CONTROL DEL MOUSE CON RECONOCIMIENTO DE GESTOS MANUALES MEDIANTE IMÁGENES E INTELIGENCIA ARTIFICIAL.

JUAN DANIEL ESPINOZA CARO
SEBASTIAN ARDILA LEAL

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

CONTROL DEL MOUSE CON RECONOCIMIENTO DE GESTOS MANUALES MEDIANTE IMÁGENES E INTELIGENCIA ARTIFICIAL.

JUAN DANIEL ESPINOZA CARO
SEBASTIAN ARDILA LEAL

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
Jaime Guillermo Barreo Pérez
Magíster en potencia eléctrica

Codirector
Carlos Alberto Flórez Arias
Magíster en Ingeniería Mecánica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

LISTA DE ANEXOS

pág.

GLOSARIO

VISION POR COMPUTADORA Es un campo de la informática que abarca la forma en que las computadoras pueden comprender imágenes y videos. Se utiliza para extraer información de imágenes y videos, información tal como: objetos, personas, texto, colores, etc.

API (Interfaz de Programación de Aplicaciones) Es un conjunto de reglas y protocolos que permite la comunicación entre diferentes sistemas o aplicaciones. Una API actúa como un intermediario entre dos aplicaciones, permitiendo que una aplicación acceda a las funciones o datos de otra aplicación.

INTERFAZ Es un punto de conexión o una forma de comunicación entre dos sistemas o componentes.

SOBREAJUSTE Es un fenómeno en el que un modelo de aprendizaje automático se ajusta demasiado bien a los datos de entrenamiento, pero no generaliza bien a datos nuevos o desconocidos.

PRECISIÓN Es una medida que indica cuán bien un modelo de aprendizaje automático es capaz de predecir o clasificar correctamente los datos. La precisión se calcula como el porcentaje de veces que el modelo hace una predicción correcta sobre el conjunto de datos de prueba.

ANCLAS Las anclas en modelos de detección, son cajas rectangulares que se colocan en diferentes ubicaciones y escalas en la imagen de entrada. El modelo de detección compara cada ancla con una serie de objetos conocidos o patrones para determinar si la ancla contiene un objeto de interés. Si se detecta un objeto, el modelo ajusta la posición y el tamaño de la caja de anclaje para que se ajuste mejor al objeto detectado. Las anclas permiten que el modelo de detección detecte

objetos de diferentes tamaños y en diferentes ubicaciones en la imagen, lo que es fundamental para la detección precisa de objetos en una variedad de situaciones y entornos.

RESUMEN

TÍTULO: CONTROL DEL MOUSE CON RECONOCIMIENTO DE GESTOS MANUALES MEDIANTE IMÁGENES E INTELIGENCIA ARTIFICIAL. *

AUTOR: JUAN DANIEL ESPINOZA CARO, SEBASTIAN ARDILA LEAL **

PALABRAS CLAVE: RECONOCOCIMIENTO DE GESTOS MANUALES, REDES NEURONALES CONVOLUCIONALES, VISIÓN POR COMPUTADORA.

DESCRIPCIÓN:

Con el constante avance de la visión por computadora y el gran número de personas que ahora cuentan con un ordenador debido a la pandemia, es esencial aprovechar todas las herramientas posibles para automatizar procesos que faciliten a las personas el desarrollo de sus actividades diarias relacionadas con un ordenador. Por ello, el presente proyecto plantea el desarrollo de un sistema basado en redes neuronales convolucionales y visión por computadora que se ejecuta con una interfaz amigable para el usuario. Luego de obtener el conjunto de datos para el reconocimiento de gestos, se realizó el entrenamiento de múltiples modelos de redes convolucionales, y después de realizar una comparación, se optó por el modelo basado en la arquitectura YOLOv8. Posteriormente, se buscó y adaptó un modelo para esqueletización de la mano, dicho modelo fue tomado de la biblioteca de Mediapipe. Al contar con los dos modelos listos para su despliegue, se realizó la integración de ambos modelos en la interfaz creada con la herramienta PyQt5, y finalmente se definieron los modos de uso para el control del mouse y la activación de macros.

---

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director:Jaime Guillermo Barreo Pérez, Magíster en potencia eléctrica. Codirector:Carlos Alberto Flórez Arias, Magíster en Ingeniería Mecánica

ABSTRACT

TITLE: MOUSE CONTROL WITH MANUAL GESTURE RECOGNITION USING IMAGES AND ARTIFICIAL INTELLIGENCE. *

AUTHOR: JUAN DANIEL ESPINOZA CARO, SEBASTIAN ARDILA LEAL **

KEYWORDS: RECOGNITION OF MANUAL GESTURES, CONVOLUTIONAL NEURAL NETWORKS, COMPUTER VISION.

DESCRIPTION:

With the constant advancement of computer vision and the large number of people who now have a computer due to the pandemic, it is essential to take advantage of all possible tools to automate processes that facilitate people in the development of their daily activities related to a computer. Therefore, the present project proposes the development of a system based on convolutional neural networks and computer vision that runs with a user-friendly interface. After obtaining the data set for gesture recognition, the training of multiple convolutional network models was performed, and after a comparison, the model based on the YOLOv8 architecture was chosen. Subsequently, a model for hand skeletonization was searched for and adapted; this model was taken from the Mediapipe library. Once the two models were ready to be deployed, both models were integrated into the interface created with the PyQt5 tool, and finally the modes of use for mouse control and macro activation were defined.

---

* Bachelor Thesis

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director:Jaime Guillermo Barreo Pérez, Magíster en potencia eléctrica. Codirector:Carlos Alberto Flórez Arias, Magíster en Ingeniería Mecánica

INTRODUCCIÓN

La visión por computadora es, sin duda, una rama del aprendizaje automático que promete lograr grandes cosas. Analizar y comprender tanto imágenes como videos es una característica que puede facilitar el proceso de interacción hombre-máquina (HCI).

Con la reciente pandemia, un gran porcentaje de la población tuvo que adaptarse a una virtualidad sofocante, en donde contar con una computadora se convirtió en una necesidad y no en un lujo. En el caso de los docentes, realizar su labor implicó un proceso de cambio en donde, al final, tenían que estar frente a una pantalla por un tiempo prolongado. Con la intención de facilitarles un poco esta tarea y generar una alternativa atractiva para personas que deseen experimentar de primera mano la interacción hombre-maquina, se propuso un sistema basado en el seguimiento y detección del gesto de la mano para realizar el control del mouse y algunos macros del sistema operativo.

Puesto que en la actualidad es común contar con cualquier tipo de cámara RGB, ya sea una cámara web, una cámara integrada en la computadora o la cámara que traen la mayoría de celulares, el software se orientó a la compatibilidad con este tipo de cámaras.

El desarrollo del software se separó en dos partes: el seguimiento de la mano y la detección del gesto. Para abordar el proceso de seguimiento de la mano, se optó por realizar la esqueletización de la mano para determinar su posición exacta. Para este proceso, se utilizó un modelo desarrollado por MediaPipe y adaptado para cumplir con la funcionalidad de un mouse. Por otro lado, para la detección del gesto, se recurrió al entrenamiento de una red neuronal profunda que fuera capaz de identificar 8 gestos diferentes con el propósito de configurarlos para controlar acciones específicas del sistema operativo.

1. OBJETIVOS

Objetivo general

- Diseñar e implementar un sistema que permita el reconocimiento de gestos de la mano mediante el uso de una cámara RGB, con el fin de controlar el cursor y facilitar el uso de algunas aplicaciones para el computador.

Objetivos específicos

- Realizar o encontrar y acondicionar un dataset que permita el entrenamiento del algoritmo de aprendizaje profundo.
- Entrenar un algoritmo de aprendizaje profundo, que permita el reconocimiento de la mano como cursor y a su vez reconozca gestos que pueda interactuar con aplicaciones orientadas a docencia.
- Realizar pruebas para verificar que el sistema cumple con las funciones establecidas y solucione la problemática expuesta.
- Realizar un manual que permita identificar los usos del sistema.

2. ESTADO DEL ARTE.

2.1. MARCO ÉTICO PARA LA IA EN COLOMBIA.

Antes de comenzar a abordar el proyecto, se analizó si existen normas éticas para la IA, de esto se encontró un marco ético para la IA en Colombia [1]. Luego de la revisión del documento, se concluyó que se trabajó bajo los estándares del Marco Ético con respecto a la I.A en Colombia.

2.2. RECONOCIMIENTO DE GESTOS MANUALES.

El reconocimiento de gestos es un tema en constante crecimiento en los últimos años. Abarca una gran cantidad de funcionalidades, por lo que este proyecto se enfocará en una pequeña sección de este tema: el reconocimiento de gestos manuales. Se utilizaron varios métodos para lograr un reconocimiento óptimo del gesto deseado, algunos de los cuales incluyen el uso de guantes con sensores[2], cámaras con acceso a información de profundidad [3] [4] [5] y la usada en este proyecto: Cámaras

---

[1] A. Uribe Guío y Pablo E. Ayerbe. *MARCO ÉTICO PARA LA INTELIGENCIA ARTIFICIAL EN COLOMBIA*. 1ra. Bogotá: Gobierno de Colombia, 2021.

[2] D.J. Sturman y D. Zeltzer. "A survey of glove-based input". En: *IEEE Computer Graphics and Applications* 14.1 (1994), págs. 30-39. DOI: 10.1109/38.250916.

[3] "MaHG-RGBD: un conjunto de datos RGB-D de gestos manuales con vista multiángulo para reconocimiento de gestos basado en aprendizaje profundo y evaluaciones de referencia". En: *2019 Conferencia internacional IEEE sobre electrónica de consumo (ICCE)*. DOI: 10.1109/ICCE.2019.8661941.

[4] "Reconocimiento robusto de la postura de la mano basado en imágenes RGBD". En: *La 26.ª Conferencia China de Control y Decisión (2014 CCDC)*. DOI: 10.1109/CCDC.2014.6852636.

[5] Rajesh George Rajan y P. Selvi Rajendran. "Gesture recognition of RGB-D and RGB static images using ensemble-based CNN architecture". En: *2021 5th International Conference on Intelligent*

RGB[5] [6]. Puesto que permiten recolectar gran cantidad de información que se puede poner a disposición del algoritmo de clasificación.

Para este proyecto se optó por el uso de cámaras RGB, ya que las otras alternativas generan un costo significativo en la implementación de las mismas. Por ejemplo la cámara RGB-D más económica se encuentra en alrededor de 149$ Dólares (Luxonis Oak-D-Lite).

Este reto de reconocer exitosamente el gesto de la mano, tiene tantas formas de abordar pero sin duda una de las más precisas son el uso de cámaras con profundidad, puesto que ellas permiten con la ayuda de una nube de puntos distinguir de forma más precisa la mano del ambiente caótico que pueda presentarse en la imagen, pero se descartó debido a su costo mas elevado, acotando el público al cual podría dirigirse esta solución.

En la captura de información por medio de sensores se pueden encontrar diversos métodos, tales como: Guantes con acelerómetro y giroscopio para determinar la posición y la postura de la mano.

Por medio de la electromiografía en donde se miden los pulsos eléctricos generados por los músculos y éstos se interpretan como el movimiento de los dedos. Algunos otros métodos involucran radares, tecnologías electromagnéticas y mecánicas. En el pasado la única tecnología capaz de satisfacer los altos estándares de la industria era la implementación de guantes que por medio de sensores realizaba la detección de pose, pero éste método tenía ciertos inconvenientes que empujaban a los investigadores a encontrar nuevos métodos para resolver el problema, algunos de estos

---

Computing and Control Systems (ICICCS). 2021, págs. 1579-1584. DOI: 10.1109/ICICCS51141.2021.9432163.

[6] Zhihua Hu y Xiaoming Zhu. “Gesture detection from RGB hand image using modified convolutional neural network”. En: *2019 2nd International Conference on Information Systems and Computer Aided Education (ICISCAE)*. 2019, págs. 143-146. DOI: 10.1109/ICISCAE48440.2019.221606.

inconvenientes eran la difícil adaptación por la que tenía que pasar el usuario para aprender el uso debido de estas herramientas y sus largos y complicados procesos de calibración.

Con este desafío de encontrar nuevas tácticas y métodos que fueran más amigables con el usuario surgió una nueva corriente para HCI (interacción hombre computadora), este método se conoce como visión artificial (CV) o visión por computadora, que pretende por medio de cámaras capturar imágenes que al ser debidamente preprocesadas proporcionen información suficiente para su interpretación por parte de un ordenador, las ventajas con más peso y por la cual se considera una fuerte competidora en el ámbito del reconocimiento de gestos es su fácil adaptabilidad para los usuarios y eliminar el uso de equipos mucho más complejos de utilizar. De esta nueva corriente de investigación surgen dos rumbos, el reconocimiento del gesto y el reconocimiento de la pose.

2.3. INTELIGENCIA ARTIFICIAL.

La inteligencia artificial surge a raíz del concepto de atribuirle a una computadora la capacidad de pensar, el estudio de esta rama de investigación logró consolidarse en los años 50 con la participación de John McCarthy, quien organizó un taller bajo la idea de que cada aspecto del aprendizaje o cualquier característica de la inteligencia puede describirse con tal detalle que se puede hacer una máquina que logre emular este atributo. Al concluir el proyecto no se logró converger a una solución completa de la problemática, pero la mayoría de los académicos que participaron del proyecto posteriormente se convertirían en pioneros en este tema [7].

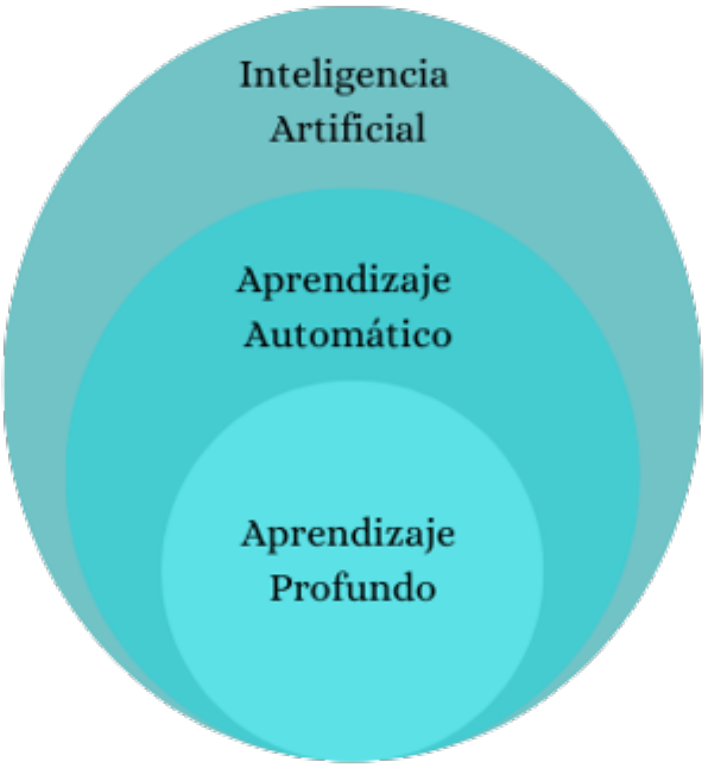
La IA específicamente se centra en el desarrollo de algoritmos que permitan realizar tareas que en principio requieren inteligencia humana, tales como problemas de

---

[7] F. Chollet. *Deep Learning with Python, Second Edition*. Manning, 2021, págs. 1-5.

razonamiento, comprensión del lenguaje, etc. “La inteligencia artificial es la rama de la informática que se ocupa de hacer que las computadoras se comporten como seres humanos” [8].

Figura 1. Estructura de la IA.



Dentro del concepto de inteligencia artificial podemos encontrar un gran pilar de esta área, el Machine Learning, una técnica de aprendizaje automático que le permite a los algoritmos aprender de los datos sin ser explícitamente programadas, se usan con el fin de crear sistemas que puedan tomar decisiones de forma autónoma con base en los datos que se tengan. Dentro del Machine Learning existen tres meto-

[8] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010, págs. 1-4.

dologías que se diferencian en la forma en la que el algoritmo realiza su proceso de aprendizaje. El aprendizaje supervisado se caracteriza por tener asignada una etiqueta a cada uno de los datos de entrenamiento para así realizar nuevas predicciones. En el aprendizaje no supervisado los datos que se le dan al algoritmo no están etiquetados y su función es hallar patrones en los datos. Por último, está el aprendizaje por refuerzo en el cual el sistema recibe una recompensa o un castigo respecto a las decisiones que toma y con esta información se retroalimenta para mejorar su desempeño.

El aprendizaje profundo es otra técnica del aprendizaje automático. Se basa en las redes neuronales artificiales con muchas capas, de allí su nombre de aprendizaje profundo. Este tipo de redes se caracterizan por aprovechar los grandes conjuntos de datos, en la actualidad son principalmente usadas en tareas de visión por computadora y reconocimiento de voz.

Algunas de las técnicas más conocidas y con mayor uso en la actualidad dentro del deep learning son las redes GAN, VAE y LLM. Las redes generativas (GAN) son un tipo de modelo de aprendizaje profundo que se utiliza para generar nuevos datos con características similares al conjunto de entrenamiento. En una red GAN se entrenan dos modelos simultáneamente: un generador y un separador. El generador crea nuevas muestras y el discriminador evalúa si estas muestras son correctas o incorrectas. A medida que se entrena la red, el generador aprende a generar muestras cada vez más realistas, mientras que el discriminador aprende a distinguir entre muestras generadas y reales.[9]

Similar al objetivo de las redes GAN las redes VAE (Variational Autoencoders) se utilizan para generar datos sintéticos. Los VAE usan una arquitectura de red similar a los codificadores automáticos, pero con un giro: los VAE generan nuevos datos al

---

[9] Ian Goodfellow et al. “Generative Adversarial Nets”. En: *Advances in Neural Information Processing Systems*. Ed. por Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014.

muestrear una distribución de probabilidad oculta en lugar de simplemente copiar la entrada. Este proceso de muestreo permite que los VAE produzcan datos nuevos y diferentes de la entrada original, lo que puede conducir a una mayor variabilidad en los datos producidos.[10]

Por otro lado las LSTM(Redes de memoria a corto y largo plazo) Es una variante de las redes neuronales recurrentes que se utilizan en el procesamiento del lenguaje natural. Las redes LSTM se diferencian de otras redes neuronales recurrentes en que están diseñadas para resolver problemas como el desvanecimiento del gradiente, para ello implementan unidades de memoria en la arquitectura de la red, esto les permite recordar patrones a largo plazo de datos secuenciales. Esta capacidad de memoria de las redes LSTM hace que sean útiles para tareas de procesamiento de lenguaje natural, como traducción automática, reconocimiento de voz y generación de texto.[11]

2.3.1. Base de datos Una base de datos o conjunto de datos, es la base de todo proceso de entrenamiento de una red neuronal, la calidad de estos datos determinarán el máximo rendimiento que podrá lograr un modelo.

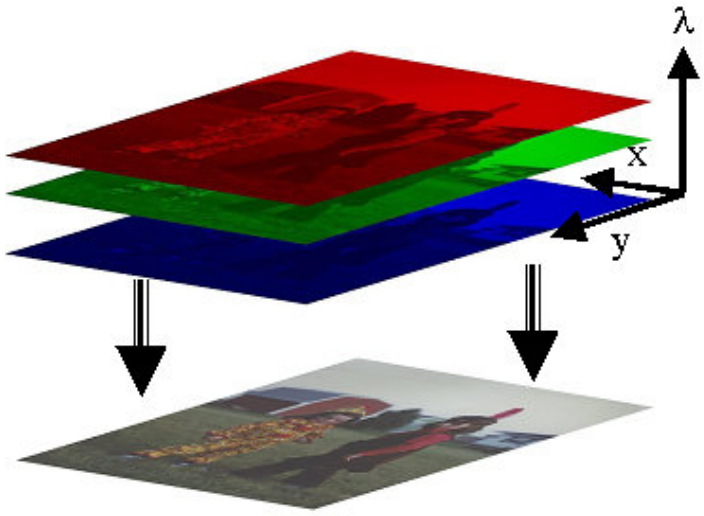
Para el proceso de clasificación de gestos, es necesario comenzar con un conjunto de datos compuesto por imágenes digitales, ellas están constituidas por tres matrices(RGB) bidimensionales que contienen información de la intensidad de color de cada pixel, este rango de intensidad se da entre 0 y 255.

---

[10] Carl Doersch. *Tutorial on Variational Autoencoders*. 2021. arXiv: 1606.05908 [stat.ML].

[11] Andrej Karpathy, Justin Johnson y Li Fei-Fei. *Visualizing and Understanding Recurrent Networks*. 2015. arXiv: 1506.02078 [cs.LG].

Figura 2. Imagen digital RGB.



Fuente: Tebow, Christopher & Dereniak, Eustace. (2004). Tunable Snapshot Spectrometer Feasibility Study. 139.

El procesamiento de imágenes es un procedimiento estándar en el mundo del aprendizaje profundo, que consiste en operar los valores numéricos que componen cada una de las capas o dimensiones de la imagen, estas operaciones se realizan con diversos fines, como normalizar las muestras, aplicar filtros, transformar a escala de grises, entre otros. Todo para extraer la mayor cantidad de información.

2.3.2. Función de costo Para el entrenamiento de una red neuronal se requiere que el modelo u algoritmo procese el estímulo entregado (muestra) y con base en esta, prediga a que categoría pertenece. Esta predicción se tiene que evaluar para definir su fiabilidad.[12]

[12] "Introduction". En: *Neural Networks: Tricks of the Trade*. Ed. por Genevieve B. Orr y Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, págs. 1-2. DOI: 10.1007/3-540-49430-8_1.

Con el fin de evaluar las predicciones del modelo, surgen distintas funciones que permiten calcular que tan alejado está la predicción del modelo, de la predicción deseada, a este grupo de funciones se les conoce como funciones de costo.

$$E^p = \frac{1}{2}(D^p - M(Z^p, W))^2 \tag{1}$$

La Ecuación 1 comprende una de las funciones de costo más sencillas que se pueden encontrar, el error medio cuadrático(MSE) que consta de M(predicción del modelo, con base en una entrada Z y unos pesos W) y D(predicción deseada), al realizar la diferencia de estas se puede calcular el error de la predicción del modelo, esto para cada muestra de una época, por último se realiza un promedio de los errores obtenidos con el fin de observar el comportamiento del modelo y reajustar los pesos para obtener mejores resultados.

2.3.3. Propagación inversa. Entrenar una red neuronal es un procedimiento arduo que sin duda puede llegar a tener complicaciones por la alta complejidad que hay en actualizar sus parámetros puesto que dependiendo de la cantidad de capas ocultas que contenga la red, el número de operaciones aumenta de manera proporcional. Para solucionar los problemas en la actualización de los parámetros de la red surge el concepto de retro propagación o propagación inversa.

El proceso de propagación inversa es la forma en que se determina el gradiente de la función de costo, con esta función y en compañía del descenso de gradiente es posible actualizar los pesos del modelo y mejorar su desempeño. Este método de obtención del gradiente se basa en obtener el jacobiano de la función de costo con ayuda de la regla de la cadena.

Para actualizar los pesos del modelo también se debe tener en cuenta que este proceso se realiza desde la última capa del modelo hasta la primera puesto que la salida de la última capa es conocida y el cálculo de su gradiente es más simple,

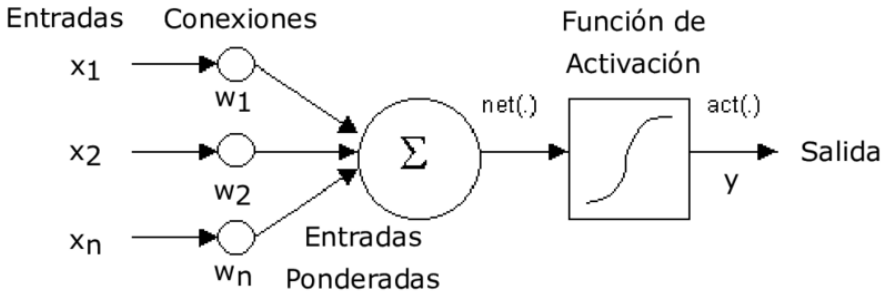
además sirve como punto de partida para los cálculos posteriores.[13]

$$W_l = W_{l-1} - \alpha(\frac{\delta Fc}{\delta W}) \tag{2}$$

La Función 2 describe el proceso que se lleva a cabo para la actualización de W donde se observa un nuevo parámetro. α o también conocido como factor de aprendizaje, es el que determina la sensibilidad del ajuste que se lleva a cabo con el proceso del descenso del gradiente, este proceso se realiza para encontrar de manera más efectiva el mínimo de la función de costo y evitar un estancamiento en el aprendizaje del modelo.

2.3.4. Función de activación En el proceso de aprendizaje de una red neuronal, hay varios parámetros que influyen en la probabilidad obtenida como salida. Como se puede observar en la Figura 3.

Figura 3. Estructura de una neurona.

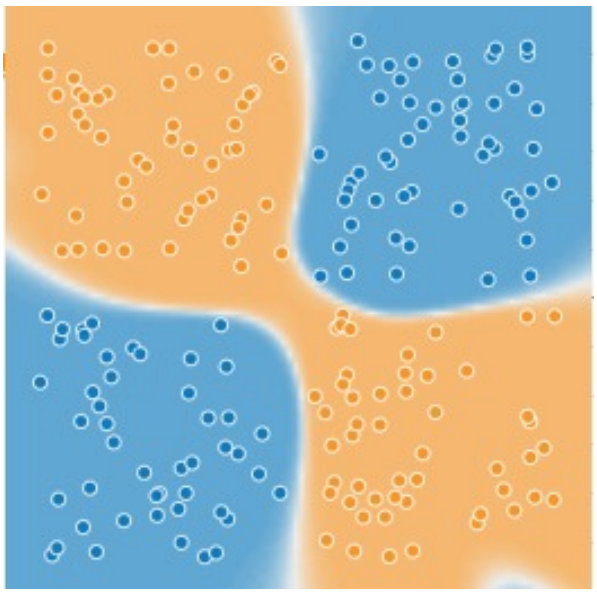


Fuente: Ardila, Jeyson & Suárez Mantilla, Luis. (2019). CLASIFICACIÓN DE REGISTROS FONOCARDIOGRÁFICOS USANDO DESCOMPOSICIÓN EMPÍRICA EN MODOS Y REDES DE GRAN MEMORIA DE CORTO PLAZO. 10.13140/RG.2.2.24258.61125.

[13] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep learning*. MIT press, 2016, pág. 197.

La arquitectura estándar de una neurona en un modelo de aprendizaje profundo está compuesta por Xm entradas y Wm pesos. Estos pesos se multiplican por cada entrada y se suman para obtener una salida que contiene la información de la probabilidad que será seleccionada.

Figura 4. Clasificación con función de activación.



En caso de estar frente a problemas que no requieren de una no linealidad para completar su proceso de clasificación, la adición de una función de activación no tiene mucho sentido. La adición de una función de activación en la mayoría de los casos está reservada para problemas más complejos de clasificación, ya que permiten a las redes neuronales aprender patrones no lineales de los datos como en la Figura 4.

Luego de identificar que el modelo se enfrenta a una tarea de clasificación de patrones no lineales, se debe escoger una función de activación. Hay una amplia variedad de funciones de activación, ya que en algunas ocasiones con fines de investigación

se realizan funciones personalizadas, pero dentro de las más conocidas podemos encontrar las siguientes: Relu, sigmoid, tanh, softmax.[14]

$$ReLU : f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{3}$$

$$sigmoid : f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

$$Tanh : f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5}$$

$$Softmax : f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{6}$$

Es importante escoger la función de activación correcta para cada capa en la red profunda, cada función de activación realiza una transformación lineal diferente: ReLU (Rectified Linear Unit) Se encarga de introducir una discontinuidad en la que todos los valores negativos se hacen cero. Esta función es la más popular en cuanto a redes convolucionales en capas intermedias, debido a su simplicidad. Las ventajas más claras de esta función de activación son:

- En el proceso de entrenamiento asegura una rápida convergencia del modelo gracias a su baja complejidad de cálculos matemáticos.[15]
- Ya que ReLU no tiene en cuenta los valores negativos de las neuronas, ayuda a evitar el problema del desvanecimiento del gradiente permitiendo el desarrollo

---

[14] Charu C. Aggarwal. *Neural Networks and Deep Learning. A Textbook*. Cham: Springer, 2018, págs. 15-16. DOI: 10.1007/978-3-319-94463-0.

[15] Xavier Glorot, Antoine Bordes y Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. en. En: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop y Conference Proceedings, jun. de 2011, pág. 318.

de redes más profundas.[16]

Por otro lado, también tiene ligeras desventajas a considerar como la muerte neuronal, que se produce debido a que ReLU ignora las entradas menores a cero, esto conlleva a que la neurona no pueda actualizarse y muera en el entrenamiento.[17]
Por otro lado, es importante mencionar que luego de escoger una función de activación es una buena práctica inicializar los pesos de cada capa acorde a la función seleccionada, esto con el fin de evitar a toda costa el desvanecimiento del gradiente.

Tabla 1. Inicialización de pesos.

Inicialización	**Función de activación**
Glorot	Ninguna, Tanh, Logística, Softmax
He	ReLU y variantes
LeCun	SELU

2.3.5. Redes neuronales convolucionales Las redes neuronales convolucionales es una clase de red neuronal artificial que surge alrededor de los años 80 para realizar tareas de reconocimiento de patrones. Posteriormente en los 90 ya era posible reconocer caracteres e imágenes, en los años siguientes su uso fue creciendo exponencialmente puesto que el nivel de potencia de procesamiento y el aumento de datos permitían sacar mejor provecho a las prestaciones que estas ofrecían. En la Figura 5 se observa la arquitectura estándar de una red neuronal convolucional. El principio de funcionamiento de estas redes está inspirado en la forma como el cerebro humano procesa la información visual, los principales componentes de una

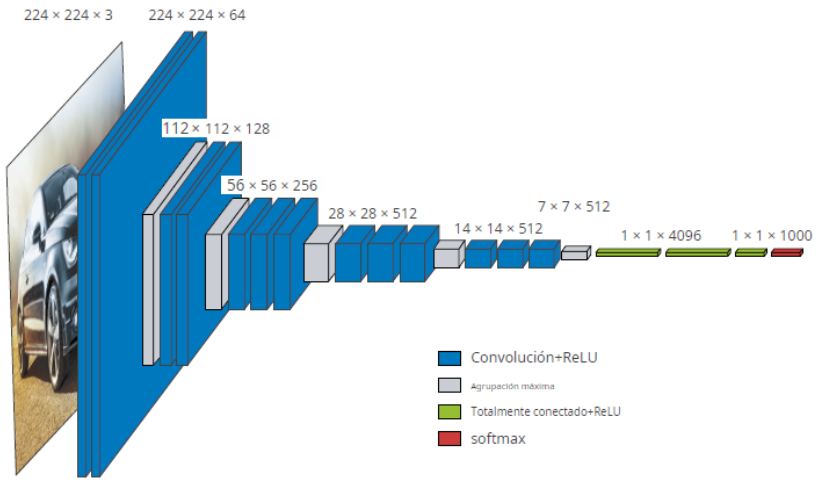
---

[16] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http : / / www . deeplearningbook.org. MIT Press, 2016, pág. 193.

[17] Xavier Glorot, Antoine Bordes y Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. En: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Geoffrey Gordon, David Dunson y Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pág. 318.

CNN (Convolutional Neuronal Network) son: filtros, capas convolucionales, capas de agrupación máxima y capas densamente conectadas.

Figura 5. Arquitectura de red neuronal convolucional.



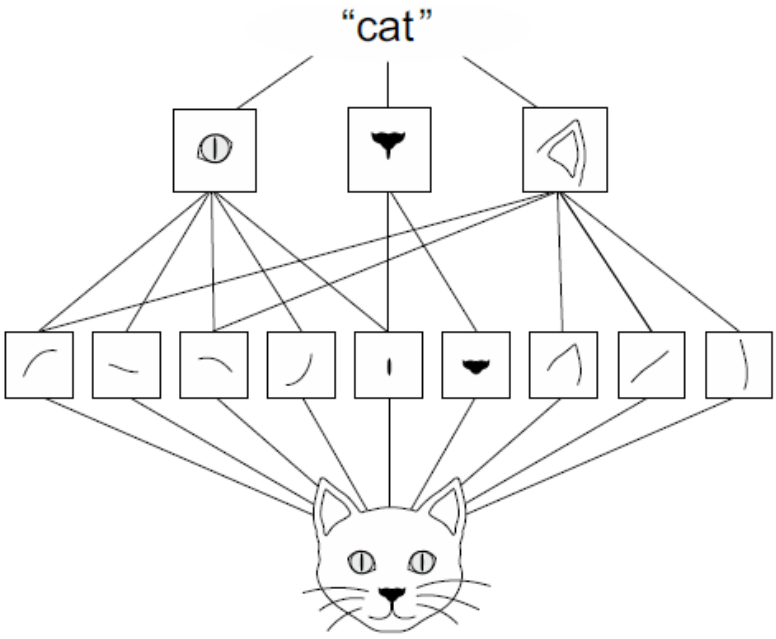
Fuente: Chollet, F. (2018). Deep Learning with Python. Second Edition. Shelter Island, NY: Manning Publications Co.

Los filtros como semejantes de las neuronas en las redes tipo perceptrón, son entrenados para reconocer patrones en los datos, por lo que sus valores se ajustan a medida que se ejecuta el proceso de entrenamiento, estos filtros tienen la tarea de recorrer toda la imagen en busca de patrones determinados como pueden ser trazos o hasta formas específicas.[18]

Normalmente, después de una capa convolucional, suele venir una capa de agrupación máxima. Esta capa permite sintetizar mejor el mapa de características obtenido por la capa convolucional, y al mismo tiempo reduce el nivel de procesamiento necesario. Los filtros de la capa convolucional tienen un tamaño específico que se ajusta de acuerdo a los datos disponibles. En algunas ocasiones, debido al tamaño del fil-

[18] F. Chollet. *Deep Learning with Python, Second Edition*. Manning, 2021, págs. 204-206.

Figura 6. Funcionamiento de los filtros.



Fuente: Chollet, F. (2018). Deep Learning with Python. Second Edition. Shelter Island, NY: Manning Publications Co.

tro, la capa convolucional también puede reducir el tamaño de la imagen. Para evitar esta pérdida de tamaño de la imagen, se recomienda usar la herramienta "padding", la cual permite redimensionar la imagen.

Por último, están las capas completamente conectadas. Estas capas reciben un mapa de características mucho más reducido, lo que ayuda a procesar la salida de las capas convolucionales. La información procesada se envía a la capa de clasificación.

2.4. YOLO

YOLO (You Only Look Once) es un algoritmo de detección de objetos en tiempo real que cataloga la detección como un problema de regresión a cuadros delimitadores,

técnica en la que son seleccionadas regiones de interés para su posterior clasificación [19], dichos cuadros son espacialmente separados y cuentan con probabilidades de clase asociadas. Es posible realizar un ajuste entre robustez del modelo y calidad de las predicciones variando el modelo base que se va a entrenar, Ultralytics en su repositorio dispone un total de 5 modelos para el entrenamiento de YOLOv8.[20]

Lo que hace a YOLO tan eficiente es su funcionamiento, se basa en una red unificada que se encarga de predecir cuadros delimitadores y probabilidades de clase, dado que todo el proceso de detección lo realiza una sola red, esta se puede optimizar de extremo a extremo.

La última capa de la red tiene la tarea de predecir tanto probabilidades de clase como las coordenadas del cuadro delimitador, así como su ancho y alto. Ya que se pueden presentar casos donde estén múltiples objetos, la red entrega a la salida un tensor con las coordenadas de cada objeto, así como su respectiva clase.

Puesto que un objeto puede abarcar más de una celda, en ocasiones se presentaba el problema de detección de múltiples cajas contenedoras para el mismo objeto, con el fin de eliminar dicho problema, YOLO implementa la funcionalidad supresión no máxima, técnica usada anteriormente por modelos como [21], la cual permite determinar cuándo las cajas se sobreponen y se puedan eliminar las cajas innecesarias.[22]

La arquitectura de la red YOLOv8 se puede encontrar en el Anexo 3.

---

[19] "Jerarquías de características enriquecidas para la detección precisa de objetos y la segmentación semántica". En: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2014.81.

[20] Glenn Jocher, Ayush Chaurasia y Jing Qiu. *YOLO by Ultralytics*. Ver. 8.0.0. Ene. de 2023.

[21] En: *2015 Conferencia Internacional IEEE sobre Visión por Computador (ICCV)*, págs. 1440-1448. DOI: 10.1109/ICCV.2015.169.

[22] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

2.5. MODELOS DE ESQUELETIZACIÓN DE LA MANO

Con el objetivo de encontrar una alternativa al mouse que permita un movimiento rápido y fluido del puntero, se decidió buscar modelos previamente entrenados capaces de esqueletizar la mano [23] [24]. Posteriormente, se utilizaría esta información para manipularla a través de programación y, de esta manera, obtener el resultado esperado. Entonces inició un proceso de investigación que nos llevó a buscar las soluciones existentes.

La investigación llevó a notar que existen distintos métodos de estimación de pose [25], estos varían al hacer una estimación 2D o 3D de la pose la mano, así como el número de puntos clave de la mano que esta reconoce ver Figura 7, esencialmente los que realizan una recreación 3D incluyen un valor adicional a cada uno de los puntos de la mano, este es la profundidad. Dicha profundidad implica un conjunto de datos tomado con una cámara de profundidad o también existe la posibilidad de usar modelos previamente entrenados que son capaces de hacer un cálculo de la misma mediante imágenes RGB [26]. Además, aunque existen muchos modelos que implementan imágenes RGB como entrada, el implementar imágenes con profundidad generalmente exhibe un rendimiento superior [27]. Sin embargo para los fines

---

[23] Octavio Arriaga et al. *Perception for Autonomous Systems (PAZ)*. 2020. arXiv: 2010.14541 [cs.CV].

[24] Google. *MediaPipe Hands*. 2019.

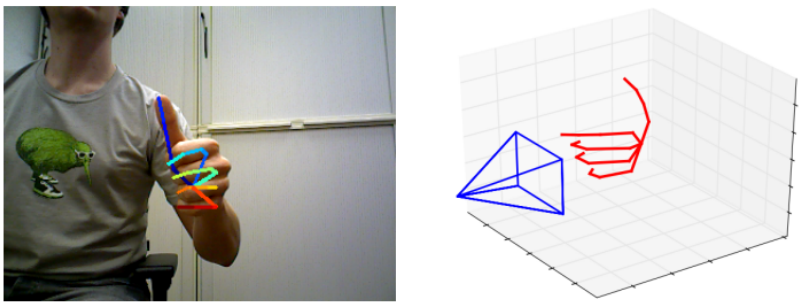
[25] Jian Cheng et al. "Efficient Virtual View Selection for 3D Hand Pose Estimation". En: 2022, págs. 5-7.

[26] Liuhao Ge et al. "3D Hand Shape and Pose Estimation From a Single RGB Image". En: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, págs. 10825-10834. DOI: 10.1109/CVPR.2019.01109.

[27] Xiao Sun et al. "Cascaded hand pose regression". En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, págs. 824-832. DOI: 10.1109/CVPR.2015.7298683.

del proyecto esta característica no se consideró como fundamental, debido a que lo más importante es obtener las coordenadas X y Y de la punta del dedo índice de la mano en pantalla.

Figura 7. Recreación de la postura de la mano.



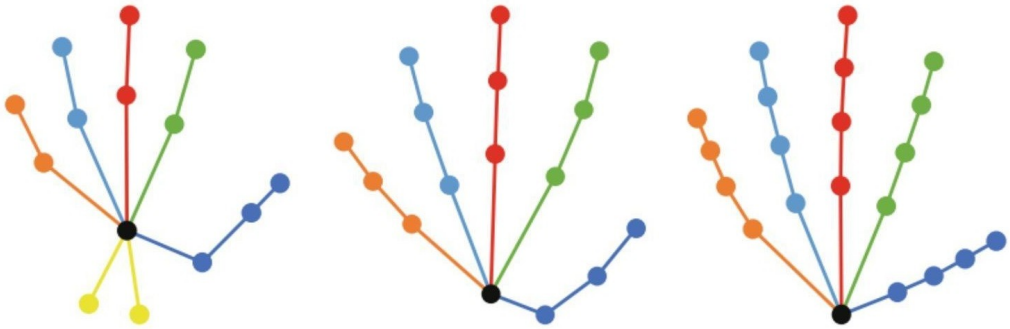
Fuente: Zimmermann, C. & Brox, T. (2017). "Learning to Estimate 3D Hand Pose from Single RGB Images". International Conference on Computer Vision, v3, pp. 1.

Por parte de la cantidad de puntos que obtenga de la mano, no existe un estándar o acuerdo global para la selección de la cantidad de puntos [28], en la Figura 8 tenemos los comúnmente usados, en este caso 14, 16 y 21 articulaciones. Para este proyecto el número de articulaciones que el modelo presente es clave, ya que este plantea un límite en el número de gestos que pueden definirse mediante programación, más puntos permiten un mayor número de combinaciones entre los mismos, es por esto que se decidió que 21 puntos serían suficientes.

---

[28] Bardia Doosti. “Hand Pose Estimation: A Survey”. En: *CoRR* abs/1903.01013 (2019), pág. 3. arXiv: 1903.01013.

Figura 8. Número de puntos usados para la esqueletización de la mano.



Fuente: Tianping Hu, Wenhai Wang & Tong Lu (2018). "Hand Pose Estimation with Attention-and-Sequence Network". Pacific Rim Conference on Multimedia, vol 11164, pp. 561.

Iniciando con el proceso, se decidió buscar modelos pre-entrenados que cumplieran los siguientes requisitos:

- **RÁPIDA INFERENCIA:** Con el fin de realizar el seguimiento de un punto de la mano y extraer sus coordenadas X y Y para usarlas directamente como la posición del cursor. Se planteó como necesario, que el modelo presente una respuesta rápida o una ”Rápida Inferencia”, esto definirá la fluidez del movimiento del cursor y hará que la herramienta sea considerada como ”funcional” o ”no funcional”.

- **NUMERO DE ARTICULACIONES:** El número de articulaciones que el modelo presente es clave, ya que este plantea un límite en el número de gestos que pueden definirse mediante programación, más puntos permiten un mayor número de combinaciones entre los mismos, es por esto que se decidió buscar modelos con 21 puntos.

2.5.1. Perception for Autonomous Systems

Perception for Autonomous Systems (PAZ) [23], se presenta como una librería de percepción jerárquica en Python, la cual ofrece 16 modelos entrenados para el reconocimiento de: Detección de objetos, clasificador de emociones, estimación de pose humana, estimación de la postura de la mano, etc. Para todo esto el repositorio PAZ cuenta con ”Abstracciones jerárquicas” [29], dichas abstracciones le permiten el usuario personalizar la implementación de los respectivos modelos como si de un sistema de bloques se tratase. En el repositorio mencionado se encontraba una amplia variedad de modelos con diferentes funcionalidades, entre los que se incluyen detección de objetos, estimación de la postura del cuerpo humano, clasificación de emociones, entre otros [29]. El modelo que llamó la atención del equipo fue ”Hand pose estimation”.

Para el proyecto se decidió realizar una prueba exigente utilizando la cámara de un dispositivo móvil de gamma baja, en este caso el Samsung J5 con una resolución de 5MP de cámara frontal, lanzado en el año 2015. Al probar el modelo entrenado por PAZ, se observó que la esqueletización inferida sobre la mano no era robusta y presenta un seguimiento a muy bajos FPS (o un tiempo de inferencia considerablemente alto para la aplicación), esto sucedía incluso en entornos correctamente iluminados. Para utilizar los movimientos de la mano como cursor se requiere un modelo robusto capaz de reconocer gestos sencillos y que cuente con un tiempo de inferencia bajo.

2.5.2. Google Mediapipe

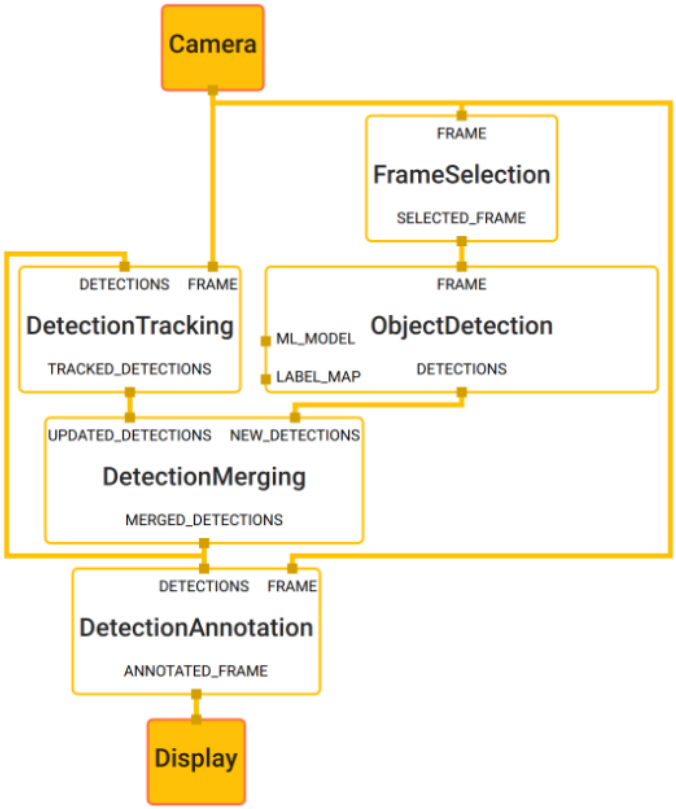
Google Mediapipe es un marco de trabajo que consta de tres partes principales (a) un marco para la inferencia de datos sensoriales, (b) un conjunto de herramientas

---

[29] Octavio Arriaga et al. “Perception for Autonomous Systems (PAZ)”. En: *CoRR* abs/2010.14541 (2020), págs. 1-2. arXiv: 2010.14541.

para la evaluación del rendimiento y (c) una colección de componentes de inferencia y procesamiento reutilizables llamados calculadoras. [30]. Permitiendo así llegar a construir de manera gráfica modelos de inferencia, algoritmos de procesamiento de medios y transformaciones de datos, etc. Ver Figura 9

Figura 9. Detección de objetos usando Mediapipe.



Fuente: Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C-L., Yong, M.G., Lee, J., Chang, W-T., Hua, W., Georg, M. & Grundmann, M. (2019). MediaPipe: A Framework for Building Perception Pipelines.

---

[30] Camillo Lugaresi et al. “MediaPipe: A Framework for Building Perception Pipelines”. En: *CoRR* abs/1906.08172 (2019), págs. 1-2. arXiv: 1906.08172.

Mediapipe se presenta con cuatro puntos fuertes: Modelos de rápida inferencia en hardware común, fácil de desplegar en las plataformas existentes (Android, IOS, desktop/cloud, web y IoT), soluciones listas para usar, gratis y de código abierto (todo esto bajo la licencia Apache 2.0, que permite a los usuarios utilizar, modificar y distribuir el software de forma gratuita, siempre y cuando se respeten ciertos términos y condiciones). Ofreciendo soluciones a problemas como: Segmentación de selfie, malla facial, seguimiento de la mano, detección y seguimiento de poses humanas, segmentación de cabello, etc [30].

Pensando en un docente o usuario con un ordenador promedio, se tendría una herramienta funcional, y al ser de código abierto permite a los desarrolladores y usuarios interesados tener acceso a su código fuente y poder utilizarlo y modificarlo de acuerdo a sus necesidades específicas.

En este caso, el modelo de interés fue "Mediapipe Hands"[31], el cual presenta una gran cantidad de parámetros configurables, tales como el máximo y mínimo de confianza en el seguimiento, número de manos a reconocer, complejidad del modelo, etc.

Se realizó la prueba del modelo con una cámara frontal de 5MP del dispositivo móvil Samsung J5. Mediante la misma se constató que era robusto y rápido en la inferencia de la esqueletización de la mano, Ofreciendo así, una alternativa al mouse al permitir un seguimiento rápido (o fluido). Por estas razones se decidió implementar Mediapipe Hands para cumplir con los objetivos del proyecto.

De acuerdo con Fan Zhang et al [31], la solución planteada está conformada por 2 modelos que trabajan en paralelo:

- **MODELO DETECTOR DE PALMA**, este opera sobre una imagen de entra-

---

[31] Fan Zhang et al. “MediaPipe Hands: On-device Real-time Hand Tracking”. En: (2020), págs. 1-2. arXiv: 2006.10214 [cs.CV].

da completa y localiza las palmas mediante un cuadro delimitador de mano orientado.

- **MODELO DETECTOR DE LOS PUNTOS CLAVES DE LA MANO**, este opera en el cuadro delimitador de mano recortada proporcionado por el detector de palma y devuelve puntos de referencia 2.5D de alta fidelidad.

Con respecto al **detector de palmas** se destaca lo siguiente:

- Se entrena el modelo detector de palma en vez del detector de mano completa con sus articulaciones, debido a que estimar cuadros delimitadores de objetos como palmas y puños es significativamente más sencillo que detectar la mano completa con sus dedos articulados [31]. Este cuadro delimitador debe ser cuadrado debido a que las palmas pueden modelarse únicamente con cuadros delimitadores cuadrados [32], esto se realiza con un modelo SSD (Single Shot Detector) optimizado para aplicaciones móviles en tiempo real similar a BlazeFace [33]. En la Figura 10 se muestra la arquitectura en alto nivel del modelo extractor de características tipo encoder-decoder, en ella se destacan las dimensiones de entrada y salida del modelo, así como el tamaño y número de anclas usadas para la correcta detección de las palmas.

- Se implementa un extractor de características tipo Encoder-Decoder similar a FPN (Feature Point Network) [34] para una mayor conciencia del contexto de la escena, incluso para detalles pequeños [31].

---

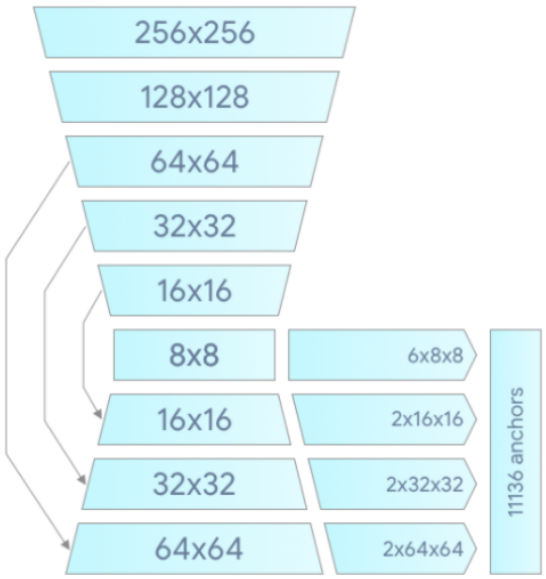
[32] Wei Liu et al. "SSD: Single Shot MultiBox Detector". En: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, págs. 21-37. DOI: 10.1007/978-3-319-46448-0_2.

[33] Valentin Bazarevsky et al. *BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs*. 2019. arXiv: 1907.05047 [cs.CV].

[34] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

- Durante el entrenamiento, se minimiza la pérdida focal como en [35], para admitir una gran cantidad de anclas resultantes de la alta varianza de escala [31].

Figura 10. Arquitectura del modelo de detector de palma.



Fuente: Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C., & Grundmann, M. (2020). MediaPipe Hands: On-device Real-time Hand Tracking.

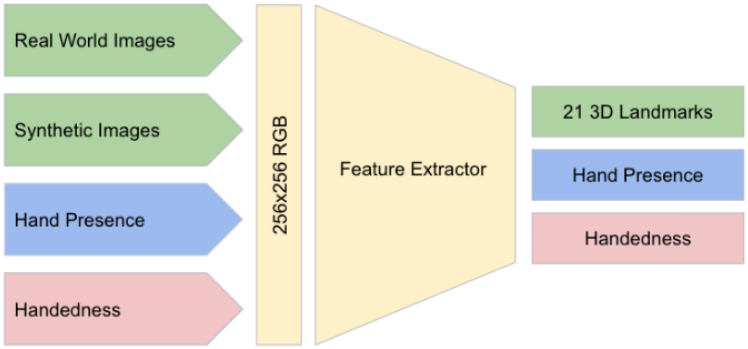
Pasando al **detector de los puntos claves de la mano**, resalta lo siguiente:

- El modelo de detección de puntos claves de la mano es robusto incluso para manos parcialmente visibles y auto-oclusiones.
- En la Figura 11 se observa la arquitectura completa del modelo, sobre la cual se destacan 3 salidas: (a) En color verde, 21 puntos clave de la mano (Ver

[35] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

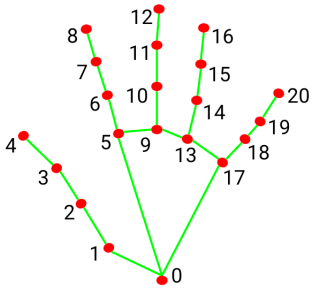
Figura 12) con coordenadas X,Y y relativa profundidad Z, (b) En color azul, una bandera que indica la probabilidad de que se encuentre una mano presente en la imagen y (c) En color rojo, una clasificación binaria que indica si la mano presente en la imagen es la derecha o la izquierda. En la parte izquierda de la imagen tenemos la relación en colores con respecto al conjunto de datos usado en el entrenamiento para obtener las respectivas salidas con el extractor de características [31].

Figura 11. Arquitectura del modelo detección de puntos claves de la mano.



Fuente: Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C., & Grundmann, M. (2020). MediaPipe Hands: On-device Real-time Hand Tracking.

Figura 12. Puntos de referencia de la mano



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fuente: Google. Mediapipe Hands. (Fecha de acceso: 29/01/2023). Recuperado de https://google.github.io/mediapipe/solutions/hands

Con respecto al **conjunto de datos usado en el entrenamiento** de los modelos se destaca lo siguiente:

- Para el detector de palmas, se usó el conjunto de datos llamado ”In the wild” que contiene 6.000 imágenes con gran variedad (diversidad geográfica, distintas condiciones de iluminación y distintas apariencias de mano), con la única limitante de que este no contiene poses complejas [31].

- Con respecto al detector de puntos claves, se usó el conjunto de datos mencionado anteriormente ”In the wild”, el conjunto de datos que llaman ”In house collected gesture” el cual está compuesto de 10.000 imágenes tomadas con varios ángulos de todos los posibles gestos de la mano, destacando que la desventaja de este es fue recolectado de solo 30 personas con poca variación o poco ruido de fondo y finalmente se usó también el conjunto de datos llamado ”Synthetic”, como su nombre lo indica es sintético, debido a que este fue tomado con ayuda de un modelador de manos 3D, lo que permitió obtener 100.000 imágenes entre las que se incluye una gran variedad de gestos, distintos ángulos de cámara y distintas condiciones de iluminación [31].

- Para la tarea de determinar si hay una mano presente o no, se tomó un sub-

conjunto de imágenes del mundo real que se utiliza como ejemplos positivos y se muestrea la región, excluyendo las regiones de la mano anotadas como ejemplos negativos [31].

- Para la tarea de determinar cual mano es derecha o cual es izquierda, se tomó un subconjunto de imágenes del mundo real con las respectivas anotaciones de cual es izquierda y cual derecha [31].

En cuanto a los parámetros configurables en el modelo tenemos:

- **STATIC_MODE_IMAGE**, el cual está encargado de definir el tipo de entrada al modelo: "True", si se quiere evaluar imágenes estáticas y "False"si la entrada es una transmisión de video.
- **MAX_NUM_HANDS**, el cual está encargado de definir el número máximo de manos a reconocer.
- **MODEL_COMPLEXITY**, el cual está encargado de definir la complejidad del modelo (0 o 1), hay que tener en cuenta que la precisión y la latencia de inferencia del modelo aumentan con respecto a la complejidad del mismo.
- **MIN_DETECTION_CONFIDENCE**, el cual está encargado de definir el valor mínimo para que la predicción del modelo encargado de reconocer la mano sea considera exitosa, varía entre [0.0 , 1.0].
- **MIN_TRACKING_CONFIDENCE**,el cual está encargado de definir el valor mínimo para que la predicción del modelo encargado de reconocer los puntos de referencia en la mano, varía entre [0.0 , 1.0].

A la salida tenemos:

- **MULTI_HAND_LANDMARKS**, contiene la información de los 21 puntos de referencia de la mano con sus coordenadas X,Y y Z.

- **MULTI_HAND_WORLD_LANDMARKS**, contiene la información de los 21 puntos de referencia de las manos detectadas X,Y y Z (con respecto al máximo número de manos en pantalla).
- **MULTI_HANDEDNESS**, contiene la información de si la mano en pantalla es izquierda o derecha.

3. DESARROLLO DEL PROYECTO

3.1. ALGORITMO DE DESPLIEGUE.

Para el proceso de despliegue se realizó una aplicación que consta de una interfaz para el usuario y un código. El código cuenta con los modelos de seguimiento de puntos clave y de reconocimiento de gestos. Para realizar la acción de mouse y activación de macros se adaptaron dichos modelos con el fin de crear dos modos de operación para el sistema.

3.1.1. RECONOCIMIENTO DE GESTOS.

Base de datos. El conjunto de datos en el proceso de entrenamiento de una red neuronal es en la mayoría de los casos quien determina el máximo desempeño que tendrá este algoritmo en su tarea de clasificación. El banco de imágenes RGB usado para el entrenamiento de la red convolucional de detección de gestos se obtuvo de Kaggle, una plataforma en línea para competiciones de aprendizaje automático, el conjunto de datos se encuentra en el Anexo 2. Este conjunto, dado que cuenta con una licencia CC BY-SA 4.0. Permite compartir, modificar y adaptar el contenido siempre y cuando se respeten las condiciones de atribución y licenciamiento.

Procesamiento de imágenes. Para adaptar el conjunto de datos de los gestos, se llevaron a cabo algunos procesos para garantizar su compatibilidad con las diversas arquitecturas probadas. Primero, se redujo significativamente el conjunto de datos para poder manipularlo de manera óptima. En total, se trabajó con aproximadamente 500 fotografías por cada clase para los modelos ResNet50 y MobileNetV1, por otro lado, para el modelo entrenado desde cero, el conjunto de datos fue de 4000

imágenes por cada clase. Luego, se realizó un redimensionamiento a (224,224,3), la dimensión estándar para ResNet50 y MobileNetV1.

Red neuronal implementada para clasificación de gestos. Para el proceso de clasificación de gestos es necesario que el modelo tenga una rápida inferencia y una precisión alta para cumplir con su funcionalidad. Con el fin de encontrar la opción que más se ajuste a estas características se realizaron pruebas con un modelo entrenado desde cero y tres modelos entrenados con la metodología de transferencia de aprendizaje ResNet50, MobileNetV1 y YOLOv8, donde se toma como base la arquitectura de dichos modelos preentrenados en un conjunto de datos extenso para clasificar un nuevo conjunto de datos. En la implementación del modelo entrenado desde cero se tomó una base de datos con 32.000 muestras para un total de 8 clases, de ellas se tomó el 80 % para entrenamiento y el 20 % para validación, luego de dividir el conjunto de datos se realizó un preprocesamiento de los datos con la herramienta de keras imagedatagenerator donde se definieron los siguientes parámetros:

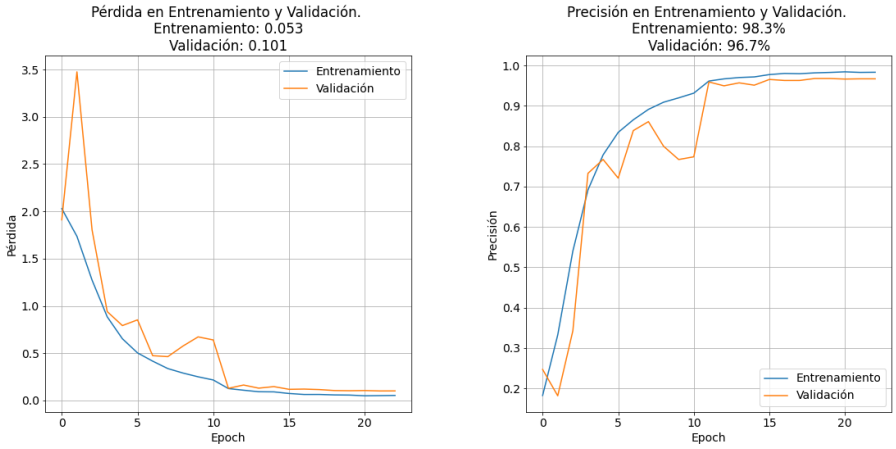
Tabla 2. Parámetros para el aumento de datos.

	Operaciones
Aumento de datos	Rango de rotación: 25 Rango de zoom: 0.1 Inversión horizontal: True Modo de llenado: constant

La arquitectura del modelo está basada en VGG16, ya que es una arquitectura fácil de implementar y con buenos resultados en competiciones para la tarea de clasificación, se realizaron algunos ajustes con el fin de normalizar los datos y se añadieron algunas técnicas de regularización como el dropout (luego de cada bloque de capas convolucionales) y la inicialización de pesos de acuerdo a la función de activación de la capa, el código se encuentra en el siguiente enlace(código). Se escogió el op-

timizador RMSProp a diferencia de los modelos con la metodología de transferencia de aprendizaje, puesto que este optimizador aunque puede tener ligeras desventajas a comparación del optimizador Adam agiliza el proceso de entrenamiento, algo crucial para la corrección y el análisis de los hiperparámetros de la red. También se aplicaron técnicas de regularización como la parada temprana que permite detener el entrenamiento en caso de no superar cierta condición (la condición se estableció en incremento menor a 0.005 en la precisión de validación por 7 épocas el modelo se detiene y guarda el valor de los pesos para la mejor época) y la herramienta de reducción de la tasa de aprendizaje que permite al modelo acercarse un poco más al mínimo de la función de perdida, este ultimo con una paciencia de 3 un delta de 0.25. Los resultados obtenidos para este modelo son los siguientes:

Figura 13. Modelo personalizado



Como se puede observar en la Figura 13, el modelo claramente se sobreajusta al conjunto de datos esto se observa al obtener la precisión de la red muy alta y alcanzar un error tan bajo en la pérdida, estos resultados impiden que el modelo generalice correctamente, el fenómeno del sobreajuste se observó con mayor facilidad al realizar predicciones sobre un conjunto de imágenes propias en donde se obtuvieron resultados muy alejados a los obtenidos en el entrenamiento y validación

del modelo. El sobreajuste pudo ser causado por la naturaleza de la base de datos, en el cual se pueden encontrar diversas fuentes de ruido que el modelo pudo haber memorizado. Por otro lado, al evaluar su velocidad (con un código de python) para generar nuevas predicciones se observó que era alrededor de 780 [ms], un tiempo extremadamente alto para el objetivo que se buscaba cumplir, por todo lo mencionado anteriormente se buscaron alternativas para la clasificación de los gestos.

Para realizar el proceso de transferencia de aprendizaje el conjunto de datos se redujo, puesto que solo se quiere reentrenar las ultimas capas de cada modelo, se trabajó con un total de 4000 imágenes donde el 80 % se tomó para entrenamiento y 20 % para validación (se realizó el mismo preprocesamiento que para el modelo entrenado desde cero) base de datos. En ambos casos se realizó el descongelamiento de algunas de sus últimas capas con el fin realizar un ajuste más fino que pueda reflejar un mejor desempeño.

El optimizador seleccionado para los dos modelos fue Adam, por último para realizar el entrenamiento se usaron la parada temprana y la reducción de la taza de aprendizaje para ambos casos la condición se estableció en incremento menor a 0.005 en la precisión de validación por 7 épocas para MobileNetV1 y 10 épocas para ResNet50 el modelo se detiene y guarda el valor de los pesos para la mejor época y para la reducción de la tasa de aprendizaje una paciencia de 3 con un delta de 0.25.

Los resultados obtenidos para estos modelos son los siguientes:

Figura 14. Modelo MobileNetV1

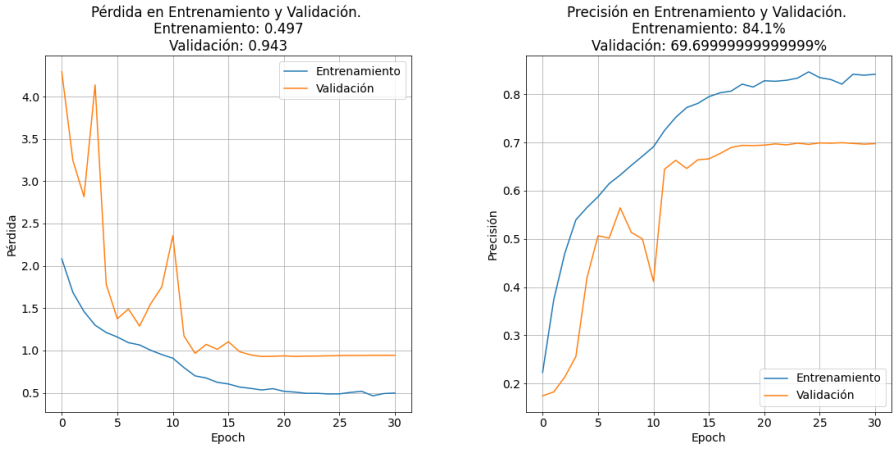
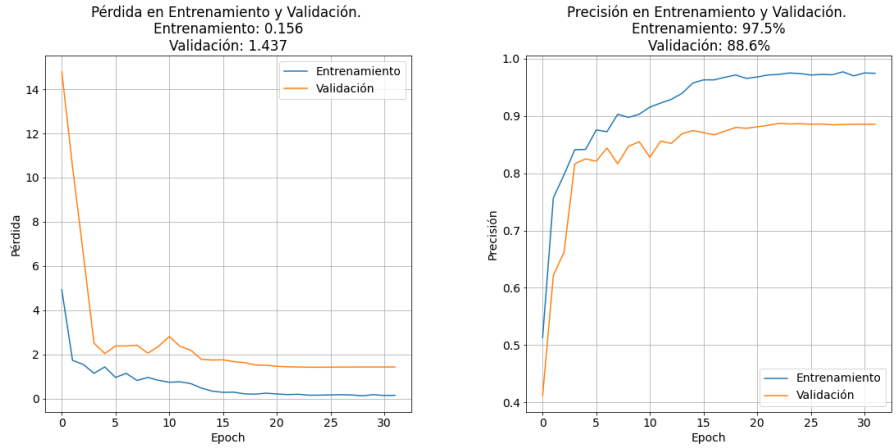


Figura 15. Modelo ResNet50



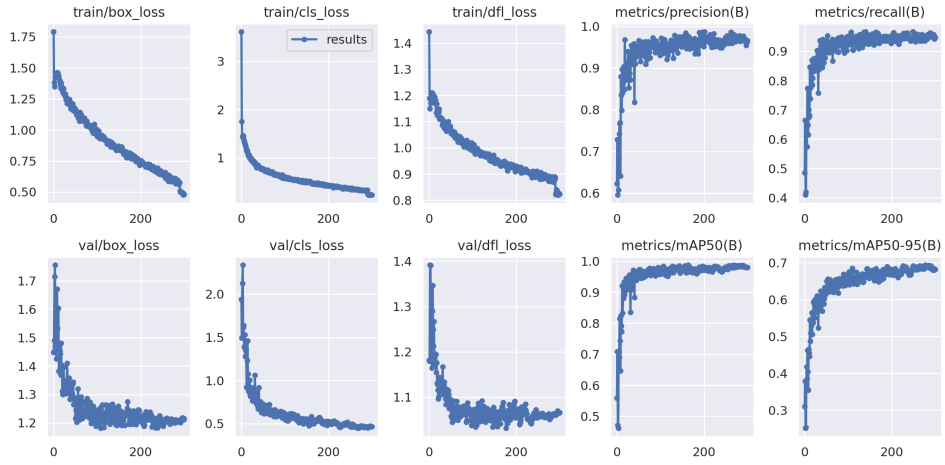
Luego de observar el desempeño de ResNet50 y el de MobileNetV1, se descartó esta última pues no lograba una precisión aceptable para el objetivo a cumplir. Por otro lado, ResNet50 obtuvo una precisión aceptable, pero al momento de evaluar su tiempo de inferencia (con un código de Python) no se encontró una mejora significativa con respecto al modelo entrenado desde cero por lo tanto se optó por cambiar el enfoque de la red, pasar de solo clasificación a detección, permitiendo una respuesta más confiable al clasificar únicamente regiones de interés especificas.

Para realizar la transición del modelo clasificador a detector, se tomó como base la red YOLOv8m, para ello se realizó un conjunto de datos personalizado, puesto que la para la tarea de detección es necesario contar con un etiquetado distinto al que se tenía del primer conjunto de datos, este proceso de generar el nuevo conjunto de datos se realizó con ayuda de un código de Python para tomar las fotografías y para el etiquetado se usó la herramienta labelImg. El nuevo conjunto de datos contó con 130 fotografías de cada clase donde el 80 % de ellas se usaron para entrenamiento y el 20 % para validación.

Para realizar el proceso de entrenamiento se escogió el optimizador SGD con un tasa de aprendizaje de 0.01, para la detención temprana se escogió una paciencia de 50 y un weight decay de 0.0005 (código).

Los resultados obtenidos se observan en las siguientes gráficas:

Figura 16. Resultados de Yolo



Por ultimo, se pudo observar que el rendimiento del modelo con arquitectura YOLO logra un desempeño favorable en métricas como mAP50 y mAP50-95, a su vez, este modelo obtuvo un tiempo de inferencia de aproximadamente 180[ms], consiguiendo de esta manera ser el modelo con mejores resultados tanto en precisión como en velocidad de inferencia, por ello fue seleccionado para ser implementado en el

proyecto.
Todos los modelos utilizados para la comparación de rendimiento se encuentran en el repositorio de GitHub del proyecto (ver Anexo 2).

Acondicionamiento del modelo entrenado para realizar macros. Posterior al entrenamiento se realizó un código de Python donde se ajustaron las salidas del modelo de reconocimientos de gestos para su uso como activador de macros.
Primero se definieron las acciones a realizar por cada gesto. Luego de definir condiciones de operación del sistema se creó un proyecto en el entorno de trabajo Pycharm, en donde se realizó la implementación tanto de la interfaz como el despliegue del modelo entrenado. Con ayuda de la librería de Ultralytics se cargó el modelo al entorno de Python para extraer sus parámetros de entrada, luego de corroborar las dimensiones que debía llevar la imagen para realizar su inferencia, se usó la librería de OpenCV para redimensionar los fotogramas de la captura de video a (225,225,3).
En el proceso de inferencia se configuró el modelo para que detectara únicamente las predicciones superiores al 60 % de confianza. Para realizar la asignación de macros se usó la librería pynput, esta librería permite emular el presionar teclas o combinaciones del teclado, con ayuda de esta biblioteca, se convirtió cada gesto detectado en un accionamiento de tecla distinto.
Al realizar la asignación de comandos específicos para cada gesto se encontró otra falla en la activación de la función, puesto que el modelo apenas detectaba algún gesto inmediatamente ejecutaba la función correspondiente. Dicho problema podía desencadenar un accionamiento múltiple y sin control de la misma acción, en ocasiones entorpeciendo el sistema operativo. Para resolver el problema de activaciones múltiples, se realizó un muestreo de las inferencias. Este muestreo se basa en una lista que se va llenando con las predicciones del modelo hasta completar 6 inferencias, luego de llegar a la inferencia 6 se determina cual es la inferencia que

más se repitió dentro de la lista y lo toma como el gesto correcto para accionar su función asignada.

Luego de ajustar en el código los parámetros mencionados anteriormente, el proceso de reconocimiento de gesto consiguió un buen desempeño. Por otro lado, pensando en el propósito de la APP para realizar presentaciones en donde el usuario puede realizar diversas gesticulaciones con las manos y accionar por error algunas funciones, se introdujo una nueva característica al sistema. Esta permite encender o apagar el proceso de ejecución de macros, es decir, si el sistema se apaga, no realizará ninguna función hasta que se vuelva a encender, esta operación de interruptor es accionada por un gesto.

Por último, se le agregaron algunos detalles que permiten al usuario detectar cuando el sistema reconoce un gesto y ejecuta una acción. Con la librería Winsound de Python se configuró un sonido para que se reproduzca al ejecutarse una acción por parte del código, adicional a ello la ventana de captura de video parpadea por un instante también con el fin de informar al usuario que se ejecutó una acción.

3.1.2. MODO MOUSE. El segundo modo de operación es el encargado de recrear todo el mouse en sí, por lo que para obtener un movimiento fluido del cursor y realizar las operaciones posibles del mismo de manera rápida. Se eligió un modelo ya entrenado del repositorio de Mediapipe llamado "Mediapipe Hands", ya que este ofrece una rápida inferencia, lista para usar, gratis y de código abierto.

La idea general fue tener 2 códigos de Python. Uno encargado de contener las funciones necesarias para realizar el seguimiento de los puntos de la mano y otras funciones que ayuden con la tarea de identificar gestos. Y el otro código que sea el encargado de ejecutar dichas funciones mediante condiciones de programación, que sean equivalentes a gestos de la mano.

Comenzando el primer código de Python llamado MouseTrackingMP.py encargado de las funciones, se crea una clase que permite instanciar el modelo desde el segun-

do código encargado de la ejecución del mismo, dicha clase contiene las siguientes funciones:

- **Init**: Función encargada de inicializar los parámetros predeterminados que tendrá el modelo (como los descritos en las páginas 39 y 39 del libro) y definir las funciones básicas del modelo de ”Mediapipe Hands” que luego serán usadas en las demás funciones.

- **FindHands**: Función encargada de encontrar las manos mediante funciones contenidas en la librería de mediapipe, y que como parámetro de entrada tiene la transmisión de video obtenido mediante la librería OpenCV. Esta retorna la captura de video con líneas dibujadas entre los 21 puntos de la mano mediante la función **draw_handmarks**.

- **FindPosition**: Función encargada de encontrar la posición de cada uno de los respectivos landmarks, como parámetro de entrada tiene la captura video y de salida retorna un arreglo de que guarda la posición de cada uno de los 21 puntos de la mano.

- **UpFingerDetector**: Función que asigna un valor 0 o 1 en caso de que el dedo se encuentre en bajo o en alto respectivamente. Esto se hace para identificar gestos mediante variaciones en los 2 estados posibles de cada dedo. Así como dibujar cuando estos se encuentran en estado alto.

- **DistanceCalculator**: Función encargada de calcular la distancia entre los dedos, en base a la distancia, la intención es recrear los clicks del mouse (izquierdo y derecho). A su vez dibuja una línea entre los puntos de interés para la acción.

- **Main**: Función que se ejecuta siempre que se corra únicamente el código con fines de prueba. Esta se refiere con respecto a la clase. Dentro de ella se

instancia la misma para observar un pequeño ejemplo de cómo funciona.

Pasando al segundo código de Python llamado VirtualMouse.py. se define como función también, debido a que se piensa incluir dentro de otro archivo .py encargado de ejecutar la interfaz y de diseñar una lógica general de funcionamiento. Dentro de él se encuentra lo siguiente: Primero se definen las dimensiones de la ventana de OpenCV y el rango de interacción de la mano en la pantalla. Es necesario definir un rango de interacción debido a que la cámara tiene un límite de visión, si el rango de interacción de la mano no es menor que los límites de la cámara, sucederá que no es posible llegar al extremo de la pantalla con el cursor del mouse.
Luego mediante la librería pyautogui, se capturan las dimensiones de nuestra pantalla que luego serán usadas para hacer una conversión con el límite de movimiento y la pantalla misma, se define también un suavizado para el mouse (este nos permitirá un movimiento fluido del cursor).
Con ayuda de la librería pynput, se importan los controladores del mouse y teclado, esta librería nos permite realizar acciones de mouse y teclado en el ordenador mediante funciones que incluye la misma.
Se instancia la función del primer código encargada de inicializar el modelo Mediapipe Hands, estableciendo que el máximo número de manos en pantalla a reconocer será 1.
Es aquí cuando se inicia el bucle que llevará condiciones que tienen por argumento los estados alto o bajo de cada dedo de la mano. Mediante estas condiciones se generan los respectivos gestos:

- **MOVIMIENTO DEL CURSOR**: Cuando se cumpla la condición de que el índice se encuentra en estado alto y los demás dedos en estado bajo, entra en la condición del gesto MOVIMIENTO DEL CURSOR, el cual dentro realiza la conversión del cuadrado creado visualmente con la librería OpenCV con las dimensiones de la pantalla, para así no presentar pérdida de visión de la mano

que resulte en la incapacidad de llegar a los bordes de la pantalla misma. Adicionalmente dentro se realiza un suavizado a la posición del cursor, de manera que esté presente un movimiento fluido y cómodo al usuario. Finalmente, la posición de la punta del dedo índice es luego intercambiada con la posición del mouse con ayuda de la función **position** de la librería **pynput**.

- **CLICK IZQUIERDO**: Cuando se cumpla la condición de que el índice y el pulgar se encuentren en estado alto (los demás en bajo), se detendrá el movimiento del cursor y se activará el cálculo de la distancia entre los puntos 5 y 3 (Ver Figura 12). Finalmente la acción de click izquierdo se realiza mediante la función **click(Button.left, 1)** (Traduce click izquierdo pulsado 1 vez) de la librería pynput.

- **CLICK DERECHO**: Cuando se cumpla la condición de que el índice y el corazón (o dedo medio) se encuentren en alto (los demás en bajo), se detendrá el movimiento del cursor y se activará el cálculo de la distancia entre los puntos 8 y 12 (Ver Figura 12). Finalmente la acción de click derecho se realiza mediante la función **click(Button.right, 1)** (Traduce click derecho pulsado 1 vez) de la librería pynput.

- **DESPLAZAMIENTO HACIA ARRIBA**: Cuando se cumpla la condición de que el meñique se encuentra en alto y los demás en bajo, se detendrá el movimiento del cursor y se realizará la acción de desplazarse hacia arriba, esta se realiza mediante la función **scroll(0, 2)** (Traduce mover en sentido vertical y 2 casillas hacia arriba debido a que es un número positivo) de la librería pynput.

- **DESPLAZAMIENTO HACIA ABAJO**: Cuando se cumpla la condición de que el pulgar se encuentra en alto y los demás en bajo, se detendrá el movimiento del cursor y se realizará la acción de desplazarse hacia abajo, esta se realiza mediante la función **scroll(0, -2)** (Traduce mover en sentido vertical y 2 casillas

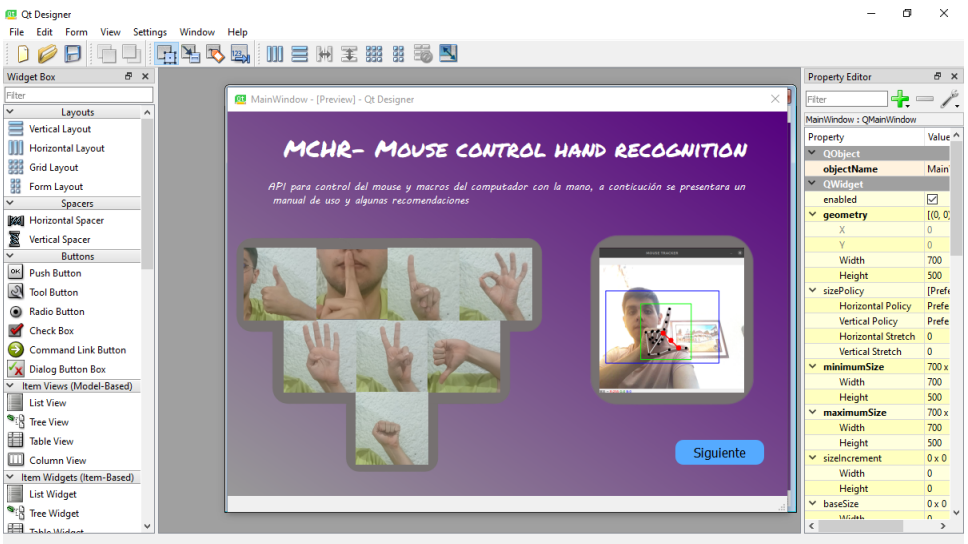
hacia abajo debido a que es un número negativo) de la librería pynput.

- **DETENER APLICACIÓN**: Cuando se cumpla la condición de que los dedos pulgar, índice y meñique se encuentran en estado alto (los demás en bajo), se detendrá la aplicación, esto mediante una bandera que luego en el código general será tomada en cuenta para detener el proceso.
- **CAMBIAR A MODO RECONOCIMIENTO DE GESTOS**: Cuando se cumpla la condición de que todos los dedos se encuentran en estado alto, se cambiará a modo reconocimiento de gestos, esto mediante una bandera que luego en el código general será tomada en cuenta para cambiar a modo reconocimiento de gestos.

3.2. INTERFAZ DE USUARIO.

Para el desarrollo de la interfaz de usuario se utilizó la aplicación Qt Designer, la cual permite crear ventanas desplegables con la posibilidad de navegar entre ellas.

Figura 17. Herramienta Qt para el desarrollo de la interfaz



En la interfaz de usuario se incluyó un breve resumen de las funcionalidades asignadas para cada gesto, así como recomendaciones para su uso. En total se crearon tres ventanas en donde la primera de ellas corresponde a la pantalla de carga al iniciar la API, la segunda presenta una breve descripción de la APP y en la tercera se encuentran todos los gestos configurados.

Para indicar si el sistema de reconocimiento está en ejecución, en la tercena ventana se presenta un mensaje donde indica el estado del proceso: en ejecución o detenido.

Para la ejecución de la interfaz se creó un archivo llamado **MCHR.bat**, este contiene las instrucciones a ejecutar en el símbolo del sistema de modo que el usuario o docente no tiene que realizar una instancia de manera manual en cmd, basta con hacer doble click y automáticamente se ejecutará todo.

Pensando en instalar las dependencias de manera automática estas se alojan en un archivo llamado requirements.txt, en el se encuentran cada una de las dependencias con sus respectivas versiones que concuerdan con las usadas para el diseño del proyecto, sin embargo, para instalarlas existe un archivo llamado **Requirements_Setup.bat**, basta con hacer doble click y automáticamente se realizará la instalación de las dependencias.

4. RESULTADOS.

En este capitulo se realizará un resumen de los resultados obtenidos a lo largo del desarrollo del proyecto, tanto los entrenamientos preliminares de las redes neuronales, como el flujo de trabajo en el cual se basa la aplicación para su funcionamiento.

4.1. Modelos para el reconocimiento de gestos.

Para la tarea de reconocimiento, primero se optó por la metodología de clasificación, en donde la red tenia que analizar por completo la imagen para realizar una predicción, para dicha tarea se usaron 3 arquitecturas, una personalizada y dos arquitecturas con la metodología de transferencia de aprendizaje, ResNet50 y MobileNetV1.

Para realizar esta comparativa, se analizaron métricas de rendimiento de cada modelo, precisión y la función de perdida. En la siguiente tabla se puede observar el desempeño de cada uno de los modelos:

Tabla 3. Resultados de entrenamiento para clasificación.

Modelo	**Precisión en validación(%)**	**Perdida en validación**
Custom	96.7	0.1
MobileNetV1	69.69	0.943
ResNet50	88.6	1.437

Como se puede observar en la tabla 3 los mejores modelos fueron ResNet50 y el modelo personalizado, el siguiente paso en la tarea del clasificador era definir el rendimiento de dichos modelos en términos de tiempos de inferencia, para realizar este proceso se realizó una modificación al código de python de clasificación para tomar el tiempo que le tomaba realizar una predicción. El tiempo obtenido para el modelo ResNet50 fue de 270[ms] y para el modelo personalizado aproximadamente

780[ms]. Con base en los tiempos obtenidos se encontró que el modelo personalizado era inviable para el objetivo del proyecto, por ende, el modelo seleccionado fue ResNet50.

Luego de realizar pruebas del modelo con captura de video, se encontraron algunos inconvenientes en cuanto al ruido en la imagen y la iluminación, puesto que dichos parámetros debían ser casi ideales para realizar predicciones confiables. Por otro lado, la distancia a la que tenia que estar el usuario del lente de la cámara era muy reducida, aproximadamente de 60cm. Para solucionar los inconvenientes ocasionados por el ruido del fondo, se cambió la idea de un modelo de clasificación por uno de detección, esto con el fin de eliminar los componentes de ruido en la imagen y ejecutar un clasificador enfocado a las regiones de interés.

Para realizar la detección de los gestos se usó el modelo YOLOv8m. En consecuencia al cambio de clasificación por detección, se realizó un conjunto de datos propio con las características necesarias para el entrenamiento del modelo detector. El conjunto de datos cuenta con 1040 imágenes para un total de 8 clases, este se dividió el 80 % para entrenamiento y el 20 % para validación.

Evaluando el rendimiento del nuevo modelo para la detección de gestos se obtuvieron los siguientes resultados:

Tabla 4. Resultados de entrenamiento para detección.

Modelo	**mAP50(%)**	**mAP50-95(%)**	**Tiempo de inferencia**
YOLOv8m	98	67	180[ms]

Como se puede observar en la tabla 4, el modelo logra alcanzar un desempeño óptimo, puesto que cuenta con un mAP50 alto y un mAP50-95 bueno, este concepto de mAP se basa en la precisión promedio de las detecciones correctas según un umbral de confianza en especifico, si observamos el mAP50 se logra apreciar que aproximadamente hay una precisión del 98 % en las detecciones realizadas con un

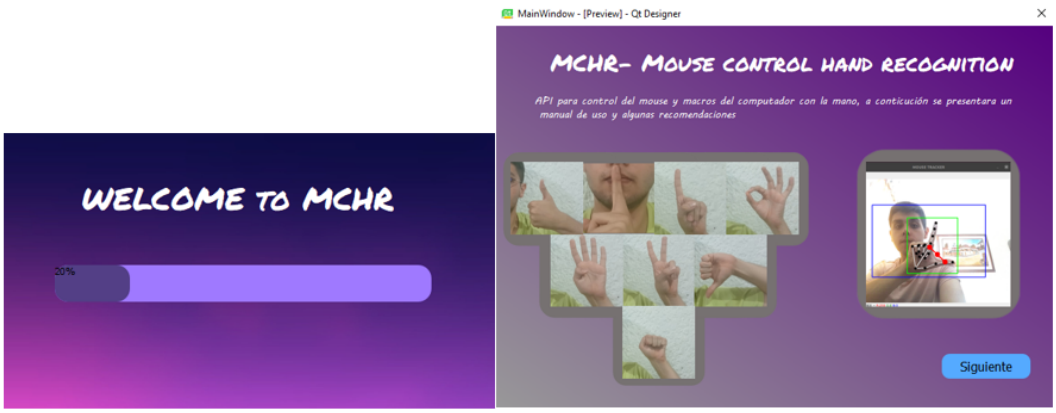
umbral de confianza del 50 %. Por otro lado el tiempo de inferencia mejoró sustancialmente a comparación del mejor modelo de clasificación ResNet50, y en cuanto a pruebas del modelo en captura de video, este logró conseguir un buen desempeño en entornos variados, por lo cual que fue seleccionado para ser implementado en la aplicación.

4.2. Flujo de trabajo de la aplicación.

El funcionamiento de la aplicación MCHR se dividió en dos procesos fundamentales. Por un lado el modelo de ”Esqueletización de la mano”, el cual está encargado de realizar el seguimiento de los puntos claves de la mano que luego son usados para actualizar la posición del cursor y definir los demás botones básicos del mouse mediante gestos definidos con programación. Y por otra parte está el modelo de ”Reconocimiento de gestos”, que mediante detección de gestos acciona teclas o comandos orientados a la presentación o exposición de diapositivas. Su funcionamiento conjunto se detalla en el Anexo 4.

4.3. Entregables.

Figura 18. API creada



En la Figura 18 se observa la aplicación desarrollada. El manual de usuario se encuentra en el Anexo 1, la guía de instalación se puede encontrar en el repositorio de GitHub en el Anexo 2, en compañía de un video demostrativo con la API en funcionamiento. Si bien la precisión con que detecta los gestos en ocasiones fluctúa por el ambiente de fondo, la aplicación logra detectar y accionar las funcionalidades en un rango aceptable de tiempo.

5. RECOMENDACIONES

- Para trabajos futuros que quieran continuar con el desarrollo de este proyecto, se recomienda abordar la posibilidad de reconfigurar los macros del teclado que serán usados para realizar atajos desde la interfaz, esto con el fin de hacer compatible el software con diferentes aplicaciones.

- Para evitar problemas a la hora de realizar un proyecto relacionado con el desarrollo de software es recomendable trabajar en un entorno virtual donde se puedan ajustar todas las dependencias de manera organizada para evitar conflictos entre las versiones de los programas con los que se va a trabajar.

- Para desarrollos futuros en base a este proyecto se recomienda incluir una funcionalidad para seleccionar la cámara principal en caso de que el usuario cuente con múltiples cámaras.

- Google Colab es una excelente herramienta para el entrenamiento de algoritmos de aprendizaje profundo, en caso de estar comenzando en este campo, su versión gratuita es más que suficiente para entrenar modelos sencillos que no excedan un máximo de tiempo de entrenamiento aproximado de 4 Horas. En caso de querer continuar en el campo, realizando modelos mucho más robustos, Colab ofrece una versión paga (Google Colab Pro). Sin embargo, debido a su distribución por unidades de cómputo limitadas puede resultar en un alto costo, por lo que en ocasiones es preferible invertir en hardware propio.

6. CONCLUSIONES

- En el proceso de desarrollo de la aplicación se logró cumplir el objetivo de utilizar los gestos manuales para el control del mouse y macros del ordenador. Con el fin de que se siga fortaleciendo el proceso de enseñanza con la incorporación de visión por computadora en las aulas híbridas, se comparten los resultados obtenidos y la API desarrollada (Anexo 2) .

- Con los resultados obtenidos, se comprobó que la cámara RGB puede resolver la problemática de reconocer en tiempo real los gestos de la mano. Aún con ruido en la imagen, ésta sigue siendo una de las mejores opciones para la implementación de sistemas de bajo costo y relacionados con el reconocimiento de gestos.

- Para la tarea de clasificación es imprescindible identificar el tamaño óptimo al cual se debe reducir el mapa de características para pasar a las capas densamente conectadas, ya que un mapa de características muy grande puede desencadenar un sobreajuste en donde el modelo aprenderá componentes de ruido irrelevantes.

- Para casos en los que se desee clasificar una parte especifica y no toda una imagen es recomendable optar por la metodología basada en detección de objetos, ya que permite eliminar la mayor cantidad de ruido de la muestra accionando un clasificador específicamente para una región de interés.

BIBLIOGRAFÍA

En: *2015 Conferencia Internacional IEEE sobre Visión por Computador (ICCV)*, págs. 1440-1448. DOI: 10.1109/ICCV.2015.169 (vid. pág. 28).

Aggarwal, Charu C. *Neural Networks and Deep Learning. A Textbook*. Cham: Springer, 2018, págs. 15-16. DOI: 10.1007/978-3-319-94463-0 (vid. pág. 24).

Arriaga, Octavio et al. *Perception for Autonomous Systems (PAZ)*. 2020. arXiv: 2010.14541 [cs.CV] (vid. págs. 29, 32).

Arriaga, Octavio et al. "Perception for Autonomous Systems (PAZ)". En: *CoRR* abs/2010.14541 (2020), págs. 1-2. arXiv: 2010.14541 (vid. pág. 32).

Bazarevsky, Valentin et al. *BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs*. 2019. arXiv: 1907.05047 [cs.CV] (vid. pág. 35).

Cheng, Jian et al. "Efficient Virtual View Selection for 3D Hand Pose Estimation". En: 2022, págs. 5-7 (vid. pág. 29).

Chollet, F. *Deep Learning with Python, Second Edition*. Manning, 2021, págs. 1-5 (vid. pág. 16).

— *Deep Learning with Python, Second Edition*. Manning, 2021, págs. 204-206 (vid. pág. 26).

Doersch, Carl. *Tutorial on Variational Autoencoders*. 2021. arXiv: 1606.05908 [stat.ML] (vid. pág. 19).

Doosti, Bardia. “Hand Pose Estimation: A Survey”. En: *CoRR* abs/1903.01013 (2019), pág. 3. arXiv: 1903.01013 (vid. pág. 30).

Ge, Liuhao et al. “3D Hand Shape and Pose Estimation From a Single RGB Image”. En: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, págs. 10825-10834. DOI: 10.1109/CVPR.2019.01109 (vid. pág. 29).

Glorot, Xavier, Antoine Bordes y Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. en. En: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop y Conference Proceedings, jun. de 2011, pág. 318 (vid. pág. 24).

— “Deep Sparse Rectifier Neural Networks”. En: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Geoffrey Gordon, David Dunson y Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pág. 318 (vid. pág. 25).

Goodfellow, Ian, Yoshua Bengio y Aaron Courville. *Deep learning*. MIT press, 2016, pág. 197 (vid. pág. 22).

— *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, pág. 193 (vid. pág. 25).

Goodfellow, Ian et al. “Generative Adversarial Nets”. En: *Advances in Neural Information Processing Systems*. Ed. por Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014 (vid. pág. 18).

Google. *MediaPipe Hands*. 2019 (vid. pág. 29).

Guío, A. Uribe y Pablo E. Ayerbe. *MARCO ÉTICO PARA LA INTELIGENCIA ARTIFICIAL EN COLOMBIA*. 1ra. Bogotá: Gobierno de Colombia, 2021 (vid. pág. 14).

Hu, Zhihua y Xiaoming Zhu. "Gesture detection from RGB hand image using modified convolutional neural network". En: *2019 2nd International Conference on Information Systems and Computer Aided Education (ICISCAE)*. 2019, págs. 143-146. DOI: 10.1109/ICISCAE48440.2019.221606 (vid. pág. 15).

"Jerarquías de características enriquecidas para la detección precisa de objetos y la segmentación semántica". En: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2014.81 (vid. pág. 28).

Jocher, Glenn, Ayush Chaurasia y Jing Qiu. *YOLO by Ultralytics*. Ver. 8.0.0. Ene. de 2023 (vid. pág. 28).

Karpathy, Andrej, Justin Johnson y Li Fei-Fei. *Visualizing and Understanding Recurrent Networks*. 2015. arXiv: 1506.02078 [cs.LG] (vid. pág. 19).

Lin, Tsung-Yi et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV] (vid. pág. 35).

Lin, Tsung-Yi et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV] (vid. pág. 36).

Liu, Wei et al. "SSD: Single Shot MultiBox Detector". En: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, págs. 21-37. DOI: 10.1007/978-3-319-46448-0_2 (vid. pág. 35).

Lugaresi, Camillo et al. “MediaPipe: A Framework for Building Perception Pipelines”. En: *CoRR* abs/1906.08172 (2019), págs. 1-2. arXiv: 1906.08172 (vid. págs. 33, 34).

“MaHG-RGBD: un conjunto de datos RGB-D de gestos manuales con vista multiángulo para reconocimiento de gestos basado en aprendizaje profundo y evaluaciones de referencia”. En: *2019 Conferencia internacional IEEE sobre electrónica de consumo (ICCE)*. DOI: 10.1109/ICCE.2019.8661941 (vid. pág. 14).

“Introduction”. En: *Neural Networks: Tricks of the Trade*. Ed. por Genevieve B. Orr y Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, págs. 1-2. DOI: 10.1007/3-540-49430-8_1 (vid. pág. 20).

Rajan, Rajesh George y P. Selvi Rajendran. “Gesture recognition of RGB-D and RGB static images using ensemble-based CNN architecture”. En: *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. 2021, págs. 1579-1584. DOI: 10.1109/ICICCS51141.2021.9432163 (vid. págs. 14, 15).

“Reconocimiento robusto de la postura de la mano basado en imágenes RGBD”. En: *La 26.ª Conferencia China de Control y Decisión (2014 CCDC)*. DOI: 10.1109/CCDC.2014.6852636 (vid. pág. 14).

Redmon, Joseph et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV] (vid. pág. 28).

Russell, Stuart J. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010, págs. 1-4 (vid. pág. 17).

Sturman, D.J. y D. Zeltzer. “A survey of glove-based input”. En: *IEEE Computer Graphics and Applications* 14.1 (1994), págs. 30-39. DOI: 10.1109/38.250916 (vid. pág. 14).

Sun, Xiao et al. “Cascaded hand pose regression”. En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, págs. 824-832. DOI: 10.1109/CVPR.2015.7298683 (vid. pág. 29).

Zhang, Fan et al. “MediaPipe Hands: On-device Real-time Hand Tracking”. En: (2020), págs. 1-2. arXiv: 2006.10214 [cs.CV] (vid. págs. 34-39).

ANEXOS

Anexo A. Manual de uso de aplicación MCHR

Se realizó un manual de uso de MCHR con el fin de dar a conocer: ¿qué es MCHR?, los requisitos para su correcto funcionamiento, una guía de instalación, cómo ejecutar MCHR, las herramientas de la interfaz, las funciones de MCHR, una descripción de cada uno de los modos de operación y consejos para el uso de MCHR.

Anexo B. Repositorio de GitHub

Con el fin de complementar el desarrollo del proyecto se creó un repositorio en GitHub, el cual contiene: Códigos de los entrenamientos de cada modelo (Usados para el reconocimiento de gestos), guía de instalación, una breve descripción del funcionamiento de MCHR, conjuntos de datos usados para el entrenamiento de los modelos y un video en el que se evidencia el funcionamiento de la aplicación.

Anexo C. Arquitectura de YOLOv8

Para el desarrollo del presente proyecto se utilizó el modelo YOLO en su versión 8m, con el fin de brindar una vista más detallada a su arquitectura, se adjunta un diagrama de bloques que explica el flujo de la información dentro del modelo.

Anexo D. Diagrama de flujo MCHR

Se ha diseñado un diagrama de flujo para la aplicación con el objetivo de dar a entender el ciclo de trabajo de esta. El diagrama incluye cada modo de operación detallando algunos gestos significativos para la transición entre los posibles estados de la aplicación.