UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE GEOFÍSICA

UNA RED NEURONAL PARA DETECTAR TERREMOTOS LENTOS: APLICACIÓN DE UNA U-NET MODIFICADA PARA LA DETECCIÓN DE SEÑALES TRANSIENTES EN BASE A SERIES DE TIEMPO GNSS.

POR JULIO ANDRÉ UTRERAS VARGAS

Tesis presentada a la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Concepción para optar al título profesional de Geofísico

DOCENTE GUÍA: DR. MATTHEW ROBERT MILLER

DOCENTE CO-GUÍA: DR. MARCOS SIMÓN MORENO SWITT

COMISIÓN EVALUADORA:

DR. JULIO BERNARDO ARACENA LUCERO

DR. CARLOS ALBERTO VILLASEÑOR PADILLA

Concepción, Chile 2024

Resumen

El monitoreo continuo de la deformación de la corteza terrestre ha proporcionado evidencia de eventos transientes de deformación intrigante, incluyendo eventos de terremotos lentos (slow earthquake events o SEEs, por sus siglas en inglés), algunos de los cuales se han identificado como posibles precursores de grandes terremotos. Sin embargo, los SEEs son difíciles de detectar y no se tiene una comprensión completa de ellos. Se han propuesto varios métodos nuevos para detectar SEEs, pero la complejidad de las series de tiempo GNSS y las características físicas desconocidas de las señales SEE en términos de magnitud y duración resaltan la necesidad de mejorar los métodos de detección.

Se presenta un método basado en una red neuronal U-Net modificada para detectar señales transientes asociadas con eventos de deslizamiento lento (slow slip events, o SSEs) en series de tiempo GNSS. Esta metodología se desarrolló y se puso a prueba mediante la detección de SSEs en datos sintéticos. Se generaron numerosas series de tiempo que incluyen ruido de fondo (compuesto de ruido coloreado, ruido estacional y anomalías) y sigmoides que representan SSEs de diferentes tamaños y duraciones. La generación de series de tiempo sintéticas incluyó entre 0 y 3 SSEs, aleatorizando parámetros como el tamaño, la duración y el tiempo de inicio para producir transientes que van de -20 a 20 milímetros en las coordenadas este y norte, y duraciones desde 10 a 130 días. Las series de tiempo generadas fueron de 1024 días. Se crearon etiquetas binarias (0 y 1) para indicar la presencia de señales transientes, donde 1 indica la presencia de un SSE, mientras que 0 indica solo ruido de fondo. La red, adaptada a un formato unidimensional para series de tiempo, se entrenó con un paradigma de aprendizaje supervisado para segmentar semánticamente estas series y entregar un vector de etiquetas binarias que indica la probabilidad de una señal transiente en cada punto.

La arquitectura de la red es una versión modificada de la U-Net original, utilizando cinco bloques de codificador y decodificador. Además, se añadieron capas de normalización por lotes para mejorar el rendimiento de la red. El modelo se optimizó usando el algoritmo de Descenso de Gradiente Estocástico con una tasa de aprendizaje de 0.9, y la función de pérdida utilizada fue una aproximación diferenciable del índice de Jaccard (Intersección sobre Unión), logrando valores cercanos a 0.65. Se utilizaron métricas como precisión, sensibilidad, tasa de falsa alarma y valor F1 para medir el éxito de las segmentaciones, basadas en pruebas de detección para una sola señal transiente, variando su tamaño o su duración. Los resultados demostraron una alta precisión, manteniéndose cercana a 1 en la mayoría de las pruebas, mientras que la sensibilidad y el valor F1 generalmente se mantuvieron por encima de 0.5. Las falsas alarmas estuvieron cerca de 0 en todas las pruebas. La efectividad del modelo en la identificación de SSEs ofrece una herramienta potencialmente valiosa para el monitoreo geodésico en zonas de subducción y el estudio de la actividad precursora de terremotos de gran magnitud.

A mis abuelos.

Agradecimientos

Quiero expresar mi agradecimiento a todos y todas quienes han sido parte de mis procesos de educación escolar y universitaria, tanto directa como indirectamente. En mi opinión, el éxito no se debe solamente al esfuerzo personal y a la dedicación, sino también, y quizás en la mayor parte, a la ayuda y apoyo invaluable que nos brindan quienes están presentes con nosotros, inspirándonos para seguir adelante.

En primer lugar, quiero agradecer a mis padres, Julio y Anamaría, mis pilares fundamentales que han guiado mis pasos desde mi infancia. A mis hermanas, Nicole y Francine, por siempre estar presentes y apoyarme en cada etapa de mi vida. Quiero agradecer también a mis abuelos (mis lelos), y a toda mi familia extendida, por su amor y aliento incondicional.

También quiero reconocer especialmente a mis profesores del colegio, Carolina Guajardo y Sergio Neira, quienes despertaron en mí el gusto por el aprendizaje y la pedagogía desde joven. A mis profesores universitarios, Cindy, Matt, Ignacia y Marcos, por ser excelentes guías y mentores en mi formación académica.

A Francisco Tassara, Vicente Yáñez, Joaquín Hormazábal, Sebastián Barra, Denisse Leal, Martín Sepúlveda, Francisco Cerda, y mis profesores Carlos Villaseñor y Julio Aracena por ayudarme a hacer posible mi trabajo de tesis, brindándome su experticia y apoyo en el entrenamiento de mi modelo.

También quiero agradecer profundamente a mis amigos y amigas, quienes son mi familia elegida. A Stephy, mi pareja y mejor amiga, por siempre acompañarme en momentos clave de mi vida. A Héctor, Omar y Juaco, mis mejores amigos y casi hermanos. A Lorena Luo, Sanjay, Pablo Urra, Víctor (Choclo) y Judith, por su amistad sincera. También a Raúl, por recibirme en Guadalajara y ofrecer una ayuda importantísima en mi trabajo.

Finalmente, quiero agradecer al proyecto FONDECYT 1221507 por el financiamiento parcial de este trabajo, que ha sido fundamental en su realización.

Índice

Índice de cuadros

Índice de figuras

1. Introducción

Las placas tectónicas que conforman la litósfera terrestre están en continuo movimiento y pasan por procesos de deformación debido a su generación y reciclaje. Los principales procesos de deformación ocurren en los bordes de las placas tectónicas donde éstas se encuentran e interactúan. Uno de los puntos de mayor interés para el estudio de la deformación en bordes de placas tectónicas es en las zonas de subducción, donde una placa tectónica se hunde por debajo de otra en áreas donde éstas convergen. La subducción de placas es responsable de haber generado algunos de los terremotos más grandes registrados en la historia, tales como el terremoto de Kamchatka de 1952, magnitud 9.0 M_w [1]; el terremoto de Valdivia de 1960, magnitud 9.5 M_w [2], el terremoto de Alaska de 1964, magnitud 9.2 M_w [3] y el terremoto del Maule de 2010, magnitud 8.8 M_w [4], los cuales son resultado de la repentina liberación de energía elástica acumulada a lo largo de decenas o cientos de años, como se describe en la teoría del rebote elástico [5].

Gracias a la implementación de estaciones GPS fijas, que miden con una alta tasa de muestreo la deformación de la placa continental, ha sido posible observar y analizar las componentes tectónicas y no tectónicas que afectan la deformación de la superficie terrestre. Dentro de las componentes no tectónicas se encuentran los procesos estacionales debido a cargas hidrológicas [6], mientras que dentro de las componentes tectónicas se encuentran la compresión intersísmica, los procesos cosísmico y post-sísmico, y varios tipos de señales transientes. Algunas estas señales transientes se asocian a la ocurrencia de terremotos lentos, también llamados eventos de deslizamiento lento o, en inglés, *slow slip events* (de aquí en adelante abreviado como SSEs). Éstos, son un tipo de deslizamiento similar al ocurrido en el período cosísmico, pero con duraciones características mucho más largas a las de un terremoto convencional. Tienen momentos sísmicos comparables a los de un terremoto normal [7] y están asociados a la liberación lenta y de manera asísmica de la energía elástica almacenada por las placas, ya sea dentro de la fase precursora a un terremoto o como parte de la deformación remanente de éste [8]. Los terremotos lentos son de gran interés en la comunidad geocientífica ya que pueden estar asociados a actividad precursora a grandes terremotos de subducción [9][10]. Sin embargo, hoy en día es difícil identificar con exactitud la posición, así como el punto de inicio y término de estos procesos, ya que en los datos de series de tiempo GNSS existe una mezcla de componentes que pueden oscurecer las señales transientes, como el ruido de fondo, las discontinuidades debido a cambios de antena de las estaciones, y los espacios con falta de datos. Esto dificulta la habilidad de las y los geodestas para identificar correctamente la pura señal transiente relacionada a los terremotos lentos.

Recientemente, se han propuesto métodos para la detección automática de este tipo de señales basados en los algoritmos de aprendizaje automático y redes neuronales. Estos algoritmos pueden aprender patrones de los datos que le son entregados y ajustar automáticamente las operaciones que realizan sobre los datos de entrada, para hacer entrega de resultados deseables. Gracias a esto, es que se han utilizado los algoritmos de aprendizaje automático y redes neuronales para detectar señales transientes de origen tectónico, posiblemente relacionadas

a terremotos lentos, mostrando grandes capacidades de detección [11][12][13]. Siguiendo la línea de la aplicación de estas herramientas, en este trabajo se propondrá un método basado en una modificación de la red U-Net [14], la cual fue originalmente propuesta para realizar segmentaciones semánticas sobre imágenes médicas. Es decir, entregar una imagen del mismo formato que la imagen original, pero donde se indique la posición de patrones que la red haya podido identificar. De esta manera, una modificación de esta red se empleará para detectar automáticamente señales transientes asociadas a terremotos lentos en series de tiempo GNSS. La manera en que se llevará a cabo el entrenamiento de esta red, es a través de la generación de series de tiempo sintéticas que simulen la forma del ruido y de los desplazamientos lentos, junto con etiquetas binarias que indiquen la posición de las transientes. De esta manera, a través de un aprendizaje supervisado, la red recibirá las series de tiempo junto con las etiquetas como datos de entrada y su salida será un vector de etiquetas que indique las posiciones donde el modelo haya detectado terremotos lentos. Para poner a prueba la capacidad de este modelo para detectar transientes de diferentes tamaños y duraciones, se realizarán pruebas donde se hagan variar estos parámetros, y se calcularán estadísticos como la precisión, sensibilidad, valor F1 y tasa de falsa alarma. De esta manera, se podrá comparar el resultado de las detecciones entre transientes de diferentes formas para demostrar la capacidad de detección de la red.

1.1. Hipótesis

(a) La detección automática de terremotos lentos en series de tiempo GNSS es posible a través del uso de redes neuronales convolucionales previamente entrenadas con datos sintéticos similares a los reales.

(b) Una red neuronal convolucional puede detectar con alta precisión señales transientes de tamaños y duraciones variables, gracias a su propiedad de invarianza traslacional.

1.2. Objetivos

1.2.1. Objetivo General

Crear una arquitectura de redes neuronales capaz de detectar automáticamente la presencia de señales transientes asociadas a terremotos lentos (SSEs) en series de tiempo GNSS sintéticas creadas a partir de un modelo de ruido que simule su forma observada en datos reales, y sigmoides que representen la forma de un deslizamiento lento. Para tal fin, utilizar una modificación de la red U-Net que lleve a cabo una segmentación semántica de los datos generados, indicando con etiquetas binarias los puntos donde existan señales transientes y donde sólo haya ruido de fondo.

1.2.2. Objetivos Específicos

- Generar una gran cantidad de ejemplos de series de tiempo GNSS con y sin presencia de terremotos lentos, donde se aleatoricen los parámetros de su

generación, con el fin de entrenar la red neuronal con la mayor cantidad de datos posible.

- Adaptar la arquitectura de la red U-Net a datos unidimensionales, como las series de tiempo, y modificarla para optimizar su rendimiento en la detección de transientes.
- Evaluar el rendimiento del modelo entrenado a través de métricas como la precisión, sensibilidad, valor F1 y tasa de falsa alarma para medir su capacidad de detección, llevando a cabo pruebas donde se generen series de tiempo con diferentes tamaños y duraciones de transientes.

2. Teoría del Rebote Elástico y Períodos Sísmicos

La teoría del rebote elástico fue presentada por H.F. Reid en su artículo "Seismological Notes" [5] de 1909, según observaciones de la deformación superficial a raíz del terremoto de California, ocurrido el 18 de abril de 1906. Esta teoría postula que las rocas pueden almacenar energía elástica y, cuando suficiente estrés se acumula, ésta puede liberarse (completa o parcialmente) de manera brusca, produciendo dislocaciones en los lugares más débiles de los bordes convergentes de fallas geológicas. De esta manera, cuando dos placas tectónicas que colisionan se han acoplado y han acumulado el suficiente estrés, pueden liberar esa energía a través de una ruptura que les hace volver a una posición de equilibrio libre de tensiones elásticas, produciendo grandes terremotos.

Este proceso de acumulación de estrés y repentina liberación de energía puede visualizarse fácilmente en las fallas de rumbo (también conocidas como *strike-slip*), donde dos estructuras se deslizan de forma horizontal y paralela, pero en sentido contrario. Al acoplarse este límite entre ambas estructuras que siguen deslizándose, se producen deformaciones en cada una de ellas que les hacen acumular estrés a lo largo de los años. Este ejemplo puede visualizarse en la Figura 1, donde se puede notar la naturaleza cíclica de este proceso.

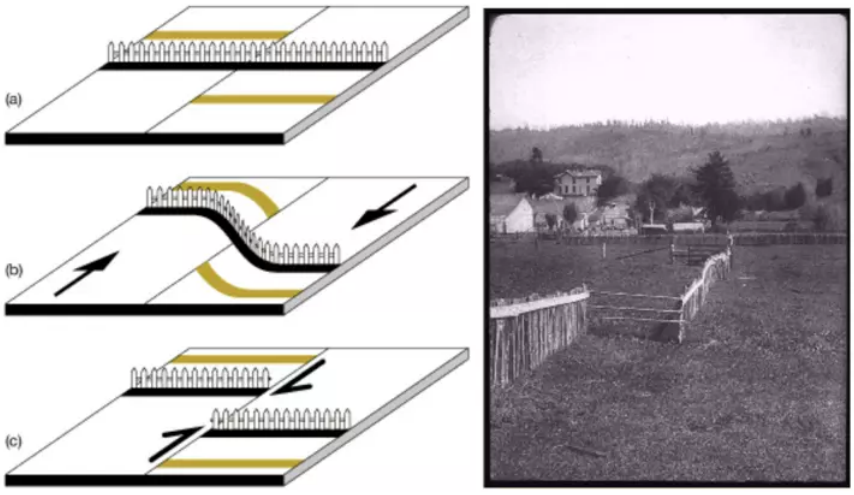


Figura 1: Izquierda: fases de la acumulación elástica en una falla de rumbo. En (a): acoplamiento entre dos placas. En (b): acumulación de energía elástica y deformación. En (c): ruptura y posición de equilibrio. Derecha: fotografía de un cerco ubicado en la falla de San Andrés, desplazado producto del terremoto de California de 1906. Fuente: Glenn Dolphin (2018) [15].

Este proceso se observa también en fallas inversas, especialmente en zonas de subducción (Figura 2). En este caso, dos placas tectónicas chocan de manera frontal, resultando en que una de ellas se hunda por debajo de la otra. Este hundimiento es debido a la diferencia de densidades entre placas, donde la placa más pesada es la que se hunde por debajo de la más liviana.

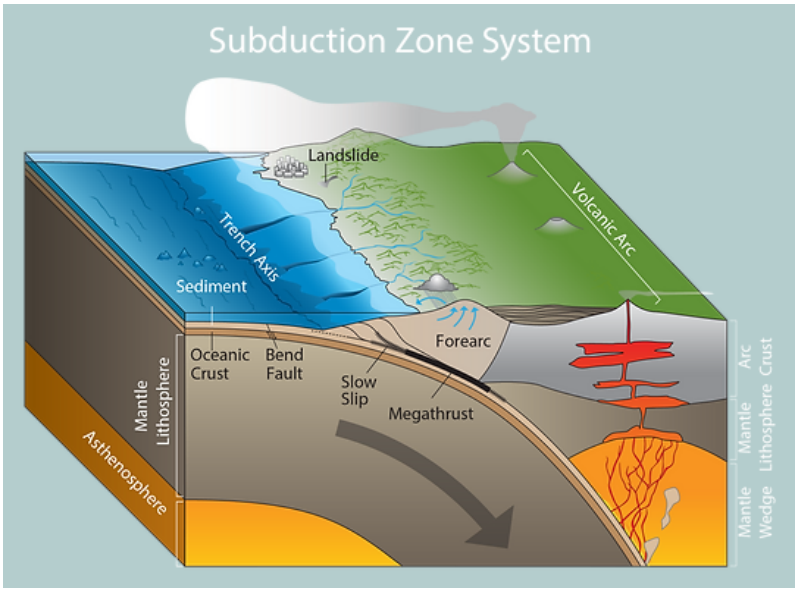


Figura 2: Diagrama en tres dimensiones ejemplificando una zona de subducción. Pueden apreciarse la placa subductante debajo del océano, y la placa continental. Fuente: Katy Cain/SZ4D [16].

La placa más densa, llamada placa subductante, es sujeta a dos fuerzas físicas que le hacen hundirse e ingresar hacia el manto terreste, a lo largo de una fosa oceánica. Estas fuerzas son: el empuje de la placa nueva que se forma en las dorsales oceánicas, llamada **slab push**; y la tensión debido al trozo de placa tectónica que ya está por debajo de la corteza y que está ingresando hacia el manto por efecto de la gravedad, llamada **slab pull**. Debido a la lentitud de la velocidad con la que se mueve la placa subductante y a la fricción que ocurre en el interfaz de las dos placas, éstas no deslizan con facilidad, sino que se acoplan (es decir, se "pegan" una a la otra). Por consiguiente, la placa continental es empujada en la dirección en que se mueve la placa subductante, hacia arriba y contra sí misma, causando que se comprima y acorte. Esto da lugar a deformaciones en la corteza y a acumulación de tensiones en la interfaz de las placas. Similarmente a lo descrito en esta sección, cuando la placa continental llega al límite de la acumulación de estrés y éste supera la resistencia al deslizamiento, la interfaz (también llamada "zona sismogénica") sufre una súbita ruptura que libera una gran cantidad de energía en forma de ondas compresionales y transversales, a la vez que la placa continental se estira hasta llegar a una configuración de equilibrio elástico. Este proceso es cíclico y se puede separar en tres fases, llamadas intersísmica, cosísmica y postsísmica. Una consideración importante respecto a la ciclicidad de este proceso es que si bien se ha observado que es repetitivo, no sigue una periodicidad fija, sino que es un proceso, hasta ahora, impredecible.

2.1. Fase Intersísmica

La fase intersísmica es el período en que se acumula energía elástica en las placas mientras éstas están acopladas. Durante este proceso la placa subductada

se deforma producto del empuje de la placa subductante, y la magnitud de esta deformación es mayor mientras más cerca se esté de la fosa, la cual es el punto más superficial de la interfaz entre ambas placas. La zona de mayor deformación y acumulación de tensión es entre la fosa hasta la línea de costa, y luego disminuye en magnitud con respecto a la distancia de esta zona. Producto de esta deformación, la placa subductada se comprime y trata de compensar la compresión elevándose (Fig. 3) a unos cientos de kilómetros de distancia a la fosa. Sin embargo, entre los 300 a 600 kilómetros en adelante, tanto la acumulación de estrés como la deformación son despreciables [17].

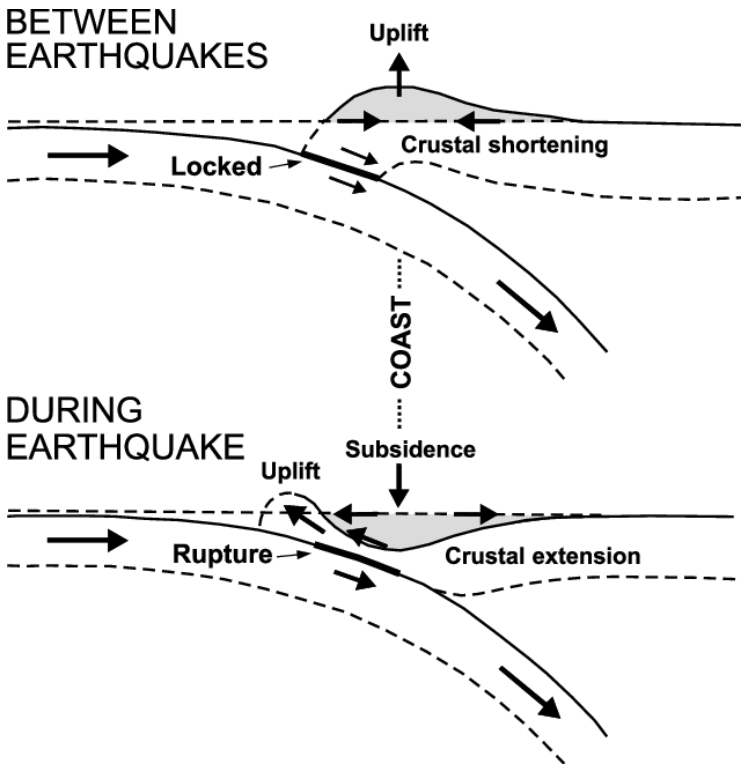


Figura 3: Diagrama que muestra los cambios en la deformación de la placa subductada. Durante la fase intersísmica existe un levantamiento de la placa que luego se hunde durante el período cosísmico. Fuente: Hughes et al. (2002) [18].

2.2. Fase Cosísmica

Esta fase, que puede durar desde segundos a varios minutos, corresponde al período desde que se rompe el parche donde las placas tectónicas se acoplaron, hasta que las placas llegan a una configuración de equilibrio. La ruptura ocurre cuando la tensión supera la resistencia al deslizamiento. Durante este período la placa continental se relaja, deslizándose varios metros en dirección a la fosa, liberando la energía elástica almacenada durante el período intersísmico. Además, se generan ondas compresionales y transversales que componen las ondas sísmicas de un terremoto. Como puede notarse en la Figura 3, la zona que se elevó mientras la placa subductada almacenó energía ahora se hunde, mientras que la parte más cercana a la fosa se extiende y también se eleva. La sección más cercana a la fosa

es la que más sufre deformación durante este período.

2.3. Fase Postsísmica

Después de un terremoto, la corteza terrestre sigue ajustándose y liberando tensión incluso meses o años después del sismo. Esto se conoce como el período intersísmico. Durante esta fase la placa continental se extiende en dirección a la fosa, pero con una velocidad mucho menor a la del período cosísmico que disminuye con el paso del tiempo, hasta que la placa continental deja de extenderse y ambas placas se acoplan de nuevo. Luego de esto, comienza la fase intersísmica otra vez. El período postsísmico suele separarse en dos fases: el postsísmico temprano y el postsísmico tardío. El postsísmico temprano corresponde a un deslizamiento lento de la placa continental, conocido también como *afterslip*. En este período ocurre la mayor cantidad de réplicas del terremoto, localizadas en la zona donde ocurre el afterslip. El postsísmico tardío ocurre debido a la relajación controlada por el comportamiento viscoelástico de la litósfera baja y del manto superior, y es más notorio en los lugares más alejados de la fosa. El postsísmico temprano ocurre dentro de los primeros meses o años después de un terremoto, mientras que el postsísmico tardío puede observarse incluso varios años después.

2.4. Terremotos lentos

Existe un amplio espectro de velocidades para los posibles deslizamientos que ocurren en la interfaz de una zona de subducción. Los terremotos típicos suelen tener velocidades de deslizamiento del orden del 75 %-95 % de la velocidad de la onda S [19], mientras que también se han registrado terremotos con velocidades más lentas, como los *terremoto tsunami* [20], los terremotos de baja frecuencia (LFEs) y los terremotos de muy baja frecuencia (VLFEs). En la parte más lenta del espectro se sitúan los terremotos lentos (también llamados eventos de deslizamiento lento o en inglés: slow slip events, abreviados como SSE). Este tipo de fenómeno tiene más que ver con la geodesia que con la sismología, ya que son estudiados según la deformación que producen en la corteza terrestre y, a diferencia de los terremotos convencionales, no generan ondas sísmicas notorias para los humanos. Los SSEs pueden tener duraciones entre días a meses y alcanzar momentos sísmicos entre 10^{18} y 10^{20} [Nm] [19], liberando una importante cantidad de tensión elástica.

Estos eventos han sido de especial interés en el estudio de los terremotos en zonas de subducción, ya que en varias ocasiones han sido parte de la actividad precursora de terremotos de gran magnitud [8][9][10][21], como también parte de la deformación posterior a ellos [22]. Además pueden entregar información sobre las propiedades friccionales de la interfaz según su profundidad y sobre los cambios del estrés acumulado en la zona de acoplamiento.

Se han detectado deslizamientos lentos en series de tiempo de desplazamiento de estaciones GNSS, el cual es el tipo principal de datos en los que se observan estas señales. Los métodos para detectar estos deslizamientos son variados. Radiguet et al. (2016) [23] utilizó la inversión del análisis de componentes principales

(PCAIM) para analizar la evolución del slip en la interfaz de placas. Donoso et al. (2021) [24] propuso un método llamado PICCA, que es una combinación entre en análisis de componentes principales (PCA) y el análisis de componentes independientes (ICA), para encontrar señales transientes dentro de uno de estos dos componentes. Bedford et al. (2020) [21] utilizó un método basado en la optimización codiciosa, llamado Greedy Automatic Signal Decomposition (GrAtSiD) [25] para descomponer una serie de tiempo GNSS en tres componentes, donde una de ellas contiene movimientos seculares (la tendencia de la serie de tiempo) y transientes. Köhne et al. (2023) [26] desarrolló el paquete DISSTANS de Python para descomponer una serie de tiempo GPS utilizando una combinación lineal de modelos paramétricos. Por otro lado, también se han utilizado arquitecturas de redes neuronales para la detección de estas señales, como se verá en la Sección 3.2.

3. Componentes del ciclo sísmico en series de tiempo GNSS

Las redes GNSS son grupos de estaciones GPS fijas en el suelo, cuya posición relativa es medida diaria o sub-diariamente por una constelación de satélites [27]. Estas redes tienen un uso importante en la geodesia, ya que permiten medir y estudiar los cambios en la corteza terrestre debido a efectos tanto antropogénicos como naturales. Los procesos de origen tectónico son efectos naturales que alteran la forma de la corteza terrestre y pueden ser estudiados a través de series de tiempo del desplazamiento de estas estaciones en tres dimensiones ortogonales: norte-sur, este-oeste y vertical (es decir, en la dirección cenit-nadir). Estas componentes suelen llamarse simplemente "este, norte y vertical" (ENV), ya que se consideran los sentidos positivos de desplazamiento (en el caso de la vertical, el positivo es hacia arriba), mientras que el oeste, sur y vertical hacia abajo son los sentidos negativos de desplazamiento. En las zonas de subducción se pueden diferenciar con facilidad las fases intersísmica, cosísmica y postsísmica asociadas al ciclo sísmico (Fig. 4). Sin embargo, muchas veces el reconocimiento de estas fases puede estar oscurecido por otras componentes no tectónicas captadas por las estaciones GPS, que no ayudan al análisis y estudio de los procesos sísmicos, tales como efectos estacionales, volcánicos, errores en el cálculo de la posición, o cambios de antena en las estaciones.

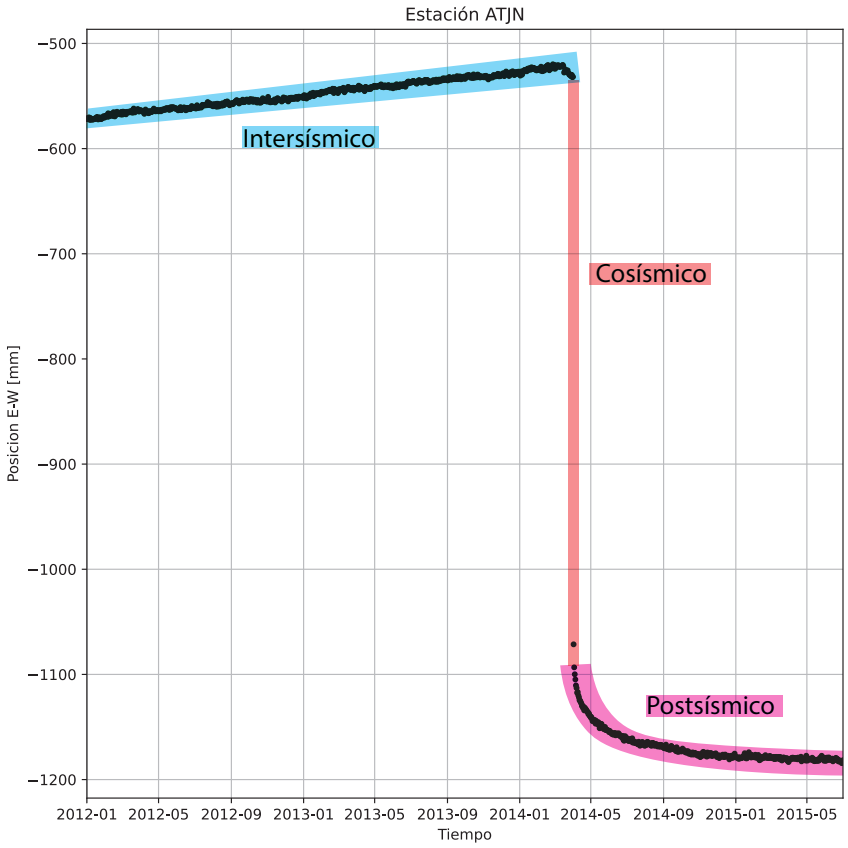


Figura 4: Forma típica de las fases del ciclo sísmico en una serie de tiempo de desplazamiento este-oeste de la estación ATJN en el norte de Chile, entre 2012 y 2015. En celeste se indica el período intersísmico, en naranja la fase cosísmica, y en rosa se indica el período postsísmico. Fuente: Hormazábal (2024) [28].

El ciclo sísmico de una zona de subducción fue explicado en profundidad en las Subsecciones 2.1, 2.2 y 2.3. A continuación se presenta una descripción de cómo se ven cada una de las fases que lo componen en una serie de tiempo de estaciones GPS.

- Fase intersísmica: Se observa como una pendiente que crece en la dirección del movimiento de la placa subductante, ya que en este período ambas placas están acopladas y éste movimiento dominante produce que la placa continental (donde se sitúan las estaciones) se comprima.
- Fase cosísmica: El abrupto deslizamiento de la placa subductada en dirección a la fosa se puede notar como una caída brusca en la serie de tiempo, normalmente asociada a una función escalón de Heaviside cuya magnitud depende de la posición de la estación y su distancia a la fosa. Esta fase es más notoria en las componentes horizontales (norte-sur y este-oeste) y va en dirección a la zona de ruptura de la interfaz de placas. En la componente vertical puede verse como una subsidencia, dependiendo de la posición de la estación GPS (ver Figura 3).
- Fase postsísmica: Esta fase puede verse como un desplazamiento que sigue

la misma dirección de la fase cosísmica, pero cuya velocidad disminuye con el tiempo. Se ve similar a una función logarítmica negativa, y su amplitud es proporcional a la del deslizamiento cosísmico.

En general, estos fenómenos son más notorios en las componentes norte y este, donde el estrés elástico tiene más efecto en la placa continental. En la componente vertical se notan en menor medida ya que se observan efectos no tectónicos como el efecto estacional, debido a la carga de fluidos. Existen varios tipos de señales que son causa de efectos no tectónicos y que son visibles en estas series de tiempo, las cuales serán descritas a continuación.

- Ruido individual: El ruido individual o ruido blanco es el fallo en el cálculo de las coordenadas de cada estación GPS de forma independiente de las demás. Este es un ruido aleatorio de alta frecuencia que no tiene una fuente tectónica.
- Modo común de error: Existe una incertidumbre en el cálculo de la posición que es consistente para toda una red de estaciones GPS, a diferencia del ruido individual que sólo aplica a cada estación por sí sola. Este error se suele abreviar como CME por su nombre en inglés *Common Mode Error* y también se manifiesta como un ruido de alta frecuencia. Sin embargo, ya que está presente en toda la red, se pueden aplicar metodologías para identificarlo y eliminarlo de las series de tiempo.
- Señal estacional: Esta señal tiene orígenes geofísicos que no tienen relación con los procesos tectónicos y tiene una periodicidad anual y semianual. Posibles causas de esta señal pueden ser el aumento en la cantidad de lluvia en ciertas estaciones del año, diferencias en las temperaturas promedio, o variaciones en la cantidad de agua contenida en la superficie [29]. Este tipo de ruido suele simularse como una suma de funciones sinusoidales con un espectro de potencia similar al del ruido rojo.
- Valores anómalos y cambios de antena: Como todo tipo de observación tomada de manera remota, los datos tomados por estaciones GPS están sujetos a posibles errores por errores muy altos en la estimación de la posición, problemas en el receptor del satélite, o fallos con la estación misma. Por otro lado, como parte del mantenimiento de las estaciones, se realizan cambios de antena que producen saltos o discontinuidades artificiales en los datos. Existen varios métodos que proponen un sistema automático de eliminación de anomalías en los datos [30], aunque hasta el día de hoy no existe un método automático universalmente aceptado.

3.1. Modelos de trayectoria

Parte de la correcta estimación de cada componente tectónica y no tectónica de las series de tiempo GNSS, en cada una de sus tres dimensiones, es la de formular modelos que simulen de forma precisa la posición de una estación con respecto al tiempo. Esto permite llevar a cabo una estimación de los parámetros físicos que gobiernan el comportamiento tectónico de un área de estudio, aislando cada señal

que compone una serie de tiempo GNSS. Siguiendo el Principio de Superposición, cada una de estas señales se puede sumar con las demás para volver a generar la serie de tiempo inicial. El modelo de trayectoria lineal estándar (SLTM, por sus siglas en inglés) se compone de tres submodelos:

$$\vec{x}(t) = \vec{x}_{tendencia} + \vec{x}_{saltos} + \vec{x}_{ciclo} \tag{1}$$

Donde $\vec{x}(t)$ es la posición simulada de la estación GPS con respecto al tiempo. $\vec{x}_{tendencia}$ es la componente lineal (también llamada secular) de la serie de tiempo, que representa una velocidad constante siguiendo el movimiento dominante del período intersísmico [31][32]. $\vec{x}_{saltos}$ son las discontinuidades producto de saltos cosísmicos o por efectos antropogénicos. Finalmente, $\vec{x}_{ciclo}$ es la señal estacional de los datos [33]. Se puede notar que el ruido no se modela ya que, por definición, el ruido es aleatorio e impredecible. Según el trabajo de Bevis y Brown (2014) [34], las componentes del SLTM pueden escribirse como se indica a continuación.

Comenzando por la tendencia, ésta puede escribirse de forma vectorial como:

$$\vec{x}_{tendencia} = \vec{x}_R + \vec{v}(t - t_R) \tag{2}$$

Donde $\vec{x}_R$ es un punto de referencia en tres dimensiones, $\vec{v}$ es el vector tridimensional de velocidad de la estación, t representa cada valor del vector de tiempo, y t_R es un tiempo de referencia. Esta ecuación puede verse simplemente como la trayectoria de un objeto que se mueve a velocidad constante, dado un punto y un tiempo de referencia. Sin embargo, esta ecuación no es útil para casos donde existan aceleraciones en la velocidad de las estaciones, por lo tanto, una mejor generalización para la tendencia de las series de tiempo es:

$$\vec{x}_{tendencia} = \sum_{i=1}^{n_P+1} \vec{p}_i (t - t_R)^{i-1} \tag{3}$$

Donde n_P representa el orden de la tendencia polinomial a usarse. Si $n_P = 1$, entonces este modelo se reduce al de la Ecuación (2), con $\vec{p}_1 = \vec{x}_R$ y $\vec{p}_2 = \vec{v}$. Si $n_P = 2$, entonces este modelo pasa a ser un polinomio al cuadrado que representa una aceleración constante, y en ese caso se utiliza el vector de aceleración $\vec{a} = 2\vec{p}_3$.

Los saltos de la serie de tiempo pueden modelarse como:

$$\vec{x}_{saltos} = \sum_{j=1}^{n_J} \vec{b}_j H(t - t_j) \tag{4}$$

Donde n_J indica la cantidad de saltos ocurridos en cada tiempo t_j. $\vec{b}_j$ es el vector que caracteriza cada salto en sus tres dimensiones, y H es la función escalón de Heaviside.

Por otro lado, se puede modelar la señal estacional de la serie de tiempo, de la forma:

$$\vec{x}_{ciclo} = \sum_{k=1}^{n_F} [\vec{s}_k \sin(w_k t) + \vec{c}_k \cos(w_k t)] \tag{5}$$

Donde n_F es el número de frecuencias utilizadas para modelar el ciclo, y los vectores $\vec{s}_k$ y $\vec{c}_k$ son los coeficientes de Fourier para cada armónico con frecuencia angular w_k. Esta frecuencia angular se define como $w_k = \frac{2\pi}{\tau_k}$, donde τ_k es el período correspondiente, tal que $\tau_1 = 1$ año, $\tau_2 = \frac{1}{2}$ año, $\tau_3 = \frac{1}{3}$ año, etc.

Sin embargo, Bevis y Brown (2014) [34] también introdujo una componente extra: los transientes logarítmicos. De esta forma se modela la deformación postsísmica de forma aproximada:

$$\vec{x}_{postsismico} = \sum_{i=1}^{n_T} \vec{a}_i \log(1 + \Delta t_i / T_i) \tag{6}$$

Donde n_T es el número de transientes logarítmicas causadas por un terremoto. $\vec{a}_i$ es el vector de coeficientes de amplitud para cada transiente. Se define $\Delta t = 0$ para los tiempos $t < t_{EQ}$, donde t_{EQ} es el tiempo de ocurrencia de cada terremoto, si no se cumple esa condición, entonces $\Delta t = t - t_{EQ}$. T es la escala temporal característica del desplazamiento de esta transiente logarítmica, y es un valor que se ajusta para conseguir el mejor modelamiento de esta señal.

Se define entonces el *modelo de trayectoria extendido* [34] (ETM, por sus siglas en inglés), como la combinación entre el SLTM y el término que modela la fase postísmica:

$$\begin{aligned} x(t) = &\sum_{i=1}^{n_P+1} \vec{p}_i (t - t_R)^{i-1} + \sum_{j=1}^{n_J} \vec{b}_j H(t - t_j) \\ &+ \sum_{k=1}^{n_F} [\vec{s}_k \sin(w_k t) + \vec{c}_k \cos(w_k t)] \\ &+ \sum_{i=1}^{n_T} \vec{a}_i \log(1 + \Delta t_i / T_i) \end{aligned} \tag{7}$$

El ETM es un modelo que puede simular la trayectoria de las estaciones GPS en la fase postsísmica de manera precisa si se lleva a cabo una buena afinación de los parámetros en la (Ec. 6). A pesar de esto, se han propuesto mejoras al modelo de trayectoria, como los métodos propuestos por Bedford y Bevis (2018) [25] y por Köhne et al. (2023) [26], mencionados en la Subsección 2.4. Otras metodologías utilizadas para analizar series de tiempo de estaciones GPS incluyen la aplicación de herramientas de aprendizaje automático y aprendizaje profundo, como se describirá en la subsección siguiente.

3.2. Usos del aprendizaje automático y aprendizaje profundo en el análisis de series de tiempo GNSS y en la detección de terremotos lentos

Los algoritmos de aprendizaje automático y aprendizaje profundo son capaces de encontrar patrones y realizar descomposiciones en grandes cantidades de datos. Estas tareas normalmente son difíciles y temporalmente costosas para un ser humano. Gracias a que pueden solventar este tipo de problemas con rapidez, es que han tenido una amplia aplicación en el análisis de series de tiempo, tanto para la revisión de sus componentes como para la detección de eventos de deslizamiento lento. Por ejemplo, Simpson et al. (2012) [35] utilizó métodos de clustering para analizar patrones de deformación y velocidad de desplazamiento de estaciones GPS. Hulbert et al. (2019) [36] estudió las señales acústicas emitidas por una cascada de micro-rupturas previas a una ruptura principal en zonas de falla creadas en laboratorio. Estas fallas tienen condiciones similares a las de una falla geológica real. Aplicando métodos de aprendizaje automático basados en árboles de decisión, descubrieron que el tiempo de ocurrencia de la ruptura, junto con la duración y la magnitud de terremotos tanto lentos como "rápidos", puede ser predicha utilizando las ondas elásticas que se generan en la zona de falla. Rouet-Leduc et al. (2020) [37] entrenó una red neuronal convolucional para detectar la ocurrencia de tremor no volcánico y deslizamiento antes y después de terremotos lentos conocidos, utilizando datos sísmicos de una sola estación. Además, utilizó este modelo para reconocer tremor no volcánico en diferentes zonas de falla, tanto de subducción, como en la falla de San Andrés. Dentro de los métodos destinados específicamente a la detección de terremotos lentos en series de tiempo GNSS, están los utilizados en los trabajos de Donoso et al. (2023) [11], que utilizó modelos de Support Vector Machines y redes neuronales artificiales para la detección automática de terremotos lentos en series de tiempo GNSS, validando ambos modelos con datos de la actividad precursora al terremoto de Iquique de 2014, donde fueron capaces de detectar la señal anómala previa al terremoto. Por otro lado, Xue y Freymueller (2023) [12] propuso un modelo de redes neuronales recurrentes para detectar deformaciones transientes en series de tiempo GNSS. Entrenando el modelo con series de tiempo sintéticas, y luego aplicándolo a datos reales de Cascadia entre 2005 y 2016, obteniendo resultados consistentes con otros métodos aplicados en la misma zona de estudio, como los obtenidos por Crowell et al. (2016) [38], que también propuso un método destinado a la detección de señales transientes en series de tiempo GNSS estación por estación, pero utilizando el Índice de Fuerza Relativa (RSI). He et al. (2020) [13] entrenó una red neuronal basada en convoluciones y en capas de LSTM bidireccionales para detectar SSEs en datos de presión del fondo marino. Luego aplicó esta red en datos reales de Nueva Zelanda entre 2014 y 2015, detectando cinco posibles eventos.

Desde la siguiente sección se describen en detalle la historia y el funcionamiento de los algoritmos de aprendizaje automático y aprendizaje profundo.

4. Aspectos fundamentales del aprendizaje automático

El aprendizaje automático (o más comúnmente llamado por su nombre en inglés: *machine learning*) es la ciencia de programar algoritmos capaces de aprender de los datos, que le son entregados durante un entrenamiento, para que pueda realizar una tarea con datos nuevos que el algoritmo nunca haya visto. Una definición más orientada a la ingeniería sobre qué se entiende por *aprendizaje* automático es la que entrega Tom Mitchell en su libro *Machine Learning* [39]:

> Se dice que un programa de computador aprende desde la experiencia E con respecto a una tarea T y una medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E.

Para comprender mejor esta definición, es bueno profundizar en los conceptos de tarea, rendimiento y experiencia. La **tarea** es el problema principal que se busca resolver a través del algoritmo de aprendizaje automático y la manera en que se resuelve el problema depende del tipo de procesamiento que se realiza sobre los datos. El **rendimiento** de un algoritmo es su capacidad para resolver la tarea en cuestión y proveer el mejor resultado posible. La **medida de rendimiento** es la medida diseñada para cuantificar la capacidad del algoritmo para resolver cierta tarea. Esto permite poder comparar dicha capacidad con experiencias pasadas y así evidenciar la mejora o empeoramiento del rendimiento. La manera en que se cuantifica la medida de rendimiento depende de la tarea que se quiera realizar, y por lo general está asociado a una función de costo que mide el error entre la respuesta esperada y la respuesta obtenida del modelo. Finalmente, la **experiencia** puede ser tanto el proceso de entrenamiento del modelo donde se le presentan los datos para que aprenda de ellos, como también los datos por sí solos. Dependiendo si el aprendizaje es supervisado o no supervisado. En el primero, el algoritmo recibe tanto los datos como el resultado esperado para aprender de ellos, mientras que el segundo caso, el algoritmo no recibe el resultado esperado, y sólo busca patrones que pueda encontrar en los datos.

Se pueden analizar algunos ejemplos tomados del libro Machine Learning [39] de la aplicación del aprendizaje automático en varios tipos de problemas para clarificar los conceptos de tarea, medida de rendimiento y experiencia:

- Un algoritmo que sepa jugar ajedrez:
  - Tarea: Jugar ajedrez.
  - Medida de rendimiento: Porcentaje de juegos ganados contra oponentes.
  - Experiencia: Juegos de práctica jugados contra sí mismo.
- Un algoritmo que reconozca texto escrito a mano:
  - Tarea: Reconocer y clasificar palabras escritas a mano en imágenes.
  - Medida de rendimiento: Porcentaje de palabras correctamente clasifi-

cadas.

    - Experiencia: Entrenamiento con base de datos de palabras escritas a mano y sus clasificaciones.

- Un algoritmo que conduzca un auto con inteligencia artificial:
    - Tarea: Conducir de forma apropiada un vehículo en la vía pública usando sensores de visión.
    - Medida de rendimiento: Distancia promedio conducida antes de cometer un error (según supervisión humana).
    - Experiencia: Entrenamiento en base a una secuencia de imágenes y comandos de conducción registrados mientras se observa un conductor humano.

Existen muchas más aplicaciones en casos reales donde las tareas, y por lo tanto las medidas de rendimiento y experiencias, cambian según el problema que se busca resolver. Algunos de estos casos son en el campo de la medicina, donde el aprendizaje automático se ha utilizado para detectar patrones en imágenes biomédicas [14] y para predecir enfermedades a partir de datos clínicos [40]. En la ingeniería se ha usado para optimizar la fabricación de semiconductores [41] y diseños de sistemas de administración de energía renovable [42]. Incluso se ha utilizado en las ciencias sociales para comprender los sentimientos de los mensajes escritos por usuarios en redes sociales y entender la opinión pública sobre temas de contingencia [43] y predecir los comportamientos de los consumidores [44].

4.1. Paradigmas del aprendizaje automático

Hasta ahora se han presentado los conceptos básicos del aprendizaje automático y sus aplicaciones en casos reales. Sin embargo, es bueno categorizar la forma en que el algoritmo aprende tomando como enfoque el nivel de supervisión humana sobre éste. Se han descubierto tres paradigmas en el aprendizaje automático según la forma en que el algoritmo recibe y experimenta los datos. Estos son:

- Aprendizaje supervisado
- Aprendizaje no supervisado
- Aprendizaje por refuerzo

El **aprendizaje supervisado** es el tipo de aprendizaje automático en el cual el algoritmo tiene acceso tanto a los datos de entrada como también a las clasificaciones, etiquetas u objetivos esperados (también llamados objetivos). En estos casos, durante cada experiencia, el algoritmo va a analizar el set de datos de entrenamiento e inferirá una serie de funciones y operaciones que entregue resultados desde el set de datos de entrenamiento. Luego, va a comparar sus propios resultados con los resultados esperados, y buscará minimizar el error que exista entre ellos para mejorar sus predicciones en el siguiente entrenamiento. Un diagrama de flujo de este tipo de aprendizaje puede verse en la Figura 5.

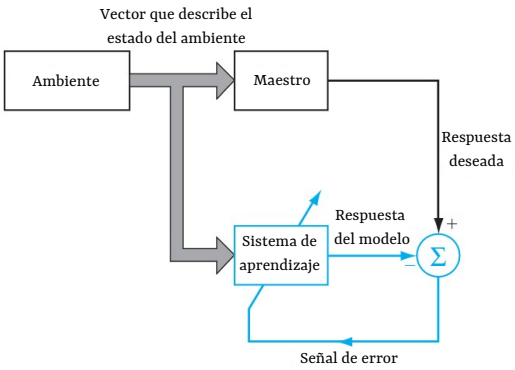


Figura 5: Diagrama del aprendizaje supervisado. La parte en azul representa el proceso de entrenamiento del algoritmo que se repite. Fuente: Modificación de figura extraída de Haykin, 2009, p. 35 [45].

En el **aprendizaje no supervisado** el algoritmo aprende sin ver el resultado esperado, y experimenta con el set de datos de entrenamiento para buscar patrones en los datos. Generalmente se utiliza este tipo de entrenamiento para encontrar patrones ocultos que hayan pasado inadvertidos para los analistas de datos, y para que el algoritmo encuentre formas de organizar o agrupar conjuntos de datos según las similitudes o diferencias que encuentre en ellos. Otros usos del aprendizaje no supervisado incluyen la reducción de la dimensionalidad, donde se busca simplificar los datos sin perder mucha información, y la búsqueda de anomalías. Un diagrama de flujo de este paradigma puede verse en la Figura 6.

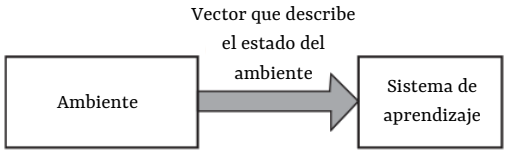


Figura 6: Diagrama del aprendizaje no supervisado. Fuente: Modificación de figura extraída de Haykin, 2009, p. 37 [45].

Además de las diferencias entre el ingreso de los resultados esperados al modelo, existe otra categoría de aprendizaje automático basada principalmente en recompensas o penalizaciones para dirigir el aprendizaje del algoritmo, llamado **aprendizaje por refuerzo**. Aquí, el algoritmo analiza un ambiente (set de datos), selecciona y realiza acciones, buscando maximizar las recompensas que recibe por dichas acciones y minimizar las penalizaciones. De esta forma el modelo debe aprender la mejor *política* para maximizar esas recompensas. Utilizando esta política, el modelo elige la mejor acción a realizar en cada situación que se le presenta. Un diagrama de flujo de este tipo de aprendizaje puede verse en la Figura 7. Comúnmente se ha utilizado para enseñar a inteligencias artificiales

a jugar juegos como ajedrez y go, pero también se ha utilizado para entrenar algoritmos de recomendaciones de productos [46] y control de tráfico de autos [47].

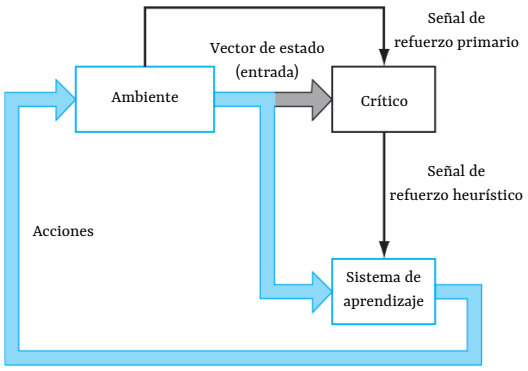


Figura 7: Diagrama del aprendizaje por refuerzo. La parte en azul representa el proceso de entrenamiento del algoritmo que se repite. Fuente: Modificación de figura extraída de Haykin, 2009, p. 36 [45].

5. Redes neuronales y aprendizaje profundo

El concepto de las **redes neuronales artificiales** se presentó por primera vez en el trabajo publicado por el neurofisiólogo Warren McCulloch y el matemático Walter Pitts en 1943 [48], donde se presentó un modelo computacional simplificado sobre cómo las neuronas en los cerebros animales son capaces de realizar procesos complejos, utilizando lógica de enunciados. Esto es, simulando la activación y desactivación de neuronas con operaciones de verdaderos y falsos. Desde entonces se han creado más arquitecturas de redes neuronales según ha aumentado la popularidad de su uso gracias a la mayor cantidad de datos que existen actualmente para entrenar estas redes y también a los avances en la tecnología que han permitido entrenar estos modelos en tiempos más cortos que en décadas anteriores.

Los algoritmos de redes neuronales se basan en capas interconectadas compuestas, generalmente, de neuronas computacionales. Hoy en día no se utilizan tan comúnmente las redes neuronales compuestas de las neuronas propuestas por McCulloch y Pitts, sino que se utiliza un operador más complejo, el cual recibe varios nombres: perceptrón, nodo, unidad, neurona, o incluso neurona artificial. En este documento se le llamará **perceptrón**, para diferenciarlo de la neurona artificial de McCulloch y Pitts. Los perceptrones son el primer algoritmo de entrenamiento de una red neuronal, responsables de realizar operaciones básicas sobre los datos de entrada antes de enviar su resultado a la siguiente capa. En la Sección 5.1 se profundizará en el concepto de perceptrón, que puede considerarse como la célula básica de las redes neuronales. Sin embargo, existe una variedad de funciones que pueden emplearse en cada capa que no dependen del uso de perceptrones, como convoluciones, submuestreos, normalización de datos, entre otras.

Cuando se utilizan arquitecturas conformadas de varias capas de redes neuronales, se debe tener una capa de entrada que reciba los datos y una capa de salida que entregue el resultado final del algoritmo. Las capas que están entremedio de la entrada y la salida se llaman capas ocultas y es donde está el principal mecanismo de la arquitectura. Debido a que: 1. estas capas están ocultas y no se ve a simple vista las operaciones que se realizan ni los resultados que entregan a la capa siguiente y 2. se pueden utilizar varias capas seguidas, con cientos de neuronas en cada una, se hace difícil comprender con facilidad la utilidad de cada operación que se realiza en la red, dándole una naturaleza de caja negra.

El **aprendizaje profundo** no está definido de manera exacta. En general, se habla del aprendizaje profundo como una subárea de la inteligencia artificial donde se utilizan redes neuronales de varias capas para analizar una gran cantidad de datos complejos, pero no existe consenso de cuántas capas se necesitan para que se pueda hablar de aprendizaje profundo, tampoco de qué tan largo debe ser el algoritmo ni de cuántos datos deben analizarse. Sin embargo, según el libro *Deep Learning* [49], el aprendizaje profundo puede ser considerado como el estudio de modelos que implican una mayor cantidad de composición de funciones o conceptos aprendidos que el aprendizaje automático tradicional.

Si bien con el aprendizaje automático es posible crear algoritmos que aprendan de los datos para que puedan reconocer patrones y realizar tareas sin haber sido explícitamente programados para ello, muchas veces se encuentra con limitaciones en casos donde sea difícil extraer las características importantes de los datos (estas características son llamadas *features*). Por ejemplo, cuando los datos no están estructurados, se sufre de la maldición de la dimensionalidad (existen muchas dimensiones para pocos datos, lo que hace que éstos sean dispersos), existen muchos datos irrelevantes para el patrón que se busca, o son de carácter abstracto, suele requerirse una extracción de características (*feature extraction*) hecha a mano antes de entregárselas al modelo de aprendizaje automático. Este trabajo requiere un sólido conocimiento de los datos y de su posible dominio (es decir, su forma, extensión, variabilidad, entre otras cosas) y suele ser un proceso tedioso e inflexible, en especial si se busca entrenar el algoritmo con una gran cantidad de datos. Algunos de los métodos de extracción de características consisten en: analizar los estadísticos de los datos, como su promedio, varianza y simetría para conocer su distribución; llevar a cabo análisis frecuenciales, utilizando la transformada de Fourier; o reducir su dimensionalidad sin perder información importante, utilizando, por ejemplo, un Análisis de Componentes Principales (PCA) [50].

Los algoritmos de aprendizaje profundo pueden superar estos obstáculos y ofrecer ventajas por sobre los algoritmos de aprendizaje automático [51][52]. En general, se busca un solo algoritmo capaz de reconocer las características importantes de los datos sin necesidad de trabajo humano previo, y que entregue los resultados deseados, en lugar de una serie de algoritmos de aprendizaje automático que se encarguen de cada tarea específica, como pre-procesamiento, ingeniería de características, clasificación, entre otras (esto es llamado *end-to-end pipeline*). Esto puede ser posible con algoritmos de aprendizaje profundo, dada una arquitectu-

ra que se ajuste bien al conjunto de datos disponibles y que consiga una buena generalización luego de su entrenamiento [53].

Por otro lado, también se ha observado que los algoritmos de aprendizaje profundo, al tener una gran cantidad de parámetros entrenables debido al tamaño y al número de las capas que incluyen, pueden entrenarse con más datos que los algoritmos de aprendizaje automático tradicionales, antes de alcanzar el punto de saturación. Esto es, el momento en que el desempeño del algoritmo no mejora aunque se entrene con más ejemplos.

5.1. El Perceptrón

El perceptrón (Fig. 8) es la evolución de la neurona artificial presentada en el trabajo de McCulloch y Pitts. Es importante, antes de continuar con la profundización de este concepto, diferenciar a lo que en este trabajo se considerará como perceptrón, el cual es el algoritmo más básico sobre el cual operan las capas de una red neuronal; del concepto de *perceptrón multicapa*, el cual es un tipo de arquitectura de red neuronal.

La diferencia del perceptrón con la de una neurona artificial, es que la segunda recibe una serie de entradas de verdadero y falso, y su vez entrega el mismo tipo de salida binaria según se cumplan las condiciones para que la neurona artificial se active o no. Además, la neurona artificial no permite llevar a cabo el descenso por gradiente (explicado en profundidad en la Sección 5.4) para entrenar la red, la cual es una razón por la que se prefiere utilizar el perceptrón en las redes neuronales modernas.

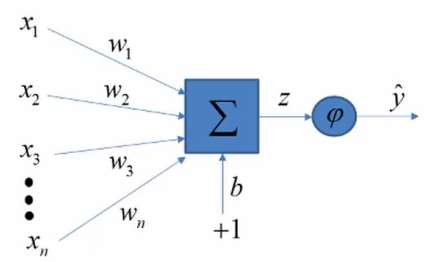


Figura 8: Diagrama de un perceptrón. Los valores x indican los datos de entrada mientras que los valores w indican los pesos. Éstos son multiplicados entre sí y sumados, junto a un sesgo b. Posteriormente, se aplica una función de activación ψ a la salida z, resultando en el valor $\hat{y}$. Diagrama extraído de las clases de Inteligencia Artificial del Dr. Carlos Villaseñor.

En el proceso que lleva a cabo el perceptrón, cada dato de entrada recibido se multiplica por un coeficiente llamado **peso**. Posteriormente, se realiza una suma de la ponderación de los datos de entrada con los pesos para luego ingresarla a una función de activación. La salida que entrega el perceptrón es, en resumen, el resultado de la función de activación sobre la suma ponderada de sus datos de entrada.

Si consideramos el vector de datos de entrada $\vec{x}$, el vector de pesos $\vec{w}$, y la función de activación $f(x)$, entonces podemos escribir la operación del perceptrón como:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \qquad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix}$$

$$S = w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n = \vec{w} \cdot \vec{x} \tag{8}$$

$$h_w(x) = f(S) = f(\vec{w} \cdot \vec{x}) \tag{9}$$

Se puede agregar un dato de entrada $x_0 = 1$, el cual se pondera con el peso w_0. De esta forma se tiene:

$$S = w_0 * 1 + w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n \tag{10}$$

$$S = \theta + w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n \tag{11}$$

Donde el parámetro θ representa al valor de $w_0 * 1$, y es llamado **sesgo** (en inglés: *bias*). Este parámetro también se ingresa en la función de activación. Se obtiene entonces la fórmula final de la operación del perceptrón:

$$h_w(x) = f(S) = f(\vec{w} \cdot \vec{x} + \theta) \tag{12}$$

Tanto los pesos $\vec{w}$ como el sesgo θ son parámetros entrenables. Es decir, después de cada repetición del entrenamiento, sus valores se ajustan para acercar el resultado entregado por el algoritmo al resultado esperado, minimizando el error que existe entre ellos. Esto se lleva a cabo buscando el valor mínimo de una función que calcula el error. Este proceso se explica en la Sección 5.4.

La función de activación que se aplica a la suma ponderada varía según el tipo de resultado que se espera del modelo. Existen varias funciones de activación comúnmente utilizadas, las cuales se explicarán en la siguiente subsección.

5.2. Funciones de activación

La función de activación tiene la tarea de recibir el resultado de la suma ponderada entre los datos y los pesos, junto con el sesgo, como si fuese un valor x en una función cualquiera, que entrega un valor y. Esto ayuda a agregar una componente no lineal a los datos, ayudando a que el modelo aprenda de datos complejos [54]. Dependiendo del tipo de tarea que se busca realizar con un algoritmo de aprendizaje automático o aprendizaje profundo, se puede elegir la función de activación que entregue los mejores resultados. Algunos ejemplos de funciones de activación son los siguientes:

- Unidad Rectificadora Lineal (ReLU): Compara el máximo entre 0 y el valor ingresado, dando como resultado una función igual a 0 para cualquier valor menor o igual a cero, y es igual al dato de entrada si éste es mayor a cero. Su ecuación tiene la forma:

$$f(x) = max(0, x) \tag{13}$$

  Y su forma puede notarse en la Figura (9).

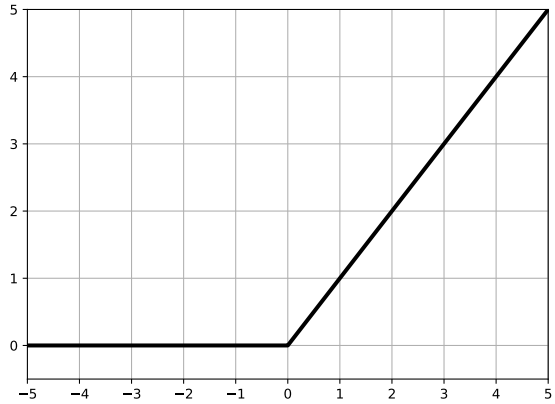


Figura 9: Gráfico de la función de activación ReLU. Fuente: Elaboración propia.

- Sigmoide: Su valor de salida se encuentra entre 0 y 1 y el centro de la sigmoide, donde el valor de entrada es 0, es igual a $\frac{1}{2}$. Su ecuación es:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{14}$$

  Y su forma puede notarse en la Figura (10).

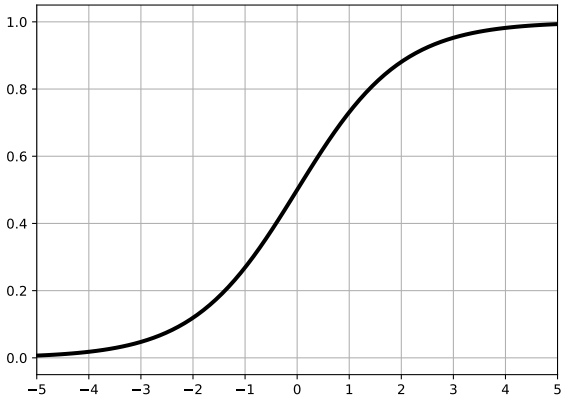


Figura 10: Gráfico de la función de activación sigmoide. Fuente: Elaboración propia.

- Tangente Hiperbólica: Su valor de salida se encuentra entre -1 y 1 y el centro de la tangente hiperbólica, donde el valor de entrada es 0, es igual a 0. Su ecuación tiene la forma:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{15}$$

Y su forma puede notarse en la Figura (11).

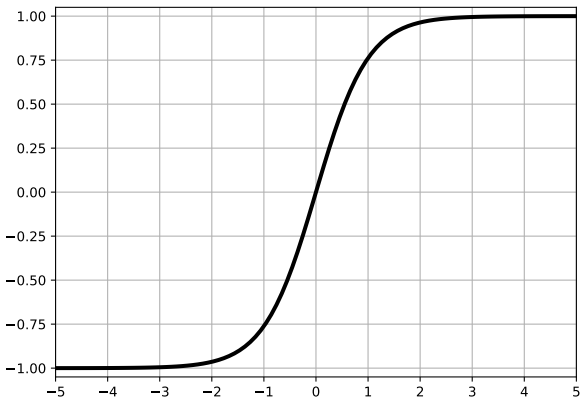


Figura 11: Gráfico de la función de activación tangente hiperbólica. Fuente: Elaboración propia.

- Identidad: Su salida es igual a su entrada. Su ecuación tiene la forma:

$$f(x) = x \tag{16}$$

Y su forma puede notarse en la Figura (12).

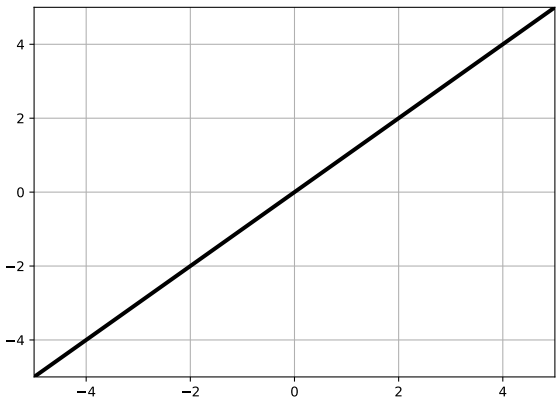


Figura 12: Gráfico de la función de activación identidad. Fuente: Elaboración propia.

- Exponencial Normalizada o SoftMax: Convierte valores en probabilidades para distintas clases, por lo que es una función de activación útil para los problemas de clasificación multi-clase. Su ecuación tiene la forma:

$$f(x)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{17}$$

  Donde K indica las diferentes clases, e^{z_i} es la función exponencial para los datos de entrada y e^{z_j} es la función exponencial para los datos de salida. El gráfico de esta función de activación es difícil de representar en dos dimensiones debido a la cantidad de parámetros que posee, pudiendo ser representada como una generalización de la función sigmoide o una función exponencial que muestra la probabilidad para cada clase.

Es importante que la función de activación tenga una derivada bien definida y que tenga la menor cantidad de puntos donde su pendiente sea cero, para que el modelo sea capaz de encontrar el valor mínimo de la función que calcula el error entre los datos esperados y la predicción del modelo. Se verán más detalles en la Sección 5.4.

5.3. Capas de una Red Neuronal

Ya que se ha definido el funcionamiento de un perceptrón y se han visto ejemplos de diferentes funciones de activación, es pertinente ahondar en cómo estos elementos se combinan para formar un un algoritmo más complejo: una capa de una red neuronal. Estas capas pueden estar compuestas no sólo de perceptrones, y pueden realizar varios tipos de computaciones matemáticas como la multiplicación de matrices, llevar a cabo submuestreos o reducir la cantidad de datos que no contengan información tan relevante. La disposición que tienen las capas de una red neuronal son consideradas la "arquitectura" de la red.

La redes neuronales se componen de tres tipos de capas: las capas de entrada, donde son recibidos los datos; las capas de salida, responsables de entregar los resultados de la red; y las capas ocultas, que están en medio de las capas de entrada y de salida, y es donde ocurre la computación de los datos. Las neuronas que componen una red neuronal pueden estar completamente interconectadas con las neuronas de la capa siguiente (a esto se le llama una red completamente conectada), o pueden estar conectadas sólo a un subconjunto de ellas, como es el caso de las redes neuronales convolucionales o recurrentes.

Una red es considerada "profunda" cuando posee múltiples capas ocultas. Esto suele ayudar en la precisión de los resultados del modelo, a costa de poseer más parámetros entrenables y requerir más tiempo y datos para ser entrenada. La cantidad de capas que posee una red neuronal y la operación que realiza cada una varía según la tarea que se busca realizar con el algoritmo, ya que existen tareas que requieren mayor cantidad de capas para realizar operaciones de mayor abstracción, como el reconocimiento de imágenes, mientras que otras tareas pueden ofrecer grandes resultados con sólo un par de capas que realicen operaciones

específicas para dicha tarea. Es por esto que es conveniente experimentar con varias arquitecturas de redes neuronales hasta encontrar un modelo específico que entregue los mejores resultados.

Algunos ejemplos de capas ocultas y sus funciones son las siguientes:

- Densa:

  También llamada capa totalmente conectada (*fully-connected layer*), ya que está compuesta de perceptrones que están completamente interconectados con los de la capa anterior. Este es el tipo de capa más comúnmente usado en las arquitecturas de redes neuronales. En general, la operación que se realiza en cada perceptón es una multiplicación de matrices entre la matriz de valores de entrada $\vec{x}$ y el vector de pesos $\vec{w}$ que asigna cada unidad, y la suma del vector de sesgos θ. Finalmente, a cada resultado se le aplica una función de activación que todos los perceptrones de la capa comparten. Si se consideran n valores de entrada y m perceptrones, se puede escribir la operación de la capa en notación matricial como:

$$\vec{w} \cdot \vec{x} + \vec{\theta} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & ... & w_{1n} \\ w_{21} & w_{22} & w_{23} & ... & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & w_{m3} & ... & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix} \tag{18}$$

$$= \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + ... + w_{1n}x_n + \theta_1 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + ... + w_{2n}x_n + \theta_2 \\ \vdots \\ w_{m1}x_1 + w_{m2}x_2 + w_{m3}x_3 + ... + w_{mn}x_n + \theta_m \end{bmatrix} \tag{19}$$

  El cual es un tipo de operación lineal, sin embargo, luego se ingresa el resultado de cada fila de esta matriz a la función de activación compartida por todos los perceptrones, de la forma:

$$\begin{bmatrix} f(w_{11}x_1 + w_{12}x_2 + ... + w_{1n}x_n + \theta_1) \\ f(w_{21}x_1 + w_{22}x_2 + ... + w_{2n}x_n + \theta_2) \\ \vdots \\ f(w_{m1}x_1 + w_{m2}x_2 + ... + w_{mn}x_n + \theta_m) \end{bmatrix} = f(\vec{w} \cdot \vec{x} + \theta) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} \tag{20}$$

  Donde el cada valor $\hat{y}_i$ es el resultado de cada perceptrón de la capa densa. El ingreso de los resultados de la multiplicación de matrices a las funciones de activación agrega un componente no lineal a la operación. Las capas densas son útiles para resolver problemas de clasificación donde no se pueden separar linealmente los datos, ya que éstas son aproximadores universales [55].

- Convolucional:

  Las capas convolucionales son el bloque de construcción básico de las redes neuronales convolucionales (CNN, por sus siglas en inglés), las cuales son redes ampliamente usadas para reconocimiento y clasificación de imágenes. Las capas convolucionales no están completamente interconectadas con la capa anterior, como es en el caso de las capas densas, sino que cada capa está conectada a un subconjunto de neuronas de la capa siguiente. Esto permite tres ventajas: la primera es que se pueden resolver problemas de reconocimiento de imágenes utilizando mucha menos memoria computacional que si se usara una capa densa, ya que cada neurona interconectada con las neuronas de una capa adyacente aumenta rápidamente la cantidad de parámetros entrenables de un modelo. La segunda ventaja es que una red compuesta de varias capas convolucionales puede destinar las primeras capas en reconocer patrones simples, y las siguientes en reconocer combinaciones de éstos, dando lugar al reconocimiento de patrones más complejos. Esto imita el funcionamiento del cerebro de los animales y ha entregado buenos resultados en este tipo de problemas [14]. La tercera ventaja de este tipo de capas es que tienen la propiedad de invarianza traslacional, lo cual permite a las redes formadas de capas convolucionales reconocer patrones de manera consistente sin importar su ubicación en los datos. Esta capa aplica una versión discreta del operador convolucional, el cual es un operador lineal. Esta convolución consiste en multiplicar un *filtro* (o *kernel*) elemento por elemento sobre los datos, actuando como una ventana móvil, y luego se suman todos los valores resultantes. Si consideramos una matriz de datos $\vec{d}$ de 4×4 y un kernel $\vec{k}$ de 3×3, la primera convolución puede ejemplificarse de la manera siguiente:

$$\begin{bmatrix} \mathbf{d_1} & \mathbf{d_2} & \mathbf{d_3} & d_4 \\ \mathbf{d_5} & \mathbf{d_6} & \mathbf{d_7} & d_8 \\ \mathbf{d_9} & \mathbf{d_{10}} & \mathbf{d_{11}} & d_{12} \\ d_{13} & d_{14} & d_{15} & d_{16} \end{bmatrix} \Rightarrow \begin{bmatrix} d_1 & d_2 & d_3 \\ d_5 & d_6 & d_7 \\ d_9 & d_{10} & d_{11} \end{bmatrix} \odot \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix} \tag{21}$$

$$\Rightarrow d_1k_1 + d_2k_2 + d_3k_3 + d_5k_4 + d_6k_5 + d_7k_6 + d_9k_7 + d_{10}k_8 + d_{11}k_9 = a_1 \tag{22}$$

Donde el operador $\odot$ simboliza una multiplicación de matrices elemento por elemento (es decir, el producto Hadamard), y el valor a_1 es el resultado de la suma de los valores que resultan de esta multiplicación. Luego, se suma un sesgo θ a este valor, y se le aplica una función de activación:

$$a_1 + \theta_1 \Rightarrow f(a_1 + \theta_1) = \hat{y}_1 \tag{23}$$

Lo cual entrega el resultado de la primera convolución $\hat{y}_1$. En la siguiente convolución, se repite el proceso pero seleccionando la siguiente ventana de datos, moviéndose de izquierda a derecha y de arriba hacia abajo. Cada

vez que se aplica la convolución se aplica el mismo kernel $\vec{k}_1$, se suma el mismo sesgo θ_1 y se aplica la misma función de activación $f(x)$. Una vez realizadas todas las convoluciones, se obtiene el resultado final, llamado *mapa de características*:

$$\begin{bmatrix} d_1 & d_2 & d_3 & d_4 \\ d_5 & d_6 & d_7 & d_8 \\ d_9 & d_{10} & d_{11} & d_{12} \\ d_{13} & d_{14} & d_{15} & d_{16} \end{bmatrix} conv \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix} = \begin{bmatrix} \hat{y}_1 & \hat{y}_2 \\ \hat{y}_3 & \hat{y}_4 \end{bmatrix}_{[1]} \tag{24}$$

El símbolo $_{[1]}$ indica que este es el mapa de características asociado a la primera convolución sobre los datos, con sesgo θ_1. Se puede volver a repetir el proceso descrito anteriormente, pero aplicando un kernel $\vec{k}_2$ y sumando un sesgo θ_2, pero manteniendo la misma función de activación, lo cual entregará un mapa de características diferente. Este proceso es análogo al de una capa densa, donde una cantidad m de neuronas resulta en m resultados, y en este caso, una cantidad m de convoluciones resulta en m mapas de características, pero con la diferencia de que las conexiones son dispersas. Ya que en este caso no hay un vector de pesos $\vec{w}$, los parámetros entrenables de la capa son el kernel $\vec{k}$ y el sesgo θ aplicados a cada mapa de características. Debido a esto, la cantidad de parámetros entrenables de este tipo de capas no depende del tamaño de los datos de entrada, sino que del tamaño del kernel. Las capas convolucionales no necesariamente deben aplicarse a datos en dos dimensiones, como las imágenes, sino que también pueden aplicarse a datos en una sola dimensión utilizando el mismo procedimiento, como se ejemplifica en la Figura (13).

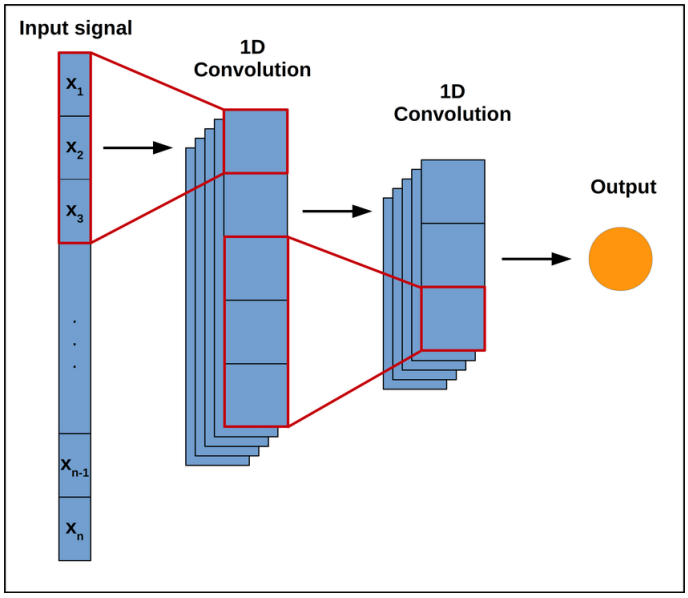


Figura 13: Arquitectura simple de una red neuronal con dos capas convolucionales de una dimensión. Fuente: Shenfield y Howarth, 2020 [56].

Al aplicarse la operación de convolución, las dimensiones de los resultados son de menor tamaño que las dimensiones de la entrada. Para mantener las mismas dimensiones que se tenían originalmente, se puede aplicar un "relleno" a la salida llamado **padding**. Exsiten varias maneras de hacer este relleno, como incluir ceros en los bordes de los datos, o hacer más pequeño el filtro de convolución a medida que se acerca a los extremos. El primer ejemplo es llamado "zero-padding", y tiene la forma:

$$\begin{bmatrix} \hat{y}_1 & \hat{y}_2 \\ \hat{y}_3 & \hat{y}_4 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \hat{y}_1 & \hat{y}_2 & 0 \\ 0 & \hat{y}_3 & \hat{y}_4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{25}$$

Otro parámetro importante en la operación de convolución es el "paso" o **stride**, que indica la cantidad de valores que se salta el filtro antes de aplicar la siguiente convolución. Esto puede hacer que las operaciones se traslapen entre ellas, es decir, que se realicen operaciones tomando valores sobre los que ya se había operado (Fig. 14), o puede hacer que en cada convolución el filtro opere solamente sobre valores no considerados anteriormente. Si se consideran datos en dos dimensiones, como una imagen, el stride horizontal y vertical pueden ser diferentes.

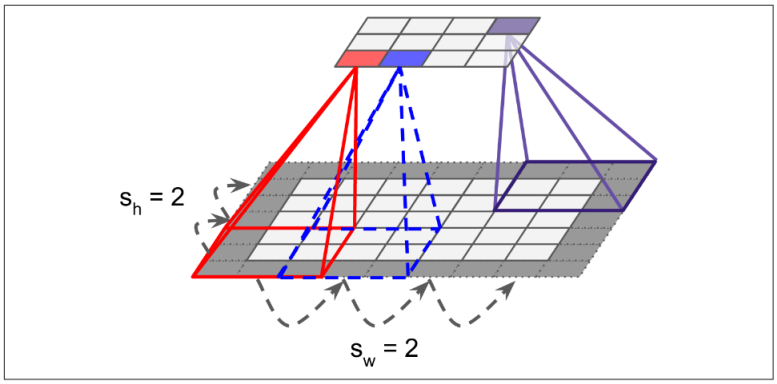


Figura 14: Ejemplo de una convolución utilizando un stride tanto horizontal como vertical igual a 2. Ya que el filtro de convolución opera sobre más datos que el tamaño del stride, entonces hay datos que se vuelven a considerar en la convolución siguiente. Es decir, existe un traslape. Fuente: Géron, 2019, p. 357 [57].

- Pooling:

  Existen varios tipos de capas de pooling, y todas ellas tienen el propósito de realizar un submuestreo sobre los datos que reciben, reduciendo la cantidad de valores que son entregados a la siguiente capa. Esto ayuda a disminuir la carga computacional de la red y limita el riesgo de que la red sobre-entrene sus parámetros (en inglés esto es llamado *overfitting*). Reducir el tamaño de los datos también ayuda a que la red neuronal tolere traslaciones en los

patrones que busca encontrar dentro de los datos. Es decir, le da invarianza espacial a la red.

Al igual que en las capas de convolución, cada neurona de la capa de pooling está conectada a solamente un subconjunto de neuronas de la capa anterior, y la función de cada unidad de pooling es operar sobre una selección de los datos que recibe la capa, ya sea obteniendo el valor máximo de esta ventana de datos (max pooling), o calculando el promedio de éstos (mean pooling). En general, las operaciones de pooling no tienen traslape, aunque esto puede especificarse a la hora de programar una capa de pooling. Tomando de ejemplo una matriz de datos donde se aplica un max pooling de 2×2, se tiene:

$$\begin{bmatrix} 1 & 2 & 0 & 1 \\ 3 & 1 & 2 & 0 \\ 0 & 2 & 1 & 3 \\ 1 & 5 & 2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{1} & \mathbf{2} & 0 & 1 \\ \mathbf{3} & \mathbf{1} & 2 & 0 \\ 0 & 2 & 1 & 3 \\ 1 & 5 & 2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & \mathbf{0} & \mathbf{1} \\ 3 & 1 & \mathbf{2} & \mathbf{0} \\ 0 & 2 & 1 & 3 \\ 1 & 5 & 2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 0 & 1 \\ 3 & 1 & 2 & 0 \\ \mathbf{0} & \mathbf{2} & 1 & 3 \\ \mathbf{1} & \mathbf{5} & 2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 0 & 1 \\ 3 & 1 & 2 & 0 \\ 0 & 2 & \mathbf{1} & \mathbf{3} \\ 1 & 5 & \mathbf{2} & \mathbf{0} \end{bmatrix}$$

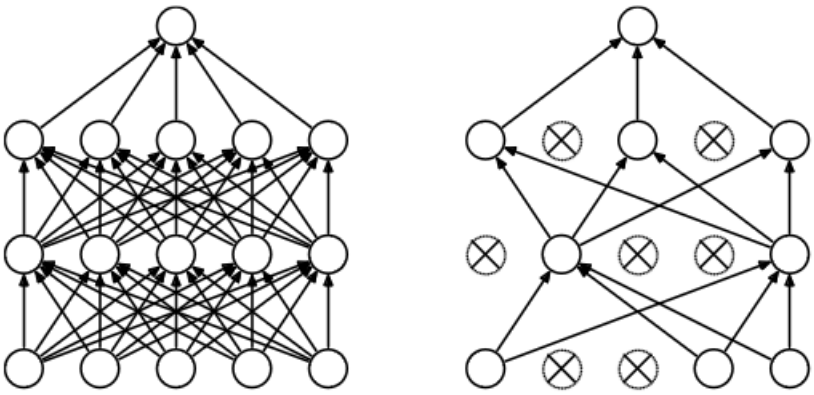
Donde se destacan los valores seleccionados en cada operación de pooling. Si en cada una de estas operaciones se toma el valor máximo de las submatrices seleccionadas, se obtiene el resultado:

$$(2 \times 2)\ \text{MaxPool}\left(\begin{bmatrix} 1 & 2 & 0 & 1 \\ 3 & 1 & 2 & 0 \\ 0 & 2 & 1 & 3 \\ 1 & 5 & 2 & 0 \end{bmatrix}\right) = \begin{bmatrix} 3 & 2 \\ 5 & 3 \end{bmatrix}$$

Los nodos en las capas de pooling no tienen parámetros entrenables: no tienen pesos ni sesgos, y sólo aplican la operación deseada sobre los datos de entrada. En el caso del max pooling, por ejemplo, sólo "sobreviven" los valores más grandes de los datos de entrada, mientras que los demás datos son olvidados por la red. Al igual que en el caso de las capas de convolución, las capas de pooling también pueden tener un padding y un stride.

- Dropout:

  La capa de dropout es una técnica de regularización en donde, al igual que en el caso del maxpooling, algunos valores que la capa recibe van a ser olvidados por la red. La diferencia es que en la capa de dropout no se realiza ninguna operación sobre los datos, sino que sólo se "apagan" algunas neuronas dentro de esta capa (igualando su valor a 0), reduciendo la cantidad de datos de entrada que recibe la capa siguiente, pero sin disminuir la dimensión de los datos. Un ejemplo de este tipo de capa se ve en la Figura 15.



(a) Standard Neural Net (b) After applying dropout.

Figura 15: Ejemplo de una red neuronal compuesta de capas densas donde: en a) se pueden ver las capas interconectadas entre ellas y en b) se aplica dropout a las capas de entrada y a las dos capas ocultas. Fuente: Srivastava et al. (2014) [58]

Esta técnica ayuda a prevenir el overfitting y aumenta la precisión de los resultados entregados por el modelo [59]. La idea detrás de la utilidad de esta capa para mejorar la generalización del modelo es la siguiente: durante el entrenamiento, cada neurona de esta capa tiene una una probabilidad p de apagarse. Puede que una neurona n se apague en la primera iteración de entrenamiento, pero vuelva a funcionar en la siguiente iteración. Esto obliga a cada perceptrón de la capa siguiente a no depender de una unidad o de un conjunto de unidades de esta capa, sino que ahora todos los nodos deben ser capaces de entregar una salida útil para la red. El parámetro p se llama **tasa de dropout**, y típicamente se usan valores del 10 % ó del 50 %. Esta capa no funciona durante la fase de prueba, cuando ya se llevó a cabo el entrenamiento, por lo que, ahora sí, todas las neuronas trabajan juntas.

- Batch Normalization:

  La capa de normalización por lotes (en inglés: *batch normalization*) se encarga de normalizar los datos que recibe antes de enviarlos a la siguiente capa de la red. Esta normalización busca que la media de los datos sea 0 y que su varianza sea 1, llevando los datos a una distribución normal (de ahí el nombre de normalización). Para esto, se debe calcular la media de los datos y su desviación estándar. Posteriormente, la capa de Batch Normalization multiplica los datos por un valor arbitrario α y luego suma otro valor arbitrario β, ambos entrenables, dando una nueva media y desviación estándar a los datos. Esto se hace de la forma:

$$\mu_B = \frac{1}{N_B}\sum_{i=1}^{N_B} x_i \tag{26}$$

$$\sigma_B^2 = \frac{1}{N_B}\sum_{i=1}^{N_B}(x_i - \mu_B)^2 \tag{27}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{28}$$

$$\mathbf{x}_{BN} = \alpha\hat{\mathbf{x}} + \beta \tag{29}$$

Es importante clarificar que estos cálculos no se realizan utilizando todos los datos de entrenamiento, sino que se realizan por secciones de los datos llamados lotes (dando el nombre de normalización por lotes). El tamaño de estos lotes se elige según la capacidad computacional al momento de entrenar la red, y se describe en más detalle en la Sección 5.6. Los valores μ_B y σ_B^2 indican el promedio y desviación estándar del lote, calculados según el tamaño N_B de éste. A cada valor x_i dentro del lote se le resta el promedio calculado, y se divide por la raíz cuadrada de la desviación estándar sumada a un número ϵ. Este término se llama **término de suavizado** y es un valor muy pequeño agregado para evitar las divisiones por cero, normalmente establecido en 10^{-3}. Por último, $\mathbf{x}_{BN}$ es la salida de la capa, donde a todos los valores normalizados $\hat{x}_i$ se les escala por α y se les traslada por β.

Gracias a la normalización de los datos de entrada, los pesos y sesgos que apliquen las capas siguientes del modelo tienen menor riesgo de ser demasiado grandes o demasiado pequeños, y en general la red puede llegar a conseguir buenos resultados más rápido [60], sin necesidad de tener que utilizar técnicas que inicialicen los pesos y sesgos.

- UpSampling:

  La capa de upsampling lleva a cabo un sobremuestreo sobre los datos, repitiéndolos una cierta cantidad de veces y aumentando su resolución espacial. Esta capa no tiene parámetros entrenables. Si se considera una matriz $\vec{x}$ de datos y se realiza un sobremuestreo de tamaño 2, se tiene:

$$(2)\ \text{Upsampling}\left(\begin{bmatrix}0 & 1 & 2\\ 3 & 4 & 5\\ 6 & 7 & 8\\ 9 & 10 & 11\end{bmatrix}\right) = \begin{bmatrix}0 & 1 & 2\\ 0 & 1 & 2\\ 3 & 4 & 5\\ 3 & 4 & 5\\ 6 & 7 & 8\\ 6 & 7 & 8\\ 9 & 10 & 11\\ 9 & 10 & 11\end{bmatrix} \tag{30}$$

- Activación:

Las capas de activación sólo aplican una función de activación a cada uno de los datos que la capa recibe (las funciones de activación se profundizan en la Sección (5.2)). Estas capas ayudan a agregar una componente no lineal a las operaciones que realiza una red neuronal, ayudando a que el modelo se ajuste a datos complejos [54].

Ya que se han abordado los principales tipos de capas de redes neuronales, entre ellas las que fueron utilizadas para esta tesis, se profundizará en cómo un modelo es capaz de ajustar los pesos y sesgos de cada neurona a través del entrenamiento para mejorar el resultado que entrega.

5.4. Descenso por gradiente y Propagación hacia atrás

El descenso por gradiente es un algoritmo de optimización que busca minimizar las diferencias entre los resultados esperados del modelo y los resultados que éste entrega. La manera general en que esto se lleva a cabo es repitiendo el entrenamiento de la red y actualizando los pesos y sesgos de cada capa hasta que se deje de repetir el entrenamiento. Estos pesos y sesgos se actualizan según una **tasa de aprendizaje** η, y la forma en que se calcula la diferencia entre los resultados esperados y predichos por el modelo es a través de una **función de costo** J.

La función de costo, también llamada función de pérdida o función de error, es una función que mide numéricamente la diferencia entre el resultado real (o esperado) y el predicho por el modelo. La función de costo debe ser idealmente diferenciable y no debe contener mínimos locales, o al menos la menor cantidad de ellos. Esto es debido a que, para poder acercar la salida del modelo a los valores esperados, se busca encontrar el valor mínimo de esta función de costo calculando su gradiente en el punto actual, en base a los pesos y sesgos del modelo. Los pesos y sesgos se establecen de forma aleatoria al inicio. Esto se llama **inicialización aleatoria**, y se pueden elegir diferentes distribuciones estadísticas para hacerlo. Luego, durante las siguientes iteraciones del entrenamiento, según la dirección en que estos parámetros hagan disminuir más rápido el gradiente, se actualizan. Esto sigue ocurriendo hasta que el modelo pueda converger en el valor mínimo de esta función, donde los resultados predichos sean iguales a los reales. Un ejemplo de esto se muestra en la Figura 16.

La tasa de aprendizaje es un hiperparámetro que controla cuánto cambian los pesos y sesgos del modelo en cada iteración, siguiento la siguiente ecuación:

$$\theta_f := \theta_i - \eta \nabla J(\theta_i) \tag{31}$$

Donde θ_f son los parámetros actualizados del modelo, calculados según los parámetros anteriores θ_i. η es la tasa de aprendizaje, y ∇J es el gradiente de la función de costo. La comprensión de esta ecuación puede simplificarse en gran parte si se considera que la sección $\eta \nabla J$ es el tamaño del "paso de aprendizaje" que da el descenso por gradiente para acercarse al valor mínimo. Este paso siempre va

en sentido negativo del gradiente, el cual representa la dirección de mayor crecimiento de una función. Luego, los parámetros nuevos θ_f son simplemente la resta entre los parámetros viejos θ_i menos el tamaño del paso. Al momento de elegir η es importante tener presente que un valor muy alto puede causar inestabilidades en la minimización de J, ya que el algoritmo puede hacer cambios demasiado grandes y saltarse el valor mínimo, rebotando entre varios valores de la función de costo y divergiendo, en lugar en converger. Por otro lado, una tasa demasiado pequeña hará que el modelo demore mucho tiempo en finalmente llegar al valor mínimo.

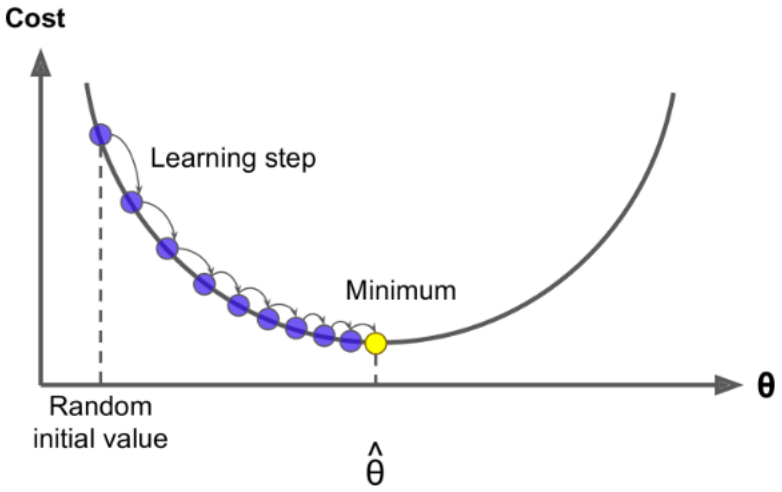


Figura 16: Visualización de la minimización de la función de costo. En este ejemplo, la función de costo es diferenciable en todos sus puntos y sólo tiene un mínimo. El valor de la función de costo durante cada iteración del entrenamiento se representa con un punto azul, y la distancia entre cada punto es el paso de aprendizaje. Fuente: Géron, 2019, p. 111[57].

Para calcular el grandiente de la función de costo ∇J, se necesita de otro algoritmo muy comúnmente utilizado en los modelos de aprendizaje supervisado, llamado **propagación hacia atrás** o, en inglés, *backpropagation*.

El backpropagation se encarga, durante cada iteración del entrenamiento, de utilizar el error de la salida del modelo para calcular la contribución de cada capa a este error. La forma en que esto se lleva a cabo es con los siguientes pasos:

1. **Propagación hacia adelante:** Utilizando los pesos inicializados de manera aleatoria, se calcula una predicción en base a los datos ingresados, computando el resultado de cada neurona desde la capa de entrada hasta la de salida.
2. **Cálculo del error:** Se mide el error de la predicción. Esto es, la diferencia entre el resultado entregado y el resultado deseado. Tomando como ejemplo de función de costo el error cuadrático, tenemos:

$$E_j = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{32}$$

$$E = \sum_{j=1}^{m} E_j \tag{33}$$

Donde es n es la cantidad de datos de entrada del modelo. y_i e $\hat{y}_i$ son las salidas esperadas y predichas, respectivamente. E_j es el error asociado a cada resultado, y E es la suma de todos los m errores calculados.

3. **Propagación del error hacia atrás:** Se calcula cuánto contribuyó la última capa del modelo en el error de la predicción. Una vez listo este cálculo, se vuelve a realizar con la penúltima capa. Esto se repite en cada capa anterior hasta llegar a la primera. De aquí viene el nombre de "propagación hacia atrás". Este paso utiliza la regla de la cadena y encuentra los gradientes del error E con respecto a cada peso w, de la forma:

$$\frac{\partial E_i}{\partial w_i} = \frac{\partial E_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial w_i} \tag{34}$$

4. **Actualización de pesos:** Finalmente, se utiliza el descenso por gradiente (Ec. 31) para actualizar los parámetros del modelo, conociendo el gradiente de la función de costo ∇J para cada capa y la tasa de aprendizaje, para reducir el error de la nueva predicción del modelo. Luego, este paso vuelve a repetirse para reducir más el error y minimizar la función de costo.

Para actualizar los sesgos del modelo, se lleva a cabo el mismo procedimiento pero utilizando las derivadas parciales con respecto a éstos.

El problema de minimización de la función de costo es no lineal, ya que las ecuaciones que ocurren dentro de los perceptrones son no lineales. Por otro lado, aquí se demuestra la importancia de que tanto la función de costo, como las funciones de activación de cada capa, sean diferenciables para asegurar que cada derivada exista y tenga un gradiente. Esto asegura que el modelo pueda encontrar un mínimo de esta función. No se garantiza que se llegue al mínimo global de la función, aunque por lo general esto no es un problema, ya que el mínimo global suele ser un punto de sobre-entrenamiento (esto se explica más a detalle en la Sección 5.6).

También existen algoritmos dedicados a la aceleración del proceso de descenso por gradiente llamados **optimizadores**, los cuales son modificaciones de la Ecuación (31). Por ejemplo, el Descenso por Gradiente Estocástico (SGD, por sus siglas en inglés) introduce un término llamado *momentum* para acelerar la actualización de pesos en la dirección que minimice la función de costo [61]. Otros ejemplos de optimizadores conocidos son RMSprop y ADAM. La selección de optimizador depende de la naturaleza de los datos y puede influir ampliamente en el rendimiento del modelo.

5.5. Funciones de costo

Una función de costo (también llamada función de pérdida o función de error) es una métrica que cuantifica el error que existe entre la salida del algoritmo y la salida esperada. La razón por la que se considera una métrica es porque proporciona una medida cuantitativa del desempeño del modelo. Un modelo con una buena capacidad para realizar la tarea esperada debería tener el menor valor de función de pérdida posible, por lo que es necesario que sea una función diferenciable (es decir, que su derivada exista y esté bien definida en cada punto) para que se pueda minimizar a través de los algoritmos de descenso por gradiente y backpropagation. En muchos casos, la función de costo se abrevia como J, pero esto puede cambiar si se utiliza una abreviación del nombre de la función utilizada. Algunas funciones de costo conocidas, junto con sus abreviaciones en inglés, son:

- Error Cuadrático Medio (MSE): Es la media de la diferencia al cuadrado de cada resultado entregado por el modelo y cada resultado esperado. Este tipo de función de costo comúnmente se utiliza en problemas de regresión. Tiene la forma:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{35}$$

  Donde n es la cantidad de resultados sobre los que se calcula esta función de costo. Se indica con y_i la salida esperada, mientras que $\hat{y}_i$ es cada predicción por el modelo.

- Entropía Cruzada Categórica (CCE): Se define como la sumatoria de cada resultado multiplicado por el logaritmo de la probabilidad de cada categoría. De esta manera, cuando se tiene un problema de clasificación multi-clase, donde cada valor puede pertenecer a una de varias categorías, esta función de costo calcula la probabilidad $s \in [0, 1]$ de que un valor y pertenezca a la categoría C. Considerando su forma normalizada, la CCE tiene la forma:

$$\text{CCE} = -\frac{1}{N}\sum_{i=1}^{C} y_i \log(s_i) \tag{36}$$

  Donde se puede notar el signo negativo que precede al símbolo de sumatoria. Este signo negativo es debido a que, siendo una función de error, se buscan los valores que encuentren su valor mínimo.

- Entropía Cruzada Binaria (BCE): Es un caso particular de la CCE donde sólo existen dos categorías. Considerando las categorías como 0 y 1, la BCE se calcula como se presenta a continuación.

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{2} y_i \log(s_i) = -y_1 \log(s_1) - (1 - y_1)\log(1 - s_1) \tag{37}$$

Donde $y_2 = (1 - y_1)$ y $s_2 = (1 - s_1)$. Cabe destacar que, si bien tanto las categorías como las probabilidades están limitadas entre 0 y 1, sólo las probabilidades se distribuyen de manera continua entre estos dos valores.

- Intersección Sobre Unión como función de pérdida (IoU loss): La intersección sobre la unión, también conocida como índice Jaccard, es una métrica que calcula la similitud entre dos conjuntos A y B a través de la división entre la intersección de ambos conjuntos, sobre la unión de ellos. Esto se puede escribir de la forma:

$$\text{IoU(A, B)} = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|} \tag{38}$$

Ya que esta es una operación sobre conjuntos, no es una función diferenciable. Sin embargo, si A y B son conjuntos de etiquetas binarias compuestas de 0 y 1, se puede utilizar una aproximación numérica para convertirla en una función diferenciable. Por lo tanto, puede utilizarse como una función de pérdida. Según el método propuesto por van Beers en 2018 [62], para dos conjuntos de etiquetas: las etiquetas reales T, que son los datos esperados; y las etiquetas predichas P, que son los datos de salida del modelo, la aproximación IoU' tiene la forma:

$$\text{IoU'(T, P)} = \frac{|T * P|}{|T + P - T * P|} \tag{39}$$

Finalmente, la función de pérdida puede escribirse como:

$$\text{IoU loss} = -\text{IoU'} \tag{40}$$

La cual difiere de la función de pérdida originalmente propuesta por van Beers, la cual tiene la forma $(1-\text{IoU'})$, donde ésta se minimiza a 0 cuando el modelo tiene mejores resultados. Sin embargo, el proceso de minimización sólo busca optimizar la función hasta un valor mínimo, independiente de si dicho valor es 0 o -1. En el caso de esta tesis, donde se utiliza la función de pérdida mostrada en la Ecuación (40), se busca que la función de pérdida llegue a -1.

5.6. Conjuntos de entrenamiento, validación y prueba

Durante el entrenamiento de un modelo de aprendizaje automático o una red neuronal, es común generar tres conjuntos de datos sobre los que trabajará el modelo: el conjunto de entrenamiento, el conjunto de validación y el conjunto de prueba [57].

El **conjunto de entrenamiento** se utiliza, como indica el nombre, para entrenar el modelo según los datos y las salidas esperadas que se le entregan. Durante cada repetición del entrenamiento, llamadas **épocas**, el modelo ajusta sus pesos

y sesgos para acercar sus resultados a los resultados esperados, a través de los métodos de minimización de una función de costo, como se menciona en la Sección 5.4. Sin embargo, entrenar un modelo sólo con un conjunto de entrenamiento ofrece algunas desventajas. La primera, es que al entrenarse con una cantidad limitada de ejemplos, un modelo de aprendizaje automático buscará aprender la mayor cantidad de características posible de estos ejemplos y minimizar lo más posible su función de costo, pero perdiendo capacidad de generalización. Es decir, la capacidad de realizar la tarea aprendida con ejemplos distintos. A este fenómeno se le suele llamar **sobre-entrenamiento**. En el caso de las series de tiempo GNSS, un sobre-entrenamiento puede producir que la red se acostumbre al modelo de ruido de los datos de entrenamiento, y al enfrentarse a datos con un modelo de ruido diferente, no sea capaz de detectar correctamente las señales transientes, o detecte muchos falsos positivos.

Este problema puede mitigarse introduciendo un **conjunto de validación**. El modelo no entrena sus parámetros sobre este conjunto, pero sí se aplica sobre ellos para calcular sus métricas y su función de costo. De esta manera, se tiene un set de datos diferente al de entrenamiento para asegurar su capacidad de generalización. Este conjunto puede utilizarse para aplicar métodos de regularización y de detención temprana, que mantienen una buena generalización del modelo en datos nuevos y se evita el sobre-entrenamiento. En el caso de la detención temprana, se detiene el entrenamiento del modelo cuando la función de costo del conjunto de validación ya no puede minimizarse. Esta detención puede ser instantánea, o después de una cierta cantidad de épocas desde que ya no se observe disminución de la función de costo sobre el conjunto de validación (a esto se le llama "paciencia"), y pueden guardarse los pesos y sesgos de la época con mejores métricas.

En casos donde los conjunto de entrenamiento y validación sean muy grandes y signifiquen un gasto computacional muy costoso, se pueden dividir en **lotes** (en inglés: *batches*), los cuales dividen estos conjuntos en partes más pequeñas sobre las cuales se entrena el modelo. Esto puede significar un tiempo de entrenamiento más largo, pero asegurando que el gasto computacional no sea muy alto y pueda entrenarse el algoritmo sin problemas de limitaciones de recursos.

Una vez finalizado el entrenamiento del modelo se utiliza el **conjunto de prueba** para obtener una evaluación final de la tarea que realiza el algoritmo. Pueden calcularse las métricas y la función de costo sobre este conjunto, pero ya no se utilizarán para la actualización de los pesos y los sesgos del modelo. Este conjunto, a diferencia de los conjuntos de entrenamiento y validación, no se utiliza en cada época de entrenamiento. Sólo se utiliza una vez, luego de darse por terminado el entrenamiento.

5.7. Redes Neuronales Convolucionales

El funcionamiento de las redes neuronales convolucionales (o CNNs) está inspirado en la forma en que trabaja el córtex visual del cerebro de los animales, y, a su vez, su uso está generalmente relacionado a la resolución de problemas como

el reconocimiento y clasificación de imágenes, o de objetos dentro de éstas. La base de la programación de neuronas computacionales inspiradas en el funcionamiento del córtex visual se llevó a cabo en el trabajo de K. Fukushima en 1980 [64], donde se propuso una arquitectura de red neuronal cuyas capas no están completamente interconectadas, como es en el caso de las capas densas, sino que cada capa está conectada a un subconjunto de neuronas de la capa siguiente. Esto imita la manera en que el cerebro reconoce imágenes: un grupo de neuronas iniciales reconocen patrones simples, que luego son enviados a una capa de neuronas encargadas de reconocer patrones más complejos, compuestos de una combinación de patrones anteriores [65][63].

La red neuronal propuesta por Fukushima [64], llamada Neocognitron, dio paso a nuevas propuestas de redes neuronales. Entre ellas, la red *LeNet-5*, propuesta por LeCun et al. en 1998 [66], usada principalmente para detectar caracteres escritos a mano en cheques bancarios. Esta arquitectura utiliza capas densas y de activación, como también capas convolucionales y de pooling (explicadas a detalle en la Sección 5.3). La arquitectura de la red LeNet-5 se puede ver en la Figura 17. Si una red tiene al menos una capa convolucional, generalmente ya es llamada una red neuronal convolucional.

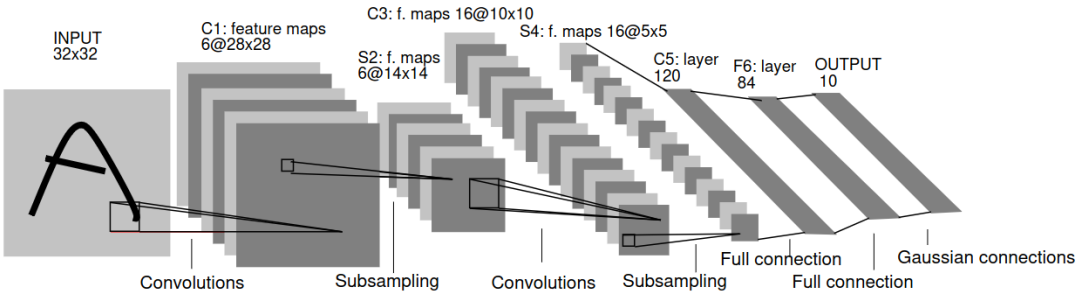


Figura 17: Arquitectura de la red LeNet-5. Luego de cada capa convolucional, que produce varios mapas de características, se aplica un sub-muestreo. Las últimas capas de esta red son capas densas. Fuente: LeCun et al. (1998) [66].

Las redes convolucionales poseen varias ventajas en comparación a las redes compuestas solo de capas densas, como por ejemplo:

- Las neuronas de una capa convolucional no están completamente interconectadas con todas las neuronas de la capa siguiente, lo cual disminuye la cantidad de parámetros entrenables y por lo tanto disminuye tanto la carga computacional como el tiempo de entrenamiento de la red.
- En cada convolución se aplica un filtro (kernel) a un subconjunto de datos de entrada, lo que significa que para este grupo de datos se aplica el mismo peso y sesgo, a diferencia del caso de las redes densas, donde se aplica un peso diferente a cada dato de entrada. Esta propiedad de las redes convolucionales se llama **compartición de pesos** (en inglés: weight sharing), y además de disminuir aún más la cantidad de parámetros, ayuda a prevenir el sobreajuste.

- Las capas de pooling usadas en las redes convolucionales disminuyen la dimensión de los datos sin perder información útil. Esto es gracias a que sacan provecho de los valores máximos de la convolución que representan una correlación mayor con los datos, y quitan los valores menores que pueden ser triviales para la red y que podrían producir sobreajuste [67].

Las capas convolucionales de estas redes generan varios mapas de características luego de cada convolución, por lo que se agrega una nueva dimensión a los datos. Es por esto que la información en estas redes se organizan en arreglos bi o tridimensionales, llamados tensores (Fig. 18). Estos tensores generalizan una matriz (e.g. una imagen), y un vector (e.g. una serie de tiempo).

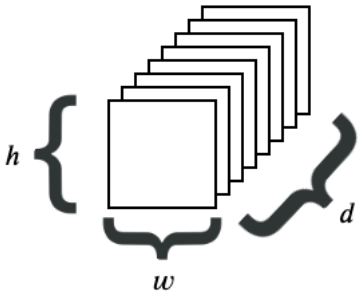


Figura 18: Tensor con dimensiones (w × h × d), para el ancho, altura y profundidad, respectivamente. Fuente: Apunte de Redes Neuronales, Dr. Julio Aracena.

Cada mapa de características del tensor de datos de la red tiene asociado operaciones de convolución, submuestreo, normalización, entre otras, por separado.

Este tipo de redes son capaces de resolver tareas que involucran datos en una variedad de dimensiones. Cuando se tratan datos en una sola dimensión, se utilizan kernels y se llevan a cabo convoluciones unidimensionales. Esto permite que puedan extraerse características en una serie de datos sin importar dónde se encuentre el objetivo y, según el tamaño de los filtros usados, con un tamaño o largo variable. Algunos ejemplos de estas aplicaciones son:

- Predicción de series de tiempo: Las CNNs de una sola dimensión han sido aplicadas a datos de electrocardiogramas para predecir arritmias debido a fibrilación auricular [68], a datos meteorológicos para predecir la velocidad y dirección dominante del viento [69] y datos de tráfico en carreteras para predecir el flujo de vehículos [70].
- Identificación de señales: Otra aplicación en datos de electrocardiogramas involucra la identificación de arritmia y otras enfermedades del corazón [71], y también para la detección de fallas y severidad de daños en rodamientos utilizados para ingeniería mecánica [72].

Mientras que para datos en dos dimensiones, algunas aplicaciones incluyen:

- Clasificación de imágenes: La red LeNet-5 [66] fue aplicada en el reconocimiento y clasificación de caracteres escritos a mano, investigadores de Facebook y Microsoft Research desarrollaron una versión más compleja de

la red ResNet llamada ResNeXt [73], usada para la clasificación de imágenes, y Jiang et al. (2019) [74] propuso una red para clasificar imágenes de tejidos mamarios con signos de cáncer.

- Detección de objetos: Este tipo de tareas también incluye clasificación de imágenes, pero comúnmente se utiliza un marco dentro de la imagen para encerrar los objetos identificados. Algunas redes que se han propuesto para este tipo de problemas son You Only Look Once (YOLO) [75] y Single Shot MultiBox Detector[76].
- Segmentación de imágenes: En el trabajo de Long et al. (2015) [77] se propuso por primera vez una red completamente convolucional aplicada a la segmentación de imágenes. Esto consiste en clasificar cada píxel de una imagen en dos o más categorías, según el objetivo que se desee lograr, dando lugar a una división de la imagen en diferentes áreas donde se identifican patrones. Existen diferentes tipos de segmentación de imágenes, como se muestra en la Figura 19, tales como la segmentación semántica, la segmentación de instancias y la segmentación panóptica. La segmentación semántica clasifica cada píxel de la imagen según lo que se busca encontrar y el "resto" que no pertenece a ninguna categoría, pero sin diferenciar instancias de la misma clase. Por ejemplo, si se busca identificar perros y gatos, todos ellos serán etiquetados según corresponda, sin distinguir entre cada gato y perro individualmente. La segmentación de instancias sí distingue individuos de la misma clase, es decir, distingue cada gato y cada perro como una instancia diferente de dicha categoría. La segmentación panóptica mezcla elementos de las segmentaciones semántica y de instancias: reconoce cada instancia de una misma clase, pero también clasifica el resto de la imagen que no contenga los patrones buscados. En este caso, no solo se reconoce cada perro y gato como individuos diferentes de una misma categoría, sino que también se clasifica el fondo según hayan árboles, personas, casas, etc. La segmentación semántica se ha llevado a cabo con redes tales como la presentada por Long et al. (2015) [77], y la U-Net [14], utilizada para reconocer estructuras en imágenes médicas. He et al. (2017) [78] junto con investigadores de Inteligencia Artificial de Facebook, desarrollaron la red *Mask R-CNN*, capaz de segmentar fotografías de personas clasificando cada individuo (o cada instancia) por separado. Por otro lado, la segmentación panóptica fue propuesta por Kirillov et al. (2019) [79], donde, utilizando una versión modificada de la Mask R-CNN, se podía llevar a cabo un tipo de segmentación que tuviese lo mejor de los dos mundos.

5.8. U-Net

La red U-Net es un modelo de red neuronal convolucional presentado en el trabajo de Ronneberger et al. (2015) [14]. Originalmente esta red se utilizó para segmentar de forma semántica imágenes biomédicas, obteniendo buenos resultados a pesar de entrenarse con pocos datos. Su arquitectura (Fig. 20) se compone de tres secciones principales: una sección de encoder, un puente, y una sección de decoder.

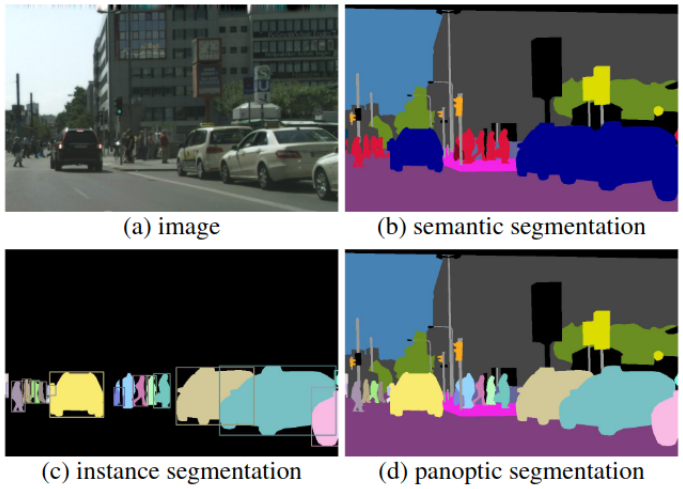


Figura 19: Tipos de segmentación en base a una misma imagen. a) imagen original, b) segmentación semántica, c) segmentación de instancias, y d) segmentación panóptica. Fuente: Kirillov et al. (2019) [79].

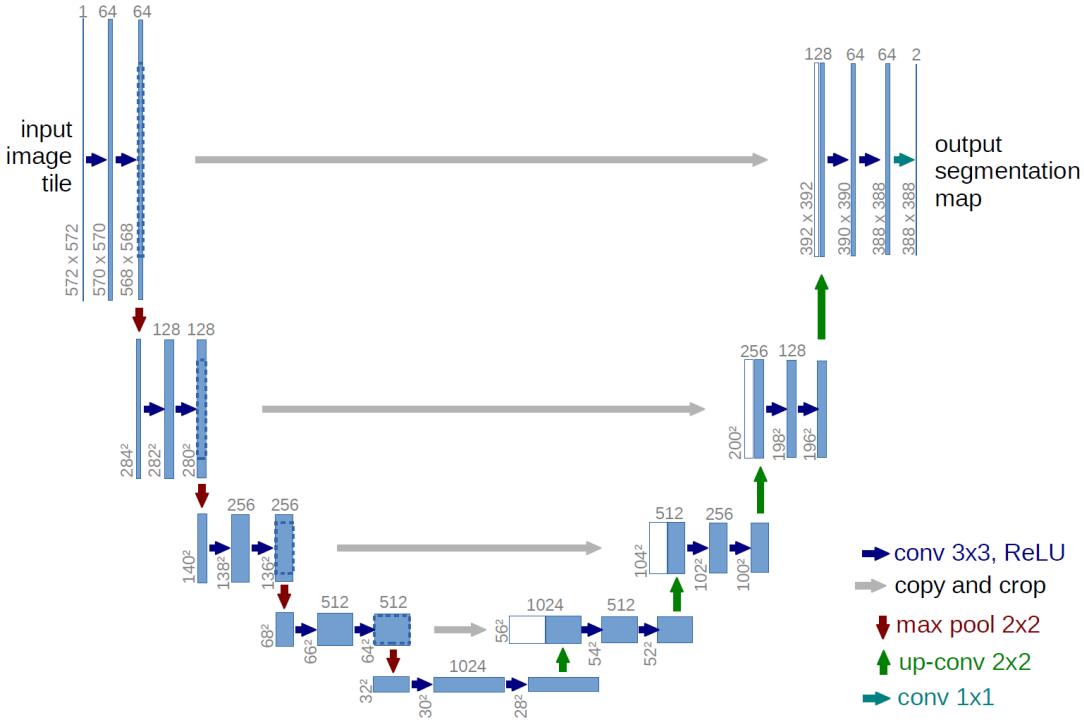


Figura 20: Arquitectura de la U-Net. Fuente: Ronneberger et al. (2015) [14].

La sección de encoder se compone de cuatro bloques de capas que siguen el mismo orden: dos capas convolucionales, seguidas de una capa de maxpooling. Las capas convolucionales usan un kernel de 3×3, mientras que las de maxpooling usan uno de 2×2. La segunda sección de la red, que corresponde al puente, está compuesto de dos capas convolucionales seguidas de una capa de up-sampling y luego otra convolución. Estas dos últimas capas son consideradas como una sola, llamada

"up-convolution". Luego, la sección de decoder contiene cuatro bloques, cada uno compuesto de dos capas convolucionales, y luego una up-convolution. La sección de decoder tiene una característica que hace sobresalir esta arquitectura por sobre otras, y es el hecho de que, aprovechando su simetría, cada bloque es concatenado con la salida de los bloques del encoder que tengan la misma cantidad de mapas de características. Hasta ahora, todas las capas convolucionales de esta red tienen función de activación ReLU. La última capa de esta red es una convolución con un kernel de 1×1 y con función de activación sigmoidal. Esta capa utiliza los últimos mapas de características para obtener la probabilidad entre 0 y 1 de que cada píxel pertenezca a la clase objetivo.

El trabajo realizado por la sección de encoder es encontrar las características más importantes de los datos y eliminar los datos triviales. En cada bloque de encoder, y en el puente, se dobla la cantidad de mapas de características resultantes empezando desde 64 en el primer bloque hasta 1024 en el puente. Cada operación de maxpooling reduce el tamaño de los mapas de características, eliminando valores que no entreguen mucha información, pero manteniendo el mismo número de mapas. Por otro lado, la sección de decoder cumple dos roles importantes: el primero, es que durante cada up-convolution aumenta la resolución de los datos, haciendo más grande cada mapa de características hasta que la salida del modelo tenga aproximadamente las mismas dimensiones que los datos de entrada (la salida es más pequeña, debido a que no se utiliza padding). El segundo rol es utilizar las concatenaciones con las salidas de los bloques de encoder para que el modelo tenga una comparación pixel por pixel de las características más importantes de los datos y del aumento de la resolución. Esto da lugar a una precisión de la segmentación a nivel de cada pixel. Debido a la limitada cantidad de datos de entrenamiento disponibles, se estiraron algunas imágenes, como si los mismos tejidos o células de las figuras lo estuvieran. Este tipo de ténicas, donde se alteran los datos y se aumenta su varianza, pero sin generar datos nuevos, se llama **data augmentation**, y permite mejorar la robusticidad de las redes neuronales sin necesidad de tener una gran cantidad de datos de entrenamiento.

La U-Net no sólo se ha aplicado para el reconocimiento de imágenes médicas, y ha demostrado ser de utilidad con otros tipos de datos. Han et al. (2022) [80] utilizó este modelo para el pronóstico de precipitaciones convectivas utilizando imágenes satelitales. Yin et al. (2023)[81] mezcló esta arquitectura con capas de Long Short-term Memory (LSTM) para predecir los cambios en los bordes de lagos, según las diferencias en la cantidad de agua que contienen a través del tiempo. Perslev et al. (2019) [82] utilizó una versión unidimensional, agregando capas de average-pooling y convolución con función de activación softmax al final de la red para aplicar el modelo a datos de series de tiempo, y luego evaluar la arquitectura para clasificar etapas de sueño en datos electroencefalográficos.

6. Metodología

En este capítulo se detallan los métodos utilizados para generar series de tiempo GNSS sintéticas y el entrenamiento de la red U-Net, a través de un aprendizaje supervisado, para realizar una segmentación sobre estos datos y detectar señales transientes. Además, se presentan las métricas utilizadas para medir el rendimiento del modelo.

6.1. Generación de series de tiempo sintéticas

No existen datos de series de tiempo GPS reales con etiquetas de ocurrencia de señales transientes perfectamente establecidas, ya que es difícil saber con exactitud cuándo comienza un deslizamiento lento y cuándo termina, a pesar de la existencia de estudios que muestran coherencia en la detección de estas señales. Para realizar un entrenamiento basado en el aprendizaje supervisado, es necesario contar con datos de entrada y datos de salida esperados. Para lograr esto, se crearon series de tiempo GPS sintéticas de 1024 días, con etiquetas que indiquen la presencia de una señal transiente. Esto permite generar una gran cantidad de datos para entrenar un modelo, evitando depender del limitado número de casos reales registrados, y solventando el problema del desconocimiento de las fechas exactas de inicio y término de la señal.

El algoritmo de generación de series de tiempo simula el ruido aleatorio de una estación GPS y agrega una componente transiente que imita la forma que tienen los terremotos lentos en este tipo de datos. Para generar los datos de salida esperados, se crearon vectores del mismo largo de las series de tiempo, pero conteniendo sólo etiquetas binarias, que consisten de ceros y unos. Esto indica con 1 los puntos de la serie de tiempo donde existe ocurrencia de una señal transiente, y con 0 los puntos donde no esté presente esta componente. Las etiquetas se generan en base a una relación señal/ruido, donde sólo las transientes suficientemente grandes en comparación al ruido de fondo son etiquetadas. Esta relación se detalla en la Sección (6.1.2).

Para la generación de estos datos y sus etiquetas se utilizó el algoritmo propuesto por Xue y Freymueller (2023) [12], que consiste en la generación de una serie de tiempo GNSS sintética, compuesta de dos tipos de señal que luego se suman (Fig. 21), las cuales son ruido y sigmoides, que representan señales transientes. El ruido se compone de tres tipos: ruido estacional, coloreado, y anomalías. El ruido estacional representa la señal homónima de origen no tectónico (mencionada en la Sección 3) presente en las series de tiempo GNSS. El ruido coloreado es la suma entre ruido blanco, rosado y rojo. El ruido blanco es un tipo de ruido aleatorio que cuenta con la misma potencia a través de todo el espectro de frecuencias. Los ruidos rosado y rojo cuentan con más potencia hacia las frecuencias bajas, pero el ruido rojo aumenta más la potencia mientras menor sea la frecuencia, disminuyendo cerca de 6 decibeles por octava, en comparación a la disminución de 3 decibeles por octava del ruido rosado. Finalmente, las anomalías (o *outliers*, en inglés) son valores anómalos aleatorios que tienen una pequeña probabilidad de sumarse a la serie de tiempo. La generación de la señal transiente se hace

creando una sigmoide donde se aleatorizan parámetros como su tamaño, extensión horizontal y punto de inicio. La amplitud y la extensión horizontal simulan la magnitud y duración de un terremoto lento, respectivamente. Las series de tiempo generadas cuentan con una cantidad de sigmoides que varía aleatoriamente entre cero y tres, y con tamaños entre -20 a 20 milímetros y duraciones entre 10 a 130 días. De aquí en adelante, se referirá al desplazamiento simulado en la generación de sigmoides como el "tamaño" del terremoto lento, y no como "magnitud", para evitar la confusión con el concepto de la magnitud de un sismo.

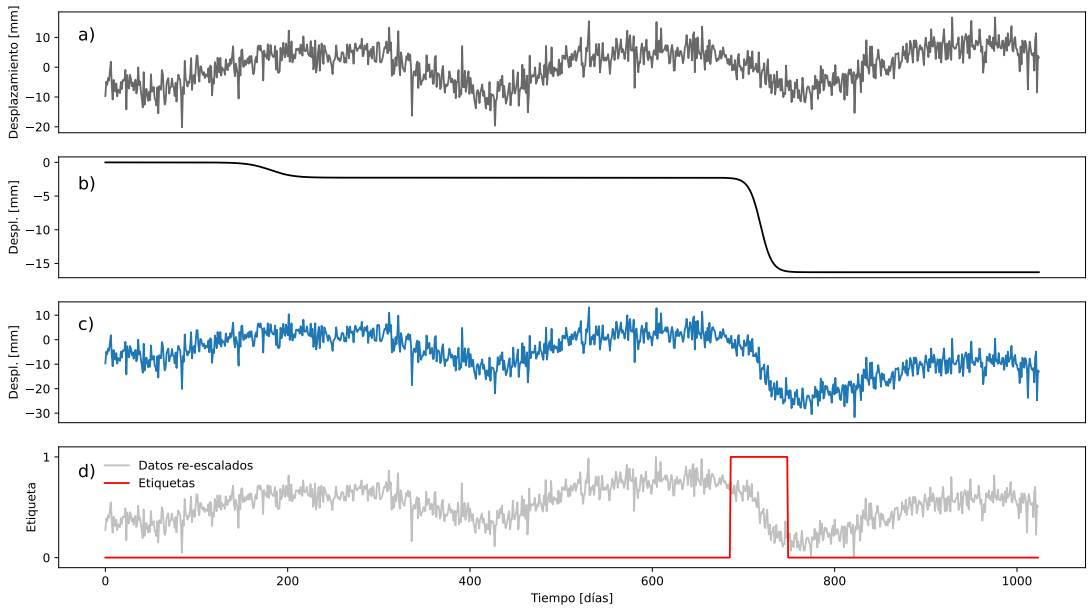


Figura 21: Componentes de los datos sintéticos generados. En (a): Ruido de fondo, compuesto de ruido blanco, ruido rosado, ruido rojo y anomalías. En (b): Señales transientes que simulan la forma de los desplazamientos lentos. En (c): Serie de tiempo resultante de la suma total de las componentes del ruido de fondo y transientes. En (d): En gris claro: serie de tiempo total re-escalada entre 0 y 1, y en rojo: vector de etiquetas binarias que indica la presencia de una transiente. Notar que sólo se generaron etiquetas para la transiente que comienza desde el día 700, ya que tiene suficiente presencia en la serie de tiempo. La transiente que comienza desde el día 180 no tiene suficiente relación señal/ruido, por lo que no se generaron etiquetas que indiquen su presencia.

De esta forma, las series de tiempo generadas pueden escribirse de la forma:

$$\vec{x} = \vec{x}_{ruido} + \vec{x}_{transientes} \tag{41}$$

Donde $\vec{x}$ representa el vector de desplazamientos de una estación GPS, $\vec{x}_{ruido}$ es el vector resultante de la suma de todas las series de tiempo de ruido generadas, y $\vec{x}_{transientes}$ es el vector compuesto de señales transientes con parámetros aleatorizados. El vector $\vec{x}_{ruido}$ se compone a su vez de:

$$\vec{x}_{ruido} = \vec{x}_{estacional} + \vec{x}_{coloreado} + \vec{x}_{outliers} \tag{42}$$

Mientras que $\vec{x}_{coloreado}$ puede descomponerse como:

$$\vec{x}_{coloreado} = \vec{x}_{blanco} + \vec{x}_{rosado} + \vec{x}_{rojo} \tag{43}$$

6.1.1. Parámetros de la generación de cada componente de las series de tiempo sintéticas

El trabajo de Xue y Freymueller utiliza un modelo de ruido basado en el modelo propuesto por Langbein y Svarc (2019) [83], donde se analizaron 740 estaciones GPS a lo largo de la zona oeste de Estados Unidos y se encontraron correlaciones temporales en el ruido de fondo de sus series de tiempo, las cuales representaron como combinaciones de ruido coloreado. En este trabajo se utilizó el mismo modelo, ya que este tipo de ruido de fondo se asimila al de un caso real. Un buen modelo de ruido es importante para generar datos de entrenamiento sintéticos que sean similares a los reales, ya que si el ruido tiene una magnitud demasiado grande con respecto a los deslizamientos lentos simulados, es posible que aquellos de menor tamaño no sean detectados, dando lugar a falsos negativos en las predicciones del modelo. En el caso contrario, donde la magnitud del ruido sea demasiado pequeña, se puede dar lugar a falsos positivos. Las ecuaciones para generar cada tipo de ruido específico se presentan a continuación:

- Ruido estacional: La forma en que se genera esta componente viene del caso particular de la Ecuación (5) donde se consideran los períodos anual y semianual, de la forma:

  $$\vec{x}(t)_{estacional} = C_1 \sin(2\pi(t + R)) + C_2 \sin(4\pi(t + R_u)) \tag{44}$$

  Donde t corresponde a las posiciones de la serie de tiempo que representa los días. C_1 y C_2 son coeficientes de amplitud, y R_u es un número aleatorio uniformemente distribuido entre [0, 1). Los coeficientes se calculan de la forma:

  $$C_1 = 4.5 + 2.4R_u \tag{45}$$
  $$C_2 = 0.7 + 0.8R_u \tag{46}$$

- Ruido blanco: Esta componente está simplemente compuesta de números aleatorios, como se muestra a continuación.

  $$\vec{x}(t)_{blanco} = R_n(t) \tag{47}$$

  Donde, para cada día t, se toma un número aleatorio R_n normalmente distribuido, con media 0 y varianza 1.

- Ruido rosado: Esta componente se genera tomando números aleatorios normalmente distribuidos con media 0 y varianza 1. Luego, se calcula su transformada de Fourier para descartar las frecuencias altas, dejando sólo desde

la mitad hacia abajo. Después, se reconstruye la serie añadiendo sus conjugados para obtener un espectro simétrico pero tomando sólo la parte real. Finalmente, se resta la media y se divide por su desviación estándar para normalizar los datos generados. Cada dato generado se multiplica por el cambio en las unidades de tiempo elevadas a 0.25, de la forma

$$\vec{x}(t)_{rosado} = \Delta t^{0.25} r \tag{48}$$

Donde Δt es el cambio entre cada unidad de tiempo, que en este caso corresponde a 1 día, y r indica los datos generados en el procedimiento descrito.

- Ruido rojo: Este procedimiento es similar al de la generación del ruido blanco, pero con la diferencia de que cada valor de la serie de tiempo se suma con los valores anteriores. Esto puede representarse como:

$$r = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_n \end{bmatrix} \tag{49}$$

Luego, cada valor de la matriz r se multiplica por el cambio en las unidades de tiempo elevadas a 0.5.

$$\vec{x}(t)_{rojo} = \Delta t^{0.5} r \tag{50}$$

Donde, similar al caso anterior, Δt es igual a 1, ya que representa el la distancia en tiempo entre cada día.

- Anomalías: En este caso, se genera una serie de tiempo de valores aleatorios con media 0 y varianza 1, los cuales se comparan con una probabilidad de 0.05. Los valores menores a esta probabilidad se seleccionan para ser multiplicados por un tamaño preseleccionado y por otro número aleatorio con media -0.5 y varianza 2, cuyo resultado se agrega a los datos sintéticos. La operación que se aplica sobre los valores seleccionados c se muestra a continuación.

$$\vec{x}_{outliers} = c * max\_size * (R_n - 0.5) * 2 \tag{51}$$

Donde c corresponde a un valor seleccionado, max_size es el tamaño preseleccionado, y R_n es un valor aleatorio normalizado. Para este trabajo, se consideró un max_size igual a 5.

La creación de las señales que corresponden a terremotos lentos se llevó a cabo generando series de tiempo compuestas de funciones sigmoide, aleatorizando parámetros de esta función para abarcar las diferentes formas que tendría uno de

estos fenómenos en la realidad. Para ello, se consideraron el tamaño, duración y el centro de la sigmode. El centro corresponde al punto donde la pendiente de la función es máxima y, junto con la duración, controla el punto de inicio y de término de la sigmoide. La función utilizada para la creación de estas señales transientes tiene la forma:

$$f(x) = \frac{Amplitud}{1 + e^{-\frac{t-Centro}{Duración/10}}} \tag{52}$$

Donde *Amplitud* es el tamaño de la transiente, generado por un número aleatorio con media 0 y varianza de 20, para asegurar valores entre -20 y 20 milímetros. *Centro* es el punto medio de la sigmoide, generado con un número aleatorio con con media 0 y cuya varianza depende del largo de la serie de tiempo. Esto evita que la transiente se "salga" de los límites de los datos, y tenga su inicio y su fin contenidos dentro de la señal. *Duración* es la extensión temporal de la señal transiente, y también se genera con un número aleatorio con media 10 y varianza 120. Esto asegura que la duración de las transientes sea entre 10 y 130 días. La implementación de estas ecuaciones en Python puede encontrarse en el Anexo, en la Sección (A).

6.1.2. Relación señal/ruido

La relación señal/ruido (de aquí en adelante SNR, por su nombre en inglés: *Signal to Noise Ratio*) es la proporción entre la potencia de una señal que se desea analizar y el ruido de fondo. Esta relación se mide en decibeles y puede utilizarse para establecer un límite de detección. Si se consideran transientes muy pequeñas en comparación al ruido de fondo para entrenar un modelo de detección, éste puede acostumbrarse a detectar falsos positivos en datos que solo sean ruido o anomalías. La forma en que se calcula esta relación es utilizando las desviaciones estándar de la transiente y el ruido, solamente durante los días en que la primera esté presente, de lo contrario, el ruido tendría siempre mayor importancia que la transiente en este cálculo. La operación completa de este cálculo se presenta a continuación.

$$SNR = 10 \log_{10}((\frac{\sigma_{transiente}}{\sigma_{ruido}})^2) \tag{53}$$

Donde σ indica la desviación estándar de las señales indicadas. Para esta tesis se estableció un límite de 0 decibeles (los decibeles pueden ser negativos) como límite de detección. Esto asegura que no se generen etiquetas cuando las sigmoides sean muy pequeñas en comparación al ruido, disminuyendo los falsos positivos que el modelo pueda encontrar.

6.1.3. Escalamiento de los datos

El escalamiento de los datos (también llamado *feature scaling*, en inglés) es un método utilizado para que los valores de entrada estén contenidos en un rango

entre 0 y 1. Esto asegura que no hayan valores en los datos que sean demasiado grandes con respecto a otros (por ejemplo, si hay diferencias en órdenes de magnitud) y que puedan tener dominancia en el modelo, si es que afectan con más fuerza el ajuste de ciertos pesos dentro de las capas de la red. Por lo tanto, escalar los datos ayuda al rendimiento del modelo y a que llegue más rápido a la convergencia [57].

Antes de aplicar este método, se le resta la tendencia a los datos. Luego, se lleva a cabo el escalamiento restando cada valor de la serie de tiempo por su valor mínimo, y dividiendo esa diferencia por la resta entre el valor máximo y el valor mínimo de los datos, de la forma:

$$x_{escalado} = \frac{x_i - min(x)}{max(x) - min(x)} \tag{54}$$

Donde $x_{escalado}$ representa cada valor escalado de los datos y x_i es cada valor original. $min(x)$ y $max(x)$ son el mínimo y el máximo de todos los datos, respectivamente.

6.2. Entrenamiento de la red U-Net

6.2.1. Arquitectura utilizada

El modelo de red neuronal utilizado en este trabajo es una modificación de la red U-Net (descrito en la Sección 5.8). Esta red se adaptó para trabajar con datos en una sola dimensión (como son las series de tiempo) y se agregaron dos bloques más a la red: un bloque extra de encoder, y un bloque extra de decoder, para mantener la simetría de la red. De esta forma, el modelo cuenta con cinco bloques de encoder, un puente, y cinco bloques de decoder. La cantidad de filtros utilizados en los bloques de encoder siguen el orden de 32, 64, 128, 256 y 512. El puente tiene 1024 filtros, y los filtros de los bloques de decoder tiene el mismo tamaño que el de los bloques de encoder, pero en orden descendente. Por otro lado, se agregó una capa de Batch Normalization después de cada capa convolucional (excepto por la capa de salida, que se mantiene igual a la de la red original). Cada bloque de encoder consta de un bloque de convolución y una capa de MaxPooling con una ventana de tamaño 2. El resultado del bloque de convolución se guarda para la concatenación, mientras que el resultado de la capa de MaxPooling se pasa al siguiente bloque de encoder. Cada bloque de convolución se compone de una capa de convolución con un kernel de tamaño 50, una capa de Batch Normalization, y una capa de activación ReLU. Luego, se repiten las capas de convolución, Batch Normalization, y ReLU, en ese orden. El puente sólo se compone de un bloque de convolución. Luego, cada bloque de decoder se compone de una capa de UpSampling con ventana de tamaño 2, seguido de una capa de convolución. Se lleva a cabo la concatenación entre la salida del bloque de convolución dentro del bloque de encoder con el resultado del proceso hasta ahora descrito en el bloque de decoder, para posteriormente aplicar una capa de convolución a los datos concatenados.

Hasta ahora, todas las capas de convolución tienen un kernel de tamaño 50. La última capa, que corresponde a la capa de salida, se compone de una convolución con un kernel de tamaño 1 (es decir, que opera punto por punto dentro de la serie de tiempo), y con una función de activación sigmoidal. Todas las capas de este modelo son unidimensionales.

6.2.2. Entradas y salidas del modelo

Para realizar el entrenamiento de este modelo se generaron 100000 series de tiempo GNSS sintéticas de 1024 días, con una cantidad de señales transientes seleccionadas aleatoriamente entre 0 a 3. Junto con esto, se generaron etiquetas para cada serie de tiempo indicando la presencia de estas señales con 0 (ruido) y 1 (terremoto lento), manteniendo el mismo largo de los datos. Tanto las series de tiempo como las etiquetas se juntaron en matrices, manteniendo el orden en que fueron generadas, donde cada fila contiene una serie de tiempo distinta y las columnas corresponden a los días. Así, se tienen los datos de entrada (series de tiempo sintéticas) y los datos de salida esperados (las etiquetas generadas). Este set de datos fue dividido en un 90 % para entrenamiento, y un 10 % para validación. Para probar las predicciones del modelo, se generaron 1.000 series de tiempo sintéticas. Los resultados deberían replicar la forma de las etiquetas, llevando 0 en los puntos donde no haya presencia de una sigmoide, y 1 en los puntos donde sí haya presencia. Sin embargo, la diferencia entre las etiquetas generadas para entrenar el modelo y las predicciones de éste, es que las predicciones no deben interpretarse tajantemente como la presencia incuestionable de un terremoto lento, sino como la probabilidad, entre 0 y 1 (es decir, entre 0 a 100 %), de que esta señal esté presente.

6.2.3. Optimizador, función de pérdida y parámetros de entrenamiento

El modelo fue compilado utilizando el algoritmo de Descenso por Gradiente Estocástico (de aquí en adelante *SGD*, por sus siglas en inglés) como método de optimización. Este optimizador ha entregado buenos resultados en problemas de clasificación de imágenes, superando en algunos casos al optimizador ADAM [61]. Se utilizó una tasa de aprendizaje de 0.9 y un momentum de 0.1.

Como función de pérdida se utilizó una aproximación del índice Jaccard, también llamado Intersección sobre Unión (o *IoU*, por sus siglas en inglés), la cual se utiliza como métrica para el éxito de un modelo predictivo. Al utilizar una aproximación de esta métrica, el IoU se convierte en una función diferenciable y, por lo tanto, se puede usar como función de pérdida [62]. Por otro lado, también se utilizó la métrica Intersección sobre Unión Binaria de Keras (originalmente llamada en inglés, *BinaryIoU*, forma en la cual se le llamará en este documento), para comparar su similitud con la función de pérdida, pero calculado con un algoritmo distinto. Una diferencia clave entre ambos tipos de Intersección sobre Unión (la función de pérdida y la métrica de Keras), es que nos interesa que una función de pérdida tenga el valor más pequeño posible (en este caso, más cercano a -1), mientras que para la métrica, nos interesa que su valor sea el más alto (más

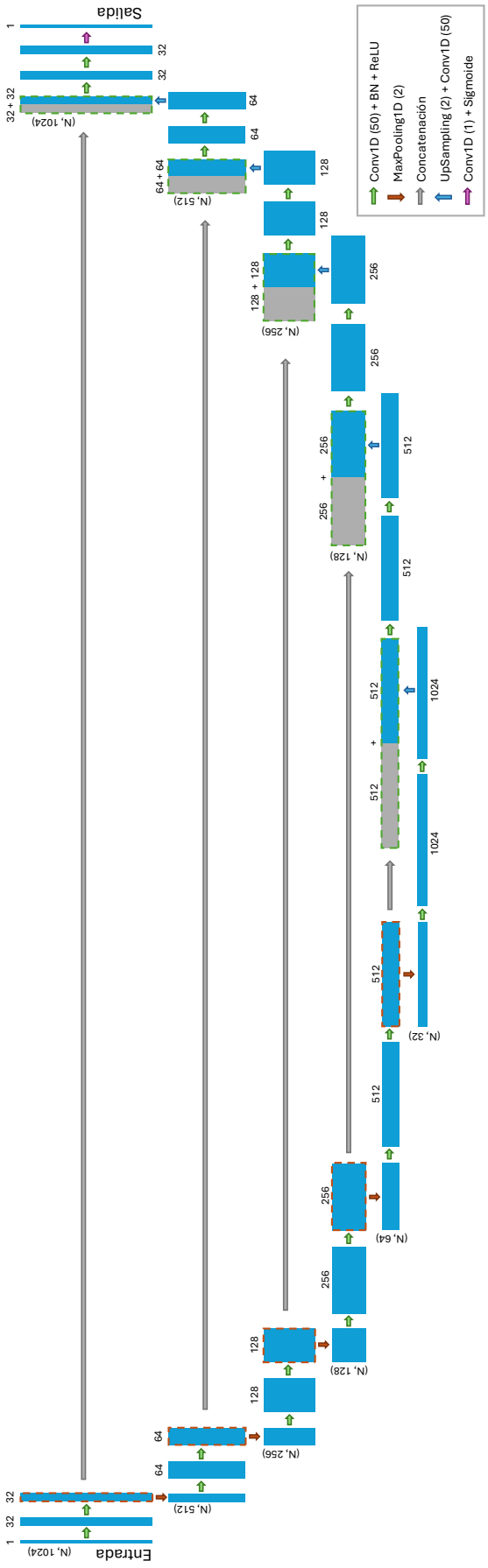


Figura 22: Versión modificada de la U-Net utilizada en la segmentación de las series de tiempo GNSS sintéticas. Los bloques azules indican los datos que se trabajan en cada capa, donde N representa la cantidad de muestras (en este caso, terremotos generados) trabajadas en cada época y 1024 es la cantidad de días de cada serie de tiempo. Los números escritos sobre o bajo los bloques indican la cantidad de filtros (resultantes de las capas de convolución) correspondiente a cada parte de la arquitectura. Los bloques grises representan los datos que son copiados y concatenados en la red.

cercano a 1). Por lo tanto, es de esperarse que uno de estos valores sea igual al negativo del otro.

El entrenamiento se llevó a cabo con un tamaño de batch de 32. Para evitar el sobreajuste, se utilizó la opción de Detención temprana (en inglés: *Early stopping*) de Keras, la cual detuvo el entrenamiento cuando la función de pérdida de la validación no mejoró durante diez épocas. Por otro lado, se utilizó el algoritmo de Retrollamadas (o *Callbacks*, en inglés) para guardar las versiones del modelo (es decir, se guardaron los pesos y los sesgos) en cada época del entrenamiento, pero sólo se mantuvo la versión con las mejores métricas una vez finalizado.

6.3. Evaluación del modelo

Se llevaron a cabo dos pruebas de detección donde, en cada una, se generaron series de tiempo con una sola señal transiente. En la primera prueba, se hizo variar el tamaño del terremoto lento generado entre 1 a 30 milímetros, aumentando en 1 milímetro, pero manteniendo una duración fija de 60 días. En la segunda prueba, se hizo variar la duración entre 5 a 150 días, aumentando en 5 días, pero manteniendo un tamaño fijo de 15 milímetros. Durante cada iteración de las pruebas se generaron 1000 series de tiempo y se calcularon estadísticos para medir el éxito de las segmentaciones.

Se seleccionaron cuatro estadísticos que entregan información útil sobre los resultados de una clasificación binaria: la precisión, la sensibilidad (también conocida como True Positive Rate, o simplemente TPR), la tasa de falsa alarma (también conocido como False Positive Rate, o FPR), y el Valor F1.

Estos cuatro estadísticos se basan en la cantidad de verdaderos positivos (VP), falsos positivos (FP), verdaderos negativos (VN) y falsos negativos (FN) que detecta un modelo, según se tenga certeza de los resultados correctos de las pruebas que se apliquen. Este es justamente el caso en este estudio, ya que en las etiquetas binarias de cada serie de tiempo, puede considerarse un 1 (presencia de señal transiente) como un "positivo", y un 0 (ausencia de señal transiente) como un "negativo". En este caso, los puntos donde el modelo detecte un terremoto lento que hayan coincidido correctamente con los datos de prueba serán contados como verdaderos positivos. Mientras que, tomando otro ejemplo, cuando el modelo no detecte un terremoto lento en puntos donde en realidad sí haya, éstos serán contados como falsos negativos.

La precisión indica el éxito del modelo en acertar en las instancias positivas que clasificó. Es decir, cuántos de los positivos encontrados por el modelo fueron realmente señales transientes, y se calcula de la forma:

$$Precisión = \frac{VP}{VP + FP} \tag{55}$$

La sensibilidad es la proporción de instancias verdaderamente positivas que el modelo fue capaz de clasificar correctamente. Es decir, la capacidad del modelo de encontrar todos los puntos donde haya ocurrencia de terremotos lentos, y su

fórmula es la siguiente:

$$Sensibilidad = \frac{VP}{VP + FN} \tag{56}$$

Es importante hacer énfasis en la diferencia entre la precisión y la sensibilidad, ya que pueden confundirse fácilmente debido a que ambos están relacionados con la detección de casos positivos. En primer lugar, la diferencia más obvia se encuentra en el denominador de estos estadísticos: la precisión disminuye con la cantidad de falsos positivos, mientras que la sensibilidad disminuye con los falsos negativos. En un sentido más teórico, la diferencia radica en que para una alta precisión, importa que los positivos clasificados hayan sido realmente positivos, independiente de si hubieron señales transientes que el modelo no encontró. Mientras que para una alta sensibilidad, importa que el modelo no haya clasificado como negativo (es decir, ruido) los puntos donde hayan positivos (transientes), sin importar que haya levantado una falsa alarma.

El Valor F1 es la media armónica entre la precisión y la sensibilidad. La motivación de este cálculo es tener una calificación de la capacidad general del modelo para clasificar instancias correctamente, tanto en su relevancia como en su exhausitividad. La razón por la que se utiliza una media armónica, en oposición a una media aritmética, es que la media armónica combina dos factores importantes: el promedio entre ambos estadísticos (es decir, su media aritmética) y la tasa entre ambos, dada por su división. El Valor F1 da un puntaje de 0 en caso de que cualquiera de los estadísticos lo sea, y da más importancia a las mejoras en valores pequeños (por ejemplo, de 0.0 a 0.1), y menos importancia a las mejoras en valores grandes (por ejemplo, de 0.9 a 1.0). Este valor se calcula como:

$$F_1 = 2\frac{Precisión \cdot Sensibilidad}{Precisión + Sensibilidad} \tag{57}$$

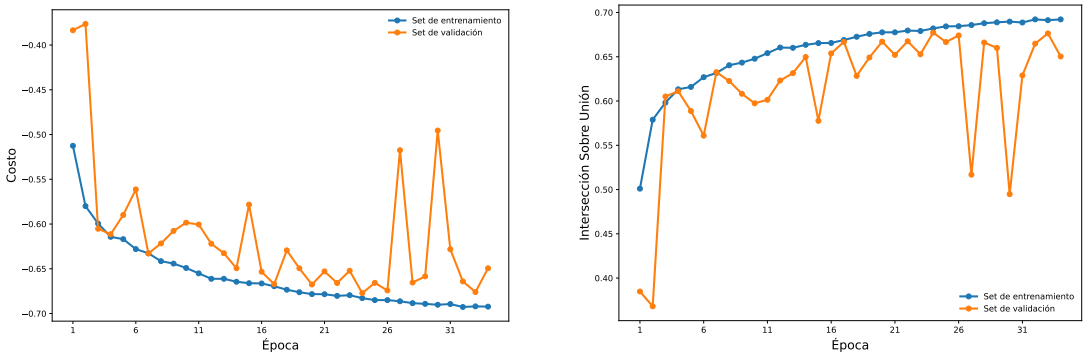
La tasa de falsa alarma indica la probabilidad de que una instancia verdaderamente negativa sea clasificada como negativa. Es decir, que un punto donde haya ruido no sea clasificado como una señal transiente. Debido a que esto implica que el modelo detecte un terremoto lento donde no lo hay, es fácil asociarlo a una falsa alarma. Este estadístico se calcula como:

$$Tasa\ de\ falsa\ alarma = \frac{FP}{FP + VN} \tag{58}$$

7. Resultados

7.1. Valores de la función de pérdida y métricas durante el entrenamiento

El entrenamiento de la red finalizó luego de 34 épocas. Al no disminuir la función de costo para el conjunto de validación durante 10 épocas, el algoritmo de Early Stopping detuvo el entrenamiento y el algoritmo de Callbacks guardó la versión del modelo (es decir, sus pesos y segos) correspondiente al de la época 24, donde se consiguió el valor de función de pérdida más bajo y la métrica más alta (Figura 23).



(a) (b)

Figura 23: Historial de valores de IoU como función de pérdida y métrica durante el entrenamiento. En (a): historial de los valores como función de pérdida para los conjuntos de entrenamiento (azul) y validación (naranjo). En (b): historial de las métricas calculadas por Keras para los mismos conjuntos: entrenamiento en azul y validación en naranjo.

Las métricas y valores de función de costo para los conjuntos de entrenamiento y validación durante la época 24 se muestran en el Cuadro 1.

	IoU (Función de pérdida)	**IoU (Métrica de Keras)**
Entrenamiento	-0.6828	0.6820
Validación	-0.6772	0.6773

Cuadro 1: Valores de la Intersección sobre Unión como función de pérdida y como métrica resultantes de la época 24, para los conjuntos de entrenamiento y de validación.

A pesar de haberse utilizado dos algoritmos diferentes, los valores de la IoU calculados tanto como función de pérdida y como métrica fueron muy similares en todas las épocas. Esto puede notarse en la Tabla 1 y en la Figura 23, donde se puede apreciar una notoria simetría.

Después de la época 24, si bien la IoU continuó teniendo un valor absoluto más cercano a 1 para el conjunto de entrenamiento, éste no fue el caso para el conjunto

de validación. En la época 27, la pérdida y métrica de la validación fueron de -0.5175 y 0.5168 respectivamente, mientras que en la época 30 fueron de -0.4955 y 0.4948. Estos casos pueden indican un sobreajuste en el modelo, donde aumenta su capacidad para realizar la tarea deseada con el conjunto de entrenamiento, pero se pierde generalización, como se demuestra con el conjunto de validación.

7.2. Detección de señales transientes en el conjunto de prueba

El modelo detectó transientes en los datos sintéticos de manera consistente, con pocos casos de transientes no detectadas y falsas alarmas. En general, las detecciones calzan en los puntos centrales de las señales transientes, pero existen discrepancias en los bordes (Figura 24), donde el modelo puede subestimar o sobrestimar la duración del terremoto lento generado.

Los resultados no consideran los puntos donde existan anomalías (en forma de picos con mucha distancia vertical respecto a los puntos cercanos) como presencia de terremotos lentos, sino que busca las secciones con pendientes notorias y consistentes en la serie de tiempo, en subida o en bajada, tal como se vería un terremoto lento. Sin embargo, debido a la naturaleza aleatoria del ruido de fondo, existen casos donde éste puede mantener una pendiente que el modelo detecta erróneamente como un terremoto lento, resultando casos donde se levantan falsas alarmas, tal como se muestra en la Figura (25). Por otro lado, también existen casos de señales transientes no detectadas por el modelo, principalmente en casos donde ésta sea difícil de detectar debido a la forma del ruido de fondo (Fig. 26).

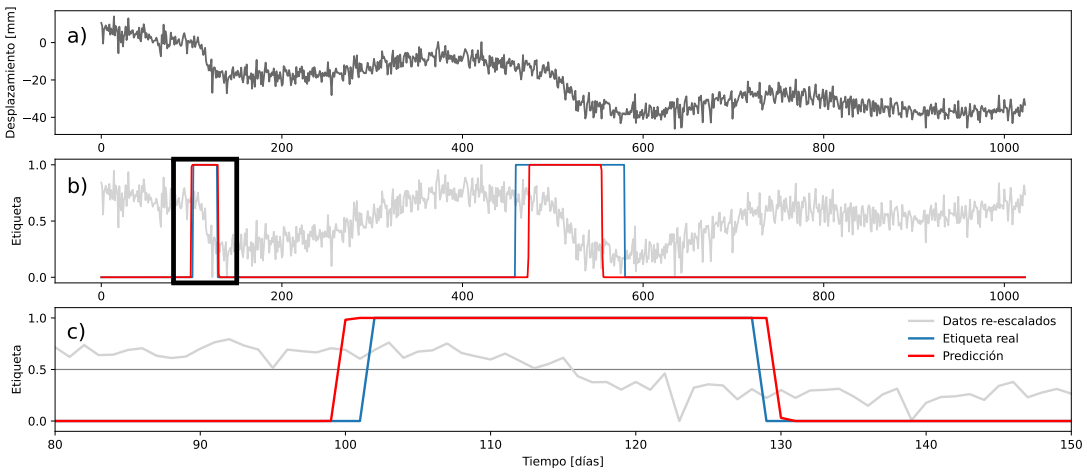


Figura 24: Serie de tiempo y etiquetas generadas en comparación a las predicciones del modelo. En (a): Ejemplo de una serie de tiempo GNSS sintética compuesta de ruido de fondo y dos señales transientes de tamaños y duraciones diferentes. En (b): En azul se representan las etiquetas generadas (reales) en base a las posiciones de las transientes, y en rojo se representa la probabilidad de una transiente predicha por el modelo. Ambas se superponen sobre la serie de tiempo re-escalada (gris claro) entre 0 y 1. En (c): Vista magnificada de la primera transiente, indicada con una caja negra en (b). Se indica con una línea recta el nivel de 0.5 de probabilidad.

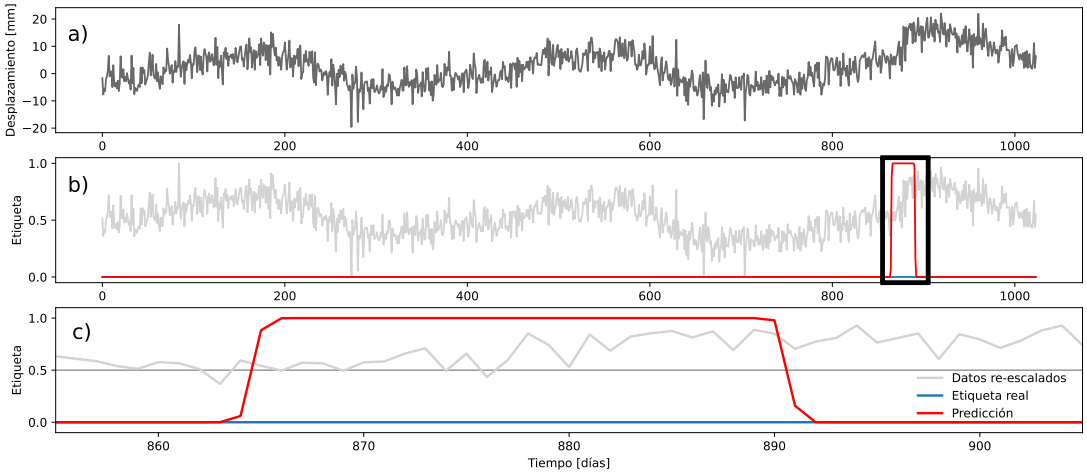


Figura 25: Ejemplo de una falsa alarma levantada por el modelo. En (a): Serie de tiempo GNSS sintética. En (b): Con azul se representan las etiquetas generadas (reales), y en rojo se representa la probabilidad de una transiente predicha. La serie de tiempo re-escalada entre 0 y 1 se representa en gris claro. En (c): Vista magnificada de la falsa alarma, indicada con una caja negra en (b).

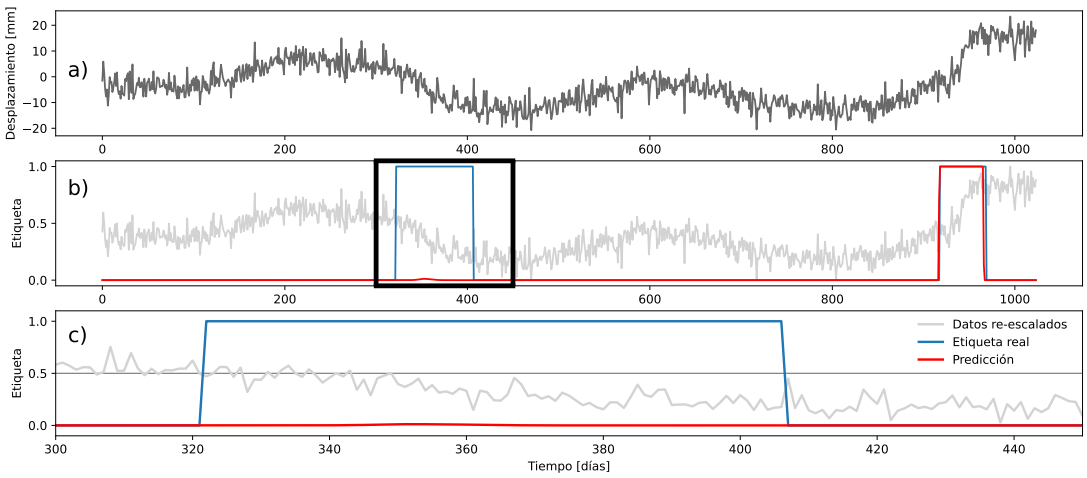


Figura 26: Ejemplo de una señal transiente no detectada por el modelo. En (a): Serie de tiempo GNSS sintética. En (b): Con azul se representan las etiquetas generadas (reales), y en rojo se representa la probabilidad de una transiente predicha. La serie de tiempo re-escalada entre 0 y 1 se representa en gris claro. En (c): Vista magnificada de la posición de la transiente no detectada, indicada con una caja negra en (b).

7.3. Pruebas de detección con tamaños y duraciones variables

La red neuronal detectó todas las señales transientes generadas para la evaluación pero subestimando la duración de ellas, tal como se observó en las detecciones del conjunto de prueba. Esto puede notarse en los ejemplos adjuntos en el Anexo B.

La alta precisión en todas las pruebas indica que el modelo detectó las transientes en las posiciones correctas, incluso para los casos donde el tamaño de la sigmoide generada fuese de pocos milímetros (Fig. 27). La sensibilidad del modelo se vió afectada por las subestimaciones de la duración de las transientes (Figuras 27) y 28)), ya que los puntos en los bordes de éstas, incorrectamente clasificados como ruido, son contados como falsos negativos. En el caso de la prueba con tamaño variable, las transientes de 1 y 2 milímetros tuvieron sensibilidades de 0.01 y 0.26, respectivamente. Para los tamaños mayores a 4 milímetros, la sensibilidad fue decayendo gradualmente hasta asentarse cerca de 0.6.

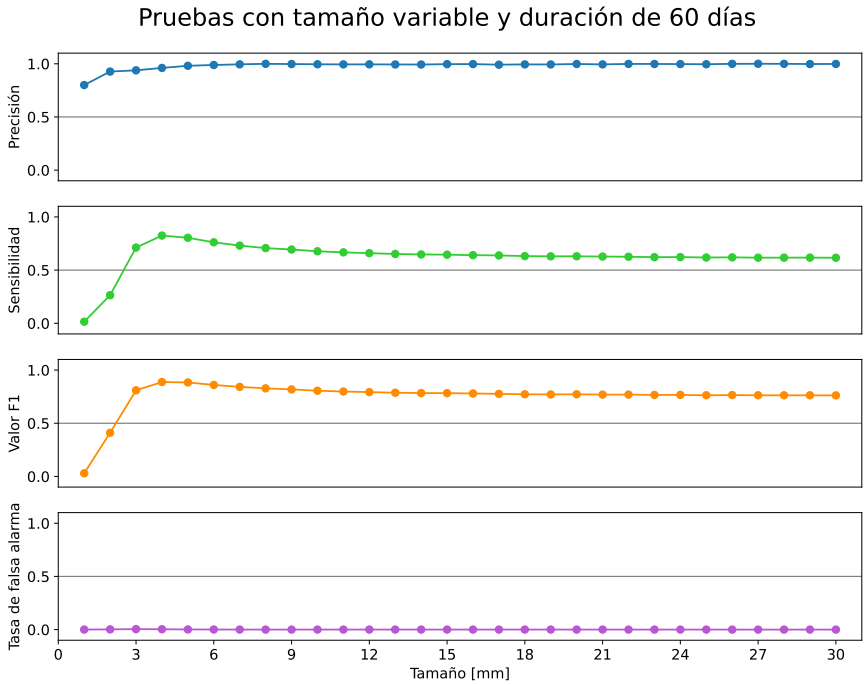


Figura 27: Métricas calculadas en cada prueba para evaluar el modelo, con tamaños de sigmoide entre 1 y 30 milímetros, y duraciones fijas de 60 días. El primer gráfico indica la precisión calculada para cada prueba, el segundo gráfico muestra la sensibilidad. El tercer gráfico indica el valor F1, el cual sigue la misma forma de la sensibilidad, ya que es la media geométrica entre sensibilidad y precisión. El gráfico de más abajo muestra la tasa de falsa alarma. Se indica con una línea gris horizontal el nivel de 0.5.

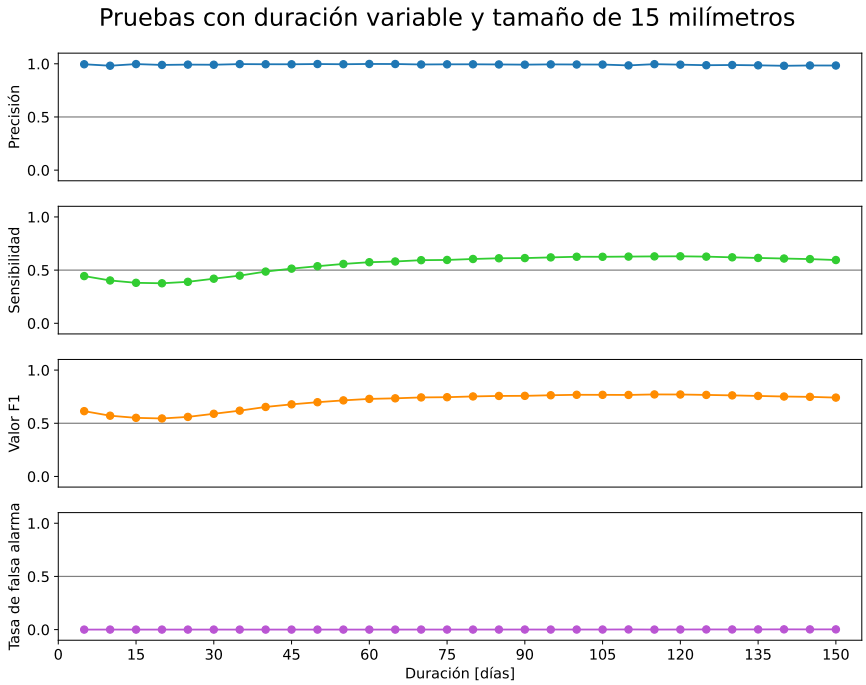


Figura 28: Métricas calculadas en cada prueba para evaluar el modelo, con duraciones entre 5 y 150 días (en intervalos de 5 días) y tamaños fijos de 15 milímetros. El primer gráfico indica la precisión calculada para cada prueba, el segundo gráfico muestra la sensibilidad. El tercer gráfico indica el valor F1, el cual sigue la misma forma de la sensibilidad. El gráfico de más abajo muestra la tasa de falsa alarma. Se indica con una línea gris horizontal el nivel de 0.5.

En las pruebas con duración variable, la precisión fue alta para todas las duraciones. La sensibilidad se mantuvo por debajo de 0.5 desde los 5 hasta los 40 días, mientras que desde los 80 a 150 días se mantuvo cerca de 0.6.

Tanto para las pruebas de tamaños como de duraciones variables, el valor F1 siguió la forma de la sensibilidad pero con valores más altos, dado que las precisiones se mantuvieron altas. En ambas pruebas este valor se mantuvo sobre 0.7 para tamaños y duraciones altas. Para tamaños de 1 y 2 milímetros, el valor F1 fue de 0.02 y 0.41 respectivamente. Los valores más altos fueron para los tamaños entre 3 y 10 milímetros, y para las duraciones entre 100 y 140 días.

En las dos pruebas, para todos los tamaños y duraciones, las falsas alarmas se mantuvieron cerca de 0.

8. Discusión

El modelo presentado en este trabajo muestra una buena capacidad para detectar señales transientes en los datos de prueba, a pesar de no tener valores de precisión y sensibilidad cercanos a 1 en todos los tests de detección. En general, el modelo es capaz de detectar los puntos más notorios de la ocurrencia de las transientes. Es decir, los puntos centrales de ésta, donde la pendiente es más pronunciada. Los puntos iniciales y finales de las sigmoides, donde su pendiente se aproxima a cero, no son detectados con tanta claridad.

Conseguir detecciones perfectas que detecten los primeros y últimos días de la ocurrencia de estas señales es un problema que aumenta rápidamente en complejidad, ya que tanto los puntos de inicio y final, como también transientes completas pero de menor tamaño, pueden estar oscurecidas por el ruido de fondo de las series de tiempo. Por otro lado, buscar un modelo más sensible puede significar un aumento en la cantidad de falsos positivos si es que el ruido de fondo presenta características similares a los de un terremoto lento de poco tamaño y pocos días.

Por otra parte, es más ventajoso detectar todos los terremotos lentos que se hayan podido registrar con una estación GNSS, aunque esto signifique el aumento (dentro de un nivel razonable) de los falsos positivos, ya que el ruido no tiene correlación espacial entre diferentes estaciones cerca de un mismo punto geográfico. Una transiente, en cambio, está presente con una potencia que depende de la cercanía de la estación al área de deslizamiento. Por lo tanto, los falsos positivos son identificables si éstos fueron entregados por el modelo en una sola estación y no hay presencia de un deslizamiento transiente en ninguna estación cercana.

Como parte del trabajo futuro de este tipo de trabajos, existen dos puntos importantes a tener en consideración. El primero, es que, a pesar de que los estadísticos reflejen buenos resultados de detección para este modelo, la mejor forma de evidenciar su utilidad es poniéndolo a prueba con datos reales, y comparando los resultados obtenidos con los de otras investigaciones dentro de la misma zona de estudio. De esta manera, se puede contrastar la sensibilidad del modelo frente a otras metodologías, como también la cantidad de falsas alarmas que entregue para distintos modelos de ruido, según la zona de estudio a la que se aplique. Esto también permitirá llevar a cabo los ajustes necesarios para mejorar la capacidad de detección la red. El segundo punto, es que para el entrenamiento de esta red neuronal, sólo se utilizaron algunos tipos de ruido común en las series de tiempo GNSS y transientes, lo cual implica la necesidad de que los datos reales hayan sido pre-procesados para quitarles la tendencia y las anomalías, como también las discontinuidades relacionadas a cambios de antena, dejando sólo las componentes con las cuales se entrenó el modelo. Esto puede significar un problema, ya que el procesamiento de los datos puede disminuir la potencia de algunas señales presentes en las series de tiempo, tales como los deslizamientos lentos. Un entrenamiento con todo tipo de componentes, incluyendo anomalías y ventanas de tiempo sin datos (como suele ocurrir en los datos reales), podría aumentar la robusticidad de una red neuronal para eliminar la necesidad del pre-procesamiento de los datos.

9. Conclusión

Este estudio propone una metodología para la identificación de señales transientes en series de tiempo GNSS a través de una segmentación semántica de los datos, realizada una red neuronal convolucional. Esta red es una modificación de la U-Net, originalmente utilizada para la clasificación de imágenes médicas. Este modelo se entrenó utilizando datos sintéticos que incluyen ruido blanco y ruido coloreado, como también anomalías y sigmoides que simulan la forma de los deslizamientos lentos (SSEs), con tamaños y duraciones aleatorias. Se realizaron pruebas con transientes que varían sólo en duración o sólo en tamaño, para medir la capacidad del modelo en la detección de transientes con distintas amplitudes y extensiones temporales. De estas pruebas, se calcularon la precisión, sensibilidad, norma F1 y tasa de falsa alarma. La precisión se mantuvo alta para todas las pruebas, pero la sensibilidad generalmente se mantuvo cerca de 0.5. Esto es debido a que el modelo detectó todas las transientes generadas, pero tuvo problemas para detectar con claridad el inicio y final exactos de las transientes, manteniendo, en cambio, una buena capacidad de detección en los puntos centrales de la transiente, donde las pendientes son más notorias. El valor F1 se mantuvo alrededor de 0.7 para la mayoría de las pruebas y la tasa de falsa alarma fue aproximadamente 0 en todas.

Esta red es una herramienta prometedora, ya que es capaz de detectar transientes de hasta 2 milímetros con gran precisión. Hasta ahora, sólo se han detectado transientes en datos GNSS sintéticos. El siguiente paso para desarrollar esta metodología es su aplicación en datos reales, integrando una correlación espacial entre componentes este y norte de varias estaciones de la misma red para asegurar que la detección de los SSE sea espacialmente consistente.

A. Anexo: Implementación en Python de la generación de series de tiempo sintéticas

En este anexo se presenta la implementación en Python del método de generación de datos sintéticos, explicado en la Sección 6.1.

A.1. Generación del ruido de fondo

Para la generación del ruido de fondo, primero se generan vectores de largo N con cada una de las componentes del ruido, para luego sumar cada vector. En el caso de este trabajo, se crearon vectores de largo 1024.

El ruido estacional se generó con la función *seasonal*, de la forma:

```
def seasonal(n, an, sa, dt=1/365.25):
    t = dt*np.arange(n)
    c1 = an[0] + (an[1] - an[0])*np.random.rand()
    c2 = sa[0] + (sa[1] - sa[0])*np.random.rand()
    d = c1*np.sin(2*np.pi*(t + np.random.rand())) + c2*np.sin(4*
        ↪ np.pi*(t + np.random.rand()))
    return d
```

El ruido blanco se generó utilizando la función *white_noise*, como se presenta a continuación:

```
def white_noise(n):
    y = np.random.randn(n)
    return y
```

El ruido rojo se generó usando la función *random_walk*:

```
def random_walk(n,dt=1):
    L = np.tril(np.ones((n,n)))
    w = np.random.randn(n)
    y = np.dot(L,w)

    return dt**0.5*y
```

El ruido rosado se generó usando la función *flicker_noise*:

```
def flicker_noise(n,dt=1):
    if np.remainder(n,2):
        nt = n + 1
    else:
        nt = n

    x = np.random.randn(nt)
    X = np.fft.fft(x)
```

```
    nf = n/2 + 1
    k = np.arange(nf,dtype=np.int64)

    X = X[k]
    X = X/np.sqrt(k+1)

    X = np.concatenate((X,np.conj(X[-1:0:-1])))

    y = np.real(np.fft.ifft(X))
    y = y[:n]

    y = y - np.mean(y)
    y = y/np.std(y)

    return dt**0.25*y
```

Los ruidos blanco, rosado y rojo se unen en una sola componente llamado ruido coloreado, utilizado la función *colored_ noise*:

```
def colored_noise(n, wn, fn, rw, dt=1/365.25):
    c1 = wn[0] + (wn[1] - wn[0])*np.random.rand()
    c2 = fn[0] + (fn[1] - fn[0])*np.random.rand()
    c3 = rw[0] + (rw[1] - rw[0])*np.random.rand()
    d = c1*white_noise(n) + c2*flicker_noise(n,dt) + c3*
        ↪ random_walk(n,dt)
    return d
```

Mientas que para la generación de anomalías, se utilizó la función *outliers*:

```
def outliers(n, max_size, prob):
    c = np.random.rand(n) < prob
    d = c*max_size*(np.random.rand(n)-0.5)*2
    return d
```

Finalmente, todas estas componentes se unen en la función *random_ noise*:

```
def random_noise(nt, noise_type=0):
  if noise_type == 0:
    d_seasonal = seasonal(nt, [4.5,6.9], [0.7,1.5])
    d_noise = colored_noise(nt, [2.0, 2.9], [2.3, 5.9], [0, 2.9])
    d_outliers = outliers(nt, 10, 0.05)
  else:
    d_seasonal = seasonal(nt, [1,2], [0.2,0.5])
    d_noise = colored_noise(nt, [0.4, 0.9], [0.3, 1.6], [0, 1.4])
    d_outliers = outliers(nt, 5, 0.05)
  return d_seasonal + d_noise + d_outliers
```

A.2. Generación de las señales transientes asociadas a terremotos lentos

Para incluir la componente del deslizamiento lento, se agregaron sigmoides con parámetros como su tamaño, duración y punto de inicio elegidos de forma aleatoria entre un rango de valores. La creación de cada sigmoide se definió con la función *transient*, la cual tiene la forma:

```
def transient(n, size, duration, center):
    t = np.arange(0,n)
    d = size/(1+np.exp(-(t-center)/(duration/10.0), dtype=np.
        ↪ float128))
    return d
```

A.3. Generación del conjunto de datos compuestos de ruido de fondo y transientes

El siguiente código de Python muestra el algoritmo que genera los datos sintéticos utilizados en este trabajo. Considerando un largo de nt valores, se generan un vector compuesto de ruido de fondo y transientes ($d\_transient + d\_noise$), y un vector de etiquetas *label*.

```
def make_dataset(nt, event_num, noise_type, label_snr_threshold):
  label = np.zeros(nt)
  d_noise = random_noise(nt, noise_type)
  n_event = np.random.choice(event_num)
  if n_event != 0:
      event_window_len = nt/n_event
  else:
      event_window_len = 50
  d_transient = 0
  for i in range(n_event):
    size = np.random.choice([-1,1])*20*np.random.rand()
    duration = 10 + 120*np.random.rand()
    center = np.random.randint(i*event_window_len + 70, (i+1)*
        ↪ event_window_len - 70)
    d_transient += transient(nt,size,duration,center)

    index_start = int(center - duration/2)
    index_stop = int(center + duration/2)
    if label_snr_threshold is None:
      label[index_start: index_stop] = 1
    else:
      if snr(d_transient[index_start: index_stop], d_noise[
          ↪ index_start: index_stop]) > label_snr_threshold:
        label[index_start: index_stop] = 1
```

```
    return d_transient + d_noise, label, d_transient
```

B. Anexo: Ejemplos de detecciones en las series de tiempo generadas para las pruebas de detección

Para la evaluación del modelo de detección de transientes, descrito en la Sección 7.3, se generaron dos datos sintéticos con una sola transiente por serie de tiempo. El primer tipo de dato consta de una transiente con duración fija de 60 días y con tamaños entre 1 a 30 milímetros. El segundo tipo es con una idea similar, pero manteniendo fijo un tamaño de 15 milímetros, pero con duraciones que varían entre 5 a 150 días, en pasos de 5. En este anexo se muestran ejemplos de las series de tiempo generadas.

B.1. Detecciones en pruebas con tamaños variables

En esta subsección se muestran figuras con duraciones de transiente de 60 días, pero con tamaños de 6 y 20 milímetros (Figuras 29 y 30, respectivamente). En las figuras mostradas en este anexo de pueden ver las etiquetas reales, y las detecciones del modelo.

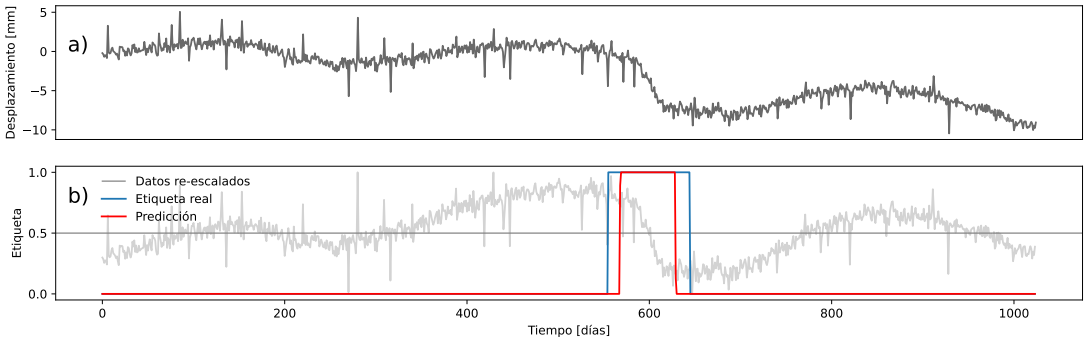


Figura 29: Detección sobre una serie de tiempo con una transiente de duración de 60 días y tamaño de 6 milímetros. En (a): Serie de tiempo GNSS sintética. En (b): Con azul se representan la etiquetas generadas (reales), y en rojo se representa la probabilidad de una transiente predicha. La serie de tiempo re-escalada entre 0 y 1 se representa en gris claro.

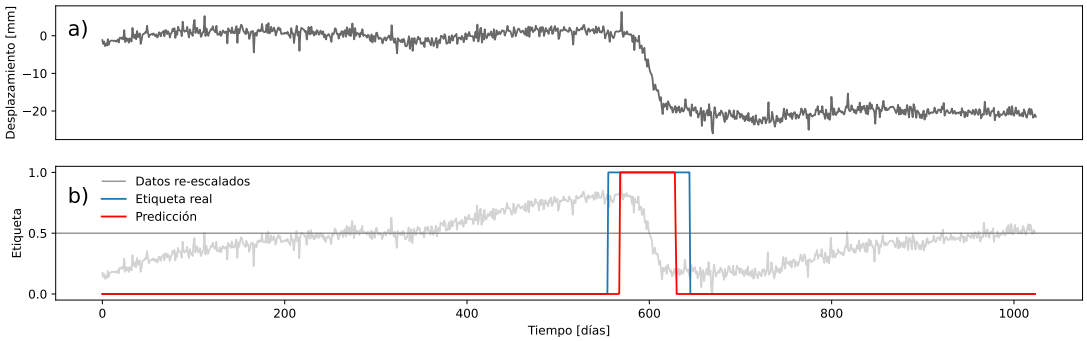


Figura 30: Detección sobre una serie de tiempo con una transiente de duración de 60 días y tamaño de 20 milímetros. En (a): Serie de tiempo GNSS sintética. En (b): Con azul se representan la etiquetas generadas (reales), y en rojo se representa la probabilidad de una transiente predicha. La serie de tiempo re-escalada entre 0 y 1 se representa en gris claro.

B.2. Detecciones en pruebas con duraciones variables

Las Figuras 31 y 32 muestran ejemplos de series de tiempo con transientes de 15 milímetros de tamaño, pero con duraciones de 20 y 110 días respectivamente.

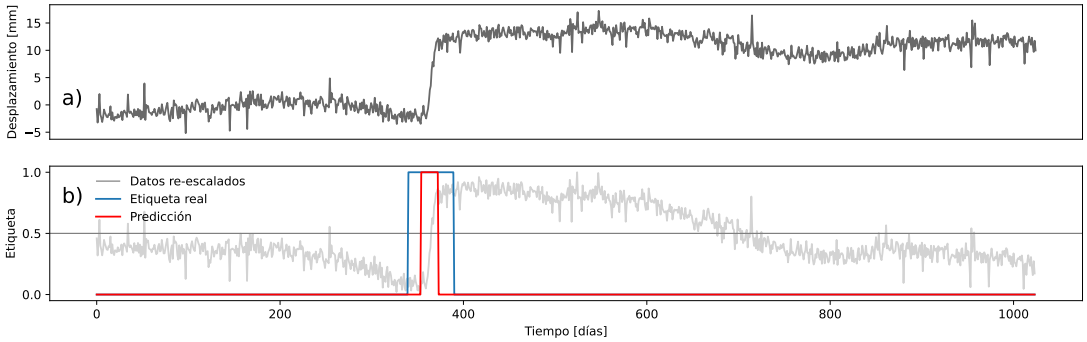


Figura 31: Detección sobre una serie de tiempo con una transiente de duración de 20 días y tamaño de 15 milímetros. En (a): Serie de tiempo GNSS sintética. En (b): Con azul se representan la etiquetas generadas (reales), y en rojo se representa la probabilidad de una transiente predicha. La serie de tiempo re-escalada entre 0 y 1 se representa en gris claro.

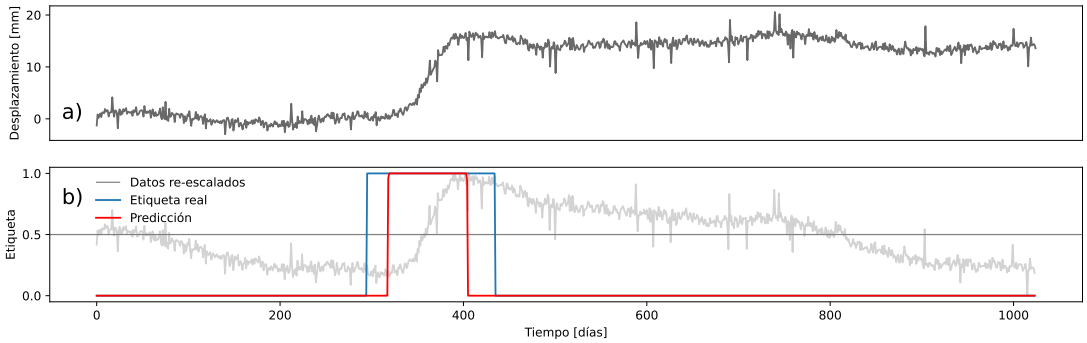


Figura 32: Detección sobre una serie de tiempo con una transiente de duración de 110 días y tamaño de 15 milímetros. En (a): Serie de tiempo GNSS sintética. En (b): Con azul se representan la etiquetas generadas (reales), y en rojo se representa la probabilidad de una transiente predicha. La serie de tiempo re-escalada entre 0 y 1 se representa en gris claro.

En las cuatro figuras mostradas en este anexo, se puede notar que la detección de las transientes resulta en un vector de etiquetas predichas, donde las predicciones tienen duraciones menores a las de las etiquetas reales. En los cuatro casos mostrados, la detección ocurre en los puntos donde se nota más claramente la pendiente de la transiente en los datos, mientras que en los bordes, donde la sigmoide tiene menor pendiente, no se detecta transiente.

Bibliografía

[1] Johnson, J. M., & Satake, K. (1999). Asperity distribution of the 1952 great Kamchatka earthquake and its relation to future earthquake potential in Kamchatka. Seismogenic and tsunamigenic processes in shallow subduction zones, 541-553.

[2] Melnick, D., Li, S., Moreno, M., Cisternas, M., Jara-Muñoz, J., Wesson, R., ... & Deng, Z. (2018). Back to full interseismic plate locking decades after the giant 1960 Chile earthquake. Nature communications, 9(1), 3527.

[3] Kanamori, H. (1970). The Alaska earthquake of 1964: Radiation of long-period surface waves and source mechanism. Journal of Geophysical Research, 75(26), 5029-5040.

[4] Moreno, M., Rosenau, M., & Oncken, O. (2010). 2010 Maule earthquake slip correlates with pre-seismic locking of Andean subduction zone. Nature, 467(7312), 198-202.

[5] Reid, H. F. (1909). Seismological Notes. Proceedings of the American Philosophical Society, 48(192), 303–312. http://www.jstor.org/stable/984159

[6] Van Dam, T., Wahr, J., Milly, P. C. D., Shmakin, A. B., Blewitt, G., Lavallée, D., & Larson, K. M. (2001). Crustal displacements due to continental water loading. Geophysical Research Letters, 28(4), 651-654.

[7] Ide, S., Beroza, G. C., Shelly, D. R., & Uchide, T. (2007). A scaling law for slow earthquakes. Nature, 447(7140), 76-79.

[8] Kumazawa, T., Ogata, Y., & Toda, S. (2010). Precursory seismic anomalies and transient crustal deformation prior to the 2008 Mw= 6.9 Iwate-Miyagi Nairiku, Japan, earthquake. Journal of Geophysical Research: Solid Earth, 115(B10).

[9] Melbourne, T. I., & Webb, F. H. (2002). Precursory transient slip during the 2001 Mw= 8.4 Peru earthquake sequence from continuous GPS. Geophysical Research Letters, 29(21), 28-1.

[10] Nishikawa, T., Ide, S., & Nishimura, T. (2023). A review on slow earthquakes in the Japan Trench. Progress in Earth and Planetary Science, 10(1), 1-51.

[11] Donoso, F., Yáñez, V., Ortega-Culaciati, F., & Moreno, M. (2023). A machine learning approach for slow slip event detection using GNSS time-series. Journal of South American Earth Sciences, 132, 104680.

[12] Xue, X., & Freymueller, J. T. (2023). Machine learning for single-station detection of transient deformation in GPS time series with a case study of Cascadia slow slip. Journal of Geophysical Research: Solid Earth, 128, e2022JB024859. https://doi.org/10.1029/2022JB024859

[13] He, B., Wei, M., Watts, D. R., & Shen, Y. (2020). Detecting slow slip events from seafloor pressure data using machine learning. Geophysical Research Letters, 47(11), e2020GL087579.

[14] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science(), vol 9351. Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28

[15] Dolphin, G. (2018, January). Online extra: The history of elastic rebound theory: How a big disaster helped us better understand how the earth works. In The Trenches - January 2018. https://nagt.org/nagt/publications/trenches/v8-n1/196287.html

[16] What is a subduction zone. Subduction Zones in four Dimensions. https://www.sz4d.org/general-8

[17] Govers, R., Furlong, K. P., Van de Wiel, L., Herman, M. W., & Broerse, T. (2018). The geodetic signature of the earthquake cycle at subduction zones: Model constraints on the deep processes. Reviews of Geophysics, 56(1), 6-49.

[18] Hughes, J. F., Mathewes, R. W., & Clague, J. J. (2002). Use of pollen and vascular plants to estimate coseismic subsidence at a tidal marsh near Tofino, British Columbia. Palaeogeography, Palaeoclimatology, Palaeoecology, 185(1-2), 145-161.

[19] Bilek, S. L., & Lay, T. (2018). Subduction zone megathrust earthquakes. Geosphere, 14(4), 1468-1500.

[20] Kanamori, H. (1972). Mechanism of tsunami earthquakes. Physics of the earth and planetary interiors, 6(5), 346-359.

[21] Bedford, J. R., Moreno, M., Deng, Z., Oncken, O., Schurr, B., John, T., Báez, J. C. & Bevis, M. (2020). Months-long thousand-kilometre-scale wobbling before great subduction earthquakes. Nature, 580(7805), 628-635.

[22] Wallace, L. M., Hreinsdóttir, S., Ellis, S., Hamling, I., D'Anastasio, E., & Denys, P. (2018). Triggered slow slip and afterslip on the southern Hikurangi subduction zone following the Kaikōura earthquake. Geophysical Research Letters, 45(10), 4710-4718.

[23] Radiguet, M., Perfettini, H., Cotte, N., Gualandi, A., Valette, B., Kostoglodov, V., ... & Campillo, M. (2016). Triggering of the 2014 M w 7.3 Papanoa earthquake by a slow slip event in Guerrero, Mexico. Nature Geoscience, 9(11), 829-833.

[24] Donoso F, Moreno M, Ortega-Culaciati F, Bedford JR and Benavente R (2021) Automatic Detection of Slow Slip Events Using the PICCA: Application to Chilean GNSS Data. Front. Earth Sci. 9:788054. doi: 10.3389/feart.2021.788054

[25] Bedford, J., & Bevis, M. (2018). Greedy automatic signal decomposition and its application to daily GPS time series. Journal of Geophysical Research: Solid Earth, 123(8), 6992-7003.

[26] Köhne, T., Riel, B., & Simons, M. (2023). Decomposition and inference of sources through spatiotemporal analysis of network signals: The DISSTANS python package. Computers & Geosciences, 170, 105247.

[27] Kaplan, E. D., & Hegarty, C. (Eds.). (2017). Understanding GPS/GNSS: principles and applications. Artech house.

[28] Hormazábal, J. (2024). Caracterización de señales de deformación cortical en series de tiempo GNSS en el norte de Chile. [Tesis de maestría, Universidad de Chile]

[29] Davis, J. L., Wernicke, B. P., & Tamisiea, M. E. (2012). On seasonal signals in geodetic time series. Journal of Geophysical Research: Solid Earth, 117(B1).

[30] Pehlivan, H. (2024). A novel outlier detection method based on Bayesian change point analysis and Hampel identifier for GNSS coordinate time series. EURASIP Journal on Advances in Signal Processing, 2024(1), 44.

[31] Herring, T. A., Davis, J. L., y Shapiro, I. I. (1990). Geodesy by radio interferometry: The application of kalman filtering to the analysis of very long baseline interferometry data. Journal of Geophysical Research: Solid Earth, 95 (B8), 12561–12581.

[32] Dong, D., Herring, T., y King, R. W. (1998). Estimating regional deformation from a combination of space and terrestrial geodetic data. Journal of Geodesy, 72 , 200–214.

[33] Heki K (2001) Seasonal modulation of interseismic strain buildup in Northeastern Japan driven by snow loads. Science 293:89–92

[34] Bevis, M., Brown, A. Trajectory models and reference frames for crustal motion geodesy. J Geod 88, 283–311 (2014). https://doi.org/10.1007/s00190-013-0685-5

[35] Simpson, R. W., Thatcher, W., & Savage, J. C. (2012). Using cluster analysis to organize and explore regional GPS velocities. Geophysical Research Letters, 39(18).

[36] Hulbert, C., Rouet-Leduc, B., Johnson, P. A., Ren, C. X., Rivière, J., Bolton, D. C., & Marone, C. (2019). Similarity of fast and slow earthquakes illuminated by machine learning. Nature Geoscience, 12(1), 69-74.

[37] Rouet-Leduc, B., Hulbert, C., McBrearty, I. W., & Johnson, P. A. (2020). Probing slow earthquakes with deep learning. Geophysical research letters, 47(4), e2019GL085870.

[38] Crowell, B. W., Bock, Y., & Liu, Z. (2016). Single-station automated detection of transient deformation in GPS time series with the relative strength index: A case study of Cascadian slow slip. Journal of Geophysical Research: Solid Earth, 121(12), 9077-9094.

[39] Mitchell, T. M. (1997). Machine learning (Vol. 1). McGraw-hill New York.

[40] Benuzillo J, Caine W, Evans RS, Roberts C, Lappe D, Doty J. Predicting readmission risk shortly after admission for CABG surgery. J Card Surg. 2018 Apr;33(4):163-170. doi: 10.1111/jocs.13565. Epub 2018 Mar 23. PMID: 29569750.

[41] Jin, Z., Lim, D.D., Zhao, X. et al. Machine learning enabled optimization of showerhead design for semiconductor deposition process. J Intell Manuf 35, 925–935 (2024). https://doi.org/10.1007/s10845-023-02082-8

[42] W. Ahmed et al., "Machine Learning Based Energy Management Model for Smart Grid and Renewable Energy Districts," in IEEE Access, vol. 8, pp. 185059-185078, 2020, doi: 10.1109/ACCESS.2020.3029943.

[43] M. H. Abd El-Jawad, R. Hodhod and Y. M. K. Omar, "Sentiment Analysis of Social Media Networks Using Machine Learning,"2018 14th International Computer Engineering Conference (ICENCO), Cairo, Egypt, 2018, pp. 174-176, doi: 10.1109/ICENCO.2018.8636124.

[44] Orogun, A., & Onyekwelu, B. (2019). Predicting consumer behaviour in digital market: a machine learning approach.

[45] Haykin, S. (2009). Neural networks and learning machines, 3/E. Pearson Education India.

[46] Chen, L., Li, R., Liu, Y., Zhang, R., & Woodbridge, D. M. K. (2017, August). Machine learning-based product recommendation using Apache Spark. In 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI) (pp. 1-6). IEEE.

[47] Lee, S., Kim, Y., Kahng, H., Lee, S. K., Chung, S., Cheong, T., ... & Kim, S. B. (2020). Intelligent traffic control for autonomous vehicle systems based on machine learning. Expert Systems with Applications, 144, 113074.

[48] McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943). https://doi.org/10.1007/BF02478259

[49] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press. http://www.deeplearningbook.org/

[50] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.

[51] Janiesch, C., Zschech, P. & Heinrich, K. Machine learning and deep learning. Electron Markets 31, 685–695 (2021). https://doi.org/10.1007/s12525-021-00475-2

[52] N. K. Chauhan and K. Singh, "A Review on Conventional Machine Learning vs Deep Learning," 2018 International Conference on Computing, Power and Communication Technologies (GUCON), Greater Noida, India, 2018, pp. 347-352, doi: 10.1109/GUCON.2018.8675097.

[53] LeCun, Y., Bengio, Y., & Hinton, G. (2015). "Deep learning."Nature, 521(7553), 436-444.

[54] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. Towards Data Sci, 6(12), 310-316.

[55] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

[56] Shenfield A, Howarth M. A Novel Deep Learning Model for the Detection and Identification of Rolling Element-Bearing Faults. Sensors. 2020; 20(18):5112. https://doi.org/10.3390/s20185112

[57] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). O'Reilly Media.

[58] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

[59] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

[60] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). pmlr.

[61] Gupta, A., Ramanath, R., Shi, J., & Keerthi, S. S. (2021, December). Adam vs. sgd: Closing the generalization gap on image classification. In OPT2021: 13th Annual Workshop on Optimization for Machine Learning.

[62] van Beers, F. (2018). Using intersection over union loss to improve binary image segmentation (Doctoral dissertation).

[63] "Receptive Fields and Functional Architecture of Monkey Striate Cortex," D. Hubel and T. Wiesel (1968)

[64] "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," K. Fukushima (1980).

[65] "Receptive Fields of Single Neurones in the Cat's Striate Cortex," D. Hubel and T. Wiesel (1959)

[66] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[67] Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems, 33(12), 6999-7019.

[68] U. Erdenebayar, H. Kim, J.-U. Park, D. Kang, and K.-J. Lee, "Automatic prediction of atrial fibrillation based on convolutional neural network using a short-term normal electrocardiogram signal," J. Korean Med. Sci., vol. 34, no. 7, p. e64, 2019.

[69] S. Harbola and V. Coors, "One dimensional convolutional neural network architectures for wind prediction," Energy Convers. Manage., vol. 195, pp. 70–75, Sep. 2019

[70] D. Han, J. Chen, and J. Sun, "A parallel spatiotemporal deep learning network for highway traffic flow forecasting," Int. J. Distrib. Sensor Netw., vol. 15, no. 2, pp. 1–12, 2019.

[71] Q. Zhang, D. Zhou, and X. Zeng, "HeartID: A multiresolution convolutional neural network for ECG-based biometric human identification in smart health applications," IEEE Access, vol. 5, pp. 11805–11816, 2017.

[72] O. Abdeljaber, S. Sassi, O. Avci, S. Kiranyaz, A. A. Ibrahim, and M. Gabbouj, "Fault detection and severity identification of ball bearings by online condition monitoring," IEEE Trans. Ind. Electron., vol. 66, no. 10, pp. 8136–8147, Oct. 2019.

[73] Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1492-1500).

[74] Y. Jiang, L. Chen, H. Zhang, and X. Xiao, "Breast cancer histopathological image classification using convolutional neural networks with small SE-ResNet module," PLoS ONE, vol. 14, no. 3, Mar. 2019, Art. no. e0214587.

[75] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 779–788.

[76] W. Liu et al., "SSD: Single shot MultiBox detector," in Proc. Eur. Conf. Comput. Vis. Cham, Switzerland: Springer, 2016, pp. 21–37.

[77] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 3431–3440

[78] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in Proc. IEEE Int. Conf. Comput. Vis., Oct. 2017, pp. 2961–2969

[79] Kirillov, A., He, K., Girshick, R., Rother, C., & Dollár, P. (2019). Panoptic segmentation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 9404-9413).

[80] L. Han, H. Liang, H. Chen, W. Zhang and Y. Ge, Çonvective Precipitation Nowcasting Using U-Net Model,ïn IEEE Transactions on Geoscience and Remote Sensing, vol. 60, pp. 1-8, 2022, Art no. 4103508, doi: 10.1109/TGRS.2021.3100847

[81] Yin L, Wang L, Li T, Lu S, Tian J, Yin Z, Li X, Zheng W. U-Net-LSTM: Time Series-Enhanced Lake Boundary Prediction Model. Land. 2023; 12(10):1859. https://doi.org/10.3390/land12101859

[82] Perslev, M., Jensen, M., Darkner, S., Jennum, P. J., & Igel, C. (2019). U-time: A fully convolutional network for time series segmentation applied to sleep staging. Advances in Neural Information Processing Systems, 32.

[83] Langbein, J., & Svarc, J. L. (2019). Evaluation of temporally correlated noise in Global Navigation Satellite System time series: Geodetic monument performance. Journal of Geophysical Research: Solid Earth, 124(1), 925-942.