EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CULIACÁN

METODOLOGÍA PARA LA IMPLEMENTACIÓN DE MODELOS DE APRENDIZAJE MÁQUINA COMO SERVICIOS

TESIS

PRESENTADA ANTE EL DEPARTAMENTO ACADÉMICO DE ESTUDIOS DE POSGRADO DEL INSTITUTO TECNOLÓGICO DE CULIACÁN EN CUMPLIMIENTO PARCIAL DE LOS REQUISITOS PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

POR:

NÉSTOR LEYVA LÓPEZ
INGENIERO MECATRÓNICO

DIRECTOR DE TESIS:

DR. RAMÓN ZATARAIN CABADA

CULIACÁN, SINALOA Agosto 2022

Dedicatoria

Dedico este trabajo a mis padres, Norberto Leyva Mendívil y Silvia Patricia López Sandoval, por haberme educado, haberme inculcado sus valores y por brindarme todo el amor y paciencia necesarios para criar a una buena persona, no me alcanzará la vida para terminar de agradecerles.

A mi novia Diana Karina Jacobo Rubio, quien ha sido uno de mis más grandes pilares durante estos últimos años, porque siempre ha creído en mí, incluso cuando yo no lo he hecho. Gracias por todo tu apoyo y cariño.

A mis hermanos, Nayely Leyva López y Norberto Leyva López por ser los mejores amigos que la vida haya podido darme, muchas gracias por estar para mí siempre que los he necesitado.

Néstor Leyva López

Agradecimientos

Agradezco a CONACYT por proporcionarme el apoyo económico para poder realizar mis estudios de posgrado.

Al Tecnológico Nacional de México Campus Culiacán, por permitirme estudiar un posgrado de calidad y proporcionarme las herramientas tecnológicas y un espacio físico para cursar las materias de maestría y realizar mi trabajo de investigación.

A mi asesor de tesis, el Dr. Ramón Zatarain Cabada, quien fue un apoyo importante durante el proceso del posgrado y la elaboración de este proyecto y su trabajo escrito. Así como a los integrantes del comité tutorial, la Dra. María Lucía Barrón Estrada y el Dr. Héctor Rodríguez Rangel, quienes me brindaron de consejo y guía para terminar este trabajo de tesis. Así mismo, al resto de profesores, el Dr. Víctor Alejandro González Huitrón, el Dr. Ricardo Rafael Quintero Meza y el Dr. Abraham Efraím Rodríguez Mata, quienes estuvieron presentes para hacer de mi proceso de formación profesional uno más completo.

A la coordinadora de la maestría M.C. Gloria Ekaterine Peralta Peñuñuri, que siempre estuvo en la mejor disposición de dar seguimiento puntal de todo tema relacionado al proceso administrativo del posgrado y CONACYT.

A Lucy López por el apoyo en la gestión de trámites administrativos durante estos dos años del posgrado.

A todos mis compañeros de maestría, Alec Guerrero, Abel Robles, David Tejeda, Jonathan Roldan, Mario Garay, José Medina, Francisco Félix, Óscar Urías y Jaqueline Parra con los cuales compartí estos años de esfuerzo y dedicación, con quienes discutí muchas ideas y en los cuales encontré siempre una respuesta a mis dudas.

Para concluir, agradezco a M.C. Víctor Bátiz, M.C. Héctor Cárdenas, M.C. Aldo Uriarte y M.C. Brandon Cárdenas por el apoyo que siempre se mostraron dispuestos a brindarme, siempre será un gusto trabajar y compartir con ustedes

Declaración de autenticidad

Haciendo uso de la presente declaro que, salvo que se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y que no vulnera los derechos de terceros, incluidos derechos de propiedad intelectual. En ese sentido, el contenido de este trabajo final no ha sido plagiado total ni parcialmente. Esta tesis es el resultado de mi propio trabajo y no incluye nada que sea resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Néstor Leyva López

Culiacán, Sinaloa, México, 2022

Resumen

En los últimos años se han realizado incontables trabajos en el área de aprendizaje automático, llámense redes neuronales, aprendizaje máquina o aprendizaje profundo. Dichos trabajos al ser tan numerosos, variados y al ser desarrollados por equipos de trabajo que buscan fines específicos con sus investigaciones, quedan relegados a ser utilizados solo en las instancias bajo las que fueron creadas. A esto se suma la dificultad que presenta el utilizar alguno de estos modelos, ya que requieren de técnicas y conocimientos específicos en el área para poder implementarlos como parte de otras aplicaciones. Dichas dificultades provocan que los usuarios interesados en usar dichos desarrollos se vean obstaculizados por estas problemáticas.

El actual trabajo nace bajo esta primicia, ya que existe la necesidad latente de una manera de poder implementar desarrollos de modelos de aprendizaje máquina o profundo de una manera sencilla y eficaz. Esto debido a que existe gran variedad de desarrollos de este tipo de trabajos, pero estos no se encuentran a disposición del usuario promedio. Para lograr este cometido, el objetivo principal de este proyecto es desarrollar una metodología para crear una aplicación web que permita utilizar modelos de aprendizaje automático, y a su vez que estos puedan ser consumidos como un servicio disponible a través de cualquier navegador y, de esta manera, cualquier persona con la intención de utilizarlos sea capaz de hacerlo sin conocer los detalles específicos de implementación. Para poder lograr esto se optó por crear una API, ya que este tipo de interfaz de programación otorga las facilidades y la flexibilidad necesarias para manejar los repositorios de modelos e intérpretes, y también cubre la necesidad de tener un servicio web disponible.

Al concluir con el desarrollo de este trabajo se demuestra que el uso de dicha plataforma no tiene un impacto negativo sobre el rendimiento de los algoritmos de aprendizaje automático, obteniendo como resultado que el uso de la aplicación web para albergar modelos de aprendizaje y que estos se encuentren disponibles para consumirse de manera pública, representa una tecnología versátil y útil capaz de aportar sustancialmente al área de investigación de aprendizaje automático.

Palabras clave

- API
- Aprendizaje máquina
- Aprendizaje profundo
- Computación afectiva
- Diferenciación de datos
- Modelos de aprendizaje
- Reconocimiento de personalidad
- Redes neuronales
- Servicios
- Servicios Web
- WSGI

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

1. Introducción

Una interfaz de programación de aplicaciones (o por sus siglas en inglés API) es un conjunto de subrutinas y procedimientos que ofrece una biblioteca que le permite a dos o más aplicaciones comunicarse entre ellas. Estas se conectan a servidores del cual se obtienen datos, se interpretan, se realizan las acciones pertinentes y se envían los resultados de vuelta al usuario que está buscando consumirles (Smola, 2018). En pocas palabras estas permiten que tanto productos y servicios se comuniquen con otros, sin la necesidad de conocer cómo es que estos fueron creados, ayudando en gran medida a los desarrolladores a hacer más eficientemente su trabajo, ya que, ofrecen una gran flexibilidad y simplicidad de diseño para hacer más sencillo tanto el manejo y su uso. Las API ofrecen gran oportunidad de innovar, un aspecto muy importante cuando se está buscando hacer una nueva aplicación.

Debido a que actualmente se vive en la era digital, existe una gran capacidad de comunicación. Los cambios en los proyectos se encuentran a la orden del día gracias a la agilidad y modularidad que permite una API. Al ser estas un medio simplificado para conectar con varias arquitecturas de desarrollo a través de distintas aplicaciones, las API permiten la colaboración entre distintos equipos de desarrolladores. Esto es debido a que hace más fácil el trabajo de integración de distintos componentes de las aplicaciones a una arquitectura. Lo anterior resulta en equipos de trabajo más competitivos, ya que permiten tanto la implementación como el desarrollo rápido de los servicios que ofrecen las aplicaciones web.

Por otro lado, el aprendizaje máquina (ML por sus siglas en inglés) es un área de la inteligencia artificial (IA) y las ciencias de la computación centrada en el uso de datos y algoritmos que buscan imitar la forma en que los seres humanos realizan el proceso de aprendizaje, con mejoras graduales en el mismo (Apt, 2003). Estos sistemas de ML suelen transformar los datos a información útil para toma de decisiones, pero, para que estos realicen predicciones se necesita un gran número de datos.

La aplicación e implementación de modelos de ML en la actualidad es de notable importancia, debido a que se ha convertido en una tecnología imprescindible en varias áreas de las ciencias computacionales, necesaria para mantenerse a la vanguardia en el desarrollo de nuevas tecnologías que aporten nuevos paradigmas de programación y entendimiento de la información, mismos que son tan necesarios en los tiempos que corren en la actualidad. Estas tecnologías van desde filtros *antispam*, que se utilizan para filtrar mensajes en aplicaciones de correo electrónico, hasta software de reconocimiento de personalidad, como *PersonApp* (Bátiz Beltrán, 2021). Esto será aún más importante en un futuro cercano, ya que permitirá ver el procesamiento de reconocimiento facial a través de una nueva perspectiva, y abrir posibilidades sin precedentes.

El problema en estos casos especiales es, que a pesar de que existen una gran variedad de desarrollos de modelos de ML, mismos que funcionan dentro de las instancias de laboratorios o en departamentos de desarrollo e investigación de empresas de iniciativa privada, estos solo cumplen con aquellos objetivos para los cuales fueron desarrollados. Esto significa que estos modelos solo están disponibles para aquellas personas, equipos de desarrollo o investigación que conocen cada detalle de la implementación con que fueron especificados estos modelos de ML. Lo anterior representa una importante limitante para su aplicación, pues presenta una brecha abismal entre quienes saben utilizarlas y quienes, a pesar de su desconocimiento, desean usarlas para realizar predicciones.

En este trabajo se presenta una metodología conveniente, con la cual se permita publicar modelos de ML como servicios públicos disponibles a través de una aplicación web o API. Para de esta manera permitir a cualquier usuario que necesite, o tenga el deseo de utilizar modelos de ML entrenados y probados previamente, pueda lograrlo desde cualquier lugar y en cualquier momento. Todo lo anterior sin la necesidad de conocer todos los detalles específicos de implementación del modelo que se requiera utilizar. Estos detalles van desde la topología de las redes neuronales de aprendizaje profundo o modelos de ML, hasta las especificaciones de preprocesamiento requeridas para cada tipo de dato que pueda ser utilizado en las redes, la compatibilidad del formato de los archivos, entre otras tantas que pueden salir de su conocimiento.

La aplicación web propuesta permite soporte a datos de imagen, vídeo y audio. Para utilizar estos, las redes propuestas son una LSTM, una CNN RestnetLike y una CNN tahn, que se encuentran en el repositorio de redes de la aplicación y con las cuales se busca realizar pruebas en los distintos tipos de archivos que permitidos. Se propone utilizar las redes propuestas en conjunto con la aplicación para realizar tareas de predicción, pero la versatilidad que ofrece el sistema permite que se utilice para resolución de otros tipos de problemas, tales como clasificación, regresión o pronóstico. También se propone un algoritmo de preprocesamiento automático, el cual da el formato necesario a los archivos para que estos sean compatibles con las diversas redes, en este caso particular se utiliza la técnica de transformación de datos para dar el formato requerido por los modelos de aprendizaje automático. Conforme se siga trabajando para crear un sistema más robusto, se prevé que la variedad de archivos soportados y la cantidad de redes disponibles en el repositorio, así como las técnicas de preprocesamiento respectivas a cada una de dichos desarrollos, se vean incrementadas para cubrir con la mayor cantidad de necesidades de los usuarios.

1.1. Descripción de problema

Existe alrededor del mundo una variedad incontable de desarrollos de modelos de ML y aprendizaje profundo, mismos que suelen funcionar dentro de las instancias de laboratorios o empresas que tienen la autoría y derecho de uso de dichos modelos, al ser desarrollados por equipos de trabajo con metas y alcances previamente establecidos. Estos modelos suelen ser diseñados para cometidos o situaciones muy específicas.

El problema presentado por esta situación es que, se requieren algunas técnicas de implementación muy específicas, así como de tipos de preprocesamientos muy especiales para estos modelos. Esto reduce la disponibilidad y añade una capa extra de complejidad, misma que el usuario interesado en usar estos desarrollos ve como una limitante. Esto debido a que no es capaz de utilizarlas para otra aplicación o implementación, tanto académica como personal, frenando el avance de esta área, debido que el uso de las tecnologías acompaña el avance de estas.

1.2. Hipótesis

El diseño de una metodología que implemente una gran variedad de modelos de ML y aprendizaje profundo a nivel aplicación, y a su vez, los presente como servicios disponibles a cualquier usuario a través de navegador *web*, les permitirá a estos la implementación de desarrollos de *software* propios, consumiendo los modelos contenidos en la aplicación web como servicios. La disponibilidad del servicio debe ser desde cualquier lugar y en cualquier momento.

1.3. Objetivos

Estos exponen los resultados que se esperan alcanzar con el desarrollo de este proyecto. Estos se dividen en generales y específicos.

1.3.1. Objetivo general

Desarrollar una metodología que permita la implementación de modelos de aprendizaje máquina y/o modelos de aprendizaje profundo como servicios disponibles de manera pública (disponible en cualquier momento, desde cualquier lugar) a través de una aplicación *web*.

1.3.2. Objetivos específicos

- Implementar modelos de aprendizaje máquina y profundo como servicio web a través de una API.
- Desarrollar un algoritmo para diferenciar tipos de datos.
- Desarrollar un algoritmo que adapte los tipos de datos sometiéndolos a técnicas de preprocesamiento de datos especiales que requieren las distintas entradas de los modelos disponibles.
- Ofrecer los algoritmos de modelos de ML previamente entrenados como un servicio público.
- Ofrecer algoritmos de modelos de aprendizaje profundo (DL por sus siglas en inglés) previamente entrenados como un servicio público.

- Realizar pruebas en la aplicación web usando las predicciones obtenidas en una aplicación.

1.4. Justificación

En la actualidad, la existencia de modelos de ML y DL se encuentra en manos solo de unos cuantos equipos de trabajo, grupos de investigación o empresas de iniciativa privada alrededor del mundo. Dicha situación representa una limitante clara, debido a que esto impide el que cualquier persona que tenga el interés o la necesidad de aplicar o implementar este tipo de modelos lo lleve a cabo de manera satisfactoria, entorpeciendo cualquier desarrollo que dicho usuario esté pensando llevar a cabo.

Lo anterior es fundamental, ya que no es necesario investigar y sumergirse en las técnicas de procesamiento e implementación requeridas para poder utilizarlos. De esta manera generando un mayor número de investigaciones relevantes en el campo de ML y DL. También se busca ofrecer una plataforma donde se encuentra disponible una variedad de modelos de ML y DL previamente entrenados, y todos estos utilizables por cualquier usuario de la API. Por último, se provee un diseño de arquitectura viable en el cual se permite montar modelos de aprendizaje automático. Esto genera que las posibilidades de la aplicación web sean muy grandes, ya que la arquitectura otorga una gran flexibilidad y posibilidad para ajustarlo a distintas necesidades. Así se pueden proveer los modelos de ML y DL como un servicio web a través de una API disponible para cualquier usuario.

1.5. Estructura de la tesis

Esta tesis consta de 6 capítulos, mismos que se encuentran organizados de la siguiente manera:

- Capítulo 1 – Introducción: Este capítulo proporciona la información básica y relevante que ayuda a establecer el contexto del proyecto.
- Capítulo 2 – Marco teórico: Este funge como soporte teórico y conceptual de los conceptos que se utilizan en el proyecto de investigación, tales como las bases de ML, aprendizaje profundo, IA y los fundamentos de la creación de API *web*, así como de

algunos de los aspectos principales de la computación afectiva. También se presenta el aporte del trabajo en cuestión, que es el proveer una metodología para ofrecer modelos de ML como un servicio.

- Capítulo 3 – Estado del arte: Este presenta un resumen de trabajos previamente investigados, que guardan una estrecha relación con el tema que le concierne a este trabajo, así como una breve descripción de cada uno de ellos.
- Capítulo 4 – Metodología: Aquí son planteados los pasos necesarios a seguir para la creación del proyecto expuesto en este trabajo. También, se incluye una explicación detallada de la arquitectura propuesta para la API, misma que permite que este sea un servicio disponible al público. También se abordan los algoritmos creados para la selección de los modelos de ML y DL óptimos para los datos posibles a los cual el servicio será expuesto. De igual manera se detalla la forma en que fueron utilizadas las herramientas y los criterios bajo los cuales estas fueron elegidas. Por último, se presenta la explicación de cómo es que funcionan los algoritmos de preprocesamiento automático.
- Capítulo 5 – Pruebas y resultados: se presentan las pruebas realizadas con la aplicación web, utilizando las distintas redes para archivos variados, exponiendo análisis y comparativas.
- Capítulo 6 – Conclusiones: Se muestra en resumen las ideas más importantes del trabajo, así como el aporte del mismo, también se trata el trabajo que se espera realizar a futuro.

Capítulo 2

2. Marco teórico

Al tratarse de un proyecto en el que se ponen a disposición distintos tipos de redes neuronales, tales como de ML y DL, cada una de estas tiene su apartado individual. Debido a que el tema principal de este trabajo es el de presentar redes neuronales como servicios disponibles, también se abordan las API como servicios. Al final, el marco teórico aborda el tema de computación afectiva, ya que, al presentar el tema de redes de aprendizaje automático como servicio, se realizan pruebas con redes de reconocimiento de personalidad.

2.1. Aprendizaje máquina

El aprendizaje máquina (ML por sus siglas en inglés), es un tipo de inteligencia artificial (IA por sus siglas en inglés) que le permite a un sistema aprender de datos, a diferencia del acercamiento tradicional, dónde los sistemas funcionan por programación explícita y de manera secuencial. El desempeño y el análisis computacional de los algoritmos ML es una rama de la estadística conocida como la teoría de aprendizaje computacional. En resumen, el ML trata de diseñar algoritmos que permitan a una computadora aprender. Así, el aprendizaje máquina usualmente se refiere a los cambios en los sistemas que realizan tareas asociadas a la IA, dónde dichas tareas involucran reconocimiento, diagnóstico, planeación, control de robots y predicciones entre otros (Nilsson, 1998). Estos cambios deben mejorar sistemas ya existentes o síntesis de nuevos sistemas.

Durante las últimas décadas, el ML se ha convertido en uno de los pilares de la tecnología de la información, ya que puede aparecer en muchas formas debido a su versatilidad. Aunque a menudo está oculto, el ML se ha convertido en parte de nuestra vida diaria. Con la creciente cantidad de datos fácilmente disponibles, hay buenas razones para creer que el análisis inteligente de estos mismos datos se convertirá en algo que se encontrará en todos los ámbitos, y por ende será esencial para el progreso tecnológico (Smola, 2018).

2.1.1. Aplicaciones de ML

El ML tiene una amplia variedad de aplicaciones, estas cubren varios dominios y subdominios. Algunos de estos dominios son visión por computadora, predicción, análisis semántico, procesamiento de lenguaje natural (NLP por sus siglas en inglés) y recuperación de datos (Apt, 2003). Algunos de los subdominios de estos son los siguientes:

- **Visión por computadora:** Reconocimiento, detección y procesamiento de objetos.
- **Predicción:** Se compone de clasificación de documentos, análisis de imágenes, diagnósticos médicos, redes de detección de intrusos y predicción de denegación de servicios de ataque.
- **Análisis semántico, NLP y recuperación de datos:** El análisis semántico es el proceso de relacionar estructuras sintácticas, dígase párrafos, oraciones y palabras al nivel escritural como un todo. El NLP es como programar computadoras para procesar correctamente los datos del lenguaje. Por último, la recuperación de datos es la ciencia de buscar información en un documento, buscando la información que describe las bases de datos de sonido e imágenes.

En la **Fig. 1** se muestra una descripción de las aplicaciones de ML y sus relaciones con sus dominios y subdominios.

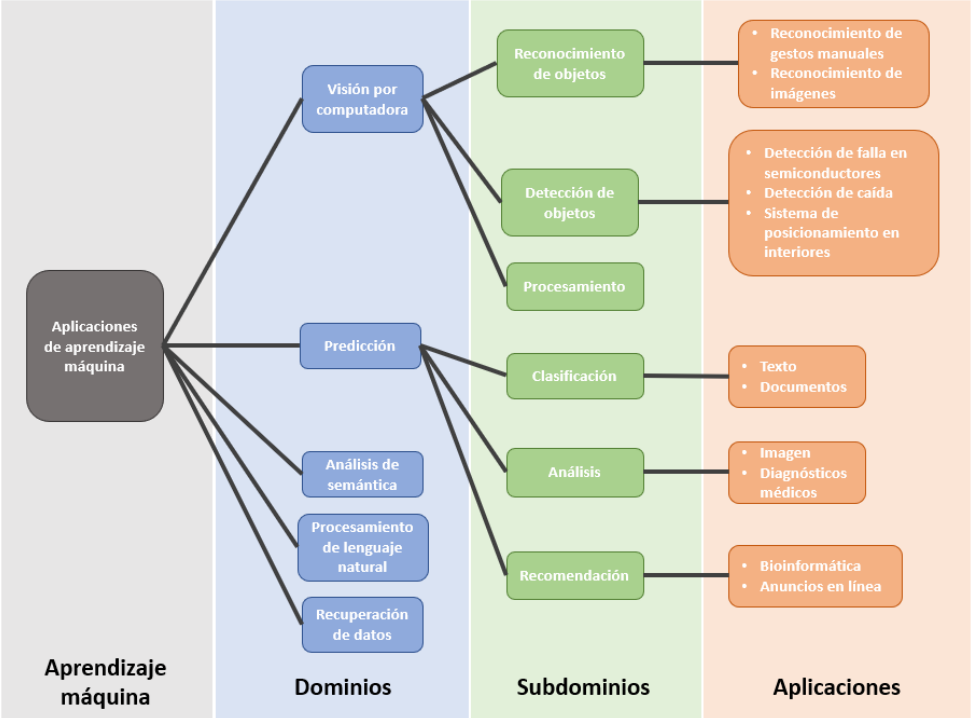

Fig 1. Aplicaciones del aprendizaje máquina

2.1.2. Datos

Los datos son representaciones simbólicas de atributos o variables cuantitativas o cualitativas. Estos son la información o valores que se procesan en algoritmos y pueden contener una gran variedad de información en distintos formatos (Editorial Etecé, 2021). La importancia de los datos en las ciencias computacionales radica en que estos permiten almacenar y analizar información de una manera organizada. Los datos se obtienen de distintas maneras, que van desde encuestas hasta el monitoreo de redes sociales. En el área de ML los datos son utilizados para enseñar a los modelos a producir mejores resultados.

Es útil el caracterizar los problemas de aprendizaje, de acuerdo con el tipo de datos que usan. Esto es conveniente cuando se encuentran nuevos desafíos, ya que estos problemas a menudo ocurren en tipos de datos similares y, por lo tanto, se pueden resolver usando técnicas similares (Smola, 2018).

Uno de los desafíos en el procesamiento de vectores es que la escala y las unidades de diferentes coordenadas puede variar ampliamente. Por ejemplo, se pueden realizar medidas,

ya sea de peso, altura o temperatura con distintos tipos de unidades. Para hacer frente a esta situación, los datos deben ser normalizados, lo cual consiste en asociar los datos de una forma estandarizada (por ejemplo, en una imagen las dimensiones se llevan a un tamaño estándar). Las formas de hacer esto automáticamente se presentan a continuación (Smola, 2018).

- **Listas:** En algunos casos, los vectores que se obtienen pueden contener un número variable de características.
- **Conjuntos:** Suelen aparecer en problemas de aprendizaje cuando hay un gran número de causas potenciales para un efecto y, que no se encuentran bien determinadas.
- **Matrices:** Son un medio conveniente de representar relaciones de parejas.
- **Imágenes:** Pueden ser vistas como arreglos bidimensionales de números, que fundamentalmente son matrices. Estas representaciones son muy primitivas, a pesar de que muestran unidades espaciales (líneas, figuras) y una estructura multirrelacional (las imágenes). Por lo tanto, la reducción de resolución de una imagen da como resultado un objeto con valores estadísticos similares a los de la imagen original.
- **Videos:** Agrega la dimensión de tiempo a las imágenes. Se pueden representar como una matriz tridimensional. Sin embargo, los buenos algoritmos tienen en cuenta la consistencia temporal de las secuencias de imágenes.
- **Árboles y gráficas:** Son a menudo utilizados para describir relaciones entre colecciones de objetos.
- ***Strings*:** Ocurre con frecuencia y principalmente en los campos de la bioinformática y procesamiento del lenguaje natural. Deben ser las entradas a los problemas de estimación, de igual manera, cuando constituyen las salidas de un sistema.
- **Estructuras compuestas:** Son el objeto más común, en la mayoría de los casos habrá una mezcla de estructuras de diferentes tipos de datos. Un buen modelado estructural tiene en cuenta las dependencias y la estructura para crear modelos que sean lo suficientemente flexibles.

2.1.3. Tipos de algoritmos de aprendizaje máquina

Los algoritmos de ML están organizados por tipo de clasificación, según lo que se espera del algoritmo (Oladipupo Ayodele, 2010). Los tipos comunes de algoritmos incluyen:

- **Aprendizaje supervisado:** Aquí es donde los algoritmos crean una función que asigna la entrada a la salida deseada. La formulación estándar de las tareas de aprendizaje supervisado es categorizar las problemáticas que se necesitan aprender. De esta manera se aproxima al comportamiento de la función asignando un vector a una o más categorías, observando varios ejemplos de ingreso a una función de salida.
- **Aprendizaje no supervisado:** Estos modelan un conjunto de entradas, pero los ejemplos no se encuentran disponibles para el aprendizaje.
- **Aprendizaje semi-supervisado:** Dónde se combinan ambos, tanto los ejemplos etiquetados y los no etiquetados para generar una función o clasificador apropiados.
- **Aprendizaje reforzado:** En él, el algoritmo aprende una política sobre cómo comportarse al dar una observación al entorno. Cada acción tiene alguna reacción sobre el entorno, y el propio entorno proporciona información que guía el algoritmo de aprendizaje.
- **Transducción:** Similar al aprendizaje supervisado, excepto que no crea una función explícitamente. En su lugar, intenta predecir nuevas salidas en función de las entradas de entrenamiento, las salidas previamente entrenadas y nuevas entradas.
- **Aprendiendo a aprender:** El algoritmo aprende su sesgo inductivo basado en la experiencia pasada.

2.1.3.1. Aprendizaje supervisado

Es bastante común en la clasificación de problemas, ya que el propósito suele ser que la computadora aprenda un sistema de clasificación generado. El reconocimiento de números es un ejemplo común de aprendizaje por clasificación. El aprendizaje por clasificación suele ser fácil de identificar. En algunos casos, puede que no sea necesario asignar clasificaciones predeterminadas a cada instancia de un problema si el agente puede trabajar con la misma clasificación.

El aprendizaje supervisado a menudo deja la probabilidad para entradas indefinidas. Este modelo no es necesario siempre que las entradas se encuentren disponibles, pero si no se encuentra algún valor de entrada, no se pude deducir nada de la salida (**Fig. 2**).

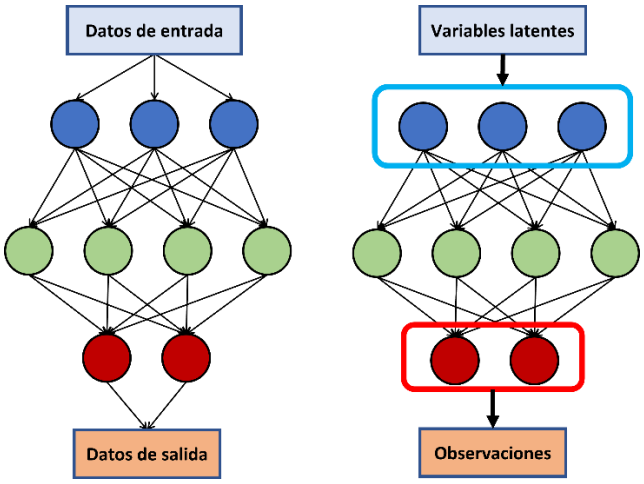

Fig. 2. Ejemplos de aprendizaje supervisado (Izquierda) y no supervisado (Derecha)

Tanto redes neuronales como árboles de decisión son altamente dependientes de la información dada por la clasificación predeterminadas. En el caso de las redes neuronales, la clasificación es usada para determinar el error en una red y, entonces ajustar los valores para minimizarlo. En cambio, en los árboles de decisión, la clasificación es usada para resolver la clasificación como si de un rompecabezas se tratase.

Los pasos para el aprendizaje supervisado (Oladipupo Ayodele, 2010) (**Fig. 3**) son los siguientes:

1. **Paso 1:** Recolectar el conjunto de datos.
2. **Paso 2:** Preparación y preprocesamiento de datos.
3. **Paso 3:** Definición del conjunto de entrenamiento
4. **Paso 4:** Algoritmo de selección
5. **Paso 5:** Entrenamiento
6. **Paso 6:** Evaluación con conjunto de prueba
7. **Paso 7:** Afinación de parámetros
8. **Paso 8:** Clasificar

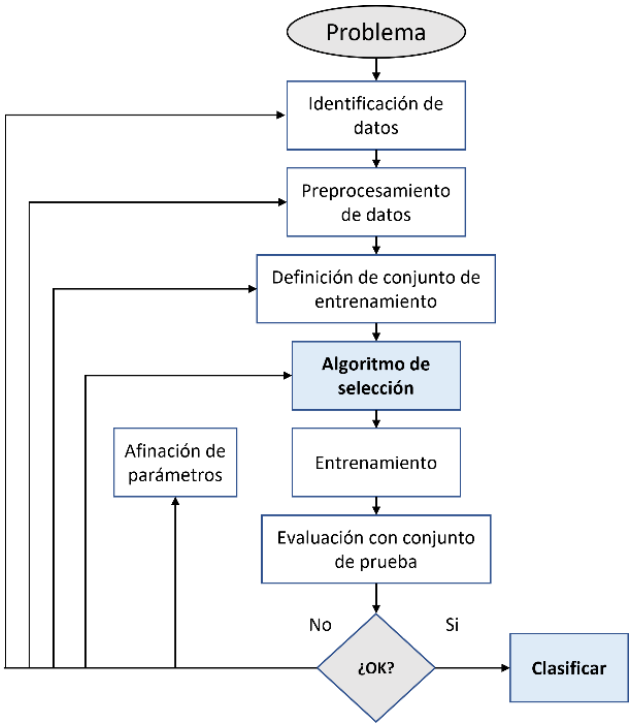

Fig. 3. Proceso de ML supervisado

2.1.3.2. Aprendizaje no supervisado

El aprendizaje no supervisado parece mucho más difícil: el objetivo es hacer que la computadora aprenda a hacer algo que no se le muestra cómo hacerlo. Para conseguirlo existen dos acercamientos para este tipo de aprendizaje. El primero es enseñar sin proporcionarle categorizaciones explícitas, sino usando algún tipo de sistema de recompensa para indicar éxito. Este tipo de entrenamiento generalmente cae en el marco de problemas de decisión, porque no se busca clasificar, sino tomar decisiones que maximicen las recompensas. A menudo se puede usar una forma de aprendizaje reforzado para aprendizaje no supervisado, en el que las acciones se basan en recompensas y castigos previos sin aprender ningún tipo de información de como las acciones afectan el mundo.

El segundo tipo de aprendizaje no supervisado es llamado *clustering*. En este no se busca maximizar la función de utilidad, sino encontrar similitudes en los datos de entrenamiento. Lo que se puede asumir es que los *clusters* encontrados coincidirán con una clasificación intuitiva. Este es un acercamiento orientado a datos que funciona bien cuando hay suficiente

información. Desafortunadamente, incluso el aprendizaje no supervisado sufre del problema de sobreajustar los datos de entrenamiento.

Los algoritmos de aprendizaje no supervisado se diseñan para extraer estructuras desde muestreo de datos. La calidad de dicha estructura es medida por una función de costo, la cual suele ser minimizada para deducir los parámetros óptimos caracterizando la estructura de los datos. Una inferencia confiable y robusta requiere una garantía de que las estructuras extraídas son típicas para los datos.

2.2. Aprendizaje profundo

Definir el aprendizaje automático es un desafío porque ha cambiado mucho durante la década pasada. La definición más útil muestra que el DL trata con redes neuronales de más de dos capas. Más precisamente, el DL (**Fig. 4**) es una red neuronal que contiene una gran cantidad de parámetros y capas que pertenecen a una de las cuatro infraestructuras de red (Patterson & Gibson, 2017), las cuales se exponen a continuación.

- Redes no supervisadas pre entrenadas
- Redes neuronales convolucionales
- Redes neuronales recurrentes
- Redes neuronales recursivas

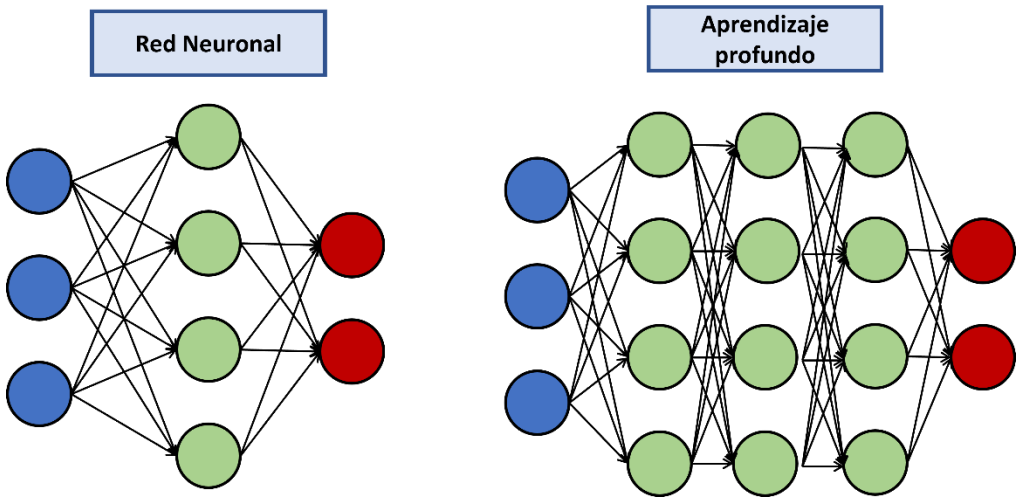

Fig. 4. Comparativa entre red neuronal (Izquierda) y red de DL (Derecha)

Las redes neuronales deben ir más allá de la arquitectura de los patrones de red anteriores (con más poder de procesamiento). A continuación, se presentan algunas de las facetas en la evolución de las redes neuronales:

- Más neuronas que redes anteriores
- Más formas complejas de conectar capas y neuronas en redes neuronales
- Explosión en la cantidad de poder computacional disponible para entrenar
- Extracción automática de características

Hay muchas variaciones de las arquitecturas fundamentales, como un conjunto de redes neuronales acumulativas y circulares. La extracción automática de características es otra de las grandes ventajas que el DL tiene sobre los algoritmos tradicionales de ML (**Fig. 5**).

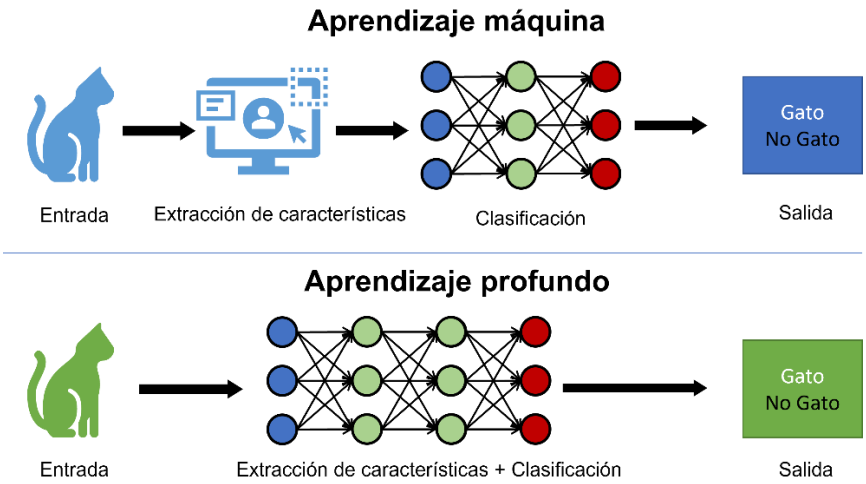

Fig. 5. Comparativa entre ML (Arriba) y DL (Abajo)

2.2.1. Componentes principales del DL

El comprender los componentes principales que conforman el DL ayuda a entender mejor estas redes, dichos componentes son:

- **Parámetros:** Estos se relacionan directamente a los pesos en las conexiones de las redes que se intentar optimizar. También se refiere a la manera en que las capas están conectadas en una arquitectura.

- **Capas:** Estas son una unidad arquitectónica fundamental en redes profundas. Distintas combinaciones de estas capas ayudan a lograr el objetivo propuesto.
- **Funciones de activación:** Se utilizan para transferir la salida de un nodo de una capa hacia la siguiente capa. Estas funciones "escalar a escalar" se encargan de la activación de las neuronas. Esto permite que sean utilizadas para activar las neuronas ocultas en la red. Dependiendo del tipo de dato con el que se esté trabajando, puede variar la función de activación adecuada para el mismo.
- **Funciones de pérdida:** Las funciones de pérdida cuantifican la concordancia entre la salida predicha (o etiqueta) y la salida real. Estas son usadas para determinar la penalización por una clasificación incorrecta de una entrada.
- **Algoritmos de optimización:** Entrenar un modelo de aprendizaje automático involucra encontrar el mejor conjunto de valores para el parámetro del modelo. Se puede pensar como un problema de optimización, en dónde se minimiza la función de pérdida con respecto a los parámetros de la función de predicción.
- **Hiperparámetros:** Son cualquier conjunto de valores de configuración que puede elegir libremente el usuario y, que pueda afectar el desempeño directamente. Estos hiperparámetros caen en varias categorías, tales como:
    - Tamaño de capa
    - Magnitud
    - Regularización
    - Activaciones
    - Estrategia de inicialización de peso
    - Funciones de pérdida
    - Configuraciones por épocas durante entrenamiento
    - Esquema de normalización para datos de entrada

2.2.2. Bloques de construcción de redes profundas

Los bloques de construcción de redes van más allá de las redes neuronales multicapa *feed-forward*. Las redes profundas combinan redes más pequeñas a modo de bloques de

construcción para redes más grandes. En otros casos, se usa un conjunto especializado de capas. Algunos de los bloques de construcción específicos son:

- **Redes neuronales multicapa *feed-forward*:** Estas son las redes neuronales más simples y se inspiran en las redes de neuronas biológicas. Están compuestas por una capa de entrada, una o varias capas ocultas y una capa de salida (**Fig. 6**).

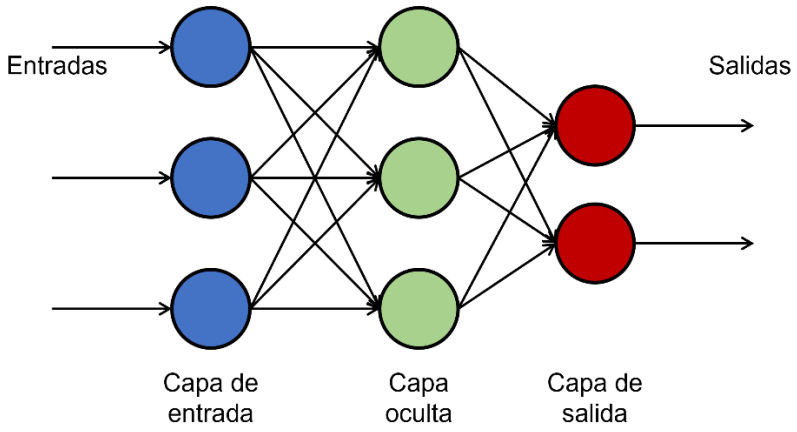

Fig. 6. Red neuronal *Feed-forward*

- **RBM:** Por sus siglas en inglés Máquina Restringida de Boltzmann, esta es un tipo de red conectada simétricamente por neuronas que realizan decisiones de si deben permanecer encendidas o apagadas (**Fig. 7**). Estas se utilizan en DL para extracción de características y reducción de dimensiones. La parte "restringida" de su nombre significa que las conexiones entre los nodos de una misma capa se encuentran prohibidas. Las RBM también son un tipo de autoencoder, y también son usadas como parte de redes más grandes como las redes de creencia profunda.

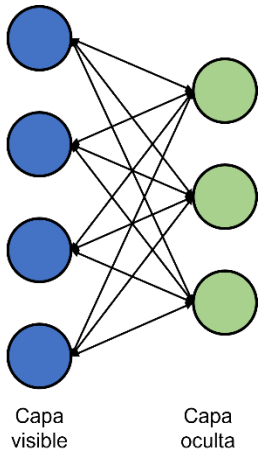

Fig. 7. Arquitectura de una red RBM

- **Autoencoder:** Se utilizan para aprender representaciones comprimidas de conjuntos de datos y, típicamente se usan para reducir la dimensionalidad de estos. La salida de una red autoencoder es una reconstrucción de los datos de entrada en la forma más eficiente. Estos comparten un fuerte parecido con una red neuronal multicapa, porque tienen una capa de entrada, capas ocultas de neuronas, y al final una capa de salida (**Fig. 8**). La diferencia para notar entre ambas arquitecturas de redes es que en el autoencoder se obtienen el mismo número de salidas que de entradas.

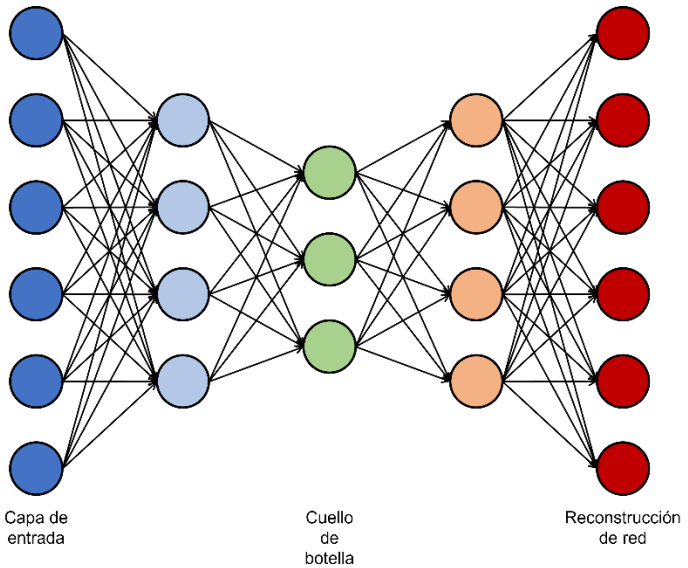

Fig. 8. Arquitectura de una red Autoencoder

2.2.3. Arquitecturas principales de redes profundas

En este apartado se abordan las cuatro arquitecturas principales de las redes profundas y como es que se usan redes más pequeñas para construirlas. Estas arquitecturas son:

- Redes no supervisadas pre entrenadas (UPN)
- Redes neuronales convolucionales (CNN)
- Redes neuronales recurrentes
- Redes neuronales Recursivas

Mismas que se revisarán a mayor detalle.

2.2.3.1. Redes no supervisadas pre entrenadas

Incluyen principalmente tres tipos específicos de arquitectura, las cuales son *Autoencoder*, Redes Neuronales de Creencia Profunda (por sus siglas en inglés DBN) y Redes Adversarias Generativas (por sus siglas en inglés GAN). Debido que los *autoencoder* ya han sido abordados anteriormente, a continuación, se describen solamente las GAN y DBN.

- **Redes neuronales de creencia profunda (DBN):** Están compuestas por capas de RBM para el preentrenamiento y después una red *feed-forward* para el ajuste y afinado (**Fig. 9**). Estas son utilizadas para extraer características de alto nivel desde los vectores de entradas. Son modelos gráficos generativos que se componen de capas múltiples con variables latentes o unidades ocultas. Una de sus características más notables es que la comunicación de este tipo de red se da entre las capas de esta, y no en las neuronas dentro de las capas. Al entrenar una DBN, esta puede aprender a reconstruir probabilísticamente sus entradas.

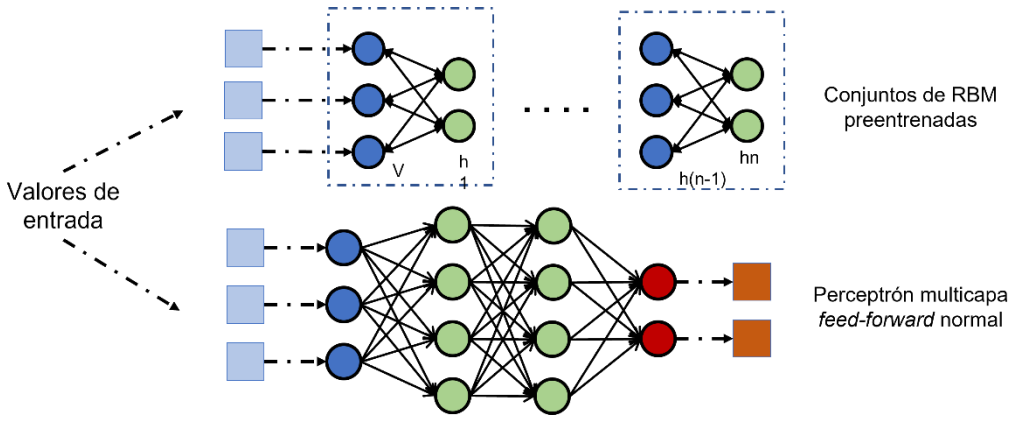

Fig. 9. Arquitectura de una DBN

- **Redes adversarias generativas (GAN):** Estas han demostrado ser adeptas para componer nuevas imágenes a partir de otras imágenes de entrenamiento. Este concepto puede extenderse al modelado de otras áreas como audio y vídeo, o la creación de imágenes a partir de descripciones escritas. Las GAN (**Fig. 10**) son ejemplos de redes que usan aprendizaje no supervisado para entrenar dos modelos en paralelo. Un aspecto esencial de las GAN es la forma en que usan un parámetro para calcular mucho menos de lo habitual, en comparación con la cantidad de datos con los que se entrena una red.

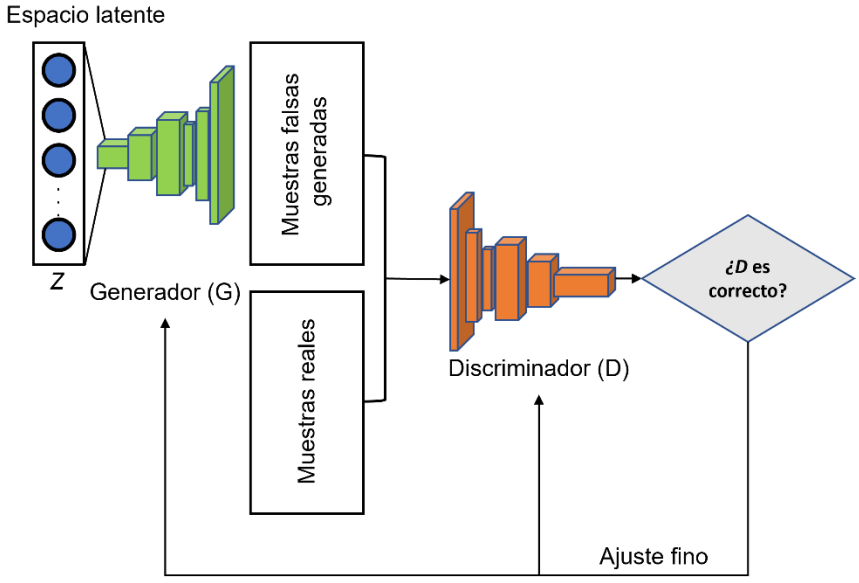

Fig. 10. Arquitectura de una GAN

Existen distintos tipos de redes GAN, como:

- **Redes discriminativas:** Al modelar imágenes, la red discriminativa suele ser una CNN estándar. Usando una red neuronal secundaria como red discriminativa permite que funcione como GAN, que entrena dos redes paralelas sin supervisión. Estas redes toman imágenes como entrada y luego clasifican a la salida. La gradiente de la salida de la red discriminativa con respecto a la salida de datos compuestos muestra cómo realizar pequeños cambios para que sea más realista.
- **Redes generativas:** Las redes generativas crean datos con un tipo especial de capa llamada "capa deconvolucional". Durante el entrenamiento, se usa retropropagación en ambas redes para actualizar los parámetros de la red generativa, lo que produce una salida más realista. El objetivo es el actualizar los parámetros de la red generativa hasta el punto en que la red está lo suficientemente engañada por la red generativa, porque la salida es muy real comparada al conjunto de datos de entrenamiento.
- **GAN Condicional:** Estas redes son implementadas en una red GAN previamente existente. La GAN condicional se utiliza agregando una nueva capa de entrada con etiquetas. Dicha capa adicional guía la GAN en términos de que muestras producir, lo que permite crear datos condicionales para una clase en particular.

2.2.3.2. Redes Neuronales Convolucionales (CNN)

El objetivo de una CNN es aprender características de orden alto en los datos a través de las convoluciones. Están adaptadas para reconocimiento de objetos con imágenes. Se pueden identificar caras, individuos, señales de tráfico y muchos otros datos visuales. Las CNN también puede ser utilizadas para analizar texto a través del reconocimiento de caracteres, pero también es útil cuando analiza palabras como unidades textuales.

La eficiencia de las CNN en reconocimiento de imágenes es una de las razones principales del porque son tan populares. Las CNN son buenas para construir características de posición y rotación invariante a partir de datos de imagen en crudo (**Fig. 11**).

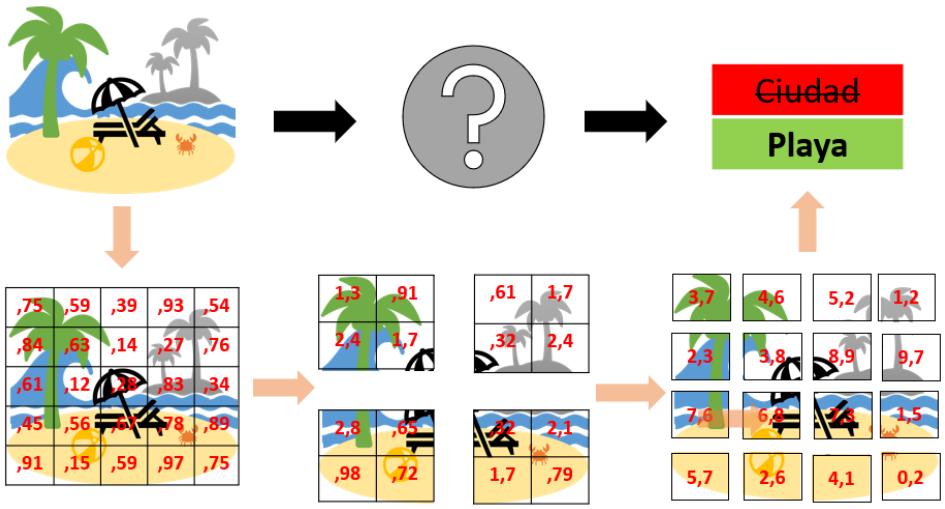

Fig. 11. CNN en visión por computadora

Las CNN tienden a ser más útiles cuando existe una estructura para los datos de entrada. Un ejemplo es la forma en que las imágenes y datos de audio que tienen un conjunto específico de repetición de patrones y valores de entradas están relacionadas espacialmente. Por el contrario, los datos en columnas exportados desde un sistema de manejo de base de datos relacional (RDBMS), tienden a no tener una relación estructural. Las CNN también son usadas para otras tareas, tales como traducción y generación de lenguaje natural y análisis de sentimientos.

Las CNN transforman los datos de entrada de la capa de entrada a través de todas las capas conectadas en un conjunto de clases dadas por la capa de salida. Existen muchas variaciones de la arquitectura de las CNN, pero normalmente están basadas en el patrón de capas (**Fig. 12**). Estas son:

- Capa de entrada
- Capa de extracción de características (Aprendizaje)
- Capas de clasificación.

La capa de extracción de características tiene un patrón de repetición general de una secuencia:

- **Capa de convolución:** Expresa la función de activación de la unidad lineal rectificada (ReLU).

- **Capa de *Pooling:*** Estas capas encuentran algunas características en imagen y construyen gradualmente características de orden superior. Esto corresponde directamente a como las características se aprenden automáticamente.

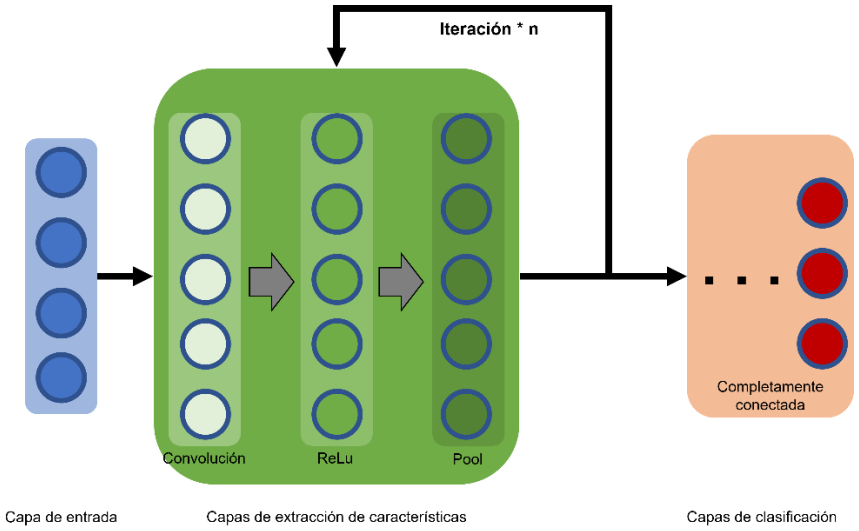

Fig. 12. Arquitectura de una CNN de orden superior

2.2.3.3. Redes Neuronales Recurrentes (RNN)

Las RNN están en la familia de las redes neuronales *feed-forward*. Son diferentes de otras redes *feed-forward* en su habilidad de enviar información sobre pasos de tiempo.

Las RNN permiten tanto la computación paralela y secuencial (**Fig. 13**), y en principio puede computar lo mismo que una computadora tradicional. La diferencia principal es que las RNN son similares al cerebro humano, que fundamentalmente es una gran red de retroalimentación de neuronas conectadas que, de alguna manera aprende a traducir una entrada sensorial en una secuencia de salidas motrices útiles.

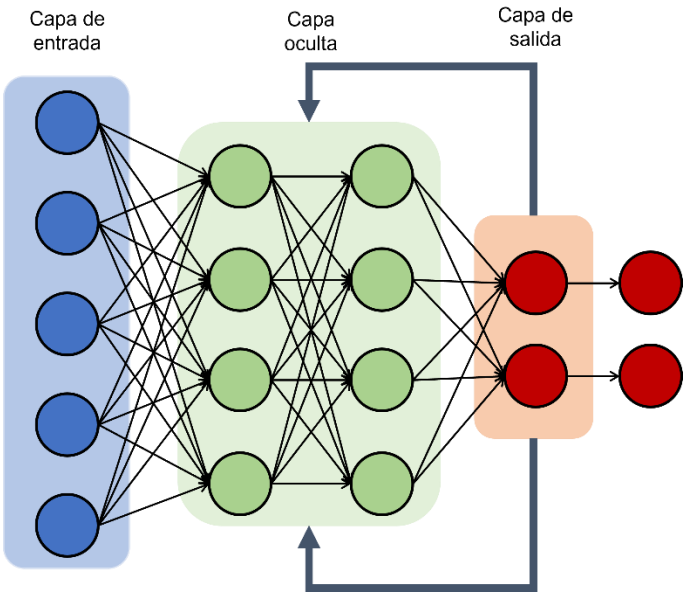

Fig. 13. Arquitectura de una RNN

Históricamente, estas redes han sido difíciles de entrenar, pero recientemente, los avances en investigación (optimización, arquitectura de red, paralelismo y unidades de procesamiento gráfico) han facilitado este proceso.

Las RNN toman cada vector de una serie de vectores de entrada y modela cada uno de ellos. Esto permite que la red conserve un estado mientras modela cada vector de entrada a través de una ventana de vectores de entrada. El modelado de las dimensiones de tiempo es una marca recurrente en las RNN.

Dentro de las RNN se encuentran las redes LSTM (**Fig. 14**), ya que son un tipo de red recurrente, las cuales son capaces de aprender la dependencia de datos a largo plazo y diseñadas, explícitamente, para evitar los problemas que dicha dependencia conlleva. Su característica principal es que la información persiste al introducir bucles, lo que permite recordar estados previos. Cómo todas las redes neuronales recurrentes, las LSTM forma una cadena de módulos de repetición, con la diferencia de que, en este módulo, se cuenta con 4 capas de procesamiento.

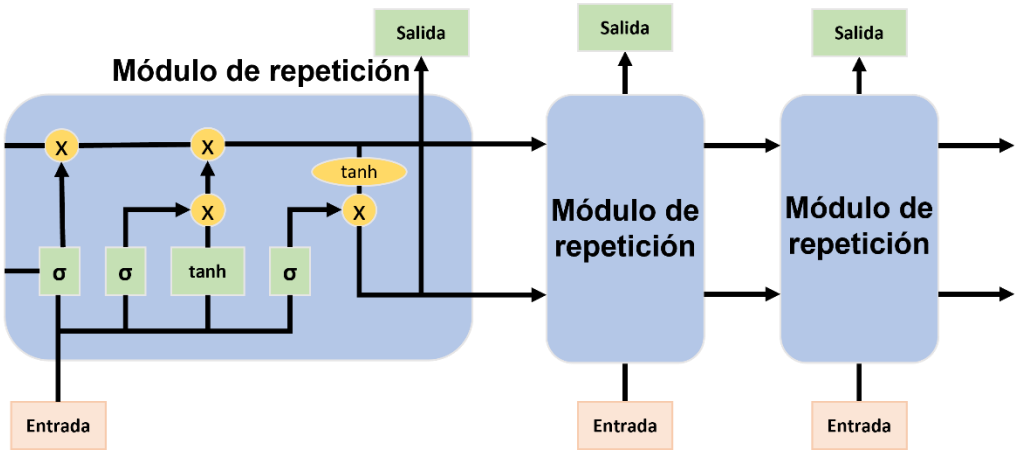

Fig. 14. Arquitectura de una LSTM

2.2.3.4. Redes Neuronales Recursivas

Las redes neuronales recursivas (**Fig. 15**), como las RNN, pueden manejar entradas de tamaño variable. La principal diferencia entre estas es que las RNN tienen la habilidad de modelar estructuras jerárquicas en un conjunto de datos de entrenamiento. Las imágenes a menudo tienen escenas formadas por varios objetos. La deconstrucción de escenas suele ser un problema de interés, pero no es trivial. La naturaleza iterativa de la deconstrucción nos desafía a no solo identificar objetos en la escena, sino también se relacionan los objetos en ella.

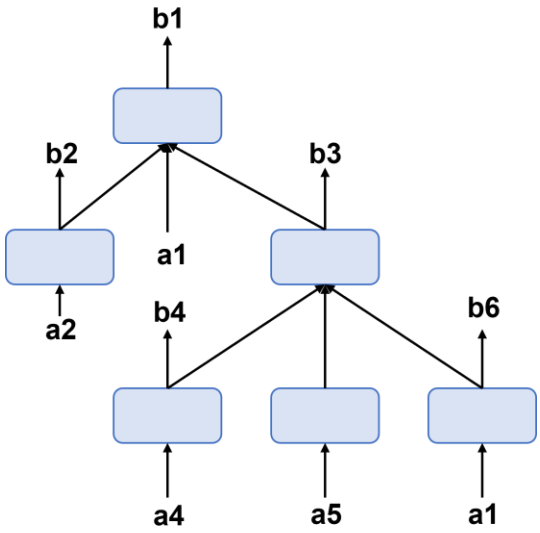

Fig. 15. Arquitectura de redes neuronales recursivas

La arquitectura de una red neuronal recursiva incluye una matriz de pesos compartidos y una estructura de árbol binario. Esto permite que la red recursiva aprenda secuencias de palabras o transforme partes de una imagen. Es útil como herramienta para el análisis de oraciones y escenas. Estas utilizan un tipo diferente de retropropagación llamado “retropropagación a través de estructuras” (BTPS).

Una red recursiva puede tener muchas variaciones. Una es el autoencoder recursivo. Justo como una red *feed-forward*, aprende a reconstruir las entradas. En el caso de los encoder recursivos semisupervisados, aprende la probabilidad de ciertas etiquetas en cada contexto. Otra variante de esta es la red neuronal supervisada, llamada red neuronal recursiva de tensores, que calcula el objetivo supervisado en cada nodo del árbol. La parte de tensor de esto calcula el gradiente un poco diferente y obtiene más información en cada nodo aprovechando una dimensión diferente de información con dicho tensor.

Ambas redes, tanto recursivas como recurrentes, comparten muchos de los mismos casos de uso. Las RNN tradicionalmente se usan en NLP, debido a sus lazos con los árboles binarios, contextos, y analizadores sintácticos basados en el lenguaje natural. Las redes neuronales recursivas pueden recuperar estructuras granulares y estructuras de jerarquías de orden superior en conjuntos de datos. Las aplicaciones más características de las redes neuronales recursivas son:

- Descomposición de imágenes
- Procesamiento de lenguaje natural
- Transcripción de Audio a texto

Las configuraciones de red específicas observada en la práctica son los autoencoder recursivos y tensores neuronales recursivas. Se utilizan autoencoder recursivos para dividir oraciones en segmentos de lenguaje natural. Los tensores neuronales se utilizan para segmentar una imagen en los objetos componentes y etiqueta los objetos en la escena.

Las redes neuronales recurrentes tienden a entrenarse más rápido, por lo que a menudo se usan en aplicaciones más temporales, pero se ha demostrado que funcionan bien en campos que dependen del NLP, como el análisis de sentimientos.

2.3. API como servicios

Las API representan el *software* como servicios (Por sus siglas en inglés SaaS). Las API como servicio son, fundamentalmente, una plataforma de software que permite a los usuarios interactuar con las API de terceros, al igual que manejar sus propias API. Estas proveen dos elementos clave, los cuales son la habilidad de crear, probar y desplegar una API, así como la facilidad para conectar la aplicación con API de otros desarrolladores. A continuación, se explica a mayor detalle que es, y cómo funciona una API.

2.3.1. ¿Qué es una API?

Una interfaz de programación de aplicaciones (por sus siglas en inglés API), permite que compañías o usuarios particulares publiquen los datos y la funcionalidad de sus aplicaciones al exterior o terceros, dígase desarrolladores, socios de negocios o departamentos internos (IBM, 2020). Esto permite que los servicios y productos se comuniquen entre ellos y así se aprovechen los datos compartidos y funcionalidad a través de una interfaz (**Fig. 16**).

Los desarrolladores no necesitan conocer cómo es que una API se implementó. Ellos solo usan la interfaz para comunicarse con otros productos y servicios. El uso de las API ha surgido durante la última década, a tal grado que la mayoría de las empresas web más importantes y sus sitios no pudiesen ser posibles de no ser por éstas.

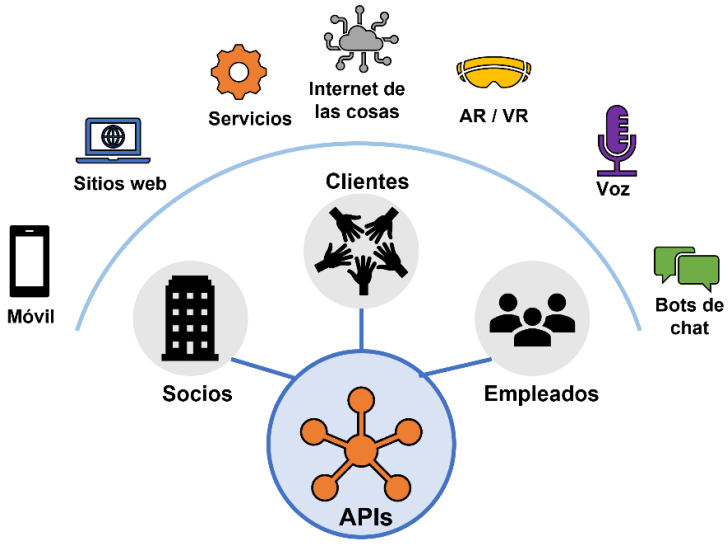

Fig. 16. Servicios ofrecidos por las API

2.3.2. Funcionamiento de una API

Una API es un conjunto bien definido de reglas que explican cómo las computadoras y aplicaciones se comunican entre sí. Las API se encuentran entre las aplicaciones y los servidores web, lo que significa que son intermediarias que procesan los datos compartidos entre los dos sistemas.

A continuación, se presenta como funciona una API (**Fig. 17**):

- **Una aplicación cliente inicia un llamado a la API:** Para recuperar información (conocido comúnmente como *request*), tal pedido se maneja desde una aplicación a un servidor web a través del Identificador Uniforme de Recursos (URI) de la API e incluye la solicitud, la dirección y, a veces, el cuerpo.
- **Después de recibir una petición válida:** La API hace un llamado, ya sea a un programa o aplicación externa o, al servidor web.
- **El servidor envía una respuesta:** Esta respuesta es enviada desde el servidor web a la API en un formato que esta pueda manipular de manera sencilla (comúnmente JSON).
- **La API transfiere los datos:** Los datos se envían a la aplicación que realizó la solicitud en primer lugar.

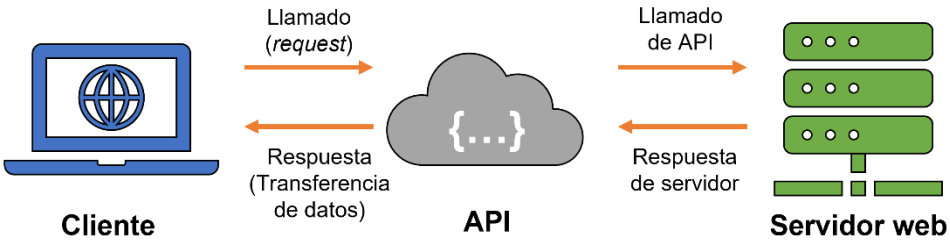

Fig. 17. Funcionamiento de una API

Aunque la transferencia de datos varía según el servicio web que se utilice, esta solicitud y respuesta se realiza a través de una API, dónde la interfaz usuario está diseñada por humanos, y las API son trazadas para su uso en una computadora o aplicación. Estas brindan seguridad porque su posición como intermediario facilita la abstracción funcional entre dos sistemas, debido a que una API desvinculan la aplicación que se consume desde la infraestructura que

provee el servicio. Las llamadas de la API generalmente incluyen la autorización de credenciales para reducir el riesgo de ataques al servidor. Además, los puertos de entrada de una API limitan el acceso para minimizar cualquier amenaza de seguridad. Durante el intercambio, el protocolo HTTP, las *cookies*, o parámetros *string* se enlistan para proveer capas adicionales de seguridad (**Fig. 18**).

El proceso que suele seguirse es que un servicio o una aplicación realiza una petición a la API. Este llamado se asegura con una llave de identificación. Al llegar al puerto de entrada, esta valida dicha llave haciendo uso de la base de datos que, en caso de que la validación se haya realizado satisfactoriamente, esta responde con una autenticación a la API. A continuación, la API hace una petición al servidor con la petición original realizada por la aplicación cliente. El servidor toma dicha petición y envía la respuesta adecuada de vuelta al puerto de la API. Entonces, esta envía una respuesta a la aplicación cliente. De esta manera se afianza la seguridad de los datos que pasan a través de una API.

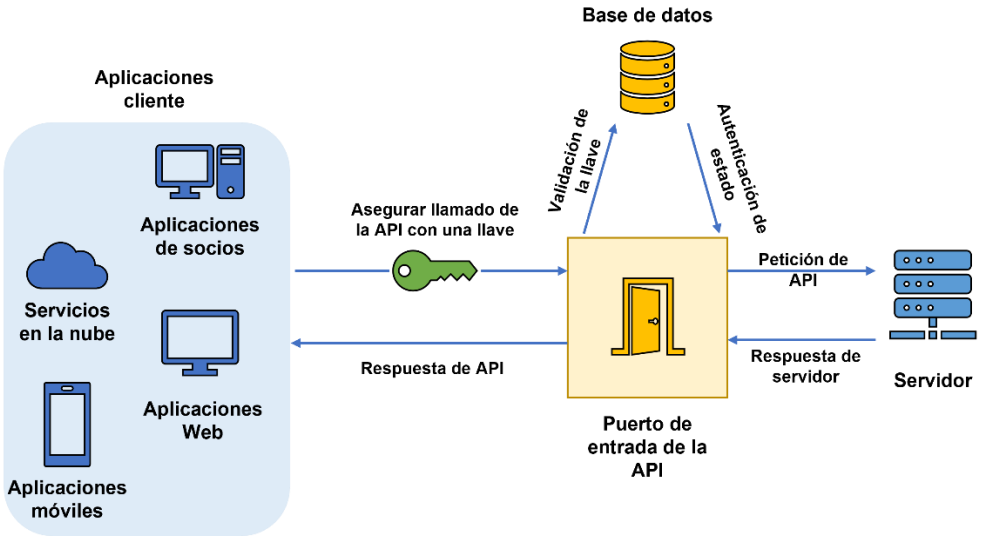

Fig. 18. Seguridad de una API

2.3.3. Beneficios de las API

Ya sea que esté administrando herramientas existentes o creando nuevas, puede usar las API para simplificar el proceso. Algunas de sus ventajas son:

- **Nivel de colaboración mejorado:** Las API permiten la integración de diferentes plataformas y aplicaciones y que puedan comunicarse entre sí sin problemas. En esta integración es posible automatizar y mejorar la colaboración con otros equipos de desarrollo.
- **Innovación más fácil:** Las API brindan flexibilidad, lo que permite a los desarrolladores conectarse con otros, ofrecer nuevos servicios y llegar a nuevos mercados. Esto aumenta la cantidad de retroalimentación que lleva a la transformación de la aplicación, ya que busca satisfacer con la mayor cantidad posible de necesidades de usuario.
- **Monetización de los datos:** Muchas compañías ofrecen API de forma gratuita, de esta manera adquieren una comunidad de desarrolladores alrededor de su marca, y así establecer relaciones con socios potenciales. No obstante, si la API brinda acceso a activos digitales valiosos, pueden monetizarla.
- **Seguridad añadida:** Una API crea una capa de protección entre los datos y el servidor. De esta manera, los desarrolladores pueden reforzar la seguridad mediante *tokens*, firmas y una cifrado de capa de seguridad de capa de transporte (TLS por sus siglas en inglés). Todo mientras implementan administradores de tráfico y validadores en los puertos de entrada de la API.

2.3.4. Aplicaciones comunes de API

Dado que las API permiten que una empresa o un grupo de desarrolladores brinden acceso a sus recursos mientras mantienen el control y seguridad, se han convertido en un activo valioso actualmente. A continuación, se explican las aplicaciones más comunes de las API.

- **Acceso Universal:** Esto permite que personas de todo el mundo se conecten a sitios como *Facebook*, *Twitter* o en cualquier servicio de *Google*. Esta característica permite que cualquier sitio web autentique rápida y fácilmente a través de los beneficios que brindan las API.
- **Procesamiento de pagos de terceros:** A medida que el comercio en línea se vuelve más popular, debe haber una forma segura y eficiente de realizar pagos. Las API

permiten a cualquier persona pagar productos en línea sin exponer o tener acceso a sus datos confidenciales.

- **Comparaciones de reservas de viajes:** Un aspecto del comercio digital es comprar boletos y reservar alojamiento. Los sitios de reserva de viajes ofrecen opciones sencillas para reservar vuelos y muestran las opciones más baratas para cada fecha y destino. Sin las API, este servicio no sería posible porque la aplicación proporciona al usuario la información más reciente sobre hoteles y aerolíneas. Esto reduce significativamente el tiempo y esfuerzo necesarios para realizar dichas reservaciones.
- **Aplicaciones de Mapas con GPS:** Un ejemplo común de buen uso de las API son servicios de mapas. Esto porque además de la API que muestra los mapas, estas usan otras API que proveen al usuario con direcciones y puntos de interés. Esto es posible a través de múltiples capas de datos y geolocalización.
- **Redes sociales:** Redes sociales como *Facebook* o *Twitter* permiten la comunicación a través de mensajes, que a menudo contienen atributos descriptivos básicos, cómo autor, número de identificación único, el mensaje, la hora en que se publicó y la ubicación de procedencia. Esto es posible gracias a la red de servicios API que utilizan.

2.3.5. Tipos de API

Actualmente, la gran mayoría de API son aplicaciones web que muestran la información de dicha aplicación, además de su funcionalidad en línea. Los 4 tipos de Aplicaciones web principales son las siguientes:

- **API abiertas:** Son aplicaciones de código abierto a las que se puede acceder a través del protocolo HTTP, también son conocidas como API públicas.
- **API Colaboradora:** Son API ofrecidas a colaboradores específicos, generalmente desarrolladores que pueden acceder a estas API en modo de semiservicio a través del portal de desarrolladores de API públicas.
- **API Internas:** Son API ocultas para usuarios externos. Estas API privadas no están disponibles para usuarios fuera del equipo de desarrollo y están destinadas a mejorar la productividad y comunicación a través de diferentes equipos de trabajo.

- **API Compuestas:** Combinan múltiples servicios y datos de API. Esto permite a los desarrolladores acceder a distintas terminales en un solo llamado. Las API compuestas son útiles en arquitecturas de microservicios donde realizar una tarea puede requerir información de distintas fuentes.

2.3.6. Tipos de protocolos de API

Con el desarrollo de las API web, se han desarrollado ciertos protocolos para proveer a los usuarios un conjunto específico de reglas que definen los tipos de datos y comandos aceptados. De hecho, estos protocolos facilitan el intercambio de información estandarizada. Los protocolos más utilizados son:

- **SOAP (Simple Objetct Access Protocol):** es un protocolo de API construido con Lenguaje de Marcado Extensible (XML), habilitando a los usuarios el enviar y recibir datos a través de un Protocolo Simple de Transferencia de Correo (SMTP) y HTTP. Con las API que usan SOAP, es más fácil compartir información entre aplicaciones y componentes de software que están siendo utilizados en diferentes ambientes de programación y escritos en distintos lenguajes.
- **XML-RPC:** Es un protocolo que se basa en un formato específico de XML para transferir datos, donde SOAP usa un formato propietario de XML. XML-RPC es más antiguo que SOAP, pero mucho más simple, y relativamente más ligero debido a que usa un mínimo de ancho de banda.
- **JSON-RPC:** Es un protocolo similar al anterior, ya que los dos son llamados que utilizan una computadora para ejecutar el código de manera remota, sin necesidad de preocuparse por las comunicaciones entre ellas. La diferencia es que, en este, el protocolo para transferir datos es JSON en lugar de XML. Ambos protocolos son simples, mientras que las llamadas pueden contener múltiples parámetros, solo se espera un resultado.
- **REST (Representational State Transfer):** Es un conjunto de principios de arquitectura de aplicaciones web. Esto significa que no son estándares oficiales (a diferencia de aquellos que tienen un protocolo). Para ser una REST API (También conocida como RESTful API), la interface debe adherirse a ciertas limitantes

arquitectónicas. Es posible construir RESTful API con protocolos SOAP, pero los dos estándares son visualmente vistos como especificaciones que compiten.

2.3.7. Servicios Web y Microservicios de API

Un servicio web es un componente de software al que se puede acceder a través de una dirección de internet. Por lo tanto, los servicios web requieren una red por definición. Dado que los servicios web muestran datos y funcionalidades de la aplicación, se consideran API. Sin embargo, no todas las API son servicios web.

Convencionalmente, API se refiere a una interfaz conectada a una aplicación que puede crear en cualquier lenguaje de programación de bajo nivel. Las API modernas se adhieren a las pautas de formato JSON y están generalmente diseñadas para HTTP, lo que resulta en interfaces amigables al desarrollador que son fácilmente accesibles y ampliamente entendidas por aplicaciones escritas en distintos lenguajes. Cuando se utilizan API, existen dos arquitecturas comunes:

- Arquitectura orientada a servicios (SOA)
- Arquitectura de microservicios

Si bien la arquitectura SOA es un paso esencial en el desarrollo de aplicaciones, la arquitectura de microservicios está diseñada a escala, brindando a los desarrolladores la flexibilidad que necesitan para crear, modificar, probar y desplegar aplicaciones a nivel granular.

2.3.7.1. Arquitectura orientada a servicios (SOA)

Es un estilo de diseño de *software* donde las características están divididas y se hacen disponibles como servicios separados en una red. SOA es implementada con servicios web, significando esto que los bloques funcionales se hacen accesibles mediante protocolos de comunicación. Los desarrolladores suelen hacer estos servicios desde cero, pero usualmente utilizan sistemas legados de otras interfaces de servicios.

2.3.7.2. Arquitectura de microservicios

Es un enfoque arquitectónico alternativo que divide las aplicaciones en componentes dependientes más pequeños. Implementando la aplicación como un conjunto de servicios separados facilita las pruebas, mantenimiento y la escala. Esta metodología ha cobrado protagonismo en la era de las aplicaciones en la nube, ya que permite a los desarrolladores trabajar en componentes independientes.

2.3.8. API y arquitectura de nube

Es necesario crear API que se adapten a las necesidades actuales. El desarrollo de aplicaciones nativas de la nube se basa en la conexión de microservicios a través de una API que comparte datos con usuarios externos.

Los servicios dentro de servicios utilizan un marco de desarrollo de mensajería común, como el que usan las RESTful API, lo que facilita la comunicación abierta con el sistema operativo, sin la molestia de integrar capas adicionales o transacciones de datos. Además, cualquier servicio o característica del servicio puede suspenderse, reemplazarse o mejorarse sin afectar negativamente otros servicios. Esta dinámica optimiza los recursos en línea, preparando el terreno para mejorar el rendimiento, las pruebas y escalabilidad de las API.

2.4. Computación afectiva

Las computadoras han comenzado a adquirir la capacidad de expresar y percibir sentimientos. La computación afectiva es la que intencionalmente se relaciona, surge o influye en las emociones (Rosalind W. Picard, 2000), y es un campo interdisciplinario que abarca las ciencias de la computación, psicología y ciencia cognitiva. Las tecnologías de computación afectiva “sienten” el estado emocional del usuario a través de periféricos como sensores, micrófonos, cámaras, etc. O lógica de software (BBVA Openmind, 2016).

La principal diferencia entre la computación afectiva y la teoría de emociones es que la teoría de emociones se centra en que son las emociones humanas y como es que se crean,

mientras que la computación afectiva involucra la implementación de las emociones a sistemas computacionales. Esto provoca que la computación afectiva sea un medio a través del cual se ayuda al desarrollo y prueba de la teoría de emociones. Sin embargo, la computación afectiva incluye más cosas, como otorgar a la computadora la habilidad de reconocer y expresar emociones, desarrollando así la capacidad de responder inteligentemente a las emociones humanas, permitiendo la regulación y uso de emociones.

La computación afectiva se traduce en métodos prácticos como el reconocimiento o síntesis de expresiones faciales, así como la síntesis de la inflexión de la voz, sin embargo, esto es solo la parte más conocida. Durante la década de los 80's Minsky dijo en *The Society of Mind*, "La pregunta no es si las máquinas pueden tener emociones, sino el si las máquinas pueden ser inteligentes sin emociones" (Minsky, 1985). Ahora se ha establecido que las emociones son una parte activa de la inteligencia, especialmente la percepción, el pensamiento racional, la toma de decisiones, la planeación y la creatividad. Estas emociones son tan importantes para las interacciones sociales, que los psicólogos y educadores han redefinido la inteligencia para incluir emociones y habilidades sociales.

2.4.1. Emociones viscerales y cognitivas

Los humanos a menudo son conscientes de sus sentimientos, y los experimentos y estudios de laboratorio nos han enseñado que la evaluación cognitiva puede preceder a la generación de emociones (Picard, 2000).

Una forma útil de clasificar las emociones generadas es de manera cognitiva y no cognitiva. Esto parte de la idea de que hay ciertas características de los estímulos emocionales que existen y a los que se responde de manera "Primaria". Estos sentimientos están en el sistema límbico, que es el encargado de dirigir las emociones y la conducta. Las emociones que se definen como "secundarias" son aquellas que se desencadenan después en el desarrollo del individuo. Para la estructura límbica, las emociones secundarias no son suficientes. El cuerpo a menudo responde a las emociones de manera física.

2.4.2. Modelos de estados afectivos

Los estados afectivos se definen con base en la valencia y la activación. La valencia indica el grado en que un estado emocional se considera placentero (valencia positiva) o no placentero (valencia negativa). En cuanto a la activación, se refiere al grado en que un estado afectivo es vivido en términos enérgicos. Una alta energía psico-fisiológica (activación alta) o una experiencia limitada de dicha energía (activación baja) (Madrid H., 2016).

La combinación de las dimensiones de valencia y activación afectiva (**Fig. 19**) describe cuatro grandes estados afectivos, los cuales son:

- **Entusiasmo:** Experimenta un estado de valencia positiva y de activación alta. El entusiasmo conlleva placer y alta energía.
- **Tranquilidad:** Vivencia de un estado de valencia positiva y de activación baja. La tranquilidad denota placer y energía limitada.
- **Ansiedad:** Sufre de un estado de valencia negativa y de activación alta. La ansiedad implica displicencia y energía alta.
- **Depresión:** Experiencia de un estado de valencia negativa de activación baja. Al sentirse deprimido conlleva displicencia y energía limitada.

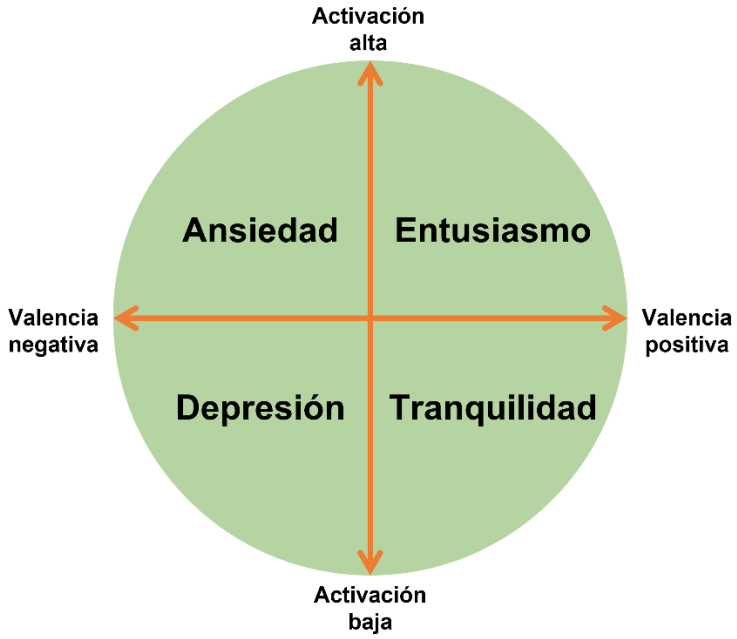

Fig. 19. Modelo de estados afectivos (Madrid, H. P., 2013)

2.4.3. Casos de uso de la computación afectiva

La computación afectiva puede ser una herramienta útil para la IA en gran variedad de casos de uso, (Dilmegani, C., 2022). Estas aplicaciones incluyen:

- ***Marketing:*** Muchos proyectos ayudan a las empresas a mejorar su inversión en *marketing* al permitir el análisis de las emociones del cliente.
- **Servicios al cliente:** Ya sea en los *call centers* o en comercios minoristas, permite que las compañías estimen las emociones del cliente, estas estimaciones son utilizadas para ayudar a comprender cómo responde el servicio al cliente y, por ende, medir la eficacia.
- **Industria de la salud:** Al usar *Wearables* (Un dispositivo conectado que se puede llevar puesto) con habilidad de detectar emociones tales como la empatía, han sido usados por los investigadores para estudiar acerca del estrés, autismo, epilepsia y otros desordenes.
- **Seguridad e identificación de fraudes:** El uso de un dispositivo biométrico ayuda a proteger el uso de algunas aplicaciones que contienen información personal confidencial, tales como las aplicaciones de banco.

Capítulo 3

3. Estado del arte

Esta sección presenta una revisión de la literatura realizada por varios investigadores sobre el tema de este proyecto. Se realizó una búsqueda, lectura y análisis de numerosas fuentes bibliográficas que se encuentran relacionadas con el tema de este trabajo. También se examinaron artículos estrechamente relacionados con el tema de este trabajo. Gracias a esta investigación, se puede definir con mayor precisión el alcance de este trabajo y derivar el flujo de trabajo más realista con el conjunto de actividades necesarias para llevarlo a cabo. Esta sección se divide en la creación de una API, la definición de un formato de archivo, el preprocesamiento de datos y como proporcionar un modelo ML con servicios a través del servidor.

3.1. Creación de API

Dado que se trata de una aplicación ofrecida como servicio, esta debe contar con una infraestructura API que permita publicarla a disposición de los usuarios. LA API requiere una arquitectura específica para permitir la conexión para formar repositorios, tanto de algoritmos de ML como de DL. Para ello, es necesario investigar trabajos que puedan ser de utilidad.

En Miura et al. (2017), se establece una infraestructura novedosa que realice el análisis de contenido autorizado (por sus siglas en inglés ASCA), misma que es una de las principales necesidades, y con esta arquitectura recopilar información. La gran diferencia con este trabajo es que Miura decide diseñar su arquitectura utilizando contenedores de Docker para migrar los servicios. Esta infraestructura propone proveer API comunes para las comunidades inteligentes desplegándolas en contenedores Docker, ya que estas permiten contener librerías y software necesario para la ejecución de procesos y servicios, siendo uno de sus beneficios el desempeño. Esta arquitectura cuenta con capas de conexión entre el cliente y la API y, además una capa para manejar el contenedor. Al concluir con esto, resulta en la posibilidad de correr hasta 8 servicios distintos sin tener una disminución en el

rendimiento y, con el uso de contenedores se mejora la portabilidad de servicios, incluyendo la migración de redes. Esto resulta en que la plataforma es capaz de transferir los datos compartidos a los procesos de servicios.

Haselböck et. al. (2018) propone un método para identificar y organizar potenciales opciones de diseño relacionadas con los conceptos que se esperan utilizar. La idea principal es usar microservicios para separar el software en un conjunto de servicios desacoplados que soporten el desarrollo, despliegue y operación independiente, proveyendo de flexibilidad de desarrollo al trabajo. Todo lo anterior permite identificar los espacios de diseño posibles para el manejo de API en sistemas basados en microservicios y captura de decisión de modelos. También presenta el proceso de los casos de uso básicos para este tipo de desarrollos.

Éste identifica y analiza el espacio de diseño, se realiza la toma de decisiones y se validan el acercamiento con la documentación pertinente. También se demuestra que el espacio de diseño puede ser manejado paralelamente, y por último propone la guía de creación de API tomando en cuenta el anterior análisis. Esto fue mostrado en pruebas de campo, dónde se identificaron las áreas de diseño para organizar las tareas en microservicios, estos no solo son un conjunto de decisiones, sino la retroalimentación en la utilidad del acercamiento para el análisis de la documentación.

La propuesta realizada por López García et. al. (2016) consta de una arquitectura distribuida para proveer un conjunto de herramientas para los usuarios de ML en un servicio disponible en la nube, y que cubra todo el ciclo de desarrollo de este. Dicha arquitectura se define como un marco de desarrollo denominado DEEP, ya que permite desarrollar y entrenar modelos de ML de forma distribuida. Esto es debido a que ofrece acceso transparente a las infraestructuras de almacenamiento a partir del desarrollo de una API estandarizada para modelos de ML, así permite que expongan su funcionalidad. También proporciona una ruta sencilla para desplegar los modelos desarrollados como servicios y para esto se han construido herramientas que permiten el compartimiento efectivo de modelos de ML ya que el marco de desarrollo DEEP está diseñado para desplegar modelos de aprendizaje automático distribuidos en arquitecturas. A partir de esto resulta que este trabajo prepara el camino para que las infraestructuras electrónicas que adoptan las técnicas de ML y DL

desarrollen servicios en base a estas mismas técnicas. Se otorgaron un conjunto de herramientas para los desarrolladores de ML y DL.

En su trabajo Olston et. al. (2017) presenta el uso de *TF serving*, como un sistema para almacenar los modelos de ML y DL haciendo uso de un servidor dentro de *Google*, disponible a cualquier desarrollador ya que este es un desarrollo de código abierto. Este desarrollo es muy flexible en término de tipo de plataformas de ML y su soporte, con las distintas maneras de integrar los sistemas que comprometen los modelos nuevos con versiones actualizadas. El estado actual del proyecto al momento de su publicación es que está siendo utilizado dentro de *Google* y, es posible que este soporte varios sistemas internos. El número total de proyectos de *Google* usando *TensorFlow serving* es de cientos, probando sus beneficios en cuestión de flexibilidad.

Lebre et. al. (2015) detalla la extensión de *SimGrid*, que consiste en un conjunto de herramientas versátiles para realizar la simulación de sistemas de cómputo distribuidos de gran escala, y teniendo capacidades de almacenamiento. Este propone la implementación haciendo uso de dos computadoras interconectadas a través de una red, un disco está acoplado a cada una de estas, pero la computadora principal también se encuentra comunicada con el disco de la secundaria y esto implica una comunicación entre operaciones de entradas y salidas. Como resultado se expone el primer acercamiento que se realizó para el primer paso para la simulación de sistemas de almacenamiento extendiendo el conjunto de herramientas que ofrece *SimGrid* con distintos modelos, abstracciones e interfaces de programación, y para archivos o componentes de almacenamiento. En el trabajo se exponen también las limitaciones de la extensión y se logró caracterizar el comportamiento del desempeño de varios equipos de almacenamiento. Y como último aporte, se listan los casos de uso pertinentes a diferentes iteraciones de infraestructuras de cómputo distribuido, así como los beneficios de dichas propuestas.

Lo que expone Li Erran Li et. al. (2016), de *Uber Technologies*, es que los sistemas de ML como servicios (MlaaS) son necesarios para el éxito de muchas compañías, debido a que necesitan poder procesar una gran cantidad de datos (denominados *big data*). En este trabajo se presenta el MlaaS desarrollado para *Uber* que opera de manera global, dónde se enfocan en distintos problemas que representa la escalabilidad de este tipo de servicios. Las

soluciones propuestas son diseños e implementaciones de características escalables de motores de cómputo y almacenamiento de estas características. También un marco de desarrollo para manejar y entrenar una jerarquía de modelos y la automatización de despliegue a un clic. Se concluye con que la implementación de modelos de ML escalables se puede construir como una plataforma para empresas transnacionales, al contar con muchos casos de uso. También aplican algoritmos de DL incluyendo visión por computadora y NLP.

La problemática anteriormente expuesta también es abordada por Ribeiro et. al. (2015), dónde se estudia que las necesidades de extracción de información han evolucionado. A medida que aumenta la cantidad de datos que se generan a partir de redes sociales o aplicaciones móviles, por dar ejemplos, las empresas e investigadores necesitan conectar todos estos datos y extraer la información más valiosa. En este artículo se propone una arquitectura para crear MlaaS flexible y escalable. Se buscó implementar una solución de código abierto tomando como caso de uso series de tiempo de demanda eléctrica y el clima, ambos usándolos en distintos algoritmos al mismo tiempo. Como resultado se obtuvo una arquitectura que permite a los usuarios a acceder a modelos, entrenarlos, probarlos y crear predicciones y ofrecer comparativas del uso de tres diferentes modelos.

En el trabajo realizado por Wang et. al. (2018) se aborda el desarrollo y presentación de un sistema para proveer entrenamiento y servicios de inferencia de modelos de ML para facilitar el procesamiento de análisis complejos en plataformas en la nube a través de una aplicación que denominaron *Rafiki*. Esta provee parámetros distribuidos para un servicio de entrenamiento, un modelado de conjuntos en línea para el servicio de deducción que, hace una transacción entre latencia y exactitud. Los resultados experimentales confirman la eficiencia, efectividad, escalabilidad y confiablidad del uso de *Rafiki*, esto utilizando varios puntos de referencia. También se utilizó un caso de estudio que demuestra cómo es que el sistema permite que un desarrollador de bases de datos use servicios de DL de manera sencilla.

3.2. Identificación de formato

Para que este proyecto funcione de manera óptima, la aplicación necesita seleccionar los diferentes tipos de archivos para seleccionar el modelo ML o DL que mejor se adapte a las

necesidades del usuario. Se han investigado varios trabajos relacionados con la identificación de formatos de archivo. Es fundamental conocer las diferentes técnicas que se pueden utilizar para diferenciar datos y determinar cuál se adapta mejor a las necesidades de este proyecto. A continuación, se presenta el trabajo previo que se fue revisado.

Las declaraciones hechas en Dhanalakshmi et. al. (2009) establecen que el formato de archivo, visto como la forma en que los datos se encuentran organizados, necesita que los programas que usan dichos archivos requieren reconocer y posiblemente acceder a estos datos. Comúnmente la extensión está separada del nombre del archivo. Para esta extracción es necesario la implementación de una herramienta para obtener una mayor exactitud al momento de identificar los tipos de archivos. Se propone el uso de los sistemas UNIX, ya que utiliza el comando *file* para identificar los tipos de archivos. Este utiliza tres métodos distintos para intentar identificar los parámetros. El primer método usa el sistema de información para reconocer los archivos de sistemas; el segundo usa los datos encontrados en los primeros 16 *bits* de un archivo; y el tercer método utiliza el contenido ASCII dentro de los archivos para categorizarlos. Se propone el método para la detección de tipos de archivos basándose en varios parámetros (cómo los títulos) y, en el caso de que estos no estén presentes, considerar los contenidos de los archivos y aplicar la técnica de modelado estadístico.

En el trabajo de Fetherston et. al. (2012), se expone que la preservación digital actualmente utiliza un variado conjunto de herramientas y procesos automáticos para identificar y validar objetos digitales, llámese archivos o datos. Esto es vital en la preservación de estos, pero los resultados de las herramientas más utilizadas para este cometido carecen de transparencia y no se pueden verificar fácilmente. En este artículo se sugiere que un corpus de prueba de objetos digitales es una forma de proveer verificación y validación, mejorando la fiabilidad en las herramientas. Este trabajo se centra un tanto más en aquellos criterios que se deben tomar en cuenta para identificación de archivos por selección de sus contenidos. Se concluye que el desarrollo del corpus que propone este trabajo provee el aseguramiento de calidad en los métodos de identificación de formato de archivos y de las herramientas que muestran carencias en este aspecto.

Para concluir este tema, se revisó el trabajo de Kuppili Venkata et. al. (2020), donde se dice que la identificación de formato de archivos es un paso necesario para una preservación digital efectiva. Debido a que esto permite tomar las acciones apropiadas para acceder a los tipos de archivo. Este trabajo se encuentra centrado principalmente en archivos de texto, utilizando la técnica de identificación por fragmentos, también llamada técnica de recuperación, o usando la identificación por contenido o estructura binaria. Para lograr lo propuesto es necesario crear un corpus de archivos que ayude al algoritmo a identificar el formato del archivo partiendo de la estructura de sus datos. También se implementó una metodología con base en ML. Esto resulta en un prototipo desarrollado con una exactitud razonablemente buena a la hora de detectar los formatos de distintos archivos.

3.3. Preprocesamiento

Para que una API tenga éxito en obtener predicciones de los modelos de ML y DL, debe haber un preprocesamiento que responda automáticamente a las necesidades de procesamiento de cada modelo propuesto. Para ello, se han estudiado diversos trabajos sobre técnicas de preprocesado. Las obras estudiadas se presentan a continuación.

En cuanto a técnicas de preprocesamiento, una variedad es presentada en el trabajo de Alam et. al. (2019), estás técnicas corresponden a cada tipo de datos y se aborda el impacto de estas técnicas en términos de exactitud en análisis de sentimientos. Esta actividad se realiza utilizando clasificadores de ML bien conocidos. Se calcula la exactitud en estos algoritmos y como es que esta mejora con la aplicación de distintos tipos de preprocesamiento. Este trabajo prueba que algunos algoritmos se benefician considerablemente dependiendo de la técnica de preprocesamiento aplicada a sus datos, afianzando su calidad y por ende obtener mejores resultados.

El trabajo de Oldham et. al. (2020) revisa y discute como es que la aplicación de los pasos de preprocesamiento tiene un impacto positivo en reducir el error en aplicaciones relacionadas con la detección de movimiento. En este trabajo se concluye que distintos acercamientos para corregir el error en el movimiento tienen distintos resultados. Mientras que algunas aplicaciones resultan en una contaminación severa de los datos, otras mitigan de manera exitosamente el error que suelen presentar este tipo de algoritmos. Con dicha

información se pueden garantizar algoritmos basados en detección de movimiento de una mayor calidad.

Lo que se establece en la investigación de Zhe Zhu (2017) es que la forma en que se obtienen imágenes de series de tiempo en la última década ha cambiado completamente el uso de estos datos. Este presenta una revisión de cuatro aspectos importantes de estudios de detección de cambio con base en series de tiempo, incluyendo frecuencias, preprocesamiento, algoritmos y aplicaciones. Se observó una tendencia, que consiste en que entre más reciente es un estudio, las series de tiempo son utilizadas más frecuentemente. En este artículo también revisan una serie de pasos para preprocesado de imágenes, incluyendo técnicas de composición, fusión y métrica. Como resultado se clasificaron los algoritmos de detección en seis categorías, incluyendo umbralización, diferenciación, segmentación, clasificación de trayectoria, límites estadísticos y regresión.

La conexión entre el big data y el preprocesamietno de datos es abordado en el trabajo de García et. al (2016). Este incluye un amplio rango de disciplinas, como la preparación, reducción, selección y discretización de los datos en el procesamiento. También habla de distintas técnicas a manera de acercamientos, tales como descargar las instancias que contienen valores perdidos, pero reconoce que el aplicar este tipo de prácticas no son benéficas en todos los casos. Este trabajo resultó en la categorización actualizada de las contribuciones de preprocesado de datos bajo el marco de desarrollo de *big data*. Esto cubre diferentes familias de técnicas de preprocesamiento de datos, tales como la selección de características, datos imperfectos, aprendizaje poco balanceado y reducción de instancias, así como los pesos máximos soportados y los marcos en los que fueron desarrollados. También se hizo notar cuales son los problemas clave en el preprocesamiento de *big data*.

En la investigación realizada por Fan et. al. (2021) se revisan trabajos existentes en el área de preprocesamiento de datos y lo expone como un paso indispensable en construir análisis de datos funcionales considerando que el crear este tipo de datos conlleva sus deficiencias intrínsecas. En este trabajo se considera el peso del preprocesamiento para analizar conjuntos de datos muy grandes. Se revisa una basta variedad de técnicas de preprocesado de datos, y a su vez propone tres técnicas de preprocesado de datos para abordar algunos desafíos prácticos en manejo de datos. El trabajo concluye en las ventajas y

desventajas de los métodos de preprocesamiento existentes y sus posibles aplicaciones. Esto es útil para el desarrollo de investigación orientada al tratamiento y construcción de datos.

Se estudió un artículo que trata sobre el preprocesamiento de archivos de Audio. En Choi et. al. (2018) se investiga de manera empírica el efecto del preprocesado de audio en cómo es que la música se etiqueta con redes neuronales profundas. Se realizan experimentos comprensivos que involucran el preprocesado de sonido usando diferentes representaciones de frecuencia de tiempo (usando diferentes espectros como STFT y LOG), compresión de magnitud logarítmica, pesado de frecuencias y escalado. Se muestra que una variedad de técnicas de preprocesamiento es redundante, a excepción de la compresión de magnitud. El resultado de este artículo consiste en demostrar cómo es que la forma de preprocesamiento de los datos de entrada afecta de manera activa el desempeño de las predicciones.

3.4. Redes neuronales

Las redes neuronales son modelos simples (ya sean de ML o DL) y representativos de la forma en que funciona el sistema nervioso. Este cuenta con unidades fundamentales denominadas neuronas y, comúnmente están organizadas en capas. Para que este proyecto funcione de la manera adecuada, es necesario adaptar la API para el uso de redes neuronales, ya que, de acuerdo con estas, será la capacidad de procesamiento que tenga la aplicación. Para este trabajo fueron elegidas redes LSTM, CNN tahn y CNN RestnetLike. Para asegurar el correcto uso de estas, se han investigado trabajos previos, mismos que se presentan a continuación.

En Ma et. al. (2019) se investiga la información capturada por las redes LSTM es muy útil para mejorar el reconocimiento de emociones multimodales, usando una variedad de señales psicológicas. Debido a esto, este artículo propone una red LSTM residual multimodal para reconocimiento de emociones. Lo que se realizó es que la red compartiera los pesos a través de distintas modalidades en cada capa de la LSTM y crear una correlación con las señales psicológicas del usuario. Esto es posible debido a la eficiencia que muestran este tipo de redes al trabajar con características de alto nivel relacionadas al aprendizaje emocional. Al terminar el trabajo, se obtuvo que esta red es muy prometedora, ya que tiene una exactitud de clasificación del 92.87%, lo cual es un valor muy alentador para futuras investigaciones.

También se revisó el artículo de Huang et. al. (2014), dónde se expone el uso de sistemas de DL, tales como las CNN, y como es que estos son capaces de inferir la representación jerárquica de datos de entrada, lo cual facilita la categorización. Este trabajo propone el reconocimiento de personalidad a partir de reconocimiento del habla usando una CNN. Esta red se entrena en dos pasos, primero las pruebas sin etiquetar son utilizadas para aprender características candidatas y, en el segundo paso se usan las entradas de la CNN para aprender y discriminar características afectivas. Los resultados de este trabajo consisten en la obtención de un conjunto de datos que permite conocer que el acercamiento utilizado funciona para un desempeño robusto y estable de reconocimiento de emociones en escenarios distintos.

Fan et. al. (2016) presenta un sistema de reconocimiento de emociones basado en vídeo. La parte central de este sistema es una red híbrida que combina RNN y redes convolucionales de 3 dimensiones (C3D). La RNN toma las características extraídas por una CNN a partir de cuadros individuales de un vídeo y los utiliza como entradas para codificar el movimiento. Mientras que la C3D modela la apariencia y el movimiento en el vídeo de manera simultánea. Todo lo anterior combinado con un módulo de audio. Este trabajo resultó en una exactitud media de 59.02%, lo cual es una mejora considerable de trabajos anteriores.

El artículo realizado por Jiang et. al. (2017) expone que, aunque la detección de objetos basada en DL ha mejorado durante los últimos años, gran cantidad de los acercamientos para reconocimiento facial aún están basados en el marco de desarrollo de R-CNN. Este trabajo se da a la tarea de investigar la aplicación de R-CNN más rápidas, que han mostrado resultados más alentadores en la misma tarea. Al entrenar una R-CNN más rápida, se reportan resultados con el conjunto de datos *WIDER*, lo cual sugiere que la efectividad de estas redes viene del módulo de red de propuesta regional (RPN por sus siglas en inglés), debido a que comparte capas convolucionales con el módulo de detección de la CNN.

Capítulo 4

4. Metodología de la API

Este capítulo presenta la explicación de los mecanismos y herramientas utilizados para el análisis de la problemática de investigación. Aquí se documentan y monitorean los hechos relevantes ocurrido durante el período de desarrollo del proyecto. A continuación, se define el nombre con el que se denominó este trabajo.

4.1. APINET: Modelos de ML y DL como servicios

El proyecto de APINET (**Fig. 20**) consiste en una API que ofrezca un repositorio de modelos de ML y DL. Para lograr esto fue desarrollada una metodología que permite implementar dichos modelos como servicios web disponibles para cualquier usuario. Lo que APINET ofrece es un desarrollo que permite que los usuarios no necesiten conocer cómo utilizar desarrollos de ML y DL, ofreciendo una variedad de algoritmos automatizados que facilitan su aplicación. Estos van desde aquellos que sirven para diferenciar el tipo de dato cargado, hasta el tener un algoritmo que automatice el preprocesamiento de estos para aplicarse en modelos de ML y DL.

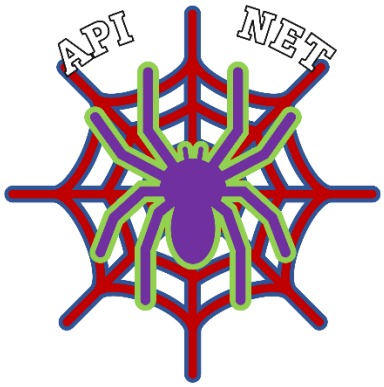

Fig. 20. Logotipo de APINET

4.2. Documentación de la API

Una API permite que las aplicaciones estén disponibles como un servicio público sin necesidad de conocer la forma de implementarlas. De esta manera, el desarrollo web se simplifica. Además, las API brindan flexibilidad y simplificación en el diseño, la administración y el uso de estas aplicaciones.

Para la realización de este proyecto se usó la librería *flask*, en conjunto con un patrón de cliente servidor, dónde el *backend* (la parte de una aplicación que está fuera del alcance del usuario final) se encuentra directamente comunicado con el modelo de repositorios que almacena los modelos, los algoritmos de preprocesamiento y los repositorios de intérpretes. En la **Fig. 21**, se aprecia que en el elemento Repositorio, se encuentra el repositorio físico de modelos de ML, repositorio de intérprete, y repositorio de preprocesamiento. Todos estos se comunican con el elemento de estructuras de modelo, dónde se hacen los llamados necesarios de cada uno de los repositorios mencionados anteriormente. Repositorio se comunica con la Interfaz Gráfica de Usuario (*GUI* por sus siglas en inglés) que se encuentra en el *frontend*. La GUI se comunica con la API en el elemento Server (la cual se encuentra en el *backend* de *flask*). La API se encarga de ofrecer todo lo anterior como un servicio local. Esta API se comunica con una Interfaz de Entrada de Servidor Web (*WSGI* por sus siglas en inglés) que actúa de puente con ayuda de un *DNS* Dinámico (Sistema de Nombres de Dominio) hasta comunicarse con el Cliente a través de un navegador Web, dónde el usuario puede cargar sus archivos.

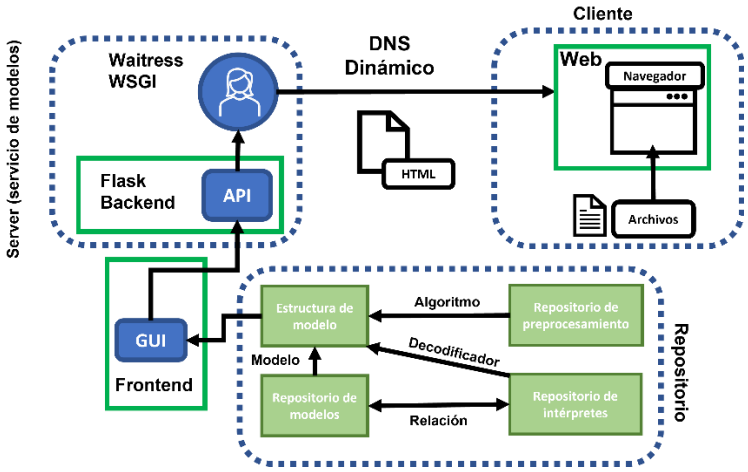

Fig. 21. Vista de desarrollo de la API

4.2.1. Análisis de requisitos

El análisis de requisitos es una tarea prioritaria en la ingeniería de software donde se analizan y definen los procesos que describen el desarrollo de la API. Esto permite especificar las características de rendimiento. Para este análisis se detalla lo que debe hacer el sistema desde un punto de vista más coloquial, teniendo en cuenta las necesidades del usuario. Sobre la base de este análisis y los requisitos específicos, se han categorizado los requisitos funcionales y de calidad, los cuales se basan en los requisitos del usuario.

Debido a que es un servicio público, APINET debe cumplir con ciertas exigencias de funcionalidad en el sistema, para que el usuario pueda utilizarlo. Dichos requisitos se presentan a continuación en la **Tabla 1**.

Tabla 1. Tabla de requisitos de usuario

Requisitos	Descripción
RU-01	Se debe permitir al usuario utilizar las distintas redes disponibles en el repositorio de modelos de ML y DL sin limitaciones.
RU-02	El usuario es capaz de elegir el tipo de dato que aplicará en las redes disponibles.
RU-03	El servicio debe estar disponible para acceder desde cualquier navegador web las 24 horas del día, los 365 días del año.
RU-04	El sistema que el usuario cargue archivos desde la plataforma que se esté utilizando.
RU-05	El usuario debe recibir los datos debidamente preprocesados por la red ML o DL que se haya elegido.
RU-06	El sistema debe ser gratuito.
RU-07	El sistema debe contar con una interfaz intuitiva en la que el usuario pueda identificar la funcionalidad de este de manera rápida y sencilla.

4.2.1.1. Requisitos funcionales

Esta sección presenta los requisitos funcionales del sistema APINET. Estos describen el comportamiento específico de ese sistema cuando se cumplen ciertas condiciones. La definición de estos está relacionada con las necesidades del usuario, ya que la funcionalidad del sistema debe satisfacer las necesidades del usuario establecidas previamente. Los requisitos funcionales son especificados en la **Tabla 2**.

Tabla 2. Tabla de requisitos funcionales

Requisitos	Descripción	Prioridad
RF-01	El sistema debe tener las redes de ML y DL debidamente referenciados en su repositorio para ser usadas cuando se realice su llamado. (RU-01, RU-03 y RU-06)	Media
RF-02	El sistema debe soportar como todo tipo de formato de archivo utilizado para aprendizaje máquina (RU-02)	Alta
RF-03	El sistema ofrece los distintos tipos de redes disponibles en el repositorio de modelos de ML y DL, y que estos puedan utilizarse sin ninguna limitante. (RU-02)	Alta
RF-04	El sistema debe entregar los datos procesados en un formato útil para el usuario. (RU-05)	Media
RF-05	El sistema permite la búsqueda y carga de archivos para utilizarlos con las redes de ML y DL. (RU-04)	Media

4.2.1.2. Requisitos de Calidad

Los requisitos de calidad son los encargados de determinar las características cualitativas que debe ofrecer el programa. Se dividen en atributos de calidad, es decir, disponibilidad y usabilidad. Se dividen de esta manera para su clasificación y es posible priorizarlos de manera que satisfagan las necesidades en términos de requisitos del usuario. En la **Tabla 3** se expone la forma en que estos fueron especificados.

Tabla 3. Tabla de requisitos de calidad

Requisitos	Atributo de calidad	Descripción	Prioridad
RC-01	Disponibilidad	El sistema debe estar disponible para acceder desde cualquier navegador web las 24 horas los 365 días del año de manera gratuita. (RU-03 y RU-06)	Media
RC-02	Usabilidad	El sistema debe mostrar una interfaz de fácil uso en que los usuarios puedan encontrar la utilidad de cada una de las opciones en un tiempo corto (no más de 10 segundos). (RU-07)	Alta
RC-03		El servicio permite el uso de las redes propuestas ofrecidas dependiendo del dato ingresado al sistema (RU-01, RU-02)	Alta
RC-04		El usuario recibe los datos introducidos debidamente procesados (RU-05)	Alta
RC-05		Se puede realizar la búsqueda de los datos de la forma en que al usuario le parezca más conveniente. (RU-04)	Media

4.2.2. Actores

Los actores se definen como entidades externas al sistema que se relacionan con él y que requieren ciertas funciones. Esto incluye usuarios, pero también los sistemas fuera del programa. Los actores considerados para APINET son el usuario, una interfaz gráfica de usuario, repositorio de preprocesamiento, repositorio de modelos y repositorio de intérpretes. En la **Tabla 4 – 4** se describe cada uno de estos y como es que interactúa con el sistema y, en la **Fig. 22** se muestra cómo es que estos se relacionan.

Tabla 4. Tabla de actores

Actor		Descripción
Usuario		Es quien utilizará el servicio de aprendizaje máquina introduciendo los datos que desea utilizar y recibiendo estos en un formato deseado.
Base de datos	Repositorio de preprocesamiento	Dirección física donde se encuentran los distintos algoritmos de preprocesamiento.
	Repositorio de modelos	Dirección física donde se encuentran los distintos modelos de aprendizaje máquina disponibles en el servicio.
	Repositorio de intérpretes	Dirección física dónde se encuentran los algoritmos que definen las relaciones entre los tipos de preprocesamiento y los modelos.

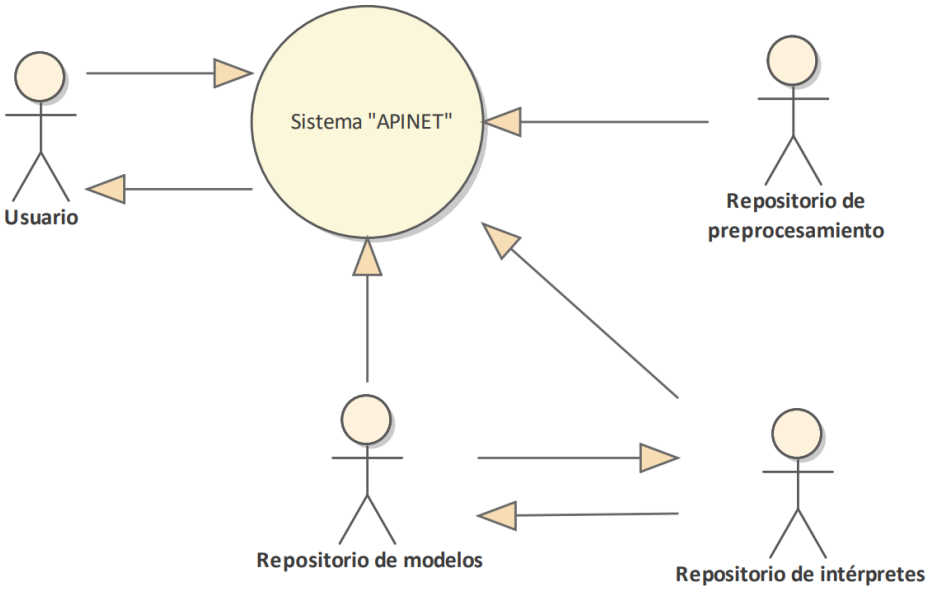

Fig. 22. Relación de Actores de APINET

4.2.3. Casos de uso

Los casos de uso describen las actividades que deberán realizarse para llevar a cabo algún proceso mostrando la funcionalidad principal del sistema. Éstos se definieron con base en el análisis de requerimientos funcionales con la finalidad de establecer los mecanismos que le dan interacción al sistema. En la **Tabla 5** y en la **Fig. 23** se pueden observar las relaciones que guardan los actores con los casos de uso.

Tabla 5. Tabla de Casos de Uso

Identificador	Nombre	Descripción
CU001	Carga de datos	El sistema permite que el usuario cargue cualquier dato en el servicio de ML y DL. (RF-02, RF-05)
CU002	Elección de tipo de red a utilizar	Dependiendo del tipo de dato introducido en el sistema, se elige entre las distintas redes que pudiesen utilizarse para procesarlo. (RF-01, RF-03)
CU003	Preprocesamiento de datos	Proceso que se realiza de manera automática al elegir el tipo de red que se utilizará. (RF-04)
CU004	Entrega de datos procesados	Proceso que se realiza una vez los datos han sido debidamente preprocesados y se han procesado en una red de ML y DL. (RF-04)

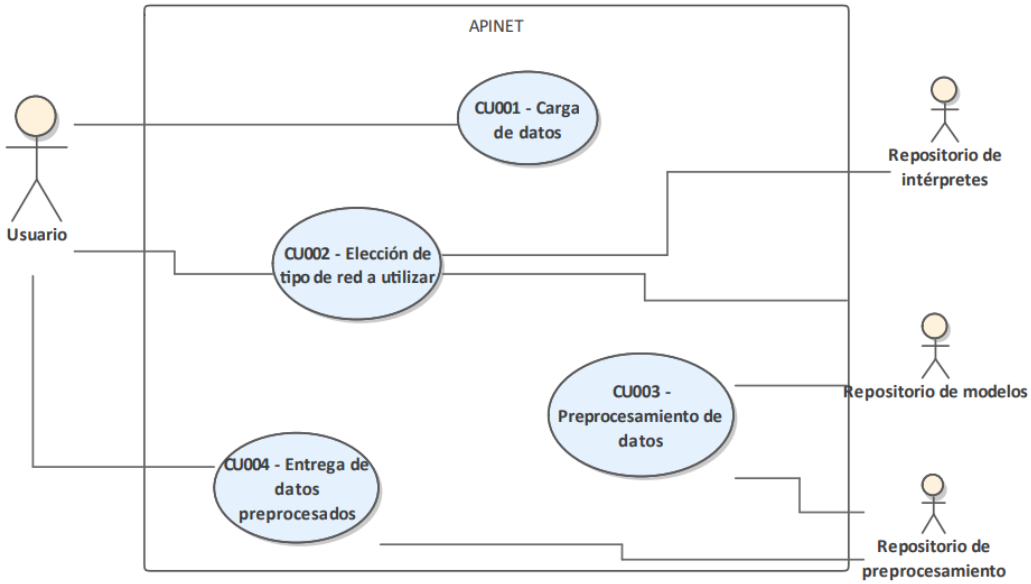

Fig. 23. Diagrama de Casos de Uso

4.2.4. Diagrama de contexto

El diagrama de contexto se refiere a un diagrama que define el límite entre un sistema o parte de un sistema y su entorno, mostrando las entidades que interactúan con él, y a menudo se dice que es un sistema de vista de nivel superior.

Como puede observarse en la **Fig. 24**, el sistema APINET es utilizado por el usuario, que será quien realice las peticiones a este. Para poder responder a dichas peticiones, APINET hace uso de distintos componentes, tales como la interfaz gráfica que es la que permite la comunicación usuario – sistema. También hace uso de varios repositorios, tales como el de modelos de ML y DL que contiene los algoritmos para procesar datos, el de algoritmos de preprocesamiento y el repositorio de intérpretes. Por último, el sistema depende de su base de datos, dónde se encuentran los distintos algoritmos y permite una gestión efectiva de los mismos.

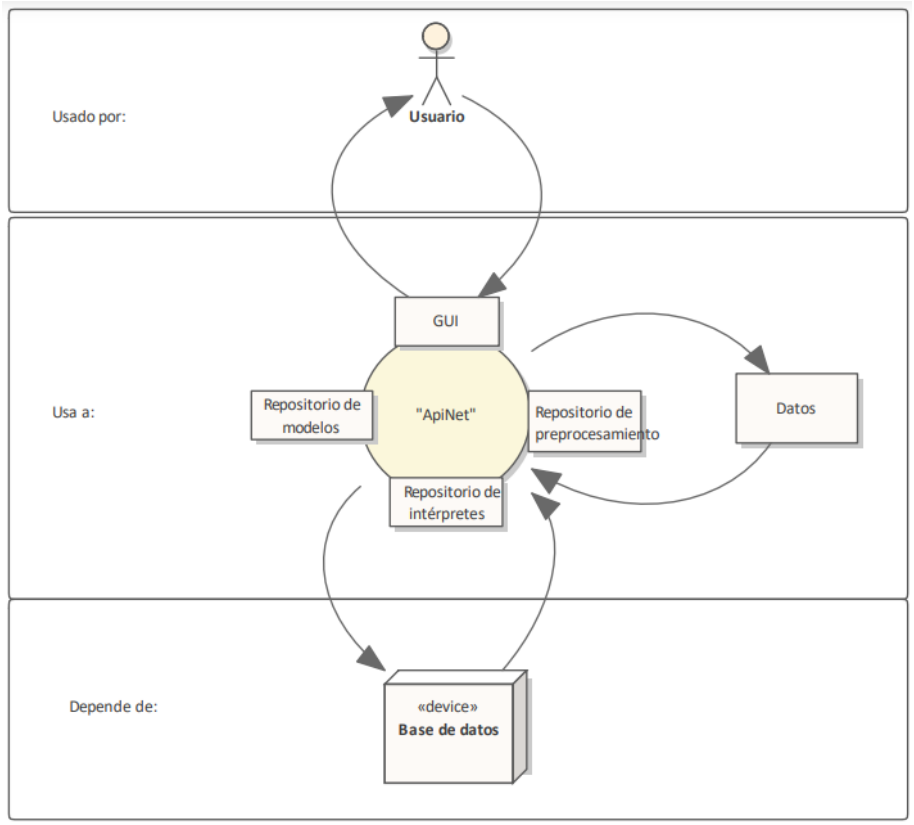

Fig. 24. Diagrama de Contexto

4.2.5. Arquetipos

Los arquetipos son entidades abstractas identificadas que representan las funcionalidades más importantes de un sistema. Estos se identifican mediante preguntas como ¿cuáles son las partes claves del sistema? Y ¿Qué representan estas entidades en el sistema? En la **Tabla 6** se describen los arquetipos seleccionados y una breve descripción.

Tabla 6. Arquetipos de APINET

Arquetipo	Descripción
Usuario	Es aquel que utilizará la aplicación web, y hará uso de las distintas funciones que el sistema ofrece.
Datos	Es un conjunto de archivos utilizados en los distintos modelos de aprendizaje máquina y aprendizaje profundo para obtener predicciones.
Modelo	Es un algoritmo que permite al usuario procesar los datos.
Interfaz	Se trata de la representación del sistema que permite que el usuario interactúe con el de manera sencilla e intuitiva.
Preprocesamiento	Es la preparación de los datos para que estén en el formato correcto que permita ser utilizados en los modelos de aprendizaje máquina y profundo
Intérprete	Define las relaciones entre los modelos, los tipos de datos a los cuales estos pueden ser expuestos y el preprocesamiento necesario para que la información sea utilizable.
Servicio	Es el que permite que el sistema esté disponible para cualquier usuario que desee utilizarlo.

Relación de arquetipos

En este apartado se muestra cómo es que los arquetipos previamente expuestos se relacionan en el sistema, así como la manera en que estos interactúan, representados en la **Fig. 25**. En esta figura se muestra como el usuario se relaciona con el sistema, realizando peticiones a través de la interfaz. Esta interfaz carga los datos que el usuario proporciona y realiza la petición al intérprete, el cual permite elegir el modelo que mejor se ajusta al tipo de dato y lo relaciona con el algoritmo de preprocesamiento adecuado para el modelo elegido. El algoritmo de preprocesamiento envía los datos con el formato correcto para ser utilizados a el modelo, que a su vez envía los resultados al servicio. Dicho servicio proporciona los datos a la interfaz que hace entrega de los resultados esperados al usuario.

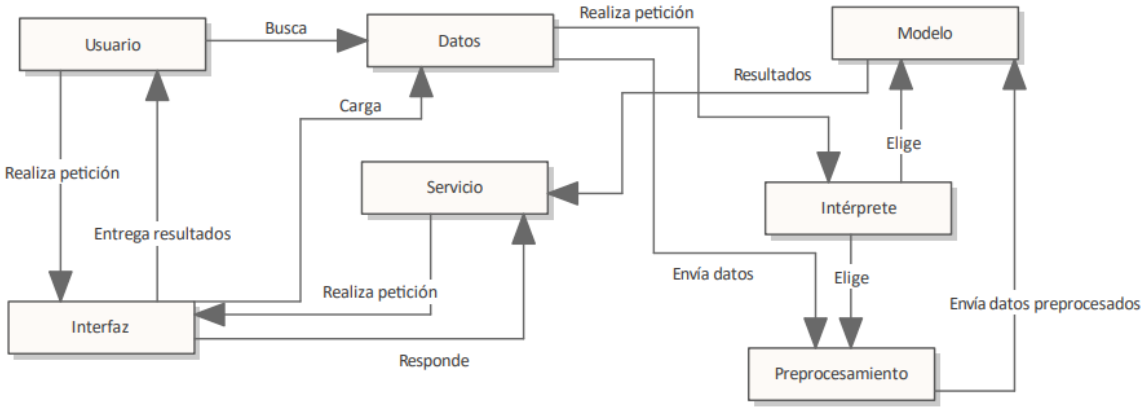

Fig. 25. Relación de arquetipos de APINET

4.2.6. Modelo arquitectónico

La arquitectura de software a menudo se conoce como el diseño de nivel superior de la arquitectura de un sistema, y consiste en un conjunto de patrones y abstracciones que brindan un marco claro para interactuar con el software. Este, a su vez, define de manera abstracta la arquitectura de los componentes que realizan una tarea, sus interfaces y la comunicación entre ellos. La arquitectura de software se selecciona y diseña de acuerdo con las necesidades y limitaciones (directamente relacionadas con la tecnología disponible).

La arquitectura elegida es una mezcla entre el modelo de servicio web y el de repositorios ya que sus características cubren las necesidades del sistema. Algunas de las características que resultan convenientes son que el modelo de repositorios permite que los subsistemas intercambien información a través de una base de datos ya que todos estos se gestionan en un repositorio central accesible por todos los componentes. Los componentes no interactúan directamente, solo mediante el repositorio, lo que hace que este sistema sea el más adecuado para aplicaciones donde hay datos que se generan por un subsistema y utilizados por otro subsistema. En el caso del modelo de servicio web, este proporciona la infraestructura necesaria para que un usuario (o consumidor) se comunique con el sistema APINET a través de un registro de servicio. Estas características lo vuelven una forma muy eficiente de compartir grandes cantidades de datos donde las acciones de seguridad (copias de seguridad, protección, control de acceso y recuperación de errores) se encuentras

centralizadas. Una desventaja que tiene este modelo es que se dificulta la redistribución de datos del repositorio, ya que al estar centralizado lógicamente hablando, suele presentar problemas con redundancia de datos e inconsistencias.

4.2.7. Capas Lógicas

En las capas lógicas de un sistema se muestran los componentes de software de una aplicación distribuida de manera que puedan representadas como miembros en dichas capas. Estas representan la independencia lógica de los componentes de software según la naturaleza de los servicios que estas prestan. Dichas capas son las siguientes (**Fig. 26**):

- **Capa de usuario:** En esta capa se presenta lo que el usuario ve en el sistema, así como la comunicación e interacción (intercambio y captura de información) entre éstos.
- **Capa de Negocio**: Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y a su vez lo hace con la capa de datos, para solicitar el gestor de base de datos el almacenar o recuperar datos.
- **Capa de Datos:** En está capa se almacenan los datos del sistema. Está formada por uno o más gestores de bases de datos que realizan el almacenamiento de toda la información, además, recibe solicitudes de almacenamiento, acceso y recuperación de información directo desde la capa de negocio.

4.2.8. Componentes

Los componentes de software proporcionan las funciones de los patrones del sistema de software. Estos pueden configurarse y definir las interacciones entre los mismos (**Fig. 25**). Estos realizan tareas específicas que cumplen con la funcionalidad del sistema, y en este caso particular, los componentes del sistema se seleccionan en base a requisitos funcionales y de calidad, plantillas predefinidas y diagramas de contexto. Los componentes que se definieron para el correcto funcionamiento del sistema son los siguientes:

- **Interfaz Gráfica:** Permite a todos los usuarios el interactuar de manera dinámica a través de una interfaz web en dispositivos de computadoras, solicitando todas las peticiones que el usuario considere necesarias y el sistema le permite realizar.
- **Manejador de memoria:** Es el componente encargado de dar manejo a las solicitudes de tal forma que la capacidad de la memoria del servicio no se sature.
- **Servicio:** Es el componente dónde se ofrece el uso de los otros componentes.
- **Repositorio de modelos:** Componente donde se encuentran los distintos modelos de aprendizaje máquina disponibles en el servicio.
- **Repositorio de intérpretes:** Componente dónde se encuentran los algoritmos que definen las relaciones entre los tipos de preprocesamiento y los modelos.
- **Repositorio de preprocesamiento:** Componente donde se encuentran los distintos algoritmos de preprocesamiento.

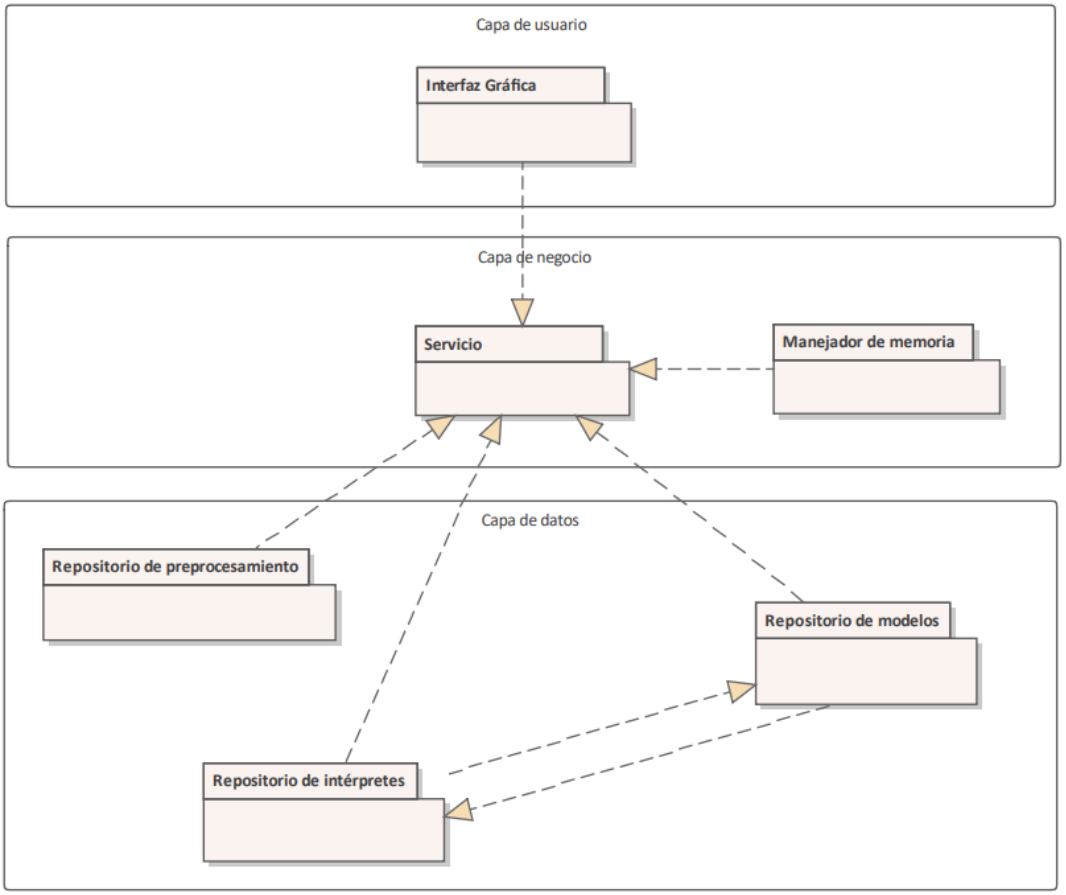

Fig. 26. Relación de componentes de APINET

4.2.9. Tablas de mapeo

Una tabla de mapeo es el conjunto de declaraciones que recopilan información de uno o de múltiples conjuntos en un solo esquema. A partir de este se pueden crear listas y derivar vistas. El mapeo de datos es una parte esencial del buen funcionamiento de muchas operaciones de datos, ya que es una parte esencial de cualquier proceso de integración de datos. En la **Tabla 7** es mostrada la tabla de mapeo dónde se relacionan los casos de uso con los componentes del sistema, esto para tener información más certera de que componentes actúan en ciertas situaciones, siendo de gran ayuda para conocer las necesidades de los componentes y sus limitantes.

Tabla 7. Tabla de Casos de –so - Componentes

Caso de uso	Descripción	Interfaz Gráfica	Repositorio de modelos	Repositorio de intérpretes	Repositorio de preprocesamiento	Manejador de memoria	Servicio	Base de datos relacional
CU001	Búsqueda de datos	✅					✅	✅
CU002	Carga de datos	✅				✅	✅	✅
CU003	Elección de tipo de red a utilizar	✅	✅	✅		✅		✅
CU004	Preprocesamiento de datos				✅		✅	✅
CU005	Obtención de datos preprocesados				✅			

En la **Tabla 8** se expone la tabla de mapeo que relaciona los Requisitos funcionales y los casos de uso, que, al igual que la **Tabla 7** permite observar de una manera más clara dichas relaciones. En este caso conocer la forma en que se enlazan dichos conceptos ayuda a delimitar mejor la forma en que el sistema actuará tomando en cuenta las diversas situaciones a las que este puede ser expuesto.

Tabla 8. Tabla de Requisitos – Casos de uso

Requisito	Descripción	Caso de uso
RF-01	El sistema debe mantener la disponibilidad de las redes durante las 24 horas del día los 365 días del año de manera gratuita.	CU001 CU002 CU003 CU004 CU005
RF-02	El sistema debe soportar como mínimo todo tipo de formato de archivo utilizado para aprendizaje máquina (RU-02)	CU002
RF-03	El sistema debe tener en sus opciones los distintos tipos de redes que pueden utilizarse y que las mismas sean sugeridas dependiendo del tipo de dato. (RU-03)	CU003
RF-04	El sistema debe ofrecer los datos procesados en un formato útil para el usuario. (RF-06)	CU004 CU005
RF-05	El sistema permite que se realicen búsquedas por archivo o por directorio. (RF-05)	CU002
Rf-06	La interfaz de usuario debe mostrar las distintas opciones de redes y de búsqueda de archivos.	CU001

4.3. Diseño de flujo de trabajo

En este apartado se describe el flujo de trabajo que fue diseñado para desarrollar APINET (**Fig. 27**). Dichos pasos fueron clasificados de la siguiente manera:

- **Delimitación de la API:** Esta actividad permite definir las especificaciones, tanto las funcionales como las de calidad de la aplicación web.
- **Elección de herramientas:** Paso en el cual se elige el lenguaje de programación, IDE (*Integrated Development Environment*) en que la aplicación se desarrolló, así como las librerías necesarias para su implementación
- **Definición de la estructura:** Se encarga de establecer la arquitectura que delimita la forma en que los módulos de la aplicación se comunican para llevar a cabo la tarea esperada.
- **Programación de API:** Actividad en que se engloban las actividades de programación del algoritmo, tales como realizar el análisis del código, creación de módulos de preprocesamiento, módulo de automatización e implementación de las librerías.
- **Aseguramiento de calidad:** Paso en que se realizan las actividades necesarias para asegurar la calidad de la aplicación, esto se engloba en dos actividades iterativas:

- **Pruebas:** Paso en que se engloban todas las actividades pertinentes a realizar pruebas de calidad a la aplicación, dónde se obtendrán las áreas de mejora de esta.
- **Correcciones funcionales:** Paso en que se realizan las correcciones a partir de los resultados de la etapa de pruebas. Después de esto se realiza otro período de pruebas, y de esta manera garantizar el buen funcionamiento de la aplicación.

- **Publicación de la API como servicio:** Una vez garantizada la calidad de la aplicación web, la API se ofrece como un servicio público disponible en línea para cualquier usuario que desee hacer uso de este.

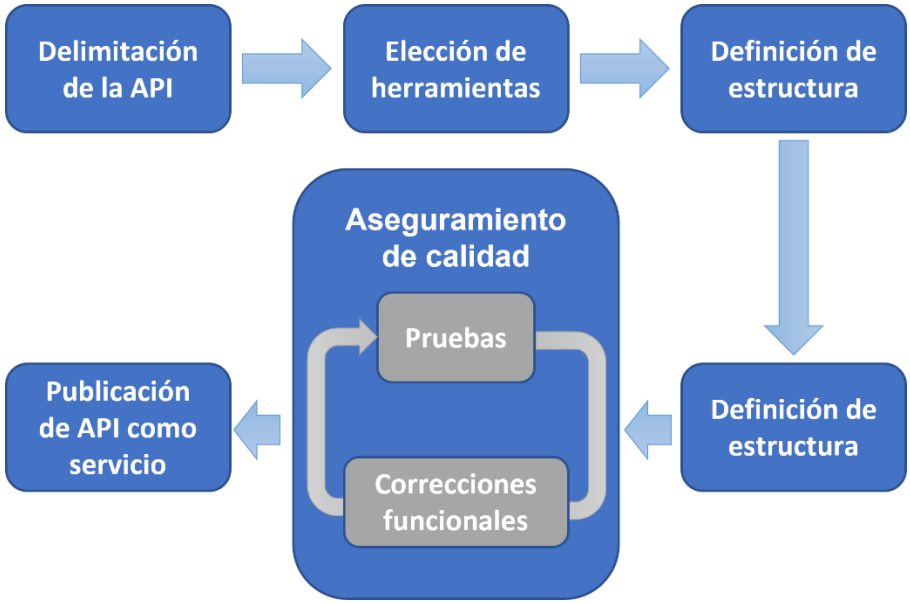

Fig. 27. Flujo de trabajo de APINET

4.4. Herramientas

Para llevar a cabo este proyecto, fue necesario el uso de distintas herramientas de desarrollo. En este caso el *IDE* que mejor se adaptó a los requerimientos de la implementación fue el de *Pycharm*. Una vez elegida la *IDE*, se definió el lenguaje de programación a utilizar, en este caso el lenguaje de programación que brinda mayores facilidades es *Python*, ya que cuenta con un gran número de bibliotecas disponibles que resultan útiles para la creación de APINET. Dichas bibliotecas contienen paqueterías de preprocesamiento y uso de modelos

de ML y DL, que hacen que el sistema sea flexible, y así cubrir las necesidades del usuario de una manera óptima.

De igual manera se hizo necesario el uso de un software capaz de visualizar de manera gráfica las redes de ML, y de esta manera tener la información de referencia necesaria para dar el formato requerido de los archivos a procesar. Para cumplir con dicha necesidad se utilizó la herramienta *Netron*, la cual es un visualizador de modelos de redes neuronales, DL y ML. En esta se muestran las capas de un modelo a manera de bloques y permite observar las especificaciones de cada una. (Netron, 2022)**.**

4.4.1. Bibliotecas

En las ciencias de la computación una biblioteca es una colección de recursos usados por los programas de computadores, comúnmente utilizadas en el desarrollo de software. Dichas colecciones pueden incluir configuración de datos, documentación, información de ayuda, plantillas, código, subrutinas, clases, valores o especificaciones de tipo. Estas también pueden referirse como una colección de implementaciones de comportamiento escritas en términos de un lenguaje que tiene una interfaz bien definida a través de la cual dicho comportamiento es invocado.

Al desarrollar este trabajo fueron utilizadas 3 bibliotecas principales. Estas cumplen el cometido de desplegar la aplicación a modo de un servicio público disponible en internet, así como el que todos los módulos de esta funcionen de manera óptima. Dichas librerías son:

- **Flask:** Es un *microframework* (un marco de desarrollo que tiene poca o ninguna dependencia de librerías externas) o un módulo de *Python* que permite el desarrollo de aplicaciones web y está basado en el kit de herramientas Werkzeg WSGI y la plantilla *Jinja*. Flask apunta a mantener el *core* simple pero escalable, dando suficiente flexibilidad para ser utilizada con otras librerías o herramientas (Flask, 2022). El uso dado a esta librería es el de proveer las herramientas, librerías y tecnologías necesarias que permiten construir la *API* de una manera práctica.
- **Waitress:** Waitress es un servidor una Interfaz de Entrada a Servidores Web (o por sus siglas en inglés WSGI). Este está basado61epend*thon* con una calidad de

producción y rendimiento aceptables ya que no tiene dependencias, excepto aquellas que se encuentran en la librería *standard*62epend*thon* (Waitress, 2022). Este es una especificación que describe la comunicación entre servidores web y aplicaciones o marcos de desarrollo de Python y de esta manera se explica cómo es que el servidor web se comunica con la aplicación web (APINET) y el marco de desarrollo (*Flask*) y como es que estas están encadenadas para procesar una petición **(Fig. 28)**.

- **Tensorflow:** Esta es una librería de código abierto para el desarrollo de algoritmos de aprendizaje computacional y fue desarrollado por Google para satisfacer las necesidades de sistemas que fuesen capaces de construir y entrenar modelos de redes neuronales que realicen reconocimiento de patrones y correlaciones (Tensorflow, 2022)**.** En el caso particular de este trabajo, el uso de *TensorFlow* es para montar los modelos de ML de tal forma que sus funciones y predicciones puedan ser llamados por la aplicación web para hacer uso de estos.

Fig. 28. Comunicación de *Waitress* con la aplicación web

4.5. Algoritmo de selección, uso de modelos y preprocesamiento

Para que la API funcione como se espera, es necesario el uso de distintos algoritmos que permitan seleccionar y utilizar los distintos modelos de ML y DL disponibles en la aplicación. También es necesario un algoritmo que realice el preprocesamiento necesario para que los datos introducidos en la aplicación sean transformados a un formato utilizable por los modelos. En este apartado se detalla cada uno de estos algoritmos.

4.5.1. Algoritmo para diferenciación de datos

El tipo de dato es un atributo asociado a una pieza de información que le dice al software como interpretarlo. El entendimiento de los tipos de datos asegura que la información es recolectada en el formato asignado y el valor de cada propiedad es el esperado (Choudhury, 2022)**.** El algoritmo de diferenciación de tipo de dato consiste en la automatización del reconocimiento de tipo de archivo a utilizar en un modelo de ML o DL. Partiendo de este, se busca identificar qué tipo de archivo se desea procesar, siendo esto de vital importancia, ya que dependiendo del resultado que arroje la aplicación de este algoritmo será posible identificar cual es la red idónea para utilizar para el archivo en cuestión. Para esto se estudiaron distintas técnicas posibles. Dichas técnicas son:

- **Identificación de tipo de archivos desde fragmentos de archivo:** Es un método comúnmente utilizado para identificar archivos corruptos a partir de datos y partes faltantes de su contenido ya que es principalmente una técnica de recuperación de datos. Su uso para identificación fue considerado ya que es posible obtener el tipo de dato a partir del reconocimiento de estos fragmentos sin necesidad de que el archivo se encuentre íntegro (Venkata, K., 2020).
- **Identificación de tipo de archivo basado en contenido (Estructura binaria):** Es un método más robusto y preciso, y parte del principio de extracción de características de los archivos. Se usa la frecuencia de la correlación entre *bytes* y el análisis de la firma de archivo (Goldman & Goldman, 2007).
- **Identificación de tipo de archivo por extensión:** El formato de un archivo es la manera en que los datos se encuentran organizados en términos del archivo. Cualquier programa que utilice los datos de un archivo debe ser capaz de reconocer y acceder a los datos dentro del mismo. Convencionalmente la extensión se encuentra separada del nombre del archivo, y contienen el identificador del formato, por lo tanto, esta técnica consiste en separar dicho identificador del nombre del archivo para obtener su tipo (Abraham, 2009).

Para el desarrollo del algoritmo de diferenciación de tipo de dato, se tomó en cuenta la técnica de identificación de archivo por extensión, ya que esta y sus convencionalidades son las aptas

para el objetivo de este trabajo. La extensión del archivo se toma a partir de la ruta de origen de este, y separando el formato de este y almacenándolo en una variable independiente que sirve para la selección de red que es compatible con este tipo de dato (**Fig. 29**).

Fig. 29. Selección de red por identificación a través de extensión

4.5.2. Algoritmo para selección de red

Una vez obtenido el tipo de dato, esto partiendo de la separación de extensión del archivo, es necesario que sea elegida una red en función al archivo que se busca utilizar, ya que no todas las redes son compatibles con todas las redes de ML o DL. Para lograr que la red sea elegida, se implementa un algoritmo comparador, dónde se discierne la extensión del archivo con aquellos formatos compatibles con los distintos algoritmos de ML disponibles en la *API*. Esto redireccionará los datos al método de preprocesamiento pertinentes a este (**Fig. 30**).

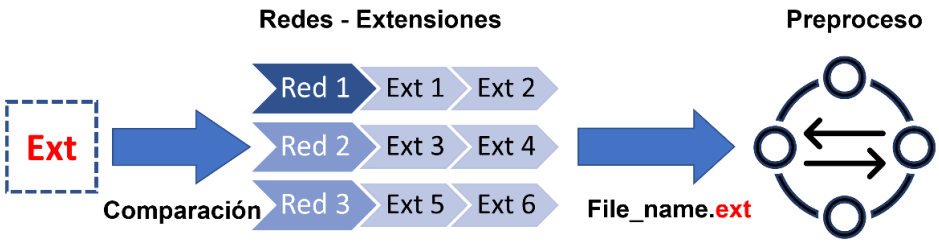

Fig. 30. Proceso de selección de tipo de preprocesamiento

4.5.3. Algoritmo de elección de preprocesamiento

Se hace llamar preprocesamiento de datos al conjunto de técnicas usadas para una aplicación de datos en algoritmos de ML, y es conocido por ser uno de los problemas más significativos en el procesamiento de datos (García et al., 2016) **.** Cuando se trata de la creación de modelos

de ML, el preprocesamiento de datos es lo primero a tomar en cuenta para la iniciación del proceso (Goyal, 2021). A menudo los datos se encuentran incompletos, son inconsistentes, inexactos, y suelen carecer de los valores de atributos específicos. Al aplicar el preprocesamiento de datos ayuda a limpiar, formatear y organizar los datos en crudo, y de esta forma tenerlos listos para su aplicación en un modelo de ML. Los pasos de preprocesamiento de datos se resumen en 5 pasos, los cuales son:

- **Evaluación de calidad de los datos (Data Quality Assesment):** Esto incluye los acercamientos estadísticos necesarios para asegurar que los datos a utilizar no tengan problemas. Los componentes tomados en cuenta para la calidad de los datos son:
    - La totalidad sin valores de atributos faltantes
    - Exactitud y confiabilidad de la información
    - Consistencia de las características
    - Mantener la validez de los datos
    - No contiene ninguna redundancia

  El proceso de evaluación de calidad involucra tres actividades principales:
    - **Perfilado de datos:** involucra explorar los datos para identificar los problemas de calidad de estos. Una vez que el análisis se ha realizado, los datos deben resumirse de acuerdo con que no se identifiquen duplicados, valores en blanco, etc.
    - **Limpieza de datos:** involucra arreglar problemas de los datos.
    - **Monitoreo de datos:** Encargado de mantener los datos en estado óptimo y hacer revisiones continuas, de tal forma que las necesidades de aplicación se vean satisfechas por los datos.
- **Limpieza de datos (Data Cleaning):** Realizado para mantener los datos limpios, rellenando valores faltantes, suavizando el ruido, resolviendo inconsistencias y removiendo valores atípicos.
- **Integración de datos (Data Integration):** Realizado para unir los datos presentes en múltiples fuentes en un solo conjunto de datos.
- **Transformación de datos (Data Transformation):** Una vez realizada la limpieza se debe consolidar la calidad de los datos en formas alternas cambiando sus valores,

estructura o formato y usando alguna estrategia de transformación de datos (Ej. Generalización, normalización, selección de atributos, agregación).

- **Reducción de datos (Data Reduction):** Proceso realizado solo cuando el tamaño del conjunto de datos es demasiado grande para manejarlo por los algoritmos de análisis. Se opta por obtener la representación reducida del conjunto de datos y que produzca resultados analíticos de igual calidad y para esto puede utilizarse una variedad de estrategias de reducción (Ej. Agregación del cubo de los datos, reducción de dimensionalidad, compresión de datos, discretización, reducción de la numerosidad, selección de subconjunto de atributos).

Para realizar la obtención de los parámetros de entrada necesarios para la capa de entrada de las distintas redes, estas requirieron ser visualizadas a través de *Netron App*. Esto permite observar cuales son las necesidades en términos de dimensiones y tipos de archivos (**Fig. 31**), de igual manera como es que la red requiere referenciar el archivo que se va a procesar.

Fig. 31. Visualización de entradas de una red

Acto seguido es necesario hacer un redimensionamiento de los archivos a los cuales se les aplica el preprocesamiento, y así hacerlas compatibles a la red que vaya a ser utilizada. Para esto es necesario importar *numpy* (Numpy, 2022), el cual es un paquete fundamental para ciencias computacionales contenida en *Python*. Esta es utilizada para obtener tanto forma como tamaño del dato. Los valores que contiene la dimensión de la capa de entrada del

modelo de ML o DL se separan en variables independientes y, estas mismas son utilizadas para hacer un redimensionamiento a los datos. De esta manera se asegura que la dimensión es la necesaria para realizar el preprocesamiento (**Fig. 32**).

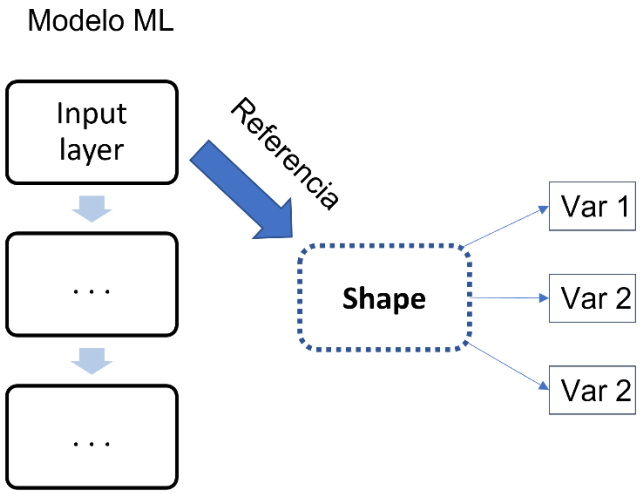

Fig. 32. Obtención de variables para redimensionamiento

Para preprocesar datos existen distintas técnicas, en las que se involucra la transformación de datos crudos a un formato entendible. Al hacer este trabajo se consideraron varias técnicas de preprocesamiento, hasta encontrar las que son aptas para dichas actividades. Debido a la naturaleza de este proyecto, dónde los datos necesitan cambiar y adaptarse a las distintas redes disponibles en la aplicación, la técnica que se decidió utilizar es la de transformación de datos. Esta técnica consiste en transformar los datos en forma y dimensión apropiada para realizar el procesado en los modelos de ML y DL.

4.5.3.1. Bibliotecas para preprocesamiento

Para realizar el preprocesamiento de los archivos, es necesario el uso de distintas bibliotecas para adaptar los distintos tipos de archivos a los valores de entrada que requiera cada una de las redes. Para preprocesar los archivos de audio se utilizan las siguientes librerías:

- **Wave:** Esta librería provee una interfaz útil para transformar archivos de formato WAV (Wave, 2022).

- **Speaker Verification ToolKit:** Proporciona las herramientas necesarias para verificar voz de una manera sencilla (Speaker-Verification-Toolkit, 2022).
- **Librosa:** Es una librería para análisis de audio y ofrece los bloques necesarios para crear sistemas de recuperación de audio (Librosa, 2022).

Mientras que para hacer preprocesamiento de imágenes y videos se utilizaron las siguientes librerías:

- **Numpy:** Esa librería proporciona objetos de arreglos multidimensionales, matrices y una variedad de subrutinas para operaciones rápidas en arreglos, incluyendo la manipulación de forma y dimensión, clasificación, operaciones estadísticas básicas y simulación aleatoria.
- **OpenCV:** Es una librería de código abierto para visión por computadora, ML y DL. Esta fue diseñada para ofrecer una infraestructura común para aplicaciones de visión artificial (OpenCV, 2022).
- **Face Recognition:** Librería que permite reconocer y manipular rostros, construida en base a algoritmos de reconocimiento (Face recognition, 2022).

4.6. Creación de la API

Para crear una API es necesario hacer uso de todo lo expuesto anteriormente, este proceso es presentado a continuación.

4.6.1. Estructura de la API

Antes de crear la aplicación, es necesario crear la estructura de la API (**Fig. 33**). Esto consiste en los tipos de archivos requeridos y cómo se almacenan. Es necesario crear un método principal que llame a los algoritmos que componen la aplicación en su totalidad. Dado que se está utilizando *Flask* como herramienta de desarrollo para esta aplicación web, es necesario seleccionar la aplicación de acuerdo con sus términos. También es necesario crear una carpeta para archivos estáticos y su contenido (por ejemplo, imágenes para la aplicación web).

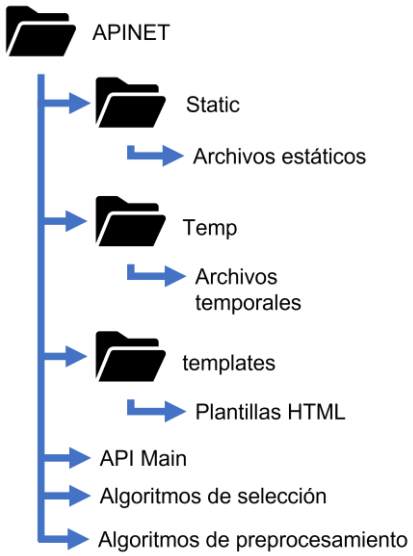

Fig. 33. Estructura de los archivos de la API

4.6.1.1. Plantillas HTML

Las plantillas HTML son un mecanismo de almacenamiento en caché para páginas HTML que no deben cargarse inmediatamente después de acceder a una aplicación web, sino que deben configurarse en tiempo de ejecución. Esto se puede considerar como una forma de almacenar estas páginas hasta que el cliente vea necesario utilizarlas.

Para esto, es necesario crear un repositorio físico para dichas plantillas, donde se ubican las interfaces de la aplicación web. Estas contienen los métodos que se pasarán a las rutas en función de las solicitudes de los usuarios (por ejemplo, Página de inicio, Acerca de, Datos de salida, Predicción y funciones de carga de archivos).

4.6.1.2. Creación de rutas

Las rutas definen qué páginas y aplicaciones redirigir (por ejemplo, ejecutar cualquier tarea de Python, mostrar páginas HTML, peticiones y respuestas). Estas rutas se crearon en la instancia del método principal.

Es obligatorio el importar nuestra aplicación web, y de esta manera ligar las rutas a las tareas del programa, así cuando se realicen las peticiones, el algoritmo le redirigirá a la

ruta en la función que debe ser usadas. También se encargan de enviar las respuestas a el usuario (cliente).

4.6.1.3. Despliegue de modelo de aprendizaje máquina

Es necesario montar un modelo previamente entrenado (como se ha hecho referencia con anterioridad). Se utiliza referenciando el módulo de preprocesamiento, que realiza todo el proceso de transformación de datos para que estos sean compatibles con el tipo de modelo de ML. Es posible discernir cual es necesario debido al uso de la función de automatización de elección de modelo tomando en cuenta la extensión del archivo que se ha cargado. Así se crea todo un proceso automatizado de carga, preproceso, procesado y obtención de resultados. Para la obtención de resultados se regresa un arreglo llamado *prediction* que interactúa con la aplicación web, el cual después de recibirlo se lo proporciona al usuario para su uso.

4.6.2. Comunicación Servidor-Internet

Para que la aplicación web pueda ser visualizada desde el exterior (Internet), se vuelve necesario configurar el módem (*router*) con el servidor. Para llevar a cabo esto es necesario configurar los puertos pertinentes (estos puertos han sido asignados con anterioridad en el código de la API) a abrirlo de tal forma que el servicio web se encuentre disponible desde cualquier lugar a través de la red.

4.6.2.1. Asignación de un DNS dinámico

Un Sistema de Nombre de Dominio Dinámico (o DNS por sus siglas en inglés) permite redirigir un dominio o subdominio a un recurso que se encuentra detrás de una puerta de enlace que contiene una dirección IP asignada de forma dinámica, esto significa que es un dominio temporal que puede utilizar el usuario (Google Doma–ns - DNS, 2022). Al crear un servicio Web, este estará ligado a una dirección IP. Al asignarle una DNS dinámica permite que, aunque la dirección IP a la que se encuentra ligado el servidor cambie, el usuario pueda ser redireccionado al dominio del servicio a través de la búsqueda de la base de datos del

servidor DNS. Este lo obtiene del caché y lo almacena para futuras peticiones **(Fig. 34)**. Para asignar este DNS dinámico se hizo uso de un proveedor de DNS gratuito. A APINET se le asignó el dominio de *"apinet.hopto.org"* en el cual se puede hacer uso de la aplicación web. La forma en que esto funciona es que el usuario accede al dominio (1), a continuación, el equipo busca en el servidor la página a que se desea acceder (2). El servidor se comunica con el servidor de DNS (3). Dicho servidor DNS hace búsqueda de la dirección IP asignada al dominio y le retorna esta información al servidor (4), y el servidor le entrega esta información al usuario (5) y de esta manera se puede acceder a la página web deseada (6).

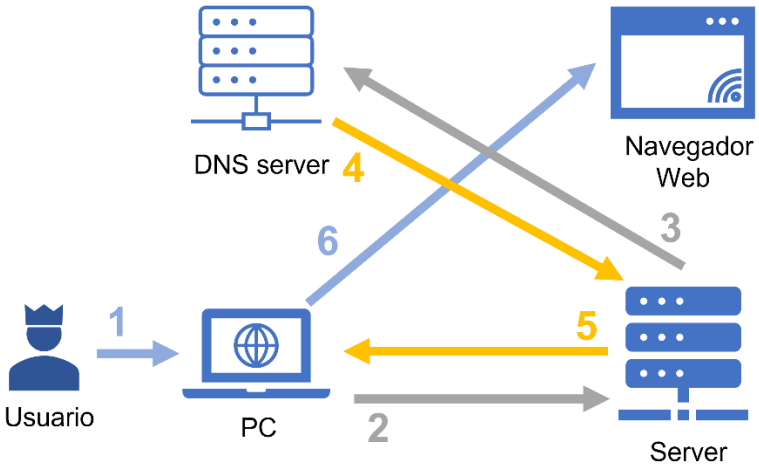

Fig. 34. Funcionamiento de un DNS dinámico

4.6.2.2. Publicación de API como servicio

Para publicar esta API como un servicio, se hizo uso de *Waitress* es un WSGI. Este provee un contenedor que permite comunicar el servidor con la aplicación de Python, y al hacer uso del DNS dinámico previamente asignado, permite desplegar la aplicación web como un servicio público disponible para cualquier persona interesada en usarlo.

Capítulo 5

5. Pruebas y Resultados

En este apartado se proporciona información sobre la implementación de la aplicación y las diversas pruebas a las que este fue expuesto. Se explica a mayor detalle cuales fueron las redes con las que se realizaron las pruebas, que tipos de archivos fueron utilizados y una comparativa de métodos de aplicación de dichos algoritmos de ML y DL. Por último, se muestra una aplicación práctica en un sistema tutor que hace uso de lectura de emociones para adecuar la enseñanza a las emociones del usuario, y de dicha forma mostrar la funcionalidad práctica que tiene la plataforma APINET.

5.1. Redes ML utilizadas

Para probar la plataforma APINET se realizaron pruebas con distintas redes. Dichas redes cumplen la función de obtener predicciones partiendo del procesamiento de archivos. Estas redes de ML requieren distintos tipos de preprocesamiento para utilizar los datos que desean procesarse. Se decidió utilizar estas redes, ya que estas han sido previamente entrenadas, y en trabajos previos se ha probado la veracidad de sus resultados. A continuación, se presentan las redes neuronales utilizadas con algunas de sus características.

- **CNN RestnetLike:** Utilizada principalmente para el procesado de archivos tanto de imagen como de vídeo. Dicha red requiere una entrada. En el caso de las imágenes es sencillo, ya que estas solo deben ser redimensionadas a las dimensiones exactas que requiere esta red para su entrada. En el caso de vídeos es un tanto diferente, ya que se debe hacer un preprocesamiento especial para que estos sean divididos en distintas imágenes y que estas sean preprocesadas de una en una, para posteriormente guardar sus respectivos valores de predicción en un arreglo que servirá para realizar la predicción real del archivo. Esta red cuenta con varias capas convolucionales con una función de activación ReLU (**Fig. 35)**. Esta función es una de las más utilizadas ya que al rectificar los datos de manera lineal no permite que exista una saturación en la red, además de que es más fácil de implementar ya que los cálculos requeridos para

usarla son mínimos en comparación con otras funciones de activación. Para la salida, la red cuenta con una capa con función de activación Sigmoide (**Fig. 35**) ya que esta permite transformar los datos obtenidos a valores de entre 0 y 1, que es la forma en que los valores de personalidad OCEAN son presentados.

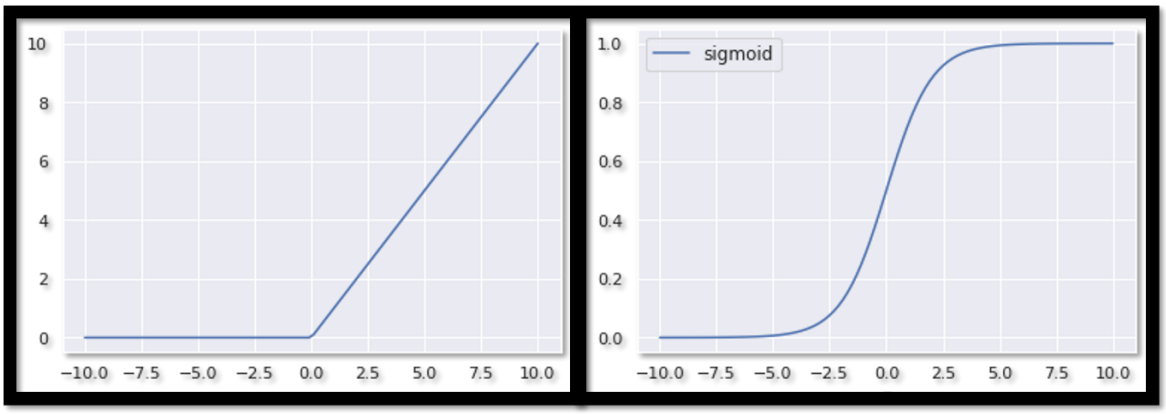

Fig. 35. Función de Activación ReLU (izquierda) y Sigmoide (Derecha)

- **LSTM:** Esta red neuronal es la establecida para procesar datos de audio. Dicha red cuenta con dos entradas en paralelo. La primera entrada es el Coeficiente Ceptral en las Frecuencias de Mel (MFCC por sus siglas en inglés) y la segunda es la inflexión de la MFCC (DMFCC por sus siglas en inglés). Por lo tanto, se necesita utilizar un preprocesamiento especial para su aplicación, en el cual es necesario aplicar ciertos filtros a los archivos de audio para obtener dichas frecuencias. A continuación, a dichos datos se les aplica una función de activación Tanh (**Fig. 36**), que es en esencia muy similar a la función sigmoide. La mayor diferencia es que esta transforma los datos a valores entre -1 y 1. Posteriormente, las siguientes capas utilizan la función de activación SELU (**Fig. 36**), una de las funciones de activación más nuevas que permite normalizar los datos de forma lineal. Esta red neuronal resulta en los valores OCEAN de un archivo de audio. Estos valores indican ciertas características de la personalidad del sujeto que habla en el archivo.

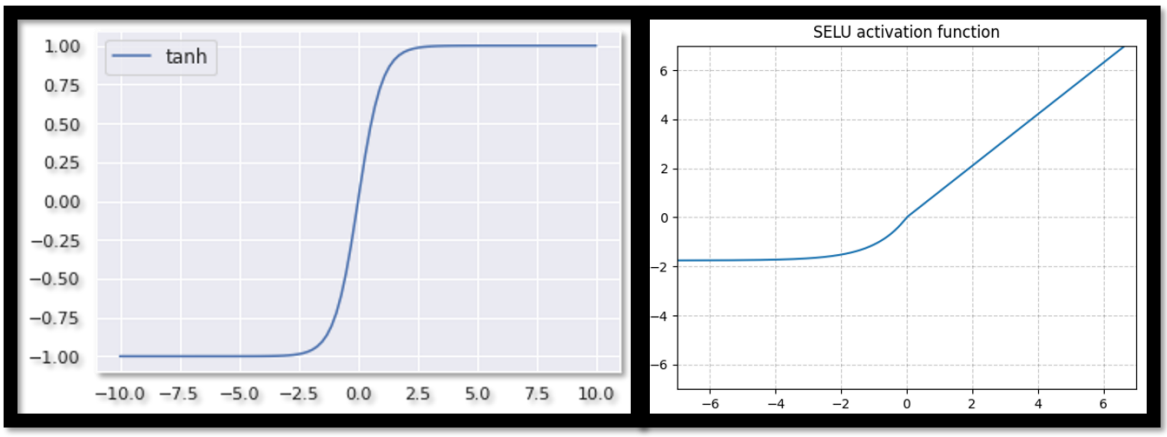

Fig. 36. Función de activación Tanh (izquierda) y SELU (Derecha)

- **CNN Tanh:** Dicha red es muy parecida en estructura a la CNN RestnetLike, con la diferencia que esta cuenta con un mayor número de capas convolucionales y con constantes usos de capas *MaxPooling2d*. Dicha capa es un filtro que reduce las dimensiones del archivo que se esté procesando y, por ende, aligera el tamaño de los datos y reduce el coste computacional. Las funciones de activación que esta red utiliza en sus capas convolucionales es la función SELU, para que, una vez normalizados los datos, aplicar una función Tanh a la salida y así obtener los valores OCEAN de un archivo.

5.2. Archivos utilizados

Los archivos utilizados fueron de tres formatos distintos. El primer formato que se utilizó fueron imágenes, todas con distintas dimensiones que permitieron poner a prueba el preprocesamiento automático. El segundo formato utilizado fueron videos, el cual guarda una relación estrecha con el preprocesamiento de imágenes, ya que, los vídeos son un conjunto de imágenes sucesivas. De igual forma se extrajeron imágenes cuadro a cuadro desde el archivo para poder procesarlos de manera correcta. Por último, se utilizaron archivos de audio, donde estos requieren un procesamiento distinto a aquellos que cuentan con imagen, ya que es necesario obtener distintas frecuencias para poder obtener los valores de predicción.

5.3. Comparativa de implementación “Tradicional vs APINET”

En esta sección se realiza una comparativa de los métodos de aplicación de las distintas redes de ML con distintos archivos. Cabe señalar que las redes utilizadas son redes de ML previamente entrenadas. Esto significa que no continuarán aprendiendo cada vez que se les utilice. Esto permite conocer si el método de aplicación (de manera local y como un servicio) tiene algún tipo de impacto en el rendimiento de las redes y sus resultados.

Para iniciar en la **Tabla 9** se muestran los valores obtenidos de las distintas imágenes en método de aplicación local, o convencional. La red utilizada es la CNN tanh preentrenada. Para realizar esta prueba se tomaron 10 imágenes como prueba, las cuales fueron preprocesadas y procesadas en la red anteriormente mencionada. Como es posible observar en esta tabla comparativa, los valores obtenidos de los distintos métodos de aplicación de redes (local y APINET) no tienen variación visible.

Tabla 9. Pruebas de métodos de aplicación con imágenes

Método de aplicación	Red	Tipo de archivo	No. De prueba	Apertura	Conciencia	Extroversión	Simpatía	Neuroticisimo
Local	CNN	Imagen	Img 1	0.511497	0.515461	0.464874	0.521294	0.411860
			Img 2	0.479524	0.452937	0.550711	0.495981	0.445307
			Img 3	0.462758	0.476843	0.505673	0.432977	0.414076
			Img 4	0.419833	0.453393	0.511498	0.465237	0.462232
			Img 5	0.499225	0.535175	0.577878	0.621478	0.568173
			Img 6	0.490451	0.513168	0.570342	0.535384	0.530633
			Img 7	0.433058	0.521709	0.525312	0.374464	0.440899
			Img 8	0.479524	0.452937	0.550711	0.495981	0.445607
			Img 9	0.511497	0.515461	0.464874	0.521294	0.411861
			Img 10	0.292883	0.248429	0.428367	0.297782	0.312538
APINET			Img 1	0.511497	0.515461	0.464874	0.521294	0.411860
			Img 2	0.479524	0.452937	0.550711	0.495981	0.445307
			Img 3	0.462758	0.476843	0.505673	0.432977	0.414076
			Img 4	0.419833	0.453393	0.511498	0.465237	0.462232
			Img 5	0.499225	0.535175	0.577878	0.621478	0.568173
			Img 6	0.490451	0.513168	0.570342	0.535384	0.530633
			Img 7	0.433058	0.521709	0.525312	0.374464	0.440899
			Img. 8	0.479524	0.452937	0.550711	0.495981	0.445607
			Img 9	0.511497	0.515461	0.464874	0.521294	0.411861
			Img 10	0.292883	0.248429	0.428367	0.297782	0.312538

En la siguiente prueba se tomaron en cuenta cuatro videos distintos. Estos vídeos constan de una persona hablando sobre distintos temas que le gustan, por lo tanto, es un buen punto de referencia para obtener las características de su personalidad. A estos se les aplicó el preprocesamiento y procesado de la CNN RestnetLike. Los valores obtenidos de dichas pruebas se presentan en la **Tabla 10**, En esta se puede observar cómo es que los valores obtenidos de ambos métodos de aplicación y con los mismos archivos se mantienen sin cambio aparente.

Tabla 10. Pruebas de métodos de aplicación con videos

Método de aplicación	Red	Tipo de archivo	No. De prueba	Apertura	Conciencia	Extroversión	Simpatía	Neuroticisimo
Local	CNN	Vídeo	Vid 1	0.358027	0.601372	0.469023	0.520716	0.466949
			Vid 2	0355930	0.592848	0.490137	0.515001	0.473548
			Vid 3	0.366277	0.585334	0.483388	0.507695	0.469852
			Vid 4	0.345484	0.594920	0.445389	0.480734	0.446282
APINET			Vid 1	0.358027	0.601372	0.469023	0.520716	0.466949
			Vid 2	0355930	0.592848	0.490137	0.515001	0.473548
			Vid 3	0.366277	0.585334	0.483388	0.507695	0.469852
			Vid 4	0.345484	0.594920	0.445389	0.480734	0.446282

Para finalizar con las pruebas, se hicieron evaluaciones con distintos audios dónde una persona habla de temas de su interés, permitiendo que sea una buena métrica para obtener los valores OCEAN de dicho individuo. En la **Tabla 11** se muestran los valores obtenidos de distintas pruebas de audio, en dónde también puede observarse que a pesar de que el método de aplicación es distinto, los valores obtenidos siguen sin presentar variaciones.

Tabla 11. Pruebas de métodos de aplicación con audio

Método de aplicación	Red	Tipo de archivo	No. De prueba	Apertura	Conciencia	Extroversión	Simpatía	Neuroticisimo
Local	LSTM	Audio	Aud 1	0.374579	0.457982	0.407694	0.416762	0.367856
			Aud 2	0.275154	0.298773	0.256116	0.463264	0.341505
			Aud 3	0.374426	0.457982	0.407694	0.416762	0.367856
APINET			Aud 1	0.374579	0.457982	0.407694	0.416762	0.367856
			Aud 2	0.275154	0.298773	0.256116	0.463264	0.341505
			Aud 3	0.374426	0.457982	0.407694	0.416762	0.367856

Tal como fue posible apreciar, en todos los casos probados dónde se implementaron redes de diferentes maneras (Local y APINET) es fácilmente observable que los valores se mantienen

sin diferencia alguna entre métodos. Por lo cual se puede concluir que APINET como servicio no tienen un impacto negativo ni repercute negativamente en la obtención de resultados, resultando en un buen nivel de confiabilidad. Ahora es posible usarlo de manera práctica en otros programas o aplicaciones.

5.4. Aplicación práctica en sistema tutor LearnPy

Tomando en cuenta los resultados de confiabilidad de las redes almacenadas en el repositorio, del cual hace uso APINET, se decidió hacer uso de un sistema tutor inteligente para aprender el lenguaje *Python,* llamado "*LearnPy*" (**Fig. 37**), el cual fue desarrollado por un estudiante de posgrado del Instituto Tecnológico de Culiacán. Se decidió utilizar este sistema tutor, debido a que utiliza un modelo de red neuronal de aprendizaje profundo, que reconoce el estado afectivo del estudiante y que es determinante para elegir la complejidad de los ejercicios que el usuario realizará. A continuación, se da una breve explicación de esta aplicación.

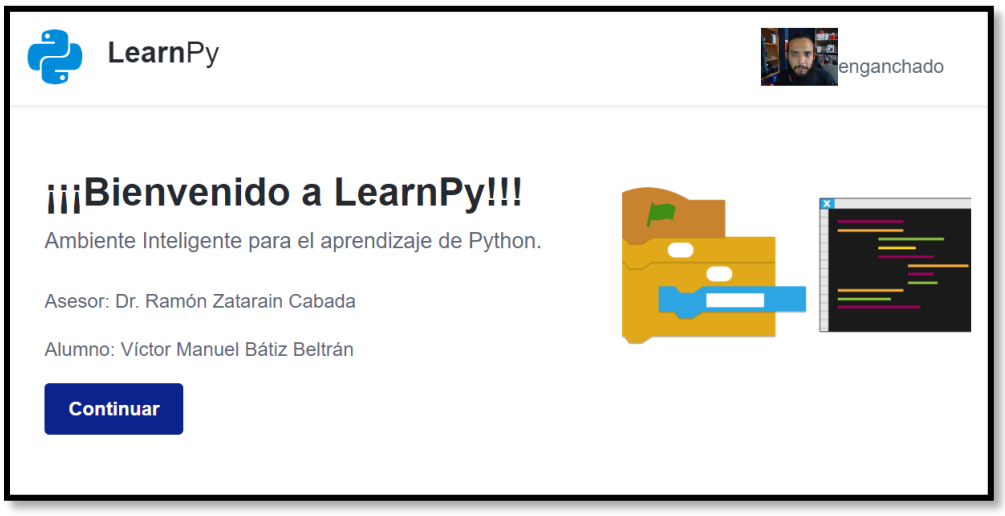

Fig. 37. Interfaz principal de *LearnPy*

5.4.1. ¿Qué es LearnPY?

El sistema *LearnPY*, es un sistema tutor inteligente para aprender el lenguaje de programación *Python* (**Fig. 38**). Este sistema presenta los ejercicios que le provee al usuario

dependiendo del estado afectivo que esté presente. Para esto hace uso de la cámara web, con la cual captura la imagen del usuario en tiempo real. Dicha captura de imagen trabaja en conjunto con una red de ML de reconocimiento de emociones. Esta red reconoce seis emociones distintas. Dichas emociones junto con el tiempo que le toma al usuario realizar un ejercicio, son utilizadas para decidir la complejidad del siguiente ejercicio a presentarle al usuario (fácil, intermedio o difícil). Las emociones se clasifican en dos grupos. Estos son emociones negativas, donde se encuentran las emociones de "aburrido", "relajado" y "neutral, y el otro grupo es el de emociones positivas, en el cual se encuentran "interesado", "enganchado" y "concentrado".

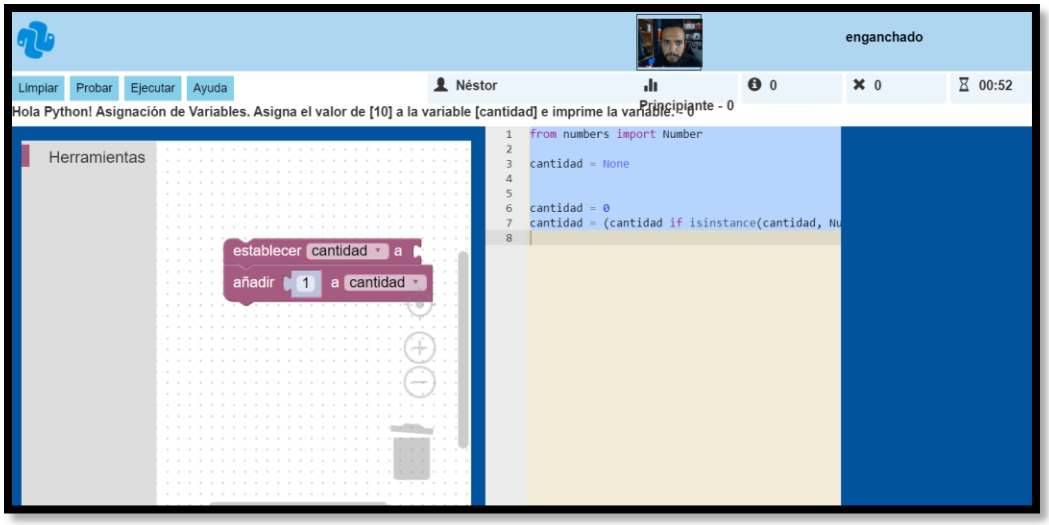

Fig. 38. Interfaz de ejercicios de *LearnPy*

5.4.2. Pruebas de *LearnPy* con APINET

Para probar que APINET puede ofrecer las redes de ML y DL que se encuentran en su repositorio como un servicio consumible, se hizo que *LearnPy* consumiera el modelo de ML de reconocimiento de emociones a través de la interfaz de aplicación que provee APINET. Esto consiste en que el sistema tutor no contenga en sus archivos ningún tipo de red de ML o DL que pueda utilizar y, en cambio hacer un llamado a través de APINET para utilizar el repositorio de modelos de ML y DL. En este caso APINET funciona como el puente entre *LearnPy* y el modelo de reconocimiento de emociones. Para realizar el proceso de comunicación Usuario – *LearnPy* – APINET (**Fig. 39**) se inicia con que el usuario hace una

petición al sistema tutor *LearnPy* (1). Al momento en que el sistema ha sido llamado, este reconoce el uso y realiza una petición a APINET a través de su API para utilizar el modelo de ML para reconocimiento de emociones (2). Después APINET se comunica con su repositorio físico de modelos para responder a la petición de *LearnPy*, en dónde busca el modelo que el sistema tutor requiere (3). El repositorio responde a la petición, y devuelve un modelo de ML o DL para ser consumido (4). A continuación, APINET se comunica con *LearnPy*, y este le proporciona el modelo que se había pedido con anterioridad (5). El siguiente paso consiste en que LearnPy utiliza la cámara web (6) y esta le regresa una imagen en tiempo real del usuario (7). A continuación, el sistema tutor utiliza el modelo de reconocimiento de emociones en conjunto con la imagen del usuario (8), entonces recibe una respuesta del modelo, la cual contiene como resultado una emoción (9). Por último, LearnPy toma en cuenta dicha emoción y el tiempo que le llevó al usuario resolver un ejercicio para proveerle el siguiente ejercicio (10) y que de esta forma tenga una curva de aprendizaje personalizada a su situación emocional.

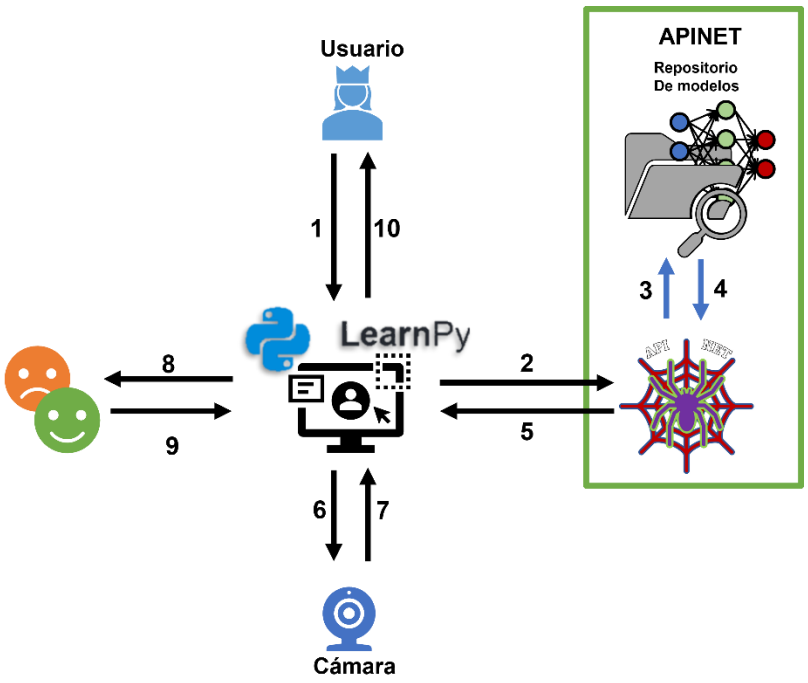

Fig. 39. Proceso de comunicación Usuario - *LearnPy* - APINET

Al utilizar LearnPy en conjunto con APINET, se pudo observar que el sistema tutor funciona sin ningún tipo de dificultad al hacer uso de la API para consumir un modelo de ML, con un desempeño óptimo. El proceso de avance del usuario con respecto a los ejercicios que le provee LearnPy, tampoco se vio comprometido, haciendo de APINET una opción más que viable para consumir modelos de ML y DL de una manera segura, fácil y eficaz, con una interfaz de aplicación de uso sencillo. Lo anterior significa, en términos de desarrollo, que adaptar una aplicación y ofrecerla como un servicio en APINET es conveniente y simple.

Capítulo 6

6. Conclusiones y trabajo futuro

En esta sección son presentadas las conclusiones obtenidas a partir del desarrollo del proyecto APINET. Son mencionadas las aportaciones y limitaciones, tanto de desarrollo como de la investigación realizada. Del mismo modo, se definen las actividades de implementación y desarrollo a futuro para este trabajo.

6.1. Conclusiones del proyecto APINET

Los resultados obtenidos permiten confirmar la verdad en la hipótesis planteada en un inicio, la cual establecía que era posible crear una metodología que implemente una variedad de modelos de ML y DL como una aplicación y, a su vez, esta sea presentada como un servicio disponible para cualquier usuario. Es fácil observar que se encontró una manera eficaz de exponer los modelos de ML y DL en una API al servicio del consumidor. Esto hace que este proyecto tenga alta aplicabilidad, debido a que, el contar con una plataforma como la desarrollada en este trabajo permite que un usuario de internet tenga al alcance de sus manos el uso de distintas redes de aprendizaje máquina y profundo, lo que conlleva a que un mayor número de investigaciones al respecto sean realizadas.

Los resultados obtenidos a partir de la aplicación de distintas redes en distintos tipos de datos proveen la información necesaria para discernir que la red muestra una alta confiabilidad con la implementación local de las redes, o bien otro tipo de uso más convencional, dígase uso de modelos almacenados en la nube. La gran ventaja y diferencia que se encuentra con las aplicaciones acostumbradas es que, al tener un preprocesamiento completamente automatizado, elimina una de las más grandes barreras al momento de usar modelos de ML y DL, volviendo más fácil que una persona sin alguna experiencia previa, o sin conocimiento exacto de los detalles de implementación de uno de estos modelos sea capaz de utilizarlos sin ningún contratiempo o inconveniente.

6.2. Aportaciones y limitaciones

La principal contribución realizada por el desarrollo de este proyecto es el de ofrecer una opción para diseñar, desarrollar e implementar una plataforma que se encuentre siempre disponible para cualquier usuario que desee utilizarla. Esta pone a disposición diversos desarrollos de modelos de aprendizaje máquina y profundo y, en conjunto con los distintos algoritmos de selección y preprocesamiento, permite que los usuarios utilicen la plataforma sin problemas y de una forma muy sencilla. Lo anterior es algo muy valioso hablando a nivel de aplicación, ya que esto significa que se tiene la funcionalidad básica cubierta y es posible agregar futuras implementaciones en caso de ser necesario.

Otra de las aportaciones que ofrece el desarrollo de APINET es el de tener centralizado un repositorio de desarrollos de modelos de ML y DL. Además, estos cuentan con un programa que permite automatizar el preprocesamiento que los archivos necesitan para poder ser utilizados con dichos modelos. Esto significa un avance significativo debido a que es aplicable en cualquier tipo de implementación que utilice modelos de ML y DL, ahorrando tiempo y trabajo en futuras investigaciones.

La mayor limitación con que se contó al momento de crear este proyecto es el tiempo disponible para la investigación, desarrollo e implementación, ya que al ser limitado no permitió producir un mayor número de pruebas e implementación de otras funciones a la plataforma. Se considera que al tener un repositorio de modelos de ML y DL más grande permitiría generar más pruebas, y por ende tener resultados más confiables.

6.3. Trabajo futuro

Es propuesto, a modo de trabajo futuro, que se incremente el número de modelos de ML y DL disponibles en la API, haciendo crecer tanto el repositorio de estos algoritmos como el de preprocesamiento. De esta manera se espera explorar una mayor variedad de aplicaciones y permitir que la plataforma sea implementada como un módulo a otros programas. Esto permitirá que estas implementaciones contengan una manera de procesar distintos tipos de archivos y hacer uso de ML y DL para obtener predicciones más precisas. Todo esto puede ser logrado al tener una comunicación más directa con los usuarios, así que un espacio dónde

puedan escribir sugerencias de tipos de redes o archivos soportados por la plataforma también sería implementado.

Otro trabajo posterior considerado es el de agregar al servicio una opción en la que el usuario pueda elegir cual red de las sugeridas usar, para así poder generar varias predicciones con un mismo dato. Al tener la oportunidad de hacer dicha elección, el preprocesamiento al que se someterán los datos también es distinto, pudiendo generar un mayor número de datos para comparar.

Referencias

A Simple Guide to Data Preprocessing in Machine Learning, Recuperado el 15 de junio de 2022, de https://www.v7labs.com/blog/data-preprocessing-guide#data-preprocessing-steps

Alam, S., Yao, N. (2019). *The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis*. *Computational and Mathematical Organization Theory*. DOI: 10.1007/s10588-018-9266-8

Apt, K. R. (2003). *Principles of constraint programming*. *Cambridge University Press.*

Bátiz Beltrán, V. M.. (2021). Reconocimiento Automático de Personalidad Aparente contra Prueba EstandarizadaSistema de evaluación de la personalidad y de las emociones en el proceso cognitivo. COMIA 2021.

Choi K. (2018). *A Comparison of Audio Signal Preprocessing Methods for Deep Neural Networks on Music Tagging. EUSIPCO 2018: 26th European Signal Processing Conference.* DOI: 10.48550/arXiv.1709.01922

Data Preprocessing in Machine Learning: 7 Easy Steps To Follow. Recuperado el 15 de junio de 2022, de https://www.upgrad.com/blog/data-preprocessing-in-machine-learning/#Why_Data_Preprocessing_in_Machine_Learning

Dato en informática. Recuperado el 15 de junio de 2022, de https://concepto.de/dato-en-informatica/

Deploy to Production. Recuperado el 15 de junio de 2022, de https://flask.palletsprojects.com/en/2.1.x/tutorial/deploy/?highlight=waitress

Dhanalakshmi, R., Chellappan, C. (2009). *File Format Identification and Information Extraction. 2009 World Congress on Nature & Biologically Inspired Computing 9.* DOI: 10.1109/NABIC.2009.5393688

Fa, C., Chen, M., Wang, X., Wang, J., Huang, B. (2021). *A Review on Data Preprocessing Techinques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data*. *Frontiers in Enery Research, 9*. DOI: 10.3389/fenrg.2021.652801

Face Recognition. Recuperado el 15 de junio de 2022, de https://github.com/ageitgey/face_recognition

Fetherston, A., Gollins, T. (2012). *Towards the Development of a Test Corpus of Digital Objects for the Evaluation of File Format Identification Tools and Signatures*. *International Journal of Digital Curation, 7*. DOI: 10.2218/ijdc.v7il.211

Flask Web development, one drop at a time. Recuperado el 15 de junio de 2022 de https://flask.palletsprojects.com/en/2.1.x/

Garcia A. L., Tran, V., Alic, A. S., Caballer, M., Plasencia, I. C., Costantini, A., Plociennik, M. (2020). *A cloud-based framework for machine learning workloads and applications. IEEE Access.* DOI:10.1109/ACCESS.2020.2964386

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., Herrera, F. (2016). *Big data preprocessing: methods and prospects. Big Data Analytics, 1*. DOI: 10.1186/s41044-016-0014-0

Haselbock, S., Weinreich, R., Buchgeher, G., Kriechbaum, T. (2019). *Microservice Design Space Analysis and DecisionDocumentation: A Case Study on API Management*. 1–8. DOI:10.1109/SOCA.2018.00008

Lebre, A., Legrand, A., Suter, F., Veyre, P. (2015). *Adding storage simulation capacities to the SimGrid toolkit: Concepts, models, and API*. *Proceedings – 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*. DOI: 10.1109/CCGrid.2015.134

Li, L.E., Chen, E., Hermann, J., Zhang, P. & Wang, L. (2017). *Scaling Machine Learning as a Service*. *Proceedings of The 3rd International Conference on Predictive Applications and APIs, in Proceedings of Machine Learning Research 67:14-29.*

Librosa. Recuperado el 15 de junio de 2022, de https://librosa.org/doc/latest/index.html

Madrid, H. P., Totterdell, P. Niven, K. (2016). *Does leader affective presence influence communication of creative ideas within work teams?*. DOI: 10.1037/emo0000183

Más información sobre el DNS dinámico. Recuperado el 15 de junio de 2022, de https://support.google.com/

Minsky, M. (1985). *The Society of Mind*. *Touchstone.*

Miura, T. (2017) *Novel Infrastructure with Common API using Docker for Scaling the Degree of Platforms for Smart Community Services. 2017 IEEE 15th International Conference on Industrial Informatics (INDIN).* DOI: 10.11097INDIN.2017.8104818

Netron app. Recuperado el 15 de junio de 2022, de https://github.com/lutzroeder/netron

Nilsson, N. J. (1998). *Introduction to Machine Learning an Early Draft of a Proposed Textbook*.

Numpy. Recuperado el 15 de junio de 2022, de https://numpy.org/doc/stable/

Oladipupo Ayodele, T. (2010). *Types of Machine Learning Algorithms*. *University of Portsmouth.* DOI: 10.5772/9385.

Oldham, S., Arnatkevičiūtė, A., Smith, R. E., Tiego, J., Bellgrove, M. A., Fornito, A. (2020). *The efficacy of different preprocessing steps in reducing motion-related confounds in diffusion MRI connectomics.* DOI: 10.1016/j.neuroimage.2020.117252

Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Slyke, J. (2017). *TensorFlow-Serving: Flexible, High-Performance ML Serving*. *Cornell University.* DOI: 10.48550/arXiv.1712.06139

OpenCV. Recuperado el 15 de junio de 2022, de https://opencv.org/about/

Patterson, J., Gibson, A. (2017). *Deep Learning: A practitioner's approach. O'Reilly Media, Inc.*

Picard, R. W. (2000). *Affective computing*. *The MIT Press*.

Program Library HOWTO. Recuperado el 15 de junio de 2022, de https://tldp.org/HOWTO/Program-Library-HOWTO

Ribeiro, M., Grolinger, K., Capretz, M. A., Capretz, M. A. M., Ribeiro, M., Grolinger, K., Capretz, M. A. M. (2015). *MLaaS: Machine Learning as a Service*. *2015 IEEE 14th*

International Conference on Machine Learning and Applications (ICMLA). DOI: 10.1109/ICMLA.2015.152

Smola, A. (2018). *An Introduction to Machine Learning*. *Cambridge University Press.*

speaker-verificatoin-toolkit. Recuperado el 15 de junio de 2022, de https://pypi.org/project/speaker-verification-toolkit/

TensorFlow. Recuperado el 15 de junio de 2022, de https://www.tensorflow.org/

Venkata, S. K., Young, P., Green, A. (2020). *EasyChair Preprint Using Machine Learning for Text File Format Identification*. *EasyChair Preprint no. 4698.*

Wang, W., Wang, S., Gao, J., Zhang, M., Chen, G., Ng, T. K., & Ooi, B. C. (2018). *Rafiki: Machine Learning as an Analytics Service System*. *Proceedings of the VLDB Endowment, Volume* 12. DOI: 10.48550/arXiv.1804.06087

wave – Read and write WAV files. Recuperado el 15 de junio de 2022, de https://docs.python.org/3/library/wave.html

What are Data Types and Why Are They Important. Recuperado el 15 de junio de 2022, de https://amplitude.com/blog/data-types

Zhu, Z. (2017). *Change detection using landsat time series: A review of frequencies, preprocessing, algorithms, and applications*. *ISPRS Journal of Photogrammetry and Remote Sensing*. DOI: 10.1016/j.isprsjprs.2017.06.013