Instituto Tecnológico de León

División de Estudios de Posgrado e Investigación

“Identificación de tipografías en ambientes no estructurados mediante redes profundas de aprendizaje”

Tesis

Que presenta:

Ing. Oscar Kevin Pérez Hernández

Para obtener el grado de:

Maestro en Ciencias de la Computación

Con la dirección de:

Dr. Raúl Santiago Montero

Con la co-dirección de:

Dr. María del Rosario Baltazar Flores

León, Guanajuato. Junio 2024

Dedicatoria:

A mi familia, mi hermano mayor quien siempre ha sido un ejemplo para seguir y que siempre cuento con su apoyo cunado tengo problemas con cualquier tema. A mi hermano menor que siempre está ahí para hablar y ayudarme a distraerme de las labores tan estresantes que demandan los estudios, y por siempre estar al pendiente de mi estado de ánimo. A mi papa que diariamente pone el ejemplo de lo que es ser dedicado y responsable, no solo con la vida academia y laboral si no también con la familia. A mi mama que se lleva la gran labor de hacer de una casa un hogar donde todos los que vivimos ahí disfrutamos de estar, por escucharme cada noche y preguntarme acerca de mis avances y metas. A mis amigos tanto a los de la licenciatura como los nuevos amigos obtenidos durante esta maestría por su infinito apoyo, paciencia y fe en mí. Esta tesis es por y para ustedes porque gracias a todos ustedes he llegado tan lejos y hay un poco de todo ustedes en este trabajo.

Agradecimientos:

Al Tecnológico Nacional de México/Instituto Tecnológico de León que me permitió conocer a mi asesor, el Dr Raúl Santiago Montero a quien le agradezco infinitamente por el gran soporte a lo largo de mi preparación y por guiarme durante todo este proceso para lograr tan anhelado y complicado objetivo. De igual manera, agradezco a mi co-asesora la Dra. María del Rosario Baltazar Flores quien siempre estuvo atenta para apoyarme, guiarme y brindarme las mejores opciones para culminas mis estudios. Finalmente, un agradecimiento especial al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) por el apoyo económico a través de su programa de becas, facilitando así mi formación y desarrollo.

Contenido

Introducción:

En la presente tesis se ha investigado el desempeño del modelo de redes neuronales convolucionales ResNet50 en la clasificación de imágenes de botellas de vino. El objetivo principal de esta investigación ha sido evaluar la capacidad del modelo para distinguir entre diferentes clases de vinos, bajo diversas condiciones experimentales. Se ha investigado con el fin de poner a prueba una red pre-entrenada y ver que tan factible es utilizarla en sistemas de asistencia visual.

La motivación para llevar a cabo esta investigación radica en la necesidad de indagar en los procesos de clasificación de las redes neuronales convolucionales (CNN, por sus siglas en inglés), las cuales son críticas en la actualidad para sistemas de asistencia diversos. La precisión y velocidad en la identificación de productos pueden tener un impacto significativo en la gestión de inventarios, así como la satisfacción del cliente. Además, comprender las limitaciones y fortalezas de los modelos de clasificación de imágenes en diferentes escenarios puede guiar futuras implementaciones y mejoras tecnológicas.

Para abordar estos objetivos, se diseñaron y ejecutaron seis experimentos utilizando el modelo ResNet50, evaluando su desempeño en diferentes contextos:

- Clases muy distintas entre sí
- Imágenes capturadas en tiempo real vs imágenes bajadas de internet
- Productos similares
- Variación en el número de vinos por clase

La metodología empleada en esta investigación incluye la captura y preprocesamiento de imágenes, la configuración y entrenamiento del modelo ResNet50, y la evaluación de su desempeño en términos de precisión, exactitud y tiempo de procesamiento. Los resultados de estos experimentos han proporcionado una comprensión detallada de cómo el modelo responde a diferentes desafíos en la clasificación de imágenes de vinos.

Esta tesis contiene una descripción detallada de los experimentos realizados, el análisis de los resultados obtenidos, y las conclusiones derivadas de estos análisis. Además, se discuten las implicaciones de estos hallazgos para el desarrollo de sistemas de clasificación de productos en entornos reales y se proponen futuras direcciones de investigación para abordar las limitaciones identificadas.

1 Planteamiento del problema

La discapacidad visual es una condición que afecta a millones de personas en todo el mundo, limitando su capacidad para acceder de manera independiente a una variedad de entornos y servicios, incluyendo la experiencia de compra en centros comerciales. La falta de información visual dificulta la identificación y selección de productos, lo que conlleva a un desafío significativo para las personas con discapacidad visual en su vida diaria.

La tecnología de visión por computadora ha avanzado considerablemente en los últimos años, y las CNN han demostrado un alto potencial en la clasificación de objetos y la detección de patrones visuales. Las CNN se han aplicado con éxito en diversas áreas, como el reconocimiento de imágenes, la detección de objetos y la visión por computadora en general. Sin embargo, aún existen desafíos significativos en la aplicación de estas tecnologías para mejorar la accesibilidad. La autonomía de las personas con discapacidad visual en entornos de compra, como los centros comerciales es uno de estos desafíos.

Por consiguiente, el presente estudio se enfoca en poner a prueba una CNN en su tarea de clasificación de productos en estantes de centros comerciales, poniendo así a prueba la capacidad de esta para tareas de asistencia visual. Esto nos será de ayuda para saber que tan factible es la implementación de este tipo de modelos de aprendizaje profundo en temas de asistencia visual en centros comerciales.

1.1 Hipótesis

Si una CNN puede reconocer y clasificar entre productos diferentes en un estante de centros comerciales, como los vinos en un porcentaje por arriba del 90 %; entonces servirá para instrumentar un sistema computacional de reconocimiento efectivo del producto.

1.2 Justificación

Es bien sabido que la discapacidad visual y la ceguera ocupan en primer y segundo tipo de discapacidad humana con mayor prevalencia, donde las patologías asociadas con el estilo de vida nutricional, actividad física, envejecimiento, actividad laboral y violencia/traumatismos, juegan un rol significativo en el deterioro visual adquirido del adulto (Escudero, 2011).

La organización mundial de la salud (OMS) en su página oficial en octubre del 2022, da a conocer que se ha alcanzado cifras de hasta 2200 millones de personas a nivel mundial con discapacidad visual cercana o lejana. Además, esta discapacidad es padecida en su mayoría por personas que superan los 50 años y que al menos la mitad tiene cura (ONU, 2023). Lo que con el tiempo ha llamado la atención de la investigación en desarrollar herramientas para el apoyo de las personas con discapacidades visuales, desarrollando sistemas electrónicos de ayuda a la visión (EVES) por sus siglas en ingles (Wolffsohn, 2003).

Buscando lograr que las personas con ceguera y con discapacidad visual puedan detectar, visualizar, navegar y reconocer obstáculos con mayor facilidad para poderse desenvolver de manera normal por los diferentes entornos que nos rodean, usando dispositivos de captura los cuales son principalmente las cámaras de video y sensores ultrasónicos (Mashiata, 2022).

Por lo tanto, la contribución principal de este trabajo de investigación es el usar técnicas de aprendizaje profundo, con el fin poner a prueba la capacidad de las CNN en el reconocimiento de productos en estantes de centros comerciales, que pudiera en un futuro ser implementado para la asistencia de personas con discapacidad visual.

1.3 Objetivo general

Entrenar una CNN que aprenda a discernir de un estante en un centro comercial una clase específica de productos, específicamente vinos tintos, mediante imágenes digitales en RGB.

1.3.1 Objetivos específicos

- Plantear el escenario o conjunto de escenarios de donde se tomarán las imágenes para la base de datos.

- Crear la base de datos de imágenes digitales para estantes de ventas de vinos en diversos contextos, tales como centros comerciales.

- Entrenar una red neuronal profunda para clasificar e identificar una clase de vino especifico, dentro de un conjunto diverso de vinos.

- Generar y diseñar las pruebas de validación estadística que muestren el alcance y eficiencia de las redes neuronal en esta tarea, y compilar los resultados para ser expuestos en un reporte científico con revisión estricta.

- Reportar los resultados en un reporte técnico.

1.4 Metodología

La metodología establecida para el presente proyecto de investigación está basada en el diseño del código y base de datos, necesarios para poner a prueba

una red neuronal convolucional en su tarea de clasificación de productos en centros comerciales.

1. Búsqueda de información: se realizó la búsqueda del estado del arte de proyectos existentes respecto a los temas de CNN aplicados en la detección y clasificación de objetos.

2. Lenguaje seleccionado: se buscó el lenguaje que prestara más y mejores herramientas en la actualidad para el desarrollo e implementación de redes neuronales en temas de detección y clasificación.

3. Diseño de la base de datos: una vez seleccionado el lenguaje con el que se trabajó, se buscó crear una variedad de base de datos. Bases de datos que fueron creadas de 2 maneras.

    - Tomando fotografías de vinos en estanterías de centros comerciales al igual que tomando fotos de objetos aleatorios en cualquier escenario, ya sean centros comerciales o en cualquier otro lugar.

    - Descargando imágenes de internet que incluyen clases diferentes de vinos y objetos aleatorios como tenedores o cucharas.

4. Diseño del Código: una vez teniendo las imágenes con las que se deseó trabajar, se buscó información acerca de las redes neuronales convolucionales, sus modelos pre-entrenadnos, así también el cómo tomar estos modelos para que clasifiquen nuevas clases de imágenes. Evitando de esta manera el entrenar una CNN completamente desde cero.

5. Hacer pruebas el modelo de CNN creado, usando la base de datos de imagen creada por nosotros. Para este propósito las imágenes se usaron en 6 experimentos diferentes.

Primer experimento: Se utilizaron las imágenes bajadas de internet de vinos y de objetos aleatorios, reescalando su resolución a una resolución por defecto necesaria para que trabaje con el modelo seleccionado, en este caso fue de 224 x 224 pixeles.

Segundo experimento: Se utilizaron las imágenes de vinos claros contra vinos obscuros, siendo estos de la misma manera, imágenes bajadas de internet reescaladas a 224 x 224 pixeles, con el propósito de poner a prueba la CNN en su clasificación entre 2 tipos de vinos.

Tercer experimento: Se repitió el experimento 1, pero ahora con imágenes tomadas directamente de estantes en centros comerciales, de vinos y objetos aleatorios. Igualmente, reescaladas a de 224 x 224, tomamos las fotografías en cualquier escenario para hacer la prueba de vinos contra otros objetos.

Cuarto experimento: Se repitió el experimento número 2, pero ahora con imágenes tomadas directamente de estantes en centros comerciales, de vinos claros contra vinos obscuros, reescaladas a 224 x 224.

Quinto experimento: Para el quinto experimento se tomaron imágenes directamente del estante de centros comerciales para dividirlas en 2 clases. La primera clase tiene imágenes de vinos que no incluyen un vino en específico en ninguna de las tomas. La segunda clase está conformada por imágenes tomas de

estantes de vinos donde si se encuentra un vino específico en todas las fotos. Al igual que los experimentos pasados se reescalaron las imágenes a de 224 x 224.

Sexto experimento: Para el sexto y último experimento, tomamos las imágenes del quinto experimento, pero a la clase 2 le agregamos otro vino especifico que no se repite en la clase uno. De la misma manera el reescalado fue de 224 x 224.

6. Lectura he interpretación de los resultados obtenidos de cada experimento realizado con el modelo CNN seleccionado.

1.5 Alcances y limitaciones

Alcance:

Se pretende poner a prueba un modelo de red neuronal convolucional “ResNet50”, en la tarea de reconocer entre productos de una solo familia (en este caso vinos), por medio de métodos de visión por computadora tomando imágenes de estantes de centros comerciales.

Limites:

No pretendemos reconocer una enorme variedad de productos sino detectar un único producto de una familia (en este caso vinos), establecidos en la base de datos utilizada para los experimentos, en conjunto con un modelo preentrenado de CNN. Además, no se pretende realizar ningún prototipo que realice estas tareas en tiempo real en un centro comercial.

2. Marco Teórico

En este capítulo se abordan los conceptos necesarios implementados en el desarrollo de nuestro modelo de red neuronal convolucional. Comenzando por los conceptos más sencillo que la conforman hasta llegar a los conceptos de redes neuronales recurrentes.

2.2 CNN:

2.1.1 ¿Qué es una red neuronal convolucional?

Para explicar lo que es una red neuronal convolucional o CNN utilizaremos el ejemplo usado por (Srinivas, 2016), el cual nos habla de la red AlexNet la cual cuenta con capas de convolución, seguido de capas de polling y termina con capas completamente conectadas como lo podemos ver en la siguiente ilustración 1.

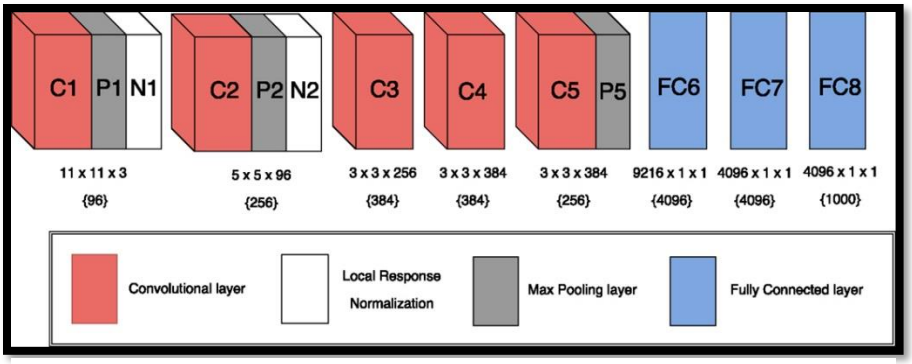

Ilustración 1. Configuración de la Red AlexNet por (Srinivas, 2016).

La red mostrar anteriormente nos es de mucha utilidad para explicar las características que componen una rede neuronal convolucional partiendo de lo que nos menciona (Goodfellow, 2016), quien nos dice que las CNN son especializadas en tomar datos en arreglos de una dimensión o arreglos en dos dimensiones como lo son las imágenes, mientras que la palabra “convolucional” indica que se utiliza una operación matemática llamada “convolución”.

2.1.2 ¿Qué es una convolución?

Partiendo por explicar lo que es un Kernel, el cual es una matriz pequeña que se utiliza para realizar operaciones de convolución, los Kernel son valores numéricos que actúan como un filtro sobre una imagen de entrada. Estos valores son los pesos que se van ajustado durante el entrenamiento de una red neuronal.

Al desplazamiento del kernel recorriendo la imagen se le conoce como convolución, el cual durante este proceso el kernel se coloca en diferentes posiciones de la imagen, y multiplica sus valores por los valores correspondientes de los píxeles de la imagen en esa ubicación como podemos verlo en la ilustración siguiente.

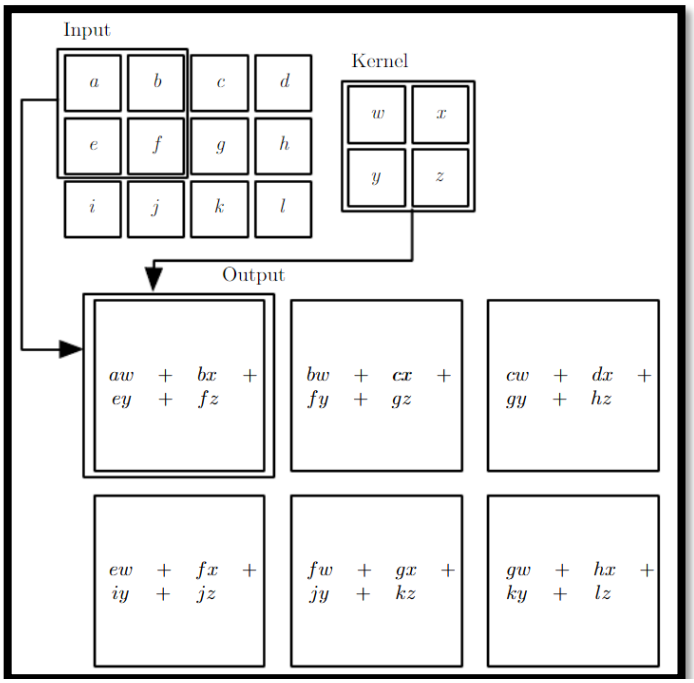

Ilustración 2. Ejemplo de cómo ocurre una convolución por el autor (Goodfellow, 2016)

En la ilustración 2 podemos ver como un kernel, recorre una matriz mucho más grande realizando multiplicaciones. En el caso de las imágenes, cada cuadrado que compone la matriz grande es multiplicado por cada cuadrado que compone la matriz pequeña (kernel). Este es un ejemplo de lo que (Goodfellow, 2016) , quien

nos dice que es una convolución en 2-D sin inversión de núcleo, la cual es la más utilizada cuando se trata de CNN.

2.1.3 ¿Por qué es importante hacer convoluciones?

Según lo que nos mencionas (Srinivas, 2016), las convoluciones nos ayudan a trabajar de mejor manera con imágenes ya que el uso de redes neuronales tradicionales para la clasificación de imágenes no es práctico por la siguiente razón: considere una imagen 2D de tamaño 200 × 200 para la que tendríamos 40.000 neuronas solo en la capad de entrada. Si la capa oculta tiene 20.000 neuronas, el tamaño de la matriz de pesos de entrada sería 40.000 × 20.000 = 800 millones. Esto es solo para la primera capa, además de que el sector de entrada ignora por completo la estructura espacial que nos proporciona una imagen en 2D. Por ello es por lo que se utilizan convoluciones, ya que al realizar operaciones con filtros de por ejemplo 5 x 5, es más manejable que realizar multiplicaciones de matrices gigantes de 500 x 500. Además de que la convolución si toma en cuenta la estructura natural de una imagen en 2D.

Gracias a (Wang, 2020) podemos ver como ocurre esta convolución en una CNN al colocar una imagen en sus 3 canales de color (RGB) y como los pesos diferentes de cada kernel afectan a la imagen dándonos así resultados muy diferentes por cada kernel, el cual es un hiper parámetro que regularmente se puede ajustar para adaptarse mejor al conjunto de datos con el que se está trabajando.

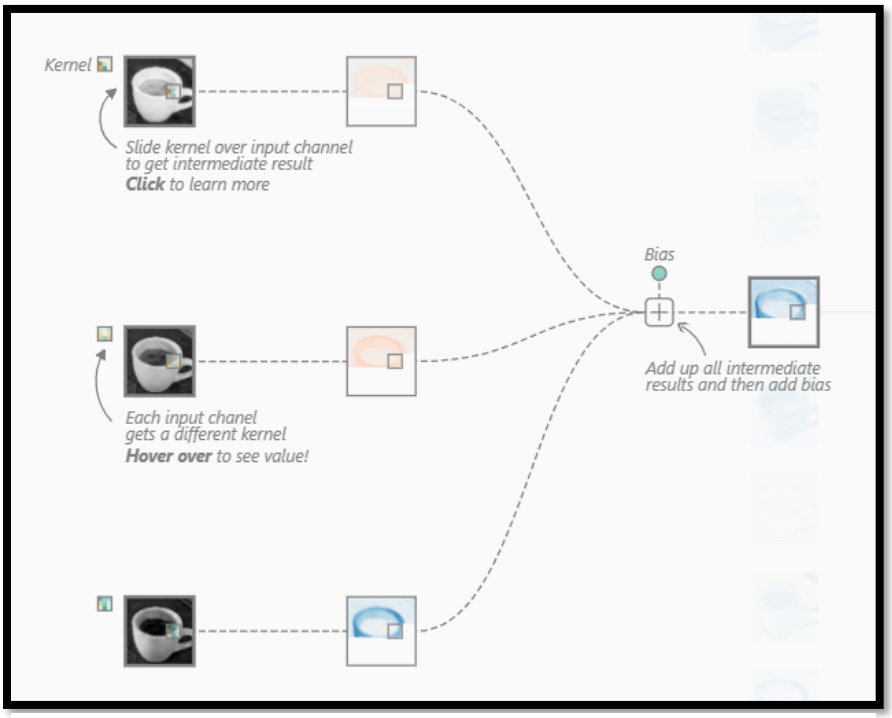

Ilustración 3. Ejemplo ilustrativo de una convolución en los 3 canales de color por el autor (Wang, 2020).

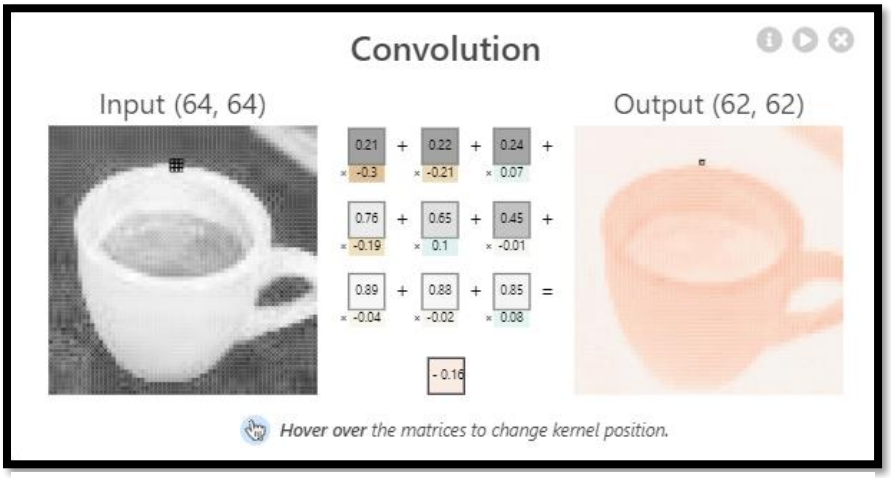

Ilustración 4. Ejemplo ilustrativo del desplazamiento del kernel sobre una imagen por el autor (Wang, 2020).

(Wang, 2020), también menciona que cuando trabajamos con kernels pequeños extraemos más información de la imagen, mientras que un kernel grande extrae menos información. Al final todo dependerá de ajustar el kernel de manera adecuada para el conjunto de datos con el que se desea trabajar asiendo ajustes de manera empírica.

De esta manera al finalizar con la aplicación de los kernel en los canales de color de la imagen, que en este caso son imágenes RGB, se obtiene un mapa de características de los 3 canales de color. Un mapa de características es una imagen bidimensional que destaca las características específicas que el kernel ha detectado en el canal de color de entrada. Cada mapa de características puede representar diferentes tipos de características, como bordes, texturas o patrones complejos, dependiendo de la naturaleza del kernel. Al final estos se suman para obtener el mapa de características final sumándole un sesgo.

Este proceso se repite según la cantidad de neuronas que se deseen utilizar en la capa convolucional, es decir si tenemos 10 neuronas en la capa de entrada lograremos que halla 10 mapas de características de una imagen. Si la imagen es RGB este mapa de características final obtenido por cada neurona, estará compuesto de 3 mapas de características diferentes según el kernel con el que se hallas realizado la convolución de ese mapa de características.

La convolución es una tarea muy importante pues según lo que nos dice (Srinivas, 2016), nos ayuda a tratar con imágenes de manera más efectiva puesto que si deseamos trabajar con redes neuronales completamente conectadas, tendríamos el problema de que se trabajaría con muchísimos más parámetros que con un red convolucional además de que estos vectoriza la imagen ignorando así las características de las imágenes, mientras que las convoluciones 2D ayudan a reducir la cantidad de parámetros a aprender y consideran la estructura espacial de las imágenes de manera más efectiva.

2.2 Funciones de activación

Partiendo de que las redes profundas suelen consistir en convoluciones seguidas de una operación no lineal después de cada capa (Srinivas, 2016). Es recomendable usar operaciones no lineales (funciones de activación), seguidas de las operaciones lineales como lo son las convoluciones para capturar y modelar relaciones no lineales en los datos, lo que las hace muy efectivas en tareas de procesamiento de imágenes y reconocimiento de patrones en general. A

continuación, hablaremos de las funciones de activación más comúnmente utilizadas en la actualidad en redes neuronales convolucionales:

2.2.1 ReLU

La función de activación ReLU (Unidad Lineal Rectificada) según (Wang, 2020), (Srinivas, 2016). Se usa específicamente como una función de activación no lineal, que a diferencia de otras funciones se ha observado empíricamente que las CNN que usan ReLU son más rápidas de entrenar que otras. A continuación, en la ilustración 5, podemos ver la función de activación ReLU.

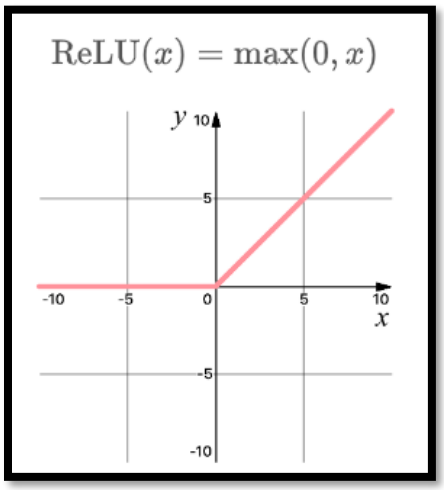

Ilustración 5. Grafica de la función ReLu

Gracias a lo que vemos en el artículo de (Daubechies, 2022), podemos decir explicar la función ReLu, partiendo de la ecuación 1.

Ecuación 1 del autor (Daubechies, 2022).

$$x = (x_1, \ldots, x_d) \in \mathbb{R}^d:$$

Esta nomenclatura usada en la ecuación 1 nos dice que:

- ***X*** es el nombre del vector.

- ***(X1,…,Xd)*** es una secuencia de ***d*** elementos que componen el vector
- ***Xd*** representa la coordenada en la ***d***-ésima dimensión del espacio vectorial
- $\mathbb{R}^d$ indica que el vector ***X*** pertenece al espacio de los números reales de ***d***-dimensiones.

Entonces, por ejemplo, si ***d***= 3, el vector ***X*** se vería así:

Ecuación 2. Del autor (Daubechies, 2022)

$$X = (X_1, X_2, X_3) \in \mathbb{R}^3$$

Esto representa un punto en un espacio tridimensional, donde X_1, X_2, X_3 Son las coordenadas de las direcciones X,Y,Z respectivamente .

Esto nos ayuda a entender la ecuación 3 que podemos ver a continuación.

Ecuación 3. Del autor (Daubechies, 2022)

$$\mathrm{ReLU}(x_1, \ldots, x_d) = (\mathrm{ReLU}(x_1), \ldots, \mathrm{ReLU}(x_d)) = (\max\{0, x_1\}, \ldots, \max\{0, x_d\}).$$

Donde podemos ver que la función ReLu se aplica a elemento por elemento a un vector **X =*(X1,…,Xd) donde:***

- ***ReLU(x1, ..., xd***): Representa la aplicación de la función ReLU al vector ***X***

- ***(ReLU(x1), ..., ReLU(xd***)): Representa el vector resultante después de aplicar ReLU a cada componente por separado. Es decir, cada elemento del nuevo vector es el resultado de aplicar ReLU al correspondiente elemento en el vector original ***X.***

- ***max{0, x1}, ..., max{0, xd}:*** La función ReLU aplicada a un número x se define como el máximo entre 0 y x. Entonces, max{0,x_i} significa que, si X_i es mayor que 0, entonces el resultado es X_i: de lo contrario, el resultado es 0.

De esta manera se describe como la función ReLU que muestra (Daubechies, 2022), donde se aplica a cada componente individual de un vector X, produciendo un nuevo vector donde cada elemento es el máximo entre 0 y el valor correspondiente de X. Esta función ReLU es comúnmente utilizada en redes neuronales para introducir no linealidades y permitir que la red aprenda patrones más complejos, dándonos para valores menores que cero una salida igual a cero y para valores superiores a cero nos arroja el mismo valor de salida.

A continuación, en las ilustraciones 6 y 7 podemos ver unos ejemplos de cómo (Wang, 2020) usa la función ReLU en una CNN.

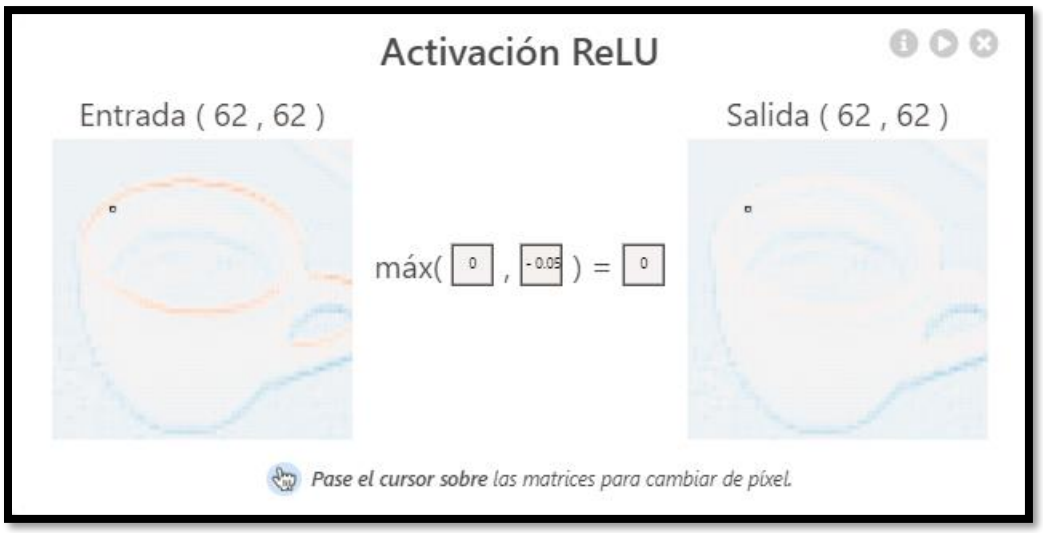

Ilustración 6. Ejemplo visual de cómo actúa la función ReLu sobre una imagen del autor (Wang, 2020).

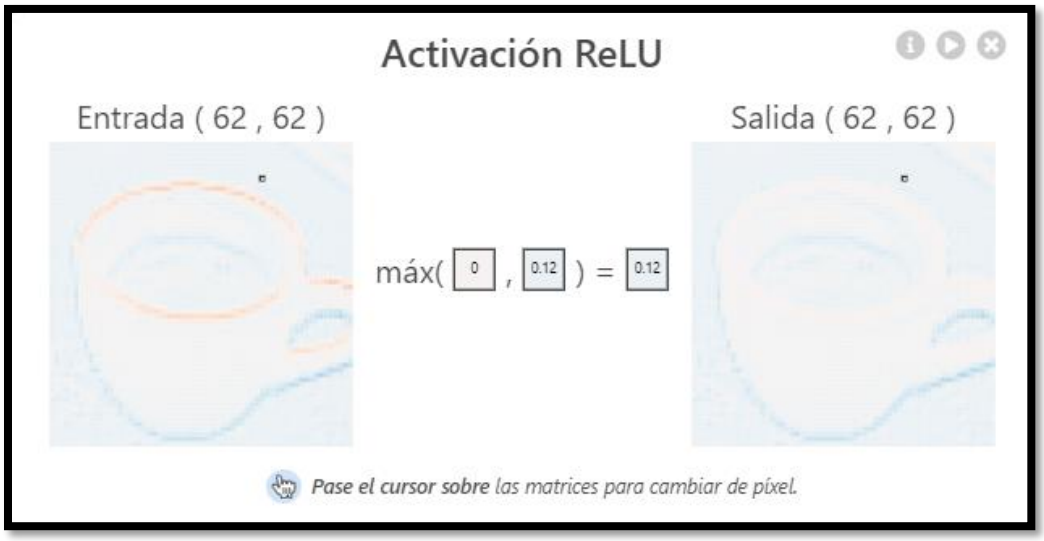

Ilustración 7. Segundo ejemplo visual de cómo actúa la función ReLu sobre una imagen del autor (Wang, 2020)

2.2.2 Softmax:

La fusión softmax que podemos ver en la ilustración 4. En una CNN es una operación que se utiliza al final de la red para convertir las salidas de la red en una distribución de probabilidad. Dicho de otra manera, una operación Softmax tiene como propósito asegurarse de que las salidas de CNN sumen 1. Debido a esto, las operaciones softmax son útiles para escalar las salidas del modelo en probabilidades (Wang, 2020).

Ecuación 4. Función softmax

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

La manera en que la función softmax logra pasar de vectores de entrada a valores probabilísticos lo explica (Banerjee, 2020) en su artículo, donde podemos ver la siguiente ecuación.

Ecuación 5. Del autor (Banerjee, 2020)

$$sm : \mathbb{R}^K \rightarrow \mathbb{R}^K$$

Lo que podemos interpretar de la ecuación 5, es que la función softmax (sm), toma un vector de K dimensiones como entrada y produce otro vector de K dimensiones como salida, manteniéndose dentro del mismo espacio vectorial.

matemáticamente lo que ocurre para para lograr esto es lo que vemos en la ecuación 6 a continuación:

Ecuación 6. Del autor (Banerjee, 2020)

$$sm(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

La cual podemos interpretar como la fórmula representa que representa a la función sm (z) aplicada a un vector Z = ($Z_{1,}$…,Z_K) de K dimensiones, como podría ser Z=(2,1,0), que al sustituirse nos quedaría de la siguiente manera:

$sm(z)_1 = e^2/e^2+e^1+e^0 = 7.389/7.389+2.718+1 = 0.83$

$sm(z)_2 = e^1/e^2+e^1+e^0 = 2.718/7.389+2.718+1 = 0.11$

$sm(z)_3 = e^0/e^2+e^1+e^0 = 1/7.389+2.718+1 = 0.02$

Por lo que nos arrojaría algo como sm(z) = (0.83, 0.11, 0.02) y cada componente de este vector representa la probabilidad de correspondiente categórica, y la suma de todas las componentes es igual a 1, lo que es característico de una función de probabilidad.

La función de softmax es comúnmente utilizada en aprendizaje automático para transformar un conjunto arbitrario de números reales en una distribución de probabilidad, donde cada componente del vector resultante sm(z) representa la probabilidad de la correspondiente categoría.

Gracias a (Wang, 2020) podemos ver cómo funciona la función softmax en una CNN, como podemos apreciarlo en la ilustración.

Ilustración 8. Funcionamiento de la función Softmax en una CNN por (Wang, 2020).

En la ilustración 8 podemos ver como entran las imágenes en forma de vectores de manera que la función softmax calcula la probabilidad de que ciertos componentes pertenezcan a una de las clases establecidas como se explicó en la ecuación 6.

2.3 Pooling Layers:

Comúnmente, una arquitectura CNN emplea esquemas de agrupación convencionales, como la agrupación máxima (max pooling) y la agrupación promedio (average pooling), para obtener un valor máximo o promedio en la capa oculta (Jie, 2020).

En las capas de pooling o capas de agrupación, se buscar disminuir gradualmente la extensión espacial de la red, reduciendo así los parámetros y el cálculo general de la red (Wang, 2020).

Esto es muy útil pues supongamos que, al trabajar con imágenes en una red neuronal, al principio, la imagen tiene una dimensión de 100x100 píxeles y a medida que avanzas en la red, las capas de pooling reducen gradualmente el tamaño de la imagen, por ejemplo, de 100x100 a 50x50 píxeles, luego a 25x25 píxeles y así sucesivamente según sea la arquitectura de la red.

Este proceso de reducción de tamaño ayuda a ahorrar recursos computacionales y memoria, ya que a medida que la imagen se hace más pequeña, también se reduce la cantidad de números (parámetros) que la red debe manejar. Además, al reducir el tamaño espacial, la red puede enfocarse en características más importantes y generales de la imagen, lo que puede hacer que la red sea más eficiente y efectiva en la tarea que esté realizando, como el reconocimiento de objetos en imágenes.

2.3.1 Max pooling

Cuando hablamos de pooling es necesario mencionar que hay dos versiones diferentes del pooling, el max pooling y average pooling. El max Pooling o "agrupación máxima" según lo que nos dice (Jie, 2020), elige el elemento más grande en cada región de agrupación y se representa matemáticamente de la siguiente manera.

Ecuación 7

$$y_{kij} = \max_{(p,\,q)\in R_{ij}} x_{kpq}$$

Y_{kij}: Representa el resultado de la operación de max pooling para el canal k (RGB) al que se está aplicando la operación de pooling, en la posición (i,j) de la salida.

max(p, q) $\in \mathbb{R}ij$: Indica que estamos tomando el valor máximo dentro de una región específica $\mathbb{R}ij$ en la entrada. Esta región esta determinada por las coordenadas (i,j) en la salida.

X_{kpq}: Representa el valor de la entrada en el canal K y en la posición (p, q).

Esto podemos explicarlo de manera que X

Esto lo podemos explicar de la manera que, si tenemos una matriz de entrada X, que bien podría ser nuestra imagen en forma de matriz en un solo canal de color k, podríamos representarlo como se ve a continuación.

$$x = \begin{bmatrix} 1 & 3 & 4 \\ 5 & 2 & 6 \\ 8 & 7 & 9 \end{bmatrix}$$

Al aplicarle max pooling lo que hacemos es pasar un kernel (en este ejemplo de 2x2) que recorre toda la matriz partiendo de las coordenadas establecidas por el valor de y_{ij} para i = 1,2 y j=1,2 la operación se vería así:

Tomamos el máximo en la región de X dada por las coordenadas (1,1), (1,2), (2,1), (2,2). Nos arrojara salidas como las siguientes.

$y_{11} = \begin{bmatrix} 1 & 3 \\ 5 & 2 \end{bmatrix}$ obteniendo un valor máximo de 5

$y_{12} = \begin{bmatrix} 3 & 4 \\ 2 & 6 \end{bmatrix}$ obteniendo un valor máximo de 6

$y_{21} = \begin{bmatrix} 5 & 2 \\ 8 & 7 \end{bmatrix}$ obteniendo un valor máximo de 8

$y_{22} = \begin{bmatrix} 2 & 6 \\ 7 & 9 \end{bmatrix}$ obteniendo un valor máximo de 9

Dándonos un total final al terminar de aplicar el max pooling a toda la red como lo podemos ver en la siguiente ecuación.

$$Y = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

Podemos ver un ejemplo de esto en el trabajo de (Wang, 2020), trabajo del cual obtenemos la siguiente ilustración 9.

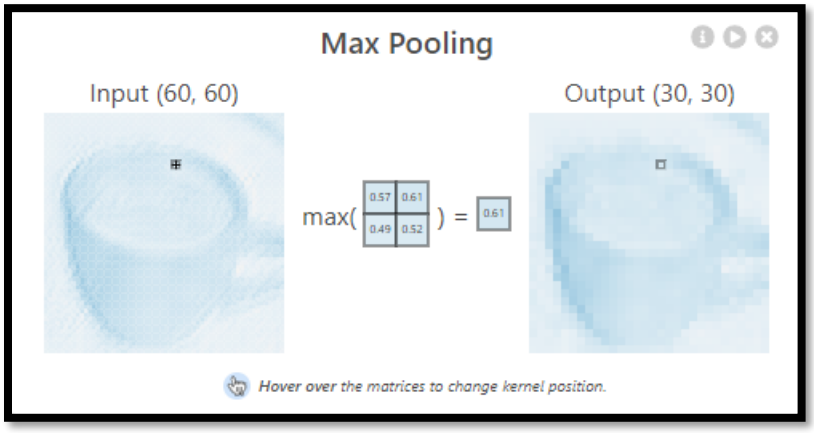

Ilustración 9. Ejemplo visual de cómo funciona la agrupación máxima (Max Pooling) en una red neuronal convolucional por el autor (Wang, 2020).

Imagen donde podemos notar que se utiliza una de las 2 versiones más utilizadas en las CNN que es “max pooling”. En esta operación, se toma el valor máximo de los píxeles dentro de la ventana de agrupación y se asigna como el valor representativo de esa región (Sudholt, 2016). Esto destaca las características más prominentes de la ventana.

2.3.2 Average pooling

Mientras que el average pooling es donde se calcula el promedio de los valores de píxeles dentro de la ventana de agrupación y se usa como el valor representativo como podemos ver en la siguiente ecuación.

Ecuación 8. Average pooling de (Jie, 2020).

$$y_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} x_{kpq}$$

Que podemos explicar suponiendo que tenemos una matriz X como la que podemos ver a continuación.

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Al cual queremos calcular y_{kij} para k = 1 (canal uno de color), en la posición (i,j) = (2,2) y consideramos una región local R_{ij} de tamaño 2x2 alrededor de (i,j):

El filtro nos quedaría de la siguiente manera:

$$R_{ij} = \begin{bmatrix} 6 & 7 \\ 10 & 11 \end{bmatrix}$$

Y a esos valores encerrados en esa ventana aplicamos la ecuación 8 de la siguiente manera:

$Y_{kij} = ¼ (x_{k,2,2} + x_{k,2,3} + x_{k,3,2} + x_{k,3,3})$

$Y_{kij} = ¼ (6+7+10+11)$

$Y_{kij} = 34/4$

$Y_{kij} = 8.5$

Y ese valor de 8.5 representaría el valor promedio de una ventana, posicionada sobre la matriz de una imagen, sacando el promedio de los valores que encierra esa ventana para obtener un solo píxel de esa ventana compuesta por varios pixeles, rediciendo así el trabajo de la CNN y quedándonos solo con las características más importantes.

A continuación, podemos ver un ejemplo rápido de cómo funciona el max pooling y el average pooling en la ilustración 10.

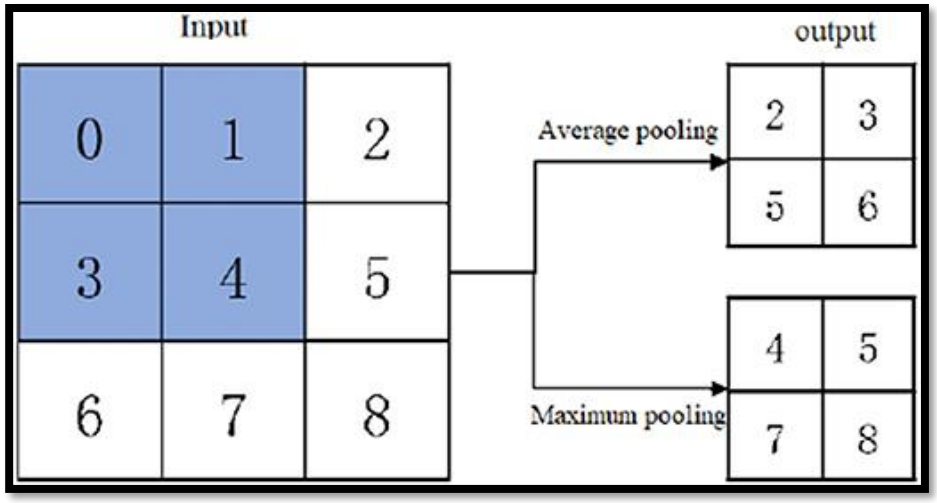

Ilustración 10. Avergare pooling y Max pooling del autor (Huang, 2022).

2.4 Flatten Layer:

En esta capa convierte una capa tridimensional (RGB), de la red en un vector unidimensional, para adaptarse a la entrada de una capa completamente conectada para su clasificación (Wang, 2020). Es decir, cada matriz de cada canal de color se convertirá en un vector y luego todos esos vectores generados por cada canal se harán uno solo, que representa las características extraídas de las imágenes por las capas anteriores. Podemos ver un ejemplo de cómo funciona la capa flatten en la ilustración 11.

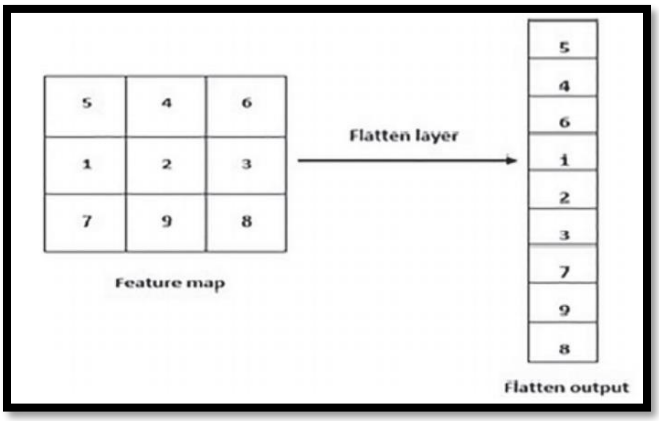

Ilustración 11. Flatten layer por el autor (Suriya, 2022)

2.5 Modelo ResNet:

Para el desarrollo de este trabajo se utilizó el modelo ResNet o “Residual Network”, más específicamente el modelo ResNet-50 (He, 2016), el cual significa que cuenta con 50 capas ocultas utilizadas en su entrenamiento. Todos los modelos ResNet cuentan con la misma arquitectura. Esta arquitectura agrega lo que llama “Residual learning” o bloques residuales (ilustración 12), con el propósito de evitar la “degradación”, palabra que se usa para referirse a ese particular erro que ocurre cuando tratamos de entrenar redes neuronales muy profundas. A menudo es difícil hacer que funcionen correctamente, puesto que a medida que agregamos más y más capas, la red no mejora, sino que su rendimiento empeora.

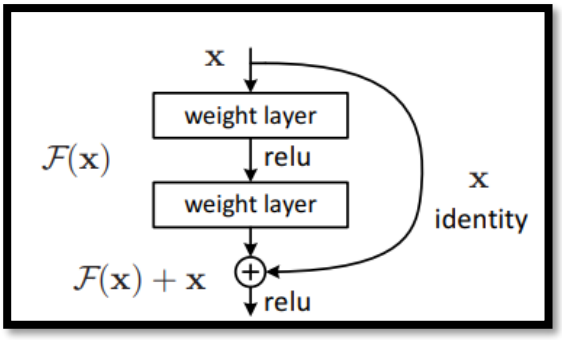

Ilustración 12. Bloque residual.

Según lo visto en este mismo artículo entendemos que la red fue diseñada de la siguiente manera:

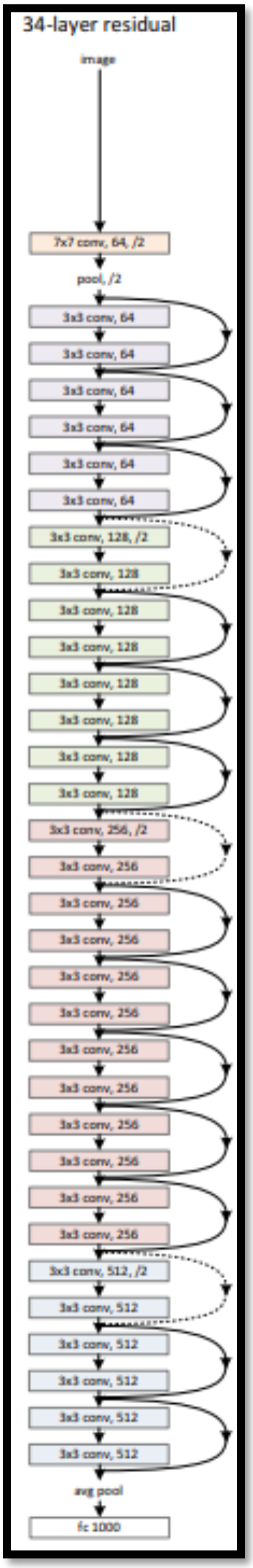

Ilustración 13. Estructura de la red ResNet

Capas Convolucionales:

La red utiliza principalmente filtros de 3x3 en sus capas convolucionales. Esto significa que estas capas buscan patrones en las imágenes usando cuadrados de 3x3 píxeles a la vez. Hay dos reglas de diseño sencillas:

(i) Si dos capas tienen el mismo tamaño de salida, también tendrán la misma cantidad de filtros. Es decir, si están buscando la misma información, usarán la misma cantidad de filtros.

(ii) Si el tamaño del resultado de una capa se reduce a la mitad (por ejemplo, la imagen se hace más pequeña), se duplica la cantidad de filtros para mantener la complejidad de esa capa.

Reducción de Resolución:

La red comienza reduciendo la resolución de las imágenes utilizando capas convolucionales que "saltan" dos píxeles a la vez. Esto ayuda a disminuir la cantidad de información en la imagen y, al mismo tiempo, extrae características importantes.

Capa de Agrupación Promedio Global:

Luego, la red utiliza una capa de "agrupación promedio global". Esta capa toma todas las características y las combina en un solo valor promedio para cada característica. Esto reduce aún más la cantidad de información y ayuda en la clasificación.

Capa Totalmente Conectada con Softmax:

Finalmente, la red utiliza una capa completamente conectada que toma los valores promedio y realiza una operación llamada "softmax". Esta operación asigna probabilidades a diferentes clases para clasificar la imagen.

3. Estado del arte

En el presente capitulo se muestra una serie de investigaciones de manera sintetizada, las cuales fueron recopiladas con respecto a proyectos que han trabajado con redes neuronales convolucionales y visión por computadora en diferentes situaciones, así como plataformas que implementan estos sistemas en la actualidad.

3.1 Aprendizaje profundo

De acuerdo con (Liu, 2017), el creciente desarrollo de técnicas de aprendizaje automático ha sido aplicado ya en una enorme variedad de áreas como el reconocimiento de patrones, lenguaje natural y aprendizaje computacional, siendo este último algoritmo con la capacidad de aprender de los datos, tomando así decisiones y teniendo la capacidad de realizar predicciones, el aprendizaje automático ha tenido un gran crecimiento en diversas áreas. Asimismo, señala que las técnicas tradicionales de aprendizaje automático a menudo no son satisfactorias para imitar los mecanismos de procesamiento humano como lo son el habla y la visión, pero el aun así ha avanzado significativamente, especialmente desde 2006. Menciona además los desafíos en el entrenamiento de redes neuronales, como la retropropagación y el problema de overfitting, además se revisan los últimos desarrollos en redes neuronales profundas, se exploran arquitecturas comunes de aprendizaje profundo y se destacan aplicaciones en visión por computadora, reconocimiento de patrones y reconocimiento de voz. Se discute la utilidad de algoritmos de aprendizaje no supervisados en aplicaciones con grandes conjuntos de datos no etiquetados y se enfatiza la flexibilidad para ajustar el equilibrio entre precisión y complejidad computacional en algoritmos de aprendizaje profundo. Finalmente, se expresa la confianza en que las redes neuronales profundas seguirán recibiendo atención y encontrarán aplicaciones más amplias en el futuro, gracias al rápido desarrollo de recursos de hardware y tecnologías informáticas.

3.2 Redes Neuronales Artificiales

Con aplicación exponencial en la rama de la inteligencia artificial (IA) donde se han desarrollado múltiples investigaciones en los años recientes, en tema de las redes neuronales artificiales (ANN) en una gran diversidad de temas como lo son los siguientes:

Por su parte (Agatonovic-Kustrin, 2000) En su artículo nos ofrece una exhaustiva introducción a las Redes Neuronales Artificiales (ANN) y sus conceptos, presentándolas como programas informáticos que imitan el procesamiento de información en el cerebro humano, estas redes aprenden mediante la detección de patrones y relaciones en los datos. Cada elemento de procesamiento tiene entradas ponderadas, una función de transferencia y una salida. El comportamiento global de la red neuronal está determinado por las funciones de transferencia de sus neuronas, la regla de aprendizaje que rige la optimización de las conexiones durante el entrenamiento, y la arquitectura general de la red. Durante el proceso de entrenamiento, las conexiones entre las unidades se ajustan para minimizar los errores en las predicciones y una vez entrenada y evaluada, la ANN puede aplicarse a nuevos conjuntos de datos para realizar predicciones. En el ámbito farmacéutico, estas redes son especialmente valiosas para modelar relaciones no lineales presentes en diversos procesos. Aunque las ANN no requieren conocimiento previo específico de la fuente de datos, es importante destacar que necesitan conjuntos de entrenamiento extensos.

Además, nos habla de que las ANN tienen la capacidad de integrar tanto datos basados en la literatura como datos experimentales, proporcionando una herramienta versátil para abordar problemas en ciencias farmacéuticas. Sus aplicaciones abarcan desde la interpretación de datos analíticos hasta el diseño de medicamentos y dosis, incluyendo tareas como clasificación, reconocimiento de patrones, predicción y modelado.

En este artículo de (Kurani, 2023), se discuten dos algoritmos significativos para la predicción de acciones: las Redes Neuronales Artificiales (ANN), que no se ven afectadas por la ausencia de algunos puntos de datos, y las Máquinas de Vectores de Soporte (SVM), que evitan el sobreajuste debido a sus límites de decisión simples. El artículo revisa diversas tecnologías aplicadas en la predicción del mercado de valores, como el análisis sentimental, árboles de decisión, algoritmos de promedio móvil y minería de datos. Se examinan estudios recientes como modelos híbridos (ANN-MLP, GARCH-MLP) que han demostrado obtener mejores resultados. Concluye que SVM y ANN han desempeñado roles destacados en abordar estos problemas y sugiere que la integración con otras técnicas novedosas puede resultar en metodologías híbridas más efectivas.

También tenemos trabajos como el de (Wu W. D., 2014) que nos habla de cómo se han aplicado las ANN en muchísimas investigaciones en la industria. El autor está comunicando que el uso ANN en la modelización ambiental y de recursos hídricos ha experimentado un aumento desde principios de la década de 1990. A pesar de que se reconoce la necesidad de un enfoque consistente en el desarrollo de modelos de ANN y la importancia de proporcionar detalles sobre este proceso, hasta el momento no existe un protocolo sistemático que guíe dicho desarrollo.

En este contexto, el artículo presenta un protocolo específico para abordar esta carencia y, además, utiliza este protocolo para realizar una revisión crítica de la calidad de los procesos de desarrollo y presentación de modelos de ANN en 81 artículos publicados desde el año 2000. Estos artículos han utilizado ANNs para modelar la calidad del agua potable.

3.3 Redes Neuronales Profundas

Llegando hasta las redes neuronales densas o profundas (DNN), las cuales han tenido un gran progreso con aplicaciones como el reconocimiento de voz, visión por computadora y análisis de imágenes.

En trabajos como el que nos presenta (Seifert, 2017) , en resumen, destaca que, en los últimos años, las Redes Neuronales Profundas (DNNs) han demostrado superar el estado del arte en diversas áreas, como el reconocimiento visual de objetos, la genómica y el reconocimiento de voz. Sin embargo, debido a la codificación distribuida de la información, las DNNs son difíciles de entender e interpretar. Para abordar este problema, se han utilizado visualizaciones con el propósito de comprender cómo funcionan las arquitecturas profundas en general, qué codifican las diferentes capas de la red, cuáles son las limitaciones del modelo entrenado y para recopilar retroalimentación interactiva del usuario. Aquí el autor presenta un estudio de visualizaciones de DNNs en el campo de la visión por computadora.

Se establece un esquema de clasificación que describe los objetivos y métodos de visualización, así como el área de aplicación. Esta revisión ofrece una visión general de lo que se puede aprender al visualizar DNNs y qué métodos de visualización se utilizaron para obtener determinados conocimientos. Se destaca que la mayoría de los documentos utilizan representaciones de píxeles para mostrar las activaciones de las neuronas. No obstante, recientemente se han propuesto visualizaciones más sofisticadas, como diagramas interactivos de nodos y enlaces. La descripción proporcionada puede servir como una guía al aplicar visualizaciones durante el diseño de DNNs.

Siendo estos últimos temas en los cuales hay una creciente cantidad de investigación que busca las mejores maneras de aplicar las redes neuronales profundas en sus diferentes perspectivas

3.4 Redes Neuronales Convolucionales

Debido al creciente éxito del uso de las DNN en las técnicas de detección de objetos. Las cuales tiene como propósito encontrar objetos de ciertas clases, con localización precisa en una imagen dada y asignar a cada instancia una etiqueta de clase correspondiente.

En esta índole tenemos trabajos como el de (Wu X. S., (2020).) el cual, el autor aborda el problema fundamental de reconocimiento visual en visión por computadora, que es la detección de objetos. A lo largo de las últimas décadas, la detección de objetos ha sido ampliamente estudiada en la comunidad de visión por computadora. Debido a los notables éxitos de las técnicas de aprendizaje profundo en la clasificación de imágenes, las técnicas de detección de objetos que utilizan aprendizaje profundo han sido objeto de estudio activo en los últimos años. En este artículo, el autor presenta un exhaustivo análisis de los avances recientes en la detección visual de objetos mediante aprendizaje profundo. Al revisar una gran cantidad de trabajos recientes en la literatura, se analizan sistemáticamente los marcos existentes para la detección de objetos y se organiza la revisión en tres partes principales: (i) componentes de detección, (ii) estrategias de aprendizaje y (iii) aplicaciones y evaluaciones de modelos de CNN como lo son VGG-16, DakNEt-19, ResNet-101 DenseNet-169, etc.

Por su parte tenemos el trabajo de (Khan, 2018), el cual nos da una guía donde el autor destaca la creciente importancia y eficacia de la visión por computadora en los últimos años, debido a sus diversas aplicaciones en áreas como vigilancia inteligente, salud, deportes, robótica, drones y vehículos autónomos. Las tareas de reconocimiento visual, como clasificación, localización y detección de imágenes son fundamentales en muchas de estas aplicaciones, y los avances recientes en las Redes Neuronales Convolucionales (CNNs) han llevado a un rendimiento excepcional en estas tareas y sistemas de reconocimiento visual de última generación. En este contexto, las CNNs se han convertido en el núcleo de

los algoritmos de aprendizaje profundo en visión por computadora. El autor presenta una guía autocontenida que beneficia a aquellos que desean comprender tanto la teoría detrás de las CNNs como obtener experiencia práctica en su aplicación en visión por computadora. La guía abarca desde conceptos esenciales de redes neuronales, como entrenamiento, regularización y optimización de CNNs, hasta funciones de pérdida, capas de red, arquitecturas populares de CNN y técnicas de evaluación. Además, se mencionan herramientas y bibliotecas comúnmente utilizadas en visión por computadora.

3.5 Redes Neuronales Recurrentes

Podemos empezar por describir estas redes con lo que nos dice en su libro ((Medsker, 2001), donde se habla de la importancia de las redes neuronales recurrentes (RNN) como un foco importante de investigación y desarrollo durante la década de 1990. Estas redes están diseñadas para aprender patrones secuenciales o que varían con el tiempo. Donde una red neuronal recurrente es aquella que tiene conexiones de retroalimentación (bucle cerrado). Ejemplos incluyen BAM, Hopfield, la máquina de Boltzmann y redes de retropropagación recurrente.

Las técnicas de redes neuronales recurrentes se han aplicado a una amplia variedad de problemas. En la década de 1980, se introdujeron redes neuronales parcialmente recurrentes simples por varios investigadores, incluidos Rumelhart, Hinton y Williams, con el propósito de aprender secuencias de caracteres. Muchas otras aplicaciones han abordado problemas que involucran sistemas dinámicos con secuencias temporales de eventos.

En esta rama tenemos investigación como las de (Schmidt, 2019) que nos destaca que las soluciones más avanzadas en áreas como "Modelado de Lenguaje y Generación de Texto", "Reconocimiento de Voz", "Generación de Descripciones de Imágenes" o "Etiquetado de Videos" han estado utilizando Redes Neuronales Recurrentes como base para sus enfoques. Comprender los conceptos subyacentes es, por lo tanto, de tremenda importancia si queremos estar al tanto de

publicaciones recientes o futuras en esas áreas. En este trabajo, se proporciona una breve visión general de algunos de los conceptos más importantes en el ámbito de las Redes Neuronales Recurrentes, lo que permite a los lectores comprender fácilmente los fundamentos, la retropropagación a través del tiempo y unidades de memoria a corto y largo plazo, así como algunos avances más recientes como el mecanismo de atención o redes de punteros. También se ofrecen recomendaciones para lecturas adicionales sobre temas más complejos cuando sea necesario.

En esta rama encontramos tesis que profundizan mucho en las RNN como es el caso del trabajo de (Sutskever, 2013), donde nos señala que este tipo de redes son modelos de secuencia poderosos pero que históricamente se pensaba que eran difíciles de entrenar. La tesis presenta métodos que superan esa dificultad y aplicaciones de RNN a problemas desafiantes.

Propone un nuevo modelo de secuencia que combina diferentes técnicas para hacer las RNN más efectivas y fáciles de entrenar. También introduce un nuevo método de optimización que permite entrenar RNN en tareas con dependencias temporales muy largas, que antes se consideraban casi imposibles. Estos enfoques se aplican con éxito a problemas como modelado del lenguaje y control óptimo, incluso en situaciones con retroalimentación demorada y perturbaciones desconocidas. Además, se presenta una idea novedosa para la inicialización de parámetros que mejora la capacidad de las RNN para aprender problemas con dependencias a largo plazo, desafiando creencias anteriores.

3.6 Visión por Computadora en la actualidad

Encontramos trabajos como el de (Tafti, 2016), en el cual el se aborda el reconocimiento óptico de caracteres (OCR) como un desafío clásico de aprendizaje automático que ha sido un tema de larga data en diversas aplicaciones, como salud, educación, seguros y la industria legal. En la actualidad el OCR se utiliza para convertir varios tipos de documentos electrónicos, como documentos escaneados, imágenes digitales y archivos PDF, en texto completamente editable. Debido a la generación rápida de imágenes digitales a diario, el OCR se destaca como una herramienta imperativa y fundamental para el análisis de datos. También menciona que los sistemas OCR han permitido ahorrar una cantidad significativa de esfuerzo en la creación, procesamiento y almacenamiento de documentos electrónicos, adaptándolos a diferentes propósitos. Existen diferentes plataformas OCR disponibles mencionadas en este trabajo, el cual realizo evaluaciones experimentales cualitativas y cuantitativas utilizando cuatro servicios OCR conocidos, que incluyen Google Docs OCR, Tesseract, ABBYY FineReader y Transym, con el fin de analizar la precisión y confiabilidad de estos paquetes OCR utilizando un conjunto de datos que comprende 1227 imágenes de 15 categorías diferentes. Además, se revisaron las aplicaciones de vanguardia de OCR en informática de la salud.

Como podemos ver en la actualidad hay múltiples desarrollos en la tecnología del reconocimiento óptico de caracteres desarrollados por empresas como Google, Amazon y ABBYY por mencionar algunas. empresas las cuales han desarrollado sus propios sistemas de reconocimiento óptico de caracteres como lo son Google como es el caso de “Textrac” por parte de (AWS, 2023). Amazon en el 2018 presento la herramienta "Amazon Textract", enfocada al OCR para interpretar el contenido y organización en documentos, esta herramienta fue presentada dentro de su evento "re:invent" el cual organiza desde hace 10 años

Amazon Textract devuelven la ubicación y la geometría de líneas y palabras de los elementos que se encuentra, así como también entrega la geometría de tablas, celdas y elementos de selección.

Amazon Textract también nos devuelve 3 categorías de extracción de documentos los cuales son, texto, formularios y tablas. Amazon Textract puede extraer tablas, celdas de tabla, elementos de celdas de tabla y es capaz de devolver los resultados en un archivo JSON, .csv o un archivo.txt. Dándonos tablas en bloques de objetos como se puede apreciar en la ilustración 14.

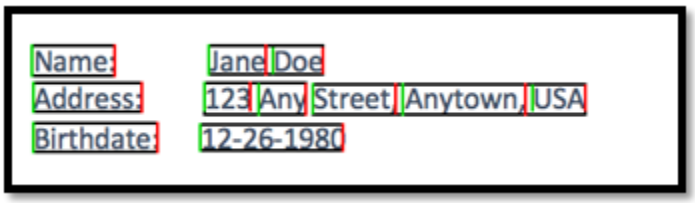

Ilustración 14. Ejemplo de detección de caracteres de (AWS, 2023)

Así como estas características también podemos hablar de muchas otras características que nos ofrece Amazon Textract como lo son la extracción basada en consulta, reconcomiendo de la escritura manuscrita, facturas y recibos, cuadros delimitadores, umbrales de confianza ajustables y flujo de trabajo de revisión humana incorporado.

Con estas aplicaciones podemos ver trabajos como el de (Ianovski, 2023) que nos propone que, implementando OpenCV y Tesseract se puede implementar el procesamiento óptico de caracteres (OCR) con ayuda de AWS sin necesidad de tener amplios conocimientos en ML (Machine Learning). Lo que es de utilidad a la hora de hablar de tiempos de desarrollo, únicamente subir las imágenes al sistema de almacenamiento en la nube de AWS conocido como S3. Ya sea a través de su plataforma en línea o programándolo a utilizando un lenguaje de programación para crear las líneas de comunicación correspondientes.

En la Ilustración15 podemos ver los resultados que nos arroja Amazon textract al cargar una imagen de un vino.

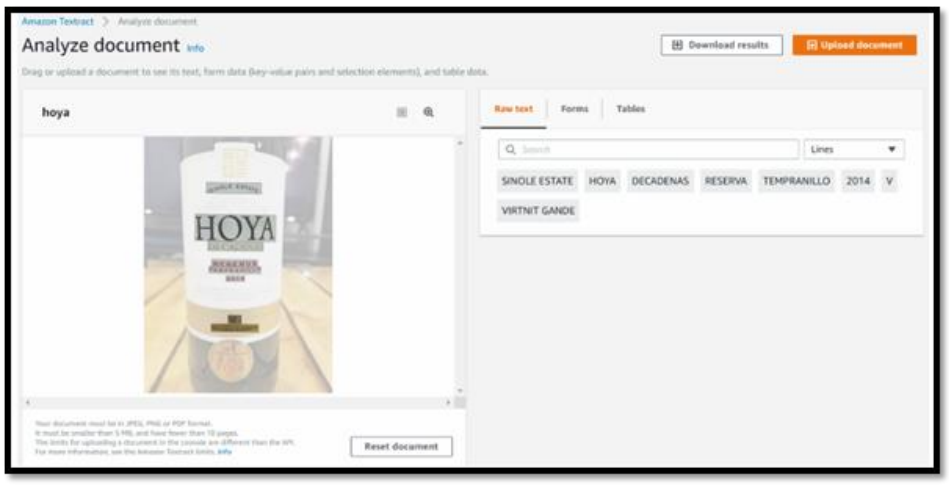

Ilustración 15. Herramienta de demostración en línea de (AWS, 2023).

(GOOGLE, 2023) por su parte en su sitio web nos habla sobre un evento llamado Global Next, igualmente celebrado cada año con la cual pretenden hablar sobre los temas de invocación que hay en desarrollo.

Google divide sus principales avances en estas conferencias en desarrollo de aplicaciones, análisis de datos, ingeniería de datos, arquitectura y desarrollos empresariales, administración de sistemas, seguridad, administración empresarial e innovación.

Entre todas estas categorías cabe destacar una de particular interés para esta investigación la cual se llama visión AI, la cual es una API (Application Progamming Interface) desarrollada para detectar emociones y comprender textos con modelos preentrenados, capaces de utilizar el aprendizaje automático para entender las imágenes con mucha precisión, detectar etiquetas personalizadas, objetos, caras, leer texto manuscrito y consigue metadatos de imágenes como se puede apreciar en las ilustraciones 16 a 19.

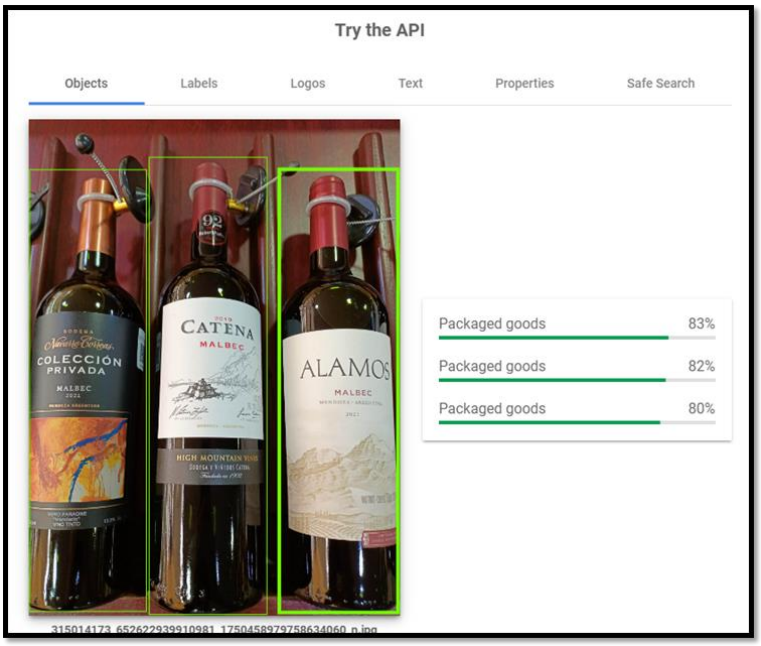

Ilustración 16. Segmentación de vinos realizado con la API "Vision AI" de (GOOGLE, 2023)

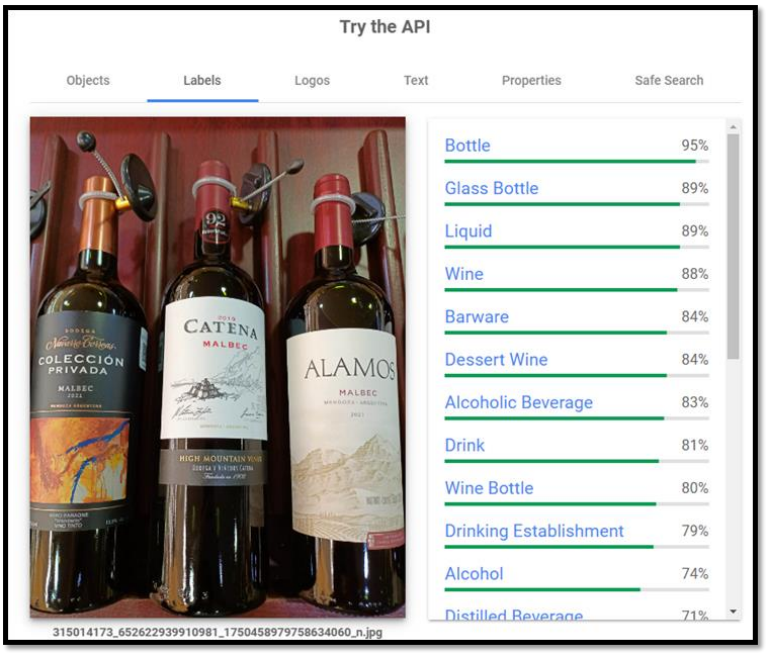

Ilustración 17 . Etiquetado de caracter´ısticas de los objetos a trav´es de la API "Vision AI" de (GOOGLE, 2023).

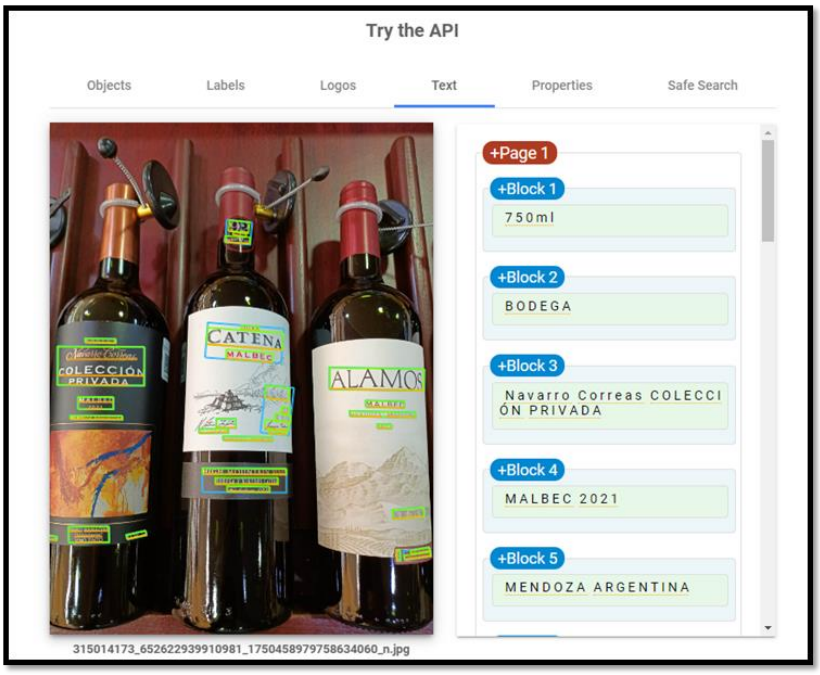

Ilustración 18. Textos reconocidos a través de la API "Vision AI" de (GOOGLE, 2023).

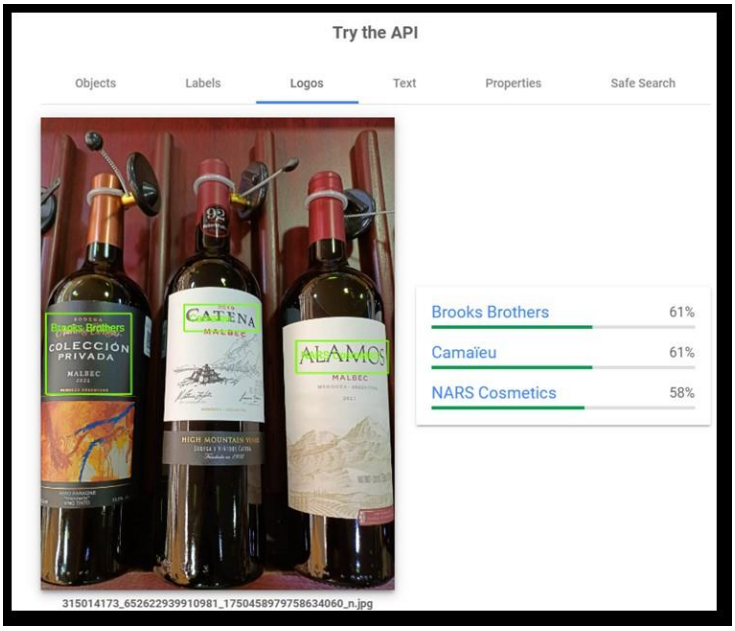

Ilustración 19. Reconocimiento de logos a través de la API" Vision AI" de (GOOGLE, 2023)

Un trabajo importante es el de (Hegghammer, 2022) en el cual, el autor aborda el tema del Reconocimiento Óptico de Caracteres (OCR) y cómo puede abrir documentos históricos poco estudiados al análisis computacional. Sin embargo, la precisión del software OCR varía. En este artículo, se informa sobre un experimento de evaluación comparativa que compara el rendimiento de Tesseract, Amazon Textract y Google Document AI en imágenes de texto en inglés y árabe, utilizando escaneos de libros en inglés y escaneos de artículos en árabe, replicados 43 veces con diferentes tipos de ruido artificial, donde “Document AI” entregó los mejores resultados, y los procesadores basados en servidor (Textract y Document AI) tuvieron un rendimiento sustancialmente mejor que “Tesseract”, especialmente en documentos con ruido. La precisión para el inglés fue considerablemente mayor que para el árabe. De manera general fue un estudio comparativo muy útil para entender cuál es el rendimiento de estos productos de OCR.

No obstante, las herramientas que brinda tanto Amazon como Google no son las únicas para enfocaros en el OCR. En este trabajo preliminar hemos encontrado quienes usan ANN, CNN y RNN. Estas técnicas fueron comparadas en el trabajo de (Memon, 2020) donde se muestran excelentes resultados en OCR. Articulo el cual cita 57 trabajos dedicaos solamente al OCR desde el 2017 hasta al 2019.

S.No	Script	Technique Employed		Year	Ref
28	Arabic	Multi-Channel Neural Network (MCNN)		2018	[207]
29	Arabic	Fast Automatic Hashing Text Alignment (FAHTA)		2018	[208]
30	Arabic	Deep Siamese Convolutional Neural Network and SVM		2018	[144]
31	Arabic	A hybrid machine learning approach that utilizes neighborhood rough sets with a b	whale optimization	2019	[178]
32	Arabic	Convolutional Neural Networks(CNN)		2019	[71]
33	Arabic	Classifier Fusion Technique based on fusion of features MI, RLM, SFIH and WD a	ssifiers MQDF, SVM and	2019	[79]
		RF			
34	Urdu	Hierarchical combination of Convolutional Neural Networks (CNN), Multi-dimenl	Long Short-Term Mem-	2017	[190]
		ory Neural Networks (MDLSTM)			
35	Urdu	BDLSTM (Bi-Directional Long Short-Term Memory), Recurrent Neural Network	)	2018	[155]
36	Urdu	Histogram of Oriented Gradient (HOG), Support Vector Machine (SVM), k Near	eighbors (kNN), Random	2018	[88]
		Forest (RF) and Multi-Layer Perceptron (MLP)			
37	Urdu	Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM)		2018	[154]
38	Urdu	1-Dimensional BLSTM Classifier based on RNN, LSTM and BRNN		2019	[156]
39	Urdu	Deep Neural Network with dropout regularization		2019	[157]
40	Urdu	Random Forest and Logistic Regression Algorithm		2019	[209]
41	Urdu	Cascade Forward Backpropagation Neural Network		2019	[210]
42	Indian	Multi-column Multi-scale Convolutional Neural Network (MMCNN)		2017	[211]
43	Indian	Convolutional Neural Network (CNN)		2018	[116]
44	Indian	Convolutional Neural Network (CNN)		2018	[138]
45	Indian	Histogram of oriented gradient (HOG) and Support Vector Machine (SVM)		2018	[115]
46	Indian	Convolutional Recurrent Neural Network (CRNN) with Spatial Transformer Netw	TN) layer	2018	[179]
47	Indian	Zoning, Discrete Cosine Transformations (DCT), gradient features, k Nearest Nei	rs (kNN), Support Vector	2018	[2]
		Machine (SVM), Decision Tree and Random Forest			
48	Indian	Tesseract OCR and google multilingual OCR		2019	[113]
49	Indian	SVM for classification and features based on geometrical properties of the characte	proposed for recognition	2019	[77]
50	Indian	SVM for classification and modified histogram of oriented gradients(HOG) for rec	on	2019	[78]
51	Indian	Convolutional Neural Network (CNN)		2019	[114]
52	Indian	Convolutional Neural Network (CNN)		2019	[70]
53	Indian	Deep Belief Network with the distributed average of gradients feature		2019	[188]
54	Indian	Modified Neural Network with aid of elephant herding optimization		2019	[189]
55	Indian	VGG (Visual Geometry Group) with 16 convolutional layers		2019	[117]
56	Indian	SVM classifier with the polynomial and linear kernel		2019	[80]
57	Persian	Chain Code Histogram (CCH), transition information in the vertical and horizo	irections, Support Vector	2017	[75]
		Machine(SVM)			
58	Persian	Zoning, chain code, outer profile, crossing count, k Nearest Neighbors (kNN), A	ial Neural Networks and	2018	[87]
		Support Vector Machine (SVM)			
59	Persian	Convolutional Neural Network (CNN)		2018	[212]

Ilustración 20 tabla de trabajos investigados por (Memon, 2020)

En la ilustración 20 podemos ver una parte de la tabla publicada en este artículo. Es evidente que para el reconocimiento óptico de caracteres la mayoría de las investigaciones se enfocan el aprendizaje profundo, especialmente con CNN puesto que se han implementado cada vez más seguido para el reconocimiento de cráteres tanto en tipografías como en caracteres manuscritos y en de manera general para trabajar con imágenes.

4. Desarrollo e implementación

Este capítulo inicia con la descripción del diseño código en el lenguaje de programación Python, además del proceso para crear la base de datos que fue necesaria hacer para poner a prueba el código.

4.1 Diseño del código:

4.1.1 Librerías

```
import os
import cv2
import numpy as np
from collections import Counter
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, roc_curve, auc
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix

# Importar el modelo preentrenado ResNet50
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
```

Ilustración 21. Librerías de Python

os: Proporciona una interfaz para interactuar con el sistema operativo. En este código, se utiliza para trabajar con directorios y archivos.

cv2 (OpenCV): OpenCV (Open Source Computer Vision) es una biblioteca de visión por computadora que proporciona herramientas para procesar imágenes y videos. En este código, se usa para leer y redimensionar imágenes.

numpy: Biblioteca para realizar operaciones matemáticas y manipulación de matrices. En este código, se utiliza para realizar operaciones en matrices de imágenes y etiquetas.

collections: Proporciona clases especializadas para contar objetos. En este código, se utiliza para contar las etiquetas y analizar la distribución de clases.

sklearn: Scikit-learn es una biblioteca para aprendizaje automático y análisis de datos. En este código, se utiliza para dividir el conjunto de datos en pliegues (folds) y evaluar el rendimiento del modelo.

matplotlib.pyplot: Biblioteca para crear visualizaciones y gráficos en Python. En este código, se utiliza para trazar la curva ROC y otras visualizaciones.

tensorflow y keras: TensorFlow es una biblioteca de código abierto para aprendizaje automático y aprendizaje profundo. Keras es una interfaz de alto nivel para construir y entrenar modelos de aprendizaje profundo. En este código, se utiliza para construir y entrenar modelos de clasificación de imágenes.

seaborn: Seaborn es una biblioteca de visualización de datos basada en Matplotlib. En este código, se utiliza para trazar la matriz de confusión.

ResNet50 (TensorFlow): Modelo de red neuronal convolucional preentrenado en grandes conjuntos de datos de imágenes. En este código, se utiliza como modelo base para la clasificación de imágenes.

preprocess_input (ResNet50): Función específica de ResNet50 para preprocesar imágenes antes de pasarlas al modelo.

4.1.2 Rutas y dimensiones

```
# Define las rutas a las carpetas de imágenes
path_vinos = "D:\MAESTRIA\Tesis\Bases_de_datos\caberne"
path_otros = "D:\MAESTRIA\Tesis\Bases_de_datos\otros"

# Define el tamaño deseado para todas las imágenes
ancho_deseado = 100
alto_deseado = 100
```

Ilustración 22. Ruta y dimensión de las imágenes

Cargamos la ruta de las imágenes y definimos el ancho y alto al que redimensionaremos las imágenes contenidas en esa ruta.

4.1.3 Etiquetas

```
# Crea listas vacías para almacenar las imágenes y sus etiquetas
imagenes = []
etiquetas = []
```

Ilustración 23. lista de imágenes y etiquetas

Creamos listas vacías fuera del ciclo For, a las cuales iremos agregando a la lista de “imágenes” cada imagen leída y a la lista de “etiquetas” cada etiqueta asignada. Esta última puede ser un “1” o un “0” dependiendo de si es vino u otro objeto, donde “1” será igual a vinos y “0” será igual a otro objeto.

4.1.4 Etiquetado de imágenes

Comenzamos a crear un Loop For para etiquetar las imágenes contenidas en un directorio (Carpeta) de la siguiente manera.

```
for filename in os.listdir(path_vinos):
    # Lee la imagen
    img = cv2.imread(os.path.join(path_vinos, filename)) #<--(1)
```

Ilustración 24

Este código se lee de la siguiente manera …

Por cada “**filname**” (nombre de archivo) en el directorio “**path_vinos**” (ruta definida anteriormente) hacer lo siguiente

Has uso de “**os.path.join(path_vinos, filename**” el cual combinara el directorio “**path_vinos**” y el nombre del archivo” **filename**” utilizando la función “**os.path.join()**”. Esta función asegura que la ruta se forme correctamente. Por ejemplo, si “**path_vinos**” es

“D:\MAESTRIA\Tesis\Bases_de_datos\caberne “

y de “**filename**” es "**imagen.jpg**", por lo que “**os.path.join(path_vinos, filename)**” devolverá

“D:\MAESTRIA\Tesis\Bases_de_datos\caberne\imagen.jpg”

“**cv2.imread(os.path.join (path_vinos, filename))**” llama a la función “**cv2.imread()**” de la biblioteca OpenCV pasándole la ruta completa de la imagen. La función” **cv2.imread()**” carga la imagen en la memoria y devuelve una matriz **NumPy** que representa la imagen.

El resultado de **cv2.imread()** se asigna a la variable img. Ahora, la variable img contiene la matriz **NumPy** que representa la imagen cargada.

Hasta este punto podemos pedir visualizar la imagen la cual ya ha sido leída y convertida en una matriz de numpy, cabe mencionar que esta matriz está compuesta de la información de los 3 canales de color de cada píxel de manera que cada fila tiene la información de color de un solo píxel y así sucesivamente, como podemos verlo en las siguientes imágenes.

Ilustración 25. ejemplo de imagen vino con la que trabaja la red neuronal.

```
la matriz de la imagen original sin redimensionar es..
[[[60 91 88]
  [60 91 88]
  [60 91 88]
  ...
  [28 45 42]
  [28 45 42]
  [28 45 42]]

 [[60 91 88]
  [60 91 88]
  [60 91 88]
  ...
  [28 45 42]
  [28 45 42]
  [28 45 42]]

 [[60 91 88]
  [60 91 88]
  [60 91 88]
  ...
  [28 45 42]
  [28 45 42]
  [28 45 42]]
```

Ilustración 26 matriz de la imagen

En resumen, la matriz que ves representa la imagen en su formato matricial, donde cada elemento corresponde a un píxel individual y contiene información sobre los componentes de color RGB de ese píxel. La imagen en sí está compuesta por una colección de píxeles, y cada píxel es representado por tres valores (R, G y B) que determinan su color o intensidad. No son matrices individuales, sino simplemente la forma en que se almacenan y organizan los píxeles en una imagen digital.

```
# Cambia el tamaño de la imagen
img = cv2.resize(img, (ancho_deseado, alto_deseado))
```

Ilustración 27. Redimensionamiento de la imagen.

Aquí podemos ver la matriz ahora ajustada al ancho y alto deseado esto con la función "**cv2resize()**" la cual usa una interpolación bilineal. Donde la palabra "interpolación" se refiera a aumentar o disminuir la cantidad de pixeles de una imagen, pero sin añadir o quietar información, es el equivalente a acomodarlo de manera distinta. En esta técnica, se calcula el valor de un nuevo píxel tomando en cuenta los cuatro píxeles más cercanos en la imagen original y ponderando sus valores según su distancia al nuevo píxel. Esta técnica proporciona resultados más suaves y menos artefactos que la interpolación vecino más cercano

Ilustración 28. Ejemplo de imagen redimensionada.

```
# Preprocesa la imagen para ResNet50
img = preprocess_input(img)
```

Ilustración 29. procesar la imagen para que la red ResNet pueda trabajar con ella.

Cabe mencionar que el preprocesamiento específico para ResNet50 generalmente incluye dos pasos principales:

1. **Centrar la imagen en cero:**

La media de los canales RGB en el conjunto de datos de ImageNet se resta de cada píxel de la imagen. Esto ayuda a centrar la distribución de píxeles alrededor de cero, lo que puede mejorar la convergencia durante el entrenamiento.

2. **Escala de los píxeles:**

La escala de los píxeles se ajusta dividiendo cada píxel por la desviación estándar de los canales RGB en el conjunto de datos de ImageNet. Esto normaliza los valores de píxeles para que estén en un rango más pequeño.

En términos de código, estas operaciones se realizan mediante la función preprocess_input proporcionado por la biblioteca Keras (importada desde tensorflow.keras.applications.resnet50).

Esta función preprocess_input realiza las operaciones descritas anteriormente, y es parte de la interfaz de Keras para trabajar con modelos preentrenados como ResNet50. Es importante mencionar que el preprocesamiento específico puede variar entre diferentes modelos y conjuntos de datos, pero este enfoque generalmente es aplicable a muchos modelos preentrenados en ImageNet.

```
# Agrega la imagen y su etiqueta a las listas
imagenes.append(img)
etiquetas.append(1)
```

Ilustración 30. Se asignan a las imágenes su etiqueta

Para terminar con el ciclo agregamos la etiqueta deseada a cada imagen correctamente leída en el archivo o carpeta.

```
for filename in os.listdir(path_otros):
    img = cv2.imread(os.path.join(path_otros, filename))
    img = cv2.resize(img, (ancho_deseado, alto_deseado))
    img = preprocess_input(img)  # Preprocesa la imagen para ResNet50
    imagenes.append(img)
    etiquetas.append(0)
```

Ilustración 31. ciclo for de la segunda etiqueta.

4.1.5 Arreglo Numpy

Repetimos todo el ciclo for para el segundo archivo o carpeta que deseamos etiquetar con alguna otra etiqueta, en este caso se asignó el valor “0” para los objetos que no eran vinos

```
imagenes = np.array(imagenes, dtype=np.float32)
etiquetas = np.array(etiquetas)
```

Ilustración 32. Arreglos NumPy

Luego se convierte la lista ”imágenes” en un array de NumPy. El argumento” imagenes“ es la lista que contiene las matrices de las imágenes redimensionadas. El argumento dtype=np.float32 se utiliza para especificar el tipo de datos de los elementos en el array. En este caso, se convierten los valores a float32, lo que significa que todos los valores de píxeles se representarán como números de punto flotante de 32 bits en lugar del tipo de datos predeterminado de NumPy, que suele ser int32 o uint8.

Mientras que np.array(etiquetas), convierte la lista etiquetas en un array de NumPy. El argumento etiquetas es la lista que contiene las etiquetas (1 o 0) correspondientes a cada imagen.

Todo esto se realiza porque al convertir las listas en arrays de NumPy, se obtiene una estructura de datos más adecuada para el procesamiento numérico y

el trabajo con imágenes en el contexto de procesamiento de datos y aprendizaje automático. Además, proporciona más opciones y herramientas para realizar operaciones numéricas y análisis de datos de manera eficiente.

4.1.6 K-Folds

```
# Define el número de pliegues (folds) que deseas utilizar
num_folds = 3
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
```

Ilustración 33. K- Folds

Esta parte del código que podemos ver en la **Ilustración** está utilizando la clase KFold de scikit-learn para realizar la validación cruzada del conjunto de datos. Donde podemos ver lo siguiente

num_folds = 3: Se define el número de pliegues o "folds". La validación cruzada divide el conjunto de datos en un número específico de pliegues, y en este caso, se han elegido 3 pliegues. Cada pliegue se utilizará alternativamente como conjunto de prueba, mientras que los demás se utilizarán como conjuntos de entrenamiento.

kf = KFold(n_splits=num_folds, shuffle=True, random_state=42): Se crea una instancia de la clase KFold con los siguientes parámetros:

n_splits=num_folds: Indica el número de pliegues en los que se dividirá el conjunto de datos. En este caso, es 3.

shuffle=True: Indica que los datos se barajarán antes de dividirlos en pliegues. Esto es útil para garantizar que las clases estén distribuidas de manera uniforme en cada pliegue, especialmente si el conjunto de datos está ordenado de alguna manera.

random_state=42: Proporciona una semilla para la aleatorización, asegurando que los pliegues sean consistentes entre ejecuciones.

4.1.7 Listas

Previo al loop for vamos creamos unas listas para almacenar las puntuaciones de precisión de cada fold y las predicciones de probabilidad como podemos verlo en la ilustración 34.

```
accuracy_scores = []
all_y_true = []
all_y_prob = []
all_y_pred = []
```

Ilustración 34. Listas previas al loop

4.1.7 Loop for principal

Al comenzar con el loop for lo primero que hacemos es implementar la lógica de un bucle que itera sobre los pliegues generados por la validación cruzada (KFold), como podemos verlo en la ilustración 35.

```
for train_index, test_index in kf.split(imagenes):
    X_train, X_test = imagenes[train_index], imagenes[test_index]
    y_train, y_test = etiquetas[train_index], etiquetas[test_index]
```

Ilustración 35. validación cruzada

Aquí hay una explicación más detallada de lo que ocurre en esta parte del Código:

for train_index, test_index in kf.split(imagenes): Este bucle se ejecutará para cada pliegue generado por la validación cruzada. “kf.split(imagenes)” devuelve índices para los conjuntos de entrenamiento y prueba en cada pliegue.

X_train, X_test = imagenes[train_index], imagenes[test_index]: Utiliza los índices obtenidos “(train_index y test_index)” para dividir el conjunto de imágenes (imagenes) en conjuntos de entrenamiento (X_train) y prueba (X_test). X_train contendrá las imágenes utilizadas para entrenar el modelo en la iteración actual del bucle, y X_test contendrá las imágenes utilizadas para evaluar el modelo.

y_train, y_test = etiquetas[train_index], etiquetas[test_index]: Similar al paso anterior, utiliza los mismos índices para dividir las etiquetas (etiquetas) en conjuntos de entrenamiento (y_train) y prueba (y_test). y_train contendrá las etiquetas correspondientes a las imágenes en X_train, y y_test contendrá las etiquetas correspondientes a las imágenes en X_test.

En resumen, este bucle está estructurado de manera que, en cada iteración, se selecciona un conjunto diferente de índices para entrenamiento y prueba según la configuración de la validación cruzada. Luego, se extraen las imágenes y etiquetas correspondientes a esos índices, y se utilizan para entrenar y evaluar el modelo en esa configuración particular de entrenamiento/prueba. Este proceso se repite para cada pliegue generado por la validación cruzada, lo que proporciona una evaluación más robusta del rendimiento del modelo al considerar diferentes divisiones de los datos.

A continuación, esta parte del código se encarga de cargar el modelo preentrenado ResNet50, ilustración 36.

```
# Define el modelo
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(alto_deseado, ancho_deseado, 3)),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(2, activation='softmax')
])
```

Ilustración 36. Código del modelo ResNet50.

En esta parte del código, se está definiendo un modelo de red neuronal utilizando "**Keras**", una biblioteca de aprendizaje profundo de alto nivel que se ejecuta sobre "**TensorFlow**". El modelo se construye como una secuencia de capas en una red neuronal secuencial.

Empezamos creando el modelo escribiendo "**Keras.sequential**" el cual tendrá dentro las capas siguientes.

keras.layers.Flatten(input_shape=(alto_deseado, ancho_deseado, 3)): Esta es la primera capa del modelo. La función "**Flatten"** se utiliza para convertir los datos de entrada en una sola dimensión (**1D**). En este caso, se espera que las imágenes tengan una forma de (**alto_deseado, ancho_deseado, 3**), que corresponde al **alto y anchos deseados** de las imágenes redimensionadas y 3 canales de color (RGB). La capa Flatten "**aplana**" la entrada para que pueda ser alimentada a las siguientes capas densas.

keras.layers.BatchNormalization(): Esta es la segunda capa del modelo y se utiliza para normalizar las activaciones de la capa anterior. La normalización de lotes (**Batch Normalization**) es una técnica que ayuda a estabilizar y acelerar el entrenamiento de redes neuronales, normalizando las activaciones en cada lote de datos durante el entrenamiento.

keras.layers.Dense(128, activation='relu'): Esta es una capa densa con **128 neuronas (nodos)** y función de activación **ReLU** (**Rectified Linear Unit**). La capa densa está completamente conectada, lo que significa que cada neurona en esta capa se conecta con todas las neuronas de la capa anterior. La función de activación **ReLU** se utiliza para introducir no linealidades en la red y permitir que el modelo aprenda representaciones complejas.

keras.layers.Dense(2, activation='softmax'): Esta es la última capa del modelo, que también es una capa densa. Tiene 2 neuronas, lo que significa que el modelo tiene 2 salidas posibles (clasificación binaria). La función de activación **softmax** se utiliza en esta capa para obtener probabilidades de clase normalizadas para las salidas. La capa **softmax** garantiza que la suma de las probabilidades para todas las clases sea igual a 1, lo que facilita la interpretación de las predicciones del modelo como distribuciones de probabilidad.

En resumen, el modelo definido consta de una capa de entrada Flatten, seguida de una capa de normalización de lotes, dos capas densas y una capa de salida con función de activación softmax. Este tipo de modelo es comúnmente

utilizado para tareas de clasificación binaria en imágenes, donde las imágenes se aplanan y luego se pasan por capas densas para realizar el aprendizaje en los datos y generar probabilidades de clasificación.

Mientras que en la Ilustración 37 se ve como se encarga de crear el modelo completo al combinar el modelo base (ResNet50) preentrenado con capas adicionales para la clasificación específica la tarea deseada, como podemos ver a continuación:

```
# Crear el modelo completo
model = keras.models.Model(inputs=base_model.input, outputs=predictions)
```

Ilustración 37. Modelo completo

keras.models.Model: Esta es una clase en Keras que se utiliza para instanciar modelos secuenciales y modelos funcionales. En este caso, estás creando un modelo funcional.

inputs=base_model.input: Establece las entradas del modelo funcional. Aquí, base_model.input representa la capa de entrada del modelo base ResNet50. Estás conectando la entrada de tu nuevo modelo a la misma entrada que la del modelo ResNet50 preentrenado.

outputs=predictions: Establece las salidas del modelo funcional. predictions se refiere a la capa densa final que has definido anteriormente para realizar la clasificación en tu tarea específica. Estás conectando las salidas de esta capa a las salidas del nuevo modelo.

model = keras.models.Model(...): Finalmente, se crea el modelo funcional completo al especificar las entradas y salidas. Este nuevo modelo incluye tanto el modelo base ResNet50 como la capa densa de clasificación que has agregado.

Así esta línea de código está creando un nuevo modelo que combina el modelo base ResNet50 preentrenado con una capa densa adicional para realizar la clasificación específica deseada, que en este caso es la clasificación de imágenes

de vinos. Este enfoque te permite reutilizar el conocimiento aprendido por ResNet50 en tareas de clasificación de imágenes, al tiempo que ajustas el modelo para adaptarse a la tarea específica que estás abordando.

```
# Congelar las capas del modelo base (ResNet50)
for layer in base_model.layers:
    layer.trainable = False
```

Ilustración 38. Código de capas congeladas.

En la ilustración 38 podemos ver la parte del código que se encarga de "congelar" las capas del modelo base ResNet50. Congelar una capa significa que los pesos de esa capa no se actualizarán durante el entrenamiento, más específicamente ocurre lo siguiente:

for layer in base_model.layers: Esto itera a través de todas las capas del modelo base ResNet50. base_model.layers es una lista que contiene todas las capas del modelo.

layer.trainable = False: Para cada capa en el bucle, se establece la propiedad trainable en False. Esto significa que los pesos de esa capa no se actualizarán durante el entrenamiento del modelo.

Cuando se utiliza un modelo preentrenado como ResNet50, congelar sus capas es una práctica común. Las capas iniciales de un modelo preentrenado, especialmente en modelos profundos como ResNet50, aprenden representaciones de bajo nivel que son útiles para muchas tareas, y congelarlas permite retener ese conocimiento preentrenado.

Al congelar estas capas, el modelo se entrena principalmente en las capas adicionales agregadas para adaptarse a la tarea específica. Esto ayuda a evitar que los gradientes durante la retropropagación afecten las capas preentrenadas, lo que

podría llevar a perder la información aprendida durante el entrenamiento en ImageNet.

Esta parte del código asegura que solo las capas adicionales que se han agregado para la clasificación de vinos se entrenen, mientras que las capas del modelo base ResNet50 se mantienen congeladas y no se actualizan durante el entrenamiento.

```
# Compilar el modelo
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Ilustración 39. Compilación del modelo.

Mientras que en la ilustración 39 vemos como esta parte del código se encarga de compilar el modelo antes de comenzar el entrenamiento, que podemos ver en seguida de manera más detallada:

model.compile: Este método de Keras se utiliza para configurar el proceso de entrenamiento del modelo.

optimizer='adam': Especifica el optimizador a utilizar durante el entrenamiento. En este caso, se está utilizando el optimizador Adam. Adam es un optimizador de descenso de gradiente estocástico que ajusta automáticamente la tasa de aprendizaje durante el entrenamiento.

loss='sparse_categorical_crossentropy': Establece la función de pérdida que se minimizará durante el entrenamiento. En este caso, la función de pérdida es la entropía cruzada categórica escasa (sparse categorical crossentropy). Esta función de pérdida es adecuada para problemas de clasificación donde las etiquetas son enteros y no necesitan estar codificadas en un formato

one-hot.metrics=['accuracy']: Especifica las métricas que se deben evaluar durante el entrenamiento y la evaluación del modelo. En este caso, se está utilizando la métrica de precisión ('accuracy'), que mide la proporción de predicciones correctas en relación con el número total de predicciones.

Y es de esta manera que “model.compile” configura el modelo para el entrenamiento al especificar el optimizador, la función de pérdida y las métricas a utilizar. Una vez que el modelo está compilado, está listo para ser entrenado utilizando los datos proporcionados.

En la Ilustración 40 podemos ver el código que inicia el entrenamiento y el número de épocas seleccionado para nuestra red.

```
# Entrenar el modelo
history = model.fit(X_train, y_train, epochs=10)
```

Ilustración 40. Entrenamiento del modelo.

Dicho de otra manera, esta línea de código se utiliza para entrenar el modelo con los datos de entrenamiento (X_train y y_train). El codigo “model.fit(X_train, y_train, epochs=10):” se descompone en:

model: Se refiere al modelo que has creado y compilado previamente, que incluye el modelo base ResNet50 y las capas adicionales para tu tarea específica.

X_train: Representa las imágenes de entrenamiento que se utilizarán para entrenar el modelo.

y_train: Son las etiquetas correspondientes a las imágenes de entrenamiento.

epochs=10: Indica la cantidad de épocas, es decir, el número de veces que el modelo pasará por todos los datos de entrenamiento. En este caso, se ha establecido en 10 épocas.

Durante la ejecución de model.fit, ocurre lo siguiente:

En cada época, el modelo toma las imágenes de entrenamiento (X_train) y las etiquetas correspondientes (y_train) para ajustar sus pesos y aprender a realizar la tarea de clasificación específica.

Durante cada época, el modelo realiza un pase hacia adelante (forward pass) utilizando las imágenes de entrenamiento y calcula la pérdida utilizando la función de pérdida especificada durante la compilación (sparse_categorical_crossentropy en este caso).

Luego, realiza un pase hacia atrás (backward pass) para calcular los gradientes y ajustar los pesos del modelo utilizando el optimizador especificado (adam en este caso). Este proceso se repite durante las 10 épocas especificadas.

El historial del entrenamiento se almacena en la variable history, que puede utilizarse posteriormente para visualizar las métricas de entrenamiento a lo largo del tiempo, como la pérdida y la precisión. es decir, es esencial para el proceso de entrenamiento del modelo, donde el modelo aprende a hacer predicciones más precisas a medida que se ajustan sus pesos utilizando los datos de entrenamiento.

```
# Evaluar el modelo en el conjunto de prueba
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
accuracy_scores.append(test_acc)
```

Ilustración 41

En esta parte del código Ilustración 41, se evalúa el modelo en el conjunto de prueba después de que ha sido entrenado de la siguiente manera:

model: Se refiere al modelo que se ha creado y entrenado previamente.

X_test: Representa las imágenes del conjunto de prueba que se utilizarán para evaluar el modelo.

y_test: Son las etiquetas correspondientes a las imágenes del conjunto de prueba.

verbose=0: Controla la cantidad de información que se muestra durante la evaluación. Al establecerlo en 0, se suprime la salida detallada durante el proceso de evaluación.

Durante la ejecución de model.evaluate. El modelo realiza un pase hacia adelante (forward pass) utilizando las imágenes del conjunto de prueba (X_test). Calcula la pérdida en función de las etiquetas reales (y_test) y la función de pérdida especificada durante la compilación del modelo (sparse_categorical_crossentropy en este caso). Además de la pérdida, también calcula la precisión (test_acc) en el conjunto de prueba.

accuracy_scores; es una lista que se ha inicializado previamente y se utiliza para almacenar las puntuaciones de precisión en cada iteración de la validación cruzada.

test_acc: Representa la precisión del modelo en el conjunto de prueba después de la evaluación.

Esta línea de código acumula las puntuaciones de precisión en la lista accuracy_scores, lo que te permitirá calcular la precisión promedio al finalizar todos los pliegues de la validación cruzada. En general, evaluar el modelo en un conjunto de prueba independiente proporciona una estimación de su rendimiento en datos no vistos.

Y terminamos con el loop for con las siguientes líneas de código mostradas en la ilustración 42.

```
# Realizar predicciones de probabilidad y etiquetas en el conjunto de prueba
y_prob = model.predict(X_test)
y_pred = np.argmax(y_prob, axis=1)
all_y_true.extend(y_test)
all_y_prob.extend(y_prob[:, 1])
all_y_pred.extend(y_pred)
```

Ilustración 42. Código de predicción de probabilidad y etiquetas en el conjunto de pruebas

Este código se utiliza para realizar predicciones en el conjunto de prueba después de que el modelo ha sido evaluado, de la siguiente manera:

model: Se refiere al modelo que has creado y entrenado previamente.

X_test: Representa las imágenes del conjunto de prueba para las cuales se realizarán predicciones.

y_prob: Almacena las probabilidades predichas por el modelo para cada clase en cada muestra del conjunto de prueba.

y_pred: Almacena las etiquetas predichas por el modelo para cada muestra del conjunto de prueba.

np.argmax(y_prob, axis=1): Utiliza la función argmax de NumPy para obtener la posición del valor máximo a lo largo del eje 1 (columnas) en las probabilidades predichas (y_prob). Esto es equivalente a asignar la clase con la probabilidad más alta como la predicción para cada muestra.

all_y_true: Es una lista que se ha inicializado previamente y se utiliza para almacenar las etiquetas verdaderas (ground truth) de todas las muestras en el conjunto de prueba.

y_test: Representa las etiquetas verdaderas correspondientes al conjunto de prueba.

all_y_prob: Es una lista que se ha inicializado previamente y se utiliza para almacenar las probabilidades predichas de la clase positiva para todas las muestras en el conjunto de prueba.

y_prob[:, 1]: Extrae las probabilidades asociadas a la clase positiva (índice 1) de las probabilidades predichas.

all_y_pred: Es una lista que se ha inicializado previamente y se utiliza para almacenar las etiquetas predichas para todas las muestras en el conjunto de prueba.

y_pred: Representa las etiquetas predichas por el modelo para el conjunto de prueba.

Estas líneas de código permiten almacenar y utilizar las predicciones del modelo en el conjunto de prueba para su posterior análisis y evaluación del rendimiento del modelo.

En concreto el loop for completo se puede apreciar en la ilustración 43.

```
for train_index, test_index in kf.split(imagenes):
    X_train, X_test = imagenes[train_index], imagenes[test_index]
    y_train, y_test = etiquetas[train_index], etiquetas[test_index]

    # Cargar el modelo ResNet50 preentrenado
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(alto_deseado, ancho_deseado, 3))

    # Agregar capas personalizadas para la clasificación
    x = base_model.output
    x = keras.layers.GlobalAveragePooling2D()(x)
    x = keras.layers.Dense(25, activation='relu')(x)
    predictions = keras.layers.Dense(2, activation='softmax')(x)

    # Crear el modelo completo
    model = keras.models.Model(inputs=base_model.input, outputs=predictions)

    # Congelar las capas del modelo base (ResNet50)
    for layer in base_model.layers:
        layer.trainable = False

    # Compilar el modelo
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Entrenar el modelo
    history = model.fit(X_train, y_train, epochs=10)

    # Evaluar el modelo en el conjunto de prueba
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
    accuracy_scores.append(test_acc)

    # Realizar predicciones de probabilidad y etiquetas en el conjunto de prueba
    y_prob = model.predict(X_test)
    y_pred = np.argmax(y_prob, axis=1)
    all_y_true.extend(y_test)
    all_y_prob.extend(y_prob[:, 1])
    all_y_pred.extend(y_pred)
```

Ilustración 43 loop for completo.

4.1.8 Resultados y gráficos:

En seguida de terminado el loop for empezamos otras líneas de codigo que nos ayudan a ver los resultados obtenido de nuestro modelo obteniendo resultados como la precisión promedio, exactitud promedio y la matriz de confusión promedio de los 3 folds hecho durante las 10 iteraciones.

Comenzando por la precisión promedio, exactitud y matriz de confusión promedio de los 3 folds ya que estas métricas son útiles para evaluar el rendimiento del modelo en datos que no ha visto durante el entrenamiento. La exactitud mide la capacidad general del modelo para clasificar correctamente las muestras, mientras que la precisión se enfoca en cuán bien el modelo realiza las clasificaciones positivas (en este caso, la clase 1). Ambas métricas son importantes para comprender cómo se comporta el modelo en diferentes situaciones y pueden ayudar a identificar posibles mejoras o ajustes en el modelo, como se muestra en la ilustración siguiente donde:

```
# Calcula el promedio de las puntuaciones de precisión de todos los folds
average_accuracy = np.mean(accuracy_scores)

# Calcula la exactitud promedio
average_roc_auc = accuracy_score(all_y_true, np.round(all_y_prob))

# Calcula la matriz de confusión promedio
cm = confusion_matrix(all_y_true, all_y_pred)

# Imprime el promedio de precisión y la exactitud promedio
print(f'Promedio de precisión a través de {num_folds} folds: {average_accuracy}')
print(f'Exactitud promedio a través de {num_folds} folds: {average_roc_auc}')
```

Ilustración 44. código de los cálculos de la precisión, exactitud y matriz de confusión promedio de los folds.

accuracy_scores: Es una lista que se ha inicializado previamente y que contiene las puntuaciones de precisión obtenidas en cada pliegue de la validación cruzada.

np.mean(accuracy_scores): Utiliza la función mean de NumPy para calcular el promedio de las puntuaciones de precisión almacenadas en la lista accuracy_scores.

average_accuracy: Almacena el valor del promedio de precisión.

all_y_true: Es una lista que contiene las etiquetas verdaderas de todas las muestras en el conjunto de prueba.

all_y_prob: Es una lista que contiene las probabilidades predichas de la clase positiva para todas las muestras en el conjunto de prueba.

np.round(all_y_prob): Redondea las probabilidades predichas a 0 o 1 para obtener las etiquetas predichas.

accuracy_score(all_y_true, np.round(all_y_prob)): Utiliza la función accuracy_score de scikit-learn para calcular la exactitud promedio comparando las etiquetas verdaderas con las etiquetas predichas.

average_roc_auc: Almacena el valor de la exactitud promedio.

Mientras que en “cm = confusion_matrix(all_y_true, all_y_pred):”

all_y_true: Es una lista que contiene las etiquetas verdaderas de todas las muestras en el conjunto de prueba.

all_y_pred: Es una lista que contiene las etiquetas predichas por el modelo para todas las muestras en el conjunto de prueba.

confusion_matrix(all_y_true, all_y_pred): Utiliza la función confusion_matrix de scikit-learn para calcular la matriz de confusión que muestra la cantidad de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

cm: Almacena la matriz de confusión.

print(f'Promedio de precisión a través de {num_folds} folds: {average_accuracy}'): Imprime en la consola el promedio de precisión calculado a través de todos los pliegues de la validación cruzada.

print(f'Exactitud promedio a través de {num_folds} folds: {average_roc_auc}'): Imprime en la consola la exactitud promedio calculada a través de todos los pliegues de la validación cruzada.

Estas líneas de código calculan y presentan métricas resumidas del rendimiento del modelo, como el promedio de precisión, la exactitud promedio y la matriz de confusión promedio, basándose en las evaluaciones realizadas en cada pliegue de la validación cruzada.

En seguida de esto se calcula y grafica la curva ROC promedio con las líneas de código mostradas en la siguiente ilustración 45.

```
# Calcula la curva ROC promedio
fpr, tpr, _ = roc_curve(all_y_true, all_y_prob)
roc_auc = auc(fpr, tpr)

# Grafica la curva ROC promedio
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='Curva ROC (área = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC Promedio')
plt.legend(loc="lower right")
plt.show()

# Imprime la matriz de confusión promedio
print('Matriz de Confusión Promedio:')
print(cm)
```

Ilustración 45. código del cálculo de la curva ROC, así como su grafica.

Donde de “fpr, tpr, _ = roc_curve(all_y_true, all_y_prob):” tenemos lo siguiente:

all_y_true: Lista que contiene las etiquetas verdaderas de todas las muestras en el conjunto de prueba.

all_y_prob: Lista que contiene las probabilidades predichas de la clase positiva para todas las muestras en el conjunto de prueba.

roc_curve(all_y_true, all_y_prob): Utiliza la función roc_curve de scikit-learn para calcular las tasas de falsos positivos (fpr) y verdaderos positivos (tpr) en diferentes

umbrales de clasificación. El tercer valor retornado (_) corresponde a umbrales y se descarta en este caso.

En “roc_auc = auc(fpr, tpr):” ocurre lo siguiente:

auc(fpr, tpr): Utiliza la función auc de scikit-learn para calcular el área bajo la curva ROC (ROC AUC) utilizando las tasas de falsos positivos (fpr) y verdaderos positivos (tpr) calculadas previamente.

roc_auc: Almacena el valor del área bajo la curva ROC.

Grafica la curva ROC promedio donde:

plt.figure(): Crea una nueva figura para la visualización.

plt.plot(fpr, tpr, color='darkorange', lw=2, label='Curva ROC (área = %0.2f)' % roc_auc): Plotea la curva ROC utilizando las tasas de falsos positivos (fpr) y verdaderos positivos (tpr). Se utiliza un color naranja oscuro, un grosor de línea de 2 y se incluye la etiqueta con el valor del área bajo la curva ROC.

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--'): Plotea la línea de referencia para un clasificador aleatorio en azul oscuro. Ajusta los límites del gráfico, las etiquetas de los ejes, el título y muestra la leyenda en la esquina inferior derecha.

plt.show(): Muestra la figura generada, que representa la curva ROC promedio.

print('Matriz de Confusión Promedio:'): Imprime en la consola un encabezado indicando que se mostrará la matriz de confusión promedio.

En “print(cm):”

cm: Es la matriz de confusión calculada previamente e Imprime en la consola la matriz de confusión promedio.

De esta manera, esta sección del código se encarga de calcular, visualizar y mostrar la curva ROC promedio junto con su área bajo la curva (ROC AUC), así como la impresión de la matriz de confusión promedio en la consola.

Terminamos con la Ilustración 46, la cual muestra las líneas de código que grafican la matriz de confusión promedio de la siguiente manera:

```
# Grafica la matriz de confusión promedio
plt.figure(figsize=(7, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', square=True)
plt.xlabel('Predicho')
plt.ylabel('Verdadero')
plt.title('Matriz de Confusión Promedio')
plt.show()

# Ver la función de pérdida
plt.xlabel("# Época")
plt.ylabel("Magnitud de pérdida")
plt.plot(history.history["loss"])
```

Ilustración 46. Código de la gráfica de la matriz de confusión promedio y función de perdida.

plt.figure(figsize=(7, 7)): Crea una nueva figura para la visualización con un tamaño de 7 por 7 pulgadas.

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', square=True): Utiliza Seaborn para crear un mapa de calor de la matriz de confusión (cm). Los parámetros específicos incluyen:

annot=True: Muestra los valores reales en cada celda del mapa de calor.

fmt='d': Formatea los valores como enteros.

cmap='Blues': Define la paleta de colores como tonos de azul.

square=True: Garantiza que las celdas del mapa de calor sean cuadradas.

plt.xlabel('Predicho'): Etiqueta el eje x con "Predicho".

plt.ylabel('Verdadero'): Etiqueta el eje y con "Verdadero".

plt.title('Matriz de Confusión Promedio'): Asigna un título al gráfico.

plt.show(): Muestra la figura generada, que representa la matriz de confusión promedio.

Y para Ver la función de pérdida:

plt.xlabel("# Época"): Etiqueta el eje x con el número de épocas.

plt.ylabel("Magnitud de pérdida"): Etiqueta el eje y con la magnitud de la pérdida.

plt.plot(history.history["loss"]): Plotea la función de pérdida a lo largo de las épocas utilizando el historial de pérdida (history.history["loss"]) del entrenamiento.

De esta manera estas líneas de código generan visualizaciones para la matriz de confusión promedio y la función de pérdida a lo largo del entrenamiento del modelo. La matriz de confusión se muestra como un mapa de calor, proporcionando una representación visual de las predicciones del modelo, mientras que la función de pérdida se plotea para visualizar cómo cambia durante el entrenamiento.

4.2 Base de datos

Para conseguir la base de datos con las que se puso a prueba la red neuronal convolucional, se realizó la toma de diversas fotografías e imágenes, algunas tomas de internet y otras tomadas directamente de estantes en centros comerciales, obteniendo las siguientes bases de datos diferentes.

4.2.1 Descripción del primero conjunto de imágenes:

El primer conjunto de imágenes utilizadas fue bajado de internet, donde descargamos imágenes en dimensiones de entre 200 x 200 hasta 1000 x 1000 pixeles, de vinos cuales quiera y tenedores, cuchillos y cucharas. Las imágenes cuentan con 634 imágenes de vinos y con 634 imágenes de tenedores cuchillos y cucharas tomadas de internet, como las que podemos ver en las ilustraciones 47 y 48.

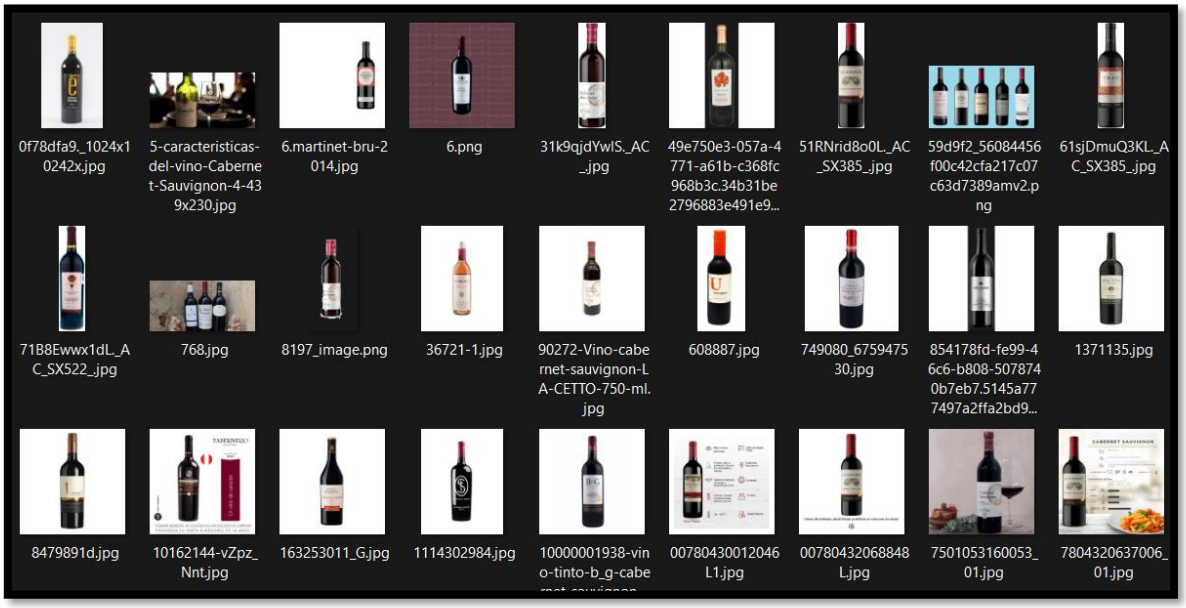

Ilustración 47 vinos aleatorios tomados de internet

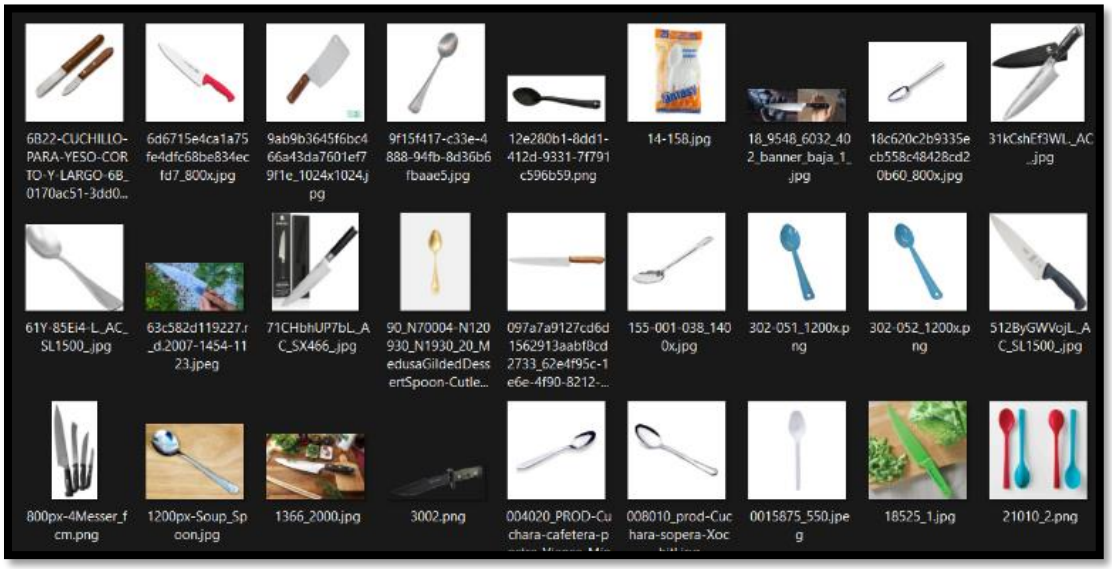

Ilustración 48. Tenedores y cucharas tomadas de internet.

4.2.2 Descripción del segundo conjunto de imágenes:

Las imágenes del segundo conjunto cuentan con 487 imágenes de vinos oscuros y con otras 487 imágenes son de vinos claros, todas estas imágenes son tomadas directamente de internet. Estas imágenes parten de unas dimensiones de entre 200 x 200 pixeles hasta 1000 x 1000 aproximadamente, Podemos ver un ejemplo de este conjunto de imágenes en las ilustraciones 49 y 50.

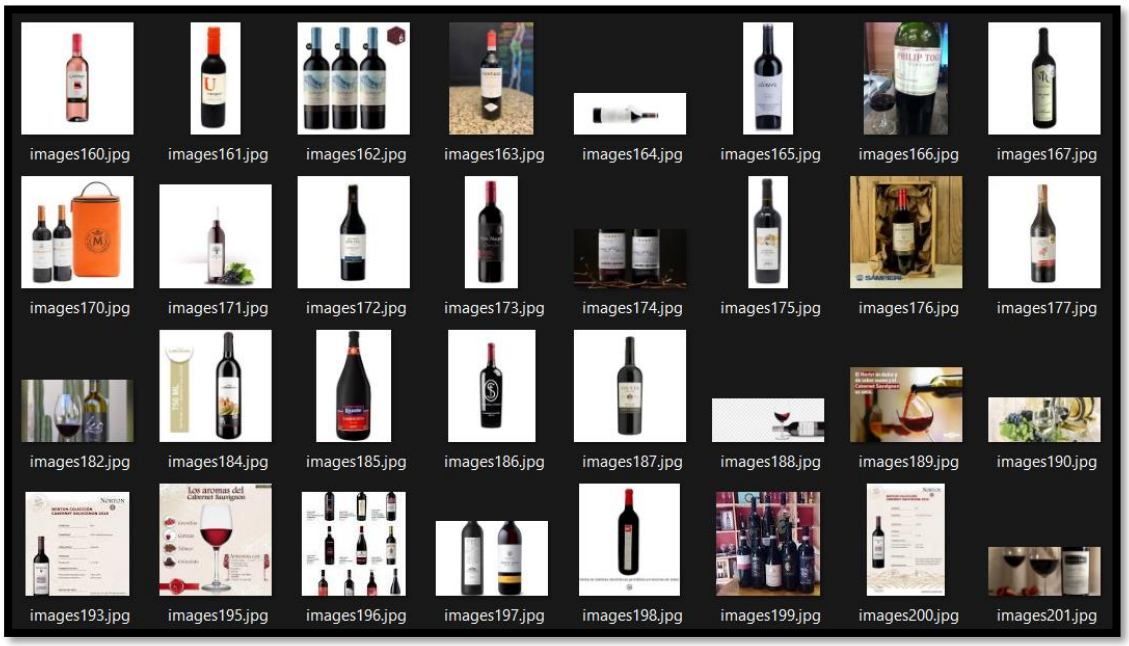

Ilustración 49. Imágenes de vinos oscuros

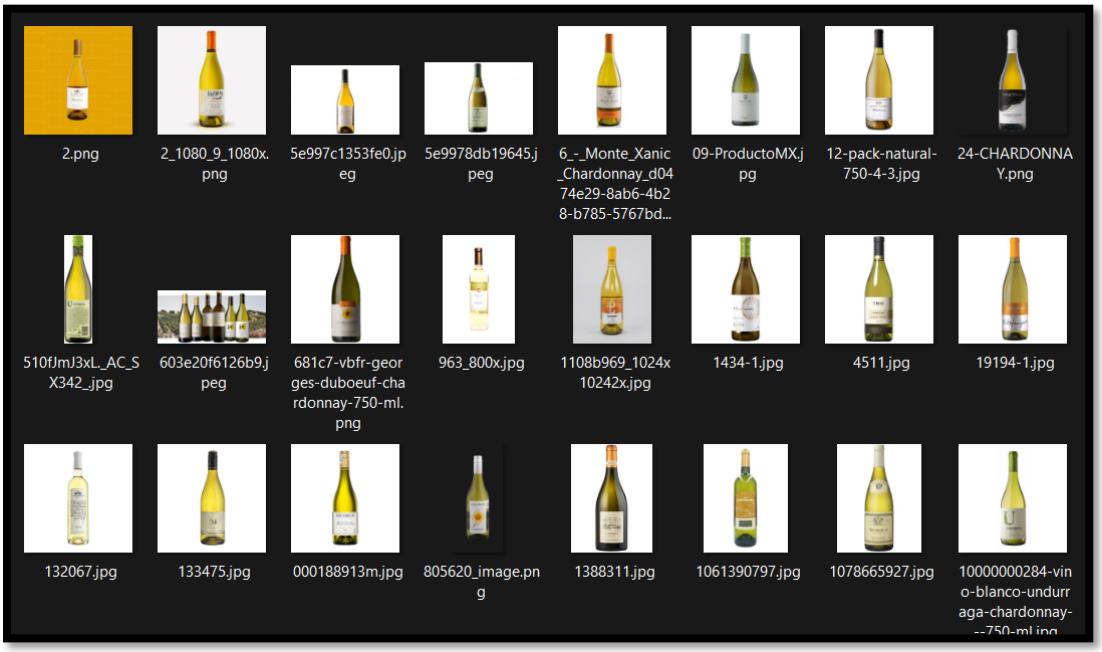

Ilustración 50. Imágenes de vinos claros

4.2.3 Descripción del tercer conjunto de imágenes

Las imágenes del tercer conjunto cuentan con 120 imágenes de vinos tomados directamente de los estantes de centros comerciales, y de 120 objetos tomados de manera aleatoria en el mismo centro comercial u otros lugares, con una dimensión que va desde 760 x 1600 hasta 3000 x 3000 pixeles. Podemos ver un ejemplo de este conjunto de imágenes en las ilustraciones 51 y 52.

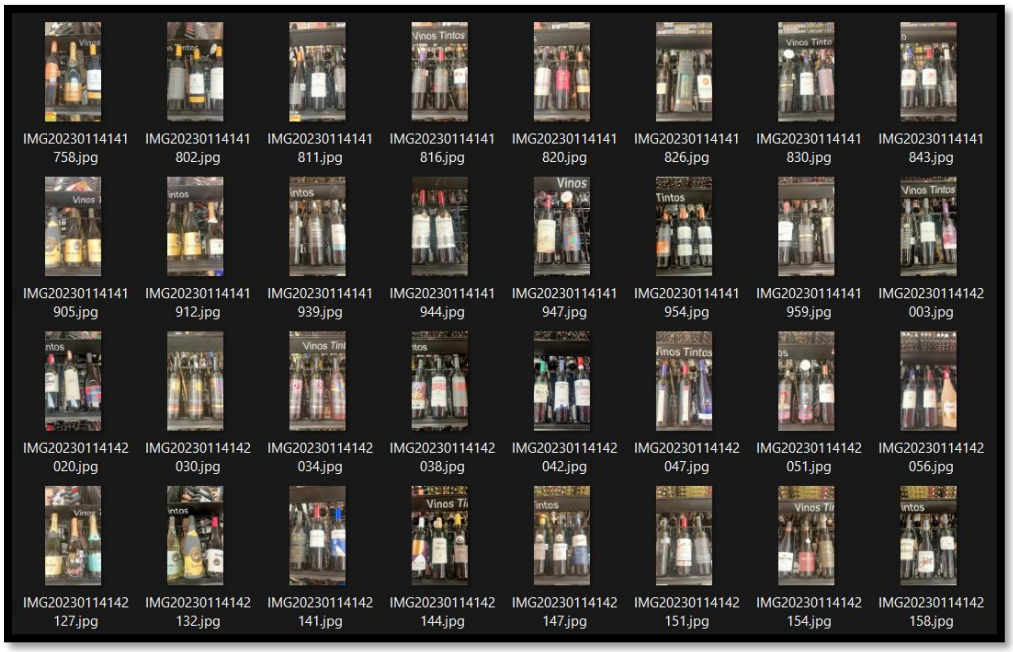

Ilustración 51. Imágenes de vinos tomados directamente de centros comerciales

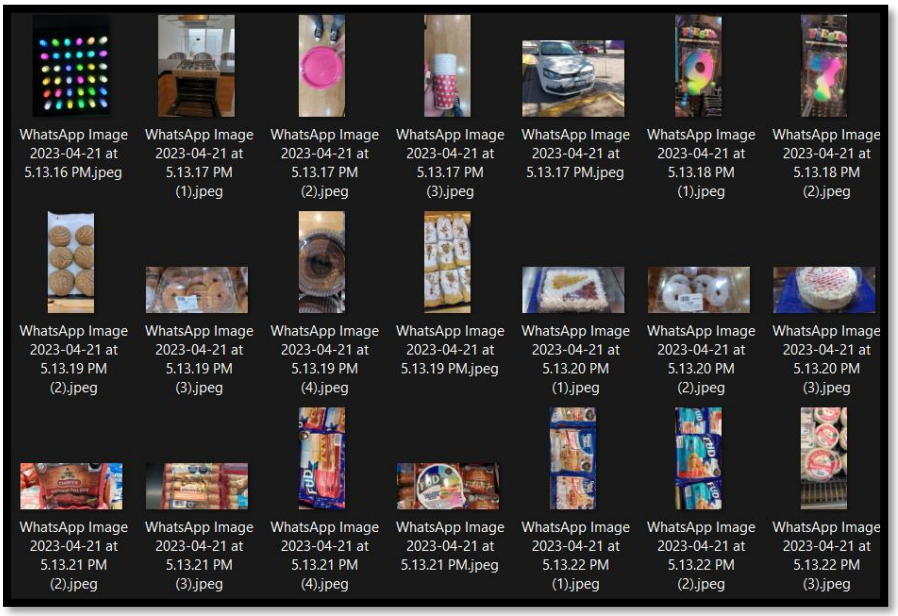

Ilustración 52. Imágenes de objetos aleatorios tomados de diversos lugares.

4.2.4 Descripción del cuarto conjunto de imágenes:

Las imágenes del tercer conjunto fueron las previamente tomadas de centros comerciales de vinos, pero las separamos en vinos claro y vinos oscuros, siendo un total de 120 imágenes de vinos claros y 120 de vinos oscuros con las dimensiones de 760 x 1600 hasta 3000 x 3000 pixeles. Podemos ver un ejemplo de este conjunto de imágenes en las ilustraciones 53 y 54.

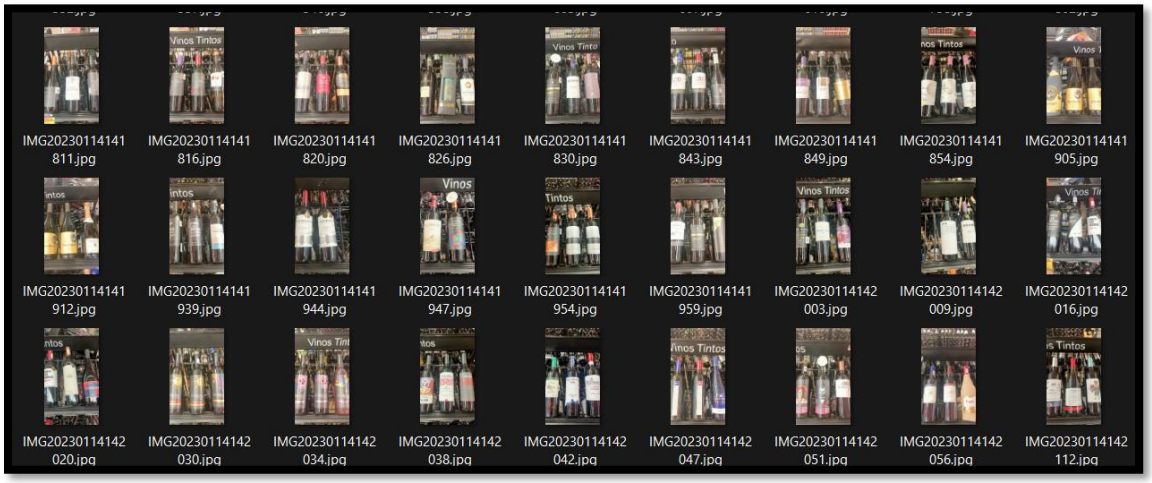

Ilustración 53. Imágenes de vinos oscuros tomadas de centros comerciales.

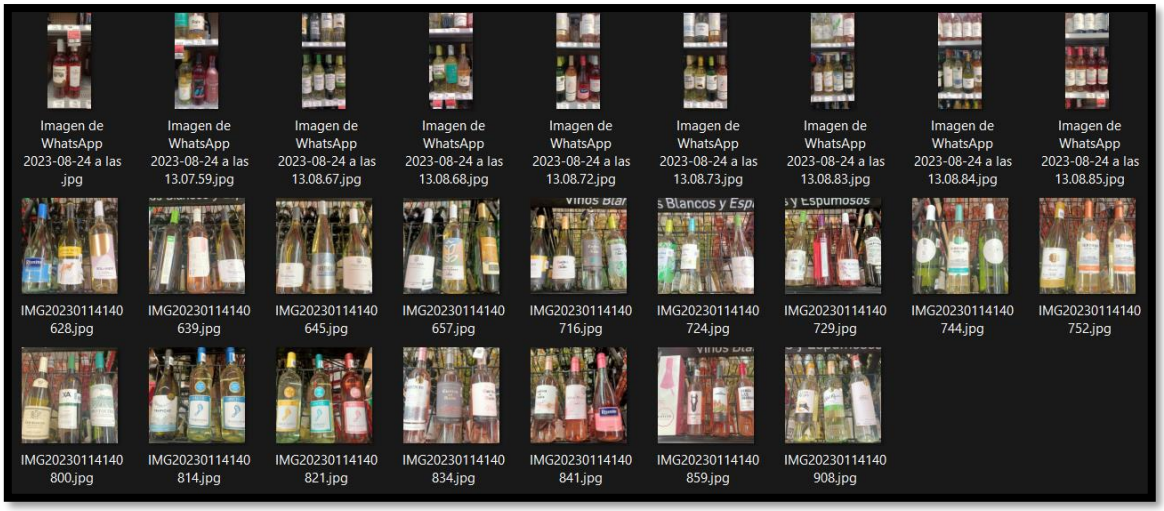

Ilustración 54. Imágenes de vinos claros tomados de centros comerciales.

4.2.5 Descripción del quinto conjunto de imágenes:

Para el quinto conjunto se tomaron imágenes directamente del estante de centros comerciales para dividirlas en 2 clases. La primera clase está conformada por 84 imágenes tomas de estantes de vinos donde si se encuentra un vino específico en todas las fotos. La segunda clase tiene 84 imágenes de vinos que no incluyen un vino en específico en ninguna de las tomas. Podemos ver un ejemplo de este conjunto de imágenes en las ilustraciones 55 y 56.

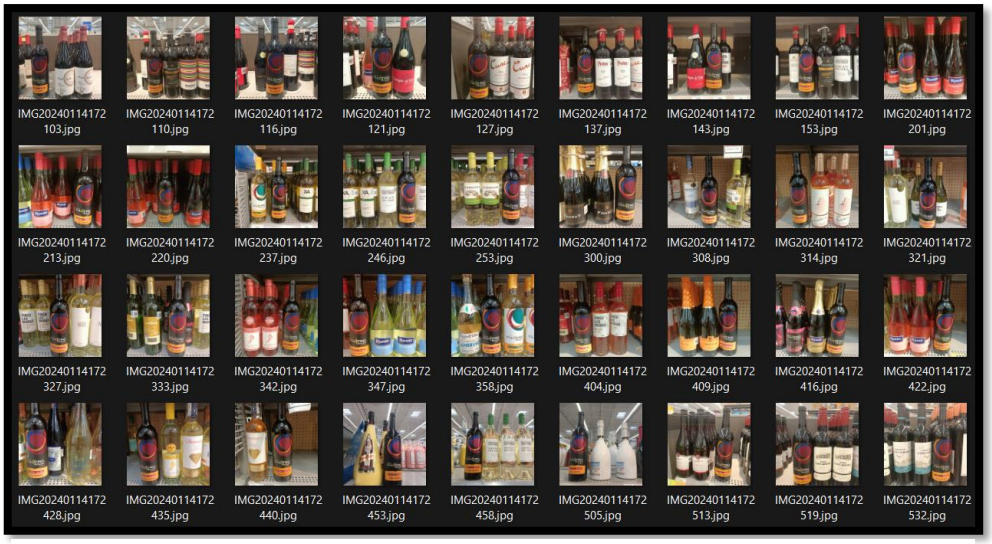

Ilustración 55, imágenes de la clase 1 donde un vino se repite en todas las fotos.

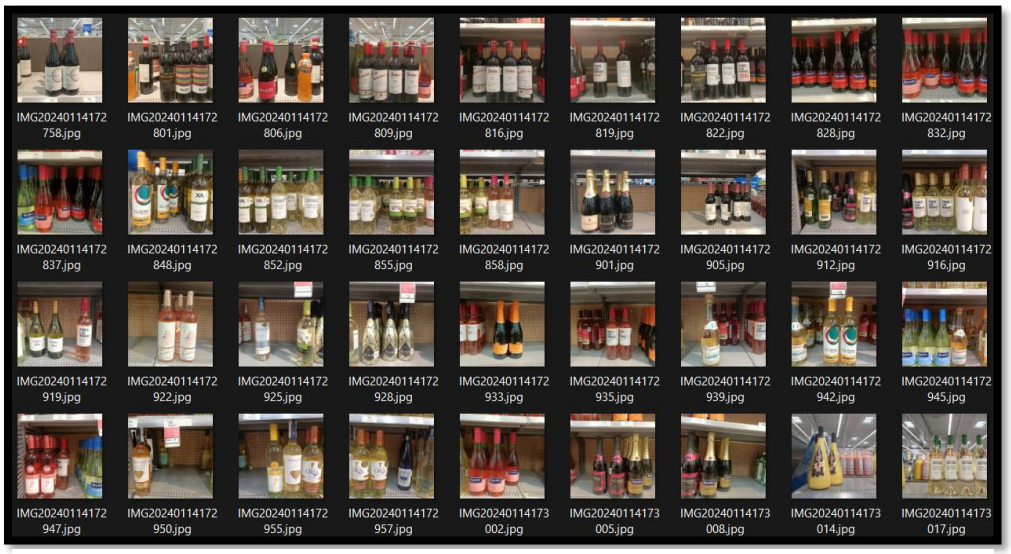

Ilustración 56, imágenes de la clase 2.

4.2.6 Descripción del sexto conjunto de imágenes:

Para el sexto y último conjunto, tomamos las imágenes del quinto experimento, pero a la clase 1 le agregamos otro vino especifico que no se repite en la misma clase, llegando a un total de 134 imágenes en la clase 1, también agregamos más imágenes donde no se encontraría ninguno de los vinos específicos de la clase 1, llegando a 132 imágenes en la clase 2. Podemos ver un ejemplo de este conjunto de imágenes en las ilustraciones 57 y 58

Ilustración 57, Imágenes de la clase 2, experimento 6.

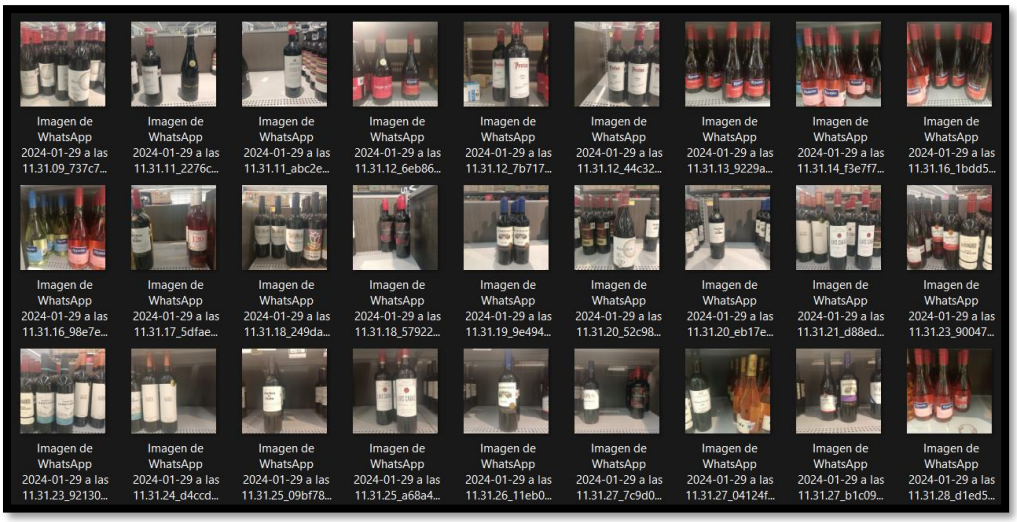

Ilustración 58, imágenes de la clase 1, experimento 6.

4.3 Modelo:

A continuación, podemos ver el modelo completo desde la entrada de los archivos que corresponden a las imágenes de entrada, hasta la salida del modelo.

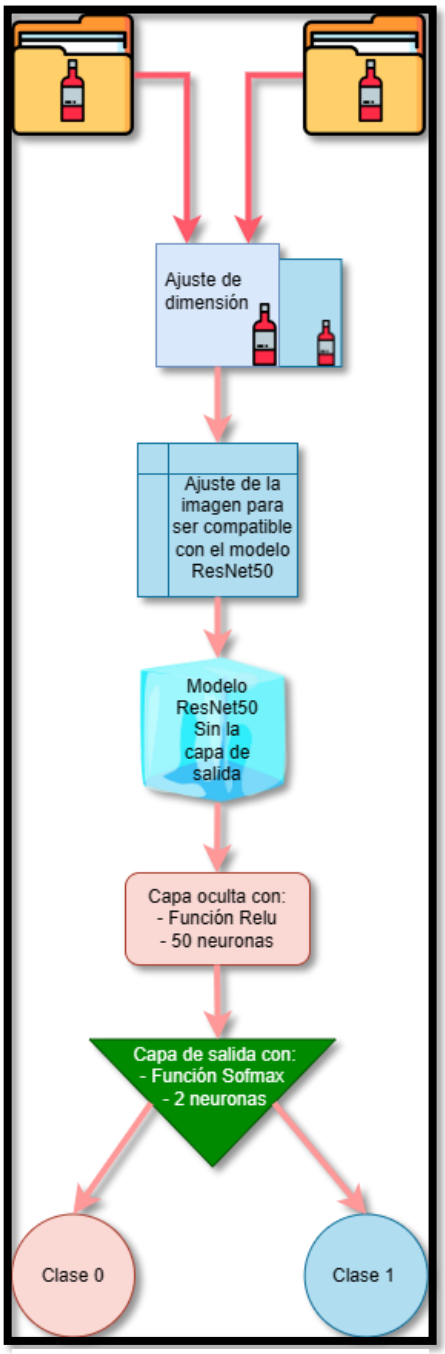

Ilustración 59. Modelo

5. Pruebas y Resultados

En este capítulo se reportan los resultados obtenidos tras el desarrollo de los experimentos, haciendo uso de las bases de datos y el código descrito en los capítulos pasados.

5.1 Descripción del primer experimento

El primer experimento tomo el primer base de datos, es decir las imágenes la que cuenta con 634 imágenes de vinos y con 634 y objetos aleatorios tales como tenedores, cuchillos y cucharas. Todas estas bajadas de internet pasando por el siguiente proceso que a grandes rasgos lo podemos describir de la siguiente manera:

1- Entra las imágenes en sus carpetas sin etiqueta
2- Las imágenes se redimensionan a 224 x 224 pixeles
3- El Código las etiqueta asignándoles el valor de 1 para la primera carpeta de imágenes y 0 para la segunda carpeta de imágenes
4- Se carga el modelo preentrenado ResNet50
5- Se congelas los pesos del modelo
6- Se agregan únicamente una capa oculta de 50 neuronas que contiene la función de activación ReLu
7- Se agrega la capa de salida de 2 neuronas que contiene la función de activación SofMax
8- Y se corre el código durante 10 épocas aplicando 3 folds para la validación estadística
9- Obtenemos los resultados promedio y los graficamos.

5.1.1 Resultados del primer experimento

Tabla 1. Resultados del primer experimento.

Primer experimento (vinos vs otros objetos de internet)						
Numero de Folds	Dimensión de la imagen	Presición	Exactitud	Área bajo la curba ROC	Épocas	Tiempo
3	224 x 224	0.98848929	0.988495575	1	10	8min 25seg

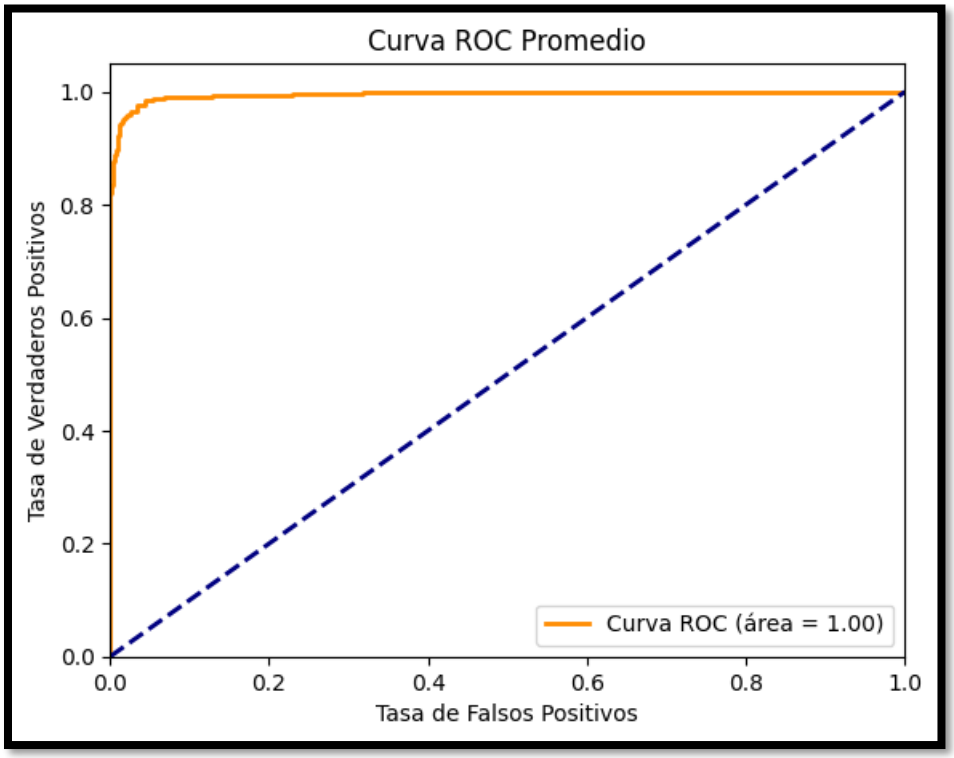

Ilustración 60. Curva ROC del primer experimento.

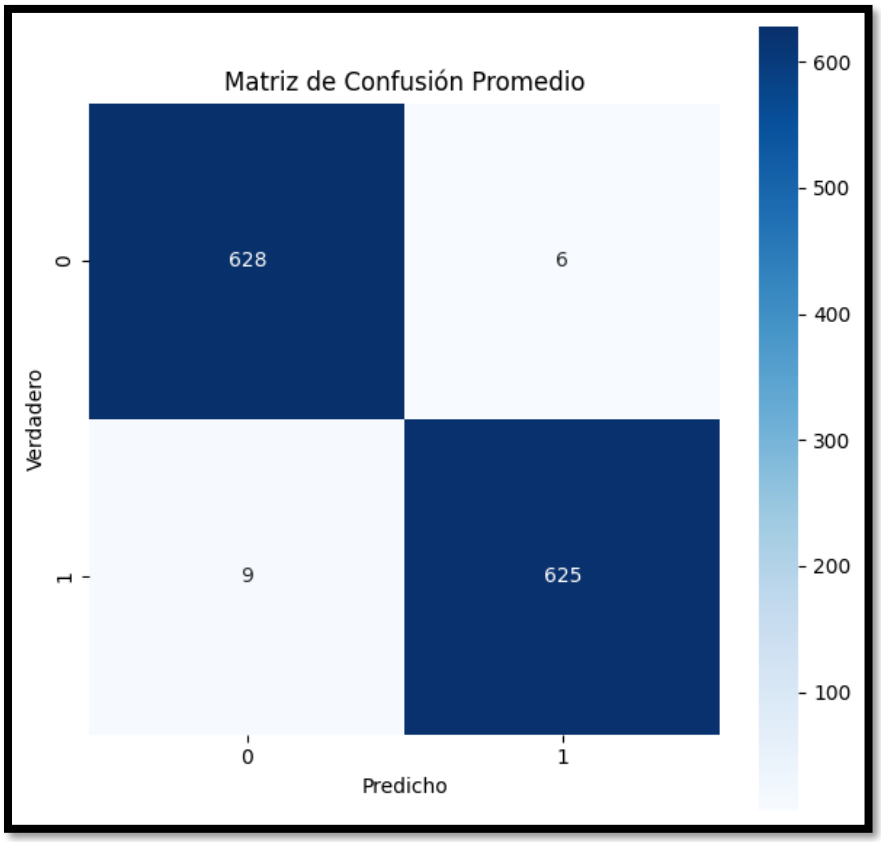

Ilustración 61. Matriz de confusión del primer experimento.

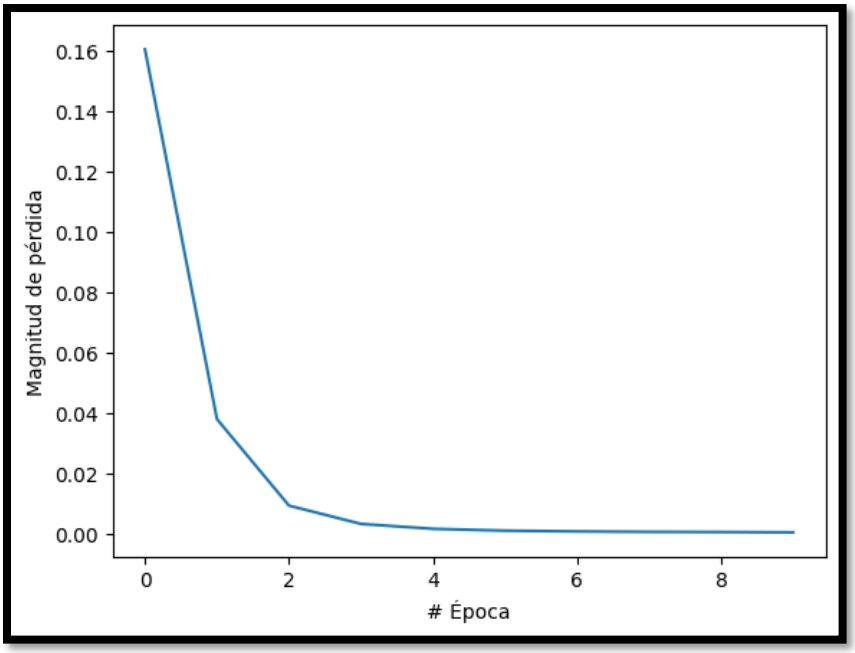

Ilustración 62 grafica de la función de pérdida del primer experimento.

5.2 Descripción del segundo experimento

El segundo experimento tomo la segunda base de datos, es decir datos de 487 imágenes de vinos oscuros con otras 487 imágenes de vinos claros. Todas estas bajadas de internet y pasaron por el mismo proceso descrito en el punto 5.1.

5.2.1 Resultados del segundo experimento

Tabla 2 Resultados del segundo experimento.

Segundo experimento (vino claro vs vino oscuro de internet)						
Numero de Folds	Dimensión de la imagen	Presición	Exactitud	Área bajo la curba ROC	Épocas	Tiempo
3	224 x 224	0.914550245	0.914547304	0.97	10	5m 27seg

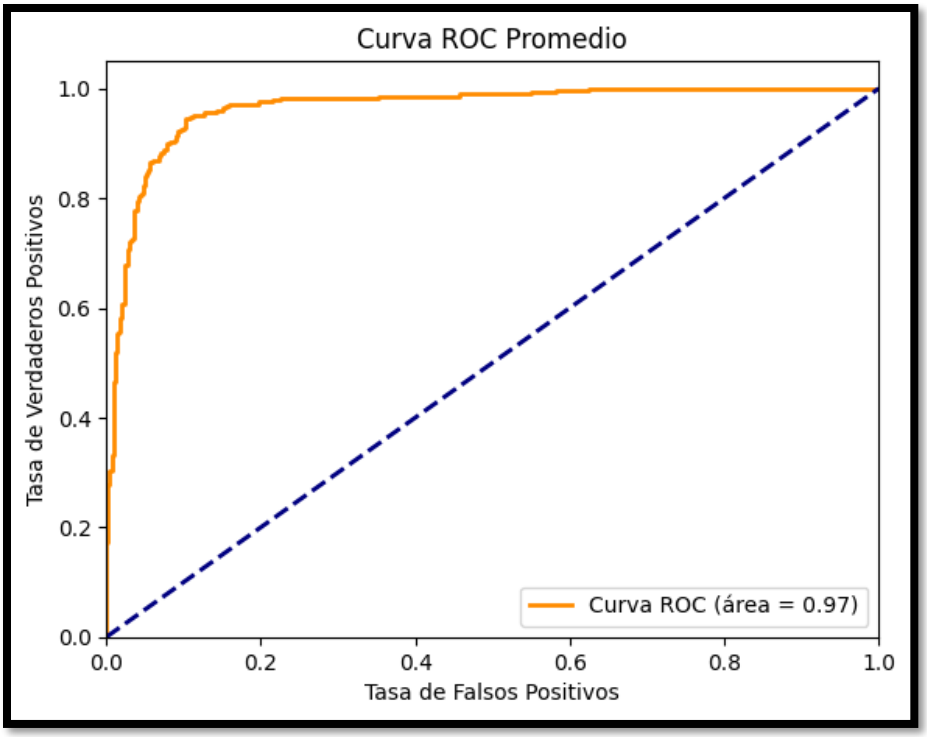

Ilustración 63. Curva ROC del segundo experimento.

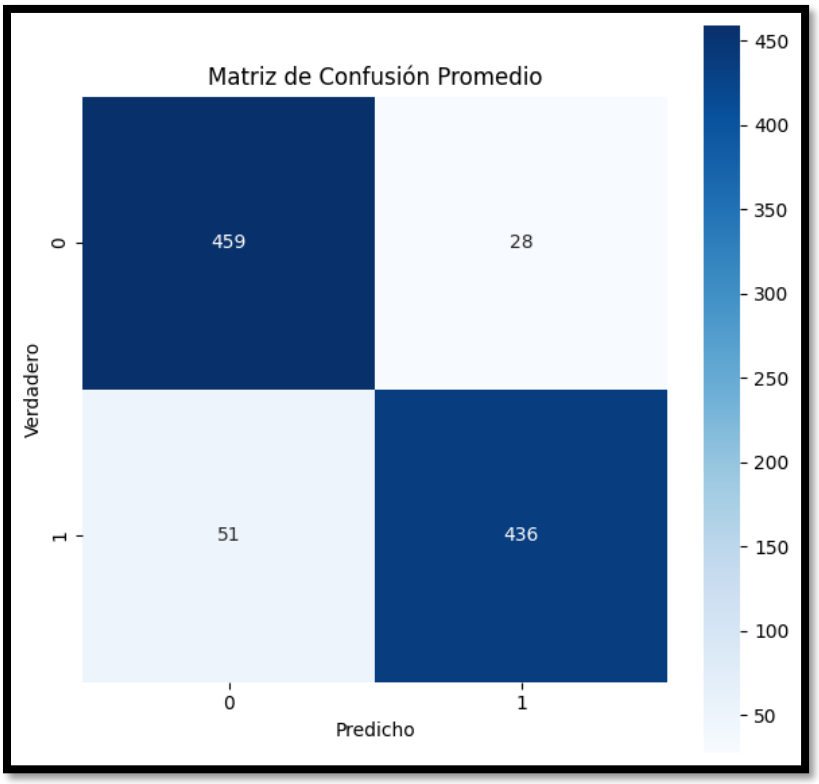

Ilustración 64.Matriz de confusión del segundo experimento.

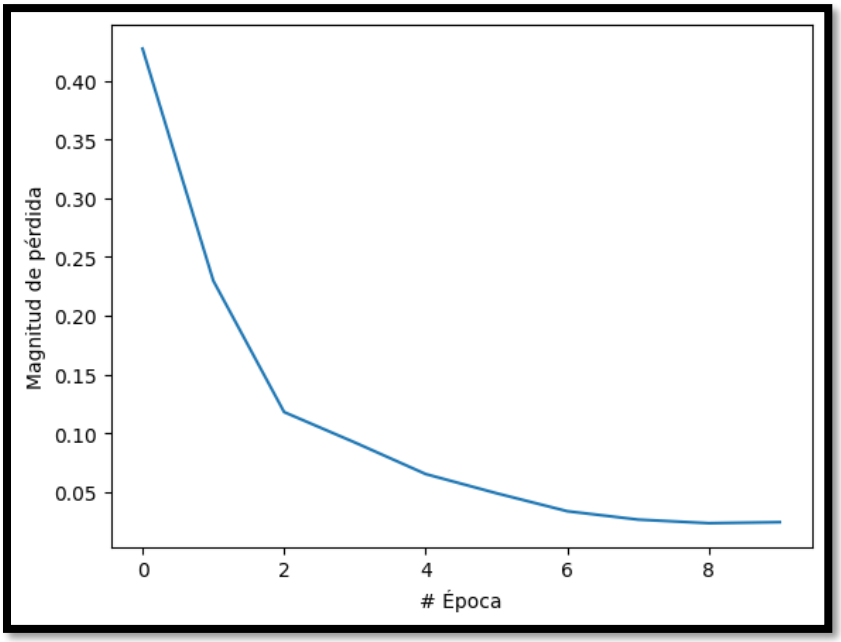

Ilustración 65. Grafica de la función de pérdida del segundo experimento.

5.3 Descripción del tercer experimento

El tercer experimento tomo la tercera base de datos, es decir son 120 imágenes de vinos tomados directamente de los estantes de centros comerciales y de 120 objetos tomados de manera aleatoria en el mismo centro comercial u otros lugares. Pasando por el mismo proceso descrito en el punto 5.1.

5.3.1 Resultados del tercer experimento

Tabla 3. Resultados del experimento 3

Tercer experimento (vinos vs otros objetos de centros comerciales)						
Numero de Folds	Dimensión de la imagen	Presición	Exactitud	Área bajo la curba ROC	Épocas	Tiempo
3	224 x 224	0.995833337	0.995833333	1	10	15m 36seg

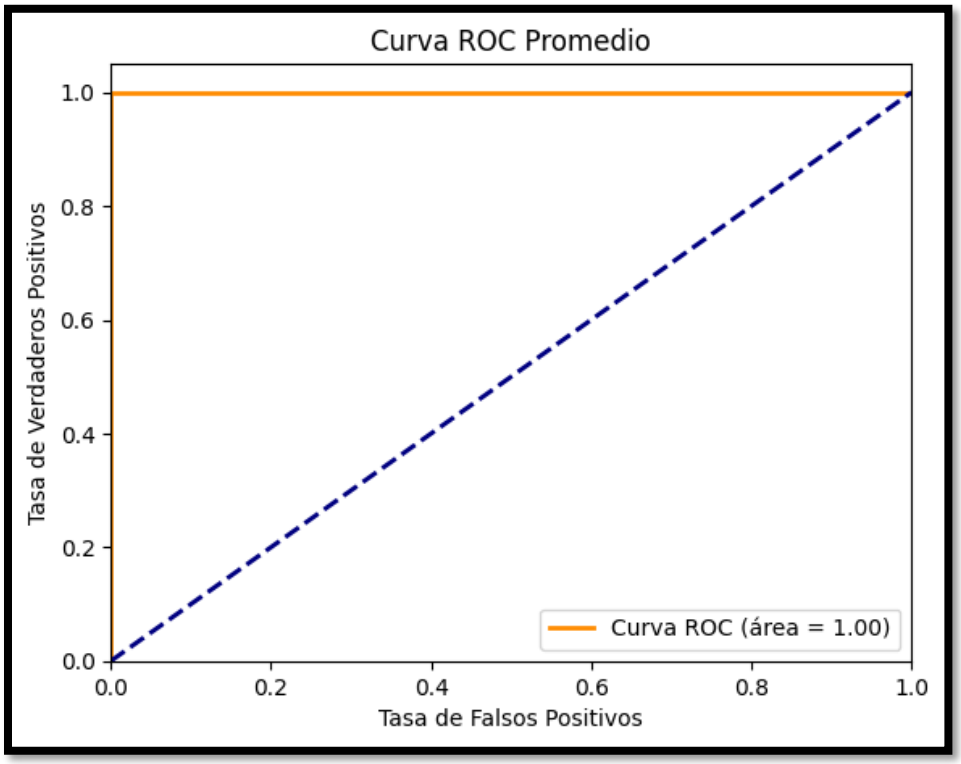

Ilustración 66. Curva ROC del tercer experimento.

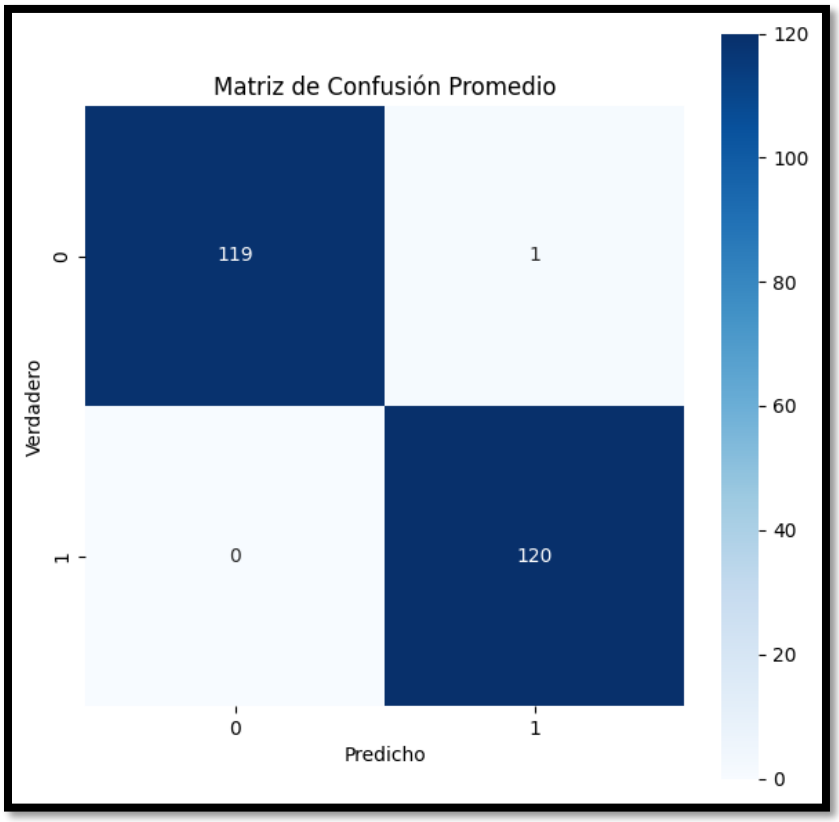

Ilustración 68. Matriz de confusión del tercer experimento.

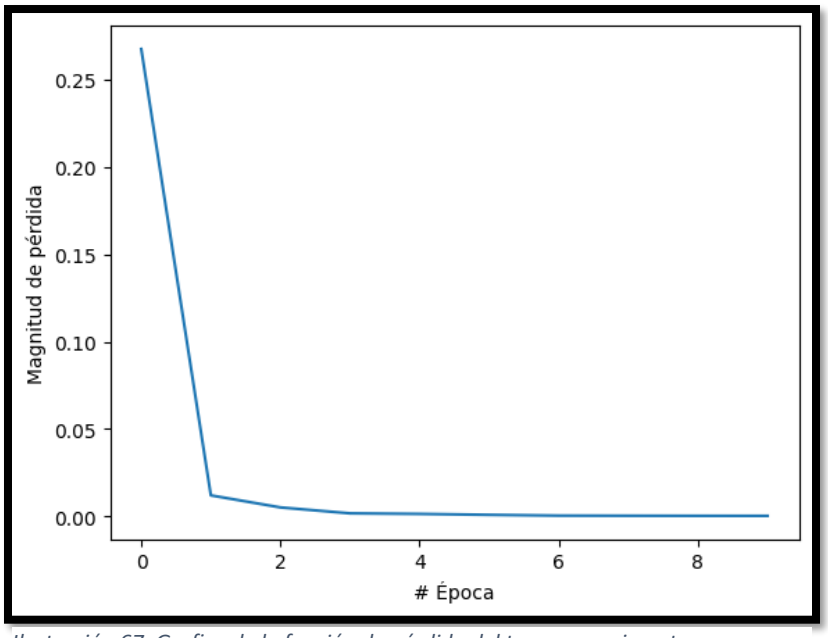

Ilustración 67. Grafica de la función de pérdida del tercer experimento.

5.4 Descripción del cuarto experimento

En el cuarto experimento se tomaron las imágenes previamente tomadas de vinos en centros comerciales, en total se tomaron solo 120 imágenes de vinos claros y 120 de vinos oscuros. Pasando por el mismo proceso descrito en el punto 5.1.

5.4.1 Resultados del cuarto experimento

Tabla 4 Resultados del cuarto experimento.

Cuarto experimento(vino claro vs vino oscuro de centros comerciales)						
Numero de Folds	Dimensión de la imagen	Presición	Exactitud	Área bajo la curba ROC	Épocas	Tiempo
3	224 x 224	0.937984506	0.937984496	0.98	10	11m 10seg

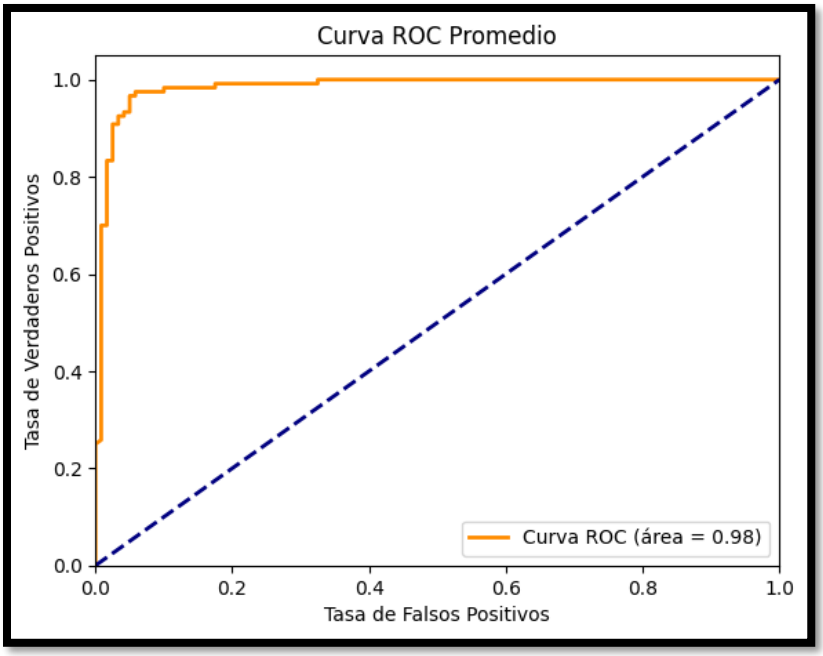

Ilustración 69. Curva ROC del cuarto experimento

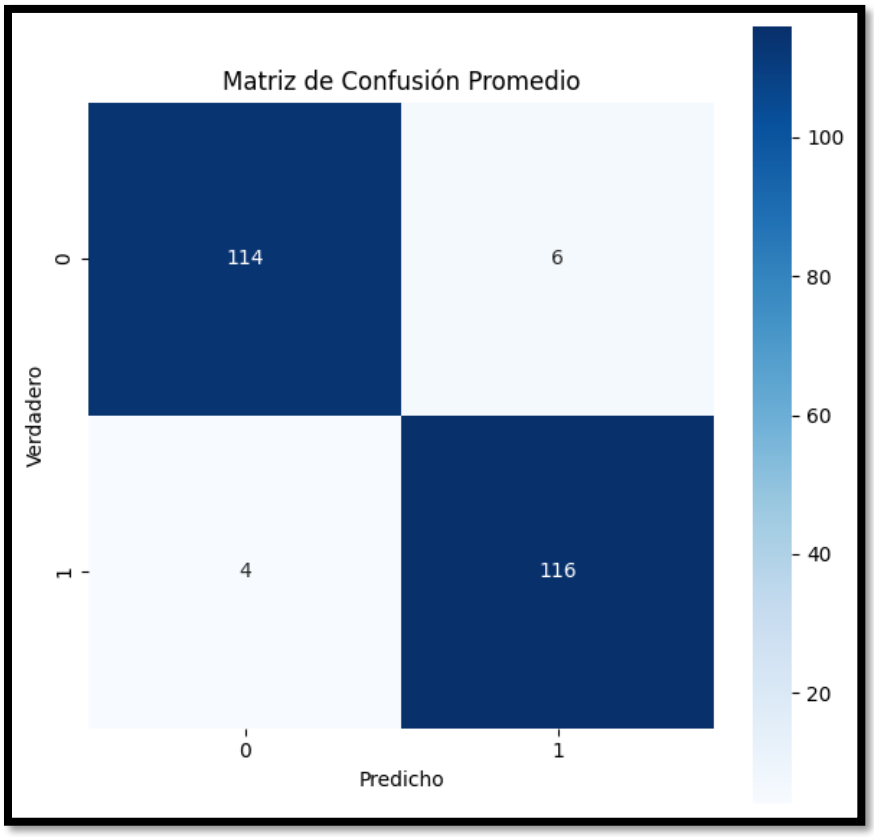

Ilustración 70. Matriz de confusión del cuarto experimento.

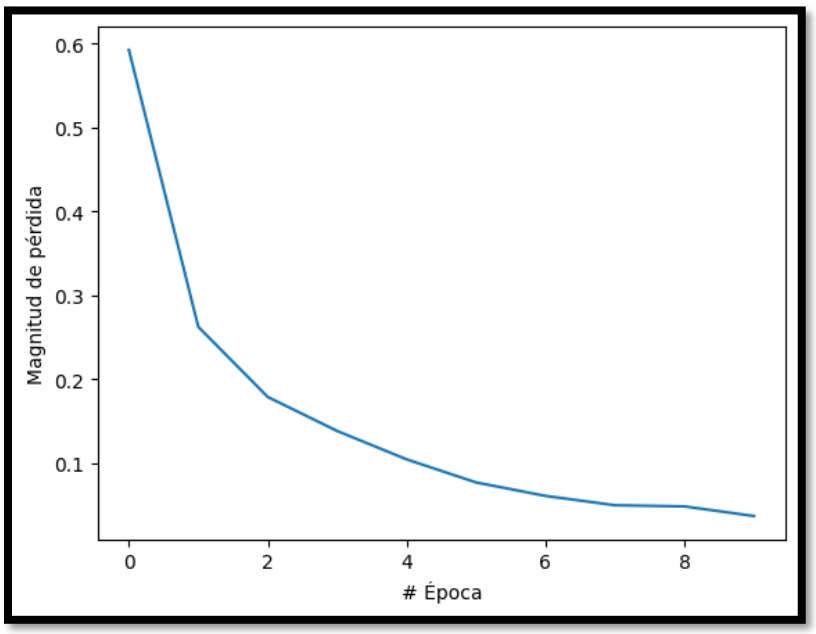

Ilustración 71. Grafica de la función de pérdida del cuarto experimento.

5.5 Descripción del quinto experimento

En el quinto experimento se tomaron imágenes de vinos en centros comerciales, en total se tomaron solo 84 imágenes de vinos, donde se colocó un vino en específico en esas 84 fotografías. Se compararon esas fotografías contra otras 84 fotos de vinos, donde no estaba en ninguna de estas el vino colocado en las anteriores imágenes, con el propósito de saber si la red era capaz de notar que la diferencia en las imágenes solo por mover un vino. Pasando por el mismo proceso descrito en el punto 5.1.

5.5.1 Resultados del quinto experimento

Tabla 5. Resultados del quinto experimento.

Quinto experimento(vino especifico mezclado entre otros vinos, de centros comerciales)						
Numero de Folds	Dimensión de la imagen	Presición	Exactitud	Área bajo la curba ROC	Épocas	Tiempo
3	224 x 224	0.845238109	0.845238095	0.93	10	4m 24seg

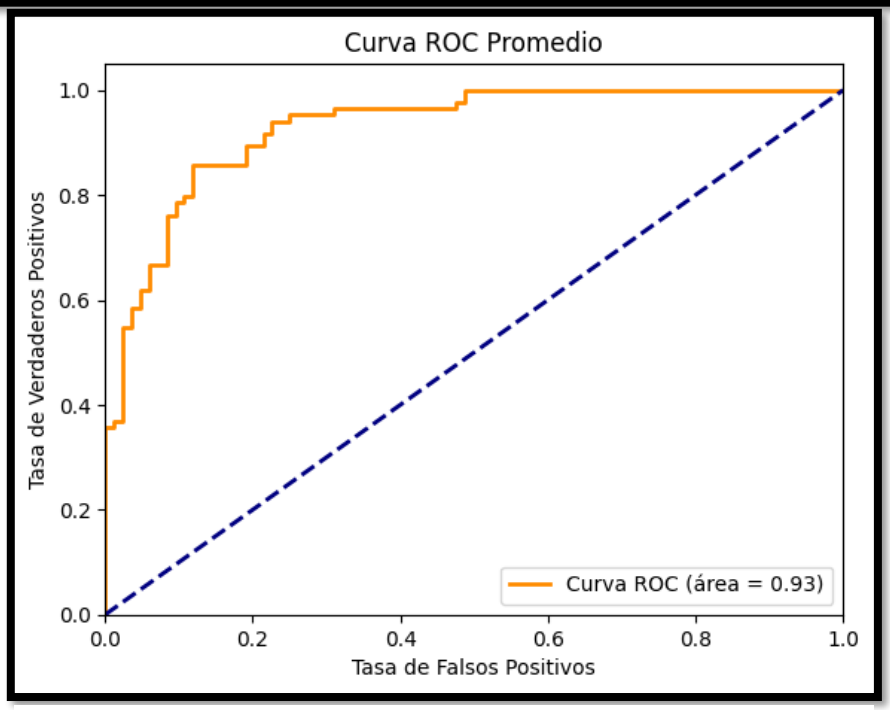

Ilustración 72. Curva ROC del quinto experimento

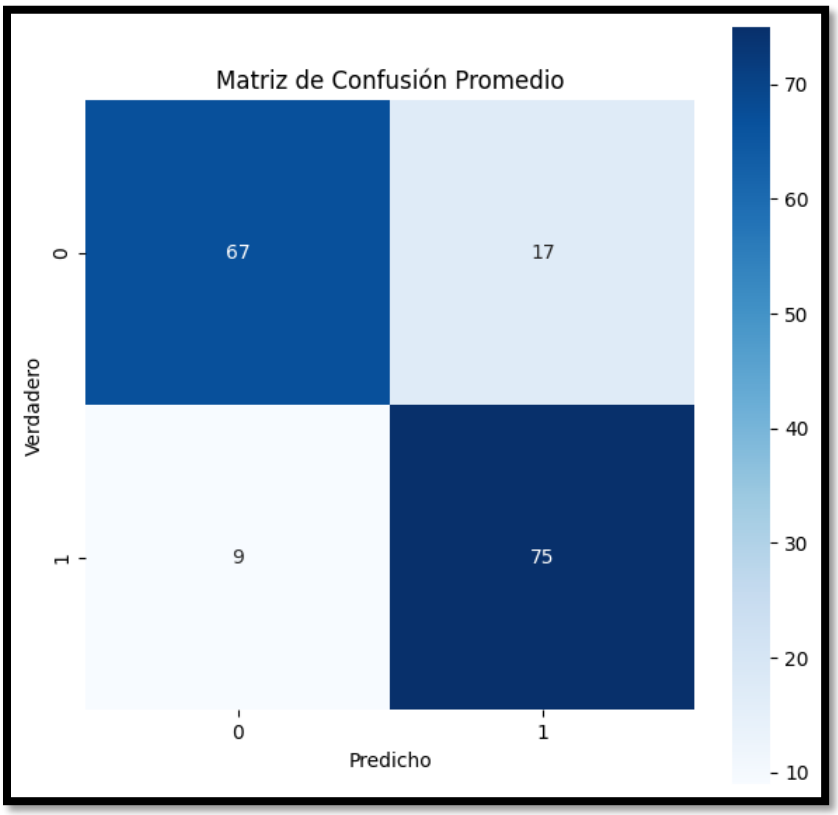

Ilustración 74. Matriz de confusión del quinto experimento.

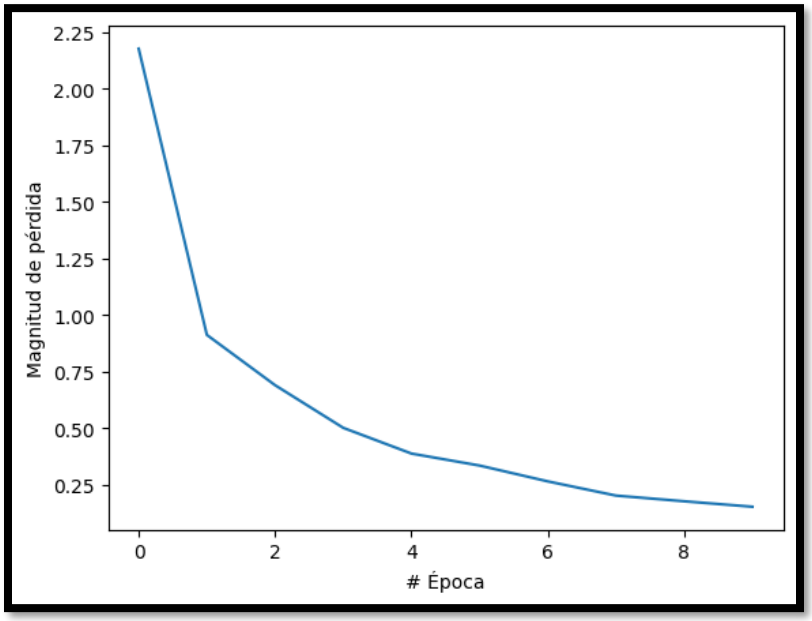

Ilustración 73. Grafica de la función de pérdida del quinto experimento.

5.6 Descripción del sexto experimento

En el sexto experimento se tomaron imágenes de vinos en centros comerciales, en total se tomaron solo 134 imágenes de vinos, donde se colocaron dos vinos en específico en esas 134 fotografías. Se compararon esas fotografías contra otras 132 fotos de vinos, donde ninguno de estos vinos específicos colocados en las anteriores imágenes, con el propósito de estresar aún más la red.

El propósito de hacer este experimento es que, con base en los resultados concluimos si la red está observando las características esenciales de un vino como para saber si se encuentra en una imagen o no, esto al notar que si agregamos más vinos a la clase que tiene vinos específicos, y a la otra clase la seguimos cargando con imágenes que no tiene ninguno de los vinos específicos de la otra clase, suponemos que la red está reconociendo las características en la botella de manera que distingue más allá del color y la forma para saber que hay una diferencia entre ambas clases.

5.6.1 Resultados del sexto experimento

Tabla 6. Resultados del sexto experimento

Sexto experimento(2 vinos especificos mezclados entre otros vinos, de centros comerciales)						
Numero de Folds	Dimensión de la imagen	Presición	Exactitud	Área bajo la curba ROC	Épocas	Tiempo
3	224 x 224	0.781877756	0.781954887	0.89	10	5m 59seg

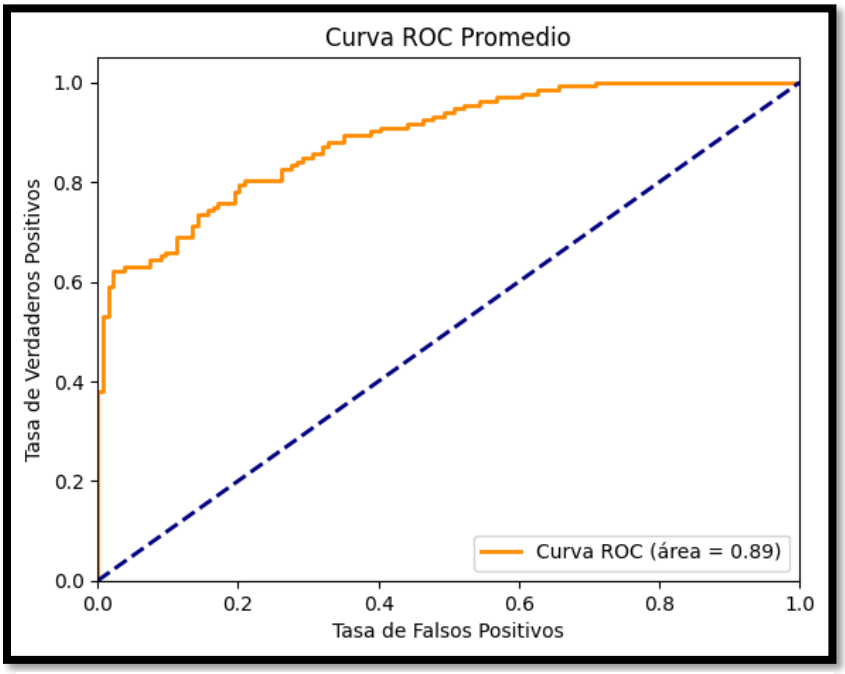

Ilustración 75. Curva ROC del sexto experimento

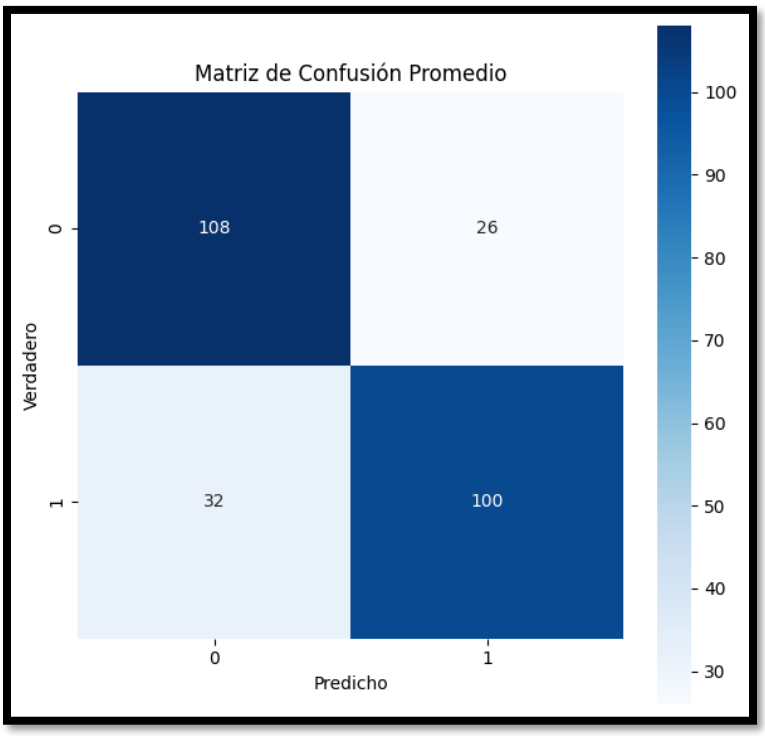

Ilustración 76. Matriz de confusión del sexto experimento.

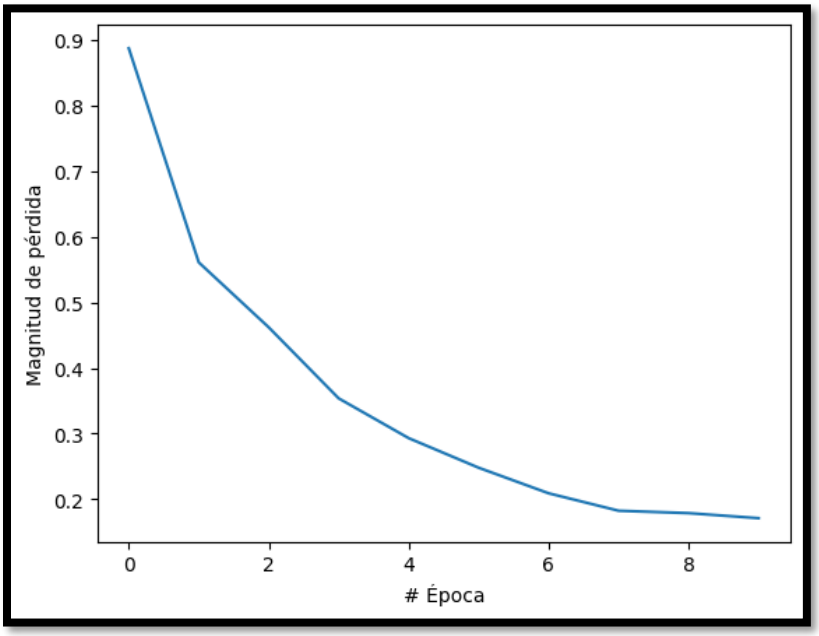

Ilustración 77. Grafica de la función de pérdida del sexto experimento.

6. Conclusiones:

Tras llevar a cabo los 6 experimentos con el modelo ResNet50, concluimos que la capacidad de clasificación del modelo es notablemente alta al enfrentar clases muy distintas entre sí, como lo pudimos notar en los experimentos 1 y 3.

La captura de fotografías directamente desde los estantes requiere más tiempo de procesamiento por parte del modelo, aunque sigue ofreciendo resultados satisfactorios, como pudimos verlo en los experimentos 3 y 4. Sin embargo, el procesamiento es aproximadamente el doble en comparación con las imágenes descargadas de internet.

Experimentar con productos similares conlleva una ligera disminución en la exactitud y precisión del modelo, aunque en un porcentaje bajo, como pudimos notarlo en los experimentos 2,4,5 y 6.

Aumentar el número de vinos dentro de una clase en comparación con otra conlleva una reducción en la precisión y exactitud del modelo, como pudimos verlo en los experimentos 5 y 6, además de aumentar el tiempo de procesamiento. Esto se debe a la necesidad de buscar con mayor detalle las diferencias entre las botellas para lograr una clasificación más precisa. Además, del experimento 5 y 6 podemos notar que al incrementar la cantidad de vinos específicos en una clase en comparación con otra que no contiene ninguno de esos vinos, se observa que la red está efectivamente reconociendo características de las etiquetas de las botellas que van más allá de la forma y el color. Esto sugiere que el modelo utiliza características de las etiquetas para identificar las diferencias entre las clases.

Bibliografía

Agatonovic-Kustrin, S. &. (2000). Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. . *Journal of pharmaceutical and biomedical analysis,* , 22(5), 717-727.

AWS. (19 de 12 de 2023). *Características de Amazon Textract*. Obtenido de AWS: https://aws.amazon.com/es/textract/features/

Banerjee, K. G. (2020). Exploring alternatives to softmax function. *arXiv preprint* .

Bishop, C. M. (2006). *Pattern recognition and machine learning.* New York: pringer.

Daubechies, I. D. (2022). Nonlinear approximation and (deep) ReLU networks. . *Constructive Approximation*, 127-172.

Escudero, J. C. (2011). Discpacidad visual y ceguera en el adulto. *Medicina UPB*, 170 - 180.

Goodfellow, I. B. (2016). *Deep Learning.* MIT Press.

GOOGLE. (19 de 12 de 2023). *Cloud Vision API*. Obtenido de Cloud Vision API: https://cloud.google.com/vision

He, K. Z. (2016). Deep residual learning for image recognition. *IEEE conference on computer vision an pattern recognition* , 770 - 778.

Hegghammer, T. (2022). OCR with Tesseract, Amazon Textract, and Google Document AI: a benchmarking experiment. *Journal of Computational Social Science*, 861--882.

Huang, L. C. (2022). Multi-scale feature fusion convolutional neural network for indoor small target detection. *Frontiers in Neurorobotics*, 16, 881021.

Ianovski, A. (19 de 12 de 2023). *medium*. Obtenido de medium: https://medium.com/@alexandre3k/analyzing-wine-bottle-labels-with-aws-textract-d255a1c9d572

Jie, H. J. (2020). RunPool: A Dynamic Pooling Layer for Convolution Neural Network. . *Int. J. Comput. Intell. Syst.,* , 13(1), 66-76.

Khan, S. R. (2018). A guide to convolutional neural networks for computer vision San Rafael . *Morgan & Claypool Publishers.*, 1-187.

Kurani, A. D. (2023). A comprehensive comparative study of artificial neural network (ANN) and support vector machines (SVM) on stock forecasting. . *Annals of Data Science*, 10(1), 183-208.

Liu, W. W. (2017). A survey of deep neural network architectures and their applications. *Elsevier*, 11-26.

Mashiata, M. A. (2022). Towards assisiting visually impaired individual. *A review on current status and future prospects Biosensor and Bioelectronic* , 10 - 12.

Medsker, L. R. (2001). *Recurrent neural networks. Design and Applications.* LR Medsker: Washington, D.C.

Memon, J. a. (2020). Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR). *IEEE Access*, 142642-142668.

ONU. (10 de Septiembre de 2023). *Organizacion Mundial de la Salud* . Obtenido de Organizacion Mundial de la Salud : https://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment

Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. . *arXiv preprint*.

Seifert, C. A. (2017). Visualizations of deep neural networks in computer vision, A survey. *Transparent data mining for big and small data*, 123-144.

Srinivas, S. S. (2016). A taxonomy of deep convolutional neural nets for computer vision. *frontiers in Robotics an AI*, 2,36.

Sudholt, S. &. (2016). A deep convolutional neural networks for word spotting in handwritten documents. *international Conference on Frontiers in Hendwriting Recognition*, 277 - 282 .

Suriya, M. C. (2022). Enhanced deep convolutional neural network for malarial parasite classification. . *International Journal of Computers and Applications*, 44(12), 1113-1122.

Sutskever, I. (2013). *Training recurrent neural networks.* Toronto ON, Canada University of Toronto.

Tafti, A. P. (2016). OCR as a service: an experimental evaluation of Google Docs OCR, Tesseract, ABBYY FineReader, and Transym. In Advances in Visual Computing. *12th International Symposium,* , 735-746.

Wang, Z. J. (2020). CNN explainer. *IEEE Transactions on visualization and computer Graphics*, 1396-1406.

Wolffsohn, J. S. (2003). A review of current knowladge on Electronic Vision . *Enhancement Systems for the visually impaired* , 35-42.

Wu, W. D. (2014). Protocol for developing ANN models and its application to the assessment of the quality of the ANN model development process in. *Environmental Modelling y fotware* , 108-127.

Wu, X. S. ((2020).). Recent advances in deep learning for object detection. . *Neuroinformática*, 396, 39-64.