UNIVERSIDAD DE PIURA

FACULTAD DE INGENIERÍA

Aplicación de Machine Learning para pronóstico de desplazamiento de lluvias usando imágenes del radar de lluvias de UDEP

Tesis para optar el título de
Ingeniero Mecánico Eléctrico

Pool Domarvi Nolasco Ramírez

Asesor:
Dr. Ing. Rodolfo Rodríguez Arismendiz

Piura, marzo de 2023

NOMBRE DEL TRABAJO

Tesis_Pool_Nolasco_Ramirez 06022023.docx

RECUENTO DE PALABRAS

28288 Words

RECUENTO DE CARACTERES

159359 Characters

RECUENTO DE PÁGINAS

116 Pages

TAMAÑO DEL ARCHIVO

4.4MB

FECHA DE ENTREGA

Feb 6, 2023 5:16 PM GMT-5

FECHA DEL INFORME

Feb 6, 2023 5:18 PM GMT-5

14% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base d

- 12% Base de datos de Internet
- 2% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossr
- 8% Base de datos de trabajos entregados

Excluir del Reporte de Similitud

- Material bibliográfico
- Material citado
- Coincidencia baja (menos de 10 palabras)

Rodolfo Rodriguez
16-02-2023

A Dios y a mi familia,
por el apoyo constante
de cada día.

UNIVERSITAS STUDIORUM PIURENSIS

Resumen

En esta tesis se usaron conceptos de Machine Learning (ML), específicamente técnicas de Deep Learning (Aprendizaje profundo) en el desarrollo de un modelo que utilice imágenes obtenidas de un radar escáner de lluvias. Los datos fueron obtenidos del radar de lluvias de la Universidad de Piura, así como también de otras instituciones que nos facilitaron el libre acceso.

El objetivo principal de realizar un modelo de Deep Learning consiste en el intento de implementar un sistema de pronóstico de desplazamiento de tormentas y con ello en un futuro sirva como una herramienta para un sistema de alerta temprana frente posibles fenómenos que afecten a la población de la región Piura.

La metodología para su desarrollo incluye la adquisición de imágenes desde el software de control del radar de lluvias, además del pre y post procesamiento de los datos recopilados. Finalmente, un ordenamiento en secuencias de cuadros para entrada al modelo de ML.

Durante la parte de experimentación del modelo construido, se probaron diferentes versiones del mismo cambiando la arquitectura del modelo, así como sus hiperparámetros. Se realizaron cambios en cada versión del modelo en cuanto a la cantidad del número de capas, unidades de cada capa, funciones de activación y funciones de costo para ver la influencia de cada hiperparámetro en el modelo.

Los resultados obtenidos en la validación con datos nuevos y luego de ajustar los hiperparámetros del modelo indicaron un buen índice de similitud respecto de datos reflejados en la realidad.

UNIVERSITAS STUDIORUM PIURENSIS

Tabla de contenido

UNIVERSITAS STUDIORUM PIURENSIS

Lista de figuras

Introducción

El ML es una rama de la inteligencia artificial que se viene desplegando en el sector empresarial y la industria de la automatización. Específicamente en los radares se está aplicando para el pronóstico intensidad de lluvia y alerta temprana, en su mayoría, en institutos de meteorología del continente asiático (Han et al., 2016; Shi et al., 2015, 2017; Tran & Song, 2019; Wang et al., 2017). Se quiere conseguir un pronóstico inmediato, de largo plazo y lo más asemejado a la realidad.

Este proyecto de tesis se centra en diseñar un modelo basado en *Machine Learning* (Aprendizaje Máquina) para predecir secuencias futuras de celdas de precipitación. Los servicios de meteorología en países del primer mundo están implementando estas técnicas debido a la gran cantidad de datos que manejan y pueden abarcar un modelo general usando algoritmos de inteligencia artificial.

Para el desarrollo de este proyecto primeramente se dispusieron de los datos del radar de lluvias instalado en la estación científica "Ramón Mugica" de la Universidad de Piura. También se obtuvieron datos extras de fuentes externas de la Universidad de Sao Paolo, quien tenía libre el acceso a los datos (Bonnet et al., 2020a). Posteriormente se requirió hacer un tratamiento a los datos y preparación para la entrada en el modelo.

Los pasos que se dieron en la realización de este estudio, fueron los siguientes:

- En primer lugar, el reconocimiento de los datos (imágenes proporcionadas por el radar de lluvias), y el tratamiento necesario para conseguir datos útiles en el entrenamiento.
- Seguidamente también establecer y configurar un modelo de *Machine Learning* (ML), analizar las métricas de entrenamiento para conseguir los mejores resultados.
- Finalmente, definir los hiperparámetros que logran un mejor ajuste de los datos en el modelo durante el entrenamiento.

La elaboración del presente informe reporta de manera organizada y consecutiva el desarrollo de este estudio. Los dos primeros capítulos a modo de introducción son para familiarizarse con el tema principal, mientras que el primer capítulo muestra conceptos teóricos acerca de las variables de clima, el capítulo 2 señala conceptos de ML, además ofrece

un resumen de los primeros modelos de pronóstico de lluvias. El capítulo 3 es importante para comprender el lenguaje de programación y lo poco que pueda entenderse de la “caja negra” de una red neuronal, además es donde se construye el modelo que propone esta tesis. Los capítulos 4 y 5 remarcan la fase experimental y resultados, indicando los inconvenientes y soluciones durante el entrenamiento del modelo. Finalmente, el capítulo de conclusiones expresa recomendaciones y sugerencias para un trabajo a futuro

Capítulo 1
Lluvias en la región Piura y pronóstico actual

1.1 Tiempo meteorológico

El tiempo meteorológico, también denominado tiempo atmosférico, se denomina al estado que presenta la atmósfera en un determinado momento. La atmósfera es propensa a sufrir una serie de cambios y fenómenos que tienen lugar en un periodo específico y en un espacio determinado. Dentro de estos eventos participan los denominados parámetros atmosféricos; temperatura, presión atmosférica, humedad, entre otros (Ucha, 2009).

Conocer cómo se manifiesta el tiempo meteorológico es muy importante en el desarrollo de muchas actividades en la vida cotidiana. Para salir de casa o de viaje, es necesario ver el pronóstico del tiempo, si va a llover se debe usar un abrigo impermeable además de usar un paraguas, por ejemplo. En casos más extremos se pueden evitar accidentes (Pérez & Gardey, 2014).

El pronóstico del tiempo es una de las actividades más importantes que realizan las agencias o institutos de meteorología. A través de la historia, ha habido intentos de reconocidos científicos e investigadores de controlar el tiempo y existe evidencia de que actividades agrícolas e industriales han ocasionado muchos cambios en los patrones atmosféricos.

Cabe mencionar que el tiempo meteorológico puede pronosticarse luego de obtener una gran cantidad de datos. Sensores especiales, y muy bien calibrados, además de un análisis estadístico de los diferentes datos y su correlación son necesarios. Desafortunadamente, el sistema es caótico; cambios tan pequeños a una parte del sistema pueden crecer para tener efectos grandes en todo el sistema (Tiempo Atmosférico, 2016).

1.2 Principales variables meteorológicas

1.2.1 Presión atmosférica

Hoy en día el instrumento más preciso para medir esta variable es el barómetro aneroide, a comparación de su predecesor los barómetros tubulares llenos de mercurio, estos miden variabilidad y deformaciones sobre casillas de metal normalmente fabricadas de berilio y cobre. Las unidades más usadas para medir la presión son el milibar (mbar),

hectopascal(hPa) y el milímetro de mercurio (mmHg). Como ya se conoce, la presión a nivel del mar para una atmósfera estándar es de 101.25 mbar = 1013.25 hPa = 760 mmHg.

La presión en inversamente proporcional a la altitud, cuanto más alto subamos en la atmósfera la presión disminuye. Mientras que la presión es mayor en la superficie terrestre, dado que soporta el peso de columnas de aire por encima de él. Las cartas sinópticas pueden ayudar entender mejor este parámetro. Las líneas en espiral en los mapas de tiempo muy parecidas a huellas dactilares son conocidas como isóbaras, líneas que conectan zonas de igual presión de aire y generalmente son trazadas cada 4 hPa. Por convención se tiene que las áreas de baja presión denotan un buen tiempo, mientras que las de alta presión representan un mal tiempo, pues aparece un aire ascendente e inestable. En el hemisferio sur los vientos circulan en sentido horario en las zonas de baja presión, mientras que en las zonas de alta presión circulan en sentido antihorario (Para el hemisferio norte es al revés). Las isóbaras significan también un indicador de la intensidad de viento; mientras más juntas estén las isóbaras, mayor será la intensidad de los vientos (Gnoza & Barberena, 2018, pp. 22-23).

1.2.2 Viento

El viento es el desplazamiento del aire en una dirección y velocidad determinada. El dispositivo que mide la velocidad del viento se llama anemómetro y consiste en un pequeño dispositivo con molinetes de aspas, cuyas terminaciones son cucharas, que giran por la fuerza del viento y posibilita medir la cantidad de vueltas que da en un determinado intervalo de tiempo (Gnoza & Barberena, 2018, pp. 25-27).

Las unidades que comúnmente se utilizan son:

- Nudos (kn en ISO o kt por el inglés *knot*) donde 1 kt equivale a 1.852 km/h
- Kilómetros por hora (km/h) donde 1 Km/h equivale a 0.27778 m/s
- Metros por segundo (m/s)

Para determinar la dirección del viento se utilizan las ya conocidas veletas de viento, muy comúnmente usadas desde hace muchos años atrás. Las veletas indican la procedencia geográfica del viento. Se habla en tal caso de viento norte, sur, noroeste, suroeste, etc. de la dirección de donde proviene éste.

En el análisis de dirección del viento además se deben distinguir las fuerzas que influyen en el movimiento horizontal del aire.

1.2.2.1 Gradiente de presión. Cuando a una misma altitud se encuentran zonas de alta y baja presión, entonces el aire desciende de la región de alta presión a la de baja presión. Si la diferencia de presiones entre ambas zonas es considerablemente grande, mayor será el movimiento del aire en este desplazamiento.

1.2.2.2 Coriolis. Esta fuerza se opone a la fuerza de gradiente de presión. Cuando la tierra rota sobre su propio eje, se origina una reorientación inercial en los vientos hacia la izquierda en el hemisferio sur y en el hemisferio norte hacia la derecha. Este fenómeno físico provoca que el viento tienda a ser paralelo en la zona de las isóbaras. Fue descubierta en 1935 por Gaspard-Gustave de Coriolis, ingeniero y matemático francés.

1.2.2.3 Centrífuga. Generalmente el comportamiento de una fuerza centrífuga hace que en isóbaras curvas actúen fuerzas radiales hacia el exterior. Son de magnitud pequeña y no tienen incidencia considerable. Es diferente en vientos fuertes, para trayectorias curvas actúa de forma acelerada o desacelerada siguiendo dicha curvatura.

1.2.2.4 Fricción. Esta fuerza actúa aproximadamente hasta los 1000 msnm y es producida por rozadura entre el aire y la superficie de la tierra. Por encima de los 1000 metros de altitud su efecto es considerado insignificante, fluyendo el viento en forma paralela a las isóbaras.

1.2.3 Humedad relativa.

La humedad relativa equivale al porcentaje de saturación de un determinado volumen de aire a una determinada temperatura. Entonces la humedad relativa indica la cantidad de vapor de agua que es transportada por el aire. Cuanto mayor es la temperatura de una masa de aire, entonces mayor será la humedad relativa.

Porcentualmente se mide desde 0% a 100%, y además es un parámetro que está directamente vinculado a la formación de nubes y precipitaciones.

Actualmente se usan higrómetros especiales para medir la humedad de un ambiente. El higrómetro usa el valor de la temperatura de condensación, conocido como punto de rocío, o también cambios en la capacitancia para medir diferencias de humedad. Se suelen usar además otros sensores como los psicómetros, que miden la diferencia de temperatura entre un termómetro con bulbo seco y un termómetro con bulbo húmedo (Gnoza & Barberena, 2018, pp. 29-30).

1.2.4 Temperatura

En meteorología se hace referencia al grado de calor específico del aire en un determinado espacio y momento determinado, así como a su evolución en un tiempo y lugar en diferentes zonas climáticas. Es una de las magnitudes más utilizadas para describir el estado de la atmósfera. La temperatura además es una magnitud relacionada con la rapidez del movimiento de las partículas que componen la materia, cuanta mayor es la perturbación presente, mayor será la temperatura.

Para su medición se usa el termómetro, calibrados en diferentes escalas, y según el Sistema Internacional de Unidades. La unidad de temperatura es el kelvin (K), cuya escala es

llamada escala Kelvin o absoluta. Se suelen usar también las escala centígrada o Celsius y la escala Fahrenheit.

1.2.4.1 Temperatura máxima. La mayor temperatura que el aire alcanza para un cierto periodo (anual, mensual, diario). Generalmente las temperaturas máximas de cada día son alcanzadas en las primeras horas de la tarde. Por otro lado, en periodos mensuales las máximas temperaturas ocurren en enero o febrero para el hemisferio sur.

1.2.4.2 Temperatura mínima. Es la menor temperatura que se alcanza en un determinado lugar y periodo. Las temperaturas mínimas suelen ocurrir durante el amanecer, mientras que en los meses de julio y agosto se presentan las mínimas temperaturas mensuales, también en el hemisferio sur.

1.2.4.3 Temperatura media. Son los promedios estadísticos que se obtienen a partir de las temperaturas mínima y máxima en un lapso de tiempo y lugar establecido. Usualmente y en conjunto con los datos de promedio de las precipitaciones del mes facilitan construir un gráfico de clima, denominado climograma (Gnoza & Barberena, 2018).

1.2.5 Radiación solar

La atmósfera es casi transparente a la radiación solar, pero la superficie terrestre y otros cuerpos situados sobre ella sí la absorben. La energía transferida desde el sol hacia la tierra se conoce como radiación o energía radiante. Ésta viaja a través del espacio en forma de ondas que llevan asociada una determinada energía (La Radiación Solar, 2018).

El instrumento comúnmente usado para medir la radiación solar es el piranómetro. Con el instrumento es posible medir la radiación semiesférica directa y difusa la cual es medida sobre una superficie horizontal en un ángulo de 180 grados, luego de calentarse dos sectores pintados alternativamente de negro y blanco en un disco plano de proporción pequeña (IDEAM, s.f.)

1.2.6 Precipitación

Precipitación es cualquier forma de meteoro constituido por agua que cae desde la atmósfera hacia la superficie terrestre; puede llamarse llovizna, lluvia, nieve o granizo según el estado en que cae al piso.

Minúsculas gotitas en suspensión forman las nubes y debido a la existencia de corrientes de aire ascendentes, empiezan a crecer a expensas de otras gotitas que se encuentran en su caída. Sobre cada gotita actúan principalmente dos fuerzas: la debida al arrastre que la corriente ascendente ejerce sobre ella, y el peso de la propia gota.

Las gotas alcanzan mayor tamaño cuanto más tiempo pasen dentro de la nube ascendiendo y descendiendo y cuanto mayor sea el contenido de agua líquida. El tamaño de gota con que el hidrometeoro llega a la superficie terrestre sirve para identificar el tipo de

precipitación líquida: llovizna (gotas pequeñas que caen uniformemente), chubasco (gotas de mayor tamaño y que caen de forma violenta e intensa), etc.

El tipo de precipitación depende de la forma de la nube y la procedencia de ésta. Las formas más habituales de precipitación son las de tipo frontal, la de tipo orográfico y la de tipo convectivo o tormentoso.

La cantidad de precipitación en un punto de la superficie es llamado monto pluviométrico o pluviosidad. Y se utiliza un instrumento denominado pluviómetro que normalmente está acoplado o conectado a una estación meteorológica. Los pluviómetros acumulan y miden las precipitaciones caídas en el sitio, que son generalmente expresadas en milímetros de altura o mediante el peso que queda atrapado en el recipiente (Gnoza & Barberena, 2018).

1.3 Clima de la región Piura

En general en la costa peruana el clima es subtropical y marítimo, con una gran uniformidad y suaves diferencias de temperatura. Este clima es en gran medida influenciado por la corriente fría peruana o de Humboldt; las corrientes que enfrían el agua del océano provocando una menor cantidad de vapor yendo hacia la atmósfera.

Los vientos alisios que provienen desde el atlántico pasan a través de la cordillera andina, que elevan a las pocas nubes propensas a precipitarse. Finalmente se dirigen hacia el oeste descendiendo mar adentro. Por ello hay poca precipitación en las costas del litoral peruano.

Las brisas que siguen el principio de la densidad de los gases, hacen que el aire cambie de lugar dependiendo de la temperatura que posea en un determinado lugar. Así el mar es más frío que la tierra durante el día, pero más cálido en las noches. En la tierra sucede lo contrario.

Debido a la cercanía con la línea ecuatorial, la costa piurana tiene un clima cálido durante todo un año. Geográficamente presenta una región yunga con clima tropical y un clima de sabana tropical en la región al nivel del mar. La temperatura promedio es 26 °C y puede alcanzar 40 °C como máxima temperatura y alrededor de unos 15 °C de temperatura mínima.

1.4 Precipitaciones en la región Piura

Un día mojado en Piura es tener por lo menos 1 milímetro de líquido o precipitación. La temporada lluviosa dura aproximadamente 2.5 meses, desde el 25 de enero al 9 de abril. Por otro lado, la temporada más seca dura 9.5 meses, desde el 9 de abril al 25 de enero.

Dentro de los días mojados, para distinguir entre los que son solo lluvia, solo nieve o una combinación de ambas. En base a lo mencionado anteriormente y el clima piurano, el tipo más común de precipitación durante todo el año es solo lluvia.

Según el artículo publicado por *Weather Spark* la temporada de lluvia en la región Piura dura 3.5 meses, desde el 30 de diciembre hasta el 14 de abril, con intervalos de por lo menos 13 milímetros. Donde la mayoría de la lluvia cae durante 31 días móviles centrados alrededor del día 5 de marzo, y con una acumulación total promedio de 38 milímetros (El Clima y El Tiempo Promedio En Todo El Año En Piura, s.f.).

1.5 Primeros pronósticos

Aunque actualmente se usan modelos de pronóstico del tiempo en un corto alcance, incluso se tiene a la mano la visualización del tiempo real para planificar actividades futuras. El pronóstico del tiempo no es tan reciente como puede parecer. Los primeros registros apuntan a las primeras culturas en formarse, los egipcios por ejemplo realizaban sus pronósticos en base al movimiento de las estrellas, o los babilónicos que se fijaban en el aspecto o color de la luna. Aristóteles por su parte fue uno de los primeros en desarrollar materia de estudio en base a la meteorología, materializando sus ideas en el libro que llamó *"Meteorologica"*.

Cabe mencionar además que las primeras civilizaciones usaban el comportamiento de plantas y animales para apoyar a sus ya pronósticos planteados. Sin más decir, destacaron en los primeros estudios las personas que necesitaban guiarse del tiempo para desarrollar sus actividades cotidianas, en ellas podemos mencionar a marineros y pastores, a quienes aluden los relatos orales que fueron pasando de generación en generación (Palazzesi, 2018).

A mediados del siglo XVII, la invención de nuevos instrumentos para medir parámetros atmosféricos, significó un gran avance para estudio científico y comportamiento de la atmósfera. Sin embargo, todas las evoluciones para un pronóstico inmediato del tiempo prontamente empezaron a mostrar limitaciones. La atmósfera desafiaba la mayoría de los pronósticos luego de las 24 horas de su producción y en su mayoría antes. Se contaba con pobres resultados en materia de pronóstico y meteorología, esto debido a la escasez de datos y observaciones de parámetros meteorológicos.

Hasta entonces, la meteorología se había desarrollado como una ciencia principalmente empírica basada en la observación de los fenómenos y básicamente de la observación de las variables observadas en la superficie de la tierra. Por supuesto que se sabía que la evolución de la atmósfera dependía de leyes físicas, pero nunca se profundizó en su interpretación. Todo se redujo a la observación y experiencia (Palomares Calderón de la Barca, 2015).

Fue sino hasta 1922, luego de muchos experimentos para lograr estudiar el comportamiento y la dinámica de la atmósfera en sus niveles altos, pues hasta ese entonces los estudios realizados se restringían a la superficie de la tierra. El matemático británico Lewis Fray Richardson propuso la primera iniciativa para el desarrollo de un modelo matemático, bajo la denominación de *Métodos Numéricos*. Desafortunadamente aún no era posible en

aquella época realizar en tiempo útil los inmensos cálculos necesarios, por ello, Richardson escribió; “quizá algún día los avances de la computación la hagan más rápida que el tiempo atmosférico y con un coste menor que los gastos implicados” (Palomares, 2015). Finalmente, luego de muchas reinvenciones de modelos posteriores, la predicción meteorológica mediante métodos numéricos comenzó a funcionar a partir de 1950 en un proyecto conjunto de la Fuerza Aérea de los Estados Unidos y la Oficina Meteorológica.

Actualmente la predicción del tiempo usa modelos numéricos de simulación de la atmósfera muy sofisticados, y ejecutados con potentes ordenadores gracias a datos atmosféricos provenientes de muchas fuentes, como satélites o radares meteorológicos. Datos que se introducen en los modelos con una serie de técnicas muy desarrolladas (Palomares, 2015).

1.6 Análisis del tiempo meteorológico actual

Aún con avanzados modelos matemáticos y poder computacional, el trabajo que se requiere para realizar el análisis de tiempo es enorme. Básicamente las tareas van desde medir, recolectar, transmitir y procesar e interpretar millones de datos en todo el planeta Tierra. Sin embargo, y dado que la atmósfera cambia continuamente, se debe buscar un pronóstico en el menor tiempo posible y muy preciso.

Por ejemplo, Inzunza (2019) resume lo siguiente en cuanto a la intervención de los especialistas:

> En la realización de un pronóstico tradicional intervienen muchos actores. El observador meteorológico, es la persona encargada de realizar las observaciones de las variables meteorológicas de la estación en las horas sinópticas, estas observaciones las puede hacer cualquier persona a la cual se le ha enseñado a leer los instrumentos, basta con que sepa leer y escribir. El observador transmite, vía teléfono o similar, los datos al centro de análisis, donde el ploteador, que debe ser un técnico en meteorología, traspasa los datos a un mapa sinóptico. Luego interviene el analista, que debe ser meteorólogo, es el encargado de dibujar el mapa sinóptico, trazando las isóbaras, identificando centros de alturas y bajas presiones, y dibujando los frentes cuanto existen, y destacando los fenómenos de tiempo significativo, como precipitaciones, por ejemplo. Posteriormente actúa el pronosticador, un meteorólogo que es el que interpreta la carta sinóptica y realiza un pronóstico preliminar. Finalmente, en los centros de análisis, una o dos veces al día se reúne un grupo de expertos para hacer la discusión del pronóstico, que es especialmente importante cuando la atmósfera presenta situaciones conflictivas, las cuales deben ser totalmente aclaradas antes de hacer el pronóstico definitivo, que se emite a todos los usuarios que lo requieren y al público en general (p. 306).

La metodología moderna comprende diferentes enfoques debido a la naturaleza dinámica y compleja de la atmósfera. Métodos estadísticos, predicción numérica del tiempo y diferentes técnicas de pronóstico de corto y largo plazo son utilizados para ayudar a obtener resultados mejores y más precisos. Los servicios meteorológicos nacionales son los encargados de generar cartas sinópticas de gran escala con métodos numéricos. Estas cartas se envían a los centros regionales y locales de pronóstico, que a su vez utilizan técnicas de pronóstico tradicional como se menciona anteriormente, finalmente realizan un pronóstico a nivel local.

1.7 Problemática actual

El clima siempre va a impactar directa o indirectamente sobre las actividades humanas. Enfoquémonos en uno de los parámetros meteorológicos, en este caso la precipitación, que es materia de estudio en esta tesis. En los campos de la economía y agrario, por ejemplo, abundantes lluvias pueden mejorar las cosechas y con ello mejor producción para una determinada población. Sin embargo, tal exceso de lluvias puede ocasionar desastres naturales también, huaicos o desbordamientos de ríos que ponen en alerta a la misma u otra comunidad.

Un lugar turístico muy concurrido por personas extranjeras puede dejar de ser visitado o aislarse debido a un desastre natural ocasionado por la variabilidad climática, ello consecutivamente perjudica a los ingresos en la comunidad. Todos y estos detalles se tienen muy en cuenta por las personas para planificar un viaje, alguna tarea, la ejecución de un proyecto, etc.

Las aerolíneas actualmente usan bastante información de los servicios meteorológicos en cada región. Si hay una tormenta cerca o mucha nubosidad es preferible postergar un viaje a concurrir en una gran tragedia. Los servicios de transporte también pueden paralizarse con la abundancia de lluvias, con ello se generan daños consecutivos económico, pues el transporte de mercadería de una ciudad a otra es inviable en estos casos.

Es por ello que es necesario tener en cuenta hoy en día el tiempo y la variabilidad climática. Sobre todo, y teniendo en cuenta, además, que la evolución demográfica de las comunidades y diferentes culturas en un ambiente, un gran avance del área industrial y tecnológico han suscitado efectos negativos para la variabilidad climática.

Por otra parte, también está la gran batalla entre los pronosticadores del tiempo y las comunidades que reciben la información. Normalmente cuando falla un pronóstico se suele adjudicar de mal pronosticador a la persona o entidad encargada. Si bien es cierto los pronósticos modernos son más confiables a lo que antiguamente se lograba dar por hecho un pronóstico, a pesar de ello es necesaria una inmensa cantidad de datos para compensar la compleja y dinámica variabilidad de los parámetros del tiempo meteorológico.

Como cita Lucas Parera (2017, párrafo 6):

> “Nosotros emitimos un informe, pero después en la televisión pueden llegar a decir algo distinto, y a veces hasta lo contrario que decimos en el Servicio Meteorológico Nacional. Me acuerdo de un caso significativo este año. Los pronósticos que emitían desde la televisión hablaban de lluvias torrenciales para el fin de semana. Y lo decían desde el miércoles de esa semana, Pero nosotros tuvimos en cuenta el viento zonal registrado en otras provincias, que terminó disipando la tormenta a último minuto. Esto pasa todo el tiempo, dice Fernández.”

Y continúa Fernández acerca de la variabilidad y dinámica atmosférica:

> “La naturaleza es bastante caótica, en un momento, en un rato, se pueden modificar las condiciones climáticas y esto afecta la efectividad de los pronósticos, dijo Fernández a LA NACIÓN. Es decir, la meteorología estudia un sistema complejo y dinámico, en el que pequeñas variaciones pueden producir cambios sensibles en la totalidad del sistema.”

Así como también hay pronósticos fallidos que se supieron sobrellevar y terminaron favorablemente para todos. También los hay en los que lastimosamente ha terminado perjudicando a más de uno, y no es de asustarse. Los pronosticadores saben muy bien que pueden fallar al emitir un resultado. A pesar de ello, tal parece que, son muy duramente criticados.

En 1986 en un reporte televisado, el meteorólogo británico Michael Fish minimizó sobre el clima los temores que había en Reino Unido ante la aproximación de un huracán y desafortunadamente se equivocó:

> “Más temprano, aparentemente una mujer llamó a la BBC y dijo haber oído que había un huracán en camino. Bueno, sí está usted viendo, no se preocupe, no hay ninguno”, dijo Fish el 15 de octubre de aquel año. La tormenta que horas después golpeó el sur de Inglaterra fue la peor en 300 años, causando pérdidas humanas y económicas (Wall, 2014, párrafo 2).

Adentrándonos más en los métodos numéricos, siempre se ha buscado el mejor método y/o modelo para lograr predecir un largo periodo de tiempo en el menor tiempo posible. Como se mencionó anteriormente desde las primeras iniciativas era muy complicado tratar con todos los parámetros atmosféricos y tomaba un extenso tiempo de preparación de resultados. Los trabajos siguientes han buscado en sus modelos equilibrar desde entonces un tiempo de predicción inmediata, menor complejidad en el modelo y la vez robustez de este.

Los primeros modelos de predicción del clima relacionadas a visión por ordenador, se basaban en extrapolación de datos de flujo óptico, usando mapas de ecos de radar, que localizaban movimiento de nubes convectivas, y que luego utilizaban para predecir futuros mapas ecos de radar utilizando advección semilangraniana (Shi et al., 2017). Dado que estos métodos se realizaban sin ninguna supervisión desde el punto de vista del ML, despertó el

interés de la comunidad de Inteligencia Artificial, pues se cree que de esta manera se pueden aprovechar todos los datos que cuentan los mapas de ecos de radar.

Como menciona (Shi et al., 2015b), el problema tratado desde el punto de vista del Aprendizaje Automático (ML), se puede ver como un problema de pronóstico espacio-temporal que puede mejorar los resultados de pronóstico a corto alcance. Usando capas convolucionales de red neuronal apiladas y que permitan reconocer diferentes características espaciales de cada mapa de eco de Radar. Posteriormente usar redes neuronales que permita guardar variables temporales de los datos en cada paso de tiempo, para finalmente aprender todo este conjunto de parámetros y lograr dar un resultado certero. Básicamente este modelo se centra en trabajar con imágenes, reconocerlas, estudiarlas y aprender de ellas.

Sin embargo, Shi también aclara un gran inconveniente para el modelo y es nuevamente el peso del modelo, debido a la alta dimensionalidad de todas las secuencias espacio temporales, y especialmente cuando se realizan predicciones de varios pasos. Todo puede compensarse si el modelo puede captar bien las relaciones espacio temporales, aún a expensas de una lenta predicción. Básicamente la solución al problema está en trabajar a un nivel profundo en el modelo. “Además, construir un modelo de predicción efectivo para los datos de eco del radar es aún más difícil debido a la naturaleza caótica de la Atmósfera” (Shi et al., 2015, p. 2)

1.8 Eventos de El Niño Costero en Piura, 2017

El Perú es uno de los países más afectados por los fenómenos climáticos costeros con el calentamiento del pacífico oriental ecuatorial, ello debido a su ubicación geográfica. Así mismo, le siguen a esto catástrofes naturales debido a las lluvias, como inundaciones o extremas sequías.

Fue en febrero del 2017, cuando se declararon en estado de emergencia las regiones de Piura, Tumbes y Lambayeque. Se estaban gestando tormentas de gran magnitud, que llegaron a un punto tal que se produjeron muchos desastres, primero en la sierra de estas tres regiones. Exceso de precipitaciones ocasionaron huaycos que truncaron el tránsito terrestre, las vías de acceso se obstruyeron, y en las partes bajas simplemente tramos de pista desaparecían.

El duro golpe lo sufrió la región de Piura el 27 de marzo del 2017, desde el 25 de marzo ya muchos centros poblados y caseríos alejados de la ciudad, ya había sufrido desbordes de ríos y cuantiosas pérdidas, muchas familias habían perdido totalmente sus viviendas. Posteriormente el gran incremento de quebradas y riachuelos terminaron en un gran cauce de agua para el río Piura, el cual se desbordó por la mañana inundando todas las urbanizaciones que encontraba en su paso.

“La capacidad máxima del río Piura, por el puente Cáceres, era de 2800 metros cúbicos por segundo (m^3/s), pero ayer el caudal llegó a un máximo de 3468 m^3/s. La consecuencia la

sufrieron todos los piuranos: desbordes en varias calles de la ciudad, y en pueblos del Bajo Piura" (El Niño Costero, El Hecho Que Marcó La Década En Piura, 2019)

> Ese día, Piura parecía una laguna extensa, pues el desborde del río se desplazó por varias áreas. Una vez hubo bajado el nivel del río, se pudo aprecia todos los desastres ocasionados. Casas derrumbadas, lodo por todas partes, calles deterioradas, algunas cerradas, y el corte de suministro de agua y luz en varios sectores de la ciudad. A ello también posteriormente se le sumó muchos malos olores en las calles, por la obstrucción de los buzones de desagüe, por otro lado, el aumento de zancudo y con ello enfermedades perjudiciales para la salud (El Impacto Del Fenómeno Del Niño Costero En La Ciudad de Piura y Su Vida Urbana, 2017).

Desafortunadamente y pese a los pronósticos y previsiones de la variabilidad climática; la elevada temperatura del mar y fuertes precipitaciones. La ciudad no estaba preparada, en un sentido de concientización no mucho, se puede decir que parte del ámbito de infraestructura y desarrollo urbanista es un hecho que no lo estaba. El fenómeno del Niño costero si se pudo prevenir, tal como lo explicó el Doctor Ken Takahashi, director del Servicio Nacional de Meteorología e Hidrología en ese entonces:

> "El niño costero ya había aparecido en 1925 en las costas del norte peruano. Así lo comprueba la gráfica compartida vía Twitter por el doctor Ken Takahashi, en donde se registra de manera comparativa el nivel de calentamiento en el mar del Puerto de Chicama en 1925 y en el presente año" (López Tarabochia, 2017, párrafo 9).

UNIVERSITAS STUDIORUM PIURENSIS

Capítulo 2
Machine Learning y su aplicación en el pronóstico de lluvias

2.1 Machine Learning (Aprendizaje Automático)

Machine Learning (ML) es un campo de las ciencias de la computación, que se ocupa del desarrollo de la Inteligencia Artificial, responsable del "aprendizaje", en el que una máquina está programada para pensar y aprender dado un conjunto de datos. Formalizando, se encarga de representar la estructura y generalizar el comportamiento de los datos. Es una tecnología que permite automatizar varias operaciones con el fin de reducir la intervención humana.

Algo muy importante menciona Márquez (Márquez, 2018), "aprender", no se trata de memorizar y recopilar datos. Se trata de crear un modelo a partir de la información proporcionada para sacar conclusiones y con ello solucionar el/los problemas(s) que se trata(n). Cada vez que los modelos son expuestos a nuevos datos, pueden adaptarse de forma independiente. Aprenden de cálculos anteriores (ya conocidos) para tomar decisiones y producir resultados verídicos o repetibles.

2.2 Modelo en Machine Learning (ML)

Un modelo, a grandes rasgos, se usa para describir un sistema (físico, social, natural, industrial, etc.) usando conceptos y lenguaje matemático. Un modelo permite determinar un resultado final en función de algunos datos de entrada.

Gracias a algoritmos se puede construir modelos que descubran conexiones entre sí para aprender. De esta manera, durante el entrenamiento del sistema (donde se detectan patrones en los datos de entrada), se genera un modelo que servirá para hacer predicciones, selecciones, reconocimiento, etc. Puede entenderse según Gonzáles (Conceptos Básicos de Machine Learning, párrafo 19), "un modelo es como un filtro en el que entran datos nuevos y cuya salida es la clasificación de ese dato según los patrones que se han detectado en el entrenamiento".

2.2.1 Requisitos para generar nuevos sistemas de ML

- Recursos de preparación de datos.
- Algoritmos, básicos y avanzados.

- Automatización y procesos iterativos.
- Escalabilidad.
  - Modelado en conjunto.
  - Importa más la calidad de los datos que la cantidad. Es mejor tener datos fiables y útiles que tener millones de datos de los que no se puede extraer valor.

2.2.2 Tipos de Machine Learning

Un modelo de aprendizaje automático utiliza la experiencia y la evidencia en forma de datos para tratar de comprender por sí mismo los patrones o comportamientos existentes. En resumen, a partir de muchos ejemplos de una situación en particular, es posible construir un modelo que pueda interpretar y generalizar el comportamiento observado, y hacer predicciones sobre situaciones completamente nuevas (Redacción APD, 2019).

2.2.2.1 Aprendizaje Supervisado. En este tipo de aprendizaje el modelo es entrenado con cierta cantidad de datos definidos o descritos mediante etiquetas, como una entrada donde se conoce el resultado deseado (Redacción APD, 2019, párrafo 16). El proceso de entrenamiento se resume a continuación; el algoritmo de aprendizaje recibe un conjunto de entradas y sus resultados correctos correspondientes, seguidamente el algoritmo aprende comparando sus resultados reales con resultados correctos para detectar errores. Luego se va modificando, adaptando el modelo, y utilizando métodos como regresión, clasificación, predicción y aumento de gradiente, el aprendizaje supervisado predice los valores de etiquetas con datos nuevos luego de utilizar patrones aprendidos (SAS Institute, 2018).

Un ejemplo común de utilidad del aprendizaje supervisado es en la detección de transacciones, evaluar la probabilidad de que una transacción con tarjetas sea fraudulenta o que un usuario de una empresa de seguros tiene la probabilidad de generar un reclamo.

2.2.2.2 Aprendizaje No Supervisado. En este tipo los datos que se usan para entrenar no contienen etiquetas que los definan. No se conoce "respuesta correcta" de cada entrada. El algoritmo debe descubrir el resultado final del conjunto de datos. El modelo se concentra en analizar los datos y encontrar alguna estructura o patrón en su interior. Durante la identificación de grupos de clientes con características similares, el aprendizaje no supervisado funciona bien con datos de transacciones que luego pueden procesarse de manera similar en campañas de marketing.

Las técnicas más populares de este tipo de aprendizaje comprenden *k-means clustering*, *mapping* del vecino más cercano, mapas con organización automática y descomposición de valores singulares (SAS Institute, 2018).

2.2.2.3 Aprendizaje Con Refuerzo. Durante el entrenamiento el algoritmo aprende a través de prueba y error las acciones que producen las mejores recompensas. Este tipo de aprendizaje identifica tres componentes principales: el agente (el que aprende o toma las decisiones), el entorno (todo con lo que interactúa el agente) y la acción (lo que el agente puede hacer). El objetivo es que el agente elija acciones que maximicen la recompensa esperada en un momento dado. Los agentes logran su objetivo más rápido si utilizan una buena estrategia. Entonces, el propósito del aprendizaje con refuerzo es aprender la mejor estrategia o política (SAS Institute, 2018).

Así mismo, Márquez (2018) clasifica cada uno de los tipos de ML mencionados, según el tipo de datos que manejan.

- **Continuos:** Información cuantitativa/numérica. Por ejemplo, predecir el precio de una casa.
- **Discretos:** Información cualitativa. Por ejemplo, predecir de qué equipo deportivo es hincha una persona.

2.2.3 Áreas de aplicación

2.2.3.1 Servicios financieros. Actualmente entidades relacionadas a la industria financiera como los bancos, por ejemplo, utilizan la tecnología de la inteligencia artificial para dos fines principales: identificar *insights* importantes en los datos y evitar posibles fraudes en cuanto a seguridad se refiere, identificar clientes con perfiles de alto riesgo o bien utilizar sistemas de cíber vigilancia para detectar señales de advertencia de estafas. Por otro lado, los *insights* también permiten reconocer oportunidades para invertir o bien ayudar a los inversionistas a saber cuándo es propicio vender o comprar (SAS Institute, 2018).

2.2.3.2 Atención a la salud. Gracias a la creciente aparición de dispositivos y sensores en prendas de vestir, se pueden obtener datos con el fin de analizar la salud y estado de un paciente en tiempo real. También, la tecnología del ML ofrece ayuda a los expertos en la salud en la identificación de enfermedades, o tendencias hacia alguna infección. Ello además puede llevar a diagnósticos y mejores tratamientos (SAS Institute, 2018).

2.2.3.3 Industria del combustible. Desde analizar minerales en una zona determinada, y de un tipo de suelo, hasta la predicción de fallos en sensores especiales de refinería. En esta industria el aumento de la utilización del ML es vasto. Incluso se puede optimizar la distribución de petróleo para hacerla más eficiente (SAS Institute, 2018).

2.2.3.4 Marketing y ventas. Normalmente muchas páginas web recomiendan artículos que podrían gustarte en base a compras anteriores, ello es debido a que usa ML, capturan datos y analizan su historial de compras para sugerirte productos relacionados (SAS Institute, 2018).

2.2.3.5 Transporte. Los aspectos de modelado y análisis de datos del ML son herramientas importantes para las empresas de mensajería, transporte público y otras organizaciones de transporte. El análisis de datos para identificar patrones y tendencias es fundamental para la industria del transporte, que a menudo desea mejorar la eficiencia de las rutas para ahorrar tiempo y costos o predecir posibles problemas potenciales de los mecanismos de un vehículo para mejorar la rentabilidad o prevenir accidentes (SAS Institute, 2018).

2.3 Deep Learning (Aprendizaje Profundo)

Es una rama de la Inteligencia Artificial, que trata de asemejar el comportamiento de las máquinas al de las redes neuronales de los seres vivos, en cuanto al aprendizaje se refiere. Mientras que los algoritmos tradicionales el proceso de aprendizaje es supervisado, y además se realiza un proceso de extracción de características. La ventaja del aprendizaje profundo se basa en que el algoritmo por sí mismo construye el conjunto de características, sin supervisión, además de ser más rápido y sobre todo más preciso (Burns, s.f.).

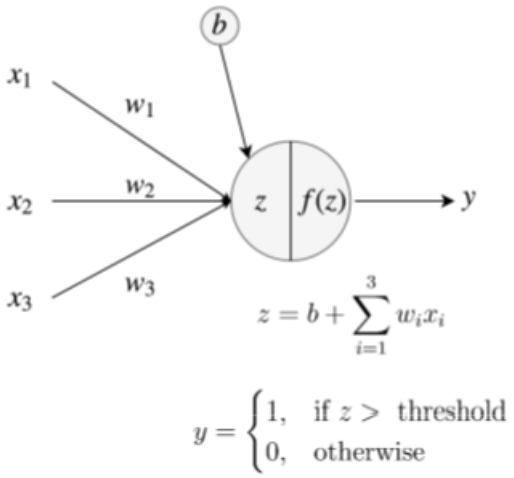
Los algoritmos que conforman un modelo de aprendizaje profundo comúnmente se encuentran divididos en diferentes capas neuronales y compuestas por pesos (un valor numérico). El modelo básico se estructura principalmente en tres capas según indica el artículo web (SmartPanel, 2018).

- Capa de entrada: Está compuesto por las neuronas que asimilan los datos de entrada.
- Capa oculta: Es la parte principal para el entrenamiento, en este punto se realiza el procesamiento de información y hacen los cálculos intermedios de los pesos. Mayor cantidad de neuronas en esta capa, implica que los cálculos sean más complejos y por tanto el tiempo de ejecución sea elevado.
- Capa salida: Es el último eslabón en la cadena del modelo de aprendizaje, y es importante también porque toma la decisión o realiza alguna conclusión aportando datos en la salida.

2.3.1 Redes Neuronales

Rosenblatt (1958) indica que “para comprender el modelo computacional de las redes neuronales artificiales, uno debe comenzar desde su componente básico, conocido como el perceptrón” (citado en Bahuleyan, 2018, p. 8). La forma de un perceptrón está basada en las neuronas del cerebro, y constituye un modelo computacional simple que toma una o más entradas y proporciona un único valor como salida. Con esta salida y un umbral predefinido, el perceptrón realiza una clasificación binaria. Si el valor de salida está por encima del umbral asignado, la entrada se atribuye a la clase 1, de lo contrario, se atribuye a la clase 0 (**Figura 1**).

Figura 1. Modelo de perceptrón simple

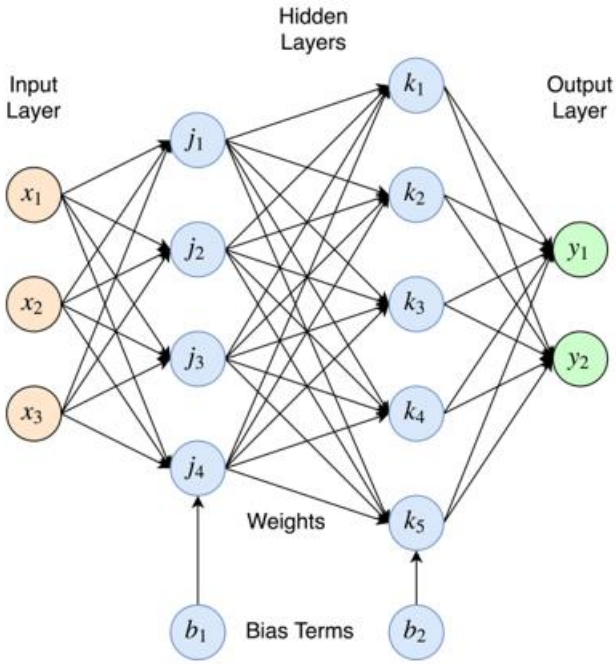


Nota. **Tomada de *"Natural Language Generation with Neuronal Variational Models"* Bahuleyan (2018).**

Sin embargo, las redes neuronales modernas ya no utilizan el perceptrón simple. Las redes han evolucionado en unidades computacionales conocidas como neuronas (o nodos), que reemplazan la función de activación binaria simple con funciones no lineales como sigmoide, tanh o unidad lineal rectificada (ReLU) (Bahuleyan, 2018).

Tal es el caso del modelo de perceptrón multicapa (MLP) desarrollado para dar densidad y complejidad a la estructura del perceptrón. En esta nueva arquitectura cada nodo individual en una capa específica está conectado a cada nodo en la siguiente capa (**Figura 2**), por lo que se convierte en una red neuronal completamente conectada (Seldon, 2022).

Figura 2. Modelo de perceptrón completamente conectado



Nota. **Tomada de *"Natural Language Generation with Neuronal Variational Models"* Bahuleyan (2018).**

Las entradas de un MLP se multiplican con pesos (valores numéricos) y se alimentan a la función de activación y luego con la característica de *back-propagation* se modifican para reducir la función de costo. Estos pesos son valores que el modelo ha aprendido durante un entrenamiento y se autoajustan según la diferencia entre las entradas y las salidas previstas (Great Learning Team, 2022).

2.3.2 Red Neuronal Recurrente (RNN)

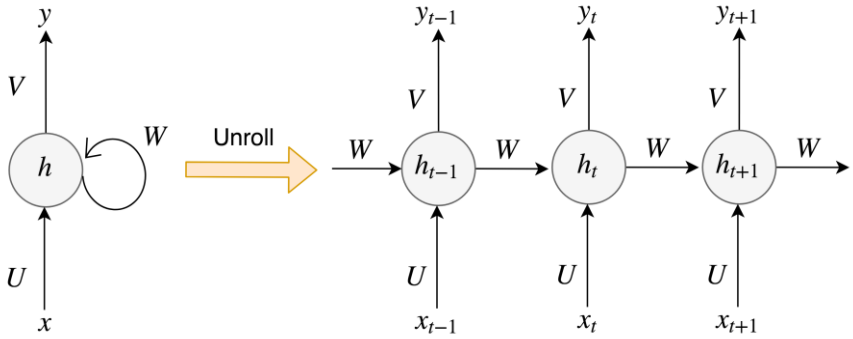
Bahuleyan (2018) añade acerca de las redes neuronales vistas en el apartado anterior: "Una de las deficiencias de las redes neuronales *feed-forward* es que supone que todos los datos de entrada son independientes entre sí. Como resultado, no logra capturar la noción de orden secuencial que está presente en algunos tipos de datos".

Entonces es necesario un modelo que retenga información necesaria para usarla en un futuro, de esta manera se pueda comprender mejor el contexto de una entrada y refinar la predicción en la salida. Es el caso de las redes neuronales recurrentes, y (Seldon, 2022) señala que es muy usada en sistemas de texto predictivo; se puede usar la memoria de una palabra anterior en una cadena de palabras y predecir mejor el resultado de la siguiente palabra.

La entrada a un RNN se proporciona de manera secuencial, y la red hace uso de las entradas en los pasos de tiempo anteriores para tomar una decisión en el paso de tiempo actual. Una red neuronal recurrente se puede representar como una red con bucles (Figura 3), a través de la cual se transfiere información entre los pasos de tiempo de la red. "Al

desenrollar la red, nos damos cuenta de que la información en cada paso de tiempo pasa a través de múltiples copias de la misma red" (Olah, 2015, citado en Bahuleyan, 2018).

Figura 3. Modelo de Red neuronal recurrente



Nota. **Tomada de *"Natural Language Generation with Neuronal Variational Models"* Bahuleyan (2018).**

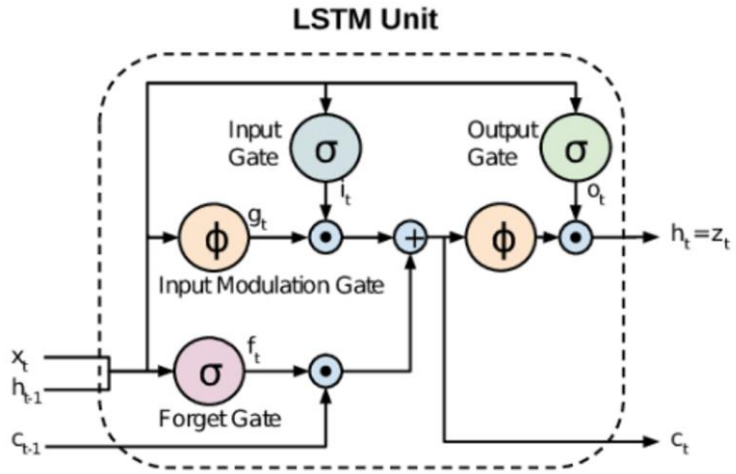
2.3.3 Long Short-Term Memory (LSTM)

En la ejecución real, los modelos RNN fallan en la poca capacidad que tienen de capturar dependencias a largo plazo. En otras palabras, cuando la longitud de la secuencia de entrada se hace grande, los RNN no pueden recordar las dependencias entre las entradas ya que están muy separadas en la secuencia. La razón de esto se atribuye al problema de desvanecimiento de gradiente (Pascanu et al., 2013, citado en Bahuleyan, 2018). Esto ocurre debido a un valor numérico por debajo o por encima, es decir, cuando la multiplicación de los términos derivados durante la retropropagación se vuelve extremadamente pequeña o muy grande. Esto se puede resolver al truncar los gradientes cuando su valor absoluto cruza un umbral previamente especificado (Bahuleyan, 2018).

Para evitar el problema de los gradientes que desaparecen, se propusieron extensiones a la arquitectura RNN, unidades de memoria a corto plazo o *Long Short Term Memory* (LSTM). Este nuevo modelo incluye una "celda de memoria" que puede mantener la información durante largos periodos de tiempo (Great Learning Team, 2022).

La unidad LSTM utiliza un conjunto de puertas para controlar el paso de información (Figura 4). Las puertas, representadas por activaciones sigmoideas (cuyos valores de salida entre 0 y 1) esencialmente deciden la cantidad de información entrante que debe fluir (Bahuleyan, 2018). La puerta de entrada decide acerca de cuánta información de la última muestra se guardará en la memoria, la puerta de salida regula la cantidad de información que pasa a la siguiente capa y la puerta de olvido controlan la velocidad de corte de la memoria almacenada (Great Learning Team, 2022).

Figura 4. Esquema de una red de tipo LSTM



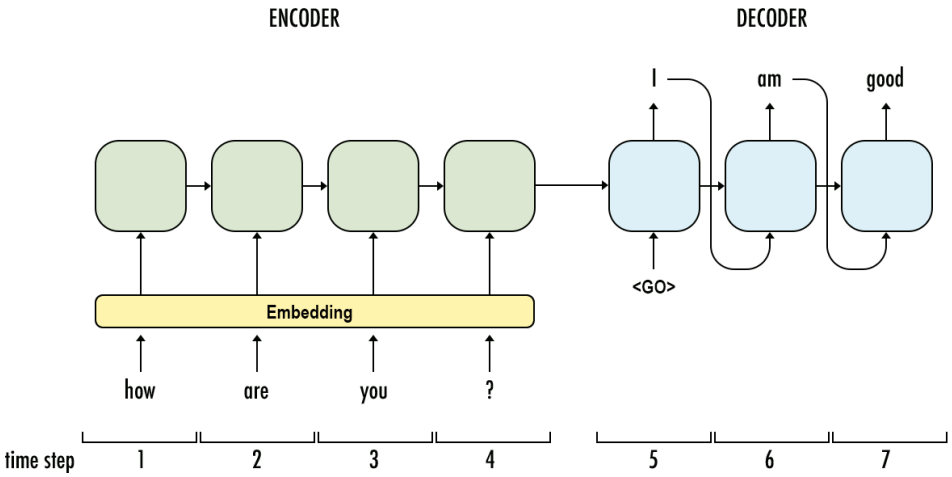
Nota. **Tomada de *"What is the main difference between RNN and LSTM | NLP | RNN vs LSTM?"* Tripathi (2021).**

2.3.4 Modelo Sequence-to-Sequence (Seq2Seq)

Para procesamiento del lenguaje natural, las tareas de secuencia a secuencia generalmente se refieren a aquellas en las que el modelo toma como entrada una secuencia y genera otra secuencia como salida (en lugar de un solo valor). En el caso de documentos u oraciones, los tokens individuales que forman la secuencia son palabras (Figura 5). Por el contrario, cuando las palabras mismas se tratan como secuencias, sus caracteres se convierten en los tokens individuales (Bahuleyan, 2018).

Generalmente en este tipo de modelo consta de dos redes neuronales recurrentes. En la versión más básica, la secuencia de entrada se alimenta token a token al primer RNN (codificador), que calcula una representación vectorial para toda la secuencia. Esta representación vectorial se convierte en el punto de partida para el segundo RNN (decodificador), que genera la secuencia de salida, nuevamente de manera simbólica (Bahuleyan, 2018).

Figura 5. Esquema de un modelo de red secuencial



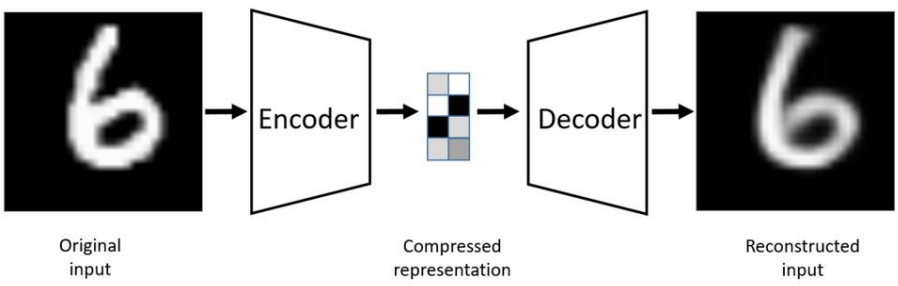
Nota. **Tomada de *"Sequence to sequence model: Introduction and concepts"* Chablani (2017).**

2.3.5 Autoencoder

El autoencoder es una técnica de aprendizaje no supervisada en la que proporcionamos una entrada (como una imagen o texto) a un modelo, se aprende una representación intermedia y luego se intenta reconstruir la entrada original a partir de esta representación (Figura 6). El modelo suele ser una red neuronal artificial, y la representación intermedia suele tener una dimensionalidad menor que la entrada original (Hinton y Salakhutdinov, 2006, citado en Bahuleyan, 2018). Finalmente, lo que hace a un autoencoder es aprender una codificación eficiente que almacene solo la información necesaria requerida para la reconstrucción.

Los autoencoders se han aplicado con éxito para resolver problemas como la súper resolución de imagen y la mejora del habla. Dado que las representaciones de dimensión inferior generadas por los autoencoders son útiles para identificar patrones relacionados con los datos originales, también se han aplicado a la detección de anomalías (Bahuleyan, 2018).

Figura 6. Autoencoder esquema general



Nota: Tomada de *"Autoencoders"* Bank, Koenigstein y Giryes (2020).

2.3.6 Mecanismo de Atención

El mecanismo de atención busca asemejarse más al comportamiento del cerebro humano. Este modelo propone la concentración selectiva, enfocándose en algunas cosas relevantes, mientras ignora a otra en redes neuronales profundas.

Para concretar, el mecanismo de atención es como un guía durante el proceso de entrenamiento, informando al modelo sobre las partes de la entradas o características en las que debe centrarse, para realizar la tarea en cuestión (Bahuleyan, 2018).

2.3.7 Variational autoencoder (VAE)

La inferencia variacional hace uso de los autocodificadores variacionales, y resultaron de una solución a la inferencia bayesiana dado que este modelo implicaba la resolución de integrales intratables y que se volvían engorrosas para el cálculo de distribuciones de probabilidad.

Los *Variational Autoencoder* (VAE) o autocodificador variacional son modelos que unen los conceptos de redes neuronales y distribución de probabilidad. El principal objetivo de un VAE es el de construir modelos generativos con la capacidad de producir datos sintéticos que sigan y aprendan los patrones de los grandes conjuntos de datos de los que se alimentan (Sancho, 2020).

Los VAE fueron introducidos por primera vez por Kignma y Welling (2013) en el dominio de una imagen con el objetivo de aprender representaciones latentes para imágenes de dígitos escritos a mano. Lo que hace que los VAE sean potentes es que estas representaciones latentes aprendidas pertenecen (aproximadamente) a una distribución predeterminada, como por ejemplo gaussiana que ya tiene una media y varianza conocidas (Bahuleyan, 2018).

2.3.8 Generative Adversarial Networks (GAN)

Según Brownlee (2019). las Redes Adversarias Generativas, o GAN para abreviar pertenecen al aprendizaje no supervisado en el *Machine Learning*. Las GAN tienen la tarea de descubrir automáticamente las regularidades o patrones de los datos de entrada para que el modelo pueda usarse para generar salidas finales o ejemplos obtenidos del conjunto de datos original.

Las GAN son una forma inteligente de entrenar modelos generativos al enmarcar el problema como un problema de aprendizaje supervisado con dos submodelos. Las dos redes neuronales que componen una GAN se denominan generador y discriminador. El generador es una red neuronal convolucional y el discriminador es una red neuronal deconvolucional. El objetivo del generador es fabricar salidas de forma artificial que puedan fácilmente confundirse con datos reales. Finalmente, el objetivo del discriminador es clasificar los ejemplos como reales o falsos. Los dos modelos se entrenan juntos, el modelo generador

intentará siempre fabricar salidas de mejor. Hasta que el modelo discriminador sea engañado, es donde el generador está fabricando ejemplos plausibles por decirlo de alguna manera.

Finalmente, Brownlee (Brownlee, 2019) asegura que las GAN son un campo emocionante que cumple la promesa de los modelos generativos debido a su capacidad para generar ejemplos realistas en una variedad de dominios o problemas. Últimamente se ha visto que pueden generar fotos de objetos, escenas y personas que no existen, pero tampoco se puede decir que son falsas.

2.3.9 CycleGAN

La CycleGAN es una implementación de una red GAN, y apareció para dar solución a los enfoques generativos anteriores, dado que se requerían conjuntos de datos exhaustivos y costosos computacionalmente.

Sobre el objetivo de una CycleGAN, Janetzky (2021) nos dice que intenta traducir una imagen de un dominio X a un dominio Y sin utilizar en el entrenamiento conjuntos de imágenes emparejadas debido a que el modelo es propenso al efecto denominado un colapso de modo, donde todas las muestras de entrada se asignan a una única muestra de salida, y con ello se estropea el entrenamiento.

Para superar las deficiencias de las condiciones anteriores, la red CycleGAN introduce una segunda transformación de las imágenes generadas al dominio de origen, generando el *cycle consistency loss* al comparar la imagen original con la imagen reconstruida a partir de la imagen traducida al otro dominio. Este proceso permite que la red, teniendo en cuenta el *loss* durante el entrenamiento, evite perder la identidad de la imagen original aunque se transfiera a otro dominio.

2.4 Parámetros y condiciones de una red neuronal

Anteriormente se ha mencionado las diferentes arquitecturas y modelos de redes neuronales más conocidos, y que siguen un orden y son escogidos de acuerdo al tipo de datos con los que se trabajan y la solución que se quiere obtener. Ahora bien, ya conocemos la parte externa de un modelo de red neuronal, en esta parte accederemos un poco a los parámetros internos de una red neuronal, y qué servirán más adelante en el capítulo 3 al momento de montar la arquitectura de un autocodificador variacional.

En primer lugar, evaluaremos los problemas a los que se enfrenta una red neuronal, y posteriormente las soluciones que se han encontrado para resolverlos.

2.4.1 Problema de descenso de gradiente

Los problemas de gradiente son el mayor obstáculo para el entrenamiento de una red neuronal. Dado que se busca llegar a una alta precisión con los resultados en la salida de la red, debe ocurrir que la pérdida que se calcule debe llegar a sus valores más mínimo. Sin

embargo, se debe ser muy cuidadoso de no caer en un punto de silla o en un mínimo local, ya que paralizarían el entrenamiento.

Kaul (2020, párrafo 3) en un artículo web propone que el proceso de aprendizaje se divide en propagación recursiva hacia adelante y hacia atrás.

2.4.1.1 Propagación hacia adelante. Las entradas son dirigidas a través de las neuronas de las capas ocultas, y además se inicializan algunos pesos aleatoriamente junto con sesgos. Una transformación lineal dada como:

$$Z = (entrada * peso) + sesgo$$

Para añadir complejidad a la red y asegurar mejores resultados, se añade una función no lineal (función de transferencia), comúnmente conocida como función de activación. La salida de la función de activación (z), es decir, la suma ponderada de las entradas, se adjunta a la siguiente función de activación (Kaul, 2020).

$$A = f(z)$$

2.4.1.2 Propagación hacia atrás. Las predicciones de un modelo (y') suelen diferir de los datos reales (y), por lo que se debe aplicar una función de costo (J), que finalmente se conocerá como pérdida en el *Machine Learning*. Esta función nos va a decir cuánta desviación hay entre lo real y lo predicho. La función básica es la suma media de la función de pérdida al cuadrado (Kaul, 2020, párrafo 9).

$$J = Loss = 1/N\left(E\ (y - y')\right)$$

Desafortunadamente la pérdida durante el entrenamiento se propaga hacia atrás hasta las capas iniciales mientras se van actualizando los pesos de cada neurona en cada capa. A este proceso de propagar el error se denomina retropropagación. Por ello se buscará minimizar los pesos de las neuronas que contribuyen más en la función de costo.

Ajustando los pesos se consigue reducir a función de costo. El método de prueba y error es un buen inicio, pero demanda tiempo y es engorroso. Es por ello, que existen los optimizadores en el *machine learning*. Para comprender cómo las ponderaciones afectan las entradas, se calcula la derivada de la función de costo, la tasa de cambio de la pérdida. El resultado final que se obtiene son los pesos optimizados y que siguen la siguiente ecuación.

2.4.1.3 Problema de la curva patológica. Bien se sabe que durante la ejecución de un modelo una variable que determina cuan bien va nuestro modelo es la pérdida. Ahora bien, las redes neuronales proporcionan una inmensidad de variables que hacen que las dimensiones sean infinitas, algo que no es visible. Pero se puede cuantificar, de este modo hay una relación entre los infinitos pesos de la red neuronal y la pérdida. Se deben optimizar los pesos para conseguir que la pérdida alcance un mínimo global.

Sin embargo, en ese trayecto el gradiente sigue una dirección hacia una zona conocida como curva patológica, debido a que en esta región es donde el descenso empieza rebotar haciendo que el desplazamiento hacia los mínimos sea lento. Una taza de aprendizaje alta podría pensarse para evitar las oscilaciones por la curva patológica. Pero además esto podría hacer que el entrenamiento se vuelva muy lento, y demande de mucho tiempo (Kathuria, 2018).

Ahora que se ha explicado los mayores problemas a los que se encuentra las redes neuronales, procedemos a explicar los parámetros que dirigen por así decirlo el desplazamiento del descenso de gradiente a lo largo de los valles hacia los mínimos locales. Elegir entre uno y otro parámetro es crucial para el aprendizaje. Más adelante se detallará.

2.4.2 Funciones de activación

Nuevamente hacemos la acotación referida desde la página web de Jason Brownlee (2021), para conceptualizar acerca de una función de activación: “Una función de activación en una red neuronal define como la suma ponderada de la entrada se transforma en una salida de un nodo o nodos en una capa de la red. También se le conoce como función de transferencia.”

Sobre la importancia de una función de activación, señala: “La elección de la función de activación tiene un gran impacto en la capacidad y el rendimiento de la red neuronal, y se puede utilizar diferentes funciones de activación en diferentes partes del modelo”.

Menciona además que una función de activación trabaja de la siguiente manera: “Técnicamente, la función de activación se usa dentro o después del procesamiento interno de cada nodo en la red, aunque las redes están diseñadas para usar la misma función de activación para todos los nodos en una capa” (Brownlee, 2021a).

Finalmente cabe señalar, que Brownlee propone una clasificación en base al tipo de capa; capa oculta o capa de salida. Aunque aquí se apoyó en la metodología no aplica para el tipo de datos que se tienen y la arquitectura.

2.4.2.1 Linear. También conocida como la función identidad o “ninguna activación”. Esta función de activación no produce cambios en las operaciones de los pesos al interior de la red, de tal manera que solo retorna el valor directamente.

Sharma (2017) señala que la activación lineal no ayuda con la complejidad o los diversos parámetros de los datos habituales que se alimentan las redes neuronales. La red no podría entrenar bien y capturar los patrones más complejos de los datos. Mayormente el uso de esta función es en tareas simples donde se desea mucho la interpretabilidad.

2.4.2.2 Sigmoide. La función Sigmoide también es conocida como logística, y el rango de valores que transforma en su salida se encuentra entre 0 y 1, entonces se interpreta normalmente como una probabilidad (Sharma, 2017).

Algo a destacar es que la función sigmoide es una función no lineal, y a diferencia de la función linear tratada anteriormente, en este caso es diferenciable lo que implica que existe una pendiente de la curva, y se tienen varias neuronas con función sigmoidea, las salidas que se obtienen también serán no lineales (Gupta, 2020).

2.4.2.3 Tangente hiperbólica. Basada en la función de activación Sigmoidea, propiamente basado en un escalamiento. Esta función comúnmente conocida como Tanh, tiene un rango de valores de salida entre -1 y 1. Tiene la ventaja de ser simétrica respecto al origen. Sin embargo, dado que es una variante mejorada de la función Sigmoide, aún padece del problema de desaparición del gradiente. Esto produce que las actualizaciones de los pesos de la red sean lo bastante pequeños o insuficientes para que la red tenga un buen aprendizaje (Freire & Silva, 2019).

2.4.2.4 Relu. Correspondiente a las siglas de *Rectified Linear Unit* o unidad lineal de rectificación. Es otra función de activación no lineal. A diferencia de las funciones Sigmoide y Tanh, Relu no activa todas las neuronas al mismo tiempo y ello eleva los beneficios de esta función de activación. El coste computacional es menor ya que se desarrollan operaciones matemáticas más simples (Seelam, 2021).

Desafortunadamente, Relu, para valores negativos el gradiente se vuelve cero y con ello disminuye la capacidad del modelo para adaptarse a entrenarse a partir de los datos correctamente. Dado que las neuronas en esos casos se desactivan y dejan de responder. A este problema se le conoce como *dying Relu*.

2.4.2.5 Leaky Relu. Es una variante mejorada de la función Relu, y está para prevenir el problema de neuronas muertas visto anteriormente en el apartado de Relu. Leaky Relu propone una ligera pendiente en el cuadrante izquierdo para que valores negativos no sean 0, de tal manera que se define una componente x pequeña. De esta manera ya no habría neuronas muertas (Gupta, 2020).

2.4.2.6 Softmax. Una función de activación diseñada para clases múltiples y normalmente se aplican en las capas de salida en una arquitectura de red neuronal. Comúnmente se describe como una combinación de múltiples funciones sigmoides. Y así como la función sigmoide desarrolla una clasificación binaria en probabilidades, de igual manera la función *softmax* extiende su rango a calcular probabilidades de múltiples clases (Seelam, 2021).

La función *softmax* devuelve la probabilidad de un punto de datos perteneciente a cada clase individual (Gupta, 2020).

2.4.3 Optimizadores

Los optimizadores son otro hiperparámetro importante dentro de una arquitectura de *machine learning*. Influyen directamente y relativo al entrenamiento y actualizaciones de los pesos, así mismo relacionado a la taza de aprendizaje en la duración del entrenamiento, y las épocas necesarias para llegar a un buen resultado.

Dentro del conjunto de optimizadores usados en Tensorflow, se tienen los optimizadores más conocidos y por ende más usados, los optimizadores adaptativos. Entre ellos SGD, ADAM, RMSprop y Adadelta. Dentro de ellos los que han perfeccionado mejor el entrenamiento para los datos de estudio, corresponden a RMSprop y ADAM.

2.4.3.1 Gradiente estocástico de descenso (SGD). O comúnmente conocido como *stochastic descend gradient*. En este caso el algoritmo de optimización realiza una actualización de parámetros para cada ejemplo de entrenamiento. Los parámetros del modelo se modifican después del cálculo de la pérdida en cada entrenamiento.

Como refiere (Maithani, 2021), SGD realiza cálculos redundantes para conjuntos de datos más grandes, ya que vuelve a calcular los gradientes para el mismo ejemplo antes de cada actualización de parámetros. Desafortunadamente esto hace que los parámetros tengan una gran variación y fluctuaciones en las funciones de pérdida a diferentes intensidades.

2.4.3.2 Adagrad. Adagrad adapta la taza de aprendizaje específica para cada característica individual. Esto significa que algunos de las ponderaciones de un conjunto de datos tendrán diferentes tazas de aprendizaje que otros conjuntos de datos. Maithani (Maithani, 2021) señala que Adagrad siempre funciona mejor en un conjunto de datos dispersos donde faltan muchas entradas.

La gran desventaja de Adagrad es de que es muy costoso computacionalmente ya que requiere calcular la derivada de segundo orden, dado que constituye ese tipo de algoritmo. Por otro lado, al ir variando y disminuyendo la taza de aprendizaje, el entrenamiento es muy lento.

2.4.3.3 AdaDelta. Según refiere (Doshi, 2019), es una extensión de Adagrad y se inclina hacia la eliminación del problema de la taza de aprendizaje en decadencia. En lugar de acumular todos los gradientes previamente cuadrados, Adadelta limita la ventana de gradientes pasados acumulados a un tamaño fijo. Sin embargo, tiene el problema de un elevado costo computacional.

2.4.3.4 RMSprop. La explicación se detalla muy bien de acuerdo a lo mencionado en (Sanghvirajit, 2021): "RMSprop se encarga del problema de desvanecimiento de gradiente o explotación del gradiente en redes neuronales profundas. Mediante el uso de un promedio de gradientes cuadrados puede normalizar el gradiente, de manera que se equilibran el tamaño del paso (*momentum*). Disminuye el paso para gradientes grandes y se evitan las explosiones o se aumenta el paso para gradientes pequeños para evitar desaparecer".

(Brownlee, 2021) menciona que el RMSprop está diseñado para acelerar el proceso de optimización, por ejemplo, disminuir el número de evaluaciones de funciones necesarias para alcanzar los óptimos, o mejorar la capacidad del algoritmo de optimización.

2.4.3.5 Adam. Adam constituye un algoritmo basado en el gradiente de primer orden de funciones objetivas estocásticas, basado en la estimación adaptativa de momentos de orden inferior. Adam, actualmente es uno de los mejores optimizadores porque combina características de otros optimizadores que suman a su funcionalidad (Sanghvirajit, 2021).

En lugar de ajustar las tasas de aprendizaje de los parámetros en función del promedio del primer momento tal como RMSprop, Adam utiliza también el promedio de los segundos momentos de los gradientes (la varianza no centrada). De este modo, Adam usa gradientes pasados para calcular los actuales (Brownlee, 2017a).

Específicamente ADAM calcula el gradiente al cuadrado y un promedio móvil exponencial del gradiente, y cuenta con los parámetros beta1 y beta2 que controlan la tasa de caída de estos promedios móviles.

2.4.4 Función de costo

Acerca de la función de costo o pérdida, se menciona en (Función de Pérdida En Machine Learning, 2019): Las funciones de pérdida es una forma de medir que tan bien un algoritmo modela los datos dados. Si las predicciones difiere demasiado de los resultados reales, la función de pérdida indica un valor demasiado grande, al contrario de cuando los resultados son cercanos a la realidad.

Tensorflow-Keras, tiene alrededor de veinte clases para evaluar la pérdida en un modelo de red neuronal. Y cada tipo de pérdida se enfoca en diferentes características de los datos de entrenamiento y según los resultados que se quieren obtener del modelo. Aquí solo se hará mención a algunas funciones de pérdida, ya que estaban más relacionadas al trabajo de predicción y el tipo de datos que se trabajaba.

2.4.4.1 Pérdidas de Regresión. En este punto la función de costo me va a dar información de cuán bien o mal el modelo lograr predecir un valor, para este caso un valor continuo.

- ***MSE:*** Se obtiene a partir el promedio de la diferencia al cuadrado entre los valores predichos y valores realmente observados. Solo se preocupa del tamaño promedio del error, independientemente de su dirección. Independiente del signo de los valores reales y previstos, el resultado siempre es positivo y el valor ideal es 0.

Sin embargo, como se menciona en (Función de Pérdida En Machine Learning, 2019) debido a la cuadratura, los pronósticos que se alejan demasiado del valor real se penalizan severamente en comparación con los pronósticos menos desviados.

- ***MAE:*** El error absoluto medio mide el promedio de la suma de las diferencias absolutas entre el valor predicho y el valor realmente observado. Al igual que el MSE, también mide la magnitud del error independientemente de su orientación (*Función de Pérdida En Machine Learning. SitioBigData.Com*, 2019).

En comparativa a MSE, esta función de pérdida necesita herramientas más sofisticadas, como la programación lineal para calcular los gradientes. Además, MAE es más resistente al sesgo porque no utiliza cuadrados.

2.4.4.2 Pérdidas de Clasificación. Por este lado, la función de costo estará en relación de los resultados obtenidos tratando el modelo con valores categóricos finitos. El valor obtenido de la función de costo me va a indicar cuánto bien o mal clasificó correctamente el modelo.

- ***Binary Crossentropy:*** Es la función de pérdida predeterminada que se utiliza para problemas de clasificación binaria. Se calcula como la entropía cruzada promedio en todo el conjunto de entrenamiento. Es decir, la entropía cruzada binaria calculará una puntuación que resume la diferencia promedio entre las distribuciones de probabilidad real y predicha para finalmente predecir la clase. Un valor mínimo de entropía cruzada perfecto es cero (*Función de Pérdida de Entropía Cruzada. ICHI.PRO*, 2020).

- ***Categorical Crossentropy:*** Es la función de pérdida que se usa para problemas de clasificación de clases múltiples. De igual manera que en la anterior función de pérdida, calcula la pérdida de entropía cruzada entre las clases verdaderas y las clases predichas (Brownlee, 2020a).

2.5 Primeras técnicas de predicción de desplazamiento de celdas de lluvia usando lenguaje por computador

Las primeras técnicas para predicción del desplazamiento de celdas de lluvia, usando radares convencionales, se desarrollaron a comienzos de la década de 1960 por medio de extrapolación de los ecos de radar (Wilson et al, 1998, citado en Novo-Cuervo, 2008). Con esta

metodología se correlacionaban dos ecos consecutivos y se determina un vector de movimiento en la dirección de máxima correlación. Sin embargo, la precisión de estos primeros pronósticos se reducía rápidamente alcanzada la primera media hora.

Resultó que para los años 70 se empezaron a desarrollar los primeros algoritmos para rastrear celdas individuales en dos dimensiones utilizando técnicas de reconocimiento de patrones. Estas nuevas técnicas se agruparon bajo el nombre de "técnicas de seguimiento del centroide" ya que basaron el seguimiento y pronóstico en las áreas, el centroide pesado u otros parámetros que se cree que representan tormentas convectivas individuales (Novo-Cuervo, 2008).

Ya para finales de la década de los años 1970, 1980 y 1990 se estaba atendiendo la formación de tormentas como entidades en tres dimensiones (Crane 1979 y Rosenfeld 1987), además de un posible empleo de la tendencia de tamaño e intensidades de eco (Tsonis & Austin 1981) en la realización de un pronóstico inmediato. También se incluyeron en el estudio las divisiones y fusiones naturales entre tormentas (Rosenfeld 1987, Witt & Johnson 1993, Dixon & Wiener 1993). Mientras aumentaba la cantidad de información obtenidas de radares permitió contar con poderosas herramientas para el estudio científico de las tormentas (Novo-Cuervo, 2008).

Johnson publicó en 1998 una versión de algoritmo para identificar y monitorear desplazamiento de celdas de lluvia usando la red de radares Doppler de estados unidos (WSR-88D). El modelo contempló 7 umbrales para la reflectividad en orden decreciente durante el reconocimiento de tormentas, sobre todo en grupos de tormentas extremadamente compactos donde varias unidades anidadas se pueden concentrar en un entorno de alta reflectividad. Finalmente, para el seguimiento se utilizó la técnica de rastreo de centroide volumétrico pesado y tomando los vectores de movimiento de los barridos anteriores (Novo-Cuervo, 2008).

2.5.1 TRACE 3D

Handwerker (2002) ideó un algoritmo automático que solo usara datos de reflectividad. La identificación determina primero los volúmenes barridos por el haz del radar con una reflectividad por encima de un cierto umbral. Luego se encuentra el valor máximo de reflectividad en cada volumen. Seguidamente, las celdas de tormenta se definen como esas regiones contiguas dentro de cada volumen que tiene una reflectividad mayor que la diferencia entre la máxima y un valor fijo. El seguimiento utiliza las velocidades de las celdas procedentes de escaneos anteriores para extrapolar la posición de una celda individual al último escaneo, y luego se buscan sucesores de esta celda en las proximidades de la posición esperada.

2.5.2 TREC

Con este procedimiento se buscaba la máxima correlación entre dos observaciones de radar consecutivas (pudiendo estas ser PPI o CAPPI con la misma inclinación o altura, respectivamente) para ecos de lluvia en un área pequeña. Luego se trazaban vectores de movimiento entre los dos centros en las regiones de mayor correlación. Esto daba como resultado un campo de velocidades bidimensional similar al obtenido con radares Doppler, el mismo que se extrapolaba para predecir inmediatamente las posiciones de los ecos (Novo-Cuervo, 2008).

TREC por otro lado presenta el problema de ecos o sombras no meteorológicos en el haz del radar, así como pequeños cambios en el patrón del radar, que ocasionan vectores de velocidad erróneos.

2.5.3 COTREC (COntinuity of TREC)

Esta técnica permitió la cuantificación de áreas crecientes en tormentas y áreas en disipación. Para lograrlo se calculó la diferencia de reflectividad promedio entre las áreas pequeñas de cada barrido asociada con los vectores de movimiento resultantes. Las áreas positivas de esta diferencia se denominan como regiones de intensificación del eco, mientras que las áreas de valores negativos indican regiones de pérdida. Además, COTREC se ha utilizado con bastante éxito en situaciones de tormentas severas (Novo-Cuervo, 2008).

2.5.4 SWIRLS (Short-Range Warning of Intense Rainstorms in Localized Systems)

Este sistema introducido por el observatorio de Hong Kong se basa en la extrapolación de ecos de radar usando la técnica TREC vista anteriormente, y que incluye otras mejoras como la verificación de consistencia y filtrado mediante el análisis imparcial de Cressman. Al seleccionar apropiadamente el tamaño de la matriz de pixeles a correlacionar, los vectores de desplazamiento resultantes pueden usarse satisfactoriamente para monitorear y extrapolar el movimiento de casi cualquier tormenta de intensidad moderada. Para una resolución de 480x480 pixeles en el campo de reflectividad obtenido del radar, utilizaron arreglos de 19x19 pixeles con una separación de 5 pixeles. Finalmente, debe aclararse que a partir de este modelo se da el comienzo a los modelos usando el aprendizaje automático (Novo-Cuervo, 2008).

2.6 Primeros modelos de predicción de desplazamiento de celdas de lluvia usando ML

A continuación, se presentan los modelos que se han utilizado en la última técnica de predicción vista anteriormente SWIRLS, se expone lo que se denomina una evolución del modelo de predicción con Redes Neuronales.

Para entrar a los modelos primero explicamos el problema de la predicción inmediata de precipitación, que básicamente se refiere a proporcionar un pronóstico de muy corto

alcance (ejemplo 0-6 horas) de la intensidad de la lluvia en una región local basada en los mapas de eco de radar, pluviómetro y otros datos de observación.

Los modelos existentes para la predicción inmediata de la precipitación pueden clasificarse aproximadamente en dos clases; los modelos basados en NWP (*Numerical Weather Prediction*) y los modelos basados en la extrapolación de ecos de Radar. Para el enfoque NWP, hacer predicciones en la escala de tiempo de predicción inmediata requiere una simulación compleja y meticulosa de las ecuaciones físicas en el modelo de la atmósfera. Por lo tanto, los actuales sistemas de emisión de precipitación operativas a menudo adoptan los modelos basados en extrapolaciones más rápidos y más precisos (Shi et al., 2015a).

Un progreso reciente de acuerdo con el segundo modelo es el algoritmo de flujo óptico en tiempo real por métodos variacionales para ecos de radar (*ROVER- Real-time Optical flow by Variational methods for Echoes of Radar*). Sin embargo, el éxito de estos métodos basados en flujo óptico es limitado porque la etapa de estimación de flujo y la etapa de extrapolación del eco del radar están separadas y es un desafío determinar los parámetros del modelo para obtener un buen rendimiento de predicción (Shi et al., 2015a).

Estos problemas técnicos pueden solucionarse viendo el problema desde la perspectiva de *Machine Learning*. En esencia, la predicción inmediata de precipitación en tiempo real es un problema de pronóstico de secuencia de mapas de radar pasados como entrada y la secuencia de un número fijo (generalmente mayor que 1) de mapas de radar futuros como salida.

2.6.1 Conv-LSTM

Se formula la predicción inmediata de precipitación como un problema de pronóstico de secuencia espaciotemporal que se puede resolver bajo el marco general de aprendizaje secuencia a secuencia. Para que se modelen bien las relaciones espaciotemporales, el modelo ConvLSTM propone estructuras convolucionales tanto en las transiciones de entrada a estado como de estado a estado. Si se apilan varias capas ConvLSTM se forma una estructura de pronóstico de codificación, entonces se puede construir un modelo entrenable de extremo a extremo para la predicción inmediata de precipitación en tiempo real (Shi et al., 2015a).

2.6.2 TrajGRU (Trajectory Gated Recurrent Unit)

A pesar de que *Convolutional LSTM* supera los métodos basados en flujo óptico, la estructura de recurrencia convolucional en los modelos convolucionales es invariable con la ubicación, mientras que el movimiento natural y la transformación (por ejemplo, la rotación) son en general, variantes de la ubicación. El modelo *TrajGRU* puede aprender activamente la estructura de ubicación variante para conexiones recurrentes. Además, presenta un punto de referencia que incluye un conjunto de datos a gran escala del mundo real. Para el nuevo modelo, se propone el modelo de *unidad recurrente cerrada por trayectoria* (TrajGRU) que utiliza una subred para generar las estructuras de conexión de estado a estado antes de las

transiciones de estado. TrajGRU permite que el estado se agregue a lo largo de algunas trayectorias aprendidas y, por lo tanto, es más flexible. De esta manera TrajGRU apunta a aprender la estructura de correlación local para datos espaciotemporales (Shi et al., 2017).

2.6.3 PredRNN ++

PredRNN ++ es una red recurrente mejorada para el aprendizaje predictivo de video. En la búsqueda de una mayor capacidad de modelado espaciotemporal, el enfoque aumenta la profundidad de transición entre estados adyacentes al aprovechar una nueva unidad recurrente, que se llama *Causal LSTM* para reorganizar las memorias espaciales y temporales en un mecanismo en cascada. Sin embargo, todavía hay un dilema en el video de aprendizaje predictivo: los modelos cada vez más profundos en el tiempo se han diseñado para capturar variaciones complejas, al tiempo que presentan más dificultades en la propagación hacia atrás del gradiente. Para mitigar este efecto indeseable, se propone una arquitectura *Gradient Highway* que proporciona rutas alternativas más cortas para flujos de gradiente desde salidas hasta entradas de largo alcance. Esta arquitectura funciona perfectamente con los LSTM causales, lo que permite a PredRNN ++ capturar las dependencias a corto y largo plazo de forma adaptativa (Wang et al., 2018).

UNIVERSITAS STUDIORUM PIURENSIS

Capítulo 3
Desarrollo del algoritmo de predicción

En este capítulo se detallan todos los pasos que se siguieron para la realización del algoritmo de *Machine Learning*, desde la obtención de los datos, el pre-procesamiento y la realización del modelo de ML. Todo esto fue implementado en el lenguaje de programación de Python.

3.1 Radar Escaneador de lluvias PIUXX

Actualmente se ha instalado, desde mediados del año 2019, el primer radar escaneador de lluvias en el Perú. Ubicado en la región de Piura, en el campus de la Universidad de Piura. Es un radar de fabricación alemana a cargo de la empresa Leonardo y bajo el dominio de la marca SELEX Gematronik.

La instalación de este sistema fue motivada por la ocurrencia del fenómeno de El Niño Costero del 2017, descrito en el primer capítulo. Entre uno de los objetivos está el seguimiento de la formación y evolución de las precipitaciones, con alta resolución espacio-temporal, durante los eventos de lluvia. A partir de conformar una base de datos y estudios posteriores lograr pronósticos de corto plazo. Por otro lado, se sostiene que la red pluviométrica en la región Piura no es suficientes para analizar el comportamiento y la variabilidad de las precipitaciones cuando se manifiestan.

La antena parabólica del radar (**Figura 7**) se encuentra instalado a una altura de 12 m del suelo, sobre una torre metálica, y a aproximadamente 60 m.s.n.m. Cuenta con una configuración tal que el ángulo de *beam* es de 2° de para evitar los problemas de *beam blockage* a los cuales un dispositivo como un radar siempre está expuesto. La radiofrecuencia emitida por la antena es de 9.4 GHz (banda X) y logra un alcance de 100 km de radio por sobre el área de la región.

Figura 7. Instalaciones de la estación científica Ramón Mugica

El sistema cuenta con dos partes; la parte física y la parte de control. La parte Física contempla el propio radar, constituido por una antena parabólica, una unidad principal donde se encuentra el motor que hace girar la antena a una velocidad de 12 RPM y también el dispositivo que emite el pulso electromagnético, conocido como *magnetrón*. Además de los componentes electrónicos que gobiernan el circuito de control en la unidad principal.

La parte de control está a cargo de la Unidad de Interfaz, que consiste en un equipo que controla el encendido y apagado de la unidad principal, y una computadora donde se puede observar en tiempo real lo que detecta el radar. Dos softwares son los que permiten interactuar al encargado con la parte física. Un software que gobierna todo el sistema, y la visualización en tiempo real de precipitación. El otro software es para post procesamiento y una visualización más amigable dirigida por ejemplo al público.

3.1.1 Relación reflectividad-precipitación (Z-R)

La reflectividad (Z) es el dato raíz obtenido por el radar de lluvia para cada "píxel" de su resolución y es debido a la potencia reflejada por el volumen correspondiente a dicho píxel. Su intensidad depende de la cantidad y tamaño de las gotas de lluvia dentro de dicho volumen.

(Marshall et al., 1947) encontró una relación entre la precipitación y reflectividad y esto pasó a formar parte importante de muchos estudios relacionados al campo de climatología. Con esta relación se pueden calcular las tasas de precipitación partiendo de la reflectividad que se obtiene en el radar de lluvias.

J.S. Marshall y W. Palmer establecieron una relación para encontrar una distribución aproximada de tamaños de gotas de lluvia en función de la intensidad de lluvia. Las

distribuciones de tamaño de gota medidas experimentalmente han sido usadas para calcular tanto la reflectividad de radares como la intensidad de lluvia.

Se relaciona la reflectividad (Z) y precipitación (R) mediante la siguiente fórmula:

$$Z = A \, . R^b$$

Donde, Z es el factor de reflectividad del radar.

R, es la tasa de precipitación.

A, b, son factores que se determinan empíricamente.

Estos parámetros son calculados al asumir que la distribución de tamaño de las gotas es conocido dentro de un volumen de aire que ha sido medido por el radar. Entonces, los parámetros dependen del tipo de lluvia, varían de un lugar geográfico a otro, etc. (Godoy Mendía, 2019)

En (Godoy Mendía, 2019) también menciona que el factor de reflectividad abarca una gran cantidad de magnitudes (desde $0.001 \frac{mm^6}{m^3}$ para neblina, hasta 36000000 $\frac{mm^6}{m^3}$ para granizo del tamaño de una pelota de béisbol, generalmente se expresa en decibeles (dB) de reflectividad o dBZ. Por tanto:

$$Z = 10^{log\,(z/\,1mm^6/m^3\,)}$$

Siendo esta función logarítmica usada para transformar los valores a una escala mucho más compresible, que es la escala que se usa en el procesamiento de la información.

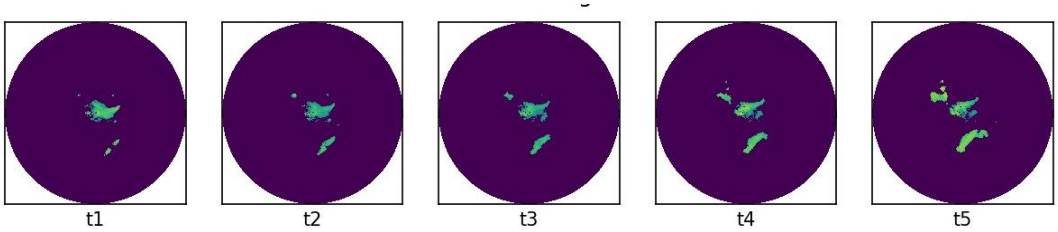
La metodología de estimación de lluvia en base a las relaciones Z-R proviene desde los primeros radares meteorológicos de polarización simple, en donde no se contaba con información adicional a la reflectividad. Referente a lo mencionado estas relaciones pueden conducir a sesgos en la estimación cuantitativa de la precipitación, además se requiere del uso de diferentes relaciones de acuerdo al tipo de lluvia. Por ejemplo, la relación Marshall-Palmer se aplica solo a la precipitación de origen estratiforme, donde las gotitas de agua que están contenidas en las nubes no tienen mucho desarrollo vertical, es decir, no son de origen convectivo.

3.1.2 Formato, contenido y forma de los archivos proporcionados

Los archivos proporcionados por el software del Radar se encuentran en formato NetCDF4, y cada uno constituye un día completo de recopilación de datos. El área cubierta es una matriz de 400x400 pixeles y se puede ver la reflectividad en la **Figura 8**. Además, cada elemento representa un área de 400 m^2. Por cada día se tiene una secuencia de 288

elementos, que desde ahora se le llamará imágenes o cuadros. Entonces el intervalo entre cada imagen es de 5 minutos. Por lo tanto, la forma de los datos para un día equivale a una matriz de 3 dimensiones (400x400x288).

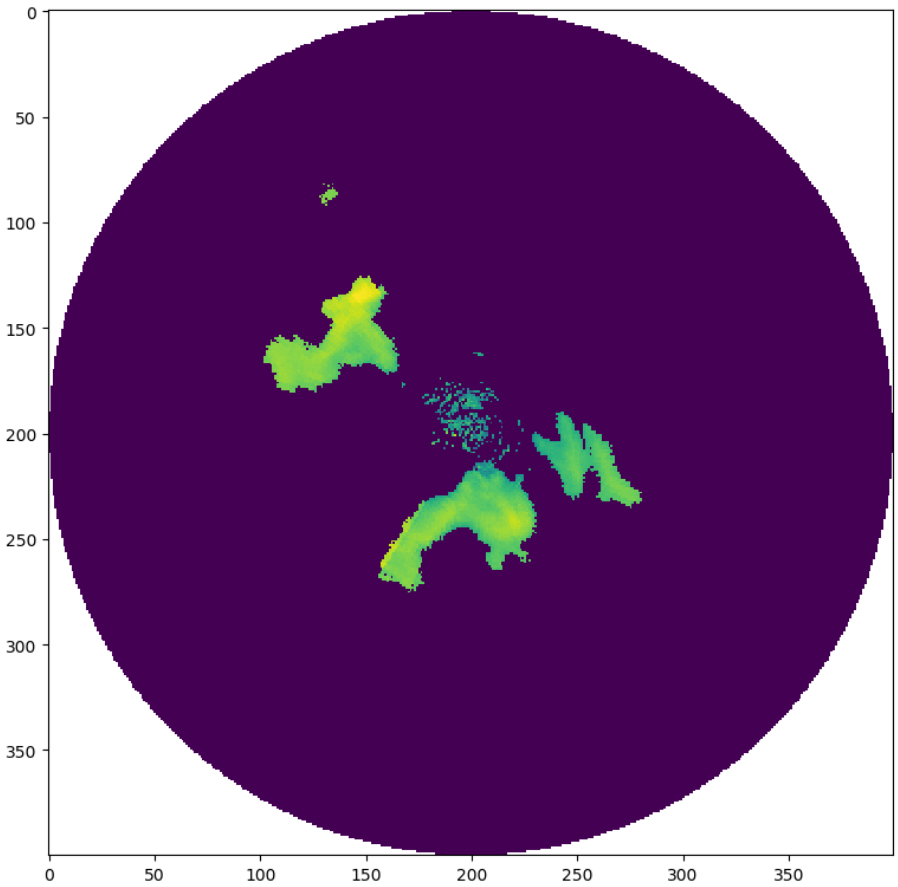
Figura 8. Secuencia de cuadros de desplazamiento de lluvia



Nota. **Obtenidos del radar de Cuenca, precipitaciones del 06/01/2017.**

Las matrices de cada dato contienen valores de reflectividad, los mismo que se miden en decibles de reflectividad, y que están en una escala de -31.5 dBZ hasta 95.9 dBZ. En la **Figura 9**, las regiones un tanto amarillas corresponden a valores positivos a diferencia de las zonas azules que corresponden a valores negativos.

Figura 9. Acercamiento de un cuadro tomado en un instante por el radar



3.1.3 Posibles inconvenientes del radar escaneador de lluvias

El radar situado sobre una torre a 10 m de altura desde el suelo expuesto al clima característico de la costa piurana. Además de vientos relativamente fuertes en las tardes de verano o las frías mañanas en invierno. Y al estar fabricado con componentes electrónicos, de alguna manera se pueden producir daños de la parte física como la parte de software en la medición de lecturas o transmisión de datos.

3.1.3.1 Hardware. Los daños por hardware son referidos al deterioro de los componentes físicos del radar y que ocasionan lecturas erróneas.

- ***Calor excesivo:*** El clima característico de la región Piura hace que se tenga un calor intenso en temporada de verano, y un calor moderado en otras épocas del año, pero ello no impide que en estas épocas haya un incesante y radiante sol, lo que ocasiona un aumento de la temperatura dentro de la unidad principal, y con ello las tarjetas electrónicas. Consecuentemente afectar las lecturas que proporciona el radar.

- ***Intervenciones en la estructura del Radar:*** Los fuertes vientos en las tardes de verano, pueden desnivelar el sistema central de la unidad principal, con ello variar el ángulo *del haz* y se produzcan lecturas erróneas o encontrarse con obstáculos, si el ángulo del haz *llega a ser 0°.*

3.1.3.2 Software. La capacidad de tratamiento de datos está limitada a los componentes que lo conforman, y cuando fallan es probable que afecte el software que realiza el procesamiento de la información (Godoy Mendía, 2019).

- ***Saturación del canal:*** En instantes donde el radar está captando una gran cantidad de información, es posible que la información llegue a sobrepasar el límite del ancho de banda del canal utilizado. En este caso el componente que procesa la información también colapsa y deja de procesar la información, en otras palabras, se pierden las lecturas que se realizaban en ese preciso instante (Godoy Mendía, 2019).

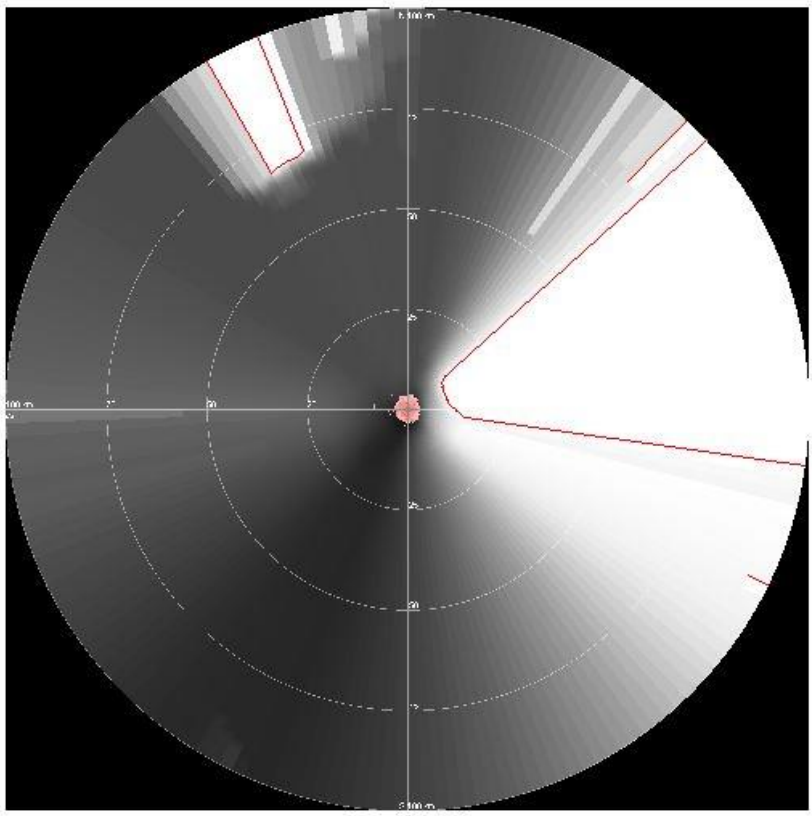
- ***Pérdida de conexión:*** Todo artefacto conectado a algún tipo de red está expuesto a este tipo de riesgo. Cuando esto ocurre con el radar, la información no puede ser enviada y por lo tanto se pierde (Godoy Mendía, 2019).

3.1.3.3 Factores físicos externos. Con factores físicos externos nos referimos a cuerpos externos a los equipos del radar pero que están dentro del alcance del pulso suministrado por el propio radar de lluvias. Estos factores influyen en las lecturas del Radar y en la mayoría de los casos son difíciles de controlar o mitigar.

Nos referimos a objetos móviles como aviones, helicópteros, animales, etc. O estáticos como montañas, edificaciones y hasta tanques elevados de agua que son algunos objetos que interfieren en la medición correcta de los datos.

Tal como menciona en su trabajo de titulación de Ecuador (Godoy Mendía, 2019) aquí en Piura también se tiene presente en menor impacto el efecto de *beam blockage* (haz bloqueado) ya que no hay presencia de alta montaña al menos hasta dónde llega el alcance del Radar. Sin embargo, en las fronteras del alcance del radar si se presentó un fuerte efecto de *beam blockage* (**Figura 10**) durante los primeros días de prueba hasta que se logró elevar el ángulo de *beam* y se pudo mitigar el problema.

Figura 10. Efecto de *beam-blockage* en el área de medición de PIUXX debido a elevaciones (lomas) al Este de su ubicación.



Como puede apreciarse en la imagen anterior, las zonas blancas se producen porque el pulso encuentra un obstáculo en su recorrido. Así, las zonas blancas no registran dato alguno a comparación de las zonas grises donde el pulso del radar no encuentra obstáculos y recoge datos en todo su rango de alcance.

3.2 Tensorflow

Antes de entrar a la generación de los datos, es importante recalcar la parte de software a implementar. Para la generación del modelo se usarán las librerías que tiene mejor desempeño en cuanto a redes neuronales, *machine learning* y con ello el *Deep learning*. La librería más accesible y de fácil entendimiento es Tensorflow, y con la cual se trabajará en esta tesis.

Tensorflow es descrito por Serdar Yegulalp en su blog (2019) como; *una biblioteca de código abierto para el cálculo numérico y el aprendizaje automático a gran escala.*

Por otro lado, en cuanto a su aplicación menciona lo siguiente: “Tensorflow puede entrenar y ejecutar redes neuronales profundas para clasificación de dígitos escritos a mano, reconocimiento de imágenes, etc. Lo mejor de todo es que Tensorflow admite la predicción de producción a escala, con los mismos modelos que se usan para el entrenamiento” (párrafo 3).

Desde la parte funcional, es de corroborar la información rescatable del mismo autor del blog. Y se ha comprobado a lo largo de la Tesis, Serdar menciona también lo siguiente, lo cual es muy ventajoso al tratarse de un lenguaje de alto nivel: “Tensorflow permite a los desarrolladores crear gráficos de flujo de datos, estructuras que describen cómo se mueven los datos a través de un gráfico o una serie de nodos de procesamiento. Cada nodo en el gráfico representa una operación matemática y cada conexión o borde entre nodos es una matriz de datos multidimensional, o tensor” (Yegulalp, 2019).

Tensorflow, y tal como su nombre lo indica realiza cálculos a base de tensores, los datos utilizados en las librerías básicamente son tensores, los cuales son un vector o matriz de n-dimensiones. En nuestro caso de estudio, los datos organizados para el entrenamiento del modelo, son tensores de 5 dimensiones.

Actualmente la librería Tensorflow ha evolucionado y apunta a convertirse en algo más que una simple biblioteca. Su propia página web, lo describe de la siguiente manera (Tensorflow, s.f.); “Tensorflow es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que permite que los investigadores innoven con el aprendizaje automático y los desarrolladores creen e implementen aplicaciones con tecnología de AA (aprendizaje automático) fácilmente”.

Serdar (Yegulalp, 2019) propone que Tensorlfow a partir de 2019, se renovó para que sea más fácil trabajar con él, al integrar APIs tales como Keras, lo que proporciona facilidad y mayor rendimiento. De hecho, para la realización de nuestro modelo se usó la API Keras, como parte de la librería de Tensorflow.

3.2.1 Keras

La definición más precisa se encuentra en la propia página de Tensorflow (Keras. Tensorflow Core, 2020):

Keras es la API de alto nivel de Tensorflow para construir y entrenar modelos de aprendizaje profundo. Se utiliza para la creación rápida de prototipos, la investigación de vanguardia (estado-del-arte) y en producción, con tres ventajas clave.

- ***Amigable al usuario:*** Keras tiene una interfaz optimizada para casos de uso común. Proporciona información clara y procesable sobre los errores del usuario.

• ***Modular y configurable:*** Los modelos en Keras se fabrican conectando bloques de construcción configurables entre sí, con pocas restricciones.

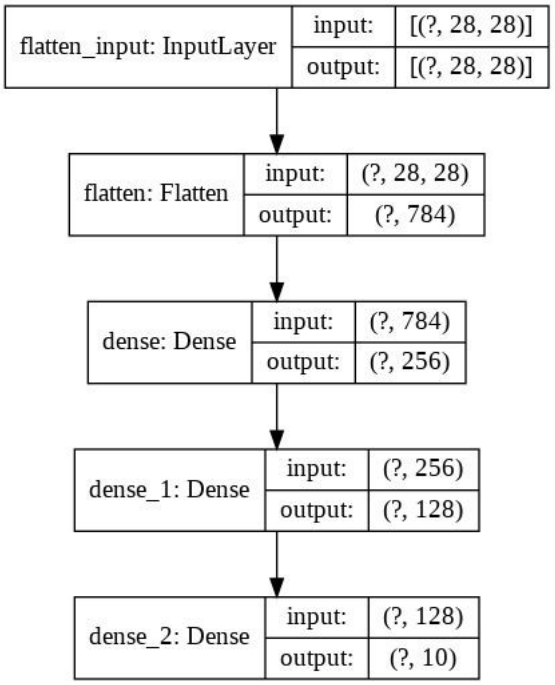
• ***Fácil de extender:*** Escribir bloques de construcción personalizados para expresar nuevas ideas para la investigación. Crea nuevas capas, métricas, funciones de pérdida y desarrolla modelos de estado-del-arte.

3.2.1.1 Modos de construcción de modelos en Keras. En Keras se pueden organizar las capas de un modelo de redes neuronales de dos formas: secuencial y funcional. Ambos muy parecidos, pero en algunas áreas las ventajas y desventajas se hacen notorias en ambos.

• ***Modelo Secuencial:*** Este modo de Keras permite construir un modelo capa por capa, apilando unas sobre otras, de tal manera que se van uniendo y se ejecutan una tras otra consecutivamente y como su nombre lo indica de manera secuencial (**Figura 11**). Una de las características principales es que solo permite una única entrada y produce también una única salida.

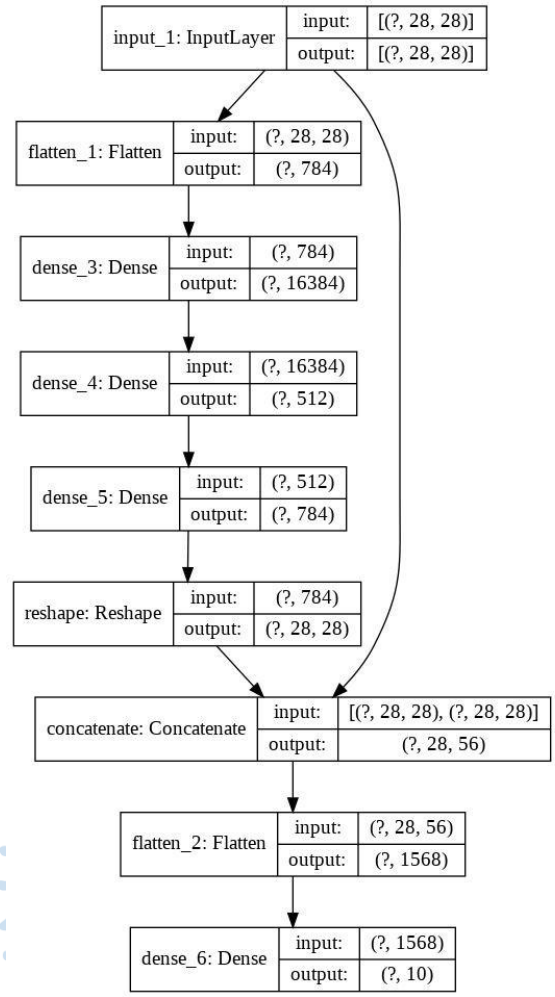
En el blog (Swain, 2021) el autor menciona lo siguiente: "El modelo secuencial es muy sencillo y fácil de usar. Pero no se permite compartir capas o ramificar capas. Además, no se puede tener varias entradas y salidas" (párrafo 3).

Figura 11. Esquema de un modelo de tipo secuencial



• ***Modelo Funcional:*** A diferencia del anterior modo, aquí se pueden realizar conexiones entre capas y robustecer aún más el modelo. Permite el acceso a múltiples entradas y por ende también lograr múltiples salidas al final del modelo (**Figura 12**). Sin embargo, como se menciona en (Programador Click, s.f.) el modelo que se construya es lento de compilar.

Figura 12. Esquema de un modelo de tipo funcional



3.2.2 Parámetros e hiperparámetros dentro de un modelo en Keras

3.2.2.1 Hiperparámetros. A continuación, una definición corta del artículo web de Brownlee (Brownlee, 2017b): “Un hiperparámetro de modelo es una configuración que es externa al modelo y cuyo valor no se puede estimar a partir de los datos”.

Así mismo, en el blog web (Nyuytiymbiy, 2020), para argumentar lo mencionado anteriormente, dice acerca de los hiperparámetros: “El algoritmo de aprendizaje utiliza los hiperparámetros cuando está aprendiendo, pero no forman parte del modelo resultante. Al final del proceso de aprendizaje, tenemos los parámetros del modelo entrenado, que es a lo que efectivamente nos referimos como modelo” (párrafo 6).

Además, hace mención de la manera en cómo se eligen de acuerdo al modelo y datos que está usando para el entrenamiento; “Como ingeniero de aprendizaje automático que diseña un modelo, usted elige y establece valores de hiperparámetros que su algoritmo de aprendizaje usará incluso antes de que comience el entrenamiento del modelo” (párrafo 5).

Finalmente, a modo de resumen, ambos autores (Brownlee, 2017b; Nyuytiymbiy, 2020) sostienen que un hiperparámetro es aquel valor que debe ser decidido por el

programador o quien desarrolle el modelo. Es un valor elegido incluso antes de que comience el entrenamiento.

Algunos ejemplos de hiperparámetros:

- Relación de división de prueba y entrenamiento.
- Taza de aprendizaje.
- Optimizador del modelo.
- Función de activación.
- Número de capas.
- Tamaño de kernel o filtro en capas convolucionales.
- Tamaño de Lote (*batch size*)
- Número de iteraciones (épocas de entrenamiento)

3.2.2.2 Parámetros de una red neuronal. Los parámetros, por otro lado, son internos al modelo, son valores que se aprenden o estiman a partir de los datos y cuando se ejecute un entrenamiento.

Nyuytiymbiy (2020) nuevamente ilustra un breve resumen de cómo se forman los parámetros:

> El entrenamiento del modelo generalmente comienza con la inicialización de los parámetros en algunos valores (valores aleatorios o establecidos en cero). A medida que avanza el entrenamiento o aprendizaje, los valores iniciales se actualizan mediante un algoritmo de optimización. El algoritmo de aprendizaje actualiza continuamente los valores de los parámetros a medida que avanza el aprendizaje, pero los valores de los hiperparámetros establecidos por el diseñador del modelo permanecen sin cambios (párrafo 10).

Básicamente, cuando se construye un modelo también se generan espacios internos que alojan a los parámetros, que una vez inicializado el entrenamiento comenzarán a actualizarse, comúnmente son los pesos que además constituyen el modelo en sí.

Ejemplos de parámetros:

- -Los coeficientes (o pesos) del modelo.
- -Sesgos.
- -Centroides del clúster.

3.2.3 Hiperparámetros comunes en modelos de redes neuronales

Usualmente dentro de la biblioteca Keras o Tensorflow, las capas ya están realizadas, solo se necesitan ajustar los hiperparámetros que lograrán los mejores resultados en la salida del modelo. Normalmente vienen valores por defecto y se muestran de la siguiente manera.

El siguiente ejemplo es para una capa convolucional 2D:

```
tf.Keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid',
    data_format=None, dilation_rate=(1, 1), groups=1, activation=None,
    use_bias=True, kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None,
    bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None,
    bias_constraint=None, **kwargs
)
```

Este otro ejemplo es para una red Convolucional LSTM 2D también (Conv2D Layer, s.f.; ConvLSTM2D Layer, s.f.):

```
tf.Keras.layers.ConvLSTM2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    activation="tanh",
    recurrent_activation="hard_sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    dropout=0.0,
    recurrent_dropout=0.0,
    **kwargs
)
```

Puede apreciarse que en cada inicio de la capa se tiene la sigla `tf` que hace referencia a la librería de Tensorflow. Seguidamente le sigue la sub-librería Keras y posteriormente el nombre de la capa. Dentro de los paréntesis están todos los hiperparámetros que se pueden modificar. Aunque usualmente solo los primeros cinco o seis hiperparámetros son los más usados.

Los hiperparámetros más usados ya sea en los modelos secuenciales o modelos funcionales, son los siguientes: *filters*, *kernel size*, *activation*, *stride*, *padding* son los más usados para modificar dentro de un modelo.

A continuación, se presenta un breve resumen de lo que significan estos hiperparámetros para más adelante en el capítulo de experimentación y resultados, comprender a qué se refieren las modificaciones que se realizaron al modelo.

3.2.3.1 Filtro y tamaño de filtro (kernel). Tal y como menciona Ramesh (2018) en un artículo web:

> Un filtro es una matriz de pesos con la que convolucionamos en la entrada. El filtro de convolución proporciona una medida de qué tan cerca se parece un parche de entrada a una característica. Una característica puede ser un borde vertical o un arco, o cualquier forma. Los pesos en la matriz de filtro se derivan mientras se entrena los datos. Los filtros más pequeños recopilan tanta información local como sea posible, los filtros más grandes representan información global, de alto nivel y representativa (párrafo 8).

Además, añade que los filtros que se usan generalmente llevan números impares, por ejemplo; los más pequeños 3x3 o 5x5 o también los filtros grandes 11x11 o 9x9.

3.2.3.2 Relleno (padding). El autor del artículo web señala también, respecto al *padding* o relleno en español:

El *padding* se usa generalmente para agregar columnas y filas de ceros para mantener los tamaños espaciales constantes después de la convolución, hacer esto podría mejorar el rendimiento ya que retiene la información en los bordes. Los parámetros para la función *padding* son: *Same* y *Valid*.

El parámetro *Same* logra que el tamaño de salida sea el mismo que el de entrada al rellenar uniformemente a izquierda y derecha, pero si la cantidad de columnas que se agregan es impar, agregará la columna adicional a la derecha.

Por otro lado, el parámetro *Valid* hace que el tamaño de salida se reduzca al máximo entero de la siguiente ecuación:

$$\frac{n + f - 1}{s}$$

Donde ‘n’ son las dimensiones de entrada, ‘f’ es el tamaño de filtro y ‘s’ es la longitud del paso. De esta manera no se produce relleno (Ramesh, 2018, párrafo 9).

3.2.3.3 Paso (stride). Generalmente es el número de píxeles que se desea omitir mientras recorre la entrada horizontal y vertical durante después de que se haya realizado una multiplicación de elementos de los pesos de entrada con los del filtro. Se utiliza para disminuir considerablemente el tamaño de la imagen de entrada, ya que después de la operación de convolución el tamaño se reduce debido a la operación del padding, mostrada anteriormente (Ramesh, 2018, párrafo 10).

3.3 Métricas de desempeño

Tal y como lo menciona la propia página de Keras (Metrics. Keras, s.f.): Una métrica es una función que se utiliza para juzgar el rendimiento de su modelo. Las funciones métricas son similares a las funciones de pérdida, excepto que los resultados de evaluar una métrica no se utilizan al entrenar el modelo. Además, señala que se puede usar cualquier función de pérdida como métrica.

Al igual que en el apartado de pérdidas, también hay una variedad de decenas de funciones métricas. Pero aquí se detallarán las más populares y las que se usaron en la arquitectura de red neuronal. Se encuentran también MAE, MSE, y RMSE, pero ya se conceptualizaron en la parte de funciones de pérdida.

3.3.1 Accuracy (Exactitud)

Según (Bajaj, 2021) nos dice que la métrica de precisión se define como el número de predicciones correctas dividido por el número total de predicciones, multiplicado por 100.

De todos los ejemplos que se han revisado mientras se desarrollaba esta tesis, la métrica de precisión *(accuracy)* es la que se ha encontrado en todos los casos. Es la más usada, la más fácil de implementar y la más rápido de calcular. Sin embargo, no es aplicable para cualquier tipo de datos como se verá más adelante en el capítulo que concierne a la experimentación del modelo.

A continuación, se detallarán dos métricas muy usadas para evaluar el rendimiento del modelo y cuando los datos son imágenes, que es a lo que se resumen los datos del radar. Hasta ahora no pertenecen al conjunto de clases de la librería Keras. Pero se pueden adaptar dentro de Tensorflow.

3.3.2 SSIM (Structural Similarity Index)

No hay mejor explicación de la métrica de índice de Similitud Estructural como define Delgado (2019, párrafo 1): “Es una métrica de percepción que cuantifica la degradación de la calidad de la imagen causada por el procesamiento, como la compresión de datos o por pérdidas en la transmisión de datos.”

En (SSIM: Structural Similarity Index. Imatest, s.f.) se indica que SSIM es una métrica de referencia completa que requiere dos imágenes de la misma captura: una imagen de referencia y una imagen procesada. Para nuestro modelo la métrica de SSIM comparará la similitud entre la secuencia de observaciones reales y la secuencia de observaciones predichas, reconstruidas.

Finalmente, SSIM opera en un rango entre 0 y 1, donde el valor de 1 equivale a un buen resultado del modelo, una similitud entre ambas imágenes.

3.3.3 PSNR (Peak Signal-to-Noise Ratio)

Esta otra métrica relacionada también a la comparación de imágenes, es importante porque trabaja en el rango de los valores de píxeles de las imágenes o los datos. De alguna manera se podrá obtener una mejor observación de cada píxel en la imagen.

En el artículo web (Peak Signal-to-Noise Ratio as an Image Quality Metric, 2020) se explica muy bien el concepto de esta métrica:

El término *Peak Signal-to-Noise Ratio* (PSNR) es una expresión para la relación entre el valor máximo posible (potencia) de una señal y la potencia del ruido distorsionante que afecta la calidad de su representación. Debido a que muchas señales tienen un rango dinámico muy amplio (relación entre los valores grandes y más pequeños posibles de una cantidad variable), el PSNR generalmente se expresa en términos de la escala de decibelios logarítmicos.

También menciona que; mientras mayor sea el valor de PSNR, mejor se habrá reconstruido la imagen degradada para que coincida con la imagen original y mejor será el algoritmo reconstructivo. Una de las desventajas de esta métrica es que no tiene en cuenta ningún nivel de factores biológicos del sistema de visión humana, como el índice de similitud estructural.

3.4 Preparación de datos

En lo que va de este capítulo, se pudo conocer la forma en que el software del Radar de lluvias presenta sus datos registrados. En esta sección se explicará todo el post-procesamiento realizado para dar una forma, calidad y cantidad de datos para el modelo de predicción.

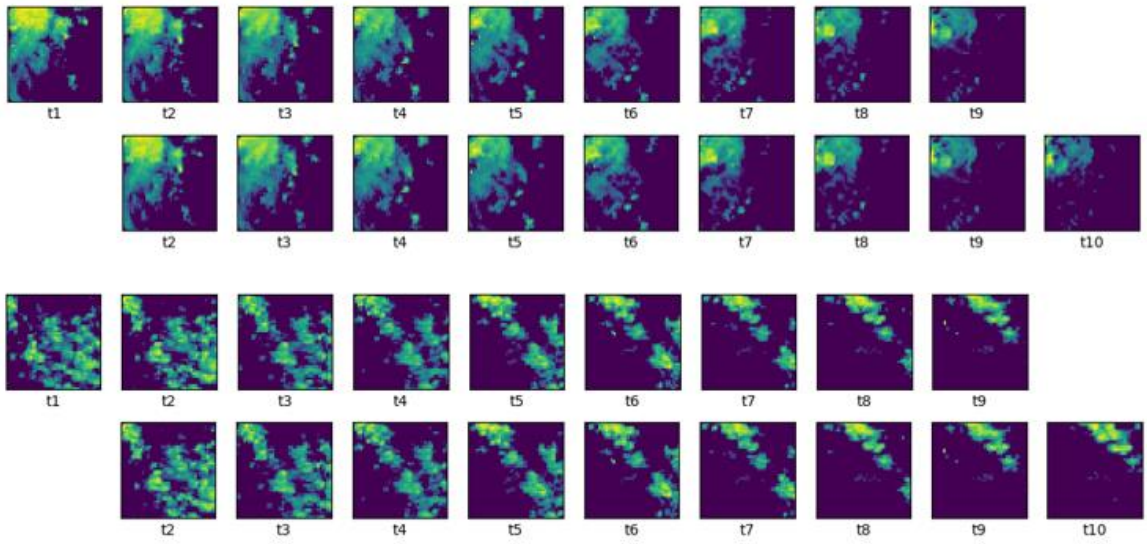
El Radar de lluvias suministra datos en archivos NETCDF (*Network Common Data Form*), según la forma en que se ha dispuesto entregar los datos; de manera diaria, semanal, mensual, anual. Para trabajar con los archivos se usa una terminal con el software Python, que proporciona las librerías necesarias para la extracción de los datos.

Durante la revisión de los datos, se observó que el total de datos no siguen una secuencia temporal consecutiva. Es decir, se encontraron saltos de tiempo horarios y diarios en su mayoría. Lo cual dificultaría y de alguna manera definiría el tamaño y forma de los datos finales de entrenamiento.

Se sabe bien que para los modelos de *Machine Learning* se requiere una vasta cantidad de datos para el proceso de entrenamiento del modelo. Desafortunadamente no era nuestro caso. En otros ejemplos de predicción de secuencias de fotogramas que usan imágenes en movimiento como MNIST o KTH, se pudo observar que usan alrededor de 10000 secuencias de entrenamiento. Y en lo que respecta a la cantidad de elementos por secuencia usan entre de 15 a 20 elementos o fotogramas. Por lo mencionado, en base a la cantidad de datos y además teniendo en cuenta el problema de saltos temporales en los archivos NETCDF, se decidió trabajar con 10 elementos o fotogramas por secuencia.

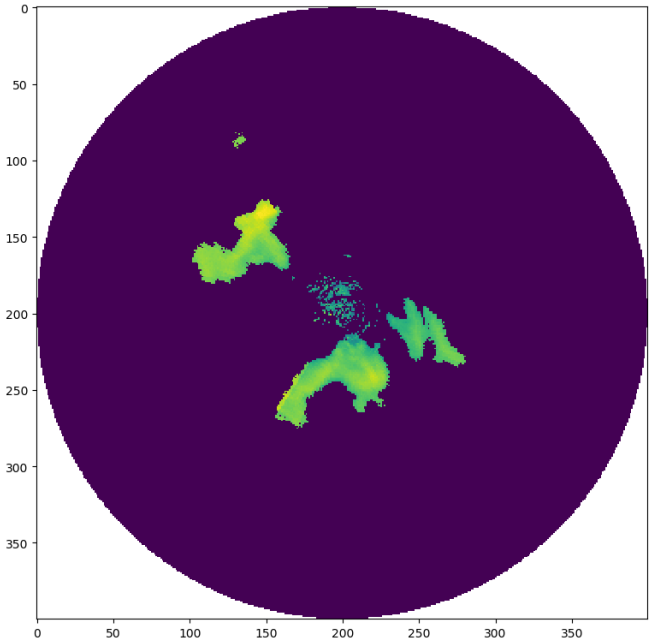
Según nuestro modelo de trabajo, requiere para tal una secuencia de entrada y una secuencia de salida. Por ello se separó cada secuencia de 10 fotogramas en 9 fotogramas de entrada y 9 fotogramas de salida. La segunda secuencia obtenida a partir de dar un salto de tiempo a la primera secuencia. El párrafo se explica mejor en la **Figura 13**.

Figura 13. Secuencia de cuadros de entrada y salida para el entrenamiento del modelo



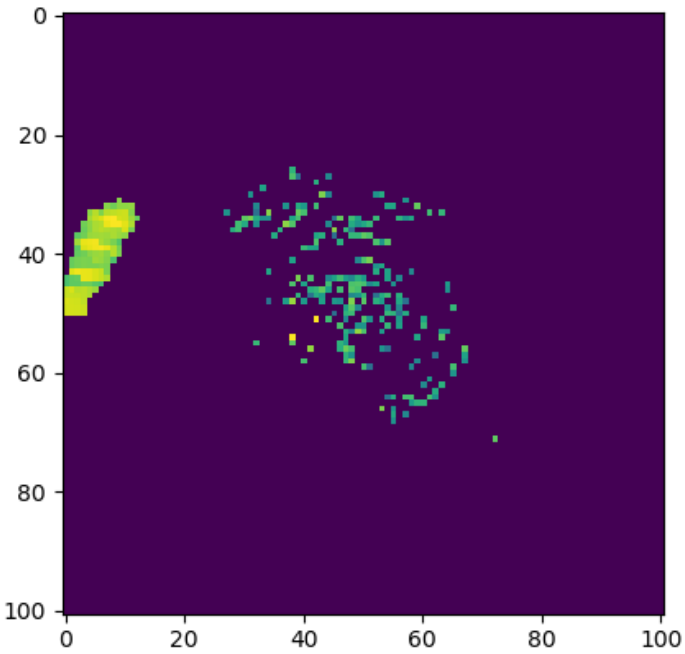
Otro detalle a tener en cuenta es el tamaño de cada cuadro. Anteriormente se mencionó que el tamaño en píxeles que se obtienen de extraer los datos es de 400x400 y corresponde a todo en rango del Radar. Lo más favorable para los resultados, sería usar los datos en su tamaño total, pero debido al costo computacional, el tiempo de entrenamiento (lo cual se evidenció durante la experimentación) y los resultados de predicción. Se redujo el tamaño de los datos, y no se trata de reescalar el tamaño de 400x400 a un tamaño menor. Sino de seleccionar el área de nxn píxeles dentro del área de NxN píxeles tal y como lo mencionan en (Bonnet et al., 2020b; Godoy Mendía, 2019)

Figura 14. Cuadro original de 400 x 400 pixeles obtenido a partir del archivo de tipo NETCDF



En la imagen anterior, el círculo muestra el alcance total del radar(100km). Dado que se va a extraer solo una parte interior del área circular, se tratará de obtener los datos más limpios para el entrenamiento. Es decir, no de tomará un área cuadrada en el centro del círculo, sino más bien en un sector central a lo largo de un radio de 100 km. Tomar el área central equivale a tomar en cuenta el *clutter* que no se corresponde con datos de precipitación. Como se mencionó anteriormente son objetos externos.

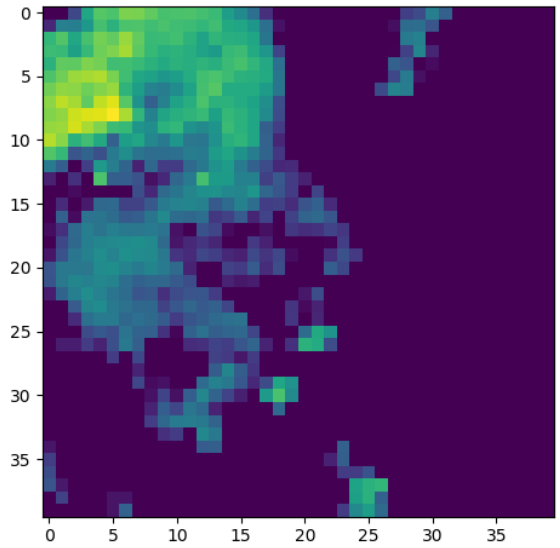
Figura 15. Recorte central de 100 x 100 pixeles en el área total del alcance del radar



En los datos obtenidos de Radar PIUXX, se aprecia en la figura anterior en color un tanto amarillento una pequeña celda de lluvia al Oeste de la ubicación del radar en el centro, mientras que los píxeles de color verde alrededor del centro corresponden a las interferencias de las construcciones de la ciudad de Piura.

De ese modo entonces se usarán datos que solo contengan precipitación. Normalmente se escogerá los datos en la zona Este del área circular del radar que corresponden a las provincias de Piura donde más suelen ser las ocurrencias de precipitación. A continuación, una imagen de lo que sería un “cuadro” de la secuencia de entrada al modelo, el cuál será de 40x40 en resolución de tamaño de píxeles.

Figura 16. Recorte de 40 x 40 pixeles en un área interna del alcance del radar que contiene solo celdas de lluvia.

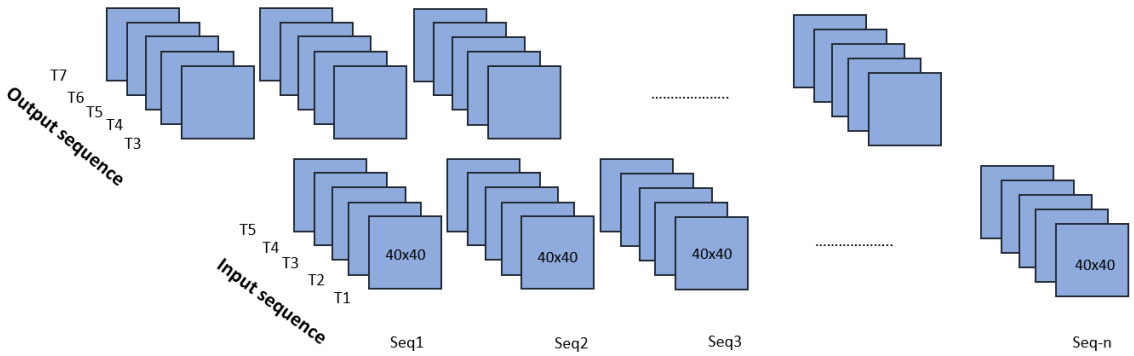


Finalmente, la forma de nuestros datos de entrenamiento será de 5 dimensiones: (n_secuencias, n_frames, ancho, alto, canal).

- **n_secuencias**, corresponde al número de secuencias totales, cada secuencia es compuesta por una cantidad de frames.
- **n_frames**, hace referencia a la cantidad de frames en una secuencia. En nuestro caso el total de frames es 9.
- **Ancho**, el ancho en píxeles de cada frame que es 40.
- **Alto**, también referido a la forma del frame, y es 40.
- **Canal**, se refiere a si es una imagen RGB o de un solo canal, que es nuestro caso.

Puede mostrarse, en la **Figura 17** la disposición en que quedan nuestros datos de entrenamiento.

Figura 17. Disposición y formación de los datos de entrada para el modelo



Ahora bien. Para un mejor entrenamiento se aconseja separar el total del conjunto de datos en dos grupos, uno de entrenamiento propiamente y el otro más pequeño de testeo o validación. Generalmente se aconseja 80% para el entrenamiento y 20% para validación que es lo que se ha seguido en esta tesis. Todo ello se hace para que la red ejecute primero los casos de entrenamiento de esta manera va aprendiendo las características de los datos, posteriormente los datos de testeo se ejecutan para ver la performance del modelo y cuánto va mejorando.

Previamente a escoger los datos entrenamiento al modelo, se realizó una normalización. Cuando los datos que se utilizan son imágenes los píxeles son convertidos a valores más pequeños, un rango entre 0 y 1. Para lograr tal conversión se definió una ecuación tal que se tomó el mayor valor de reflectividad del conjunto total de datos y el valor mínimo, el cual ya era conocido (-31.5dB) de tal manera que hay una relación entre ambos valores para la normalización. La siguiente ecuación define la relación entre ambos valores para llevar los píxeles a valores entre 0 y 1.

$$dato\_normalizado = \frac{dato - dato\_min}{dato\_max - dato\_min}$$

Donde dato es el dato actual, y dato_min y dato_max corresponde a los valores de píxel mínimo y máximo respectivamente del conjunto total de datos. Pixel_normalizado corresponde al actual valor del pixel y está entre 0 y 1.

3.5 Preparación de arquitectura de autocodificador variacional para predicción de los datos de precipitación

En base a todo el aspecto teórico relacionado a redes neuronales desde el punto de vista de Tensorflow y Keras, junto con sus parámetros e hiperparámetros. Así como la obtención y preparación de los datos dirigidos en tamaño y forma para la arquitectura que se va a usar. Ahora es momento de presentar el modelo de autocodificador variacional seleccionado, para predecir secuencias posteriores de precipitación.

El modelo a simple vista es pequeño y puede tratarse de simple. Por otro lado, durante las pruebas que se hicieron al modelo, se pudo encontrar que hay una estrecha relación con el tipo, tamaño y forma de los datos. Es por ello que implica además de presentar los datos para el modelo, también se deben hacer ajuste de los parámetros e hiperparámetros del modelo.

A continuación, presentamos el modelo “genérico” que se usó para este proyecto. Es un modelo que aplica Sellat (2019) en un artículo web, para su proyecto de detección de anomalías en cuadros de video. Pero que simplemente el modelo se puede adaptar a la predicción de precipitación.

```
seq = Sequential()
seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=4,
padding="same"),
    batch_input_shape=(None, None, 40, 40, 1)))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same")))
seq.add(LayerNormalization())
# # # # #
seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(32, (3, 3), padding="same", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
seq.add(LayerNormalization())
# # # # #
seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2,
    padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=4,
    padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid",
    padding="same")))
```

De acuerdo a lo explicado al inicio de este capítulo, puede entenderse que el modelo que se ha usado es de tipo secuencial. Ya que ahorrará un poco más de tiempo para el entrenamiento, y además la forma en la que están dispuestos los datos permiten su uso.

Puede notarse además que en la estructura del modelo hay una separación en tres bloques. Un autocodificador primero codifica todos los datos de entrada, luego en medio de la secuencia tiene un espacio latente, posteriormente decodifica todos los datos estructurados.

También se aprecia que en cada línea del código se agrega una capa, en este caso en el codificador y decodificador capas convolucionales de dos dimensiones. En el bloque central se tienen capas convolucionales LSTM (Long-Short Term Memory).

Para detallar el funcionamiento de cada capa seleccionada, aquí se expone. Las capas convolucionales 2D se usan para el aprendizaje espacial, es decir las características de la forma de celdas de precipitación, bordes de las celdas y en base a los valores de los píxeles. Las capas Convolucionales LSTM buscarán por otro lado el aprendizaje temporal, es decir el cambio de la forma de la celda de precipitación entre cuadros de una secuencia.

Hay algo importante de mencionar y es la presencia de los hiperparámetros dentro de cada capa añadida, y en cada línea de la estructura del código. También ya se mencionaron los hiperparámetros más usados y cada uno de ellos son modificables. En el capítulo siguiente se detallará más sobre el resultado que se obtiene al interactuar con algunos de ellos y que nos ha llevado a una estructura final y modelo definitivo para el tipo de datos que se tienen.

Por ahora solo se puede mencionar que un modelo depende del tipo de datos y de los hiperparámetros que se usen en su estructura.

Por último, mencionar los hiperparámetros más importantes, que también se añaden a la estructura del autocodificador. La función de pérdida y la función de optimización.

```
seq.compile(loss='mse',
    optimizer=Keras.optimizers.Adam(lr=1e-4, decay=1e-5, epsilon=1e-6))
```

Para la función de pérdida (hiperparámetro *loss*) la estructural que se toma de ejemplo (Sellat, 2019), toma a la pérdida de MSE para sus datos de entrenamiento. Para la función de optimización elige ADAM, y una taza de aprendizaje de 0.0001. Para un primer inicio se realizará un primer entrenamiento con los valores del ejemplo, y en base a los resultados se decidirá por modificar o mantener los hiperparámetros. Todo ello se explica en el capítulo siguiente.

UNIVERSITAS STUDIORUM PIURENSIS

Capítulo 4
Experimentación y validación

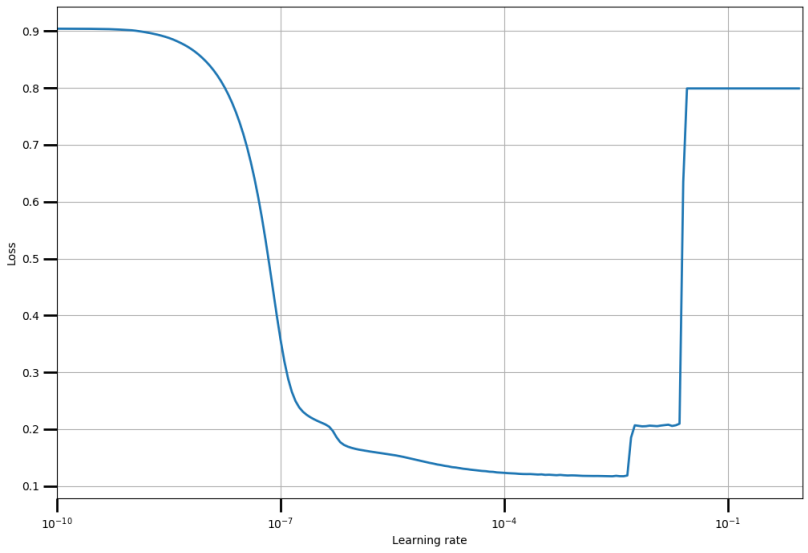
En este capítulo se detallará la fase experimental y los resultados que se obtuvieron de probar diferentes hiperparámetros a lo largo de la red neuronal seleccionada.

La literatura en cuanto a redes neuronales y aprendizaje automático ha enseñado que la mejor manera de obtener resultados óptimos es a través de la prueba y el error. No hay valores establecidos para los hiperparámetros y parámetros en las redes neuronales.

Basándose en estas propuestas, se tendrá en cuenta varias modificaciones a lo largo del modelo de autoencoder variacional desarrollado, tal y como se mencionó en el capítulo 3 debido al ajuste de los hiperparámetros en el modelo.

Un primer procedimiento que se realizó, consistió en evaluar la tasa de aprendizaje (*learning rate)*, para evaluar un punto inicial por el cuál comenzar a entrenar nuestro modelo. Por ello se realizó una prueba de entrenamiento de 200 iteraciones para lo cual la taza de aprendizaje se encuentra entre 1e-10 y 1e-0. De la gráfica obtenida se podrá identificar en qué valor de taza de aprendizaje la pérdida es menor.

Figura 18. Selección adecuada de la taza de aprendizaje de acuerdo al menor valor de pérdida



Un método para hallar el valor óptimo de tasa de aprendizaje (Rosebrock, 2019) es tomar un intervalo de valores de taza de aprendizaje en donde la pérdida sea mínima, luego ir cerrando cada vez más el intervalo. Para nuestro caso se va a tomar un rango de valores, desde la gráfica mostrada anteriormente, entre 5e-5 y 2e-3. El primer valor debido a que es donde el valor de pérdida comienza a tener valores muy bajos, y el valor de 2e-3 se selecciona porque es el último punto donde la pérdida es menor, antes de que comience a incrementarse.

Ahora se prosigue a probar valores intermedios en el rango seleccionado y analizar los resultados.

4.1 Análisis del modelo teniendo en cuenta el optimizador

A continuación, en una primera fase experimental se usó la metodología expuesta por Brownlee (2020b), solo ajustando hiperparámetros en las capas del autoencoder convolucional, tales como función de activación, función de pérdida y elección del optimizador. En una primera instancia se comprobó que no son suficientes sintonizar los hiperparámetros mencionados. Por tal también se realizará una evaluación de los parámetros propios del Optimizador.

En base a los primeros experimentos se obtuvo que el optimizador ADAM supera a los demás optimizadores (al menos en el modelo propuesto y de acuerdo a los datos que estamos trabajando), tal parece que ADAM se adapta mejor al problema de desvanecimiento de descenso del gradiente. Por ello en las siguientes figuras exponemos los resultados obtenidos según los tres optimizadores más conocidos: *Stochastic Gradient Descent (SGD)* (Figura 20)*, Adam* (Figura 21) *y Adadelta* (Figura 22). La **Figura 19** es la secuencia real que se debe obtener. Los tres optimizadores usaron la tasa de aprendizaje mencionada al final del capítulo 3, el modelo genérico lr = 1e-4.

Figura 19. Secuencia real de datos, la salida

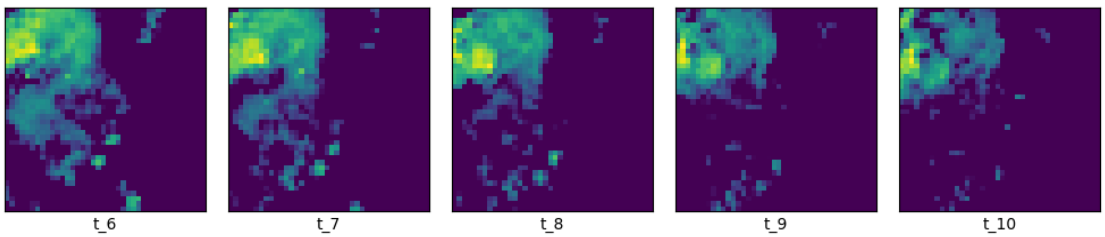


Figura 20. Secuencia predicha obtenida por el modelo usando el optimizador SGD

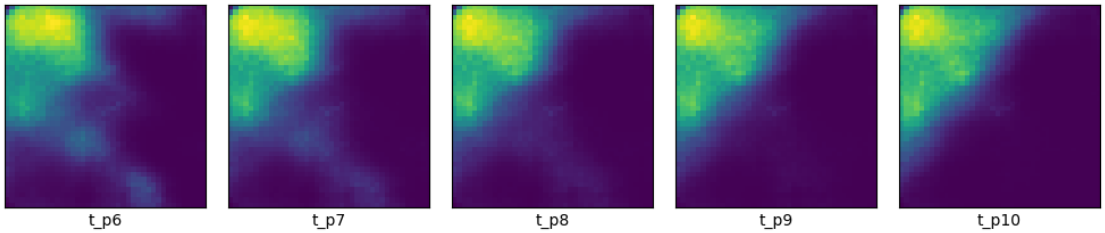


Figura 21. Secuencia predicha obtenida por el modelo usando el optimizador Adam

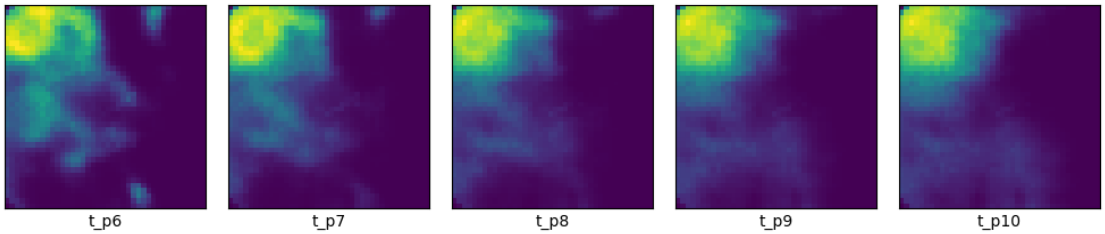
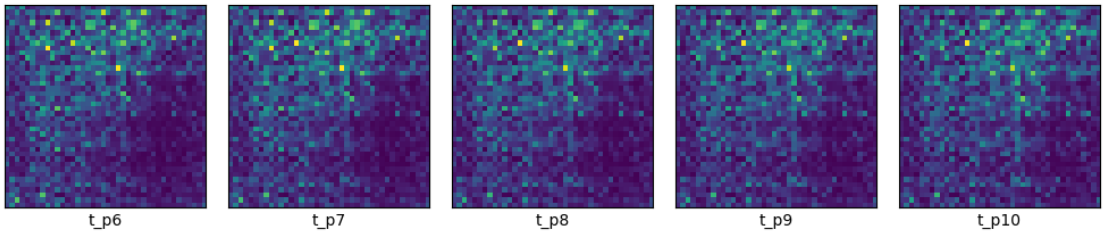


Figura 22. Secuencia predicha obtenida por el modelo usando el optimizador Adadelta



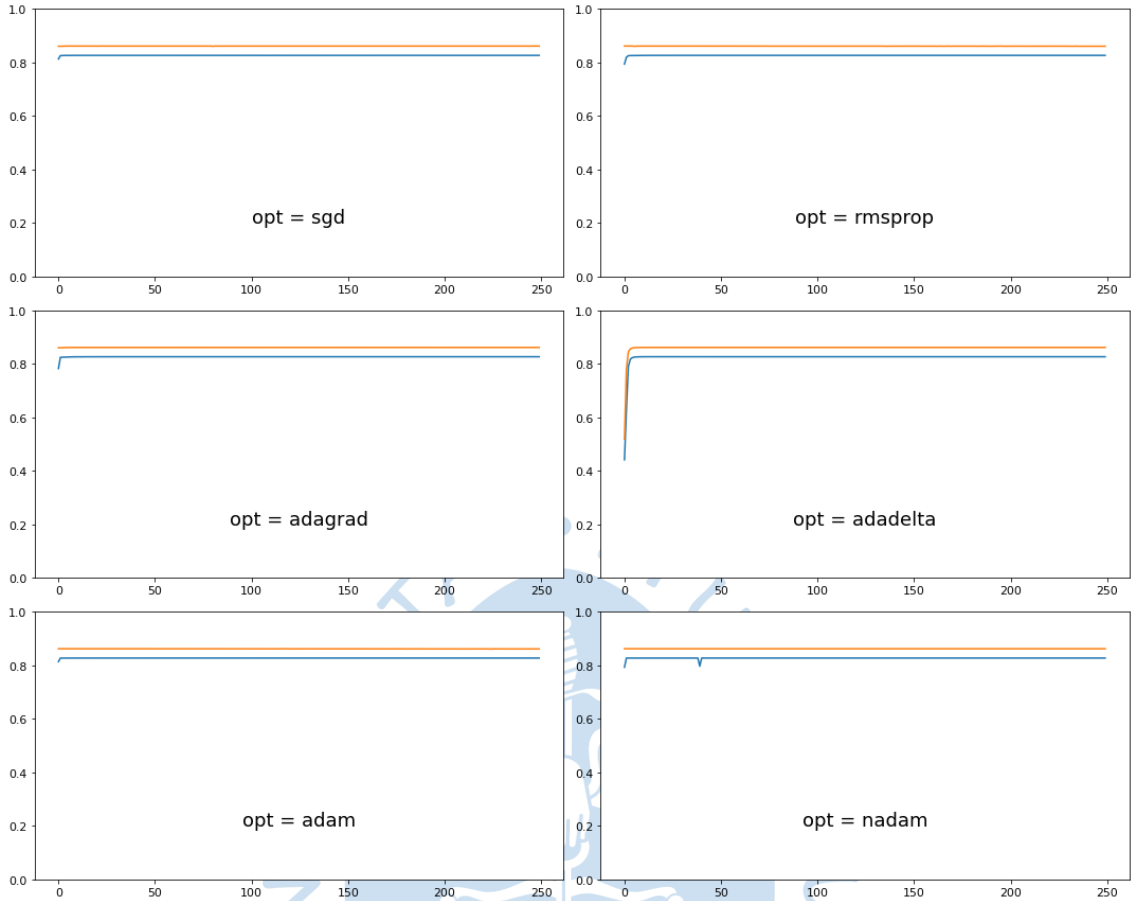
Como se aprecia en los resultados anteriores, los optimizadoras ADAM y SGD visualmente se acercan bastante a la predicción solo usando unas 100 épocas de entrenamiento, significa que han guardado la información de forma e intensidad de las celdas de lluvia. ADADELTA por su parte tiende a formar un cúmulo de píxeles desordenados en el campo de la imagen y es evidente el poco aprendizaje captado por el modelo.

Entre los dos mejores optimizadores, puede apreciarse que entre ADAM y SGD, el primero predice valores pequeños de intensidad de precipitación observados por el radar, a diferencia de SGD que elimina totalmente los valores pequeños. En ambos se aprecia un suavizado en el contorno de las celdas de precipitación, siendo aún mayor con el optimizador SGD. Que anula los pequeños valores logrando un límite notorio en las fronteras de la celda de lluvia.

Finalmente se decidió por el optimizador ADAM que parece desempeñarse mejor. A partir de este punto solo será de interés el estudio de los parámetros internos del optimizador ADAM. Se conoce a estos parámetros como: *learning_rate, decay, épsilon*.

La **Figura 23** nos muestra una evaluación del entrenamiento usando seis optimizadores utilizados para la evaluación. Las dos curvas corresponden a la métrica de exactitud (*accuracy*), en color azul el entrenamiento y en color naranja la validación.

Figura 23. Exactitud frente a los diferentes tipos de funciones de optimización



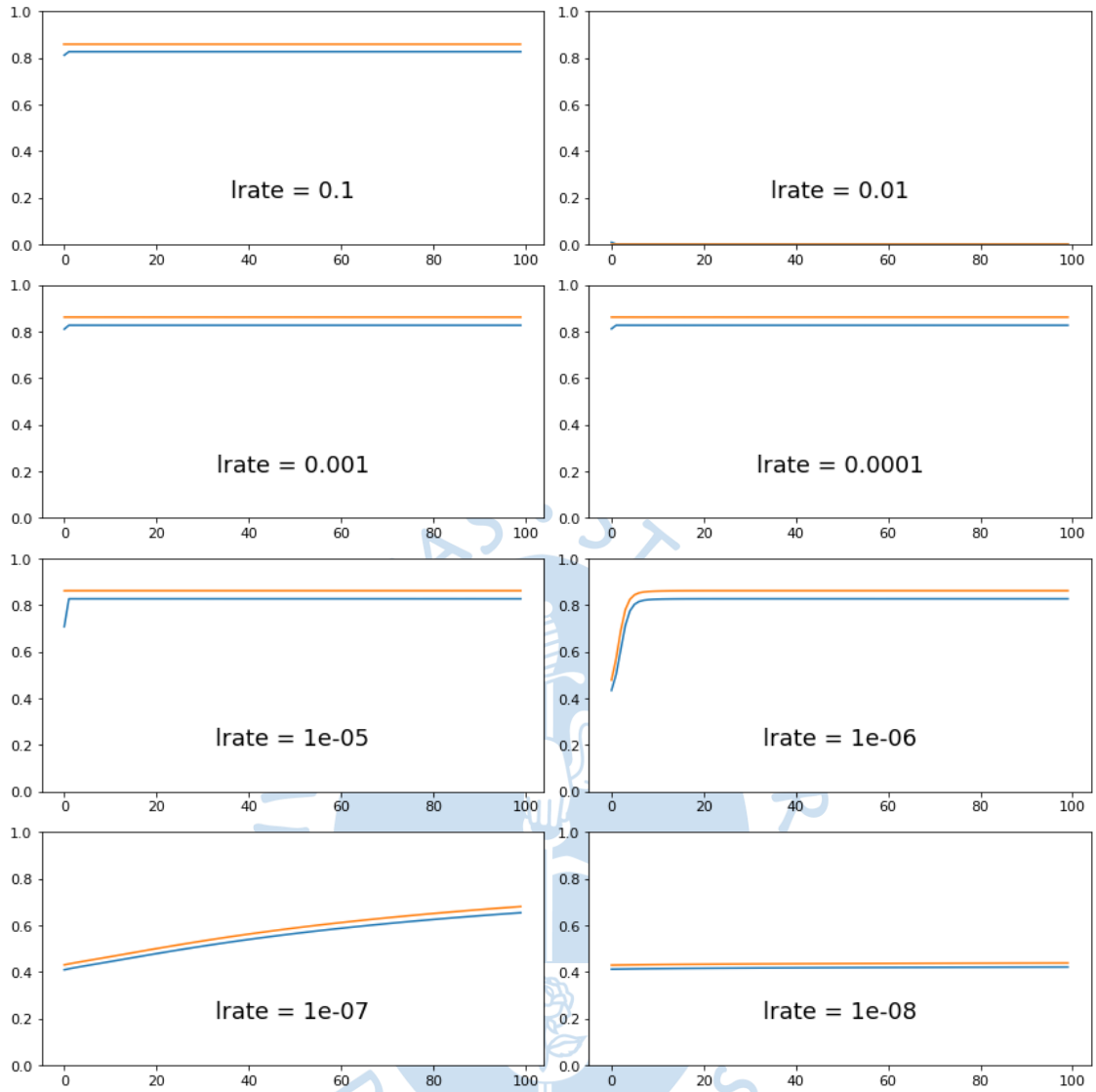
De las gráficas anteriores se deduce que la mayoría de los optimizadores alcanzan un aprendizaje bastante rápido logrando una exactitud del 85% aproximadamente, al menos en las 30 primeras épocas de entrenamiento. Un caso peculiar se obtiene con el optimizador ADADELTA que se toma un tiempo de aprendizaje, en comparación a los otros optimizadores, para alcanzar un valor estable. Aunque se aprecia la evolución de la curva se parece mucho más a una curva de aprendizaje, los resultados de la **Figura 22** se contraponen a los resultados obtenidos en este análisis usando la métrica *accuracy*.

También se aprecia que para el tipo de datos que se usan en el entrenamiento, tal parece que se requieren muy pocas épocas. Ello es porque la curva alcanza su estabilidad rápidamente. Sin embargo, el entrenamiento requerirá tomarse un tiempo extra ya que necesita que haya pequeñas actualizaciones de los pesos de la red neuronal.

La taza de aprendizaje (*learning rate*) será un parámetro clave a evaluar, algunos autores aseguran que la taza de aprendizaje puede ser el parámetro más importante de una red neuronal. El *decay* o decaimiento hace referencia al descenso en valor de la taza de aprendizaje cuando la pérdida durante la ejecución del entrenamiento empieza a aumentar.

Se presentan a continuación la variabilidad de la taza de aprendizaje entre un rango de 1e-1 hasta 1e-8, teniendo en cuenta los otros parámetros constantes (**Figura 24**).

Figura 24. Prueba con diferentes tasas de aprendizaje usando el optimizador Adam



En las gráficas anteriores, en el eje horizontal las iteraciones o épocas en el entorno de Python y en el eje vertical se indica la exactitud del modelo (*accuracy*). En cuanto a los resultados obtenidos se consideró usar un rango para la taza de aprendizaje entre 1e-1 hasta 1e-8, ya que el método de Brownlee (2020b) lo señala como unos valores para los cuales mejor se adapta el entrenamiento de una red neuronal. Y además la gráfica de tasa de aprendizaje vs épocas de entrenamiento (Figura 18) refuerza lo mencionado acerca del rango de valores adoptado.

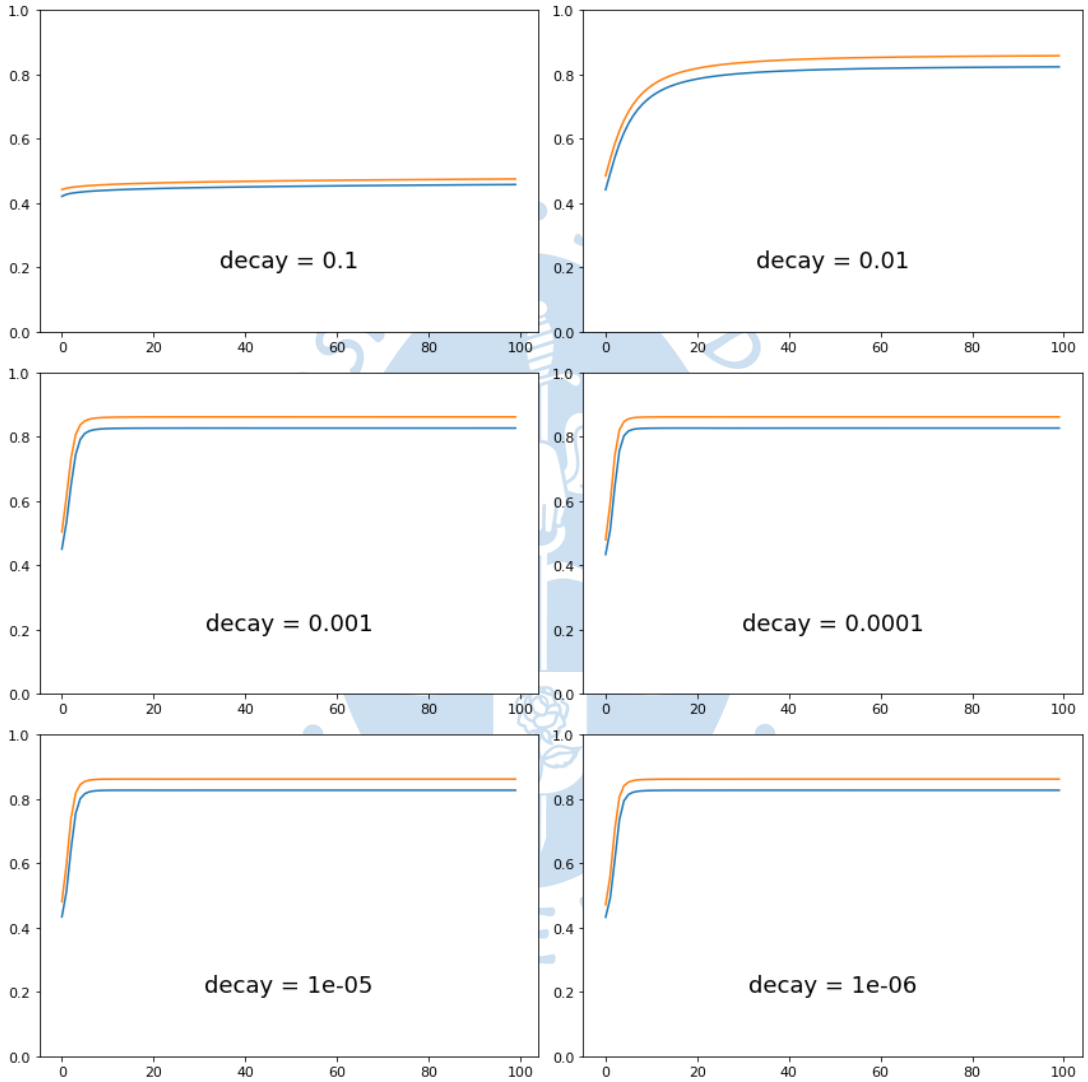
Usando el optimizador ADAM se tiene que los mejores valores se encuentran entre 1e-4 y 1e-5, ya que presentan una convergencia rápida en pocas épocas de entrenamiento. La tasa de aprendizaje de 1e-6 logra entrenar aún más los pesos de la red neuronal, aunque llega a establecerse por debajo de lo que alcanzan sus anteriores valores.

La taza de aprendizaje lr = 1e-7, toma más tiempo de entrenamiento y aun así no logra alcanzar el máximo valor alcanzado por los valores de taza de aprendizaje mencionados

anteriormente. Así mismo, el valor de lr = 1e-8 no logra converger a ningún valor, no logra elevar la precisión del entrenamiento.

Ahora una vez seleccionados los valores de tazas de aprendizajes iniciales de ambos optimizadores, se procede a trabajar el segundo hiperparámetro. *Decay rate* para el optimizador ADAM y *rho* para ADADELTA. Los resultados se muestran a continuación:

Figura 25. Pruebas variando la tasa de decaimiento frente a un valor te tasa de aprendizaje Lr=1e-4

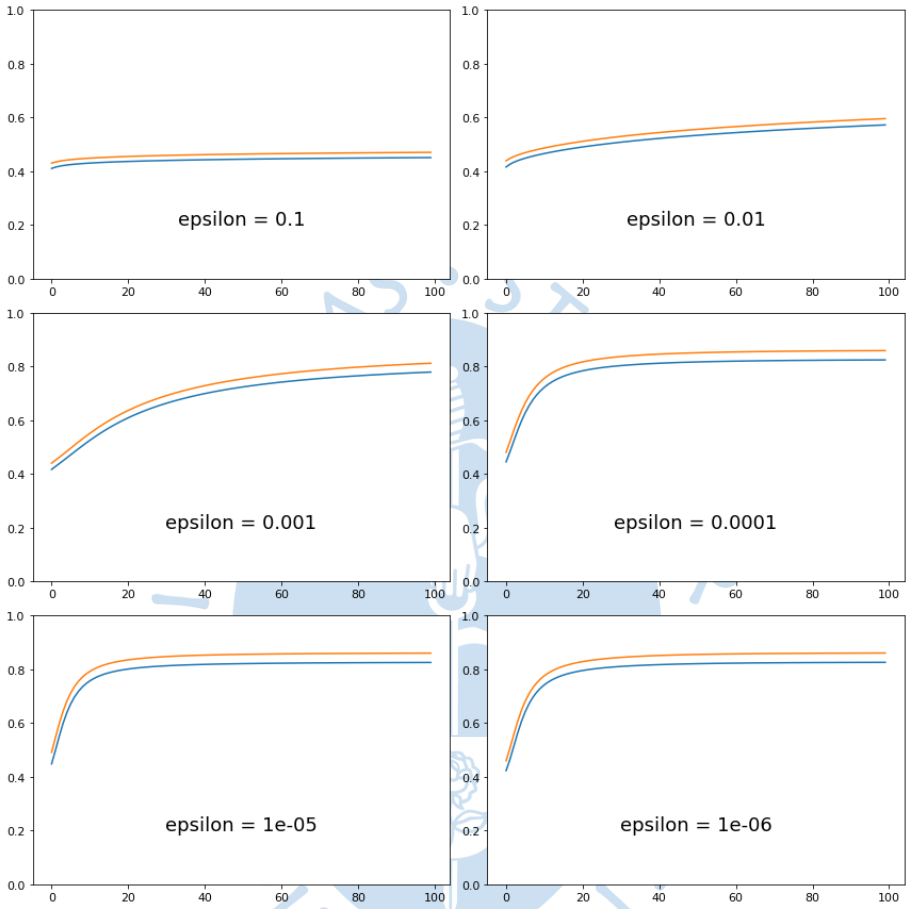


Mientras que el eje horizontal equivale al número de épocas, el eje vertical es la precisión del modelo. Valores altos de tasa de decaimiento se requieren bastante entrenamiento y no se logra la convergencia al máximo valor de precisión. Por otro lado, valores más pequeños logran un rápido entrenamiento. Para este procedimiento un valor de taza de decaimiento de 0.001 resuelve mejor el entrenamiento y actualizaciones de los pesos internamente en el modelo.

Ahora el último hiperparámetro, épsilon. Épsilon promete dentro del proceso de entrenamiento mejorar la rapidez y la actualización de los pesos respecto al descenso de

gradiente, valores mayores funcionan mejor que valores pequeños. Para la evaluación de encontrar los valores de épsilon en ambos optimizadores, se tendrá en cuenta un rango de valores entre 1e-1 y 1e-6 para ambos optimizadores, valores muy pequeños no son recomendables.

Figura 26. Desenvolvimiento del parámetro épsilon frente al optimizador Adam



Con ADAM se obtienen mejores resultados durante el entrenamiento cercanos en la frontera menor, es decir alrededor de 1e-6. Para valores grandes de épsilon le toma mucho tiempo aprender y se mantiene constante en un valor muy bajo, por ejemplo, valores de épsilon 0.1 y 0.01.

La primera conclusión que se obtiene en base a este primer estudio es la factibilidad en usar los valores de hiperparámetro del modelo genérico. Tasa de aprendizaje 0.0001, aunque se puede ir cambiando el valor de decaimiento de tasa de aprendizaje y el valor de épsilon. Por otro lado, en cuanto a los resultados visuales, secuencias de predicción, ya se observó presentaba un desvanecimiento en los contornos de las celdas.

4.2 Análisis del modelo teniendo en cuenta el número de capas y filtros

Inicialmente el modelo contaba con dos capas de entrada en el codificador, tres capas centrales y dos capas al final en el decodificador. Primero se procedió aumentando 1 capa en

la zona central, donde se tiene en cuenta el aprendizaje temporal con las capas LSTM. Se esperaba que los resultados tengan mejor resolución espacial y temporal en las últimas salidas.

Figura 27. Resultados comparados, secuencia real y predicha

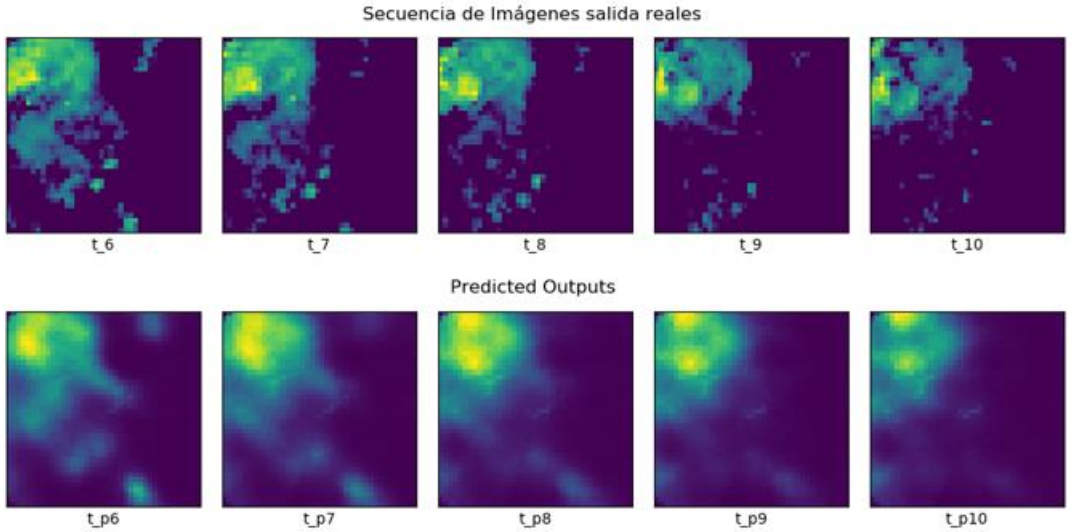
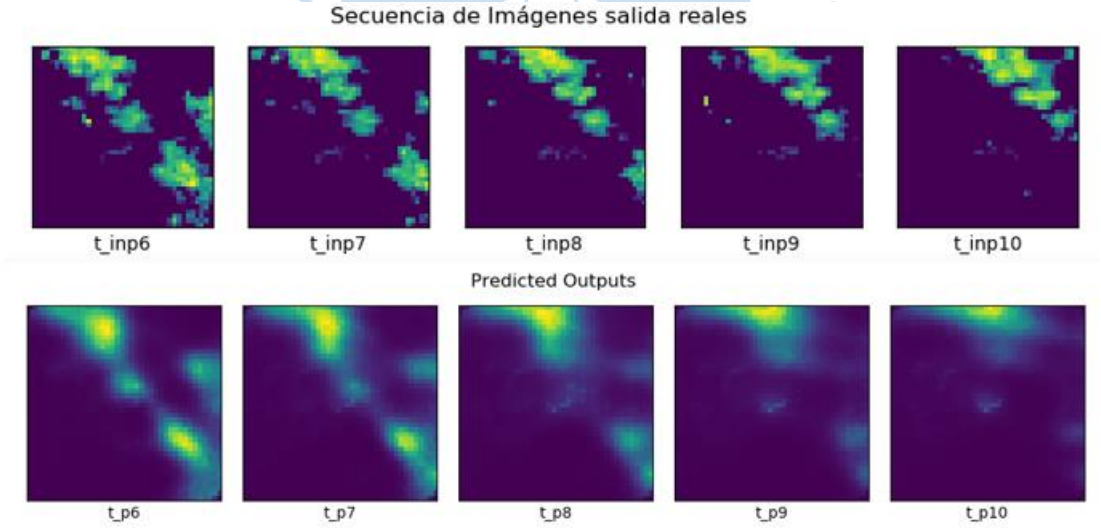


Figura 28. Resultados comparados real y predicha, respecto a otra secuencia seleccionada al azar



Ambos resultados se obtuvieron con valores de lr = 1e-4, decay-rate = 0.00075 y eps = 0.000075, se puede observar que hay una mejor resolución en las últimas predicciones, de alguna manera las capas LSTM han logrado captar más información temporal y lograr mejorar el resultado. Sin embargo, la segunda secuencia, pierde datos en las últimas predicciones de la secuencia.

Es posible disminuir las capas en una red neuronal cuando no se obtienen buenos resultados. Sin embargo, en el ejemplo anterior, aumentando una capa en la sección central se logró mejorar el aprendizaje temporal. Aunque la reconstrucción espacial fallaba debido al desvanecimiento en las fronteras de la celda de lluvia.

Por ello, se decidió quitar una capa en los extremos de la red neuronal. Una capa convolucional a la entrada y una capa deconvolucional en la salida. Además, se experimentó añadiendo una capa LSTM más en la sección central, en total cinco capas centrales. Se cambió también la dimensión de la ventana de kernel de (5, 5). Finalmente, los resultados se muestran a continuación (**Figura 29** y **Figura 30**).

Figura 29. Resultados obtenidos de una nueva modificación en las capas

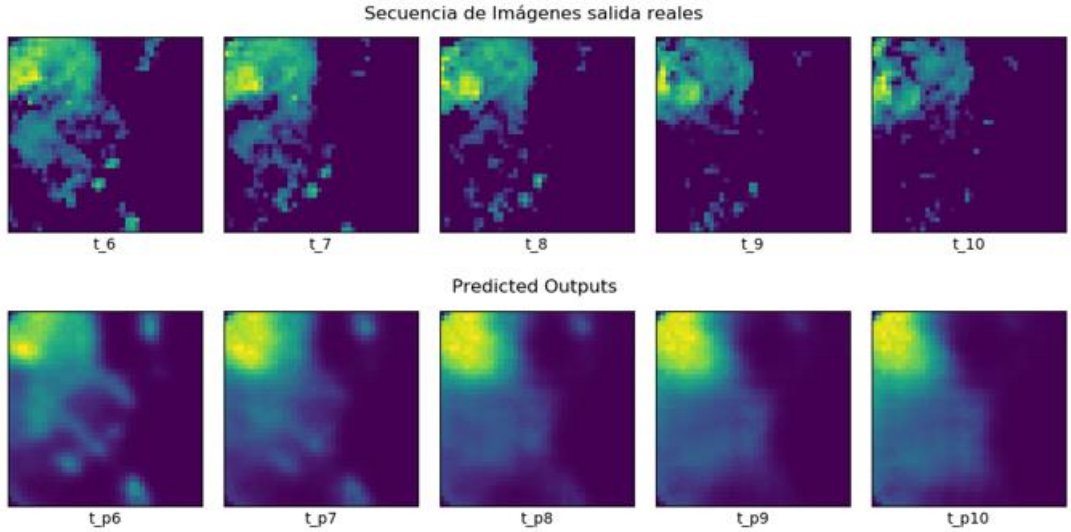
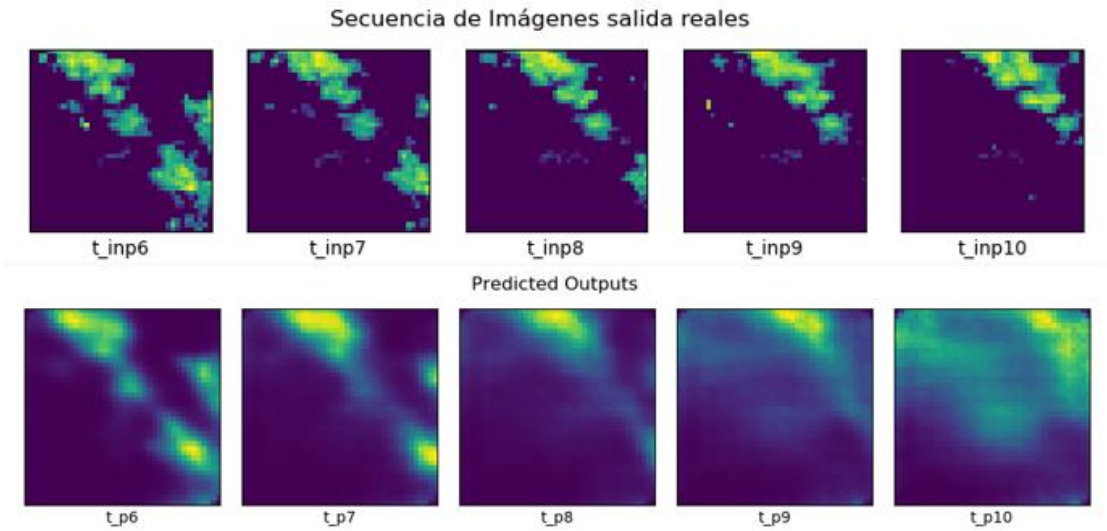


Figura 30. Resultados obtenidos de una nueva modificación para otra secuencia



Los resultados obtenidos, hacen notar que hay una mejor predicción temporal en la segunda secuencia (**Figura 30**). Que representa mejor la celda de lluvia en sus fronteras, sobre todo en los cuadros iniciales de predicción. No pasa lo mismo con la primera secuencia que desvanece los bordes superiores de la celda de lluvia en las predicciones finales. Aunque también logra mejorar la resolución del cúmulo de lluvia en la parte superior y que desaparece en t_p9.

Tal parece que aumentar el número de capas en la sección central del modelo añadiendo capas LSTM, la red aprende mejor la relación temporal entre un cuadro y sus

consecutivos. Por otro lado, disminuir las capas externas convolucionales no mejoran mucho el aprendizaje espacial y resolución de los píxeles. Sin embargo, el gradiente de intensidades en los píxeles tiene variabilidad grande y ello ocasiona mejore un poco más la definición de los datos de predicción para poder diferenciar en los cuadros la lluvia de lo que no lo es.

Es así, que entonces se decidió aumentar la cantidad de capas en el módulo central del modelo a un total de 7 capas, manteniendo las dos últimas de este módulo con funciones de activación RELU y SIGMOID respectivamente. Las capas convolucionales externas mantuvieron su forma y tamaño. Los resultados resaltan a continuación (Figura 31 y Figura 32)

Figura 31. Resultados obtenidos secuencia real y predicha, luego de aumentar el número de capas

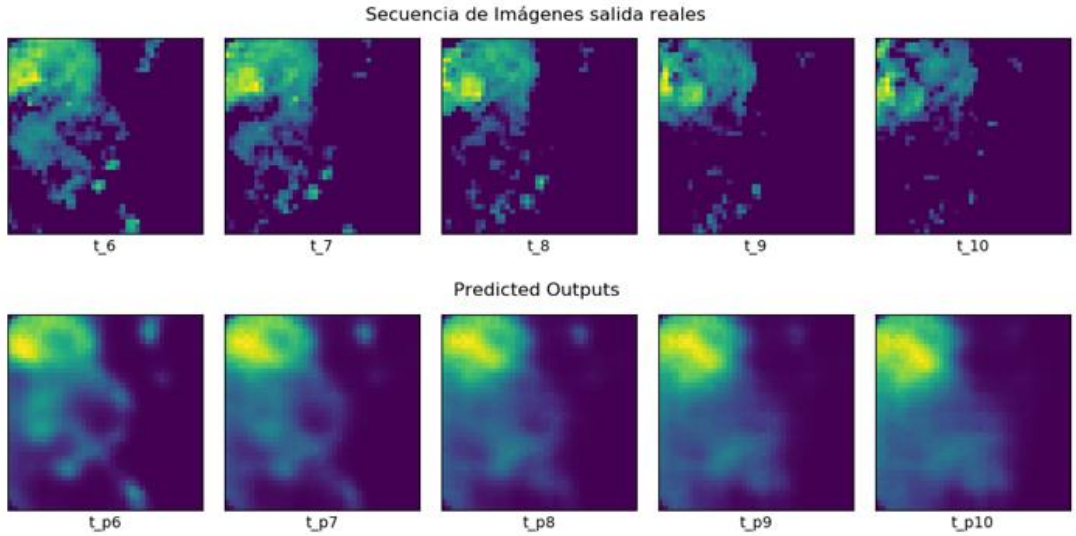
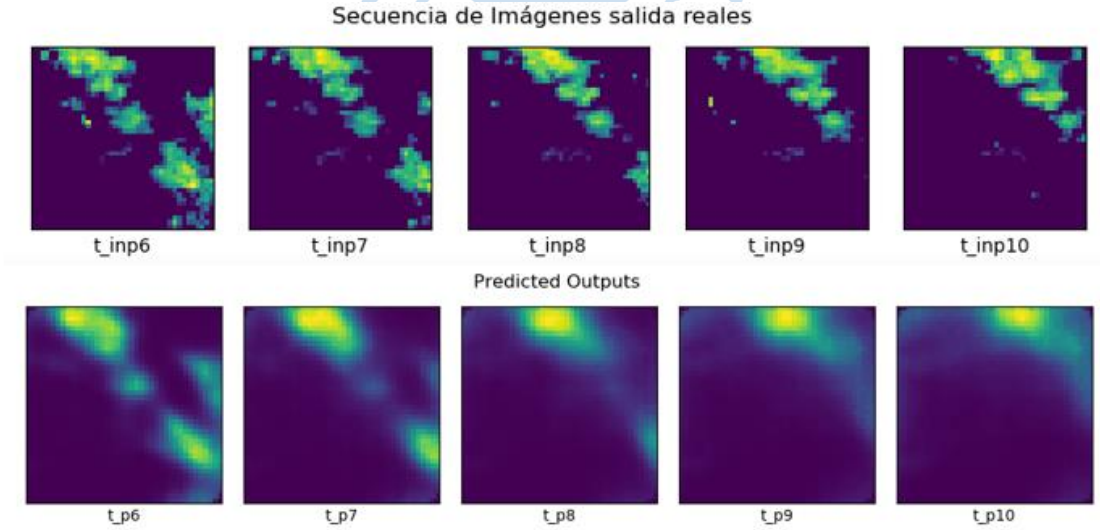


Figura 32. Resultados obtenidos secuencia real y predicha para otra secuencia, luego de aumentar el número de capas

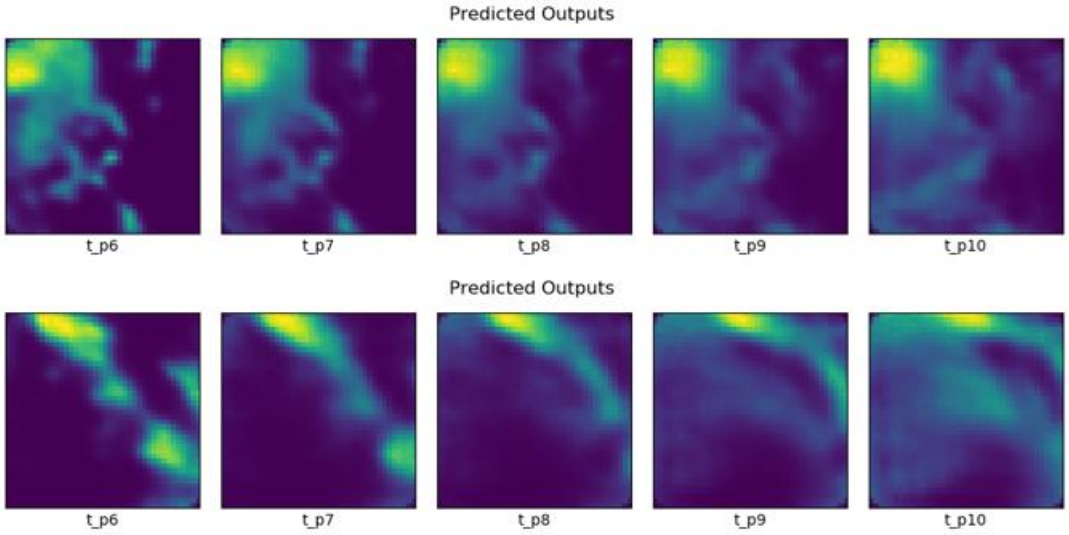


En este caso, se puede apreciar que la correlación temporal mejora y las salidas predichas en la segunda secuencia son mejores. Sin embargo, la resolución espacial empeora. Hay mucho desvanecimiento de valor pixeles en los bordes de las celdas de lluvia.

Se intentó probar disminuir el tamaño de *stride* (paso) a un valor de 2 unidades, de tal manera que la red pueda captar mayor información de los píxeles de los datos de precipitación y lograr mejorar la definición de los bordes en las salidas.

Los resultados empobrecieron aún más las predicciones y se presentó el desvanecimiento de gradiente en las últimas salidas. Tal parece que al disminuir el tamaño del paso en consecuencia la red debe analizar más patrones en los datos y ello genera un sobre entrenamiento en los pesos del modelo. Los resultados a continuación (Figura 33).

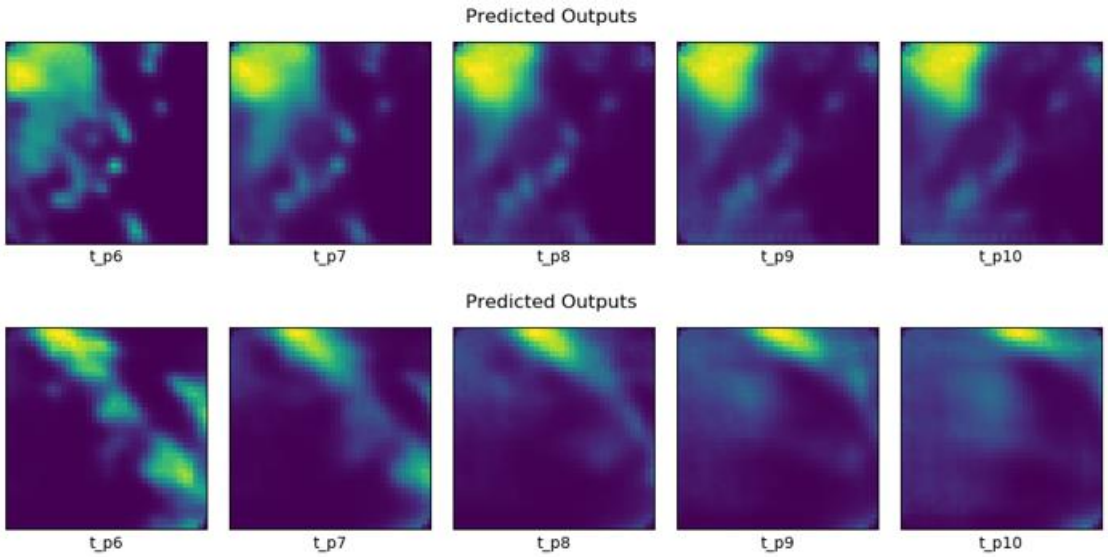
Figura 33. Resultados obtenidos, predicciones, luego de realizar modificaciones en los hiperparámetros



La segunda secuencia, secuencia inferior en la Figura 33, prácticamente pierde toda su información respecto a los bordes de la celda de lluvia en los últimos cuadros predichos, ni siquiera se logra mantener la forma de la propia celda de lluvia. Lo rescatable es que el primer cuadro de la predicción está mejor definido frente al de los ejemplos anteriores.

Otra prueba consistió en disminuir la cantidad de capas LSTM centrales a seis solamente. Y mantener el tamaño en los extremos de la red en su conjunto. De igual manera el valor del *stride* (paso) se mantuvo en 2, para evaluar si produciría un cambio al disminuir la salida del modelo. Los resultados se presentan a continuación (Figura 34).

Figura 34. Resultados obtenidos, predicciones, luego de disminuir la cantidad de capas centrales a solamente 6



Al igual que en el caso anterior se comprobó que se pierde información en los cuadros finales de la secuencia. Aunque nuevamente el desvanecimiento en el primer cuadro desaparece y se encuentra con una celda de lluvia más definida.

4.3 Solución frente al desvanecimiento en los contornos de celdas de precipitación

En el apartado 4.1 se sostuvo la métrica de *Accuracy* para ver la performance del modelo. Sin embargo, la literatura manifiesta que esta métrica no tiene un buen comportamiento para algunos modelos convolucionales.

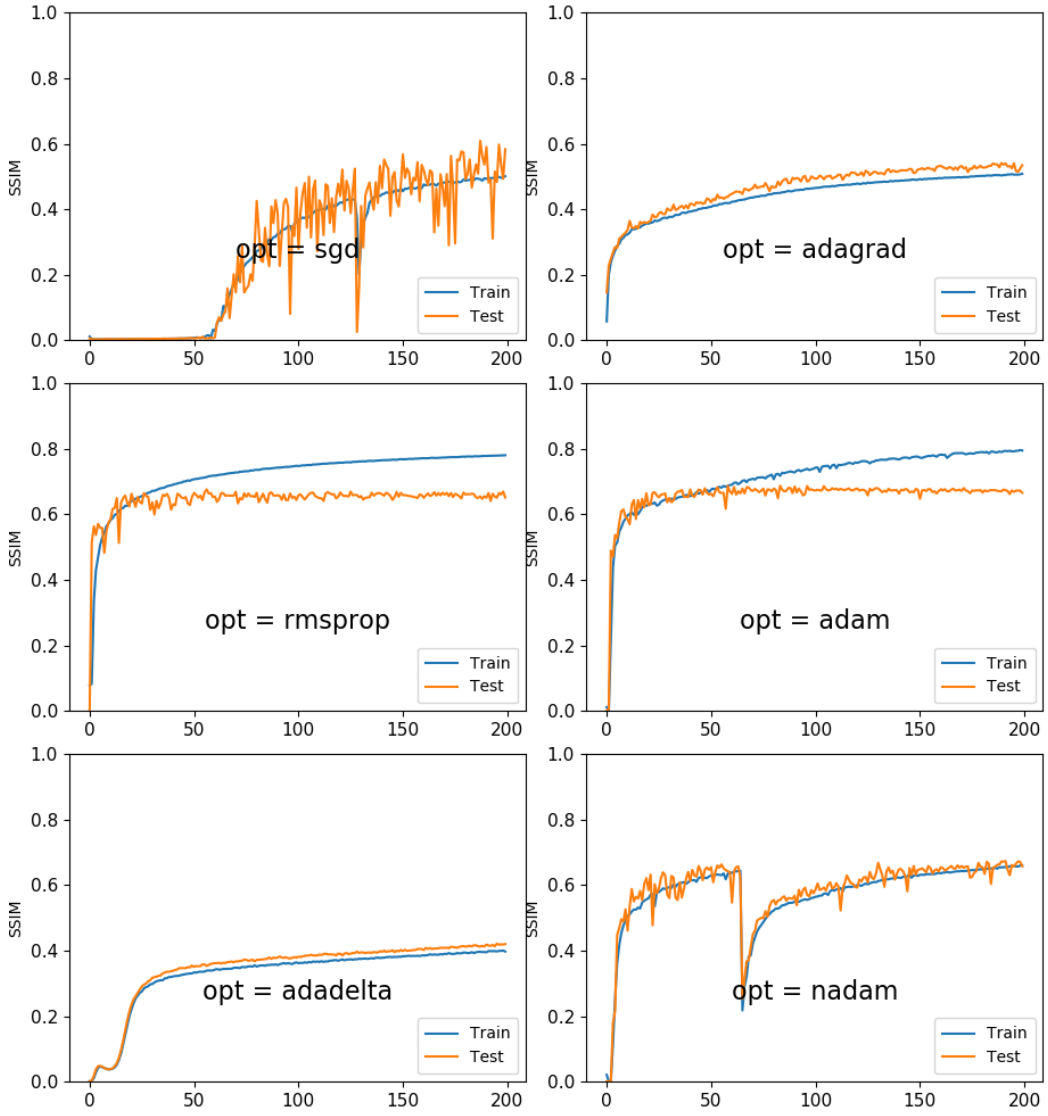
Nuestros ejemplos y gráficas en el apartado anterior (Figura 23 y Figura 24) debidos a los tipos de optimizadores señalaban un rápido alcance a la convergencia en pocas épocas. Incluso Adam mostraba que en las 20 primeras épocas se llegaba a la convergencia y era extraño que no presentara la forma característica de un proceso de aprendizaje, por el contrario, Adadelta mostraba la forma característica de una curva de aprendizaje, pero los resultados visuales de los cuadros predichos eran realmente pobres. De alguna manera, los resultados en gráficas se contraponían frente a los resultados en los cuadros predichos.

Entonces se decidió por cambiar la métrica de exactitud por dos métricas relacionadas a modelos que usan imágenes como datos de entrenamiento. SSIM y PSNR evaluarán los cuadros de secuencia reales y predichos píxel a píxel, están muy bien implementados en el procesamiento de imágenes. Además, se encontró que los resultados mostrados en gráficas corresponden con lo observado en las imágenes de predicción. Anteriormente usando la métrica *accuracy* no había ninguna correspondencia, pero fue útil para minimizar el rango de tasa de aprendizaje.

Cabe resaltar que la métrica principal a implementar es SSIM, es por ello que se usará más en las gráficas que se mostrarán en las siguientes pruebas. PSNR se ha incorporado cómo

métrica de respaldo. En la Figura 35 se presenta la evolución de SSIM respecto a las épocas o iteraciones de entrenamiento.

Figura 35. Desenvolvimiento de la métrica de SSIM para diferentes optimizadores



Como se aprecia, los optimizadores que alcanzan un mayor valor de la métrica SSIM son *Rmsprop* y *Adam* los cuales alcanzan un valor cercano al 80% se similitud entre la comparativa de datos reales y de predicción del modelo, ambos convergen a un valor muy alto. Por otro lado, *SGD* presenta un comportamiento muy oscilatorio y comienza a converger luego de 60 épocas de entrenamiento y además no alcanza un valor tan alto en las 200 iteraciones. El optimizador *Nadam* por su parte presenta un comportamiento anómalo al disminuir y volver a aumentar el score de la métrica SSIM.

A continuación, se muestran los resultados luego de aplicar sobre una secuencia de cuadros real (**Figura 36**), para 300 épocas de entrenamiento, para los optimizadores RMSprop (**Figura 37**) y Adam (**Figura 38**).

Figura 36. Secuencia real, representa una buena distribución de celdas de lluvia

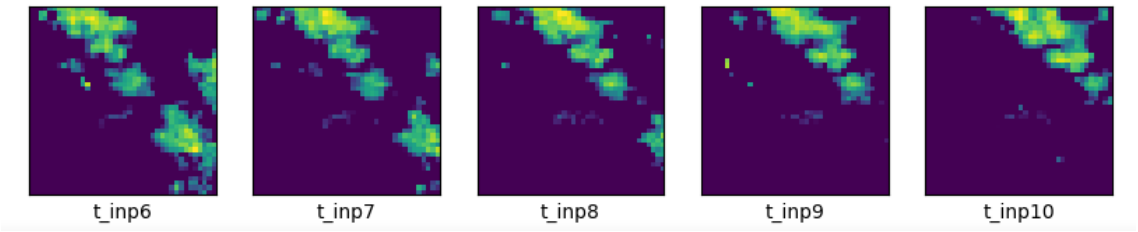


Figura 37. Secuencia predicha por el modelo usando el optimizador RMSprop

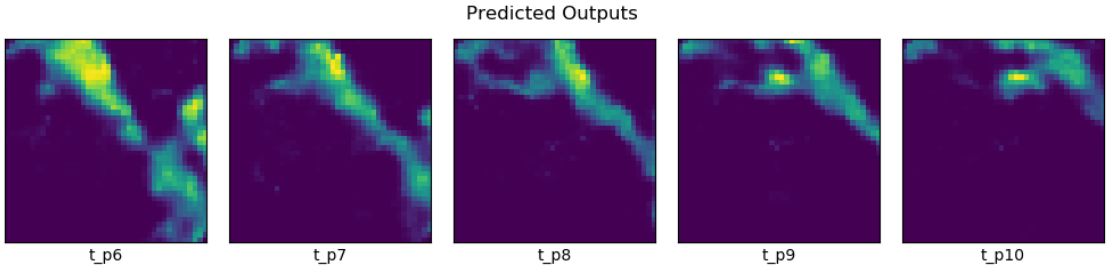
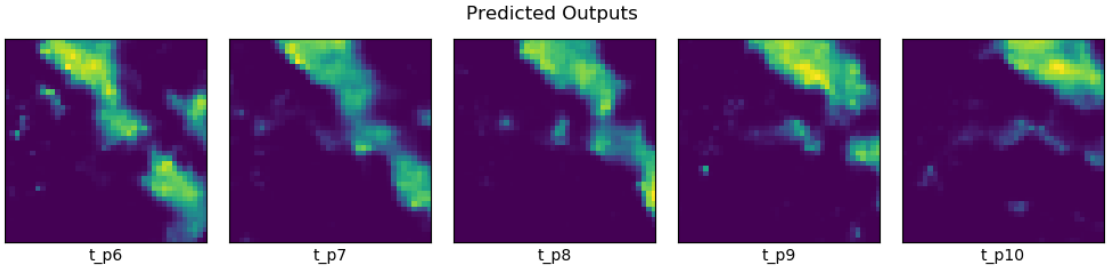


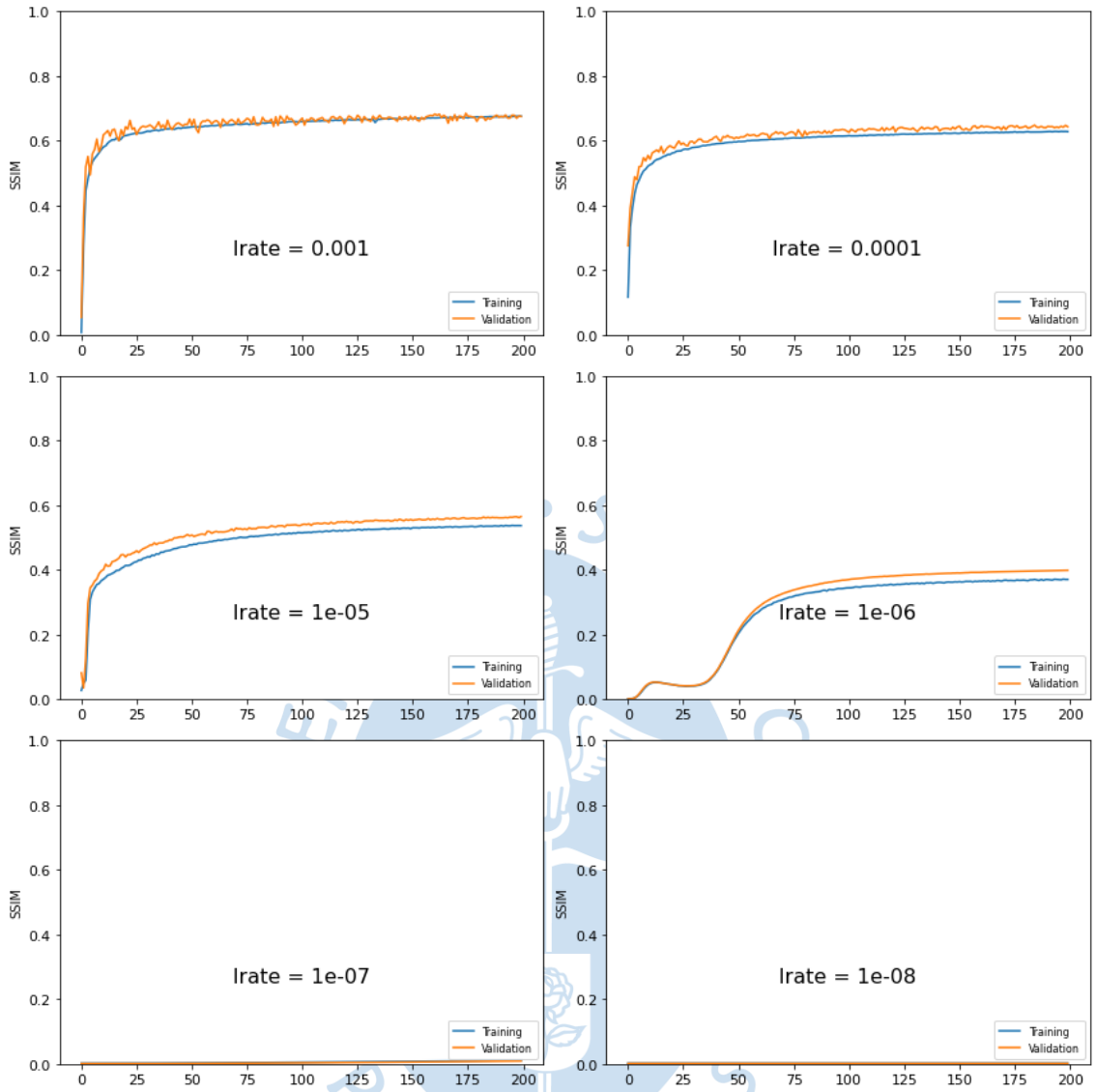
Figura 38. Secuencia predicha por el modelo usando el optimizador Adam



Los resultados visuales de las imágenes anteriores son más que suficientes para seguir eligiendo el optimizador Adam, pero ahora con una taza de aprendizaje de 1e-3, dado que esa fue la taza de aprendizaje con la que se entrenó el modelo tanto para RMSprop como Adam. Otra razón, aunque no tan notoria, para elegir Adam es lo que resulta en las gráficas (Figura 35). Aunque no se pueda ver en las imágenes, se encontró que para las 300 épocas de entrenamiento la métrica SSIM tiende al 81% mientras que RMSprop tiene un valor de SSIM de 79%. Adam es más rápido, lo que requerirá menos entrenamiento.

Es de notar que las gráficas que usan la métrica SSIM mostraron mejores aspectos cuantitativos para elegir entre un optimizador y otro. A continuación, se muestra las gráficas que corresponde a los valores de taza de aprendizaje entre 1e-3 y 1e-8. (**Figura 39**) Y nuevamente se corrobora que las tazas entre 1e-3 y 1e-4 alcanzan mejores valores de entrenamiento, además de una convergencia.

Figura 39. Diferentes valores de métrica SSIM usando el optimizador Adam

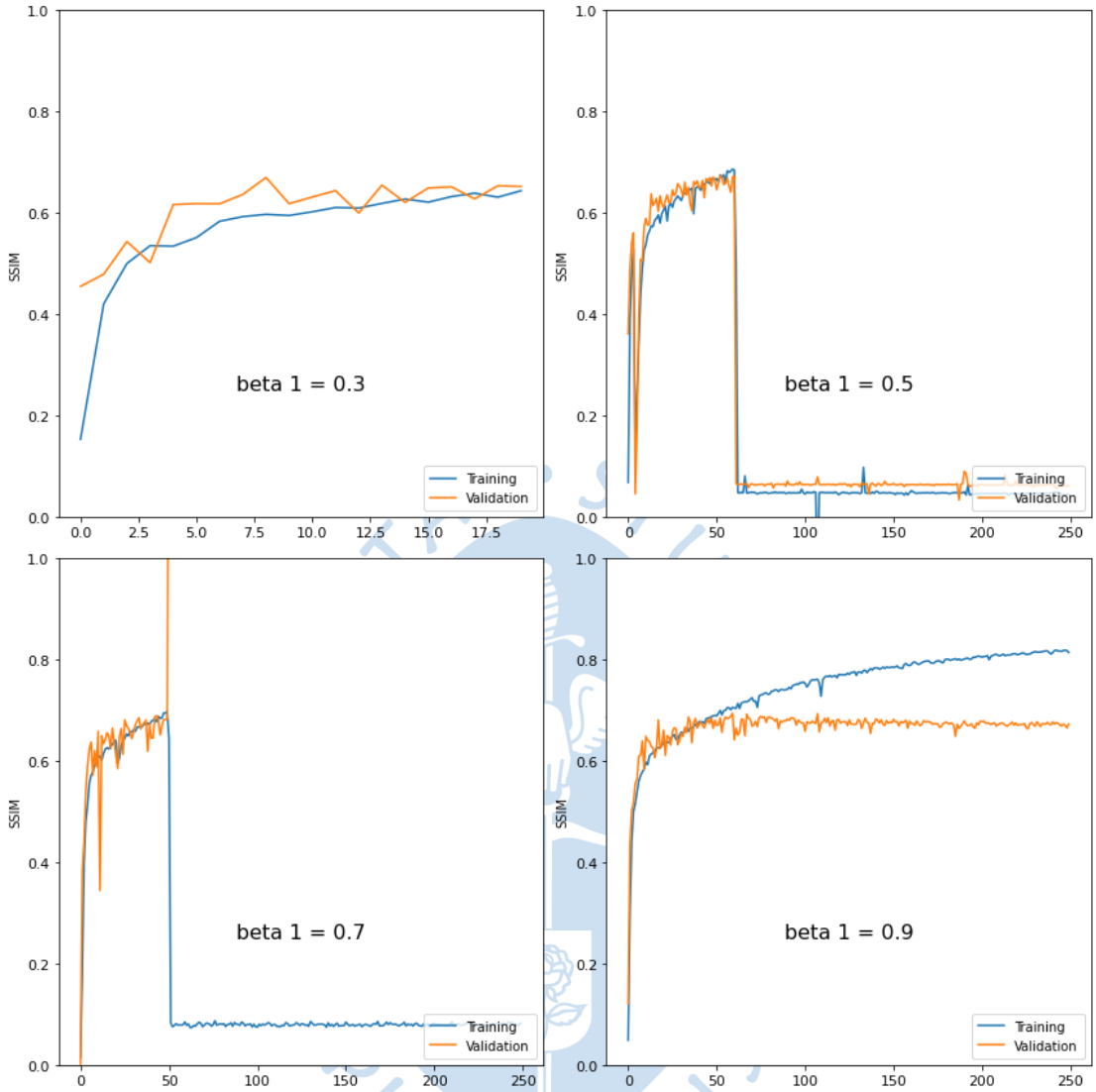


El mejor aprendizaje y convergencia a un valor alto se encuentra con un valor de taza de aprendizaje de 1e-3 o 1e-4. Valores más pequeños señalan un lento aprendizaje o en el peor de los casos jamás aprende. Aun así, el valor que mejor entrena y en menos tiempo, es la tasa de aprendizaje de 1e-3. Finalmente, ese será nuestro valor de tasa de aprendizaje.

Además, Se ha logrado solucionar el problema de desvanecimiento de píxeles en los contornos de las celdas de lluvia, en los resultados de los cuadros de predicción. Hay una mejor consistencia en la formación de cada cúmulo o celda de lluvia.

Otro hiperparámetro del optimizador Adam corresponde a los momentos, se conoce como b1 y b2 en Tensorflow. Se comprobó además que el valor de b1 puede mejorar el entrenamiento. B1 se encuentra entre 0 y 1, donde su valor por defecto en la librería de Tensorflow es 0.9. A continuación, presentamos las gráficas obtenidas (**Figura 40**).

Figura 40. Pruebas diferentes variando los valores del hiperparámetro b1 en el optimizador Adam



Queda muy bien establecido el valor por defecto, 0.9. Aunque el valor de 0.3 muestra un comportamiento convergente pero muy lento su aprendizaje en el tiempo. El valor de SSIM respectivo al entrenamiento tiende al 70%, mientras que para el valor de b1 = 0.9, se encuentra alcanzando el 80%.

Nuestros resultados de predicción en base a todos los ajustes del modelo y además de los hiperparámetros nos ha llevado al resultado de usar el optimizador ADAM y con tasa de aprendizaje 1e-3, en nuestro modelo. Se cumple además que nuestra tasa de aprendizaje se encuentra en el rango [5e-5, 2e-3] tal y como lo muestra la Figura 18 al inicio de este capítulo.

4.4 Complementos finales para el modelo

Hasta este punto ya se han encontrado los hiperparámetros que ajustan mejor el modelo de predicción de celdas de lluvia. Sin embargo, como paso final se requiere aplicar una regularización durante el entrenamiento. En la Figura 35, en la prueba de los

optimizadores, puede apreciarse que las curvas de entrenamiento y validación empiezan casi juntas y a medida que avanzan en el entrenamiento se separan.

La parte del entrenamiento sigue subiendo a medida que aumentan el número de épocas, no pasa lo mismo con la parte de validación, la cual llega a un punto en el cual el valor de SSIM se mantiene constante. Esto puede suponer dos cosas; nuestro modelo se está sobreajustando solo a los datos de entrenamiento. O es que los datos de validación no suponen un desafío para el modelo, y el modelo ha empezado a predecir correctamente los cuadros de las secuencias.

En cuanto a la segunda proposición, del párrafo anterior, la literatura establece usar una técnica denominada validación cruzada. En este caso, ya no se establece una única división inicial en el conjunto de datos (80% entrenamiento y 20% para validación). Lo que propone la validación cruzada es fijar los porcentajes de división de datos de entrenamiento y pruebas. Posteriormente el algoritmo dividirá aleatoriamente en todo el conjunto total tantas veces como el número de veces que se va a entrenar el modelo.

La gran ventaja de aplicar la validación cruzada, es importante ya que podrá escoger aleatoriamente los datos y sobre todo para la parte de validación. Ya que el modelo aprenderá de nuevos valores. Sin embargo, implica un elevado coste computacional, ya que el algoritmo trabaja en base a la cantidad de veces que va seleccionando particiones. Para este caso solo se ha realizado 6 validaciones cruzadas.

Ahora en cuanto al primer punto acerca del sobreajuste. Es necesario utilizar una regularización, básicamente se le conoce como *dropout* o abandono de neuronas. En el entorno de Tensorflow se presentan como capas, tal y como las convolucionales, es por ello que se aprovecha su forma para poder añadirlas dentro de la arquitectura.

Puede apreciarse en la **Figura 41** y **Figura 42** para la métrica de SSIM, es evidente como la parte de validación alcanza cierto valor y luego prácticamente es constante. No hay aprendizaje. Por otro lado, la curva de pérdida de validación comienza a subir en lugar de disminuir como lo hace la curva de pérdida entrenamiento, ello confirma el sobreajuste del modelo.

Figura 41. Métrica de SSIM para el modelo sin agregar capas dropout

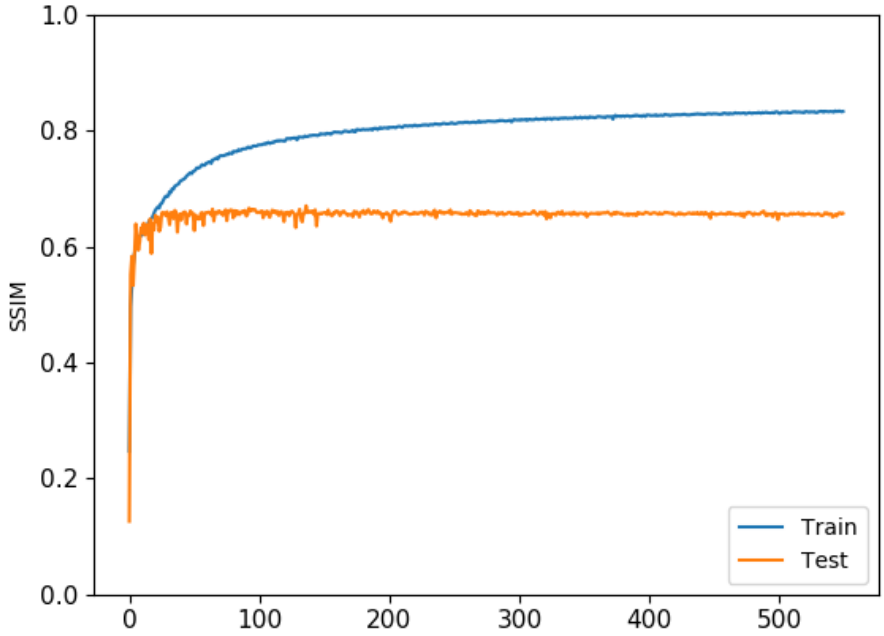
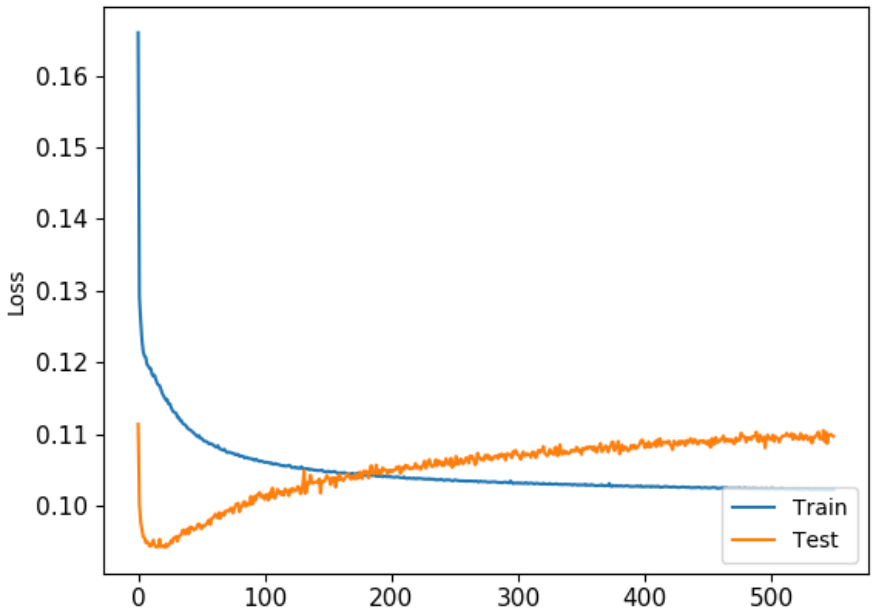
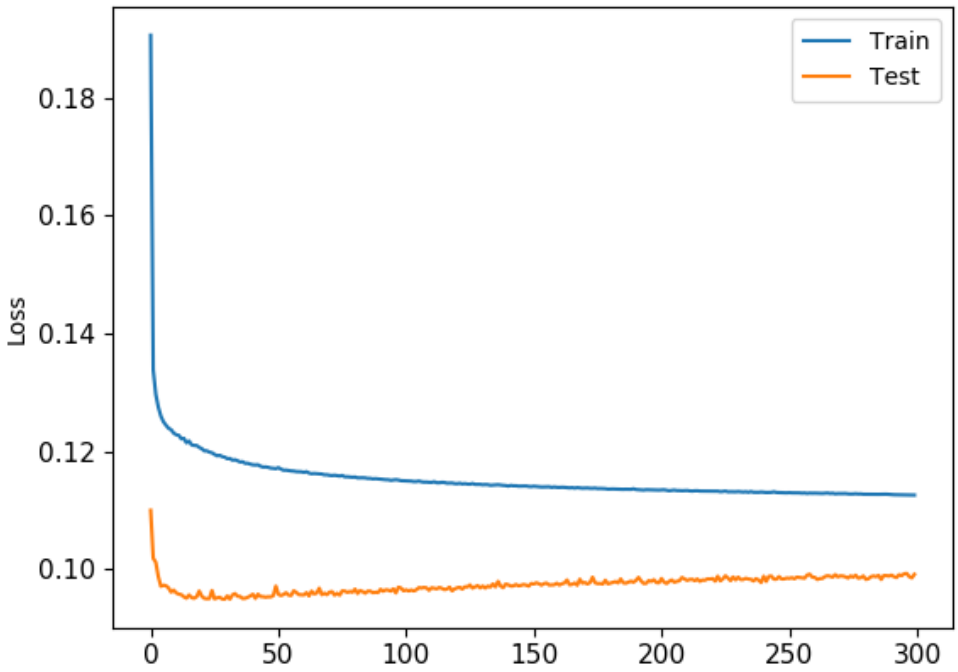


Figura 42. Variación de la métrica *Loss* durante el entrenamiento sin capas *dropout*



La literatura propone empezar usando un valor de *dropout* de 0.2, y así es como se planteó en este proyecto. También se probó con el valor de *dropout* de 0.5 (**Figura 43**), sin embargo, un valor bajo de *dropout* consiguió mejores resultados.

Figura 43. Valor de la métrica *Loss* usando capas *dropout* con un valor de 0.5



De esta manera es como se han ajustado los hiperparámetros del modelo final. Desde un inicio se plantea como es que se han modificado los distintos hiperparámetros para llegar a la mejor aproximación dada por el modelo. Además de los resultados visibles, secuencias de predicción. También se han mostrado gráficas que corroboren la evolución de los hiperparámetros.

Finalmente, nuestra arquitectura de red autocodificador variacional quedará de la siguiente manera:

```
seq = Sequential()
seq.add(TimeDistributed(Conv2D(64, (11, 11), strides=4, padding="same"),
    batch_input_shape=(None, None, 40, 40, 1)))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same")))
seq.add(LayerNormalization())
seq.add(Dropout(0.2))
# # # # #
seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(32, (3, 3), padding="relu", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(64, (3, 3), padding="sigmoid",
return_sequences=True))
seq.add(LayerNormalization())
seq.add(Dropout(0.2))
# # # # #
seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2,
    padding="same")))
```

```
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2DTranspose(64, (11, 11), strides=4,
    padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid",
    padding="same")))

seq.compile(loss=Tensorflow.Keras.losses.binary_crossentropy,
    optimizer=Tensorflow.Keras.optimizers.Adam(learning_rate=1e-3),
    metrics=[SSIM, PSNR], run_eagerly=True)
```

Y los resultados visibles se muestran en la **Figura 44** hasta la **Figura 46**. Primero se presenta la secuencia de salida real, y debajo la secuencia de predicción que se obtiene del modelo.

Figura 44. Resultados obtenidos con el optimizador Adam para una primera secuencia luego de ajustar el modelo

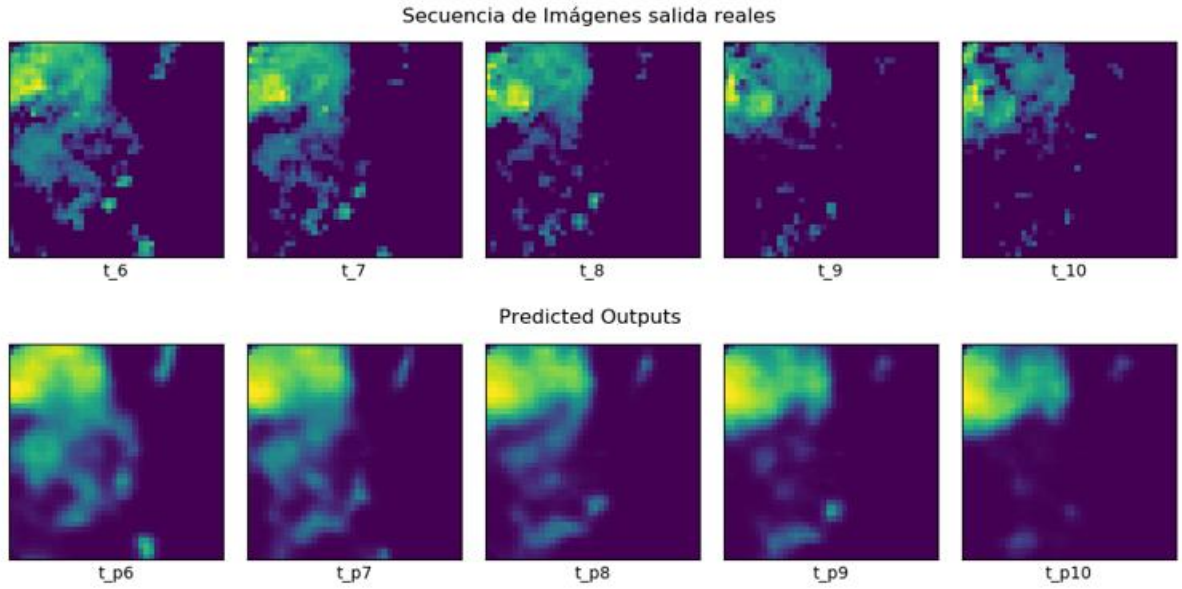


Figura 45. Resultados obtenidos para una primera secuencia luego de ajustar el modelo, poca precipitación

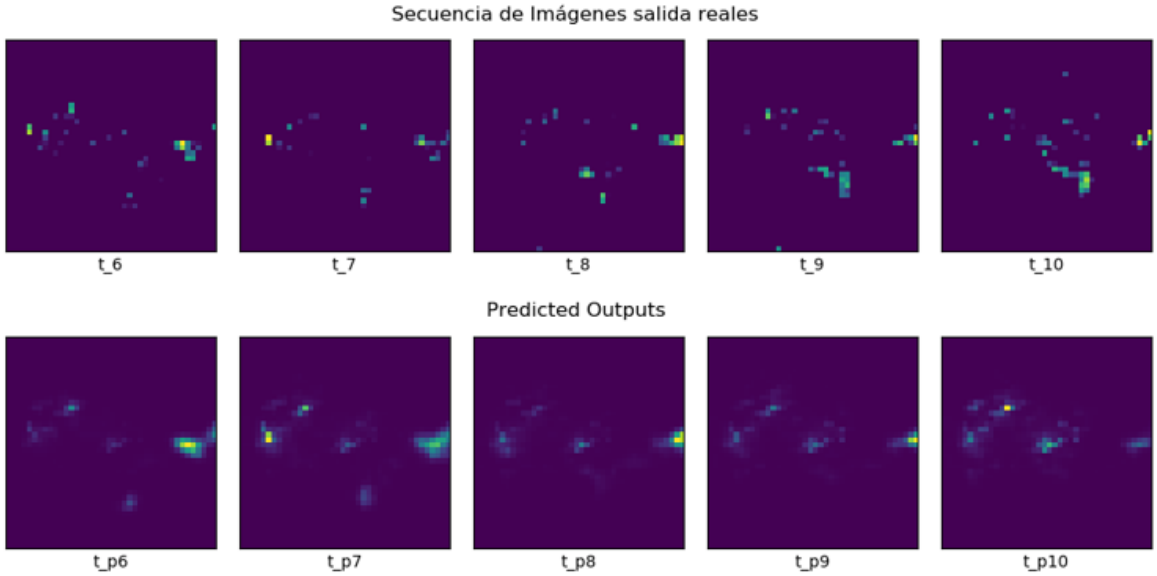
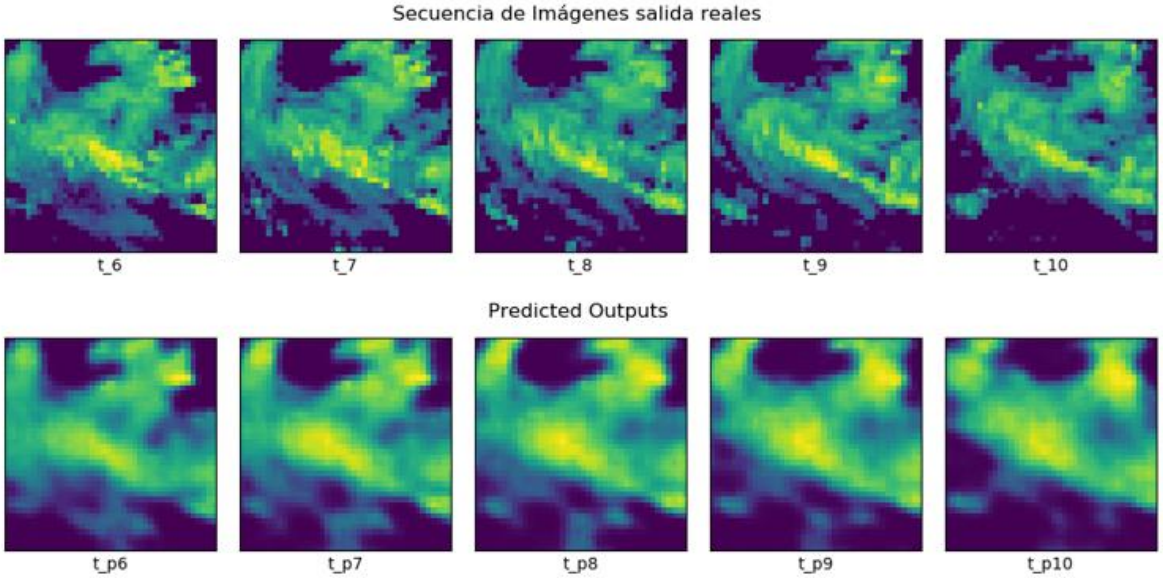


Figura 46. Resultados obtenidos para una primera secuencia luego de ajustar el modelo, alta precipitación



Los resultados visuales son más que suficientes para aprobar el modelo ajustado, que ha logrado captar tanto información temporal como espacial de los datos de precipitación de Radar de lluvia. Hay un problema en cuanto al leve desvanecimiento en los contornos de las celdas de precipitación, pero ello es propio de las redes convolucionales del variacional autoencoder. Esto y otros puntos se entenderán mejor en el siguiente capítulo.

UNIVERSITAS STUDIORUM PIURENSIS

Capítulo 5
Resultados y discusiones

A lo largo del capítulo 3 y 4 se ha explicado todo el proceso que se ha seguido en la construcción de una red neuronal: preparación de datos, elección de parámetros, elección de hiperparámetros y modificaciones en la estructura externa del modelo de red.

Este último capítulo dará a conocer las razones por que se eligieron ciertos valores de hiperparámetros y por qué otros no fueron lo suficientemente buenos, a modo de conclusiones para esta tesis.

Es interesante notar que una red neuronal es como una caja negra con muchos diales móviles que son ajustables a las necesidades. Una caja negra a la que de ninguna manera se piensa acceder a ella, debido a la complejidad en su interior y a la gran cantidad de variables. Lo que se puede hacer es solo girar esos distintos diales y combinarlos de la mejor manera para obtener los resultados que se desean. Nuestros diales son los distintos hiperparámetros que se han estudiado.

Y aunque la literatura no tiene nada generalizado, ni siquiera para un caso en particular. Es de notar que seguir un camino a través de la prueba y error no basta. Aquí también se sigue la lógica y ciertos conocimientos en base a otros investigadores y personas que ya han estado en el camino de combinar los diales de esa caja negra llamada red neuronal.

5.1 Número de capas

En el capítulo 3 se corroboró que aumentar el número de capas en la parte central de autocodificador variacional, lograba abstraer mejor características temporales de los datos de entrenamiento. Se aumentaron a 5 capas convolucionales LSTM en base a la cantidad de los elementos por cada secuencia, de esa manera se pensaba que pueda relacionarse cada capa con cada elemento.

Además, se disminuyó una capa en el codificador y decodificador, se creía que el modelo tenía muchas características para aprender, y podría caer en sobreajuste. Ello además demandaba más tiempo de entrenamiento. Finalmente se comprobó que era un experimento fallido, y se volvió a forma inicial. Dos capas convolucionales en el decodificador y dos capas en el decodificador.

Por otro lado, los resultados obtenidos, usando cinco capas convolucionales LSTM en el bloque central de autocodificador, no variaron mucho luego de que se hizo la prueba con solo tres capas y una taza de aprendizaje de 1e-3 como se estudió en el capítulo anterior. Así que se decidió por usar solo tres capas en la zona central, ello para no elevar el coste computacional y lograr más épocas de entrenamiento.

5.2 Cantidad de unidades por cada capa

Las unidades solo están presentes para capas de entrada y salida, se puede apreciar entre paréntesis en el código, luego de invocar la red. Puede ser 32, 64, 128 múltiplos de 2. En un principio se tenía 128 unidades en los extremos del autocodificador. Posteriormente se cambiaron esos valores a 64 en las capas externas. Esto porque 128 unidades, implicaba mayor coste computacional y por otro lado el sobreajuste.

El sobreajuste puede ocurrir por la gran cantidad de características que se le dan al modelo para que aprenda. La literatura recomienda empezar a usar valores a partir de 32 en adelante. Sin embargo, se decidió dejarlo en 64 unidades, ya que proporcionaba buenos resultados sin incurrir en tomarse demasiado tiempo para la ejecución del entrenamiento.

5.3 Funciones de activación

Nuestro modelo final, tiene en su parte central tres capas convolucionales LSTM, además puede apreciarse que también se han escrito funciones de activación en las dos últimas capas. Durante la parte experimental, se han probado diferentes funciones de activación y arreglos en la parte central, las funciones y el arreglo que han logrado mejores resultados se expone a continuación.

Una capa convolucional LSTM tiene una función de activación RELU, ello para lograr que las neuronas en la capa permitan concentrar mejor las características de los datos en secuencia, de alguna manera permite captar mejor la temporalidad en los datos.

Por otro lado, la última capa del bloque central usa una función de activación SIGMOID. La función de activación sigmoide permite dar una salida del bloque central en un rango de valores de 0 y 1. Ya que nuestros datos se encuentran en ese rango normalizado.

Existe una función de activación llamada SOFTMAX, la cual también se experimentó en el entrenamiento. Pero los resultados que mostraban eran muy por debajo de lo que mostraban las otras funciones de activación ya mencionadas.

5.4 Función de optimización

ADAM sin duda alguna ha logrado sobrepasar a los demás optimizadores, RMSprop también es un muy buen optimizador también. Aunque para el tipo de datos que se usan aquí, no constituye el optimizador a elegir.

La literatura aconseja comenzar con el optimizador ADAM, ya que es el más reciente que se ha encontrado y que se usa en la mayoría de los casos. Aquí también se intentó probar con otros optimizadores como SGD, ADADELTA y el propio RMSprop.

SGD lograba mantener una correlación temporal de las celdas de lluvia, sin embargo, la espacialidad perdía totalmente en los valores de píxeles. Podría haber mejorado la resolución de la imagen con más iteraciones, pero era algo que no se quería para este trabajo. Sin embargo, SGD superó a ADADELTA.

ADADELTA por su parte, es el que peores resultados ha producido en la experimentación. No lograba mantener ni la espacialidad ni la temporalidad de los datos. Solo se apreciaba una gran mancha en toda la cuadrícula de 40x40 del tamaño del dato.

RMSprop mejoró bastante en comparación a SGD y ADADELTA. Los resultados obtenidos eran de buena resolución y también había un aprendizaje temporal. Aunque en comparación al optimizador ADAM, RMSprop requería más tiempo de entrenamiento, lo cual hacía se encuentre por debajo del optimizador ADAM.

5.5 Función de pérdida

Para las funciones de pérdida en el modelo también solo se intentaron con dos de las muchas funciones de pérdida. Mean Square Error (MSE) y la perdida de entropía cruzada. Amabas en base a la literatura, y además son las más usadas al menos al tratar con imágenes como datos.

Se comenzó usando la función de pérdida de MSE, era lo que establecía el modelo genérico visto en el capítulo 3. Desafortunadamente los resultados en gráficas de la pérdida descendían rápidamente y se mantenían constante en el valor más bajo que calculaba, y constante por casi todas las épocas de entrenamiento que se asignaba a la ejecución del algoritmo.

En base a ello, se buscó en diferentes foros las soluciones frente al problema explicado con MSE. La mayoría de las recomendaciones apuntaba a usar la pérdida de entropía cruzada, pues esta función trabaja en un rango de valores de 0 y 1. Nuestros datos justamente se encontraban en ese rango de valore. Se intentó con la nueva función de pérdida y se obtuvieron mejores resultados para la curva de pérdida, algo más aceptable y realista.

También se probó con la función de pérdida categórica. Esta función aplica la función de entropía cruzada, pero a multi clases. Se obtuvieron resultados bastante alejados de la realidad, el valor de pérdida era menor a cero, negativo, y se mantenía constante durante todo el entrenamiento.

5.6 Cantidad de paquetes para el entrenamiento (*batch*)

Ya se han discutido el porqué de los hiperparámetros para el modelo en los ítems anteriores. Ahora se discutirá un hiperparámetro importante para la ejecución del entrenamiento del modelo.

El batch o paquete en español. Es un valor que indica al modelo una partición del total de datos y que va a tomar en cada época de entrenamiento. Durante la ejecución del entrenamiento el algoritmo no puede tomar todas las secuencias al mismo tiempo y luego entrenarlas, ello implica un poco aprendizaje de las neuronas en el modelo. Es por ello que el modelo se reparte del total de datos en pequeños grupos a los que sí puede entrenar fácilmente y captar más características para el aprendizaje.

Ahora bien, la literatura en general propone comenzar con un valor de batch de 32, generalmente también son un valor múltiplo de 2. Pero se considera además que el valor del batch puede ir disminuyendo, pues al ser más pequeño más variedad y características importantes puede aprender. Por otro lado, implica un mayor coste computacional, y si es muy pequeño el batch puede llegar al problema del sobreajuste.

Aunque no se ha seguido al pie de la letra con la literatura, aquí también se comenzó con un valor de batch de 32, pero posteriormente se intentó con un valor de *batch* de 5. Tal y como se explicó anteriormente el tiempo de ejecución sobrepasó un día para 500 épocas de entrenamiento. Mientras que con un *batch* de 32 se lograba tener resultados de las 500 épocas de entrenamiento en tan solo 3 horas.

Los beneficios conseguidos tras haber disminuido el valor del *batch* fueron la mejora del aprendizaje de los datos de entrenamiento y con ello aumentaron la resolución en los datos de predicción. Finalmente se decidió elevar un poco el valor del *batch* a 8 para compensar el tiempo de entrenamiento, reducir un tanto menos.

5.7 Regularización

Aquí otro hiperparámetro del modelo, y es que es muy importante resaltar su función. Como ya se mencionó en el capítulo anterior, nuestro modelo llegaba a sobreajustarse a los datos de entrenamiento, en ese punto no lograba buenas predicciones. Entonces y gracias a la ayuda de foros que recomendaban usar una regularización en el modelo.

Básicamente *dropout* (abandono) como su nombre lo indica abandona neuronas durante el proceso de aprendizaje. *Dropout* deja de tomar en cuenta durante el *back propagation* las neuronas que se mantienen con bajos valores en sus pesos y constantes, además que no suman al aprendizaje.

La literatura en foros recomienda usar valores de *dropout* entre 0.2 y 0.5. Siendo el valor más bajo el elegido para este modelo, pues no hace decaer en exceso el aprendizaje de

la red. Sin aplicar *dropout* se lograba llegar a 81% en la métrica de SSIM, con un *dropout* de 0.5 se alcanzó 70% de valor de SSIM y con un valor de 0.2 de *dropout* se logró alcanzar 78%.

Así es, desafortunadamente al aplicar un *dropout* se requerirá mayor tiempo de entrenamiento, pero lo ventajoso es que se evita o elimina el sobreajuste.

UNIVERSITAS STUDIORUM PIURENSIS

Conclusiones

El *machine learning* actualmente se ha convertido en una gran técnica para resolver la gran mayoría de procesos de optimización. Y sobre todo porque da soporte a la inteligencia artificial ya sea desde un caso simple como en nuestro propio hogar para encender una pequeña bombilla, o en casos de más alto nivel como generar una alarma luego de analizar en tiempo real todas las fotos de los rostros de las personas que ingresan a un aeropuerto.

Por otro lado, y en cuanto a meteorología se refiere, la literatura también manifiesta la implementación de *machine learning* para la predicción de parámetros meteorológicos. Claro está que aún los avances son un poco tempranos y por tal el rango de predicción temporal no pasa un día de predicción. Pero si se ha logrado alcanzar el *nowcasting* o predicción inmediata que permitirá dar aviso a una población frente a un desastre.

Para este proyecto de tesis no fue suficiente con datos propios de la Estación Científica Ramón Mugica, dado que el modelo necesita variabilidad de ejemplos y gran cantidad. Por ello se emplearon también datos proporcionados desde la Universidad de Cuenca, quien cuenta con el mismo tipo de radar. Además de datos facilitados por la Universidad de Sao Paolo que también usa un radar del mismo fabricante, pero en este caso consta de un radar es de banda C y sus datos proporcionados está de libre acceso (Bonnet et al., 2020a).

La parte de procesamiento de datos, programación de redes neuronales, herramientas de programación y conocimientos de aprendizaje automático. En un inicio solo se tenía un conocimiento general, así que se tuvo que profundizar en cada uno de los temas. Ello significó un mayor esfuerzo y sobre todo un gran desafío al adentrarse en el entendimiento de la denominada “caja negra” de un modelo de machine learning.

Durante la elaboración de este proyecto cabe destacar lo siguiente.

Primero. Puede parecer sencillo el implementar un modelo ya trabajado en otras investigaciones. Sin embargo, depende mucho de los datos que se utilizan, es decir; tamaño, forma, tipo y otras características. Es necesario en primer lugar reconocer los datos y luego evaluar el modelo a utilizar. Aquí se tuvo que realizar además un post-procesado de los datos para prepararlos para la entrada del modelo. Fue una ardua tarea organizar las secuencias temporales debido a que no había continuidad en todos los archivos, en la vida real no llueve

todos los días y continuamente. Las técnicas de programación ayudaron bastante para lograr reducir el tiempo de procesamiento y organización de datos.

Segundo. Durante la investigación y evaluación de modelos se probaron distintos modelos, incluso hasta se llegó estudiar implementar uno de elevado nivel computacional. Tal y como menciona la literatura acerca de probar distintos parámetros y usar el sentido común además de usar métricas como método de evaluación de modelos. Finalmente, se encontró un modelo que logró comprender mejor la forma y patrón de los datos, y significó el comienzo en la construcción en la arquitectura del modelo.

Tercero. Otro punto por considerar y relacionado al punto anterior es el tiempo de procesamiento. Los datos para el entrenamiento en esta tesis no son excesivos a diferencia de otras investigaciones, aun así, para una PC de escritorio implica un elevadísimo coste computacional y demanda un gran tiempo de entrenamiento, tan solo 10 iteraciones o épocas de entrenamiento tomaba alrededor de 3 días. Pocas iteraciones no lograban mostrar resultados visuales que permitan evaluar nuestro modelo. Google Colaboratory es una plataforma web (con suscripción gratis y de paga) que ofrece servidores en la nube y es la herramienta que facilitó el entrenamiento del modelo. Con esta nueva herramienta se logró 100 iteraciones entre 6 y 10 horas con la suscripción gratuita. Recientemente el año 2021 se integró en Latinoamérica el servicio de Google Colaboratory Pro, con un coste de 9 dólares mensuales. A este punto, el tiempo de entrenamiento para 500 épocas se redujo a tan solo entre 3 y 5 horas. Significó un gran avance.

Cuarto. Por último, acerca de los resultados obtenidos. Al final del capítulo 4 se pudo apreciar en algunos de los ejemplos de secuencias elegidas para probar el modelo, se acercan muy bien a la salida real (la secuencia real). Pero siempre estará el desvanecimiento en los bordes de cada celda y ya se explicó que es debido a las redes convolucionales. El modelo aquí presente si infiere la forma y desplazamiento de las celdas de lluvia, y se ha comprobado, muy bien en el capítulo de la parte de resultados. El plus que se buscaba era también predecir las intensidades de precipitación en cada celda, aunque de igual manera las primeras salidas predichas en cada secuencia si se asemejan a la parte real. No pasa lo mismo con las últimas imágenes en cada secuencia, se pierde un poco de información relacionado a la intensidad.

Recomendaciones

Como recomendaciones de mejora a futuro se puede mencionar.

Primero. Suele pasar que a veces el modelo converge en la pérdida hacia un mínimo local y no hacia un mínimo global, normalmente se puede hacer dos cosas, o se convierte el mínimo local a un mínimo local óptimo o en todo caso se debe guiar el entrenamiento por otra curva de descenso para que llegue al mínimo correcto de valor de pérdida. Sería óptimo poder implementar un método que permita guiar el entrenamiento por el camino correcto, y claro está que sería un gran desafío también.

Segundo. Resuelto el punto anterior. Tener una máquina con grandes capacidades de cálculo es recomendable para entrenar el modelo durante más tiempo. El modelo puede aprender más de las características de los datos. Se ha comprobado que el modelo que se ha usado, converge rápidamente pero que también al final actualiza pequeños valores de los pesos en las neuronas del modelo. Se pretende que estas pequeñas actualizaciones permitan aprender mejor los contornos de las predicciones en las celdas de lluvia., pues las redes convolucionales tienden a generar un desvanecimiento en sus predicciones.

UNIVERSITAS STUDIORUM PIURENSIS

Referencias bibliográficas

Bahuleyan, H. (2018). *Natural Language Generation with Neural Variational Models*. http://arxiv.org/abs/1808.09012

Bajaj, A. (2021). *Performance Metrics in Machine Learning [Complete Guide]*. Neptune Blog. https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide

Bonnet, S. M., Evsukoff, A., & Rodriguez, C. A. M. (2020a). *Normalized radar reflectivity factor data 2015-2019 from Ponte Nova weather radar (São Paulo – Brazil) - Harvard Dataverse*. HARVARD Dataverse. https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/ZADDNQ

Bonnet, S. M., Evsukoff, A., & Rodriguez, C. A. M. (2020b). Precipitation Nowcasting with Weather Radar Images and Deep Learning in São Paulo, Brasil. *Atmosphere 2020, Vol. 11, Page 1157*, *11*(11), 1157. https://doi.org/10.3390/ATMOS11111157

Brownlee, J. (2017). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Machine Learning Mastery. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Brownlee, J. (2017). *What is the Difference Between a Parameter and a Hyperparameter?.* Machine Learning Mastery. https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

Brownlee, J. (2019). *Una suave introducción a las redes generativas de confrontación (GAN).* Machine Learning Mastery. https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

Brownlee, J. (2020*). How to Choose Loss Functions When Training Deep Learning Neural Networks.* Machine Learning Mastery. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

Brownlee, J. (2020). *Understand the Impact of Learning Rate on Neural Network Performance.* Machine Learning Mastery. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Brownlee, J. (2021). *How to Choose an Activation Function for Deep Learning*. Machine Learning Mastery. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

Brownlee, J. (2021b). *Gradient Descent With RMSProp from Scratch.* Machine Learning Mastery. https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch/

Burns, E. (s.f.). *Aprendizaje profundo (deep learning*). ComputerWeekly. Recuperado el 28 de noviembre de 2021, de https://www.computerweekly.com/es/definicion/Aprendizaje-profundo-deep-learning

Chablani, M. (2017) *Sequence to sequence model: Introduction and concepts.* Towards Data Science. https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d

Conv2D layer. Keras. (s.f.). Recuperado el 30 de noviembre de 2021, de https://keras.io/api/layers/convolution_layers/convolution2d/

ConvLSTM2D layer. Keras. (s.f.). Recuperado el 30 de noviembre de 2021, de https://keras.io/api/layers/recurrent_layers/conv_lstm2d/

Delgado, C. (2019). *How to calculate the Structural Similarity Index (SSIM) between two images with Python*. Our Code World. https://ourcodeworld.com/articles/read/991/how-to-calculate-the-structural-similarity-index-ssim-between-two-images-with-python

Doshi, S. (2019). *Various Optimization Algorithms For Training Neural Network.* Towards Data Science. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

El clima y el tiempo promedio en todo el año en Piura. Wheather Spark. (s.f.). Recuperado el 26 de noviembre de 2021, de https://es.weatherspark.com/y/18257/Clima-promedio-en-Piura-Per%C3%BA-durante-todo-el-a%C3%B1o

El impacto del fenómeno del niño costero en la ciudad de Piura y su vida urbana. FLACSO. (2017). https://flacso.edu.ec/cambioclimatico/el-impacto-del-fenomeno-del-nino-costero-2017-fen-en-la-ciudad-de-piura-y-su-vida-urbana/

El Niño Costero, el hecho que marcó la década en Piura. El Tiempo. (2019). https://eltiempo.pe/el-nino-costero-el-hecho-que-marco-la-decada-en-piura/

Freire, E., & Silva, S. (2019). *Redes neuronales*. Medium. https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

Función de pérdida de entropía cruzada. ICHI.PRO. (2020). https://ichi.pro/es/funcion-de-perdida-de-entropia-cruzada-267783942726718

Función de pérdida en Machine Learning. SitioBigData.com. (2019). https://sitiobigdata.com/2019/12/24/funciones-comunes-de-perdida-en-el-aprendizaje-automatico/

Gnoza, N., & Barberena, M. (2018). *Estudio de factibilidad del uso de Machine Learning con múltiples fuentes de datos en el pronóstico del tiempo* [Universidad ORT Uruguay]. https://dspace.ort.edu.uy/handle/20.500.11968/3761

Godoy Mendía, A. S. (2019). *Aplicación de 'aprendizaje profundo' para el pronóstico de precipitación a partir de datos de reflectividad de radar meteorológico* [Universidad de Cuenca]. http://dspace.ucuenca.edu.ec/handle/123456789/32551

Gonzáles, A. (s.f.). *Conceptos básicos de Machine Learning. Cleverdata*. Recuperado el 28 de noviembre de 2021, de https://cleverdata.io/conceptos-basicos-machine-learning/

Great Learning Team. (2022, August 4*). Types of Neural Networks and Definition of Neural Network*. Artificial Intelligence. https://www.mygreatlearning.com/blog/types-of-neural-networks/

Gupta, Dishashree. (2020). *Activation Functions - Fundamentals of Deep Learning*. https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

Han, L., Sun, J., Zhang, W., Xiu, Y., Feng, H., & Lin, Y. (2016). *A Machine Learning Nowcasting Method based on Real-time Reanalysis Data*. Journal of Geophysical Research, 122(7), 4038–4051. https://doi.org/10.1002/2016JD025783

IDEAM. (s.f.). *Características de la radiación solar*. Recuperado el 26 de noviembre de 2021, de http://www.ideam.gov.co/web/tiempo-y-clima/caracteristicas-de-la-radiacion-solar

Inzunza Bustos, J. (2019). *Meteorología descriptiva*. Universitaria. http://nimbus.com.uy/weather/Cursos/Curso_2006/Textos%20complementarios/Meteorologia%20descriptiva_Inzunza/cap11_Inzunza_Analisis%20y%20pronostico.pdf

Janetzky, P. (2021). *Generative Networks: From AE to VAE to GAN to CycleGAN*. Towards Data Science. https://towardsdatascience.com/generative-networks-from-ae-to-vae-to-gan-to-cyclegan-b21ba99ab8d6

Kathuria, A. (2018). *Intro to optimization in deep learning: Momentum, RMSProp and Adam*. Paper Space Blog. https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

Kaul, S. (2020). *Gradient Descent Problems and Solutions in Neural Networks*. Medium. https://medium.com/analytics-vidhya/gradient-descent-problems-and-solutions-in-deep-learning-8002bbac09d5

Keras. TensorFlow Core. (2020). https://www.tensorflow.org/guide/keras?hl=es-419

La radiación solar. Nuestroclima. (2018). https://nuestroclima.com/la-radiacion-solar/

López Tarabochia, M. (2017). *El ABC del Niño costero: 5 preguntas para entender el fenómeno climático que afecta al Perú.* Mongabay. https://es.mongabay.com/2017/03/peru-nino-costero-desastres-inundacion/

Maithani, M. (2021). *Guide to Tensorflow Keras Optimizers.* Analytics India Mag. https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/

Márquez, V. (2018). *¿Qué es exactamente Machine Learning?* Medium. https://medium.com/latinxinai/qu%C3%A9-es-exactamente-machine-learning-77441201a65b

Marshall, J. S., Langille, R. C., & Palmer, W. M. (1947). *Measurement of rainfall by radar*. Journal of Meteorology, 4 (Issue 6), 186–192.

Metrics. Keras. (s.f.). Recuperado el 29 de noviembre de 2021, from https://keras.io/api/metrics/

Novo-Cuervo, S. (2008). *Pronostico inmediato de tormentas convectivas por radar: una actualizacion*. Revista Brasileira de Meteorologia, 23(1), 41–50. https://doi.org/10.1590/S0102-77862008000100005

Nyuytiymbiy, K. (2020). *Parameters, Hyperparameters, Machine Learning.* Towards Data Science. https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac

Palazzesi, A. (2018). *¿Cómo se realizaban los primeros pronósticos del tiempo?* https://www.neoteo.com/como-se-realizaban-los-primeros-pronosticos-del/

Palomares Calderón de la Barca, M. (2015). *Breve historia de la meteorología* (pp. 1–8). Agencia Estatal de Meteorología. https://repositorio.aemet.es/handle/20.500.11765/900

Parera, L. (2017). *¿Por qué fallan los pronósticos del tiempo?* La Nación. https://www.lanacion.com.ar/sociedad/por-que-fallan-los-pronosticos-del-tiempo-nid2047293/

Peak Signal-to-Noise Ratio as an Image Quality Metric. NI. (2020). https://www.ni.com/es-cr/innovations/white-papers/11/peak-signal-to-noise-ratio-as-an-image-quality-metric.html

Pérez, J., & Gardey, A. (2014). *Definición de tiempo meteorológico - Qué es, Significado y Concepto*. https://definicion.de/tiempo-meteorologico/

Programador Clic. (s.f.) *Aprendizaje profundo: Modelo Secuencial de Base de Keras (secuencial)*. Recuperado el 30 de noviembre de 2021 de https://programmerclick.com/article/6135555252/

Ramesh, S. (2018). *A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST.* Towards Data Science. https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

Redacción APD. (2019). *Qué es Machine Learning, cómo funciona y a qué se aplica*. APD. https://www.apd.es/que-es-machine-learning/

Rosebrock, A. (2019). *Keras Learning Rate Finder*. PyImageSearch. https://www.pyimagesearch.com/2019/08/05/keras-learning-rate-finder/

Sancho, F. (2020,). *Variational AutoEncoder*. http://www.cs.us.es/~fsancho/?e=232

Sanghvirajit. (2021). *A Complete Guide to Adam and RMSprop Optimizer*. Medium. https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be

SAS Institute. (2018). *Aprendizaje automático: Qué es y por qué es importante*. SAS. https://www.sas.com/es_pe/insights/analytics/machine-learning.html

Seelam, S. (2021*). Fundamentals of Deep Learning - Activation Functions and When to Use Them?* Medium. https://medium.com/mlearning-ai/fundamentals-of-deep-learning-activation-functions-and-when-to-use-them-e7d8e2cb231b

Seldon. (2022). *Neural Network Models Explained*. https://www.seldon.io/neural-network-models-explained

Sellat, H. (2019). *Anomaly Detection in Videos using LSTM Convolutional Autoencoder.* Towards Data Science. https://towardsdatascience.com/prototyping-an-anomaly-detection-system-for-videos-step-by-step-using-lstm-convolutional-4e06b7dcdd29

Sharma, S. (2017). *Activation Functions in Neural Networks*. Towards Data Science. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. Advances in Neural Information Processing Systems, Enero-2015, 802–810. https://doi.org/10.48550/arxiv.1506.04214

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W., & Woo, W. (2015). *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. http://arxiv.org/abs/1506.04214

Shi, X., Gao, Z., Lausen, L., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2017). *Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model*. Advances in Neural Information Processing Systems, 2017-December, 5618–5628. https://doi.org/10.48550/arxiv.1706.03458

SmartPanel. (2018). *¿Qué es el Deep Learning?* SmartPanel. https://www.smartpanel.com/que-es-deep-learning/

SSIM: Structural Similarity Index. IMATEST (s.f.). Recuperado el 29 de noviembre de 2021, de https://www.imatest.com/docs/ssim/

Swain, S. (2021). *Understanding Sequential Vs Functional API in Keras*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/07/understanding-sequential-vs-functional-api-in-keras/

TensorFlow. Tensorflow. (s.f.). Recuperado el 30 de noviembre de 2021, de https://www.tensorflow.org/?hl=es-419

Tiempo atmosférico. Wikipedia. (2016). https://es.wikipedia.org/wiki/Tiempo_atmosf%C3%A9rico

Tran, Q. K., & Song, S. K. (2019). *Computer Vision in Precipitation Nowcasting: Applying Image Quality Assessment Metrics for Training Deep Neural Networks*. Atmosphere 2019, Vol. 10, Page 244, 10(5), 244. https://doi.org/10.3390/ATMOS10050244

Ucha, F. (2009). *Definición de Tiempo Meteorológico*. Concepto en Definición ABC. https://www.definicionabc.com/medio-ambiente/tiempo-meteorologico.php

Wall, M. (2014). *Cómo predecir el tiempo de forma casi perfecta*. BBC News Mundo. https://www.bbc.com/mundo/noticias/2014/09/140925_ciencia_exactitud_pronostico_meteorologico_np

Wang, Y., Gao, Z., Long, M., Wang, J., & Yu, P. S. (2018). *PredRNN++: Towards A Resolution of the Deep-in-Time Dilemma in Spatiotemporal Predictive Learning*. 35th International Conference on Machine Learning, ICML 2018, 11, 8122–8131. https://doi.org/10.48550/arxiv.1804.06300

Wang, Y., Long, M., Wang, J., Gao, Z., & Yu, P. S. (2017). *PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs*. Advances in Neural Information Processing Systems, 30. https://papers.nips.cc/paper/2017/hash/e5f6ad6ce374177eef023bf5d0c018b6-Abstract.html

Yegulalp, S. (2019). *What is TensorFlow? The machine learning library explained*. InfoWorld. https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

UNIVERSITAS STUDIORUM PIURENSIS

Apéndices

UNIVERSITAS STUDIORUM PIURENSIS

Apéndice A: Modelo VAE

```
######################################################################
# MODELO VAE PARA PREDICCIÓN DE DESPLAZAMIENTO DE CELDAS DE LLUVIA
######################################################################

import sys
import numpy as np
import os
from datetime import datetime
import tensorflow
from tensorflow.keras.layers import Conv2DTranspose, ConvLSTM2D,
BatchNormalization, TimeDistributed, Conv2D, Dropout
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LayerNormalization
from tensorflow.keras.layers import LeakyReLU

# IMPORTE DE DATASET DE ENTRENAMIENTO

train_input =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/inp5_train.npy')

train_target =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/out5_train.npy')

test_input =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/inp5_test.npy')

test_target =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/out5_test.npy')

# MODELO DE ENTRENAMIENTO

seq = Sequential()
seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=4, padding="same"),
batch_input_shape=(None, None, 40, 40, 1)))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same")))
seq.add(LayerNormalization())
seq.add(Dropout(0.2))
# # # # #
seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(32, (3, 3), padding="same", return_sequences=True))
seq.add(LayerNormalization())
seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
seq.add(LayerNormalization())
seq.add(Dropout(0.2))
# # # # #
seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2,
padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=4,
padding="same")))
seq.add(LayerNormalization())
seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid",
padding="same")))

# MÉTRICAS DE EVALUACIÓN:

def SSIM(y_true, y_pred):
```

```
  maxp = tensorflow.reduce_max(y_pred).numpy()
  minp = tensorflow.reduce_min(y_pred).numpy()
  return tensorflow.reduce_mean(tensorflow.image.ssim(y_true, y_pred,
max_val=maxp-minp))
  #return 1 - tensorflow.reduce_mean(tensorflow.image.ssim(y_true, y_pred,
1.0))

def PSNR(y_true, y_pred):
  maxps = tensorflow.reduce_max(y_pred).numpy()
  minps = tensorflow.reduce_min(y_pred).numpy()
  return tensorflow.reduce_mean(tensorflow.image.psnr(y_true, y_pred,
max_val=maxps-minps))

# FUNCIÓN DE COSTO Y DE OPTIMIZACIÓN

seq.compile(loss=tensorflow.keras.losses.binary_crossentropy,
optimizer=tensorflow.keras.optimizers.Adam(learning_rate=1e-3),
metrics=[SSIM, PSNR], run_eagerly=True)

# ENTRENAMIENTO
from datetime import datetime
nro_epocas = 500

start_time = datetime.now() # Para controlar el tiempo de ejecución

history = seq.fit(train_input, train_target, batch_size=8,
epochs=nro_epocas, validation_data= (test_input, test_target),
callbacks=[tensorboard_callback])

end_time = datetime.now()

duration = end_time - start_time
print("Model Training time: {}".format(duration))

# GUARDAR EL MODELO:

seq.save('/content/drive/My Drive/DATA_Rad/MODEL.h5')
print("Model Saved...")

# GAURDAR LAS MÉTRICAS DE ENTRENAMIENTO EN CADA ÉPOCA

import pickle
with open('/content/drive/MyDrive/DATA_Rad/History-model', 'wb') as
file_pi:
    pickle.dump(history.history, file_pi)
```

Apéndice B: Entrenamiento de modelo

```
#######################################################
# ENTRENAMIENTO DEL MODELO USANDO VALIDACIÓN CRUZADA
#######################################################
import sys
import numpy as np
import os
from datetime import datetime
import tensorflow
#import keras
from tensorflow.keras.layers import Conv2DTranspose, ConvLSTM2D,
BatchNormalization, TimeDistributed, Conv2D, Dropout
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LayerNormalization
from tensorflow.keras.layers import LeakyReLU

# IMPORTE DE DATASET DE ENTRENAMIENTO

train_input =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/inp5_train.npy')

train_target =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/out5_train.npy')

test_input =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/inp5_test.npy')

test_target =
np.load('/content/drive/MyDrive/DATA_Rad/40x40_SP/out5_test.npy')

# CONCATENACIÓN DE DATASETS

inputs = np.concatenate((train_input, test_input), axis = 0)
targets = np.concatenate((train_target, test_target), axis = 0)

num_folds = 6 # Cantidad de folds

# MÉTRICAS DE EVALUACIÓN:

def SSIM(y_true, y_pred):
  maxp = tensorflow.reduce_max(y_pred).numpy()
  minp = tensorflow.reduce_min(y_pred).numpy()
  return tensorflow.reduce_mean(tensorflow.image.ssim(y_true, y_pred,
max_val=maxp-minp))
  #return 1 - tensorflow.reduce_mean(tensorflow.image.ssim(y_true, y_pred,
1.0))

def PSNR(y_true, y_pred):
  maxps = tensorflow.reduce_max(y_pred).numpy()
  minps = tensorflow.reduce_min(y_pred).numpy()
  return tensorflow.reduce_mean(tensorflow.image.psnr(y_true, y_pred,
max_val=maxps-minps))


# Define contenedores de puntaje por cada fold
acc_per_fold = []
loss_per_fold = []

# Define el validador cruzado K-fold
```

```
kfold = KFold(n_splits=num_folds, shuffle=True)

# primer validador K-fold
fold_no = 1

#################  ejecución del entrenamiento  ########################
nro_epocs = 500

for train, test in kfold.split(inputs, targets):

  # Define the model architecture
  seq = Sequential()
  seq.add(TimeDistributed(Conv2D(128, (11, 11), strides=4, padding="same"),
batch_input_shape=(None, None, 40, 40, 1)))
  seq.add(LayerNormalization())
  seq.add(TimeDistributed(Conv2D(64, (5, 5), strides=2, padding="same")))
  seq.add(LayerNormalization())
  seq.add(Dropout(0.2))
  # # # # #
  seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
  seq.add(LayerNormalization())
  seq.add(ConvLSTM2D(32, (3, 3), padding="same", return_sequences=True))
  seq.add(LayerNormalization())
  seq.add(ConvLSTM2D(64, (3, 3), padding="same", return_sequences=True))
  seq.add(LayerNormalization())
  seq.add(Dropout(0.2))
  # # # # #
  seq.add(TimeDistributed(Conv2DTranspose(64, (5, 5), strides=2,
padding="same")))
  seq.add(LayerNormalization())
  seq.add(TimeDistributed(Conv2DTranspose(128, (11, 11), strides=4,
padding="same")))
  seq.add(LayerNormalization())
  seq.add(TimeDistributed(Conv2D(1, (11, 11), activation="sigmoid",
padding="same")))

  seq.compile(loss=tensorflow.keras.losses.binary_crossentropy,
optimizer=tensorflow.keras.optimizers.Adam(learning_rate=1e-3),
metrics=[SSIM, PSNR], run_eagerly=True)

  # Generate a print
  print('------------------------------------------------------------------------
------')
  print(f'Training for fold {fold_no} ...')

  # Fit data to model
  history = seq.fit(inputs[train], targets[train],
              batch_size=8,
              epochs=nro_epocs)

  seq.save('/content/drive/My Drive/DATA_Rad/500epcs_MODEL_kfold_'+
str(fold_no) + '.h5')
  print("Model Saved...")

  with open('/content/drive/MyDrive/DATA_Rad/trainHistory_kfold_'+
str(fold_no), 'wb') as file_pi:
    pickle.dump(history.history, file_pi)

  # Generate generalization metrics
  scores = seq.evaluate(inputs[test], targets[test], verbose=0)
```

```
  print(f'Score for fold {fold_no}: {seq.metrics_names[0]} of {scores[0]};
{seq.metrics_names[1]} of {scores[1]*100}%')
  acc_per_fold.append(scores[1] * 100)
  loss_per_fold.append(scores[0])

  # Increase fold number
  fold_no = fold_no + 1

  del seq
  del history


# IMPRIME LOS SCORES
print('------------------------------------------------------------------------
----')
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('------------------------------------------------------------------------
------')
  print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - SSIM:
{acc_per_fold[i]}%')
print('------------------------------------------------------------------------
----')
print('Average scores for all folds:')
print(f'> SSIM: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
print('------------------------------------------------------------------------
----')
```