización como forma de aprendizaje de representación auto supervisada, donde los datos en bruto son utilizados como su propia fuente de supervisión. Se prueba como el modelo realiza pruebas de generalización, comparado a algoritmos auto supervisados previos y actuales, se logra un rendimiento que en algunas ocasiones podría considerarse el estado del arte en esta materia.

Los autores indican que sus contribuciones en este paper son en dos áreas. Primero, se realiza un progreso en el problema de la colorización automática de imágenes introduciendo un nuevo marco de trabajo para probar los algoritmos de colorización, potencialmente aplicables a otras tareas en sintetización de imágenes. Segundo, se introduce la tarea de la colonización como un método competitivo y directo para la representación de aprendizaje autosupervisado.

4.8.9. Understanding the Lomb-Scargle Periodogram

[17] El periodograma de Lomb-Scargle es un algoritmo conocido para detectar señales periódicas en datos de tiempo muestreados de manera irregular y es particularmente utilizado en la comunidad astronómica. Se muestra una figura en la que está la data como se obtiene originalmente, luego se aplica el algoritmo de periodograma y se muestra un gráfico de Lomb-Scargle Power vs Periodo. Aquí es donde se comienza a ver el patrón de periodicidad que en la figura anterior no era visible a simple vista, el que muestra el período más repetido. Luego se puede ver un gráfico de la data "doblada" sobre sí misma, de esta manera se puede ver claramente la oscilación periódica original de la fuente, en esta última el gráfico está en función de la fase en vez del tiempo.

El periodograma Lomb-Scargle produce una estimación de la potencia de Fourier como una función del período de oscilación, del que se puede estimar una oscilación, para luego obtener el gráfico final.

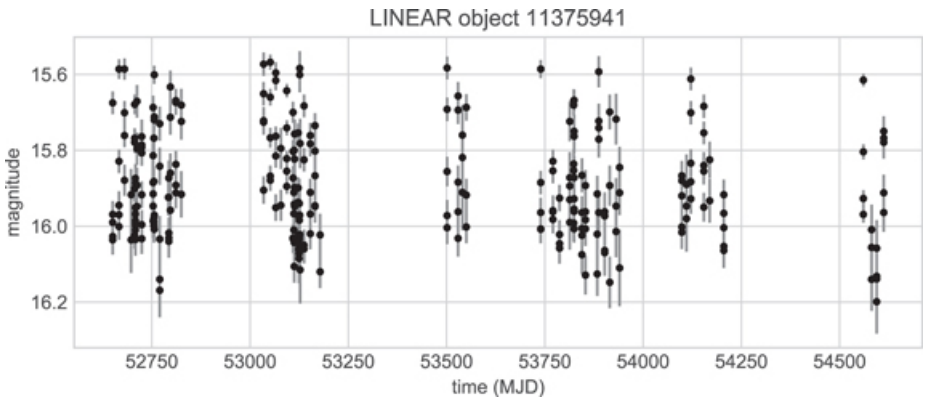


Figura 4: Curva de luz observada de un objeto. Las incertidumbres. Las incertidumbres se indican mediante las barras grises en cada punto.[17]

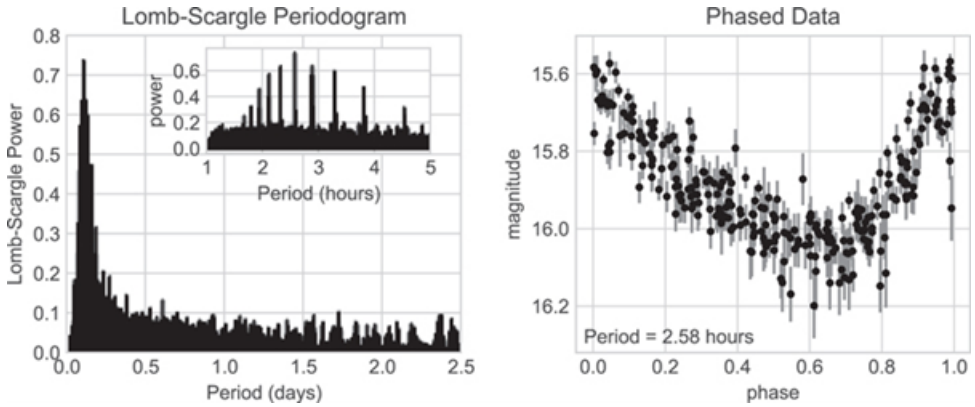


Figura 5: Panel izquerdo: Periodograma de Lomb–Scargle calculado a partir de los datos que se muestran en la Figura 4, con un recuadro que detalla la región cerca del pico. Panel derecho: datos de entrada en la Figura 4 doblados sobre el período detectado de 2,58 horas para mostrar la variabilidad periódica coherente[17]

5. Metodología

El objetivo principal de los experimentos que se realizarán es comprobar la factibilidad de utilizar aprendizaje autosupervisado por medio de transfer learning, es decir, utilizar una red pre entrenada para una tarea en una diferente. Para lograr este objetivo es que se procesa la data de esta manera, es decir un data frame utilizando el método data2vec para aprendizaje no supervisado, y el siguiente data frame con un método de clasificación balanceada para el aprendizaje supervisado. En primera instancia se tomará como base el rendimiento que obtengan los algoritmos de clasificación, que será probado con tres variaciones: Baseline, LSTM, GRU. El baseline nos proveerá una base con la que podremos comparar nuestro rendimiento, pero no será útil para los experimentos. Cada una de las redes neuronales LSTM y GRU será probada con el 100 %, 50 %, 25 %, 10 % y 5 % de la data, como se explicó anteriormente y se medirá la certeza que obtenga cada red con cada dataset. Es decir, se generarán 5 valores de accuracy por cada una de las 2 redes.

Luego se entrenará ambas redes neuronales con el método data2vec y se guardará el aprendizaje de los modelos para ser utilizado posteriormente con la tarea de clasificación, es decir, se entrenará para una tarea y luego será utilizado para realizar una diferente. El objetivo, como se menciona al comienzo, es comprobar si es factible obtener un mejor resultado con una menor cantidad de muestras a la hora de clasificar, lo que podría ayudar a realizar trabajos de clasificación con menos cantidad de datos etiquetados, que como se mencionó al comienzo del trabajo, es data muy costosa de obtener.

Otra de las experimentaciones que se realizará tiene que ver con la cantidad de features que se utilizarán. El dataframe viene con dos features luego del pre procesamiento, y en esta etapa se experimentará agregando dos más, estas serán: timediff y magdiff. Que corresponden a la diferencia de cada fila con su sucesora, como se explicó en el preprocesamiento de los datos. La idea es que agregando más características se pueda hacer uso de una heurística que ayude a la red a encontrar los pesos óptimos para mejorar su rendimiento.

Se comparará los rendimientos de ambas redes con ambos métodos y se medirá el tiempo

de cómputo además del accuracy obtenido en cada experimento. Además, se busca encontrar si es factible realizar este entrenamiento previo para mejorar el rendimiento.

5.1. Análisis y pre-procesamiento

Con el set de datos anteriormente mencionados es que se realizará el experimento. Si bien se tenían otros set de datos a disposición por un tema de capacidad de cómputo y alcance del proyecto es que se enfocaron los esfuerzos en llevar a cabo el análisis y experimentos con los datos de Gaia.

Cada objeto o curva de luz tiene la siguiente estructura:

```
time,mag,mag_err,flux,flux_error
1709.89070807,18.63196240,0.01600309,664.60152249,22.55571290
1745.70337215,18.63948098,0.00516483,660.01514171,7.22938747
1791.74144735,18.58765190,0.00579844,692.28597224,8.51310711
1791.91761030,18.84529460,0.00343031,546.04502147,3.97240244
```

De estas 5 columnas se trabajará con las dos primeras, es decir, time y mag. Además de lo anterior se agregaron dos columnas como features para mejorar el rendimiento de la red que contemplan la diferencia entre las filas continuas para las columnas de tiempo y magnitud. Obteniendose data con la siguiente estructura:

```
[[ 1.71130843e+03  1.97558993e+01  2.18716168e+01  1.62956690e-01]
 [ 1.73318005e+03  1.99188560e+01  1.76145710e-01 -6.15443670e-01]
 [ 1.73335619e+03  1.93034123e+01  2.50145300e-01  5.67699680e-01]
 [ 1.73360634e+03  1.98711120e+01  2.50142620e-01 -4.26121000e-01]
 [ 1.73385648e+03  1.94449910e+01  5.00268309e+00  4.79982870e-01]
```

5.1.1. Data2Vec

El primer entrenamiento será realizado utilizando el método data2vec. Este consiste en quitar parte de la data de entrada para ser proveída como salida. En este caso y luego de experimentar con los datos obtuvimos los mejores resultados en tiempo de cómputo y error absoluto medio quitando 5 datos de entrada y asignando estos mismos y en orden como salida.

Se utilizarán sólo los objetos que tengan más de 23 observaciones y se truncará el resto de las mismas para darle así a la red un tamaño fijo de entrada, que corresponderá a 22 observaciones de entrada, una observación es removida para utilizar el método de diferencia entre filas np.diff(), y 5 observaciones de salida. De esta forma obtenemos 167685 objetos con 27 observaciones cada uno, lo que nos resulta en un dataframe de 4527495 filas y 4 columnas.

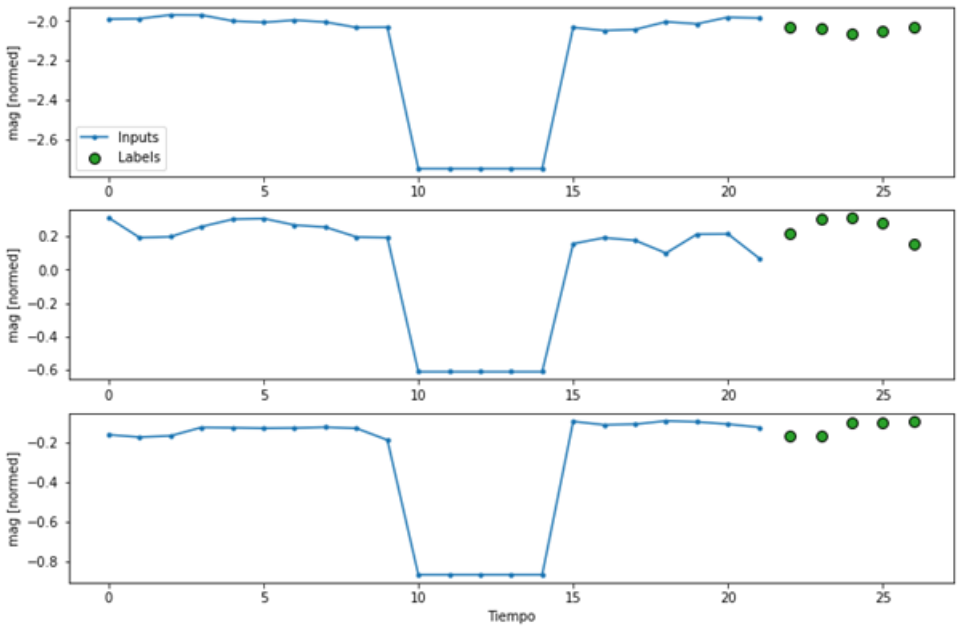


Figura 6: Diagrama con distintas curvas de luz que muestra el método data2vec en el dataset Gaia. Se observa el valle con los datos que se alteran para ser entregados como salida al algoritmo, marcado en puntos verdes.

5.1.2. Obtener el dataframe, primera parte de aprendizaje autosupervisado

El dataframe es armado en el código a continuación:

```
labels_df = pd.read_csv("..\Gaia\GAIA_dataset.dat")
labels_df = labels_df[labels_df['N'] >= 23]
path = "..\Gaia\LCs\\"
frames = []
for index, row in labels_df.iterrows():
    filename = path + str(row['ID']) + ".dat
    frame = pd.read_csv(filename, header=0)
    lower_bound = 11
    upper_bound = lower_bound+5
    serie = frame.iloc[1:24,0:2]
    patch = frame.iloc[lower_bound:upper_bound, 0:2]
    temp_patch = frame.iloc[lower_bound:upper_bound+1, 0:2]
    min_serie = serie.iloc[:,1]
    min_value = min(min_serie) - 2
    patch_min = np.full((5, 2), min_value)
    patch_min[:,0] = patch.iloc[:,0]
    temp_newframe = np.concatenate((frame.iloc[1:lower_bound, 0:2],
                                    patch_min,
                                    frame.iloc[upper_bound:23, 0:2],
                                    temp_patch))
    newframe_diff = np.diff(temp_newframe, axis = 0)
    temp_newframe = pd.DataFrame(temp_newframe)
    temp_newframe = temp_newframe.iloc[0:27, 0:2]
    newframe_diff = pd.DataFrame(newframe_diff)
    newframe = np.concatenate((temp_newframe,
                               newframe_diff), axis = 1)
    frames.append(newframe)
```

Luego de completar el dataframe, estos son guardados en formato hdf en disco, para posteriormente ser utilizados en el entrenamiento.

5.1.3. Segunda parte aprendizaje autosupervisado, clasificación.

Para esta parte es necesario utilizar ambos elementos presentes en el dataset, es decir, el archivo .csv que nos proveerá con la etiqueta buscada, y el set de curvas de luz que corresponden a cada clase.

En primera instancia es necesario leer el .csv y descartar las clases que tengan menor cantidad de objetos que los establecidos, en este caso menos de 500 objetos por clase, por lo que de 9 clases en primera instancia pasamos a obtener 7. Luego es necesario cambiar el nombre de cada clase por un ID para que este pueda ser interpretado por la red neuronal.

```
labels_df = pd.read_csv("..\Gaia\GAIA_dataset.dat")
unique_vals = labels_df['Class'].unique()
labels_df['Class'].replace(to_replace=unique_vals,
           value= list(range(len(unique_vals))),
           inplace=True)
```

Posteriormente seleccionamos sólo los objetos con más de 23 observaciones y truncamos los que sobrepasen este número, de la misma forma que se hizo anteriormente. Revisamos la distribución de clases que tiene la siguiente forma:

```
labels_df['Class'].value_counts()
2    79834
0    66944
1    18346
4     5858
5     4024
6     1151
3      504
Name: Class, dtype: int64
```

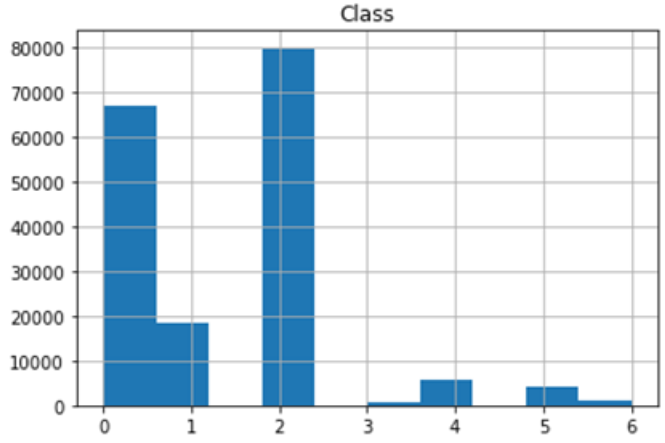


Figura 7: Histograma que muestra el desbalance de clases con el que se descrito en el cuadro anterior. La clase más presente (2) tiene más de 150 veces la cantidad de elementos que la menor (3)

Este desbalance de clases será abordado más adelante por medio de la utilización del método SMOTE.

Ahora procedemos a armar ambos dataframe, la entrada y la salida, la entrada contendrá 22 observaciones de la curva de luz, y la salida será la clase.

```
path = "..\Gaia\LCs\\"
xframes = []
yframes = []
for index, row in labels_df.iterrows():
    filename = path + str(row['ID']) + ".dat"
    frame = pd.read_csv(filename, header=0)
    temp_newframe = frame.iloc[0:23, 0:2]
    newframe_diff = np.diff(temp_newframe, axis = 0)
    newframe = pd.DataFrame(newframe_diff))
    xframes.append(newframe)
    yframes.append(row['Class'])
```

Esto nos deja con un yframes de 176661 elementos, misma cantidad de xframes, pero el total de este es 3886542 y 2 columnas. Las 2 columnas con la información de diferencia entre filas es concatenado posteriormente, se verá en detalle cuando se realice el proceso de SMOTE. Posteriormente este dataframe también es guardado en formato hdf para su posterior trabajo en la red neuronal.

Luego de ser armado el dataframe y dividida la data en set de entrenamiento, validación y prueba como se verá más adelante, esta es normalizada utilizando una función z score, es decir, la media estará centrada en el 0.

5.2. División de data

Luego de resolver el problema del desbalance, el siguiente paso es hacer la división de la data entre train, test, validación.

El primer intento se realizó de forma secuencial esto quiere decir que se dividió el dataframe total y se recortaron trozos según se iba recorriendo en orden desde principio a fin. Se tomó

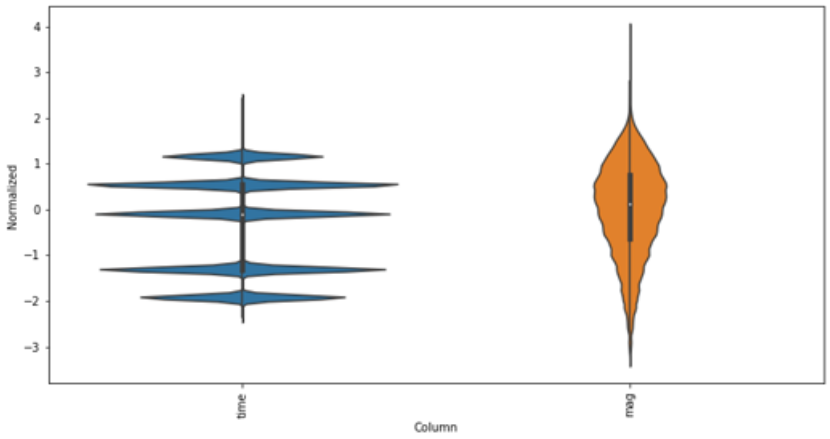


Figura 8: Distribución de las features luego de ser normalizadas, se observa que la variable time es discontinua en algunos puntos, debido a la manera en la que se tomaron las muestras, y la variable mag tiene una distribución más uniforme entorno a la media.

desde el 0 al 65 % del dataframe como train del 65 % al 90 % como validación y del 90 % al 100 % como test. En el siguiente código se muestra como se hizo la división.

```
#División de data \secuencial"
column_indices = {name: i for i, name in enumerate(df.columns)}
n = len(df)
n = n/sample_size #Debe ser valor entero
train_df = df[0:int(n*0.65*sample_size)]
val_df = df[int(n*0.65*sample_size):int(n*0.9*sample_size)]
test_df = df[int(n*0.9*sample_size):]
```

El problema que se presentó al realizar la división de la data de esta forma fue que las algunas clases no estaban contenidas dentro de la data de entrenamiento y en su mayoría eran lecturas de la clase mayoritaria, esto como obvio resultado provocó que al ejecutar las redes la LSTM y GRU entregaban resultados de accuracy muy bajos y sin variación en cada epoch y el baseline se volvió imbatible ya que por la heurística de la red y su funcionamiento le era muy fácil realizar las predicciones.

Como segundo intento se usó pipeline de la librería sklearn para poder seleccionar los datos

de cada dataframe de forma aleatoria, pero considerando la etiqueta, así poder asegurar muestras de todas las clases en cada dataframe. Además se generó un data frame llamado trash para poder y poder analizar el comportamiento de cada red según la cantidad de data de entrenamiento le proporcionamos.

En el código siguiente se puede visualizar la creación de los 3 data frames de forma randomizada y el nuevo data frame trash, es el que se queda con el resto de data que se va quitando desde el data frame de entrenamiento.

```
df_array = df.to_numpy()
array_flat = np.reshape(df_array,((int(len(df)/sample_size),sample_size,3)))
column_indices = {name: i for i, name in enumerate(df.columns)}
X_train_val,test_df,y_train_val,y_test_df = train_test_split(array_flat,
                                            yframes, test_size=0.1,
                                            random_state=42)
X_trainp, val_df, y_trainp, y_val_df = train_test_split( X_train_val,
                                        y_train_val, test_size=0.11,
                                        random_state=42)
train_df, trash_x, y_train_df, trash_y = train_test_split(X_trainp, y_trainp,
                                         test_size=0.0001, random_state=42)
```

Aquí se puede ver cómo queda dividida la data considerando el 100 % del dataframe de entrenamiento.

```
train_df:  (141489, 22, 4)
y_train_df:  (141489, 1)
val_df:  (17490, 22, 4)
test_df:  (17667, 22, 4)
trash_x (15, 22, 4)
```

Según la configuración del parámetro test_size al definir el train_df y el trash_df se puede apreciar la cantidad de data que se guarda en train_df y trash_df, en este caso el 0.0001 va al trash_df y el resto al train_df, esto como se menciona anteriormente es para poder medir el rendimiento de la red con distinta cantidad de data de entrenamiento.

5.3. Equilibrio de clases con metodo SMOTE

El desafío de trabajar con conjuntos de datos desequilibrados es que la mayoría de las técnicas de aprendizaje automático ignorarán la clase minoritaria, y a su vez tendrán un rendimiento deficiente.

Una forma de resolver este problema es sobre muestrear los ejemplos en la clase minoritaria. Esto se puede lograr simplemente duplicando ejemplos de la clase minoritaria en el conjunto de datos de entrenamiento antes de ajustar un modelo. Esto puede equilibrar la distribución de clases sin afectar el resultado ya que no proporciona ninguna información adicional al modelo. (Como recomendación siempre hay que aplicar el balanceo al dataframe de entrenamiento, ya que aplicarlo al dataframe de test puede alterar los resultados.).

Quizás el enfoque más utilizado para sintetizar nuevos ejemplos se llama Synthetic Minority Oversampling Technique, o SMOTE para abreviar.

SMOTE funciona seleccionando ejemplos que están cerca en el espacio de entidades, dibujando una línea entre los ejemplos en el espacio de entidades y dibujando una nueva muestra en un punto a lo largo de esa línea.

Específicamente, primero se elige un ejemplo aleatorio de la clase minoritaria. Luego encuentra los k vecinos más cercanos típicamente k = 5, posterior a eso se elige uno del vecino seleccionado aleatoriamente y se genera un ejemplo sintético en un punto seleccionado aleatoriamente entre los dos ejemplos en el espacio de entidades.

"... SMOTE primero selecciona una instancia de clase minoritaria al azar y encuentra sus vecinos de clase minoritaria k más cercanos. La instancia sintética se crea eligiendo uno de los k vecinos más cercanos b al azar y conectando a y b para formar un segmento de línea en el espacio de entidades. Las instancias sintéticas se generan como una combinación convexa de las dos instancias elegidas a y b"..

Página 47, Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.

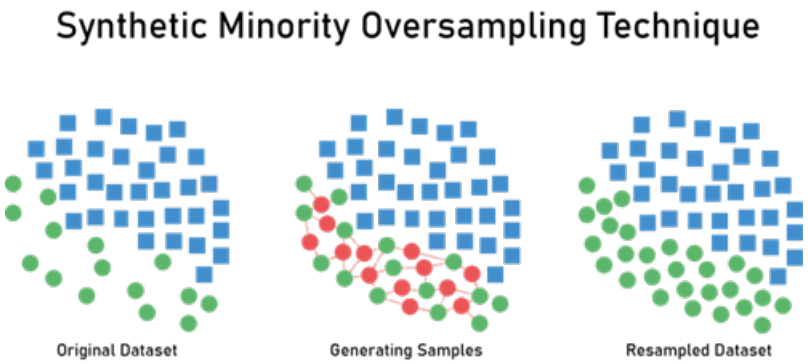


Figura 9: Diagrama que muestra visualmente como el algoritmo SMOTE genera ejemplos sintéticos para combatir el desbalance de clases.[18]

Ahora que el concepto de SMOTE está claro y la división de data está realizada toca hacer el balanceo de la data de entrenamiento ya que como se mencionó anteriormente la data de validación no hay que hacer ningún tipo de modificación ya que con esta vamos a comprobar el rendimiento de las distintas redes luego de su entrenamiento.

En el siguiente código se explica paso a paso cómo se modifica el dataframe de entrenamiento para poder ser balanceado por la función SMOTE.

Se realiza un reshape a train df para dejarlo de la forma (3112758,4).

```
train_df = train_df.reshape(141489*22 , 4)
```

Transforma y_train_df que contiene las salidas a numpy y repetir 22 veces cada dato, esto se hace para poder concatenar de forma correcta con el y_train_df y el train_df ya que esta configurado para cada 22 entradas tenga una salida.

```
y_train_df = y_train_df.to_numpy()
repetitions = 22
y_train_df = np.repeat(y_train_df, repetitions)
#Se transforma de numpy a pandas para poder dar nombre a las
#columnas y poder concatenar ambos dataframes.
train_df = pd.DataFrame(train_df)
y_train_df = pd.DataFrame(y_train_df)
train_df.columns = ['Time','Mag','diffTime','diffMag']
y_train_df.columns = ['Class']
labels_gaia_train = pd.concat([train_df, y_train_df], axis=1)
```

Ahora con el dataframe configurado de forma correcta se ve de esta forma.

```
                Time        Mag   diffTime   diffMag  Class
0        1706.454215  13.316761   0.074029  0.005147      2
1        1706.528244  13.321909  34.301776 -0.162099      2
...              ...        ...        ...       ...    ...
3112753  2128.105280  17.604920   0.074001 -0.241334      3
3112754  2128.179281  17.363586  28.913071 -0.130828      3
```

Visualizamos la data antes del balanceo, como se puede observar la Clase 2 posee casi el 50 % del total.

```
Class=2, n=1409540 (45.283%)
Class=6, n=20614 (0.662%)
Class=0, n=1176186 (37.786%)
Class=4, n=102784 (3.302%)
Class=1, n=323466 (10.392%)
Class=5, n=71302 (2.291%)
Class=3, n=8866 (0.285%)
```

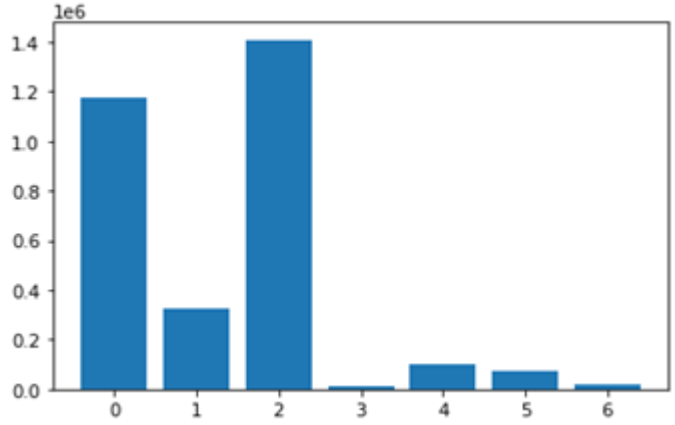


Figura 10: Histograma con la distribución de las clases, similar al mostrado anteriormente, la diferencia es que este cuenta con las 22 muestras de entrada por cada observación.

A continuación vamos a ver el código para poder aplicar SMOTE, realizar el oversample a las clases mas pequeñas y así poder balancear la data.

```
data = labels_gaia_train.values
# dividir en elementos de entrada y salida
X, y = data[:, :-1], data[:, -1]
# etiqueta y codifica la variable de destino
y = LabelEncoder().fit_ transform(y)
# Transforma el dataset y aplica smote para sobremuestrear
oversample = SMOTE()
X, y = oversample.fit resample(X, y)
counter = Counter(y)
for k,v in counter.items():
    per = v / len(y) * 100
    print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# Gráfico luego del sobre muestreo
plt.bar(counter.keys(), counter.values())
plt.show()
```

Podemos observar que ahora las clases representan el mismo porcentaje del total del dataframe de entrenamiento.

```
Class=2, n=1409540 (14.286%)
Class=6, n=1409540 (14.286%)
Class=0, n=1409540 (14.286%)
Class=4, n=1409540 (14.286%)
Class=1, n=1409540 (14.286%)
Class=5, n=1409540 (14.286%)
Class=3, n=1409540 (14.286%)
```

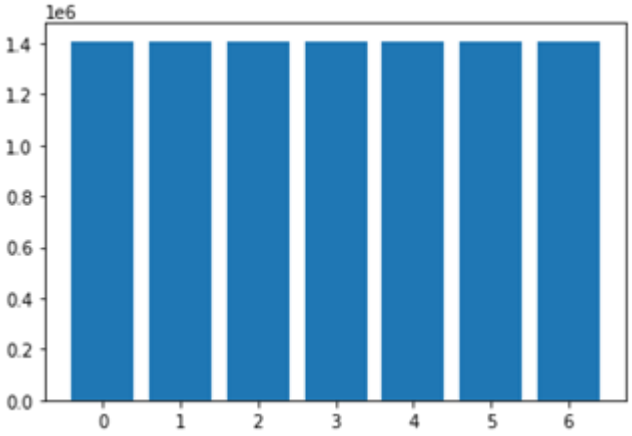


Figura 11: Histograma de las clases luego de aplicar algoritmo SMOTE, en este se ve que las clases están perfectamente balanceadas con este método, lo que nos permite realizar una experimentación con un sesgo más bajo.

Como siguiente paso hay que volver a dejar los dataframes y_train_df y train_df de la misma forma que estaba en un principio se hace un reshape a el train_df para dejarlo de la forma (448490,22,4) y se eliminan las 22 repeticiones que se aplicó a cada dato del y_train_df para tener las 22 entradas y 1 salida.

```
train_df = X
y_train_df = y
train_df = np.asarray(train_df).reshape(448490,22,4)
y_train_df = pd.DataFrame(y_train_df)
y_val_df = pd.DataFrame(y_val_df)
y_test_df = pd.DataFrame(y_test_df)
y_train_df= y_train_df[y_train_df.index % 22 == 0]
train_df:  (448490, 22, 4)
y_train_df:  (448490, 1)
val_df:  (17490, 22, 4)
y_val_df:  (17490, 1)
```

6. Experimentos

El objetivo principal de los experimentos que se realizarán es comprobar la factibilidad de utilizar aprendizaje autosupervisado por medio de transfer learning, es decir, utilizar una red pre entrenada para una tarea en una diferente.

Para lograr este objetivo se procesa la data de la siguiente manera, un data frame utilizando el método data2vec para aprendizaje no supervisado, y el siguiente con un método de clasificación balanceada para el aprendizaje supervisado. Primero se tomará como base el rendimiento que obtengan los algoritmos de clasificación, que será probado con tres variaciones: Baseline, LSTM, GRU. El baseline nos proveerá una base con la que podremos comparar nuestro rendimiento, pero no será útil para los experimentos. Cada una de las redes neuronales LSTM y GRU será probada con el 100 %, 50 %, 25 %, 10 % y 5 % de la data, como se explicó anteriormente y se medirá la certeza que obtenga cada red con cada dataset. Es decir, se generarán 5 valores de accuracy por cada una de las 2 redes.

Luego se entrenará ambas redes neuronales con el método data2vec y se guardará el aprendizaje de los modelos para ser utilizado posteriormente con la tarea de clasificación, es decir, se entrenará para una tarea y luego será utilizado para realizar una diferente. El objetivo, es comprobar si es factible obtener un mejor resultado con una menor cantidad de muestras a la hora de clasificar, lo que podría ayudar a realizar trabajos de clasificación con menos cantidad de datos etiquetados, que como se mencionó al comienzo del trabajo, es data muy costosa de obtener.

Otra experimento que se realizará será sobre la cantidad de features que a utilizar. El dataframe viene con dos features, y en esta etapa se experimentará agregando dos más, estas serán: *timediff* y *magdiff*. Que corresponden a la diferencia de cada fila con su sucesora, como se explicó en el preprocesamiento de los datos. La idea es que agregando más características se pueda aprovechar una heurística que ayude a la red a encontrar los pesos óptimos para mejorar su rendimiento. Se comparará los rendimientos de ambas redes con ambos métodos y se medirá el tiempo de cómputo además del accuracy obtenido en cada experimento. Además, se busca encontrar si es factible realizar este entrenamiento previo para mejorar el rendimiento.

Diagrama experimentos

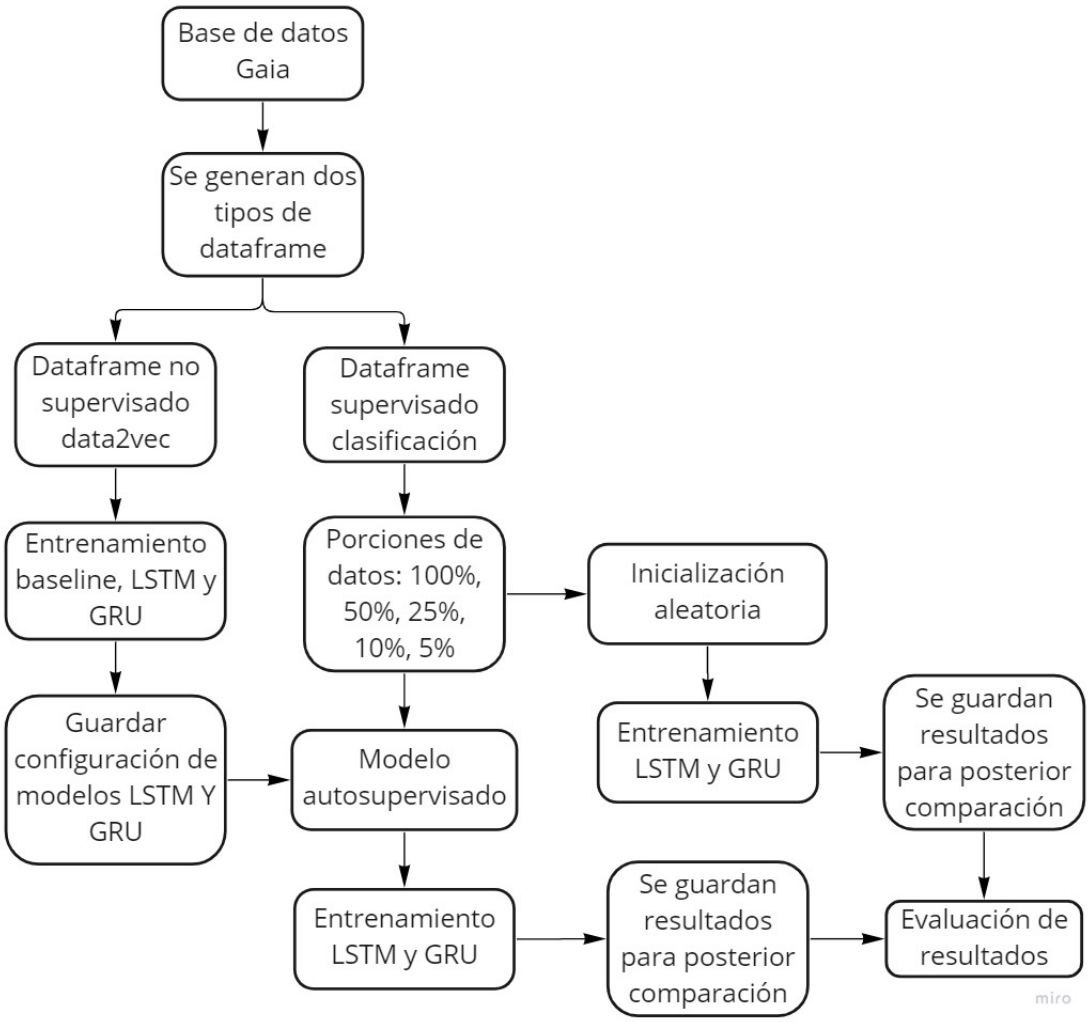


Figura 12: Diagrama de bloques que representa los distintos procesos que se llevarán a cabo en la experimentación, desde el preprocesamiento de la base de datos a los dos caminos que harán posible la comparación de los resultados obtenidos.

6.1. Arquitectura de la red

Para seleccionar la arquitectura usada en las redes neuronales se armó un dataframe con 8000 muestras de prueba para poder probar el rendimiento de las distintas configuraciones de la red. En la siguientes tablas se muestra el rendimiento de cada configuración según capas y dropout midiendo mean absolute error en cada tarea.

LSTM 1 layer (MAE)			
Layers	droptout=0,3	droptout=0,1	no dropout
16 nodes	0,0332	0,0324	0,0229
32 nodes	0,0380	0,0265	0,0243
64 nodes	0,0368	0,0324	0,0279
128 nodes	0,0304	0,0467	0,0242

LSTM 2 layer (MAE)			
Layers	droptout=0,3	droptout=0,1	no dropout
32 - 16 nodes	0,0364	0,0337	0,0324
64 - 32 nodes	0,0535	0,0439	0,0324
128 - 32 nodes	0,0268	0,0865	0,0324
128 - 64 nodes	0,1018	0,1018	0,0387

LSTM 3 layer (MAE)			
Layers	droptout=0,3	droptout=0,1	no dropout
64 - 32 - 16 nodes	0,0403	0,0319	0,0311
64 - 64 -32 nodes	0,0326	0,0315	0,0330
128 - 64 - 16 nodes	0,0319	0,0313	0,0310
128 - 64 - 32 nodes	0,0348	0,0443	0,0302

Como se observa, en las mediciones el valor de mae más bajo está en la configuración de 3 capas de 128-64-32 sin dropout, pero para evitar posible overfitting en los experimentos se decidió usar la segunda configuración mas rápida con 3 capas de 128-64-16 con dropout 0,1.

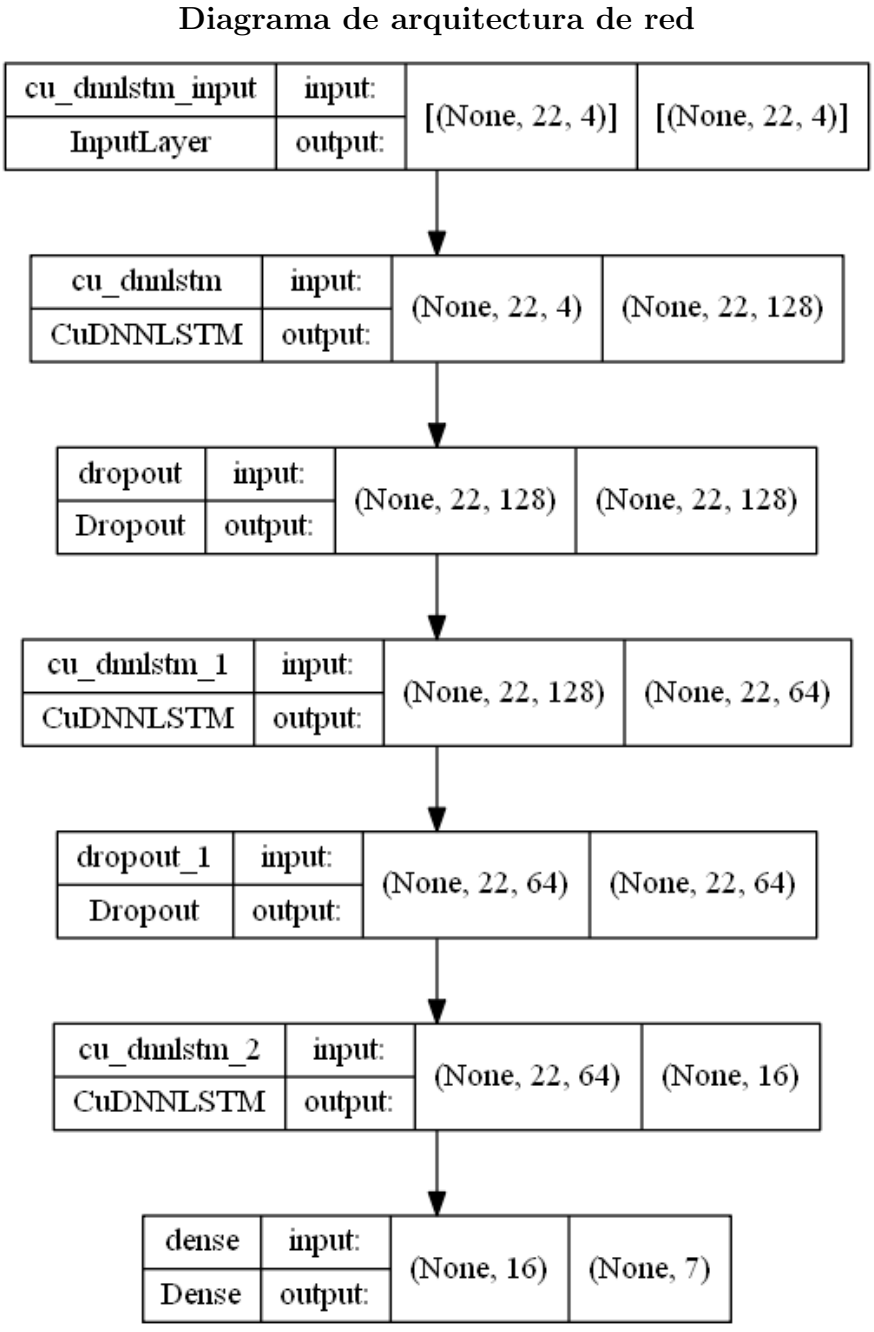


Figura 13: Diagrama que representa la arquitectura de la red utilizada para aprendizaje autosupervisado. En este caso se muestra el modelo LSTM.

A continuación se muestra el código para construir el modelo con la arquitectura seleccionada. La capa de entrada constará de 22 unidades en ambos casos, no supervisado o data2vec y autosupervisado. A diferencia de la capa de salida que para clasificación constará de 7 unidades como muestra el código, y en data2vec se tiene 5 unidades de salida, por el motivo que se explicó en su sección correspondiente.

```
lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(128, return_sequences=True),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.LSTM(16, return_sequences=False),
    tf.keras.layers.Dense(units=7 , activation = 'softmax')
])
```

6.2. Preparación de datos

Otro experimento que se uso para optimizar el rendimiento fue comparar cómo se comportan las redes al usar distinta cantidad de features como entrada. Se probaron tres experimentos distintos con (n,2) *time* y *mag*, (n,3) *time*, *mag* y *timediff* y (n,4) *time*, *mag*, *timediff*, *magdiff*.

	(n,2) 100 %	(n,3) 100 %	(n,4) 100 %
Acc LSTM	0.9411	0.9443	0.9556
Acc GRU	0.9245	0.9333	0.9340
Tiempo LSTM	192,1 min	200,3 min	250,8 min
Tiempo GRU	161,8 min	162,7 min	184,8 min

Se observa en la tabla de resultados que la red entrenada con menos features demora menos, pero su rendimiento es peor, por lo que se decidió usar la opción con (n,4), ya que la diferencia de tiempo era más baja en comparación con la mejoría del rendimiento de la red.

6.3. Experimento no supervisado

Al comenzar el trabajo se buscó una manera de vencer el *baseline*. El primer intento fue utilizar la desviación estándar de la serie como salida, pero el *baseline* lo supera sin problema. Luego se implementa el método data2vec con el que se utilizan varios pasos de salida, lo que dificulta la tarea en general, pero se esperaba que la red neuronal tuviera un mejor comportamiento frente a esta tarea que el *baseline*, como se ve a continuación. Además,se realiza la prueba considerando la red linear, pero esta no rinde mejor que las redes neuronales LSTM y GRU, como se espera. Una vez finalizado estos experimentos se guarda la configuración de cada modelo LSTM y GRU para posteriormente ser utilizado en la tarea de clasificación.

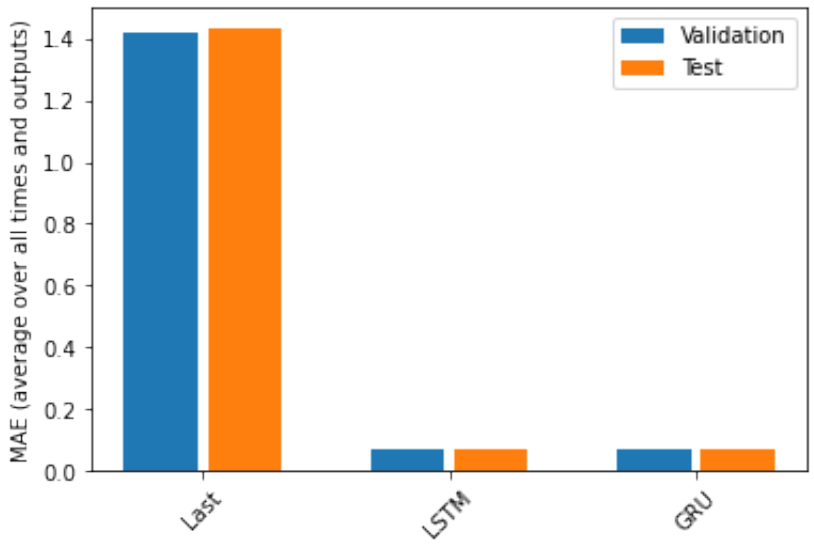


Figura 14: Rendimiento de los distintos modelos en la fase de pruebas de arquitectura para la red utilizando el modelo data2vec.

Acá se muestran los resultados obtenidos por las tres redes en el primer experimento data2vec, no supervisado.

```
Last     : 1.4328
LSTM     : 0.0723
GRU      : 0.0726
```

Predicción modelo baseline

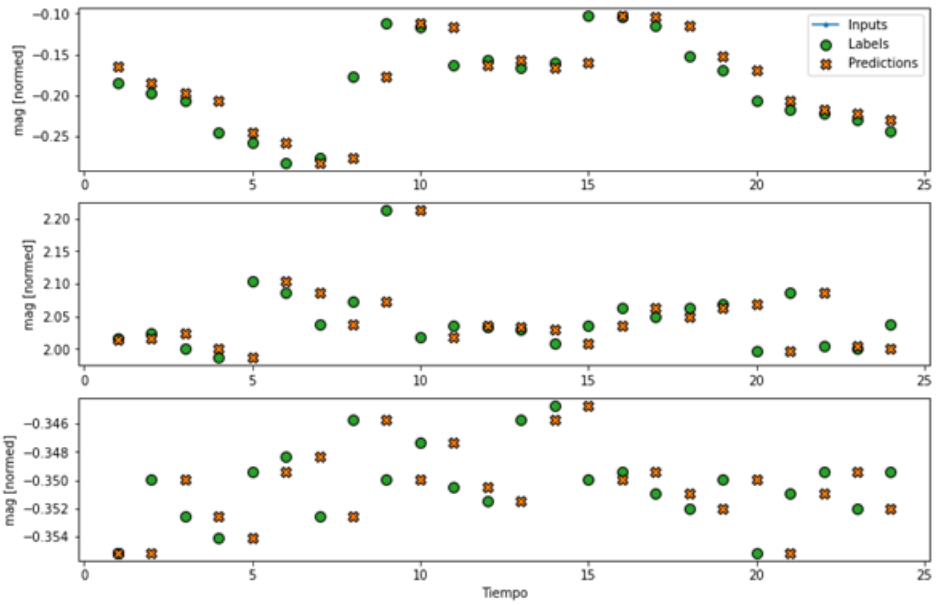


Figura 15: El modelo predice que la siguiente salida será igual a la actual, por eso se ve un desfase de un paso. Es el modelo a batir.

Predicciones paso simple LSTM

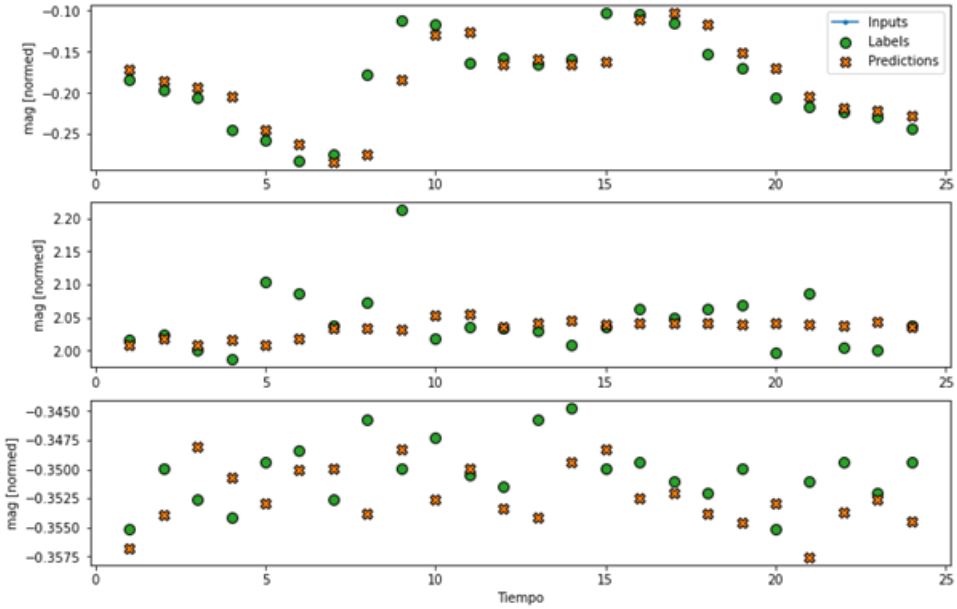


Figura 16: En esta figura se logra apreciar como la red neuronal aprende a predecir la siguiente salida. Se observa que la predicción no es aleatoria y se acomoda mejor a los cambios que el ejemplo anterior.

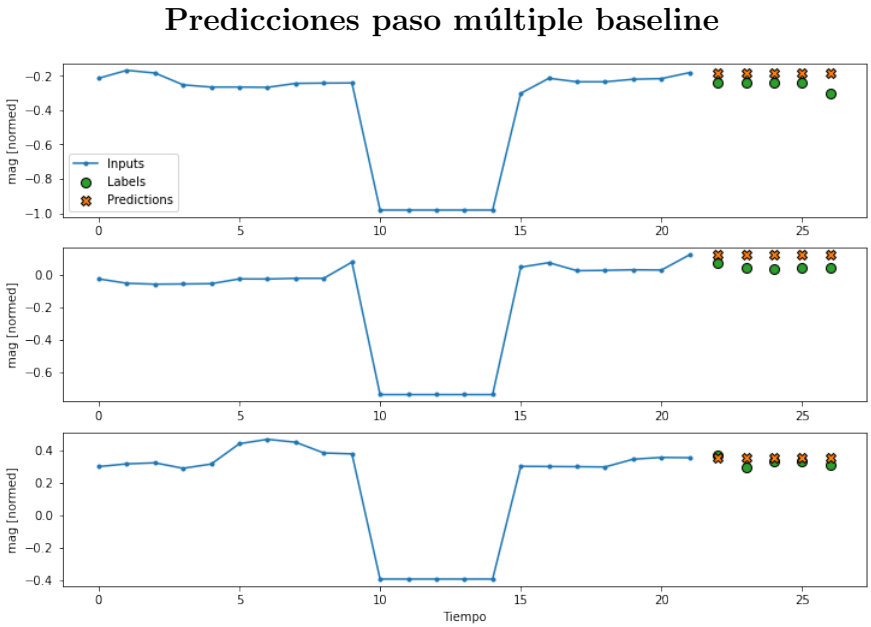


Figura 17: Predicciones realizadas por el modelo más básico, repite la última salida la cantidad de veces necesarias para completar el requerimiento.

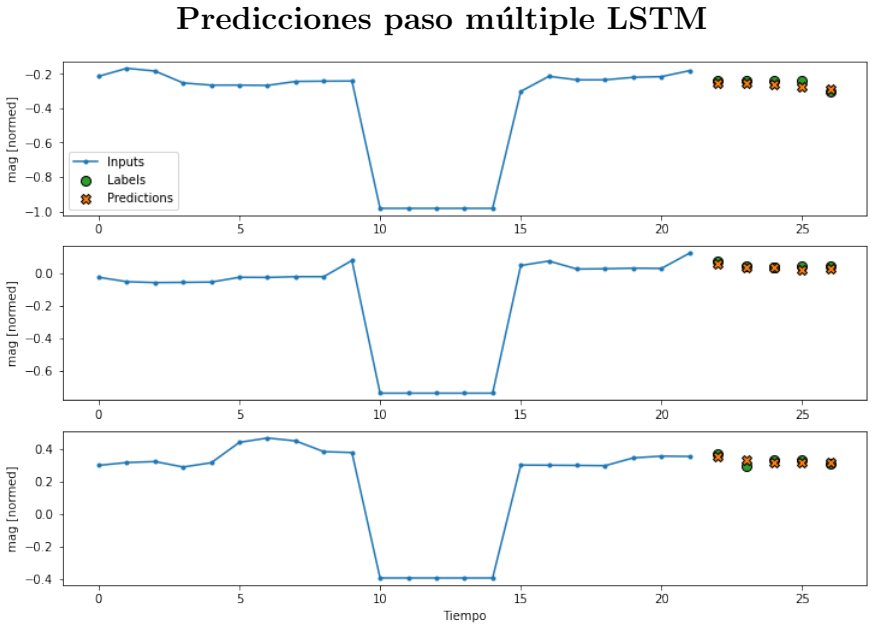


Figura 18: En esta figura se logra apreciar como la red neuronal aprende a predecir la siguiente salida. Se observa que la predicción no es aleatoria y se acomoda mejor a los cambios que el ejemplo anterior.

6.4. Experimento autosupervisado

Luego de entrenar el modelo este debe ser guardado y cargado para realizar una tarea distinta para la que fue entrenado, con la intención de que este pueda realizar una tarea distinta sin tener que aprender los patrones de datos desde cero. Para esto luego de seleccionar la mejor arquitectura utilizando el método data2vec guardamos el modelo de la siguiente manera:

```
model.save("model_data2vec.h5")
```

Para posteriormente ser cargado y entrenado como se muestra a continuación:

```
from keras.models import Model
#Buscar que coincida número de params con summary
model2 = Model(loaded_model.input, loaded_model.layers[-2].output)
model2.summary()
x=tf.keras.layers.Dense(units=7,activation='softmax')
                        (loaded_model.layers[-2].output)
model3 = Model(loaded_model.input, [x])
model3.summary()
history = compile_and_fit(model3,
                        train_df,
                        y_train_df,
                        val_df,
                        y_val_df)
val_performance = {}
performance = {}
val_performance['LSTM'] = model3.evaluate(val_df, y_val_df)
performance['LSTM'] = model3.evaluate(test_df, y_test_df, verbose=1)
```

7. Resultados

Como se explicó en la sección anterior, se obtuvieron los resultados del mejor modelo para realizar el entrenamiento previo, es decir, el no supervisado. El método que mejor resultado nos entregó en este aspecto fue el método data2vec. Luego de terminar el entrenamiento con toda la data en este método, se guardó el modelo y se se prosiguió a comparar el rendimiento de la red con una inicialización aleatoria versus una inicialización auto supervisada, o sea, el modelo guardado anteriormente. La comparación de los resultados de los distintos experimentos se realizará de la siguiente manera para su mejor comprensión.

- Se mostrará el resultado obtenido con cada porción de los datos para ambas redes neuronales (LSTM y GRU).
- Se mostrará el resultado obtenido para la misma cantidad de datos siguiendo los dos métodos. Estos son clasificación y auto supervisada.
- Se comparará ambos resultados en sus dos variantes.
- Se prosigue a realizar el mismo análisis con la porción de datos siguiente.

Las porciones de datos utilizadas como se mencionó en el capítulo anterior serán: 100 %, 50 %, 25 %, 10 % y 5 %. Se comenzará con la más pequeña para ir avanzando hacia el total de los datos.

Tabla de resultados experimento de Clasificación versus Autosupervisado según el tamaño del dataframe de entrenamiento y rendimiento.

Accuracy	Clasificación		Autosupervisado	
Porción	LSTM	GRU	LSTM	GRU
5 %	0,6903	0,7379	0,7752	0,7600
10 %	0,7063	0,7521	0,7812	0,7800
25 %	0,7874	0,7671	0,7668	0,9093
50 %	0,9313	0,9052	0,8201	0,9111
100 %	0,9556	0,9340	0,9113	0,9287

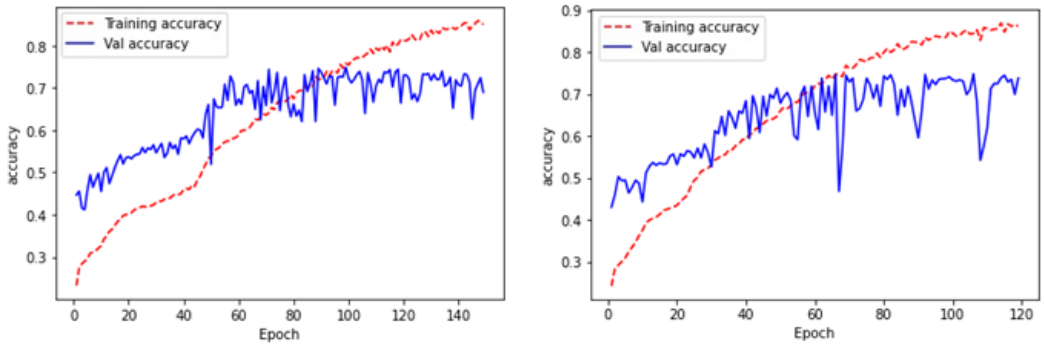
Tabla de resultados experimento de Clasificación versus Auto supervisado según el tamaño del dataframe de entrenamiento y tiempo de ejecución.

Tiempo (min)	Clasificación		Autosupervisado	
Porción	LSTM	GRU	LSTM	GRU
5 %	14,9	9,9	15,6	10,8
10 %	20,3	13,7	23,0	15,3
25 %	100,81	49,3	258,3	241,7
50 %	152,25	86,7	491,7	483,3
100 %	250,8	184,8	708,3	591,6

7.1. Resultados con 5 %

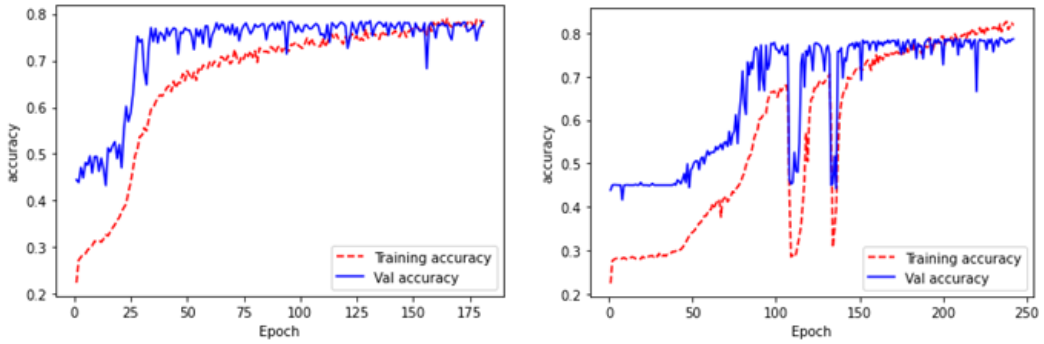
Como se observa en las tablas obtenidas a partir de los resultados, se puede extraer que el aprendizaje autosupervisado tiene un rendimiento mejor que la clasificación regular, en un tiempo de cómputo similar. Pero como se verá en el resto de los resultados, esto es sin contabilizar el entrenamiento previo que realizó el modelo, aunque este es sólo realizado una vez para todos los experimentos.

Se observa una mejoría de sobre un 8 % con el aprendizaje autosupervisado, llegando a casi un 80 % de accuracy en LSTM. En GRU la mejora es menor, pero se atribuye principalmente a que rindió mejor en clasificación con inicialización aleatoria que LSTM. Se observa que 5 % GRU autosupervisado tuvo unas caídas pero se pudo recuperar.



(a) LSTM (b) GRU

Figura 19: Clasificación 5 %

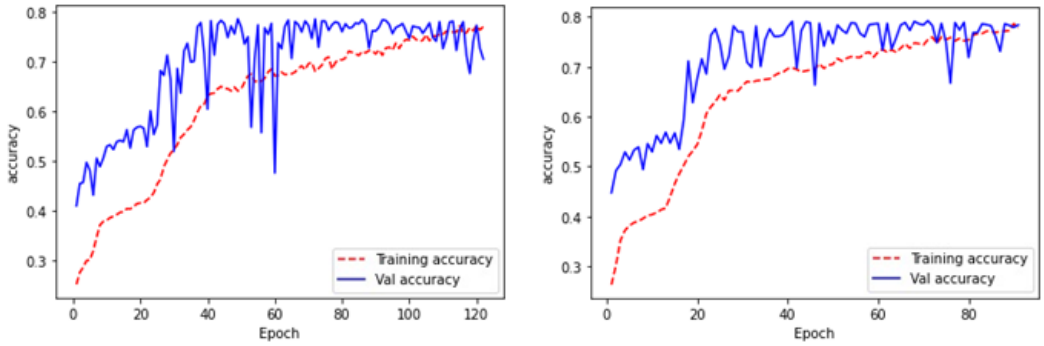


(a) LSTM (b) GRU

Figura 20: Autosupervisado 5 %

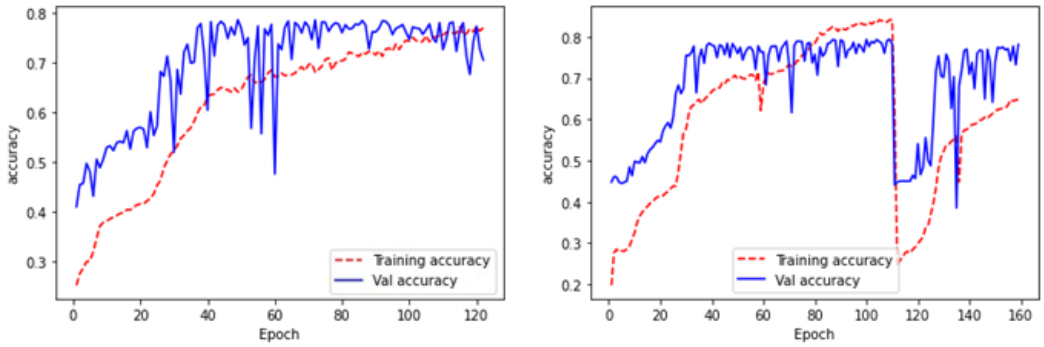
7.2. Resultados con 10 %

La mejora de los resultados en un 10 % también es notoria, sobre todo LSTM, ya que la diferencia entre ambos métodos es mayor, pero el resultado con autosupervisado es similar entre ambas redes. Nuevamente los tiempos de cómputo son mejores en GRU, ya que es una red con una puerta menos (indicado en capítulo 6). Un resultado interesante es que podemos obtener un 78 % de accuracy en 15,3 minutos de cómputo, utilizando GRU autosupervisado, casi igualando el 78,74 % de accuracy que tiene LSTM 25 % de clasificación con 100,81 minutos de cómputo. La generalidad nos indica un aumento constante en el accuracy hasta lograr el aprendizaje esperado y determinado en nuestro early_stopping.



(a) LSTM (b) GRU

Figura 21: Clasificación 10 %

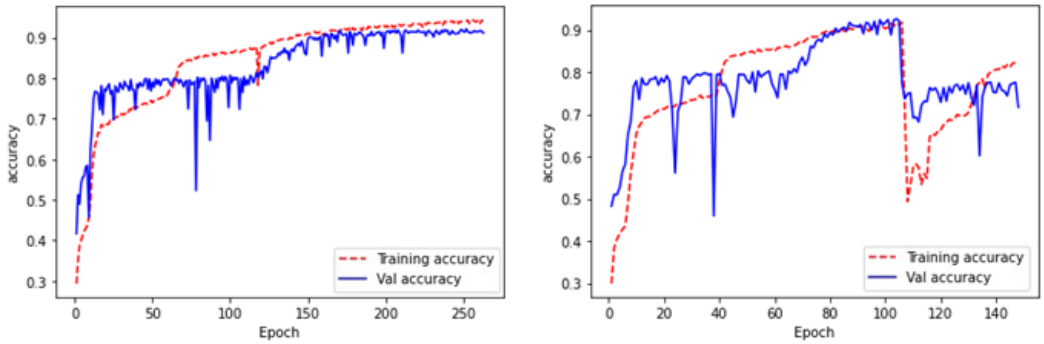


(a) LSTM (b) GRU

Figura 22: Autosupervisado 10 %

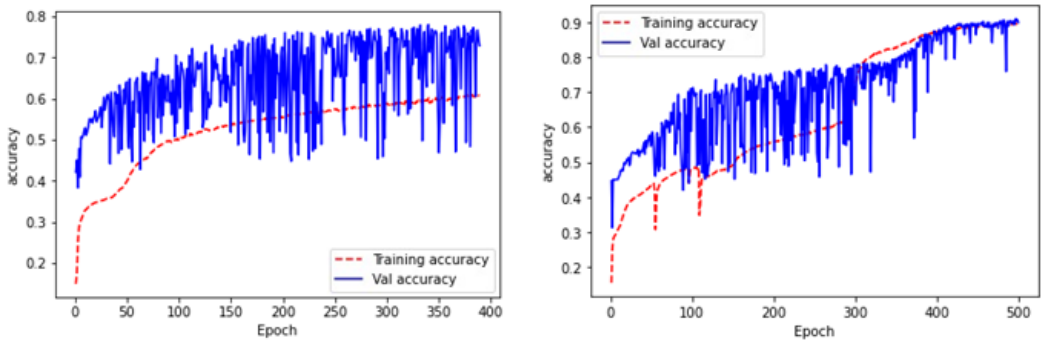
7.3. Resultados con 25 %

En esta parte notamos que las redes comienzan a tener un comportamiento divergente. La razón de aprendizaje de LSTM deja de escalar como la de GRU y vemos resultados interesantes. GRU alcanza sobre el 90 % de *accuracy* con un 25 % de los datos en autosupervisado, superando en más de un 14 % el rendimiento de clasificación. Pero los tiempos de cómputo aumentaron mucho ya que se tuvo que modificar el parámetro de *learning_rate* que será explicado en las conclusiones. LSTM por primera vez no supera el rendimiento de clasificación en autosupervisado. El comportamiento dispar de ambas redes apunta a que GRU es la mejor candidata en esta ocasión. Aumenta la fluctuación de *val_accuracy* en experimentos autosupervisados variando a veces casi en 30 puntos.



(a) LSTM (b) GRU

Figura 23: Clasificación 25 %

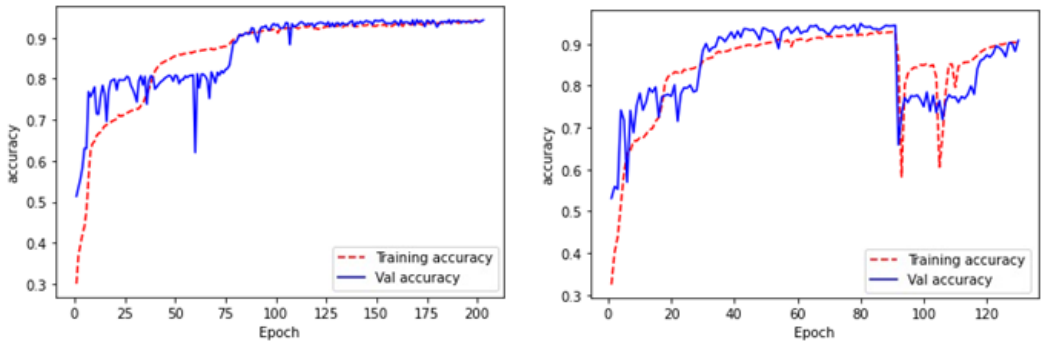


(a) LSTM (b) GRU

Figura 24: Autosupervisado 25 %

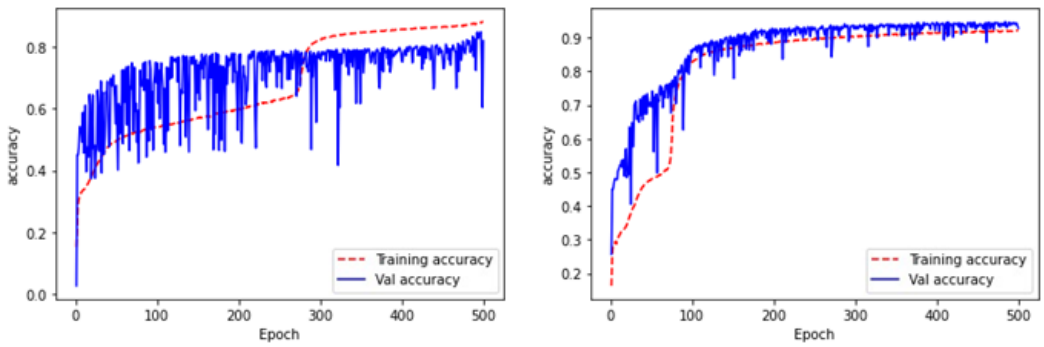
7.4. Resultados con 50 %

Siguiendo la tendencia salta a la luz que LSTM no está rindiendo mejor con el aprendizaje autosupervisado, a diferencia de GRU que si bien no tiene la misma mejora tampoco empeora, además, sigue mostrando mejores resultados que el experimento base de clasificación. Se ven bajas en el aprendizaje de clasificación GRU y luego le cuesta recuperar su rendimiento anterior, pero esto se soluciona en el aprendizaje autosupervisado luego de modificar algunos parámetros en la función *compile_and_fit*.



(a) LSTM (b) GRU

Figura 25: Clasificación 50 %

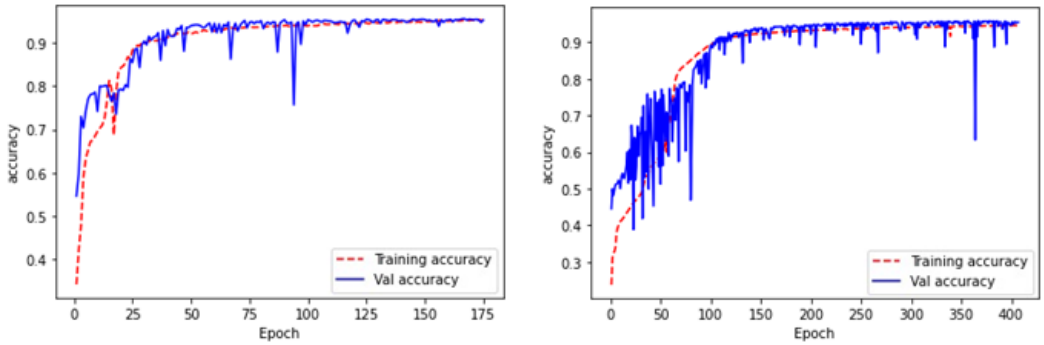


(a) LSTM (b) GRU

Figura 26: Autosupervisado 50 %

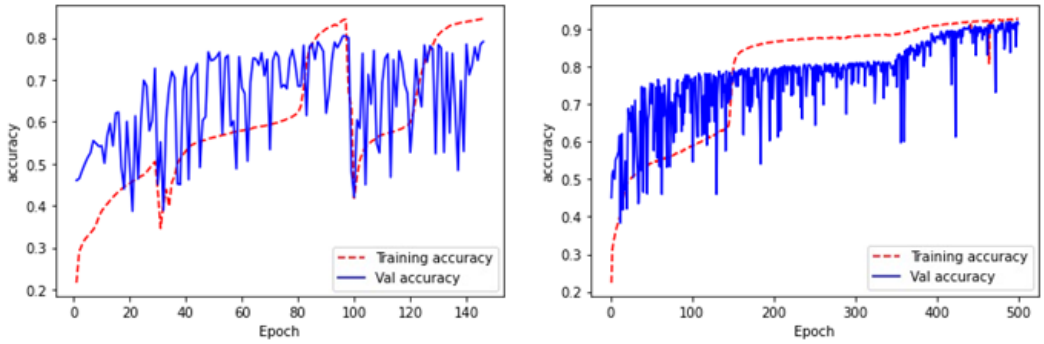
7.5. Resultados con 100 %

Cuando es considerada toda la data notamos también el mejor resultado posible. LSTM obtiene un 95,56 % de *accuracy* con toda la data, pero al costo de una data bien etiquetada y sin hacer uso del aprendizaje autosupervisado. Por otra parte tenemos que el rendimiento más bajo también es LSTM pero en el caso de autosupervisado, siendo casi más bajo que el 50 % de la data con GRU en autosupervisado.



(a) LSTM (b) GRU

Figura 27: Clasificación 100 %



(a) LSTM (b) GRU

Figura 28: Autosupervisado 100 %

8. Conclusiones

Antes de hablar de las conclusiones es importante aclarar unos puntos y que pueden haber interferido en el desarrollo de la última parte de los experimentos, esta es, cuando se aplicó el *"transfer learning"* a la tarea de clasificación para llevar a cabo el motivo de esta tesis, el aprendizaje autosupervisado. Una vez realizada la primera parte de los experimentos que servirían como base comparativa para el resto del avance del trabajo, se llevó a cabo el objetivo del mismo.

Encontrándonos con un comportamiento errático de la red neuronal a la hora de aprender patrones para resolver el problema. Este comportamiento no se había presentado anteriormente en ninguno de los experimentos, por lo que la primera acción fue revisar que el código de transferencia de aprendizaje del modelo estuviera bien implementado. En orden de realizar lo anterior había una inconsistencia, y es que los modelos que acumulaban mayor cantidad de datos, es decir, los modelos con una porción del 25 % y más, eran los que estaban presentando los problemas, pero con el mismo código las porciones más pequeñas se estaban desenvolviendo bien, obteniendo así el rendimiento esperado de un aprendizaje autosupervisado, superando con creces el modelo de clasificación regular, descartando así que el problema estuviera en la implementación del código de transferencia de aprendizaje. Se estaban obteniendo curvas de aprendizaje como la que se muestra a continuación, sin que el algoritmo fuera capaz de retomar *checkpoint* definido por nosotros mismos para así seguir disminuyendo el valor de la *función de pérdida*.

Se determinó luego de investigar el problema que el parámetro *learning_rate* debía ser alterado para que la red neuronal no utilizara el *optimizer* que traía y utilizara un valor fijo. Esto quería decir que la red iba a aprender a una razón más lenta, pero también más segura. Esto tiene que ver con la función de pérdida y el descenso de gradiente, en la que es probable que la función de pérdida quede atrapada en un valle o mínimo local, y sea difícil escapar del mismo haciendo que finalmente la red deje de aprender y termine en un resultado no óptimo. Al modificar el *learning_rate* y hacerlo constante aumentamos considerablemente el tiempo de cómputo, pero también evitamos caer en uno de estos mínimos locales y obligamos a la red a avanzar en la dirección que se presume correcta.

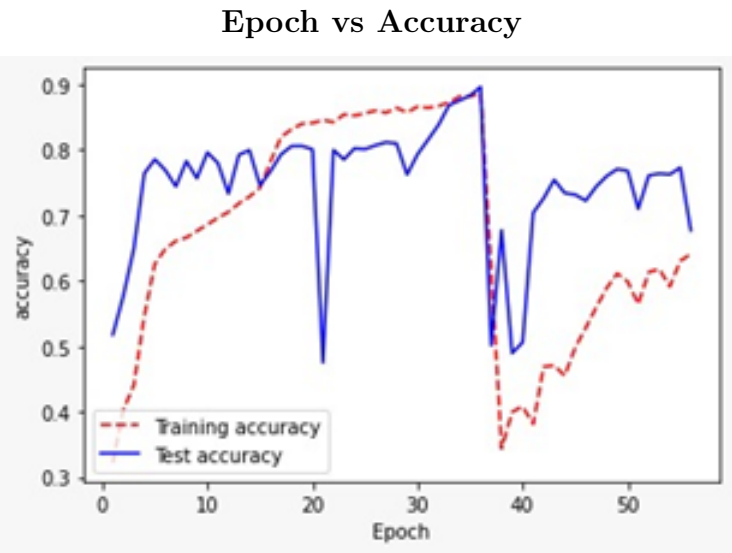


Figura 29: Se observa que aproximadamente en la epoch 35 la red disminuye considerablemente su rendimiento y no logra recuperarse.

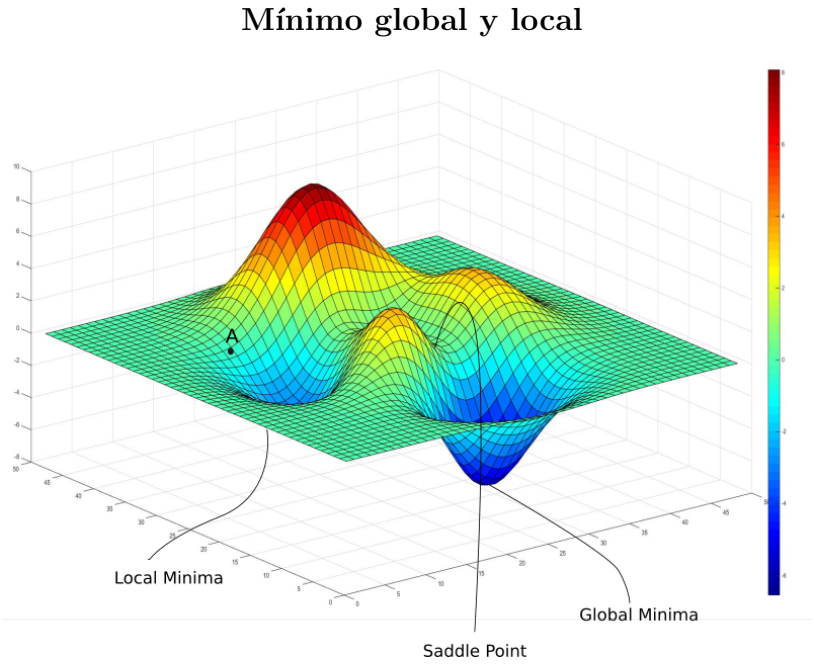


Figura 30: Representación tridimensional de mínimos y máximos locales. Se puede entender que la función de pérdida nos acerque a un mínimo, pero que este no necesariamente sea el óptimo.[19]

Una vez sorteado este problema, el obstáculo pasa a ser que no todas las redes estarán medidas de una manera exacta, ya que al final del experimento es que se presentó el problema. Y tomando en consideración la duración de cada experimento, se hacía inviable realizar todos los experimentos anteriores utilizando exactamente la misma metodología. Si bien se realizó una modificación en alguno de los parámetros, estos no tienen que ver con el proceso de aprendizaje o el método utilizado, más bien con la taza de aprendizaje que, si bien puede afectar al resultado final, queda constatado el cambio y es considerado a la hora de evaluar los resultados.

El resultado esperado era que la red se comportara de mejor manera, es decir, un mejor rendimiento en su medición de accuracy, cuando la inicialización del modelo no fuera aleatorio, sino, que fuera inicializado con los pesos del aprendizaje anterior no supervisado utilizando el método data2vec. Este resultado se cumplió de forma parcial. En algunos de los experimentos funcionó como se hipotetizó y en otros no. Pero el patrón de este comportamiento está dado por la cantidad de datos entregados a la red para que sea entrenada.

Cuando se entregaron largos volúmenes de datos no se logró mejorar el rendimiento de la red respecto a la clasificación inicializada aleatoriamente, debido probablemente, a que al tener grandes cantidades de datos la red logra aprender los patrones de aprendizaje y predecir una salida con un éxito de 95,56 %, lo que es bastante respetable. Esto utilizando una red compleja como es LSTM en un tiempo de aprendizaje de 4 horas y 12 minutos aproximadamente. El desafío del experimento iba a ser analizar el rendimiento de las redes neuronales LSTM y GRU con pocos datos, y además comparar el rendimiento que obtuvo cada una en los distintos experimentos.

8.1. LSTM

Para LSTM tenemos que el mejor rendimiento es logrado con toda la data en clasificación aleatoria, y su peor rendimiento es el 5 % de los datos en clasificación aleatoria. Es decir, tenemos el mayor espectro de rendimientos en la misma red. Podemos concluir de esto que LSTM se comportará mejor mientras más datos sean proveídos, independientemente del método de

inicialización del modelo.

La relación costo/rendimiento más alta para LSTM es con el 5 % de los datos obteniendo un 77,52 % de *accuracy* en 15,6 minutos de cómputo, superando al 25 % de rendimiento en menos de un décimo del tiempo. Esto se debe, como se explicó en un comienzo, a la variación en la razón de aprendizaje a la que se vio forzado el trabajo para evitar pérdidas muy notables en el *accuracy*.

Según los resultados obtenidos, este método tiene validez de uso siempre y cuando la data etiquetada de la que se disponga sea baja respecto a la cantidad no etiquetada. En ese caso y para este tipo de problema utilizando la arquitectura propuesta se prevé un buen resultado en el rendimiento, donde se supera la competencia por casi un 10 %.

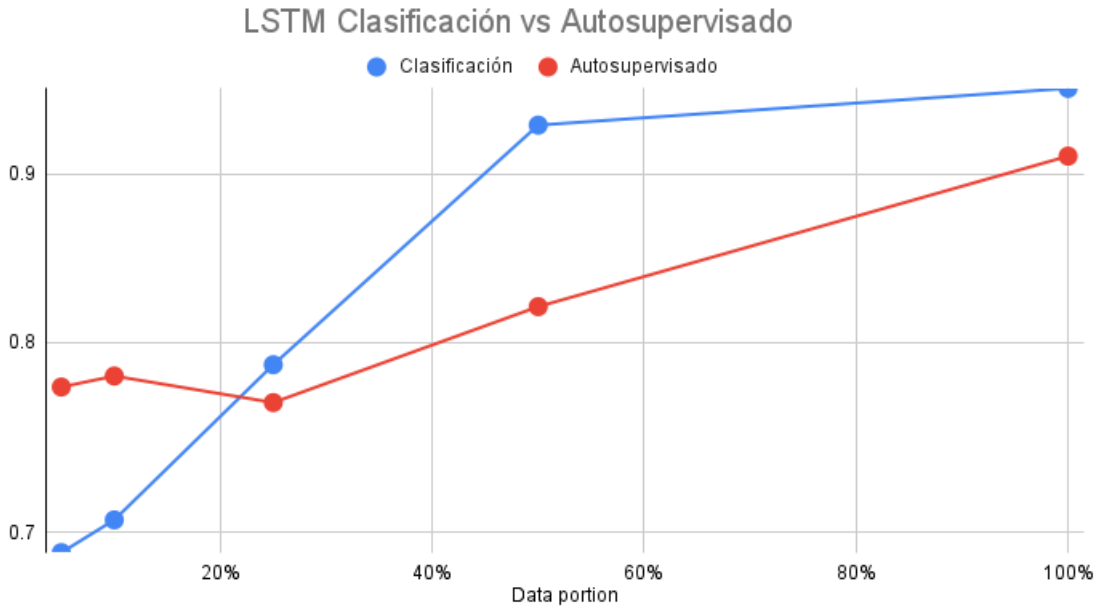


Figura 31: Accuracy vs Porción de datos para GRU

8.2. GRU

GRU obtiene buenos resultados en situaciones especiales. Al tomar en consideración la arquitectura de la red y compararla con LSTM, trabajo realizado en capítulos 6 y 7, se entiende que es una red neuronal más simple que LSTM, pero que tiene sus beneficios por lo mismo. Se observa que cuando tiene poca data, el comportamiento de la red no es notoriamente mejor que la inicialización aleatoria, pero sigue siendo una red muy rápida de aprender y que sin duda lleva a cabo una tarea cuando no se busca la perfección, pero sí una buena aproximación a la respuesta.

Además, GRU entrega la mejor relación costo/rendimiento de todo el trabajo con el 25 % de los datos en aprendizaje autosupervisado, donde llega al 90,93 % de accuracy en 241 minutos de cómputo, esto explicado por la baja razón de aprendizaje, condición que comparten ambas redes. Pero el punto a destacar es la alta certeza del modelo con un cuarto del total de datos a disposición, donde destaca el potencial de la red neuronal GRU.

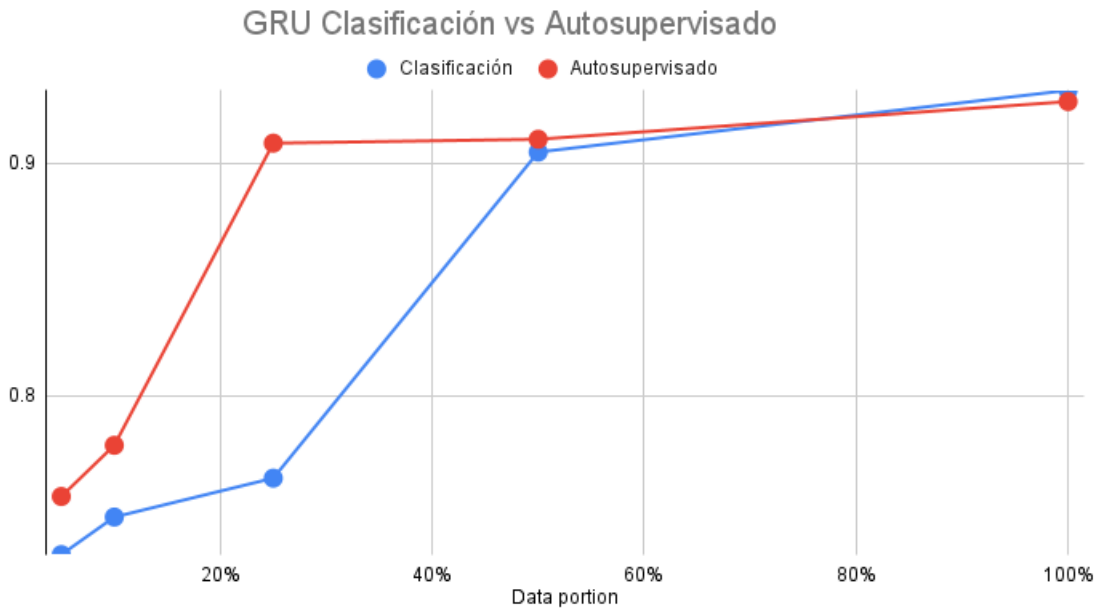


Figura 32: Accuracy vs Porción de datos para GRU

9. Limitaciones del trabajo y trabajos futuros

Uno de los problemas comunes y asociados al aprendizaje profundo es la gran cantidad de recursos necesarios para que este tenga un resultado óptimo, siento en ocasiones un resultado prometedor como se ha demostrado en grandes proyectos y colaboraciones de la industria tecnológica actual como por ejemplo GPT-3.

En una primera instancia se lograron buenos resultados cuando se trabajó con las porciones del dataset más pequeñas, debido justamente a que estas no requieren un procesamiento de los datos que consuma tantos recursos y pudieron ser realizados sin mayor problema, y aunque el foco del trabjo iba hacia estos resultados que presentaban la menor cantidad de datos, de igual manera se esperaba ver el comportamiento del algoritmo cuando la cantidad de datos fuera la mitad o todos los datos.

En este punto es donde se encontró la mayor limitación del trabajo y es que la capacidad de cómputo de la que se disponía no fue capaz de entrenar en un tiempo prudente como para repetir los experimentos o incluso llegar a realizarlos una sola vez.

Al limitar la cantidad de iteraciones a 500, se esperaba que estas fueran más que suficientes para que el entrenamiento de las redes se detuviera sólo utilizando el parámetro de *patience*, pero no fue el caso. Luego de estudiar el comportamiento de la red y los gráficos obtenidos se puede observar que las redes que utilizan aprendizaje autosupervisado podrían haber seguido aumentando su *accuracy*.

Relacionado con lo anterior es que se pueden asociar los trabajos futuros, ambas propuestas tienen que ver con la limitación mencionada anteriormente.

El primer trabajo futuro tiene que ver con utilizar esta misma arquitectura de redes y metodología explicada, pero limitar las iteraciones a 1500 con un *patience* de 50. Se espera que con esta configuración la red logre utilizar el parámetro *early_stopping* para lograr su mejor rendimiento. Todo esto con el parámetro *learning_rate* en 0,01, como fue utilizado en este trabajo.

El siguiente trabajo propuesto sería replicar este experimento con la base de datos OGLE-III, que está compuesto por una estructura similar, pero cada objeto de estrella variable contiene más observaciones. Se propone utilizar 360 entradas, y un parche en data2vec del 10 %, es decir 36 salidas.

Bibliografía

[1] learnchile. Why You Should Study Astronomy in Chile. 2020.

[2] NASA/JPL-Caltech/STScI. Little Galaxy Explored. https://www.spitzer.caltech.edu/image/ssc2010-02a1-little-galaxy-explored, 2010. Accessed: 2022-20-05.

[3] E. V. ; Durlevich O. V. ; Kireeva N. N. ; Pastukhova E. N. Samus, N. N. ; Kazarovets. Vizier Online Data Catalog: General Catalogue of Variable Stars (Samus+, 2007-2017), 2009.

[4] Eric Rothstein Morris Ralf C. Staudemeyer. Understanding lstm – a tutorial into long short-term memory recurrent neural networks., 2019.

[5] Christopher Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2022-25-05.

[6] Rahul Dey and Fathi M. Salem. Gate-variants of gated recurrent unit (gru) neuralnetworks., 2017.

[7] Vijaysinh Lendave. LSTM vs GRU in Recurrent neural network. https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/. Accessed: 2022-25-05.

[8] Kundu R. The beginner's guide to self-supervised learning. https://www.v7labs.com/blog/self-supervised-learning-guide, 2022. Accessed: 2022-01-05.

[9] Joseph W. Richards Joshua S. Bloom. Data mining and machine-learning in time-domain discovery classification., 2011.

[10] López M. Sarro L. M., Debosscher J. and Aerts C. Automated supervised classification of variable stars. II. Application to the OGLE database., 2018.

[11] D.-W. Kim J.B. Marquette P.Tisserand. K. Pichara, P. Protopapas. An improved quasar detection method in EROS-2 and MACHO LMC data sets., 2012.

[12] Brandon Sim Ming Zhu Rahul Dave Nicolas Castro Karim Pichara Isadora Nun, Pavlos Protopapas. FATS: FEATURE ANALYSIS FOR TIME SERIES, 2015.

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

[14] Márcio Catelan Pavlos Protopapas Carlos Aguirre Fatemeh Nikzat Ignacio Becker, Karim Pichara. Scalable End-to-End Recurrent Neural Network for Variable Star Classification., 2020.

[15] M. K. Szymański Ł. Wyrzykowski K. Ulaczyk R. Poleski P. Pietrukowicz S. Kozlowski D. Skowron J. Skowron P. Mróz M. Pawlak K. Rybicki A. Jacyszyn-Dobrzeniecka I. Soszyński, A. Udalski. The OGLE Collection of Variable Stars. Classical, Type II, and Anomalous Cepheids toward the Galactic Center, 2018.

[16] Alexei A. Efros Richard Zhang, Phillip Isola. Colorful Image Colorization., 2016.

[17] Jacob T. VanderPlas. Understanding the Lomb-Scargle Periodogram, 2017.

[18] Zaki Jefferson. Bank Data: SMOTE. https://medium.com/analytics-vidhya/bank-data-smote-b5cb01a5e0a2. Accessed: 2022-22-06.

[19] Pradyumna Yadav. The journey of Gradient Descent — From Local to Global. https://medium.com/analytics-vidhya/journey-of-gradient-descent-from-local-to-global-c851eba3d367. Accessed: 2022-23-06.