Universidad Autónoma del Estado de México

Centro Universitario UAEM Zumpango

TESIS

Juego de posturas

Presenta

Joel David González Del Ángel

Que para obtener el Título de

Ingeniero en Computación

Asesores

Dr. Rafael Rojas Hernández

Dr. Asdrúbal López Chau

Febrero 2025

Resumen

Hoy en día se encuentra el interés en la detección de los movimientos del cuerpo humano, de igual manera, en el uso de dispositivos electrónicos como diversión. En este trabajo se hace uso de la herramienta de programación llamada MediaPipe, este nos ayudará a obtener las poses del cuerpo humano para poder compararlos con otro cuerpo y determinar si las poses son similares.

El tema de esta tesis es crear un juego compatible con una computadora personal que disponga de una cámara web, en donde el juego consta de al menos 10 imágenes de personas con una pose definida, el cual el usuario tendrá que imitar para poder continuar el juego, esto sin importar el género, edad o contextura física de la persona.

Hipótesis

Aunque la mayoría de las personas creen que copiar una postura corporal a partir de una imagen es una tarea sencilla, la implementación de un sistema de visión artificial revela que la precisión en la imitación de la postura es significativamente más baja de lo esperado.

ÍNDICE GENERAL

Índice de figuras

1. Introducción

La visión artificial es un campo de la inteligencia artificial el cual permite que los sistemas tengan entradas como imágenes, videos u otros recursos digitales para poder tomar acciones a base de esa información procesada. Los sistemas de visión artificial son cada vez más populares en la parte de reconocimiento de objectos por lo que la hace aún más interesante el poder detectar poses de nuestro propio cuerpo.

Estas técnicas ya se han aplicado a diferentes ámbitos, en este caso en específico se utilizará para crear un juego, ya que las aplicaciones y juegos que existen son compatibles únicamente para consolas de videojuegos el cual es necesario de tener un complemento llamado Kinect, por lo que se decidió hacerlo en un ordenador para que más personas puedan disfrutar, aprender e interesarse en la visión artificial para el reconocimiento de poses de un cuerpo humano.

Así que se aplicarán los avances de la visión artificial y el reconocimiento de objetos, en la realización de una tesis en donde se le tendrá que pasar como entrada al sistema un conjunto de N imágenes el cual con ayuda de una herramienta enfocado especialmente a la detección de poses de cuerpos humanos, regresara como resultado N puntos clave con el cual se podrá realizar una comparación con otro resultado el cual saldrá de hacer el mismo proceso pero con una entrada de un video en tiempo real, para posteriormente compararlas y así el sistema podría tomar la decisión de si es correcta dicha pose o no.

1.1 Planteamiento Del Problema

Anteriormente la visión artificial clásica tenía problemas en la detección de múltiples objetos, actualmente esos problemas se han solucionado pero en consecuencia para poder detectar objectos se debe primero que entrenar un sistema con una gran cantidad de datos para obtener mejores resultados, por lo que se hace un proceso un tanto tedioso y difícil para una persona ajena al área de las ciencias de la computación, por lo que se han desarrollado herramientas el cual nos facilita el proceso de detección de objetos.

No existe un juego que sea compatible para computadoras, ya que los que se han centrado en desarrollar este tipo de juegos han sido personas y empresas que se centran en desarrollo de videojuegos para consolas y no para computadoras. Por lo que se busca realizar un juego que sea para toda la familia en donde se puedan detectar estas poses y compararlas con otras poses dadas y así verificar si es correcta y seguir avanzando dentro del juego, esto se hará teniendo controlada el área de la visión de la cámara, tratando de estar en un fondo claro, y con una única persona con ropa deportiva.

Hoy en día cualquier persona es capaz de interactuar con una computadora por lo que es interesante que pueda aprender más sobre el uso de la visión artificial y sus aplicaciones en la rama del entretenimiento, además de que nos ayudara a que las personas puedan estirar, mover y divertirse con este juego a desarrollar.

Este proyecto tiene como objetivo el explorar soluciones para poder desarrollar juegos para computadoras basados en la detección de objetos, específicamente en el reconocimiento de poses de una persona en un lugar controlado.

Para esto son necesarios los elementos básicos de un sistema de visión artificial que son los siguientes:

1. Cámara WEB que capture un video de la persona centrada en el área de visión de la Cámara.
2. Una computadora para la visualización de imágenes y video.
3. Un sistema capaz de procesar videos e imágenes y obtener las poses de un cuerpo humano en tiempo real.
4. Un sistema capaz de poder comparar 2 poses en tiempo real.
5. Lenguaje de programación compatible con el sistema anteriormente mencionado.

1.2 Justificación

La visión artificial tiene una gran importancia en diveros ámbitos, pero porque no mostrarles a las personas que también puede ser divertida y que mejor aún sin tener que gastar mucho dinero comprando consolas de videojuegos e incluso complementos para estos.

Se puede hacer desde cualquier computadora y cualquier persona que esté interesado en esto, así aprenden el funcionamiento de la visión artificial y al mismo tiempo que se diviertan.

Se ha optado por utilizar una computadora personal como plataforma principal, ya que hoy en día es de fácil acceso y además de que el modelo de visión artificial esta específicamente diseñado y optimizado para este entorno, así se ahorrarán el tener que hacer de cero el modelo, y se centrarán únicamente para adaptarlo a nuestras necesidades. Este modelo se utiliza junto con un lenguaje de programación llamado Python, el cual es muy intuitivo y fácil de aprender por lo que cualquier persona con poco conocimiento en el mundo de la programación puede adaptar y mejorar proyectos fácilmente.

El modelo que se utilizará es de Google del conjunto de herramientas de Mediapipe y específicamente el modelo “Pose Landmarker”, que es una red neuronal convolucional. Este modelo es el más apto para el proyecto, ya que reconoce 33 puntos de referencia del cuerpo humano, regresando una lista de coordenadas de cada uno de estos puntos. Además de que este modelo es de código abierto con licencia de Apache 2.0. Para poder visualizar las imágenes a comparar, se hará uso de una librería de Python llamada Opencv.

El objetivo es utilizar el resultado obtenido del modelo de visión artificial seleccionado para poder analizar y procesar estos datos, y poder enfocarse en validar, comparar y confirmar si existe una pose similar dentro de un conjunto de imágenes capturadas en un video. También con este enfoque del proyecto se

podrá poner a prueba la capacidad del modelo en imágenes en tiempo real, ya que este modelo se ejecuta cada vez que cambia de fotograma un video en tiempo real.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar una aplicación que utilice una computadora y la visión artificial para determinar si una persona está haciendo una pose del cuerpo que se le indica para ofrecer una nueva forma de diversión.

1.3.2 Objetivos Específicos

- Implementar un sistema capaz de detectar poses en tiempo real.
- Desarrollar un algoritmo capaz de comparar poses de 2 imágenes.
- Diseñar la forma de mostrar los resultados obtenidos de una comparación.
- Diseñar los niveles de un juego de poses de cuerpos.

1.4 Alcance

La tesis presentada se enfoca a los usuarios de computadoras, y a aquellas personas que quieren interactuar con la visión artificial y también puede llegar a tener alcance en un mayor porcentaje a la población infantil, ya que como será un juego de posturas tiende a llamar la atención de estos.

1.5 Limitaciones

La aplicación se desarrollará en un lenguaje de programación llamado Python, así que las librerías, plugin y complementos son compatibles con este mismo lenguaje, por lo que los usuarios deberán tener instalado en su computadora el intérprete de Python 3, las versiones específicas de cada una de las herramientas a utilizar.

Una de las más grandes limitaciones es la cámara de videograbación, ya que, al usar una computadora personal, estas tienen incluida una cámara de baja resolución, lo que puede llegar a provocar imágenes con baja resolución, distorsiones ópticas y problemas de enfoque lo que puede afectar directamente al modelo de visión artificial.

La iluminación es de igual forma muy importante a la hora de procesar imágenes, ya que cualquier variación de la iluminación como el sombreado, luz directa, cambios de intensidad puede afectar a la detección y precisión de las poses en estas imágenes.

El entorno en que se capturara la imagen puede introducir ruido visual y elementos de distracción que afectan la capacidad del modelo.

1.6 Organización de la tesis

Capítulo 1 Introducción

En esta sección se establece el contexto del porque salió la necesidad de intentar solucionar el problema, se presentan las herramientas empleadas, el objetivo general, el alcance que este tendrá y así mismo se definen las limitaciones.

Capítulo 2: Antecedentes

En esta sección se presenta todos los conocimientos previos que serán necesarios para poder entender el desarrollo de las próximas secciones, se podrá tomar como referencia para poder entender cualquier punto dentro del proyecto.

Capítulo 3: Estado del arte

Dentro de este capítulo se presenta una recolección de proyectos desarrollados con visión artificial que sean muy similares a lo que se intenta desarrollar, explicando en forma general su funcionamiento y mencionando su similitud que tiene con nuestro proyecto.

Capítulo 4: Marco teórico

En este capítulo se recolecta información técnica a lo largo de la historia ya que son conceptos específicos que serán de ayuda para poder seguir desarrollando los capítulos posteriores de nuestro tema de investigación.

Capítulo 5: Desarrollo y pruebas

En este capítulo se encontrará las herramientas utilizadas para el desarrollo de este proyecto, la instalación necesaria para su correcto funcionamiento, algunas librerías necesarias para el tratamiento y procesamiento de imágenes, el entorno

de donde y como se programará la parte de los algoritmos, y partes importantes de como interactuaran estos algoritmos con la entrada de imágenes y videos en tiempo real.

Capítulo 6: Conclusión y trabajo a futuro

En este capítulo se presenta la conclusión y resultados obtenidos del presente proyecto, así mismo se explicará un breve resumen de trabajos futuros que se podrán realizar con ayuda de este documento.

Capítulo 2

2. Antecedentes

En este capítulo se presenta los conocimientos necesarios para entender lo desarrollado en los próximos capítulos.

2.1 Visión artificial

La visión artificial tiene un funcionamiento parecido al de la visión humana, con diferencia que los humanos aprendemos de experiencias y relaciones para poder diferenciar formas, colores y un sin fin de características de objetos.

La visión artificial entrena a las máquinas para realizar estas funciones, pero tiene que hacerlo en menos tiempo con cámaras, datos y algoritmos en lugar de retinas, nervios ópticos y una corteza visual. Debido a que un sistema capacitado para inspeccionar productos o la manufactura de estos puede analizar miles de productos o procesos por minuto puede superar rápidamente las capacidades humanas, notando defectos o problemas imperceptibles (IBM, IBM, s.f.).

La visión artificial es un sistema de herramientas que permite obtener, comprender y analizar imágenes con el fin de que toda esta información se pueda procesar. En este sistema se combinan hardware y software para que los dispositivos puedan ejecutar las funciones (RAMÍREZ, 2022).

Para 2023, se espera que el mercado de la visión artificial alcance los 48.600 millones de dólares a medida que los nuevos desarrollos, como los vehículos autónomos, los drones de reparto y la detección de una «falsificación profunda», dependan de la tecnología (RAMÍREZ, 2022).

La visión artificial necesita una gran cantidad de datos. Analiza estos datos una y otra vez hasta identificar diferencias y finalmente reconocer imágenes. Por

ejemplo, para reconocer una imagen de gatos, este debe ser alimentado con grandes cantidades de gatos en diferentes escenarios para poder reconocerlos.

Se utilizan dos tecnologías principales para lograr esto, uno es de tipo machine learning llamado Deep learning y el otro es una red neuronal convolucional (CNN).

Machine learning utiliza modelos algorítmicos permitiendo que una computadora se enseñe así misma el contexto de los datos visuales. Si se alimenta de grandes cantidades de datos, será capaz de diferenciar un objeto de otros con mayor precisión, los algoritmos permiten que una maquina aprenda por si misma, en lugar de que alguien la tenga que programar para reconocer un objeto.

2.2 Sistemas de visión artificial

Un sistema de visión artificial es un conjunto de componentes, el cual hacen posible el procesamiento de datos para obtener un resultado. Se muestra a continuación con más detalles cada componente:

- Dispositivos de iluminación: sensores de luz en todas las frecuencias, sensores 2d y 3d, etc.
- Cámaras inteligentes: además de adquirir la imagen, sirve para realizar la conversión A/D (analógico-digital) de la captura de la imagen o pixeles digitalizados en una imagen de columna N por M.
- Un ordenador personal o un sistema de microprocesador: ayudara a proporcionar almacenamiento en disco de imágenes y capacidad de cálculo.
- Monitor de alta resolución: ayudara a visualizar las imágenes y los defectos de varias rutinas de análisis de imágenes.

La visión artificial tiene el propósito de lograr obtener una imagen y poder interpretar la información que hay en el justo como lo hacen nuestros ojos y el procesamiento que hay en el cerebro. Para poder tratar de duplicar este comportamiento hay varias etapas en el sistema, como se muestra en la Figura 1.

Figura 1 Etapas de un sistema de visión artificial (Ingenieril, 2021).

El escenario para analizar es el área que queremos capturar, donde se encuentra la información de interés para procesar.

Adquisición y digitalización es el proceso de tomar la imagen con ayuda de una cámara fotográfica y pasarla a algún formato digital para después enviarla a una unidad donde pueda ser procesada.

Procesamiento previo, esta etapa es muy importante para poder obtener los mejores resultados, ya que normalmente la cámara captura ruido, por lo que con este preprocesamiento trata de eliminar la mayor cantidad de ruido posible con ayuda de algunos filtros, a veces se tienen que hacer transformaciones a dicha imagen como recortes o rotaciones.

En la obtención de características se resaltan las características de interés de la imagen, los elementos más comunes son como detección de esquinas, colores, flujo óptico, formas, realce de bordes, entre otros.

En el reconocimiento e interpretación de información se procesa la información obtenida en la etapa anterior, se interpreta y se aplica una acción de acuerdo con lo analizado, dicha acción controlara la aplicación.

La aplicación es el objeto final que se controlara de acuerdo con la información que se procesó. Por ejemplo, un brazo robótico, motores, drones, niveles de videojuegos, etc.

Los beneficios de la visión artificial son los siguientes:

- Precisión: aumentan el nivel de precisión durante la fabricación de productos y eliminan los errores humanos en el producto final.
- Calidad: garantizan la máxima calidad y hacen que los productos cumplan con las especificaciones dadas.
- Fiabilidad: las computadoras y cámaras no tienen el factor humano del cansancio.
- Seguridad: mejoran la seguridad y la protección en entornos peligrosos para las personas.

Aunque igual se tienen algunos fallos por parte de la visión artificial como cuando la maquina o el dispositivo falle, no anuncia o anticipa ese problema, cuando el dispositivo falla debido a un virus o problemas de software es muy probable que el procesamiento de imagen falle, por lo que, si no se resuelve el problema, las funciones del dispositivo pueden desaparecer y pueden llegar a paralizar todo el proceso.

Los sistemas de visión artificial pueden desempeñar varias funciones dependiendo de las necesidades de la organización, entre ellas estas:

- Detección de objetos.
- Reconocimiento óptico de caracteres (OCR).
- Detección de fallas.
- Localización.
- Reconocimiento facial.
- Coincidencia de característica.

2.3 Redes neuronales

Las redes neuronales es un método de la inteligencia artificial que enseña a las computadoras a procesar datos de una manera similar en la forma en que lo hace el cerebro humano, es un tipo de aprendizaje profundo que utiliza nodos interconectados en una estructura de capas. Crea un sistema adaptable que las computadoras utilizan para aprender de sus errores mejorando continuamente. Las redes neuronales intentan resolver problemas complicados, desde reconocimientos de imágenes hasta identificar posibles fallos en la manufactura de productos.

Las redes neuronales se pueden ocupar en diversos sectores, por ejemplo, identificación de compuestos químicos, marketing orientado mediante filtrado de redes sociales, diagnostico medico mediante la clasificación de imágenes médicas, etc.

La arquitectura de una red neuronal básica como se muestra en la Figura 2, consta de neuronas artificiales interconectadas de tres capas:

- Capa de entrada: esta primera capa consta de recibir información del mundo exterior, los nodos d entrada procesan los datos, lo analizan o los clasifican y los pasan a la siguiente capa.

- Capa oculta: su entrada de datos lo toman de la primera capa, o de otras capas ocultas, las redes neuronales pueden tener una gran cantidad de capas ocultas, cada capa oculta debe analizar la salida de la capa anterior procesándola cada vez más y pasando el resultado a la siguiente capa.

- Capa de salida: proporciona el resultado final de todo el procesamiento previo de las capas anteriores, esta capa puede tener uno o más nodos, por ejemplo, si tenemos un problema de clasificación binaria, la capa de salida tendrá un nodo de salida que dará como resultado 1 o 0. Sin embargo, si tenemos un problema de clasificación multiclase, la capa de salida puede estar formada por más de un nodo de salida.

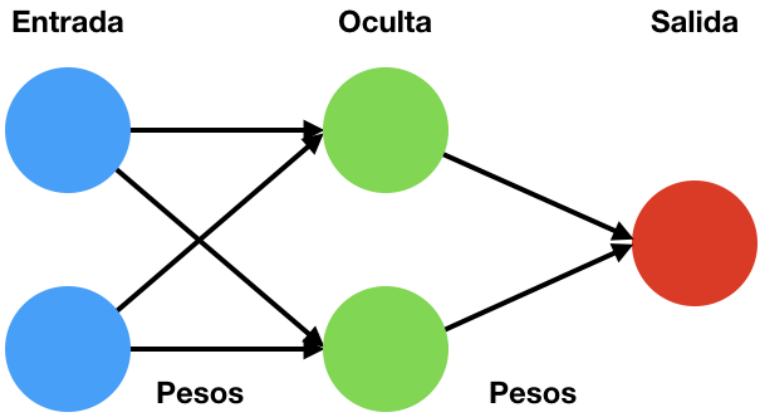

Figura 2 Ejemplo de una red neuronal simple (Rodríguez, 2018).

El entrenamiento de las redes neuronales es el proceso de enseñar a una red neuronal a realizar una tarea, en principio las redes neuronales aprenden procesando conjuntos grandes de datos etiquetados o sin etiquetar, gracias a esto pueden procesar datos de entrada desconocidos con mayor precisión.

Existen diferentes tipos de redes neuronales artificiales, cada una con su propia arquitectura y función específica. Algunos de los tipos más comunes de redes neuronales son:

- Redes neuronales feedforward también conocidas como redes neuronales de propagación hacia adelante, son las más simples y comunes. En estas redes la información fluye en una sola dirección, desde la capa de entrada hasta la capa de salida sin ciclos o retroalimentación. Estas redes son ideales para clasificación, predicción y reconocimiento de patrones (Brutti, 23).

- Redes neuronales recurrentes, a diferencia de las redes feedforward, las redes neuronales recurrentes permiten la retroalimentación de la información en la red. Esto significa que la salida de una capa puede ser utilizada como entrada en la siguiente iteración del proceso. Estas redes son ideales para tareas de procesamiento de lenguaje natural y series de tiempo (Brutti, 23).

- Redes neuronales convolucionales. Las redes convolucionales son un tipo de red neuronal que se especializa en procesamiento de imágenes y videos. Estas redes utilizan filtros para extraer características relevantes de las imágenes, y luego combinan estas características para hacer predicciones. Las redes convolucionales son comúnmente utilizadas para tareas de reconocimiento de objetos, clasificación de imágenes y segmentación de imágenes. Así como lo veremos a detalle en la próxima sección (Brutti, 23).

- Redes neuronales autoencoder. Estas redes neuronales son una forma de redes neuronales feedforward que se utilizan para la reducción de la dimensionalidad y la compresión de datos. Estas redes constan de una capa de entrada, una o más capas ocultas y una capa de salida que es una réplica de la capa de entrada. Estas redes son utilizadas para tareas de compresión de datos y reducción de ruido (Brutti, 23).

La inteligencia artificial es el campo de las ciencias de la computación que investiga métodos para dar a las máquinas la capacidad de realizar tareas. El machine learning (ML) es una técnica de inteligencia artificial que otorga a las

computadoras acceso a un conjunto de datos muy grandes y les enseña a aprender de estos. La técnica de ML encuentra patrones dentro de estos datos y los aplica a datos totalmente nuevos para tomar decisiones inteligentes. Aunque este método requiere la intervención humana para que funcione lo suficientemente bien, un científico de datos determina de forma manual el conjunto de características relevantes que el software de Ml debe analizar.

Por ejemplo, si se entrenara un software de ML para identificar las partes del cuerpo humano de forma correcta, debería seguir estos pasos:

- Encontrar y etiquetar de forma manual miles de imágenes de personas, en diferentes entornos y en diferentes poses.

- Indicar al software de Ml que características buscara para poder identificar la imagen mediante la eliminación, por ejemplo, podía detectar el número de pies, manos, hombros, piernas, etc.

- Evaluar y cambiar de forma manual los conjuntos de datos etiquetados para mejorar la precisión por ejemplo si se tiene la imagen de una persona parada frente a la cámara, tendera a detectar con mayor precisión a una imagen similar a esta característica, que a una que está sentada o acostada.

Existen varios tipos de algoritmo de aprendizaje utilizados en el campo de la inteligencia artificial y el aprendizaje automático. Algunos de los tipos más comunes son:

- Aprendizaje supervisado: este tipo de algoritmo utiliza datos etiquetados para entrenar un modelo que pueda predecir las etiquetas de nuevos datos. Los datos refieren a datos que ya tienen una respuesta correcta. Por ejemplo, si se está entrenando un modelo para clasificar imágenes de gatos y perros, cada imagen se etiquetaría como ‘gato’ o ‘perro’ (IBM, 2024).

- Aprendizaje no supervisado: este tipo de algoritmo no utiliza datos etiquetados y en cambio busca patrones y estructuras en los datos. El aprendizaje no supervisado se utiliza a menudo para tareas de agrupamiento (clustering) o reducción de la dimensionalidad (IBM, 2024).

- Aprendizaje por refuerzo: este tipo de algoritmo utiliza un sistema de recompensas para entrenar un modelo para tomar decisiones en un entorno determinado. El modelo aprende a través de la retroalimentación en forma de recompensas y castigos.

- Redes neuronales: este tipo de algoritmo se basa en el funcionamiento del cerebro humano y está diseñado para aprender a través de la retroalimentación y la adaptación. Las redes neuronales se utilizan a menudo para tareas de clasificación, reconocimiento de patrones t predicción (ISO, 2024).

- Aprendizaje por transferencia: este tipo de algoritmo utiliza el conocimiento adquirido de una tarea para ayudar a resolver otra tarea relacionada. Por ejemplo, si un modelo ha sido entrenado para reconocer imágenes de coches, se puede utilizar parte de ese conocimiento para ayudar a clasificar imágenes de motocicletas.

2.4 Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales derivan de las redes neuronales y se consideran uno de los modelos más eficientes para la clasificación de imágenes.

Están compuestas de capas de nodos, que contienen una capa de entrada, una o más capas ocultas y una capa de salida. Cada nodo se conecta a otro y tiene un peso y umbral asociados. Si la salida de cualquier nodo individual está por

encima del valor del umbral especificado, ese nodo se activa y envía datos a la siguiente capa de la red. De lo contrario, no se pasa ningún dato a la siguiente capa de la red (IBM, IBM, 2023).

Antes de que existieran las CNN, la obtención de características de las imágenes se tenía que hacer manualmente, por lo que requería un costo mayor de tiempo. Las CNN nos ofrece la ventaja de hacerlo mucho más rápido con ayuda de algunas operaciones del algebra lineal, aunque se requiere de una mayor cantidad de poder de cómputo.

La convolución es una operación matemática que combina dos funciones para describir la superposición entre ambas. La convolución toma dos funciones, “desliza” una sobre la otra, multiplica los valores de las funciones en todos los puntos de superposición, y suma los productos para crear una nueva función. Este proceso crea una nueva función que representa cómo interactúan las dos funciones originales entre sí (Mathworks.com, 2024).

Así como se muestra en la Figura 3.

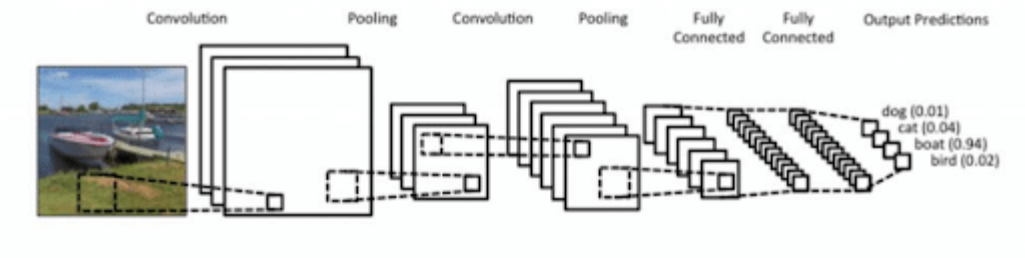

Figura 3 Arquitectura de una CNN (DatasCientest, 2021).

Los tres tipos principales de capas que tienen las CNN son:

- Capa convolucional: es la capa más importante de nuestra red ya que esta capa conlleva el mayor peso de cálculo y requiere de tres principales componentes, imagen de entrada, un filtro o kernel que por lo general es una matriz de 3x3 con ciertos pesos, y un mapa de características. Supongamos que nuestra imagen de entrada es una imagen a color, entonces sería una matriz de 3D. El filtro se aplica obteniendo el producto

de esta matriz con la de la matriz de la imagen de entrada, pasando por cada uno de los pixeles de la imagen, guardando el resultado en una matriz de salida. Esta matriz resultada se conoce comúnmente como mapa de características, mapa de activación o característica convolucionada.

Después de toda la operación de la capa de convolución, al mapa de características se la aplica una transformación de unidad lineal rectificada (ReLu), creando la no linealidad en el modelo.

- Capa de agrupamiento: es común agregar una capa de agrupación entre capas sucesivas de convoluciones, esta capa conocida como reducción de muestreo sirve principalmente para reducir la dimensionalidad y, por lo tanto, la cantidad de parametros y cálculos en la red, controlando el sobreajuste, mejorando la eficiencia y reduciendo la complejidad. Esta operación de agrupación aplica un filtro a toda la entrada, pero a diferencia de este filtro con el filtro de la primera capa, este no tiene pesos. Este filtro puede aplicar uno de estos dos tipos de agrupación:

  La agrupación máxima selecciona el píxel con el valor máximo dentro de la matriz de entrada a medida que el filtro se mueve dentro de esta, enviándoselo a la matriz de salida. Este tipo de agrupación es la más usada, una representación gráfica es como se muestra en la Figura 4.

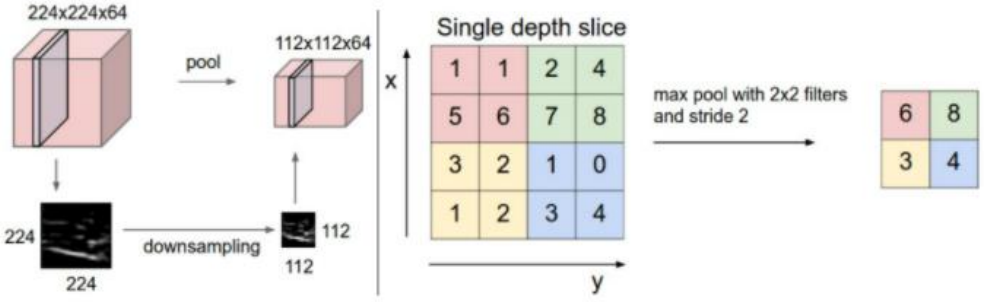

Figura 4 Representación de la agrupación máxima (Aplicada, s.f.).

La agrupación promedio calcula el promedio dentro de la matriz de entrada a medida que el filtro se mueve dentro de esta, enviándoselo a la matriz de salida.

- Capa totalmente conectada: esta capa está totalmente conectada la matriz de entrada con la matriz de salida aplicando una combinación lineal y una función de activación con el objetivo de clasificar la imagen de entrada (IBM, IBM, 2023).

2.5 Función de activación

Es una función que transmite la información generada por la combinación lineal de los pesos y las entradas, es decir son la manera de transmitir la información por las conexiones de salida. (Tech, s.f.).

Estas funciones tienen como características que deben ser no lineales, entre las funciones de activación más comunes son:

Función escalón: conocida de igual manera como función escalón unitario o función de Heaviside es una función matemática simple que devuelve una salida binaria, es decir que toma el valor cero si está por debajo de cierto umbral, en caso contrario toma el valor uno.

Esta función tiene la siguiente forma.

$$f(x) = \begin{cases} 0, & x < umbral \\ 1, & x \geq umbral \end{cases}$$

Esta función es como un interruptor cambiando su valor de uno a cero en un punto especifico, manteniendo ese valor después de ese punto. El uso de esta función es por su simplicidad y no consume muchos recursos computacionales, aunque esta función no puede ser utilizada para algoritmos de aprendizaje

automático que requieren de cálculos de gradiente y tampoco es adecuada para modelar problemas en donde las salidas deben ser continuas. Estas funciones se pueden utilizar para detectar correos spam donde solo nos interesa saber si es o no es spam.

Función sigmoidal: esta función toma un valor de entrada y lo comprime en un valor entre el rango 0 y 1 lo que la hace una función para modelar probabilidades y realizar clasificaciones binarias, se utiliza como función de activación para las capas ocultas y se representa con la siguiente formula:

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

Su gráfica se muestra en la figura 5.

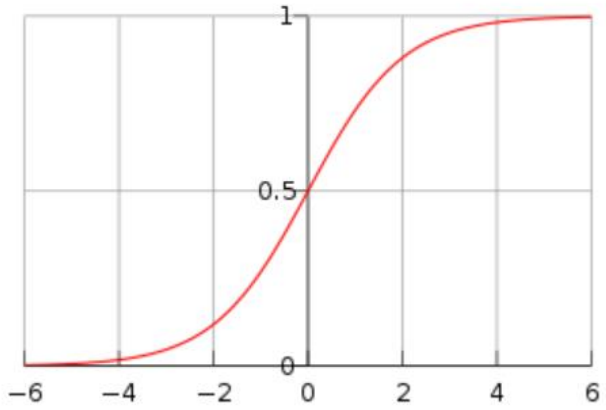

Figura 5 función Sigmoidal (Javi, La Función Sigmoide: Una Herramienta Clave en Redes Neuronales, 2023).

Uno de los problemas es que la función se satura cuando los valores de entrada son grandes, lo que significa que la derivada de la función se acerca a cero y el aprendizaje se ralentiza. Además, la función sigmoide no es simétrica, lo que significa que las entradas negativas y positivas se procesan de manera diferente, lo que puede afectar el rendimiento de la red (Javi, La Función Sigmoide: Una Herramienta Clave en Redes Neuronales, 2023).

Función tangente hiperbólica: es una función matemática que mapea un rango de valores de -1 a 1, es la relación entre el seno y el coseno hiperbólicos de un número. La Tangente Hiperbólica toma un número x y lo transforma en otro número que cae dentro del rango [-1, 1] (Javi, 2023).

Esta función tiene la siguiente forma:

$$(x) = \frac{2}{1 + e^{-2x}} - 1$$

Su gráfica se representa como se muestra e la Figura 6.

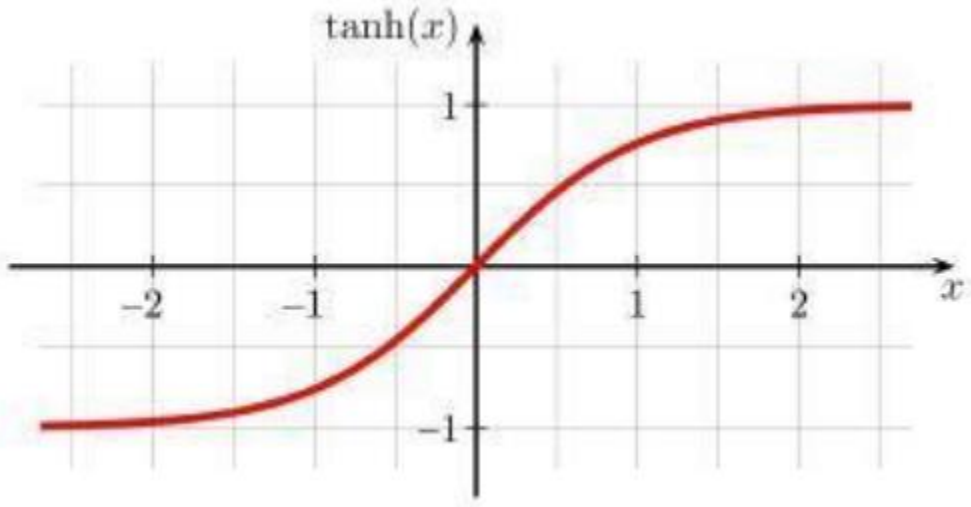

Figura 6 Grafica de la función Tanh.

Función rectificadora (ReLu): Esta función es una de las más usadas actualmente porque resuelve problemas de saturación que otras funciones tienen. Es una función umbraliza la entrada en cero, regresando un 0 para los valores negativos y si son positivos las entradas regresa la propia entrada.

Esta función tiene la siguiente forma:

$$f(x) = max(0, x)$$

Técnicamente ReLu es una función no lineal ya que tiene un punto no diferenciable en x=0, esta no linealidad permite a las redes neuronales aprender patrones complejos. La función ReLu es computacionalmente poco costosa porque implica un simple umbral en cero. Esto permite a las redes escalar a muchas capas sin un aumento significativo de la carga computacional, en comparación con funciones más complejas como la función tanh o la sigmoide (Ali, 2024).

Su gráfica se muestra en la Figura 7.

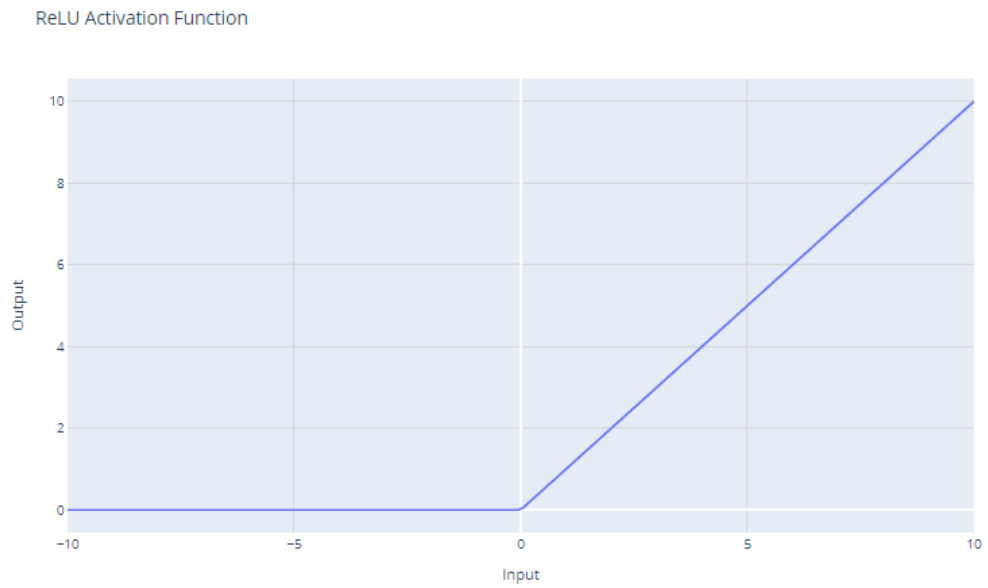

Figura 7 Grafica de la función ReLu. (Ali, 2024).

2.6 Procesamiento digital de imágenes

El tratamiento o procesamiento de imagen digital es un proceso que consiste en modificar o mejorar la apariencia de una imagen. Esto se realiza mediante una variedad de técnicas, como el ajuste de brillo y contraste, la corrección de color,

la eliminación de ruido, la eliminación de objetos no deseados, la adición de efectos, entre otros. El objetivo final es mejorar la calidad de la imagen para una determinada aplicación.

El procesamiento de imagen es una etapa crucial en la visión artificial ya que permite a la computadora analizar y comprender los datos visuales capturados por una cámara o un dispositivo similar. Incluye tareas desde la adquisición de imágenes como la correlación y la mejora de calidad de la imagen. La información resultante es utilizada por los algoritmos de visión artificial para tomar decisiones y realizar tareas específicas, como el seguimiento de objetos, detección de patrones, entre otros.

Caracterización de una imagen:

- Discretización: La imagen se representa mediante un conjunto finito de puntos llamados pixeles.
- Formato: La imagen se almacena en un archivo digital utilizando un formato especifico, como JPEG, PNG, GIF, entre otros.
- Resolución: La resolución es el número de píxeles que contiene la imagen, entre mayores pixeles contenga, mayor la claridad y detalle.
- Profundidad de color: Se refiere al número de bits utilizados para representar cada píxel de la imagen.
- Escala de grises: Algunas imágenes están en blanco y negro, por lo que estas se representan como una escala de grises (esta característica se utiliza mucho en los sistemas de visión artificial).
- Compresión: permite reducir el tamaño de la imagen sin reducir la calidad.
- Metadatos: Son datos adicionales sobre la imagen, como la fecha de creación, la ubicación geográfica, entre otros.

Una imagen se expresa de forma matemáticamente para que las computadoras puedan procesarla como se muestra en la Figura 8, algunas de estas formas son mediante una función que asigna un valor numérico a cada punto del plano (o el espacio en el caso de las imágenes tridimensionales). Por ejemplo, una imagen en blanco y negro puede ser expresada con la siguiente función f(x,y) que toma los valores 0 (negro) a 1 (blanco) en cada punto (x,y) del plano. Una imagen a color podría ser expresada mediante tres funciones R(x,y), G(x,y) y B(x,y) que indica la intensidad de los componentes rojo, verde y azul en cada punto.

Un píxel (abreviatura de “picture element”) es la unidad básica de una imagen digital. Es el elemento más pequeño que se pueda mostrar en pantalla de visualización o impresora, y se representa mediante un valor numérico que indica su color y su intensidad. Se agrupan para formar una imagen completa (Valente, 2018).

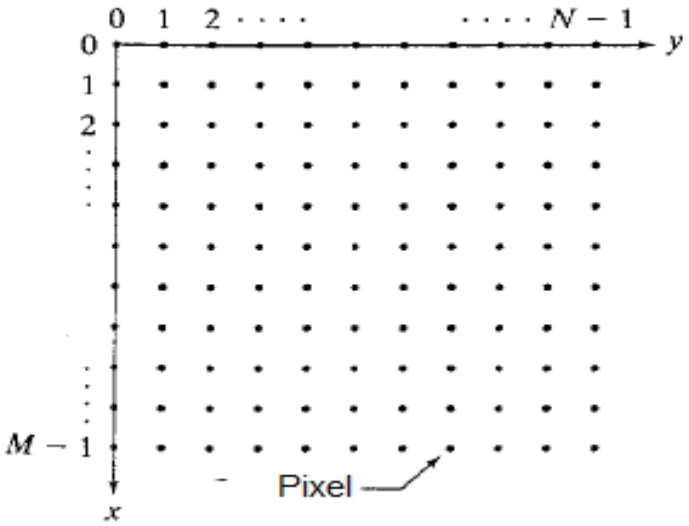

Figura 8 Representación de una imagen en forma digital. (Mejía, 2015).

Hay varios tipos de imágenes que se utilizan comúnmente en el procesamiento de imágenes. Algunos de los tipos más comunes son:

Las imágenes de mapa de bits (bitmap) como se muestra en la Figura 9, se componen de una cuadricula de pixeles, donde cada píxel tiene un valor de color

especifico. Los formatos de archivo comunes para las imágenes de mapa de bits incluyen JPEG, PNG y BMP (Valente, 2018).

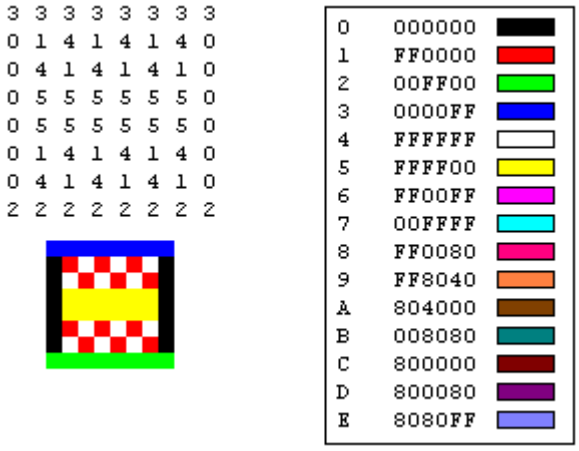

Figura 9 Imagen de mapa de bits (Microsoft, 2024).

Las imágenes vectoriales como se muestra en la Figura 10, se componen de formas geométricas, como líneas curvas, que se describen mediante ecuaciones matemáticas. Debido a esto, las imágenes vectoriales se pueden escalar sin pérdida de calidad. Los formatos de archivo comunes para las imágenes vectoriales incluyen SVG y AI (Valente, 2018).

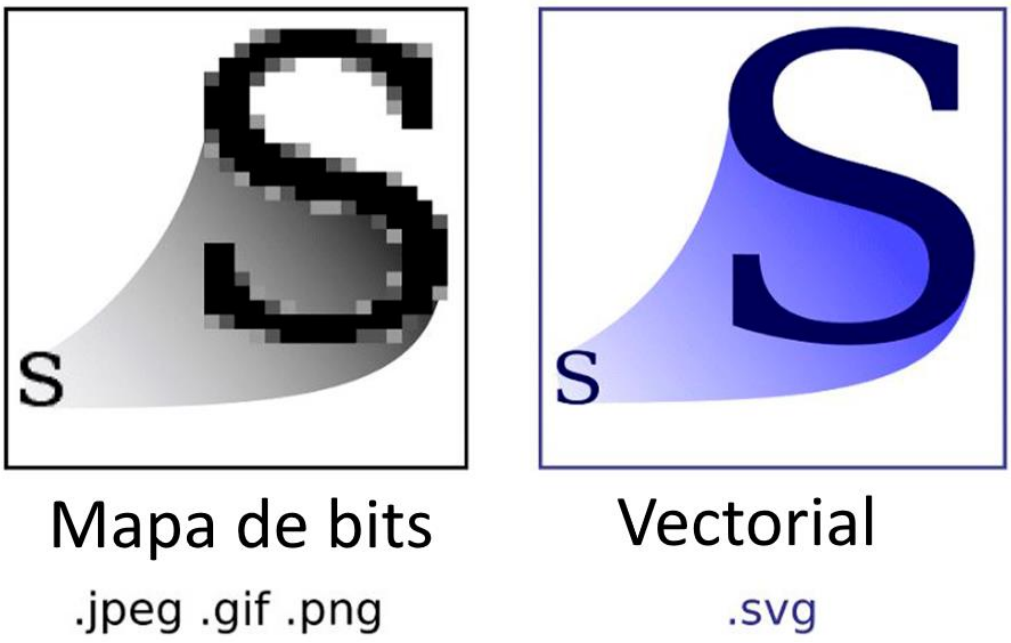

Figura 10 Imagen vectorial (Lara, 2020).

Las imágenes a escala de grises como se muestra en la Figura 11, se componen de tonos de gris en lugar de colore. Cada píxel tiene un valor de intensidad de gris que varía de 0 (negro) a 255 (blanco).

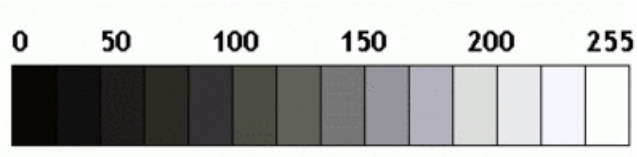

Figura 11 Escala de grises (values, 2024).

Las imágenes en 3D como se muestra en la Figura 12, se componen de modelos tridimensionales que se pueden renderizar desde diferentes ángulos para crear una imagen en 3d. los formatos de archivos comunes para las imágenes en 3d incluyen OBJ y STL.

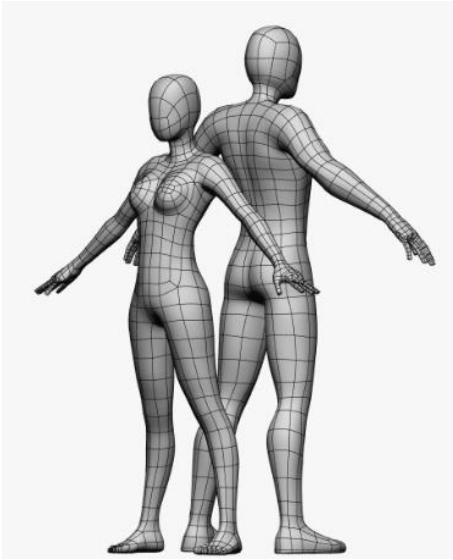

Figura 12 imágenes en 3D (TurboSquid, 2010).

Las imágenes térmicas como se muestra en la Figura 13, se componen de información de temperatura, donde cada píxel representa una temperatura especifica. Las imágenes térmicas se utilizan comúnmente en la industria y la medicina para detectar cambios en la temperatura.

Figura 13 imágenes térmicas (Brajovic, 2021).

Los filtros en el tratamiento de imágenes son muy importantes, ya que estos son herramientas o algoritmos que se utilizan para modificar o mejorar la apariencia de una imagen. Hay diferentes tipos de filtros, por ejemplo:

El filtro de brillo y contraste que ajusta la intensidad de los pixeles en una imagen para mejorar su visibilidad.

El filtro de nitidez que enfatiza los detalles de las imágenes aumentando su claridad.

Los filtros de colores que cambian la tonalidad de una imagen, como o cuando se aplica un tono sepia o saturar colores.

Los filtros de enfoque que simulan la captura de la imagen con un lente de enfoque suave.

Un ejemplo de filtro a escala de grises es básicamente recorrer todos los pixeles de la imagen, para cada píxel calcular el valor de gris del píxel, este valor se calcula promediando los valores de sus tres componentes de color, el rojo, el

verde y el azul. Y por último asignar ese valor al píxel, una vez terminando el recorrido de los pixeles de la imagen estará a escala de grises (Valente, 2018).

Este proceso sería muy similar a cualquier otro filtro, solo sería cambiar el cálculo del nuevo valor del píxel de la imagen como se muestra en la Figura 14.

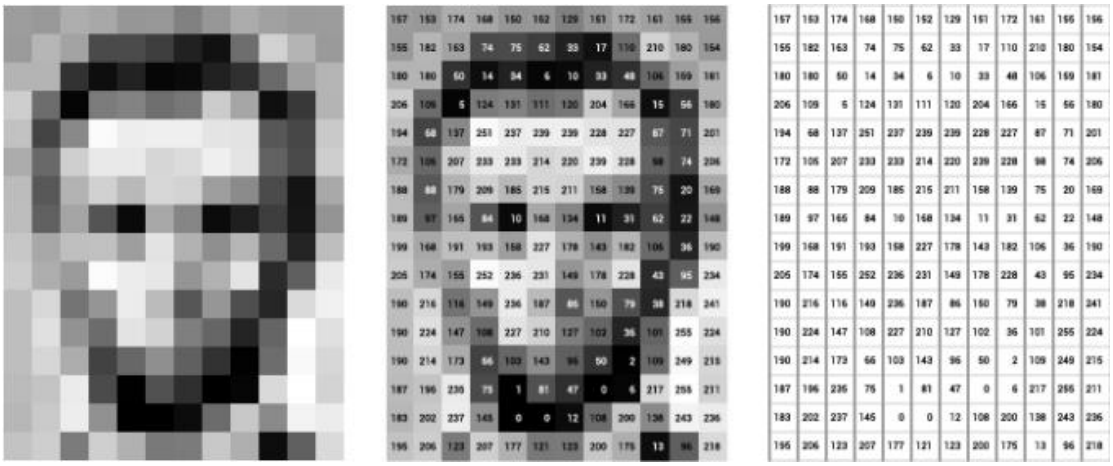

Figura 14 Representación de una imagen a escala de grises (Martínez, 2018).

2.7 Sensores de visión

Un sensor de visión es un dispositivo electrónico que permite la captura y la conversión de información visual en una forma digital para su posterior procesamiento y análisis. Estos sensores se utilizan en una amplia variedad de aplicaciones, incluyendo fotografía, robótica, vigilancia, automatización y control de calidad.

Hay varios tipos de sensores de visión, cada uno con características y aplicaciones propias.

Las cámaras digitales funcionan capturando la luz reflejada por los objetos en su campo de visión y convirtiéndolas en una imagen digital. Esto se logra a través de un proceso en varias etapas.

La apertura que es la luz que entra a través del objetivo y llega a la abertura, también conocida como diafragma, que controla la cantidad de luz que entra en la cámara.

El enfoque que es la lente en el objetivo que enfoca la luz en un punto detrás de la abertura.

La exposición es el sensor de imagen o también conocido como sensor CCD, CMOS, capta la luz y la convierte en una señal eléctrica.

El procesamiento es la señal eléctrica procesada por l procesador de imagen en la cámara para producir un archivo de imagen digital.

Por último, el almacenamiento que es donde se almacenara la imagen digital, ya sea en una tarjeta de memoria o en la memoria interna de la cámara.

Estos son los elementos básicos de funcionamiento de una cámara digital para producir una imagen digital.

Los sensores de profundidad se utilizan para medir la distancia a objetos en el espacio y para crear mapas en 3D. Estos sensores se utilizan en sistemas de realidad aumentada y robótica.

Los sensores de línea se utilizan para detectar líneas en una imagen y se utilizan en aplicaciones industriales para el control de calidad y detección de fallas.

Los Sensores de patrones se utilizan para detectar patrones específicos en una imagen y se utilizan en aplicaciones como la identificación de objetos y la detección de movimientos (Ridgway, 2022).

Capítulo 3 Estado del arte

3. Estado del arte

En este capítulo se recolectaron proyectos muy similares a lo que se desarrollara al final de este documento, se presenta el nombre y una pequeña descripción por cada elemento, así como la plataforma compatible de estos.

3.1 Kinect

El Kinect es un dispositivo de entrada que utiliza sensores de movimiento desarrollado por Microsoft para su consola de juegos Xbox y PC con Windows. Utiliza una combinación de cámaras, sensores infrarrojos y micrófonos para detectar y rastrear los movimientos de los jugadores frente a él, permitiendo experiencias de juego inmersivas y el control de otras aplicaciones mediante gestos naturales y comandos de voz. El dispositivo fue lanzado por primera vez en 2010 y desde entonces ha experimentado varias interacciones con una funcionalidad mejorada, aunque está diseñado principalmente para juegos, Kinect ha encontrado aplicaciones en campos como la atención médica, la educación y la robótica (Page, 2019).

El Kinect consta de los siguientes componentes:

- Sensor Kinect, es el componente principal del dispositivo, que contiene las cámaras, sensores infrarrojos y micrófonos que se utilizan para detectar y rastrear movimientos y la voz del usuario como se muestra en la Figura 15.

- Cable de alimentación que se conecta a la fuente de alimentación para proporcionar energía del sensor Kinect, se observa en la Figura 16.

- Cable USB que se conecta a la consola de Xbox o pc para transmitir los datos del sensor Kinect, se observa en la Figura 16.
- Adaptador de corriente que se conecta a la fuente de alimentación y al sensor Kinect para proporcionar energía al dispositivo, como se observa en la Figura 16.

Figura 15 Kinect.

Figura 16 Cables necesarios para conectar el Kinect.

3.2 EA Sports Active 2

Este título es un gran juego para consolas de videojuegos como para la ps3, Wii, Xbox 360, el cual para las 2 primeras consolas se necesitan 2 complementos, un sensor de brazo y un sensor de pierna. Para el caso del Xbox solo se requiere de

un complemento llamado Kinect. Es opcional un pulsómetro para medir el ritmo cardiaco para las tres consolas. Es un videojuego de ejercicio lanzado por EA Vancouver. Fue lanzado el 16 de noviembre de 2010 (ltucker, 2010). Es un juego que ayuda a poder crear una rutina de ejercicio, para el caso de Xbox con ayuda del Kinect, detecta los movimientos que hace una persona para poder seguir avanzando en esta rutina como se muestra en la Figura 17.

Figura 17 EA Sports Active 2.0 (Baker, 2010).

3.3 The Biggest Loser Ultimate Workout

Este título trae un estilo de vida saludable a la sala de estar de la familia como nunca, brindando nuevas formas de jugar juntos y motivarse mutuamente a través del ejercicio. Es solo compatible para la consola de Xbox con Kinect, como se muestra en la Figura 18.

Figura 18 The Biggest Loser Ultimate Workout (Newton, 2010).

3.4 Your Shape Fitness Evolved

Este título fue lanzado para Xbox en el año 2010 como título de lanzamiento para Kinect. Es un juego orientado al ejercicio, ya que reconoce las posturas del cuerpo para poder seguir avanzando dentro del juego como se muestra en la Figura 19.

Figura 19 Your Shape Fitness Evolved (contributors, 2024).

Capítulo 4 Marco teórico

4. Marco teórico

En este capítulo se explicará a detalle el funcionamiento de las herramientas utilizadas para desarrollar el proyecto final de este documento.

4.1 Python

Python fue desarrollado originalmente por Guido Van Rossum en 1989 como un proyecto de hobby mientras trabajaba en el centro de matemáticas e informática de los países bajos. Van Rossum buscaba un lenguaje de programación que fuera fácil de aprender y usar pero que también fuera poderoso y escalable (AWS, s.f.).

El primer lanzamiento público de Python fue en 1991, y desde entonces ha sido desarrollado por una comunidad global de desarrolladores y usuarios. Python ha evolucionado significativamente desde su lanzamiento inicial, con varias versiones importantes lanzadas a lo largo de los años (AWS, s.f.).

Actualmente, Python es uno de los lenguajes de programación más populares y ampliamente utilizados en todo el mundo. Su sintaxis clara y legible combina con una gran cantidad de bibliotecas y herramientas de terceros, lo convierte en un lenguaje de elección para una amplia variedad de aplicaciones, desde el desarrollo de aplicaciones web hasta la ciencia de datos y el aprendizaje automático.

Python es un lenguaje de programación de alto nivel, interpretado y multiparadigma. De alto nivel significa que está diseñado para ser fácil de leer y escribir. Interpretado significa que el código de Python se ejecuta directamente por el intérprete, sin necesidad de compilarlo previamente. Y multiparadigma significa que admite varios estilos de programación, incluyendo programación

orientada a objetos, programación funcional y programación procedimental (AWS, s.f.).

Para descargar e instalar Python tenemos que ir a su página oficial y descargar la versión para nuestro sistema operativo y la versión especifica de Python, en nuestro caso será la versión 3.11.4. La instalación es muy sencilla solo seguimos los pasos y listo.

Para comprobar que se tiene instalado Python correctamente se puede colocar el siguiente comando “Python -V” como se muestra en la Figura 20.

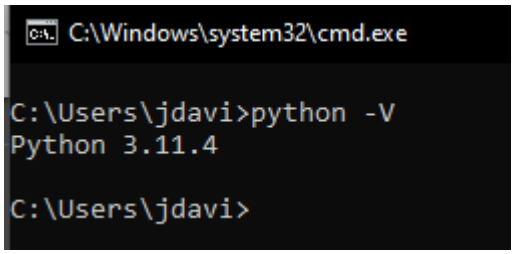

Figura 20 Versión de Python (Propia del autor).

La sintaxis general de Python es la siguiente:

- Identación: Python utiliza la identación en lugar de llaves para delimitar bloques de código. Esto significa que la cantidad de espacio en blanco (generalmente cuatro espacios) antes de una línea de código indica la estructura del programa.

- Comentarios: los comentarios en Python comienzan con el símbolo ‘#’ y se extienden hasta el final de la línea, o un bloque de comentarios se hace con este símbolo’”””” tanto para el inicio como para el cierre del bloque. Así como se muestra en la Figura 21.

```
"""
Comentario
de varias
lineas
"""

# Comentario de una sola linea
```

Figura 21 Comentario (Propia del autor).

- Variables: las variables en Python se crean mediante la asignación de un valor a un nombre.

- Tipos de datos: Python admite varios tipos de datos incluyendo números enteros, números flotantes, cadenas, booleanos y objetos complejos.

- Funciones: las funciones en Python se definen utilizando la palabra reservada ‘def’ seguida del nombre de la función y sus parámetros entre paréntesis. El cuerpo de la función se indenta debajo de la línea de definición. Así como se muestra en la Figura 22.

```
def funcion(a,b):
    return a + b;

print(funcion(5,9))
```

Figura 22 Definición de una función (Propia del autor).

- Control de flujo: Python admite estructuras de control de flujo como ‘if’, ‘else’, ‘elif’, ‘while’, ‘for’ y ‘try/except’, así como se muestra en la Figura 23 algunas estructuras de control.

```
num = -8;
if(num > 0 ):
    print(f"El numero {num} es positivo" )
else:
    print(f"El numero {num} es negativo" )

for i in range(0,10):
    print(i)

a = 0
while a < 10:
    print(a)
    a += 1
```

Figura 23 Estructuras de control (Propia del autor).

- Clases: Las clases proveen una forma de empaquetar datos y funcionalidad juntos. Al crear una nueva clase se crea un nuevo objeto, permitiendo crear nuevas instancias de este tipo. (Foundation, 2021) Estas clases de Python funcionan con todas las características de la programación orientada a objetos. Estas clases normalmente se les agrega una función llamada "__init__" para inicializar el objeto creado. Así como se muestra en la Figura 24, se creó una clase llamada MiClase en donde esta contiene 2 funciones, la primera llamada "__init__ recibe dos parametros y la segunda función llamada "suma" lo único que hace es sumar los dos parametros anteriores, en la línea 27 se crea una instancia de la clase llamándola "variable_clase" y se le pasan los dos parametros de __init__ y si se manda a llamar a la función suma, esta regresa en este caso la suma de ambos números que se le paso a esta clase.

```
class MiClase:

    def __init__(self,numero1,numero2):
        self.num1 = numero1
        self.num2 = numero2
    def suma(self):
        return self.num1 + self.num2

variable_clase = MiClase(5, 9)
variable_clase.suma
```

Figura 24 Clase en Python (Propia del autor),

4.2 Spyder IDE

Spyder es un entorno científico gratuito y de código abierto escrito en Python, para Python, y diseñado por y para científicos, ingenieros y analistas de datos. Cuenta con una combinación única de la funcionalidad avanzada de edición, análisis, depuración y creación de perfiles de una herramienta de desarrollo integral con exploración de datos, ejecución interactiva, inspección profunda, etc., y hermosas capacidades de visualización de un paquete científico. (Contributors, 2023)

En la Figura 25 se puede observar la ventana del IDE Spyder.

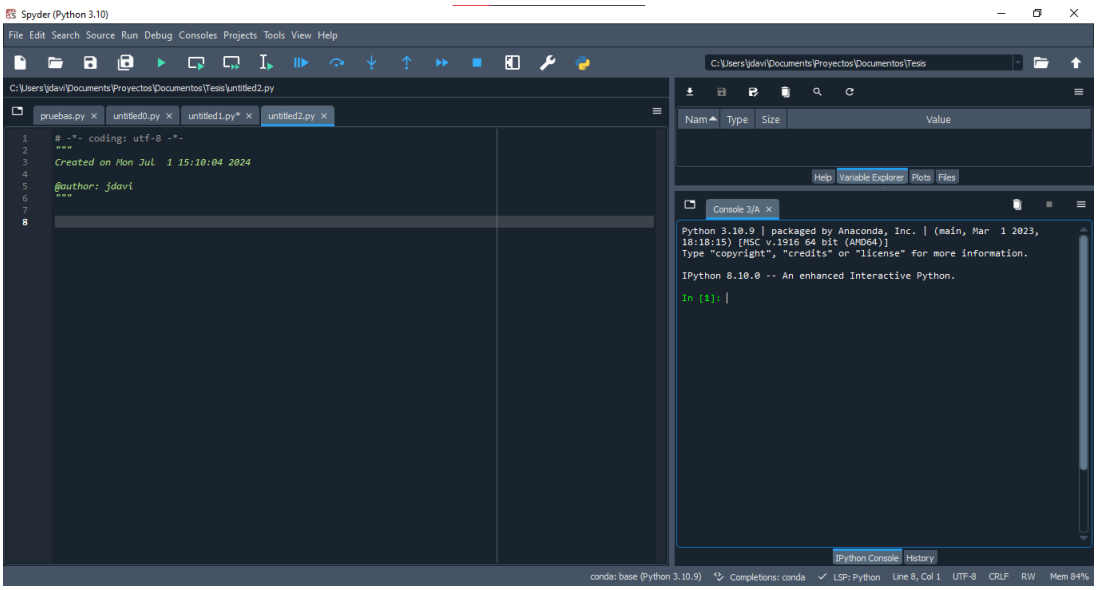

Figura 25 Spyder IDE (Propia del autor).

Cuenta con las siguientes herramientas:

- Un editor que permite trabajar de manera eficiente con diferentes idiomas, herramienta de análisis y con un diseño adaptable a nuestras necesidades. Así como se muestra en la Figura 26.

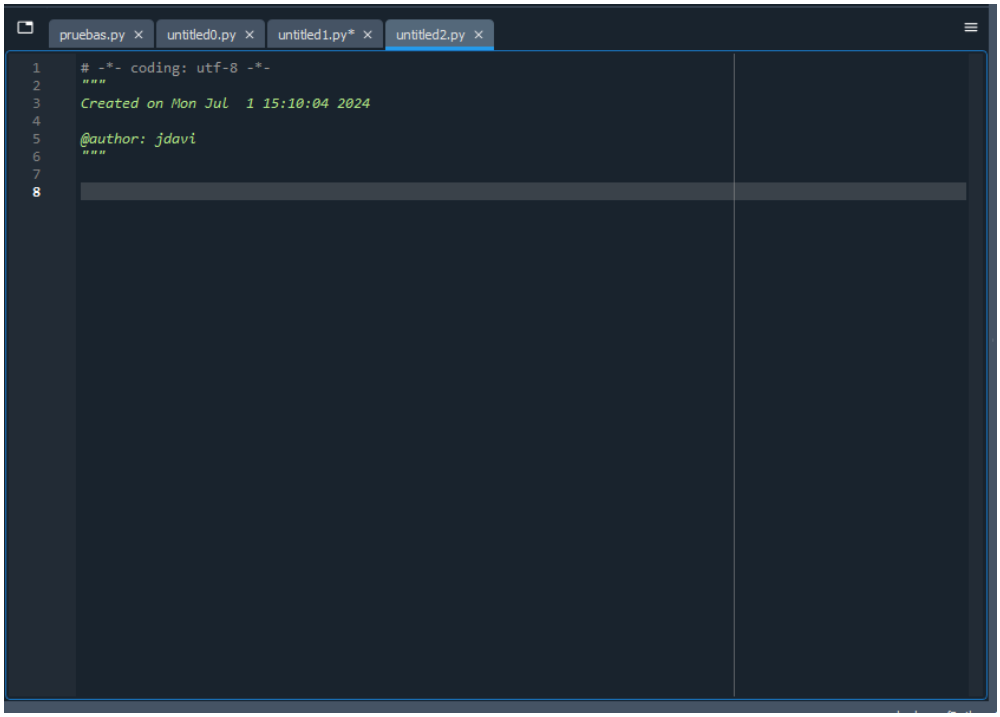

Figura 26 Editor Spyder (Propia del autor).

- Consolas de IPython que permite tener varias consolas a la vez así como se muestra en la Figura 27, se ejecutar una línea un segmento o todo un archivo de código, permite el poder depurar código y además tiene algunos comandos mágicos.

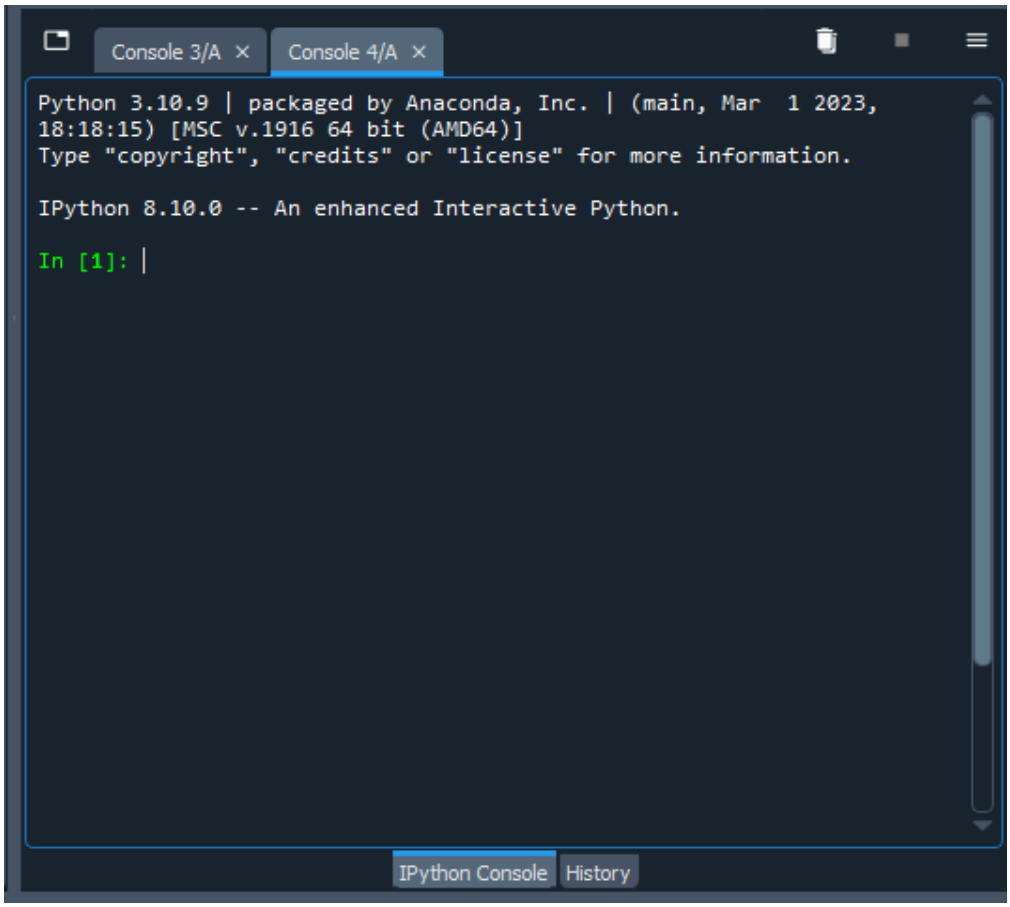

Figura 27 Consolas de Spyder (Propia del autor).

- Explorador de variables donde no solo se pueden ver los valores de las variables, sino que también nos se puede modificarl estas variables sobre la marcha de la ejecución. Así como se muestra en la Figura 28 que se tienen varias variables de las que pueden modificar su valor en tiempo de ejecución del programa.

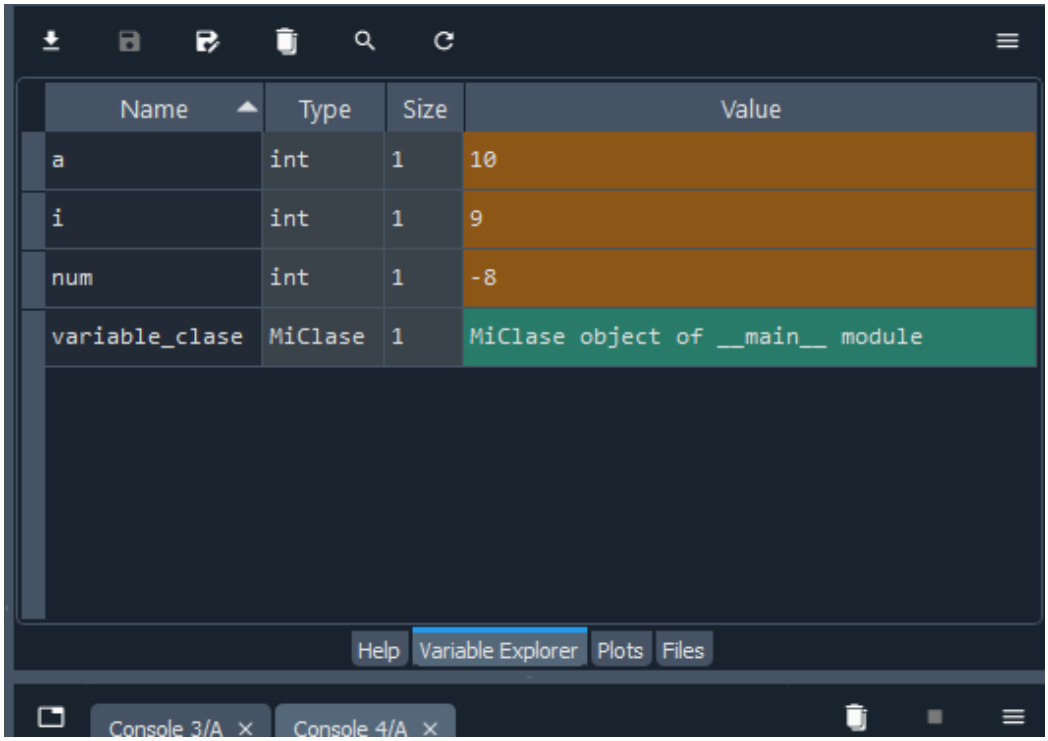

Figura 28 Explorador de variables (Propia del autor).

- Los plots (parcelas) no son mas que un visualizador de imágenes donde se puede navegar, manejar el zoom, copiar y guardar figuras e imágenes creadas. Así como se observa en la Figura 29.

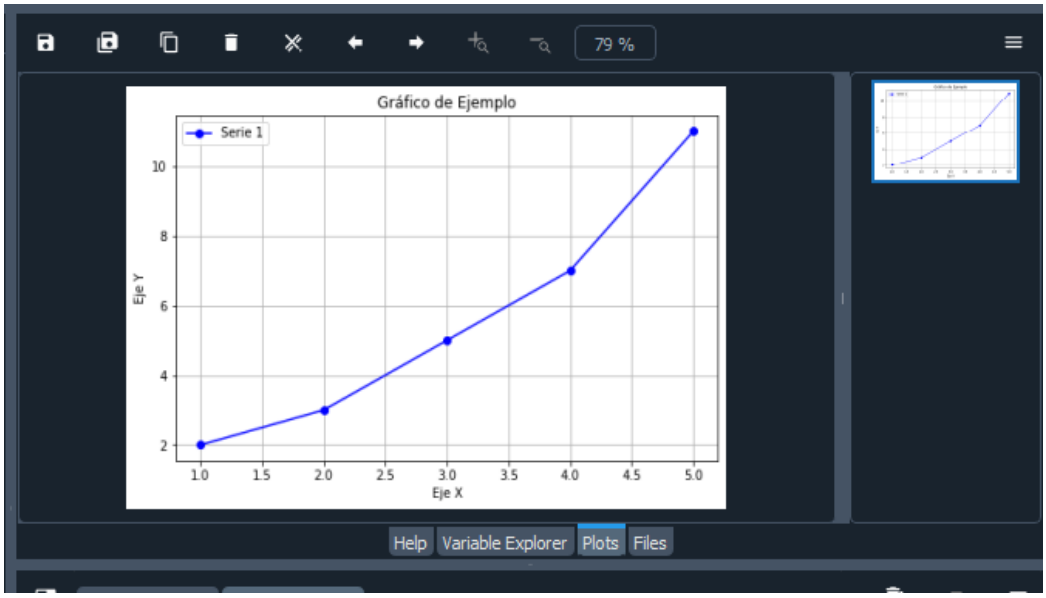

Figura 29 Plots (Propia del autor).

4.3 Mediapipe

MediaPipe fue desarrollado por Google Research en colaboración con un equipo interno de ingenieros y desarrolladores de Google. El proyecto comenzó en 2014 y se presentó públicamente en 2018 como un marco para el desarrollo de aplicaciones de procesamiento de datos multimedia en tiempo real (Fernandez, s.f.).

Desde entonces MediaPipe se ha convertido en una plataforma popular y ampliamente utilizada para el desarrollo de aplicaciones de visión por computadora, seguimiento de objetos, reconocimiento facial y corporal, detección de gestos y muchos otros casos de uso.

Es un módulo de la plataforma de procesamiento de datos en tiempo real que permite la estimación precisa de la pose humana en tiempo real a partir de imágenes de video en tiempo real o pregrabadas. El módulo utiliza algoritmos de aprendizaje profundo para detectar y rastrear las posiciones de las articulaciones del cuerpo humano y generar una representación en 3D de la pose del cuerpo. Utiliza redes neuronales profundas para detectar las articulaciones, en donde estas redes se entrenan con grandes conjuntos de datos etiquetados para que pueda aprender patrones y características comunes de las poses humanas. Una vez entrenadas las redes pueden procesar nuevas imágenes de video en tiempo real y estimar la pose del cuerpo humano con alta precisión (Fernandez, s.f.).

MediaPipe es capaz de realizar el seguimiento de varias personas en un solo cuadro de video y puede procesar múltiples cuadros de video en tiempo real. El módulo se puede utilizar en una amplia variedad de aplicaciones como juegos, realidad virtual, animación, deportes, salud y fitness, y más.

Este módulo predice explícitamente dos puntos clave virtuales adicionales que describen firmemente el centro del cuerpo, la rotación y la escala como un círculo. Inspirado por el hombre Vitruvio de Leonardo se predice el punto medio de las caderas de una persona, el radio de un círculo que circunscribe a toda la

persona y el ángulo de inclinación de la línea que conecta los puntos medios del hombro y la cadera (Fernandez, s.f.), así como se muestra en la Figura 30.

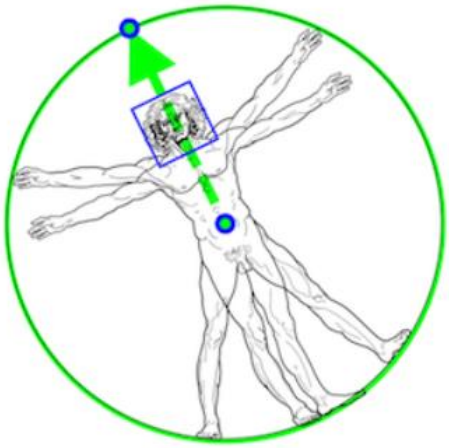

Figura 30 El hombre vitruvio de Leonardo (PÉREZ, 2020).

Este modelo ML de MediaPipe pose predice la ubicación de 33 puntos de referencia, como se muestra en la Figura 31.

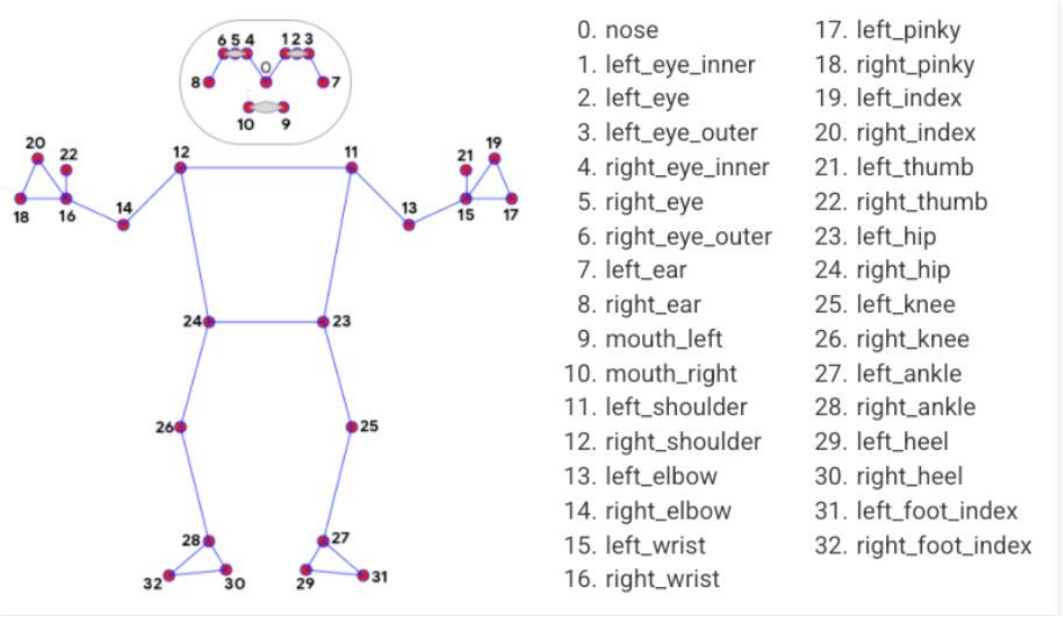

Figura 31 Marcadores de salida del modelo de ML de MediaPipe (Gerges H. Samaan, 2022).

Para instalar MediaPipe en Python, se deben seguir los siguientes pasos:

- Abrir una terminal o línea de comandos en el sistema operativo
- Verificar tener instalado pip, el administrador de paquetes de Python. Se puede verificar escribiendo el siguiente comando 'pip -- version' en la terminal. Si no está instalado, se puede descargar e instalarlo desde 'https://pip.pypa.io/en/stable/installation/'.

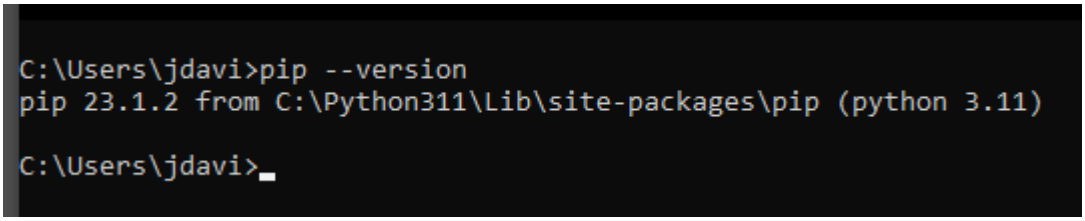

Figura 32 Versión de pip (Propia del autor)

- Ejecutar el comando 'pip install mediapipe' en la terminal. Esto descargara e instalara el paquete de MediaPipe y todas sus dependencias en el sistema. Tendrá que aparecer algo así como se muestra en la Figura 33.

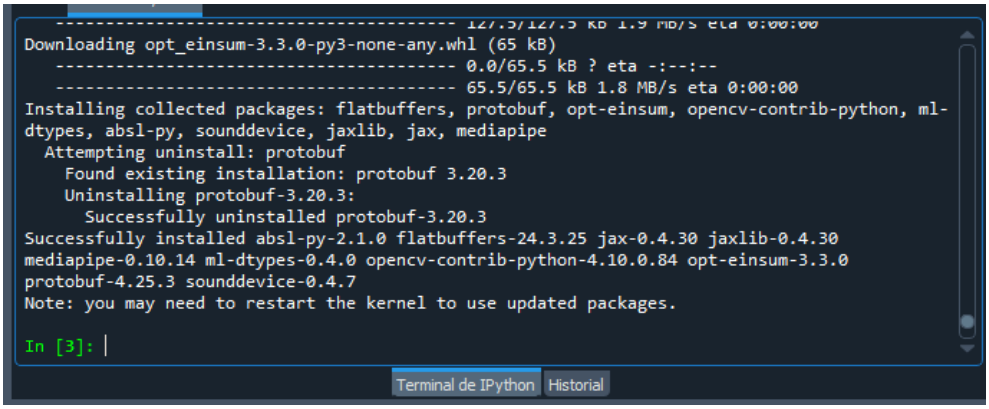

Figura 33 Instalación de MediaPipe (Propia del autor).

- Una vez que se complete la instalación, se puede importar la biblioteca en el código de Python utilizando la declaración 'import mediapipe'

4.4 OpenCV

OpenCV es una biblioteca de software de visión por computadora y aprendizaje automático de código abierto. Fue desarrollado inicialmente por Intel en 1999 y

ahora es mantenida por la comunidad de código abierto. OpenCV proporciona una amplia variedad de algoritmos y herramientas para procesamiento de imágenes y video en tiempo real (Navarro, 2024).

OpenCV es compatible con varios lenguajes de programación, incluyendo C++, Python, Java y MATLAB. También es compatible con diferentes sistemas operativos lo que lo hace adecuado para diferentes plataformas.

OpenCV es una herramienta poderosa y versátil que en este proyecto nos permitirá realizar algunas tareas, como el mostrar y leer imágenes, obtener imágenes de un video en tiempo real y procesarlas al mismo tiempo con la aplicación de algunos filtros y la redimensión de estas mismas imágenes.

Para instalar OpenCV se necesita colocar el siguiente comando, y en este caso instalaremos una versión especifica de OpenCV. Así como se muestra en la Figura 34.

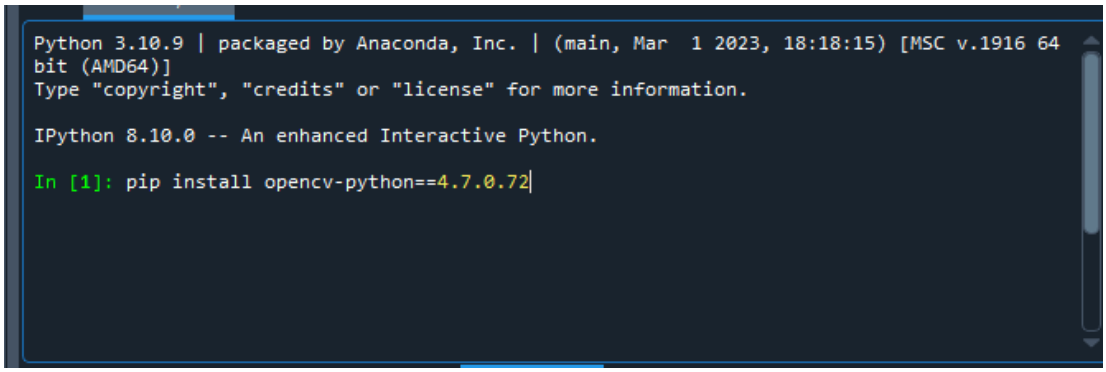

Figura 34 Instalación de OpenCV (Propia del autor).

Una vez instalada aparecerá algo parecido a lo que se muestra en la Figura 35.

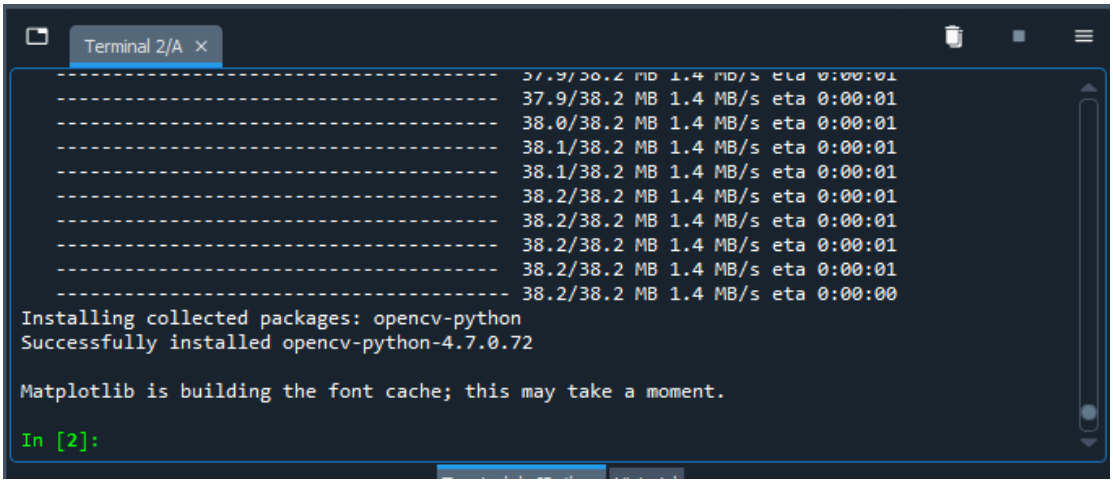

Figura 35 Finalización de la instalación de OpenCV (Propia del autor).

Esta herramienta permite cargar las imágenes al programa, así como se observa a continuación. En la Figura 35 se puede observar que se está indicando en una variable, la ruta de una imagen, en la línea 12 se puede observar que se llama a la función de imread de OpenCV para poder cargar la imagen y tenerla en memoria, en la línea 14 se llama a otra función llamada imshow para poder visualizar la imagen en una nueva ventana, la línea 16, espera a que se presione una una tecla para poder cerrar la ventana de la imagen, y la línea 18 cierra todas las ventanas que se hayan creado.

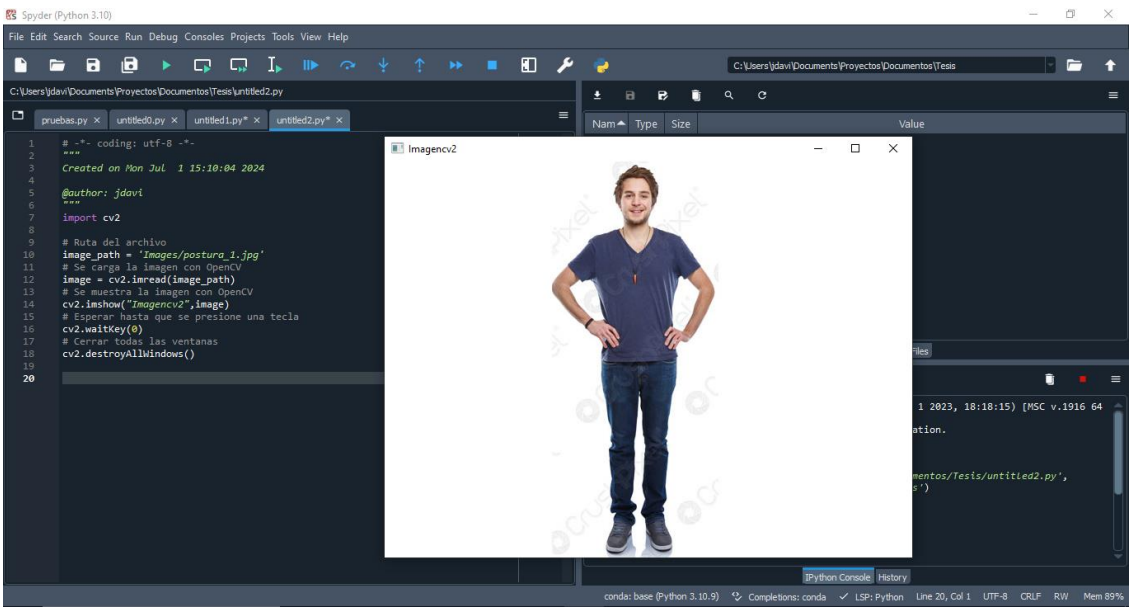

Figura 35 Cargar y mostrar imagen con OpenCV (Propia del autor).

Se puede cargar una imagen existente, se puede crear una matriz de imagen con numpy que es una librería de Python. Como se observa OpenCV permite trabajar directamente en la matriz de valores de la imagen en sus 3 canales (RGB), así se puede crear cualquier dibujo o forma con OpenCV y permite visualizar su forma en imagen. Así como se ve en la Figura 36, la imagen y a un lado la matriz de 3 dimensiones con valores de 0.

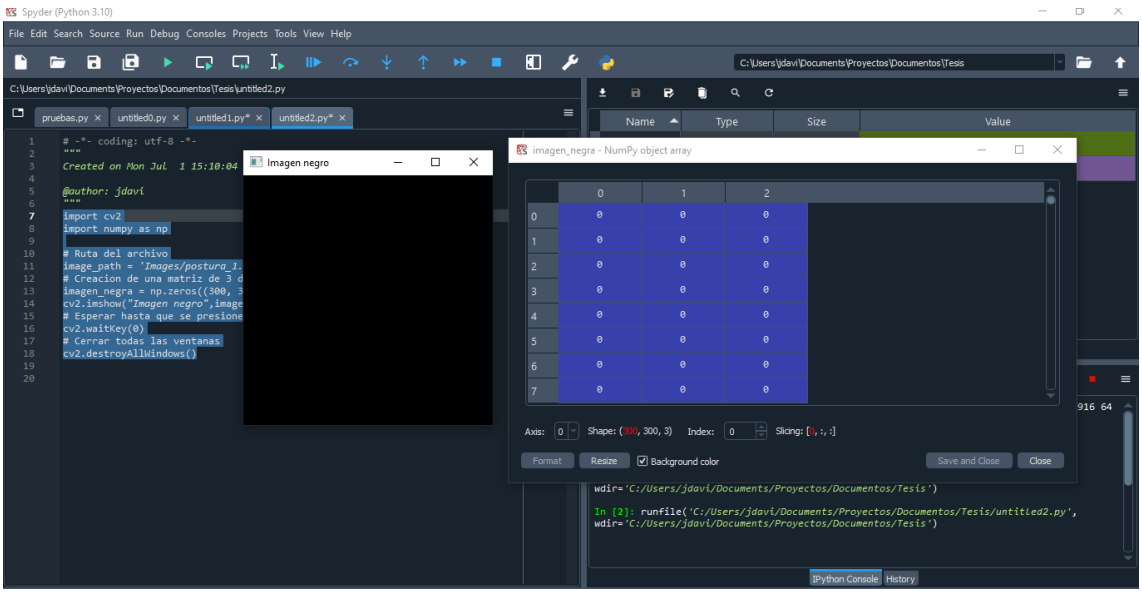

Figura 36 Creación de una imagen a partir de una matriz de 0 (Propia del autor).

Capítulo 5 Desarrollo y pruebas

5. Desarrollo y pruebas

Se deben instalar las librerías necesarias para el funcionamiento del presente proyecto; los requisitos para poder instalar y usar la librería de mediapipe son las siguientes:

- Python: versión 3.8 - 3.11.
- PIP: versión 20.3+.
- OpenCV: versión 4.7.0.72.

Las pruebas se realizaron en una computadora portátil con las siguientes especificaciones:

- Modelo del sistema: HP Pavilion Laptop 15-cw0xxx.
- Procesador: AMD Ryzen 3 2300u with Radeon Vega Mobile Gfx (4CPUs), ~2.0GHz.
- Memoria 12288MB RAM.
- Sistema operativo: Windows 10 Home Single Language 64 bits.
- Cámara: 0.9MP 16:9 (1280*720) .

Primero se importaron las bibliotecas necesarias para el procesamiento de imágenes y videos. NumPy es utilizada para operaciones matemáticas, operator para operaciones de ordenamiento, cv2 para el procesamiento de imágenes y videos, MediaPipe para la detección de posturas humanas, y time para gestionar el tiempo de ejecución. Así como se observa en la Figura 37.

```
import numpy as np
import operator
import cv2
import mediapipe as mp
import time
```

Figura 37 Importación de bibliotecas (Propia del autor).

En seguida se definió una lista de nombres de imágenes, las cuales serán las imágenes de este juego, Figura 38.

```
15
16      #Imagen de prueba y parametros
17      listImages = ["postura_1",
18                    "postura_2",
19                    "postura_3",
20                    "postura_4",
21                    "postura_5",
22                    "postura_6",
23                    "postura_7",
24                    "postura_8",
25                    "postura_9",
26                    "postura_10",]
27
```

Figura 38 Lista de imágenes (Propia del autor).

Las imágenes por utilizar se muestran en la Figura 39.

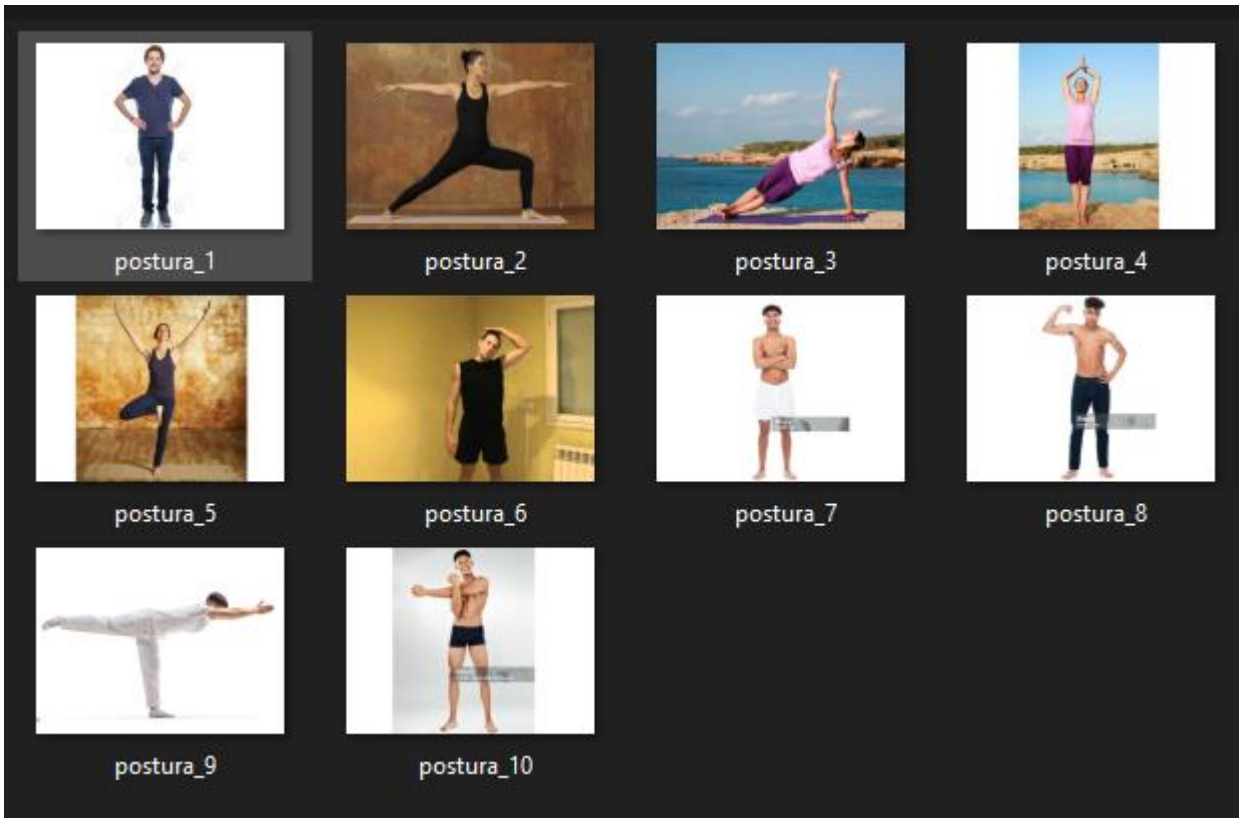

Figura 39 Conjunto de imágenes para el juego (Propia del autor).

Se creo una clase llamada RunGame con su respectiva función de __init__ pasando los siguientes parametros, donde img es una imagen estática, la cual es la imagen para comparar con la de la cámara web del equipo utilizado, los siguientes parametros numPuntos y parámetro, sirven para definir qué tan preciso sea nuestro juego, Figura 40.

```
class RunGame:
    def __init__(self, img, numPuntos, parametro):
        self.img = img
        self.numPuntos = numPuntos
        self.parametro = parametro
        self.pose = False
```

Figura 40 Clase principal (Propia del autor).

Dentro de esta clase se crearon 2 funciones de ayuda, la función que se muestra en la Figura 41, ayuda para darle un efecto de acercamiento a nuestra imagen

```
def zoom(self,normal):
    height, width, _ = normal.shape
    for i in np.arange(0.1,1.1,0.01):
        imgout = normal.copy()
        M = cv2.getRotationMatrix2D((height//2,width//2),0,i)
        imgout = cv2.warpAffine(normal,M,(width,height))
        cv2.imshow("NEw ",imgout)
        cv2.waitKey(10)
```

Figura 41 Función zoom (Propia del autor).

La segunda función ayuda a colocar un borde en el visor de la cámara, en donde se colocará rojo si el algoritmo detecte menos del porcentaje de puntos de referencia y verde donde detecte un porcentaje mayor al 75%. Así como se muestra el código en la Figura 42 y en la Figura 43 se muestra un cuadrado de color verde.

```
def cuadrado(self,frame,color,width,height):
    cv2.line(frame, (width,0),(0,0),color, 10)
    cv2.line(frame, (width,0),(width,height),color, 10)
    cv2.line(frame, (width,height),(0,height),color, 10)
    cv2.line(frame, (0,height),(0,0),color, 10)
    cv2.imshow('MediaPipe Pose', cv2.flip(frame, 1))
```

Figura 42 Función cuadrado (Propia del autor).

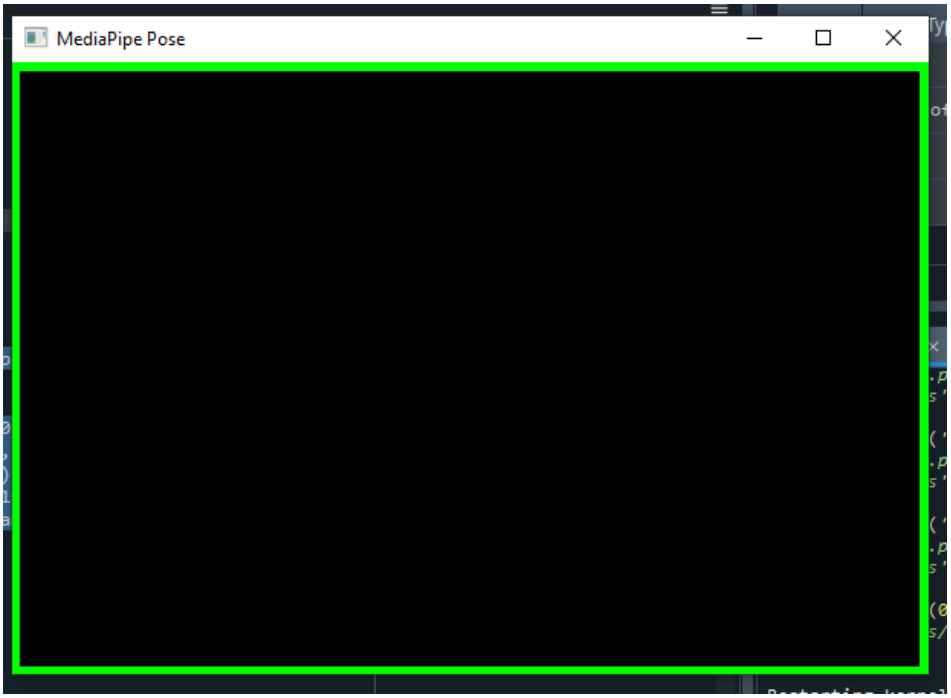

Figura 43 Cuadrado verde hecho con Python (Propia del autor).

Por último, se creó una función llamada solution que contiene la lógica principal del juego, Figura 44. Utiliza MediaPipe para detectar y rastrear puntos clave de la postura humana en una imagen estática y en un video en tiempo real. Las líneas subsiguientes inicializan algunas herramientas de dibujo y la detección de posturas de MediaPipe.

```
def solution(self):
    #Herramienta para pintar los puntos
    mp_drawing1 = mp.solutions.drawing_utils;
    mp_pose1 = mp.solutions.pose;
    mp_drawing_styles1 = mp.solutions.drawing_styles
    with mp_pose1.Pose( static_image_mode = True) as pose1:
        datos_imagen_estatica=[]
        image1 = cv2.imread(self.img)
        image2 = image1.copy()
        self.zoom(image2)
        altura, ancho, _ = image2.shape
        image_rgb = cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
        results1 = pose1.process(image_rgb)
```

Figura 44 Función solution (Propia del autor).

El fragmento de código de la Figura 44 se encarga de realizar varias tareas relacionadas con el procesamiento de imágenes utilizando la biblioteca MediaPipe. En primer lugar, se importan las utilidades necesarias de MediaPipe. Luego, se

configura el modelo a utilizar dentro de un contexto definido por la palabra reservada 'with'. Posteriormente, se carga una imagen específica utilizando la función de lectura correspondiente y se invoca la función 'zoom' para aplicar una animación a la imagen. Finalmente, se pasa la imagen en escala de grises al modelo y se almacenan los resultados obtenidos en la variable 'results1'.

En la Figura 45 se muestra la figura y nos muestra los puntos que detecta el modelo de MediaPipe Pose y en la parte de la derecha de la imagen se muestran las coordenadas de los puntos encontrados, si en dado caso no se encuentre un punto en la imagen se encontrará dentro de visibility un valor muy cercano a 0.000000000.

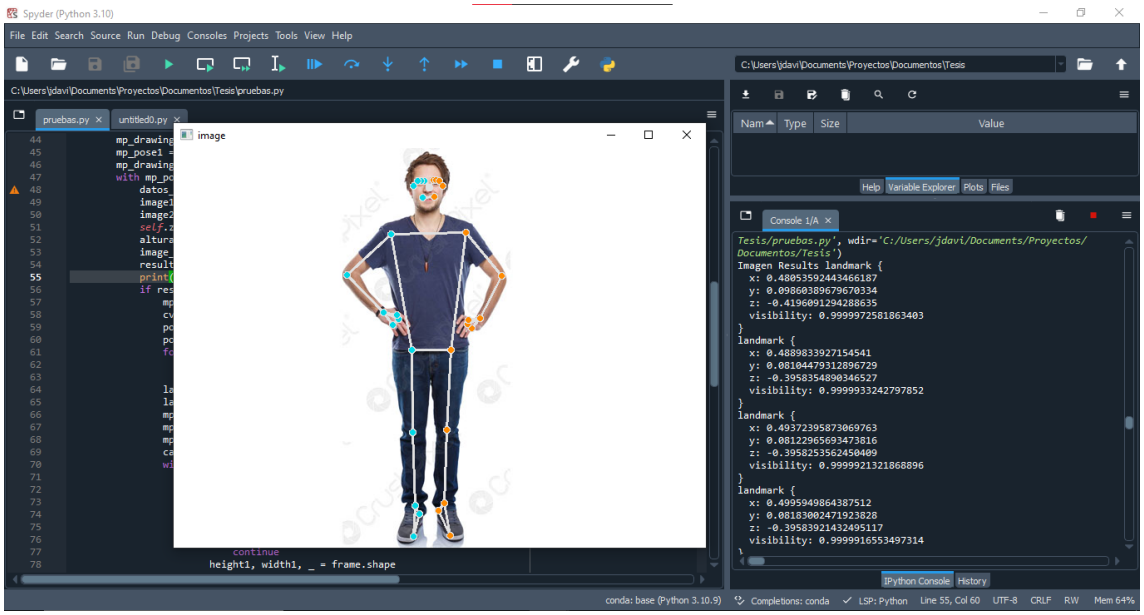

Figura 45 Resultado del modelo de MediaPipe Pose (Propia del autor).

```
if results1.pose_landmarks is not None:
    mp_drawing1.draw_landmarks(image2,results1.pose_landmarks,mp_pose1.POSE_CONNECTIONS,landmark_dr
    cv2.imshow("image", image2);
    pointsx = {}
    pointsy = {}
    for i in range(0,32):
        pointsx[i] = int(results1.pose_landmarks.landmark[i].x * ancho)
        pointsy[i] = int(results1.pose_landmarks.landmark[i].y * altura)
    landmarkx_sort = sorted(pointsx.items(), key=operator.itemgetter(1), reverse=False)
    landmarky_sort = sorted(pointsy.items(), key=operator.itemgetter(1), reverse=False)
```

Figura 46 Código (Propia del autor)

En este otro fragmento de código como se muestra en la Figura 46, se realiza una validación para asegurar que la variable 'results1' no esté vacía. En caso de

cumplirse esta condición, se procede a utilizar las herramientas de pintado proporcionadas por MediaPipe para dibujar la imagen correspondiente, aunque solamente como una referencia visual. Luego, se procede a sacar las coordenadas de los puntos, ya que estos se retornan en un formato específico, como se muestra en la imagen adjunta. Posteriormente, se ordenan estos objetos teniendo en cuenta tanto las coordenadas como el índice del elemento. Esta acción es necesaria debido a que MediaPipe retorna una lista de objetos, donde cada índice corresponde a una parte del cuerpo predefinida. Al ordenar estos objetos, se utiliza la información de píxeles junto con el índice para determinar qué parte del cuerpo ha sido detectada. Así como se muestra en la Figura 47.

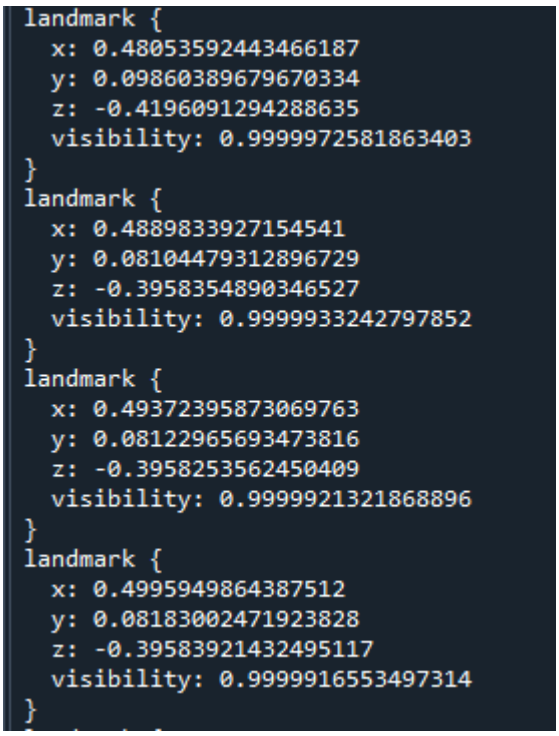

Figura 47 Datos que regresa MediaPipe (Propia del autor).

Después de completar este proceso con la imagen estática, se procede a realizar el mismo procedimiento, pero ahora con un video. En este caso, se utilizará el video proveniente de la cámara web en tiempo real.

Se inicializa una captura de video desde la cámara web utilizando OpenCV con el fragmento de código que se muestra en la Figura 48

```
mp_drawing = mp.solutions.drawing_utils;
mp_pose = mp.solutions.pose;
mp_drawing_styles = mp.solutions.drawing_styles
cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
```

Figura 48 Herramientas de MediaPipe (Propia del autor).

Se inicializa la detección de posturas en el video utilizando la clase Pose de Mediapipe.

Se puede observar en la Figura 49 la configuración que se usara en MediaPipe.

```
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5,static_image_mode = False) as pose:
```

Figura 49 Clase pose (Propia del autor).

Se inicia un bucle que procesa continuamente los fotogramas de la cámara web y se dibuja un cuadrado en cada fotograma. Así como se muestra en la Figura 50

```
while cap.isOpened():
    success, frame = cap.read()
    frame.flags.writeable = False
    if not success:
        print("Ignorando el marco de la cámara vacía.")
        continue
    height1, width1, _ = frame.shape
    self.cuadrado(frame,(0,0,255),width1,height1)
```

Figura 50 Activar cámara web (Propia del autor).

Se procesa cada fotograma para detectar y rastrear puntos clave de la postura. Esto se hace con el siguiente fragmento de código mostrado en la Figura 51.

```
framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = pose.process(framergb)
```

Figura 51 Resultados de la clase pose (Propia del autor).

Se comparan las coordenadas de los puntos clave detectados en el fotograma con las coordenadas de los puntos clave detectados en la imagen estática para determinar si se ha completado una postura específica, así como se muestra en la figura 52. Se dibujan cuadrados de diferentes colores alrededor de las partes del cuerpo detectadas según el resultado de la comparación.

```
91
92  for i in range(0,32):
93      if( landmarkFramex_sort[i][0] >= (landmarkx_sort[i][0] - self.numPuntos)   and landmarkFramex_sort[i][0] <= (landmarkx_sor
94          print("Entro al if1")
95          if(  landmarkFramey_sort[i][0] >= (landmarky_sort[i][0] - self.numPuntos) and landmarkFramey_sort[i][0] <= ( landmarky_
96              print("Entro al if2")
97              self.cuadrado(frame,(177,3,252),640,480)
98              if( landmarkFramex_sort[i][1] >= (landmarkx_sort[i][1] - self.parametro) and  landmarkFramex_sort[i][1] <= (landma
99                  print("Entro al if3")
100                 self.cuadrado(frame, (3,186,252),640,480)
101                 if( landmarkFramey_sort[i][1] >= (landmarky_sort[i][1] - self.parametro) and landmarkFramey_sort[i][1] <= (lan
102                     print("Entro al if4")
103                     self.cuadrado(frame, (0,255,0),640,480)
104                     suma = suma +1
105                     break
106
107 #for
```

Figura 52 Comparación de coordenadas (Propia del autor).

Si se detecta al menos 25 coordenadas iguales dentro de un rango de precisión en un fotograma, se imprime un cuadro color verde y se espera 3 segundos antes de salir del bucle como se muestra en la Figura 53.

```
if(suma >= 25):
    self.cuadrado(frame, (0,255,0),640,480)
    time.sleep(3)
    break
```

Figura 53 Cierre de la cámara web (Propia del autor).

Finalmente, se crea una instancia de la clase RunGame para cada imagen de la lista listImages, y se llama al método solution para ejecutar el sistema de reconocimiento de posturas en cada imagen. Después de eso, se libera la instancia como se muestra en la Figura 54.

```
for x in listImages:
    instancia = RunGame("Images/" + x + ".jpg" , 2, 15)
    instancia.solution()
    instancia = None
```

Figura 54 Recorrido de las imágenes (Propia del autor).

A continuación se muestra el funcionamiento completo del juego.

En la Figura 55 se observan dos ventanas, del lado izquierdo es la imagen a copiar, y del lado derecho es la cámara web de una computadora portátil, se observa un cuadrado verde de la ventana derecha ya que como se validó que la pose sea muy parecida a la de la primera imagen, entonces el programa congela la imagen por 3 segundos y avanza a la siguiente imagen.

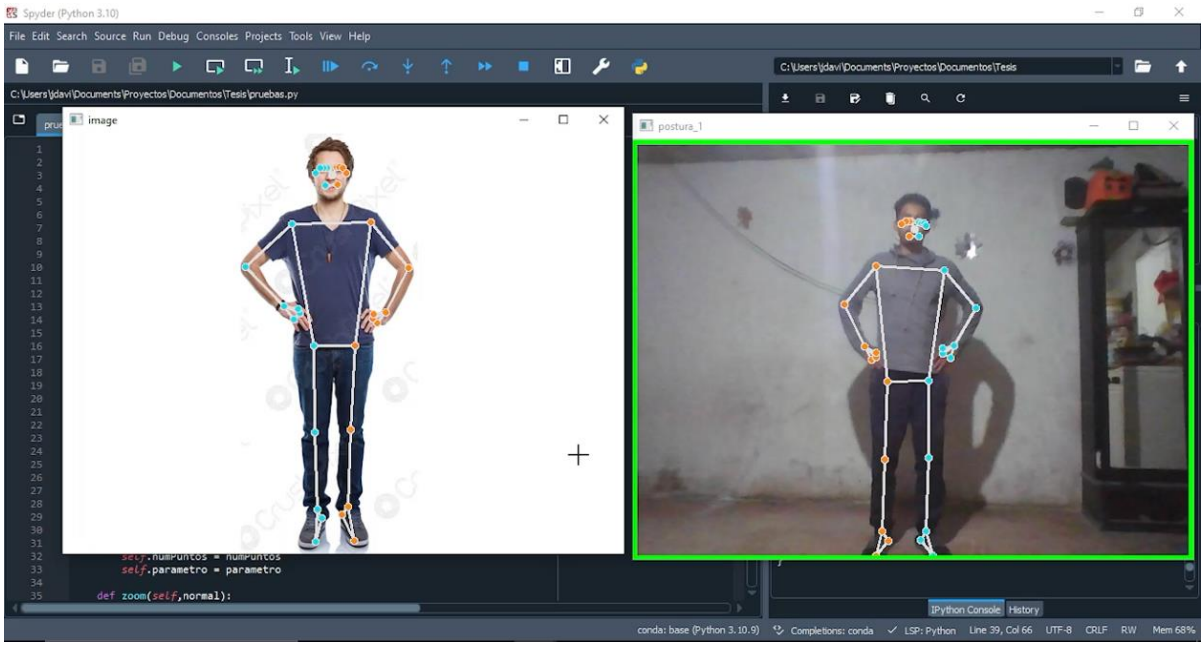

Figura 55 Postura 1 (Propia del autor).

En la Figura 56 se observan las mismas dos ventanas, pero con una pose diferente, en este caso se observa un contorno morado, esto quiere decir que la pose aun no es lo suficientemente parecida, como para pasar a la siguiente imagen, entonces se deberá seguir intentando copiar la imagen para que permita avanzar y se pueda visualizar la siguiente imagen.

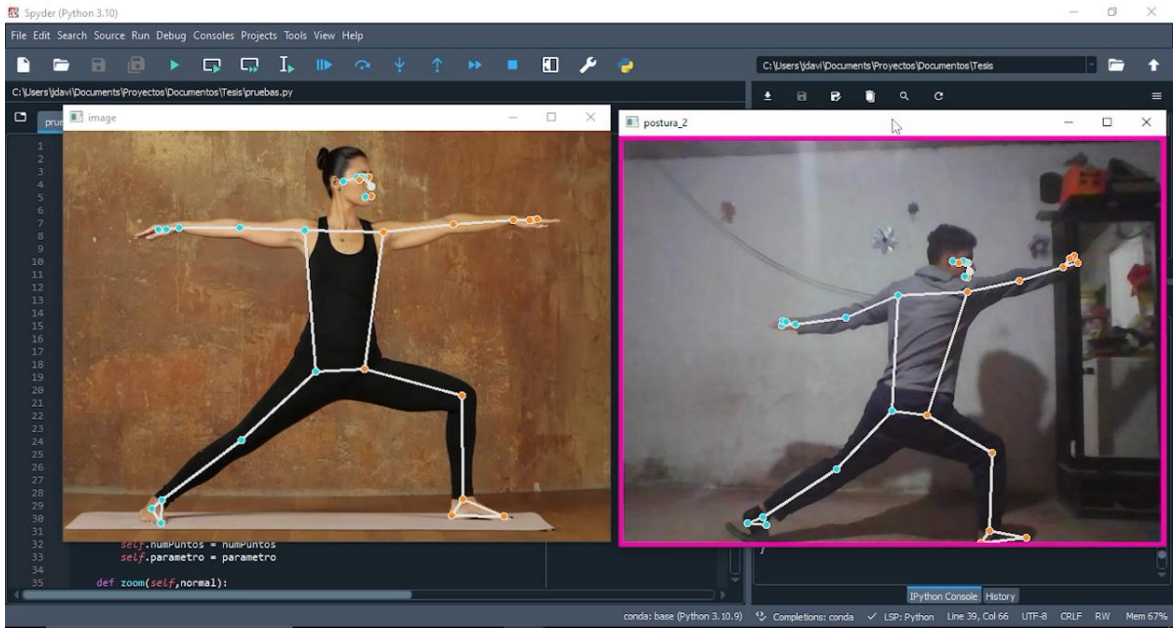

Figura 56 Postura 2 (Propia del autor).

En la figura 57 se muestra correctamente la postura de la figura anteriormente mostrada, con lo cual se presenta un cuadrado verde.

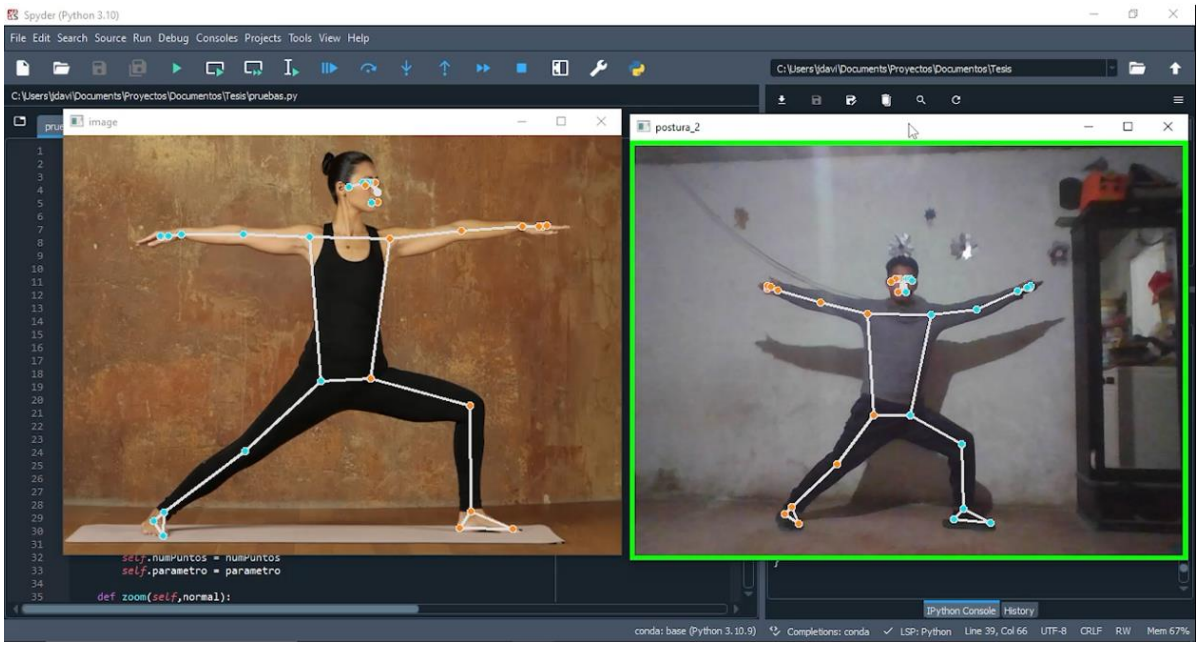

Figura 57 Postura 2 correcta (Propia del autor).

En la Figura 58 se muestra la postura tres, en este caso se muestra un contorno de color morado ya que no se pudo realizar una pose similar, en este caso que se complicó la postura y no nos deja avanzar, lo que se puede hacer es que con la tecla “esc” se permite omitir esa pose y pasar a la siguiente.

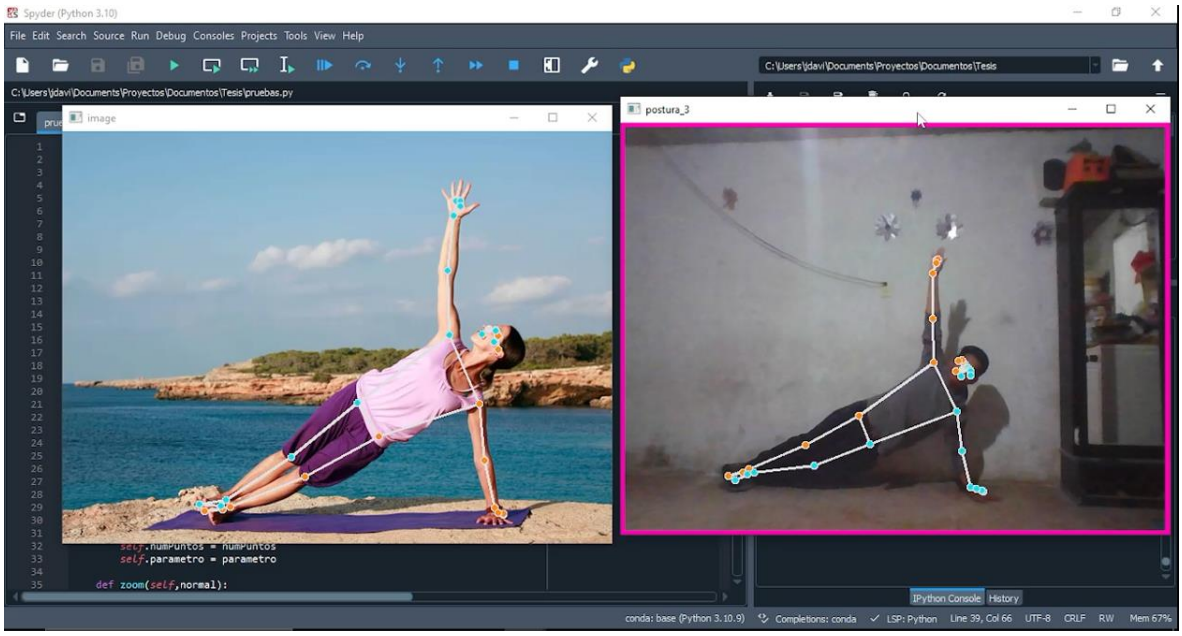

Figura 58 Postura 3 Incorrecta (Propia del autor).

En la Figura 59 se muestra que se avanzó de postura, por lo cual se ha llegado hasta la postura o pose 4 en la cual si se pudo hacer una pose similar por lo que pinta el contorno verde.

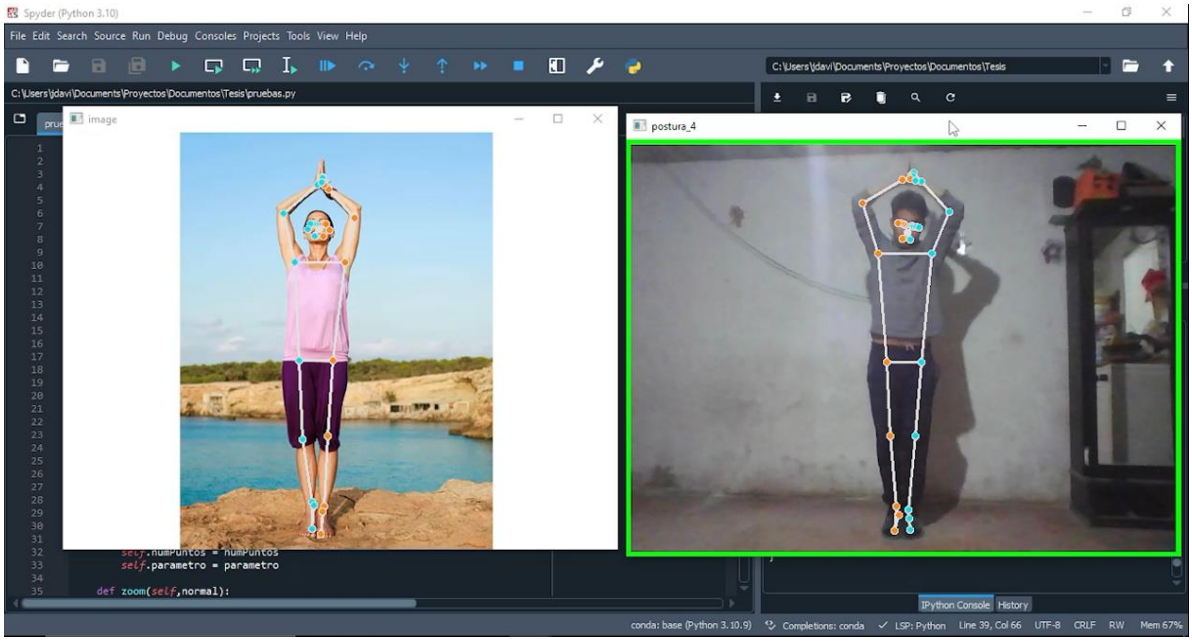

Figura 59 Postura 4 (Propia del autor).

En la Figura 60 se muestra la postura 5, que esta pose esta fácil de copiar por lo que se procede a avanzar a la siguiente imagen con mucha facilidad.

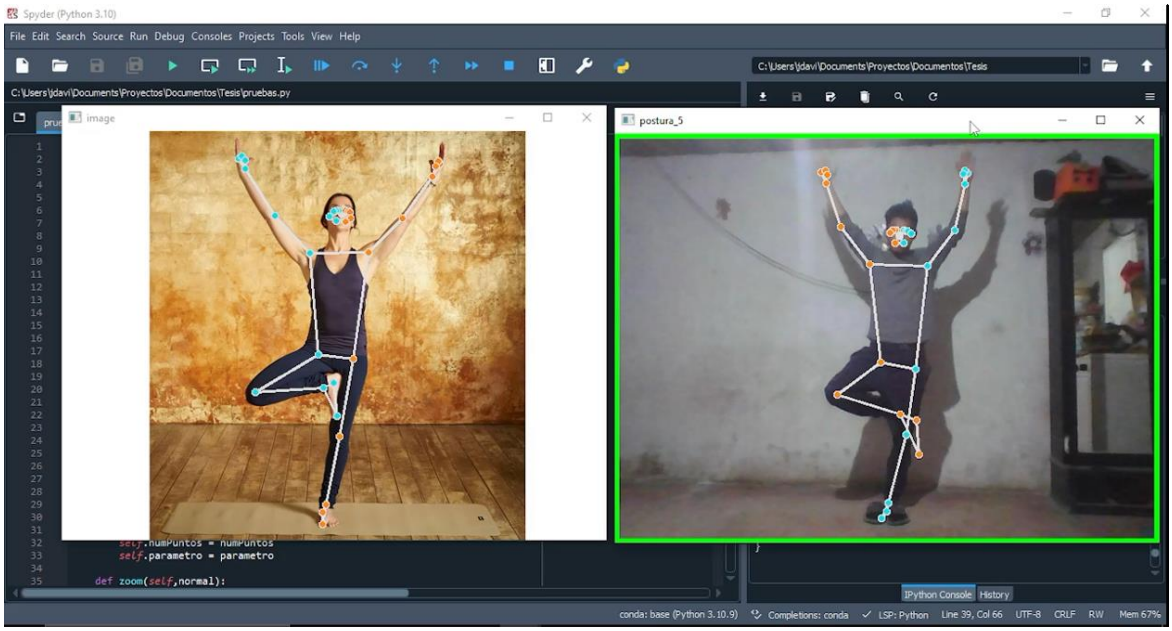

Figura 60 Postura 5 (Propia del autor).

En la Figura 61 se muestra una pose, pero en este caso no es de cuerpo completo, pero se colocó para que se pueda visualizar que aunque no este completa la pose del lado izquierdo, se puede aun así copiar la pose siempre y cuando la mayoría de puntos coincidan, en este caso se observa que la pose del lado izquierdo es la que manda cuantos puntos se compararan.

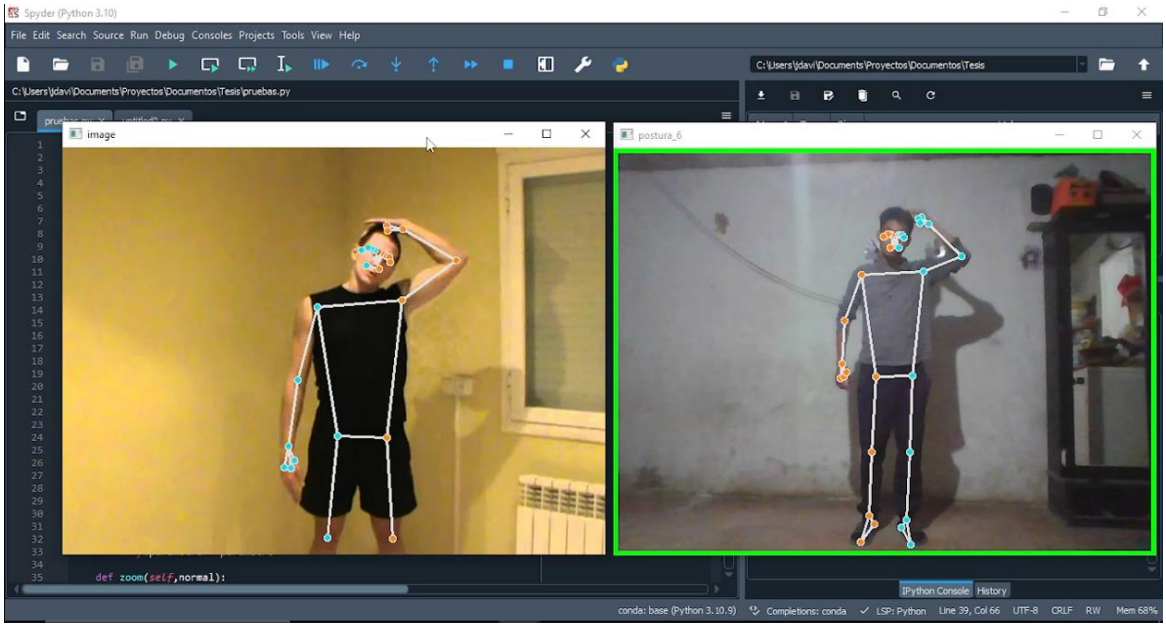

Figura 61 Postura 6 (Propia del autor).

En la Figura 62 se puede observar como la postura no es tan similar, bueno en este caso los brazos no son tan similares las posturas, pero eso tiene que ver con los números de puntos a comparar, en este caso llego a los puntos necesarios para tomar la decisión de que son similares ambas posturas.

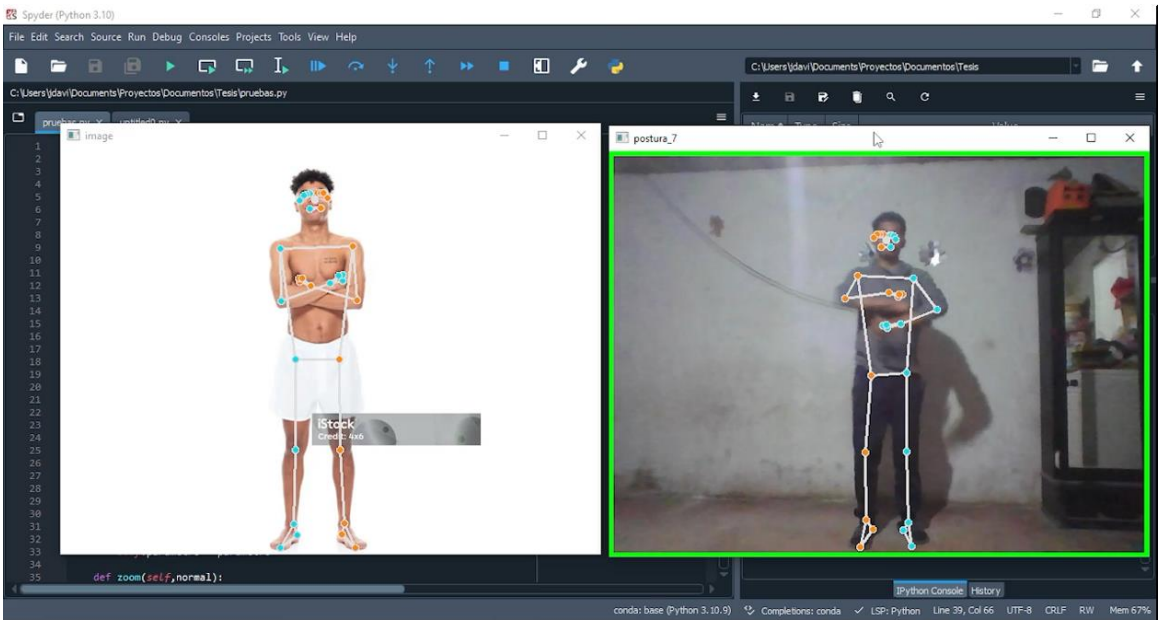

Figura 62 Postura 7 (Propia del autor).

En la figura 63 se muestra la postura 8 la cual es fácil de copiar, por lo que se avanzó a la siguiente postura.

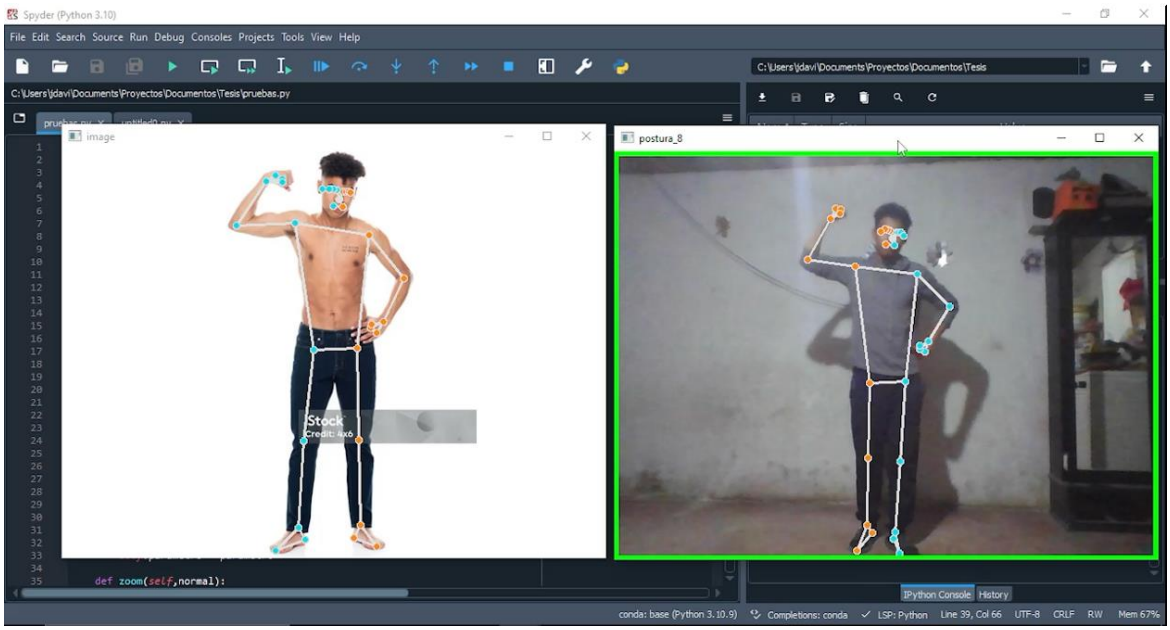

Figura 63 Postura 8 (Propia del autor).

La Figura 64 muestra a la postura 9, la cual en este caso no se pude realizar una postura similar por lo que se omitió esa imagen presionando la tecla “esc”.

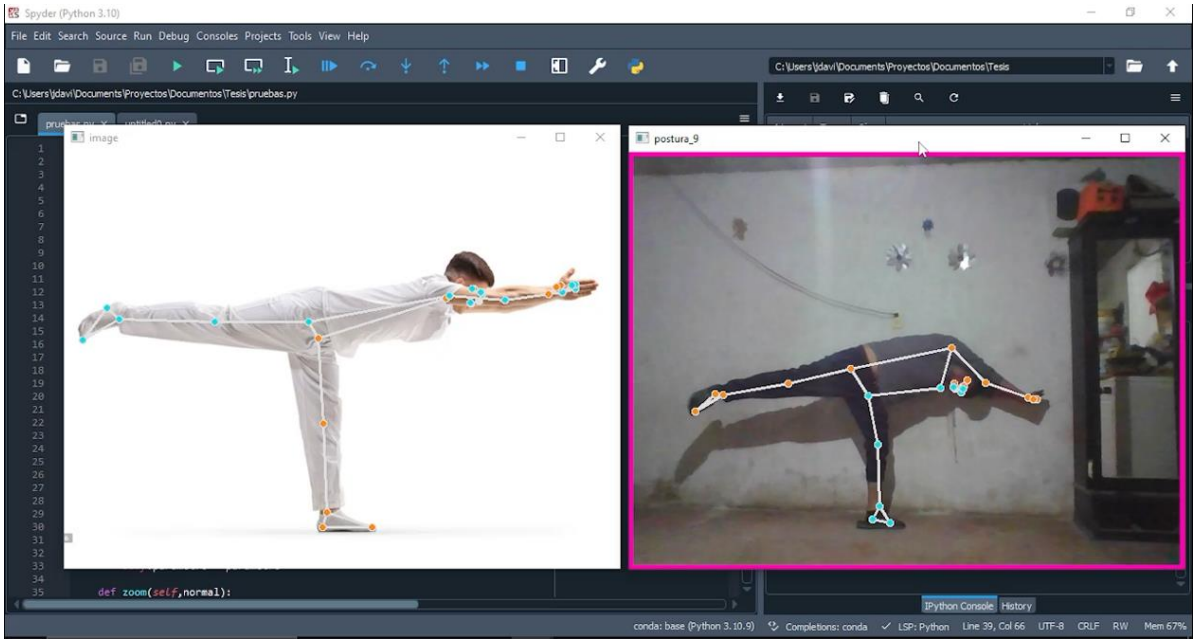

Figura 64 Postura 9 (Propia del autor).

La Figura 65 esta fácil de copiar por lo que esta es la última postura para copiar, y con esto se muestra el funcionamiento de principio a fin de este juego de posturas.

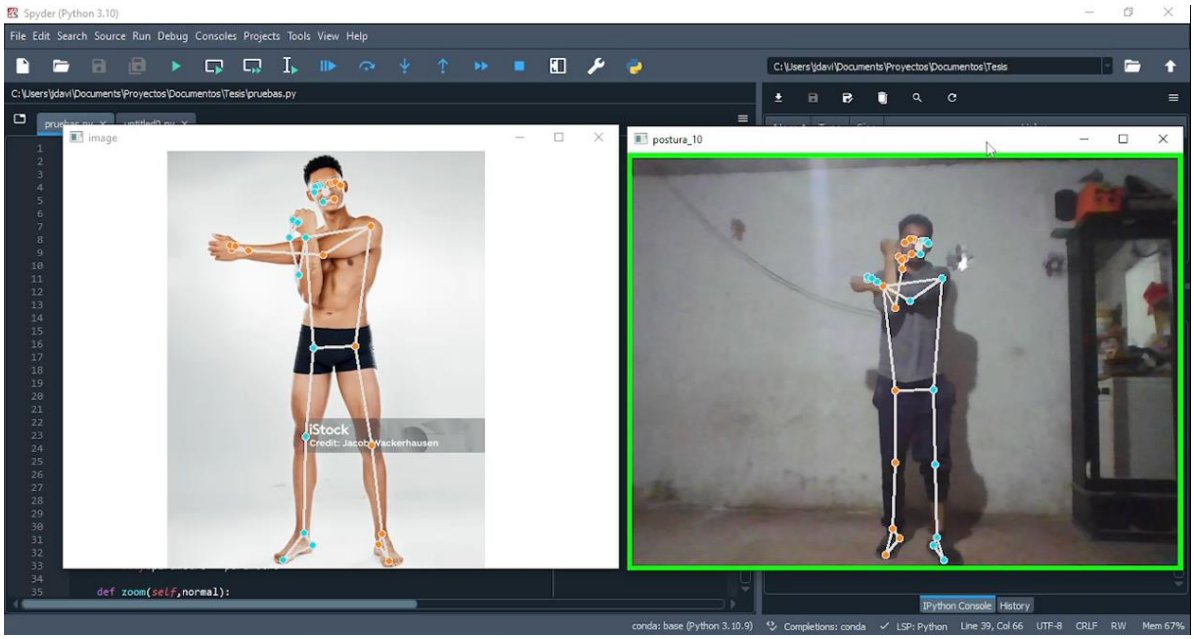

Figura 65 Postura 10 (Propia del autor).

Capítulo 6 Conclusiones y trabajo a futuro

6. Conclusiones y trabajo a futuro

En conclusión, de esta tesis se puede mencionar que es un proyecto interesante para comenzar en el mundo de la visión artificial sin tener conocimientos previos o muy avanzados en el mundo de la programación.

Gracias a la librería de Python llamada MediaPipe y OpenCV se pueden crear un sinfín de ideas detectando las poses de una persona y analizando esa información a nuestra conveniencia.

En especial puede ser muy útil en la parte de hacer ejercicio, para poder detectar que se está haciendo correctamente un tipo de ejercicio sin necesidad de tener un entrenador en el mismo lugar, si no que la computadora este monitoreando tu ejercicio y te advierta si detecta un mal ejercicio o una mala pose que puede provocar alguna lesión.

En la parte de medicina es una rama en donde es de gran importancia ya que con el reconocimiento de posturas se puede evaluar la postura de un cuerpo humano para poder identificar problemas en la alineación de la columna vertebral y o en extremidades.

De igual manera puede ayudar en la parte de rehabilitación y recuperación física desde casa, ya que se podría monitorear el progreso del paciente desde cualquier lugar. Y podría ser mui útil integrando un sistema de reconocimiento de poses a dispositivos diseñados para asistir a personas con discapacidades físicas.

Se encontraron algunos puntos importantes a mejorar en este juego de posturas, específicamente en los siguientes puntos:

- Interfaz gráfica.

- Creación del proyecto multiplataforma.

A interfaz gráfica se refiere a no tener que estar corriendo el programa desde nuestro IDE de programación, hay varias herramientas que permiten trabajar con una interfaz gráfica con Python como por ejemplo PyQT, Kyvi, WXPython etc. Estos son algunos frameworks que permiten trabajar con la parte visual de este programa, por ejemplo, para este proyecto se puede generar una vista con un menú donde se pueda subir más imágenes de poses complicadas para subir el nivel del juego, otra vista con un botón para poder terminar, pausar o continuar el juego, y por último una vista donde se muestre la puntuación obtenida.

Esta parte puede ser muy importante, porque a pesar de que el juego esta realizado para cualquier persona sin importar edad, puede ser más llamativa para niños pequeños, lo cual no están tan familiarizados con usar una computadora, y mucho menos correr un programa desde un IDE de programación, así que con esta interfaz gráfica se limitarían a con un solo click empezar a jugar.

El segundo punto encontrado para mejorar es la parte de poder hacer que este juego sea multiplataforma, que significa esto; significa que nuestro juego pueda correr tanto en computadora, dispositivo móvil o desde un navegador web.

Esto suena complicado pero gracias a que MediaPipe nos ofrece la ventaja de poder correr su modelo de detección de poses en Android, Web, Python e IOS, podemos fácilmente migrar este juego a cualquiera de estas plataformas sin tener que preocuparse por la compatibilidad del modelo, o en dado caso que se pueda pasar este programa a cualquiera de estas plataformas gracias a los frameworks ya mencionados anteriormente, que permiten crear a la vez aplicaciones creadas con código Python a aplicaciones compatibles para el navegador web e inclusive para Android e IOS.

Referencias

Ali, M. (Abril de 2024). *Introducción a las funciones de activación en las redes neuronales*. Obtenido de https://www.datacamp.com/es/tutorial/introduction-to-activation-functions-in-neural-networks

Aplicada, D. d. (s.f.). *Redes Neuronales Convoluciones*. Obtenido de https://dcain.etsin.upm.es/~carlos/bookAA/05.7_RRNN_Convoluciones_CIFAR_10_INFORMATIVO.html#:~:text=Capa%20de%20Agrupaci%C3%B3n%20(Pooling),_Es%20com%C3%BAn%20insertar&text=La%20capa%20de%20agrupaci%C3%B3n%20funciona,H%201%20%C3%97%20D%201%20)%20.

AWS. (s.f.). *AWS AMAZON*. Obtenido de https://aws.amazon.com/es/what-is/python/#:~:text=Guido%20Van%20Rossum%2C%20un%20programador,durante%20las%20vacaciones%20de%20Navidad.

Baker, S. (27 de Noviembre de 2010). *EA Sports Active 2 Xbox 360 Kinect Review*. Obtenido de https://www.tweaktown.com/gaming/3683/ea_sports_active_2_xbox_360_kinect_review/index.html

Brajovic, F. (14 de Mayo de 2021). *Imágenes térmicas: ¿Qué usos tienen en la ciencia?* Obtenido de https://www.cromtek.cl/2021/05/14/imagenes-termicas-que-usos-tienen-en-la-ciencia/

Brutti, F. (10 de 8 de 23). *Redes Neuronales: Aprendizaje y Resolución de Problemas*. Obtenido de https://thepower.education/blog/redes-neuronales

Contributors, S. W. (2023). *https://www.spyder-ide.org/*. Obtenido de https://www.spyder-ide.org/

contributors, W. (22 de Mayo de 2024). *Your Shape: Fitness Evolved*. Obtenido de https://en.wikipedia.org/w/index.php?title=Your_Shape:_Fitness_Evolved&oldid=121478638

DatasCientest. (6 de Diciembre de 2021). *Convolutional Neural Network : definición y funcionamiento*. Obtenido de https://datascientest.com/es/convolutional-neural-network-es

Española, R. A. (2024). *https://dle.rae.es/*. Obtenido de Real Academia Española: https://dle.rae.es/postura?m=form

Fernandez, J. (s.f.). *GitHub*. Obtenido de https://github.com/google-ai-edge/mediapipe/blob/master/docs/solutions/pose.md

Foundation, P. S. (2021). *Clases*. Obtenido de https://docs.python.org/es/3/tutorial/classes.html

Gerges H. Samaan, A. R.-I. (8 de Octubre de 2022). *MediaPipe's Landmarks with RNN for Dynamic Sign Language Recognition*. Obtenido de https://www.mdpi.com/2079-9292/11/19/3228

IBM. (7 de Noviembre de 2023). *IBM*. Obtenido de https://www.ibm.com/mx-es/topics/convolutional-neural-networks

IBM. (2024). Obtenido de https://www.ibm.com/mx-es/topics/supervised-learning

IBM. (s.f.). *IBM*. Obtenido de https://www.ibm.com/mx-es/topics/computer-vision

Ingenieril, S. (2021). *solucioningenieril*. Obtenido de https://solucioningenieril.com/vision_artificial/etapas_de_un_sistema_de_vision

ISO. (2024). *ISO*. Obtenido de https://www.iso.org/es/inteligencia-artificial/redes-neuronales#:~:text=Las%20redes%20neuronales%2C%20tambi%C3%A9n%20conocidas,el%20funcionamiento%20del%20cerebro%20biol%C3%B3gico.

Javi. (24 de MArzo de 2023). *La Función Sigmoide: Una Herramienta Clave en Redes Neuronales*. Obtenido de https://jacar.es/la-funcion-sigmoide-una-herramienta-clave-en-redes-neuronales/

Javi. (24 de Marzo de 2023). *La Tangente Hiperbólica: Función de Activación Potente*. Obtenido de https://jacar.es/la-tangente-hiperbolica-funcion-de-activacion-

potente/#:~:text=La%20Tangente%20Hiperb%C3%B3lica%20se%20implementa,debe%20%C2%ABdisparar%C2%BB%20o%20no.

Lara, C. E. (2 de Diciembre de 2020). *Qué son las imágenes vectoriales y cuál es su uso*. Obtenido de https://hispaprint.com/blog/disenografico/que-son-las-imagenes-vectoriales-y-cual-es-su-uso/

ltucker. (14 de 06 de 2010). *Electronics Arts Home*. Obtenido de https://www.ea.com/en-ca/news/ea-sports-active-2-in-development-for-kinect-for-xbox-360

Martínez, J. (22 de Septiembre de 2018). *DataSmarts*. Obtenido de https://datasmarts.net/es/como-se-representan-las-imagenes-en-una-computadora/

Mathworks.com. (2024). *Convolución*. Obtenido de https://la.mathworks.com/discovery/convolution.html

Mejía, G. A. (Diciembre de 2015). *ResearchGate*. Obtenido de https://www.researchgate.net/figure/Figura-11-Representacion-de-una-imagen-digital_fig1_344371228

Microsoft. (22 de Mayo de 2024). *Tipos de mapas de bits*. Obtenido de https://learn.microsoft.com/es-es/dotnet/desktop/winforms/advanced/types-of-bitmaps?view=netframeworkdesktop-4.8

Navarro, S. (23 de 08 de 2024). *KeepCoding*. Obtenido de https://keepcoding.io/blog/que-son-bibliotecas-de-vision-artificial/

Newton, J. (23 de Noviembre de 2010). *The Biggest Loser: Ultimate Workout Review (Xbox 360)*. Obtenido de https://www.purexbox.com/reviews/xbox-360/the_biggest_loser_ultimate_workout

Page, B. (28 de 4 de 2019). *La vanguardia*. Obtenido de https://www.lavanguardia.com/tecnologia/20190428/461843755133/kinect_accesorio-jugar-convertido-herramienta-artistas.html

PÉREZ, S. (10 de Febrero de 2020). *Detección de poses humanas mediante Deep Learning*. Obtenido de https://blog.damavis.com/deteccion-de-poses-humanas-mediante-deep-learning/

RAMÍREZ, L. (26 de Julio de 2022). *Visión artificial en la industria 4.0: ¿Qué es y qué aplicaciones tiene?* Obtenido de iebsschool: https://www.iebschool.com/blog/vision-artificial-en-la-industria-4-0-que-es-big-data/#:~:text=Para%202023%2C%20se%20espera%20que,%C2%BB%2C%20dependan%20de%20la%20tecnolog%C3%ADa.

Ridgway, J. (5 de 11 de 2022). *¿Qué son los sensores de visión?* Obtenido de https://www.cognex.com/es-mx/blogs/machine-vision/what-are-vision-sensors#:~:text=Los%20sensores%20pueden%20incluirse%20en,otros%20dispositivos%20durante%20la%20producci%C3%B3n.

Rodríguez, D. (23 de Mayo de 2018). *Analytics Lane*. Obtenido de https://www.analyticslane.com/2018/05/23/implementacion-de-una-red-neuronal-desde-cero/

Tech, T. (s.f.). *Función de activación*. Obtenido de https://aiofthings.telefonicatech.com/recursos/datapedia/funcion-activacion

TurboSquid. (23 de Agosto de 2010). *Turbosquid.com*. Obtenido de https://www.turbosquid.com/es/3d-models/female-male-3d-model/552628

Valente, P. P. (2018). Obtenido de https://www.famaf.unc.edu.ar/~pperez1/manuales/cim/cap2.html

values, P. (05 de Mayo de 2024). *Esa.int*. Obtenido de https://www.esa.int/ESA_Multimedia/Images/2008/12/Pixel_values

Anexos

Código fuente.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 26 16:07:43 2022

@author: joel
"""

import numpy as np
import operator
import cv2
import mediapipe as mp
import time
import pygetwindow as gw


#Imagen de prueba y parametros
listImages = ["postura_1",
              "postura_2",
              "postura_3",
              "postura_4",
              "postura_5",
              "postura_6",
              "postura_7",
              "postura_8",
              "postura_9",
              "postura_10",]


class RunGame:
    def __init__(self, img, numPuntos, parametro):
        self.img = img
        self.numPuntos = numPuntos
        self.parametro = parametro
```

```
        self.screen_height = gw.getWindowsWithTitle('')[0]._rect.height
        self.screen_width = gw.getWindowsWithTitle('')[0]._rect.width
        ##screen_width, screen_height =
gw.getWindowsWithTitle('')[0]._rect.width,
gw.getWindowsWithTitle('')[0]._rect.height

    def zoom(self,normal):
        height, width, _ = normal.shape
        for i in np.arange(0.1,1.1,0.01):
            imgout = normal.copy()
            M = cv2.getRotationMatrix2D((height//2,width//2),0,i)
            imgout = cv2.warpAffine(normal,M,(width,height))
            cv2.imshow( self.img.split("/")[1].split(".")[0] ,imgout)
            windows =
gw.getWindowsWithTitle(self.img.split("/")[1].split(".")[0])
            windows[0].moveTo(0, 0)
            cv2.waitKey(10)

    def cuadrado(self,frame,color,width,height):
        cv2.line(frame, (width,0),(0,0),color, 10)
        cv2.line(frame, (width,0),(width,height),color, 10)
        cv2.line(frame, (width,height),(0,height),color, 10)
        cv2.line(frame, (0,height),(0,0),color, 10)
        cv2.imshow(self.img.split("/")[1].split(".")[0]+"_Camara",
cv2.flip(frame, 1))
        windows =
gw.getWindowsWithTitle(self.img.split("/")[1].split(".")[0]+"_Camara")
        windows[0].moveTo(self.screen_width - 650 , 0)

    def solution(self):
        #Herramienta para pintar los puntos
        mp_drawing1 = mp.solutions.drawing_utils;
        mp_pose1 = mp.solutions.pose;
        mp_drawing_styles1 = mp.solutions.drawing_styles
        with mp_pose1.Pose( static_image_mode = True) as pose1:
            datos_imagen_estatica=[]
            image1 = cv2.imread(self.img)
```

```
            image2 = image1.copy()
            self.zoom(image1)
            altura, ancho, _ = image2.shape
            image_rgb = cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
            results1 = pose1.process(image_rgb)
            print("Imagen Results",results1.pose_landmarks)
            if results1.pose_landmarks is not None:

mp_drawing1.draw_landmarks(image2,results1.pose_landmarks,mp_pose1.POSE_CONNEC
TIONS,landmark_drawing_spec=mp_drawing_styles1.get_default_pose_landmarks_styl
e())
                cv2.imshow(self.img.split("/")[1].split(".")[0], image1);
                pointsx = {}
                pointsy = {}
                for i in range(0,32):
                    pointsx[i] = int(results1.pose_landmarks.landmark[i].x *
ancho)
                    pointsy[i] = int(results1.pose_landmarks.landmark[i].y *
altura)
                landmarkx_sort = sorted(pointsx.items(),
key=operator.itemgetter(1), reverse=False)
                landmarky_sort = sorted(pointsy.items(),
key=operator.itemgetter(1), reverse=False)
                mp_drawing = mp.solutions.drawing_utils;
                mp_pose = mp.solutions.pose;
                mp_drawing_styles = mp.solutions.drawing_styles
                cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
                with mp_pose.Pose(min_detection_confidence=0.5,
min_tracking_confidence=0.5,static_image_mode = False) as pose:
                    suma = 0;
                    while cap.isOpened():
                        success, frame = cap.read()
                        frame.flags.writeable = False
                        if not success:
                            print("Ignorando el marco de la cámara vacía.")
                            continue
                        height1, width1, _ = frame.shape
```

```
                        self.cuadrado(frame,(0,0,255),width1,height1)
                        framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                        results = pose.process(framergb)
                        if results.pose_landmarks is not None:

mp_drawing.draw_landmarks(frame,results.pose_landmarks,mp_pose1.POSE_CONNECTIO
NS,landmark_drawing_spec=mp_drawing_styles.get_default_pose_landmarks_style())
                            pointsFramex = {}
                            pointsFramey = {}
                            for i in range(0,32):
                                pointsFramex[i] =
int(results.pose_landmarks.landmark[i].x * 640)
                                pointsFramey[i] =
int(results.pose_landmarks.landmark[i].y * 480)
                            landmarkFramey_sort = sorted(pointsFramey.items(),
key=operator.itemgetter(1), reverse=False)
                            landmarkFramex_sort = sorted(pointsFramex.items(),
key=operator.itemgetter(1), reverse=False)

                            for i in range(0,32):
                                if( landmarkFramex_sort[i][0] >=
(landmarkx_sort[i][0] - self.numPuntos)   and landmarkFramex_sort[i][0] <=
(landmarkx_sort[i][0] + self.numPuntos) ):
                                    #print("Entro al if1")
                                    if(  landmarkFramey_sort[i][0] >=
(landmarky_sort[i][0] - self.numPuntos) and landmarkFramey_sort[i][0] <= (
landmarky_sort[i][0] + self.numPuntos) ):
                                        #print("Entro al if2")

self.cuadrado(frame,(177,3,252),640,480)
                                        if( landmarkFramex_sort[i][1] >=
(landmarkx_sort[i][1] - self.parametro) and  landmarkFramex_sort[i][1] <=
(landmarkx_sort[i][1] + self.parametro)):
                                            #print("Entro al if3")
                                            self.cuadrado(frame,
(3,186,252),640,480)
```

```
                                            if( landmarkFramey_sort[i][1] >=
(landmarky_sort[i][1] - self.parametro) and landmarkFramey_sort[i][1] <=
(landmarky_sort[i][1] + self.parametro)):
                                                #print("Entro al if4")
                                                self.cuadrado(frame,
(0,255,0),640,480)
                                                suma = suma +1
                                                break

                            #for
                            if(suma >= 25):
                                self.cuadrado(frame, (0,255,0),640,480)
                                time.sleep(3)
                                break
                        if cv2.waitKey(5) & 0xFF == 27:
                            break
        cv2.destroyAllWindows();


for x in listImages:
    instancia = RunGame("Images/" + x + ".jpg" , 2, 15)
    instancia.solution()
    instancia = None
```