Inteligencia artificial (IA) en la traducción de algunas expresiones corporales de la lengua de Señas Colombiana en la atención médica a pacientes

Carlos Manuel Alarcón

Yhon Fredy Torres Pedraza

Universidad de Boyacá

Facultad de Ciencias e Ingeniería

Ingeniería de Sistemas

Tunja

2022

Inteligencia Artificial (IA) en la traducción de algunas expresiones corporales de la Lengua de Señas Colombiana en la atención médica a pacientes

Carlos Manuel Alarcón Alarcón
Yhon Fredy Torres Pedraza

Trabajo de grado para optar al título de:
Ingeniero de Sistemas

Director (a):
Carmen Constanza Uribe Sandoval
Doctora en Ingeniería con Especialidad en Ingeniería de Sistemas

Universidad de Boyacá
Facultad de Ciencias e Ingeniería
Ingeniería de Sistemas
Tunja
2022

Nota de aceptación:

Los estudiantes Carlos Manuel Alarcón Alarcón y Yhon Fredy Torres Pedraza, del programa de Ingeniería de Sistemas, realizaron su alternativa de grado en la modalidad de Trabajo de grado obteniendo una nota final de aprobación de (4.2) cuatro punto dos.

_____

Firma del Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

Tunja, 31 de octubre de 2022

"Únicamente el graduando es responsable de las ideas expuestas en el presente trabajo".

(Lineamientos constitucionales, legales e institucionales que rigen la propiedad intelectual).

Este proyecto está dedicado a nuestras familias, padres, madres y hermanos, también a nuestros compañeros de carrera profesores y personas que, a lo largo de este proceso, han estado brindándonos su apoyo en los momentos más difíciles.

Agradecimientos

A la Universidad de Boyacá por brindarnos la oportunidad de pertenecer a su familia y desarrollar nuestra vida estudiantil en sus recintos; al personal administrativo, por su colaboración, disposición y eficacia en los procesos académicos o administrativos en los que nos vimos involucrados; a la planta de docentes del programa de Ingeniería de Sistemas; a nuestra directora de trabajo de grado, la Doctora Carmen Constanza Uribe Sandoval, por apoyarnos desde el inicio de nuestra carrera y, aún más, por guiarnos en la construcción de este proyecto de grado; y a todos aquellos que participaron en nuestra formación, por ser amables y comprensivos y por brindar la mejor disposición para nuestra enseñanza.

Agradecemos también a Nicholas Renotte por compartir su investigación y sus desarrollos en Redes Neuronales Artificiales y en general en Inteligencia Artificial.

Por último, agradecemos a nuestras familias por brindarnos la oportunidad de ingresar en esta academia, formar nuestra vida profesional y darnos el apoyo moral que esto requirió.

Contenido

Pág.

Lista de Figuras

Pág.

Lista de Anexos

Glosario

Accesibilidad: este término encapsula una parte de la evolución humana que busca acercar a las tareas cotidianas, o incluso a aquellas que no lo son, a las personas que por motivos ajenos, no pueden desarrollar estas tareas, incluyendo a la comunidad de personas que cuenten con algún tipo de discapacidad; es la búsqueda y la ejecución de métodos en la que las herramientas sean fáciles de usar para todos, sin importar sus limitaciones ya sean pequeñas o amplias; accesibilidad por sí misma lleva a pensar en acceso, lo que implica que todos puedan tener acceso a lo que requieran y les sea necesario para un buen desarrollo personal.

Aprendizaje automático: mayormente conocido como Machine Learning, por su traducción en inglés. Esta es una rama de la ingeniería de sistemas, una ciencia podría llamarse ya que ha sido ampliamente estudiada. Se trata de métodos, algoritmos o procesos utilizados para que las computadoras aprendan, esto a partir de datos, puesto que se las computadoras por sí mismas no tienen esta facultad, sino más bien la de recibir órdenes; por ello, ya no se tiene que programar cada solución a las posibles peticiones o requerimientos del programa, esta tecnología permite que la máquina aprenda sobre el camino y pueda resolver problemas naturalmente y sin ayuda externa (Bobadilla, 2020).

Aprendizaje supervisado: se asocia a la manera de entrenar la máquina, cuando los datos de entrada del sistema están asociados una etiqueta o un valor de salida, en el libro Machine Learning y Deep Learning del autor Jesús Bobadilla se encuentra el siguiente ejemplo, donde la etiqueta es la intensidad oficial del terremoto:

> Es un conjunto de muestras de terremotos cuyos datos contienen la intensidad de la vibración previa tomada de sensores y cuyo objetivo es determinar la intensidad oficial del terremoto ([7.1,6.3, …],5.4), ([3.2,9.7, …], 7.1). Este es un problema de regresión y su utilidad podría ser la de generar información acerca de la intensidad predicha por el modelo de regresión cuando se la aporta a una nueva muestra (valores sistémicos en tiempo real) (Bobadilla, 2020, p. 14).

Otro ejemplo: Se nutre un modelo con un set de datos, conformado de imágenes de verduras, cada imagen con información como la de su color o tamaño, acompañada de una

etiqueta que indica a qué clase pertenece, así si el al sistema se le presenta por ejemplo un pepino, el modelo lo identifica al ver su color y tamaño y lo reconocería con certeza.

Aprendizaje no supervisado: en este tipo de aprendizaje se encuentra que los datos de entrada no están asociados a ninguna etiqueta o categoría, por ende, la máquina los procesa por sí sola, sin ayuda del científico de datos, encontrando similitudes o patrones entre los datos. Un ejemplo puede ser en la clasificación de verduras por medio de imágenes, donde los modelos agrupan las verduras por rasgos básicos como el color o el tamaño. (Bobadilla, 2020, p. 17)

Aprendizaje semi-supervisado: según Jesús Bobadilla en su libro Machine Learning y Deep Learning, el aprendizaje semi-supervisado hace referencia a:

> Se trata de conjuntos de datos en los que una porción de los datos está etiquetada y el resto no. Normalmente la cantidad de muestras etiquetadas es mucho más pequeña que las no etiquetadas. La mayoría de los algoritmos de aprendizaje semi supervisado son una mezcla de métodos supervisados y no supervisados (Bobadilla, 2020, p. 18).

Aprendizaje por refuerzo: este concepto está relacionado por Jesús Bobadilla en su libro Machine Learning y Deep Learning y dice lo siguiente:

> Está inspirada en mecanismos naturales. En este caso, el algoritmo de aprendizaje recibe información de un entorno real o simulado. Cuando el sistema realiza una acción es recompensado o penalizado, tal y como pasa con los seres vivos. Tales algoritmos de aprendizaje se denominan agentes. y aprenden estrategias, denominadas “políticas”, que maximizan las recompensas y minimizar las penalizaciones. La mayoría de los sistemas de inteligencia artificial actuales que están especializados en juegos, están basados en el enfoque de aprendizaje por refuerzo (Bobadilla, 2020, p. 18).

Comunicación: el congreso de la República en la ley 982 de 2005 decreta la siguiente definición:

> Es todo acto por el cual una persona da o recibe de otra, información acerca de las necesidades personales, deseos, percepciones, conocimiento o estados afectivos. Es la base y requisito obligatorio de toda agrupación humana ya que hace posible la constitución, organización y preservación de la colectividad. Es un proceso social, para que la comunicación se produzca es necesario que exista entre los interlocutores motivación para

> transmitir y recibir. Es preciso que haya intervenido explícita o implícita, un acuerdo entre los interlocutores respecto de la utilización de un código que permita la organización de los mensajes transmitidos tomando un medio o canal de comunicación determinado." (Congreso de la República de Colombia, 2005, p. 2).

Diccionario básico de la Lengua de Señas Colombiana: este es un documento que ha sido desarrollado por el Gobierno Nacional y el Instituto Nacional para Sordos, en él se encuentra qué es, cómo lo hicieron y el porqué del desarrollo de este, este documento significa la base para la estandarización del lenguaje y la construcción de un país más conocedor de ella. Este diccionario se inicia con 400 señas en agosto del 2000 y posteriormente se le adicionaron 800 señas más teniendo un total de 1200 (Ministerio de educación nacional, Instituto nacional para sordos (INSOR) y Instituto Caro y Cuervo, 2006).

Discapacidad: el Congreso de la República en la ley 762 de 2002 decreta:

> El término "discapacidad" significa una deficiencia física mental o sensorial, ya sea de naturaleza permanente o temporal, que limita la capacidad de ejercer una o más actividades esenciales de la vida diaria, que puede ser causada o agravada por el entorno económico y social. Congreso de la República de Colombia (2002, 31 de Julio, p. 1).

Inteligencia artificial: su sigla es IA en español o AI en inglés (Artificial Intelligence). Es una rama completa de la Ingeniería de Sistemas, que se encarga de resolver grandes medianos o pequeños problemas de conocimiento, asociados mayormente a la inteligencia humana, como el aprendizaje, memorización, la toma de decisiones, cálculos y la resolución de problemas cotidianos. Lo anterior lo logra por medio de Algoritmos, métodos matemáticos, patrones de datos, etc (Amazon Web Services, s.f.).

Lenguaje: en la ley 1346 de 2009 el Congreso de la República define lo siguiente "*Por "lenguaje" se entenderá tanto el lenguaje oral como la lengua de señas y otras formas de comunicación no verbal*" (Congreso de la República de Colombia, 2009).

Lengua de señas: el Congreso de la República en la ley 982 de 2005 decreta la siguiente definición:

> Es la lengua natural de una comunidad de sordos, la cual forma parte de su patrimonio cultural y es tan rica y compleja en gramática y vocabulario como cualquier lengua oral. La Lengua de Señas se caracteriza por ser visual, gestual y espacial. Como cualquier otra lengua tiene su propio vocabulario, expresiones idiomáticas, gramáticas, sintaxis diferentes del español. Los elementos de esta lengua (las señas individuales) son la configuración, la posición y la orientación de las manos en relación con el cuerpo y con el individuo, la lengua también utiliza el espacio, dirección y velocidad de movimientos, así como la expresión facial para ayudar a transmitir el significado del mensaje, esta es una lengua viso gestual. Como cualquier otra lengua, puede ser utilizada por oyentes como una lengua adicional. (Congreso de la República de Colombia, 2005).

Lengua de Señas Colombiana- LSC: según el Ministerio de Salud y Protección Social en la resolución número 1904 de 2017, este concepto "hace referencia a la lengua natural de carácter viso-gestual, cuyo canal de expresión son las manos, los ojos, el rostro, la boca y el cuerpo, y su canal de reproducción es visual. Esta lengua de señas es propia de la población con discapacidad auditiva colombiana." (Ministerio de Salud y Protección Social, 2017).

Librería de pyhton: una librería hace referencia a un grupo de funcionalidades que no son nativas del lenguaje de programación o que no se podrían conseguir de manera fácil, en caso específico de Pyhton "responde al conjunto de implementaciones que permiten codificar este lenguaje, con el objeto de crear interfaz independiente." Estas llamadas librerías son variables en la versión de Python, así como en el sistema operativo trabajado y su finalidad, teniendo como ejemplo algunas enfocadas en Deep Learning (Aprendizaje Profundo), Machine Learning (Aprendizaje Automático), cálculo numérico, visualización, procesamiento lenguaje natural etc (Immune Technology Institute, 2022).

Librería keras: esta es una librería de código abierto, como se puede observar en el repositorio principal GitHub. Está escrita en el lenguaje de programación Python y tiene como fin acelerar la creación de redes neuronales, por ello tiene un enfoque de experimentación; se ejecuta sobre Tensor Flow. Dentro de sus características se encuentra que es simple de manejar, pero no en sus resultados; que es flexible, porque los flujos de trabajos simples se ejecutan primero y los de mayor complejidad se ejecutan tomándose en cuenta la experiencia obtenida

en los otros; y, por último, que tiene gran potencia, y es por eso por lo que plataformas de alto calibre la utilizan con resultados verdaderamente impresionantes (GitHub, s.f.).

Librería numpy: esta es una librería de código abierto, diseñada para las tareas matemáticas dentro de Python, lenguaje en el que está escrita; creada en 2005, esta se desarrolla mayormente en GitHub, donde cuenta con una comunidad amplia. (Numpy, s.f.) Esta librería es la básica para la computación científica, empezando desde el manejo de matrices hasta abarcar campos como la matemática, lógica, álgebra lineal, estadística, simulación, aleatoriedad y demás relacionadas con este campo (NumPy, s.f.).

Librería matplotlib: Esta es una librería que está encargada de crear visualizaciones en base a datos estadísticos, estas visualizaciones son animadas, interactivas, y ayudan a comprender mejor la información. Dentro de estas graficas se encuentran diagramas de barras, de sectores, de cajas, áreas, contornos, mapas de color, histogramas, etc. Esta librería, además de mostrarnos toda esta multimedia, permite exportarla en diferentes formatos, lo que la convierte en una herramienta casi indispensable en diversos proyectos (Matplotlib, s.f.).

Librería opencv: esta librería está definida como una biblioteca de visión artificial de código abierto. OpenCV; se creó para proporcionar una infraestructura común para las aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. Esta biblioteca tiene alrededor de 2500 algoritmos de aprendizaje de máquina y visión por computadora, algunos clásicos y otros muy actuales. En la página principal de esta librería encontraremos sus usos descritos de la siguiente forma,

> Estos algoritmos se pueden usar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D de cámaras estéreo, unir imágenes para producir una alta resolución imagen de una escena completa, encontrar imágenes similares de una base de datos de imágenes, eliminar los ojos rojos de las imágenes tomadas con flash, seguir los movimientos de los ojos, reconocer el escenario y establecer marcadores para superponerlo con la realidad aumentada, etc (openCV, 2020).

Esta información ha sido recopilada en la página de la librería donde se encuentra su comunidad y activamente se debate y se nutre (OpenCV, 2020).

Librería pytorch: esta librería cuenta con una comunidad amplia, como se puede observar en GitHub donde se aloja y se describe, está enfocada en hacer cálculos matemáticos mediante la programación de tensores, como numpy, brindando aceleración de GPU, estos sistema es basado en cintas siendo uno de autoguardado, por lo general el uso de esta librería está sujeto a reemplazar a numpy y darle uso al potencial de la GPU en el ordenador en caso de contar con una, siendo rápida en sus funciones, también tiene bajo gasto de memoria. (GitHub, s.f.)

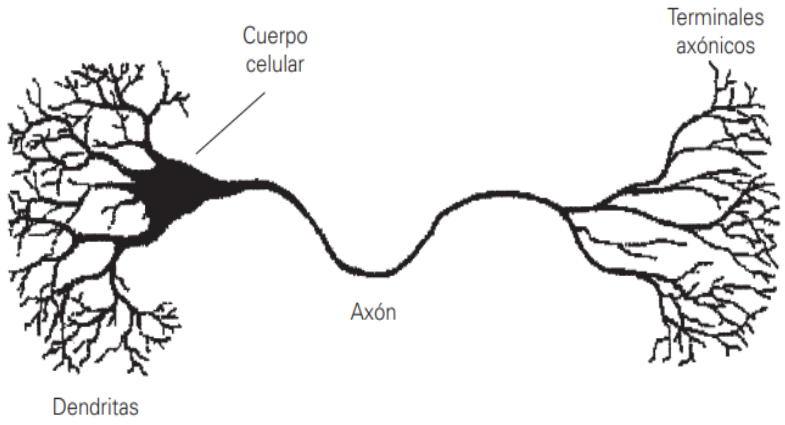
Librería tensorflow: en la página principal de esta librería donde se puede descargar obtener documentación y acceso a la comunidad que la compone es donde se puede encontrar la definición más certera y orientada de la misma;

> TensorFlow es una interfaz para expresar algoritmos de aprendizaje automático y una implementación para ejecutar dichos algoritmos. Un cómputo expresado con TensorFlow se puede ejecutar con poco o ningún cambio en una amplia variedad de sistemas heterogéneos, desde dispositivos móviles como teléfonos y tabletas hasta sistemas distribuidos a gran escala de cientos de máquinas y miles de dispositivos computacionales como tarjetas GPU. El sistema es flexible y se puede usar para expresar una amplia variedad de algoritmos, incluidos algoritmos de entrenamiento e inferencia para modelos de redes neuronales profundas, y se ha usado para realizar investigaciones y para implementar sistemas de aprendizaje automático en producción en más de una docena de áreas de ciencias de la computación y otros campos, incluidos el reconocimiento de voz, la visión por computadora, la robótica, la recuperación de información, el procesamiento del lenguaje natural, la extracción de información geográfica y el descubrimiento computacional de fármacos. Este documento describe la interfaz de TensorFlow y una implementación de esa interfaz que hemos creado en Google. La API de TensorFlow y una implementación de referencia se lanzaron como un paquete de código abierto bajo la licencia Apache 2.0 en noviembre de 2015 y están disponibles en www.tensorflow.org. (Abadi et al., 2015, p. 1)

Neurona: en la página oficial del Instituto Nacional del Cáncer se define una neurona como: “Tipo de célula que recibe y envía mensajes entre el cuerpo y el encéfalo. Los mensajes se envían por medio de una corriente eléctrica débil. También se llama célula nerviosa.” (Instituto Nacional del Cáncer, s.f.). Un esquema de la neurona se observa en la figura 1.

Figura 1

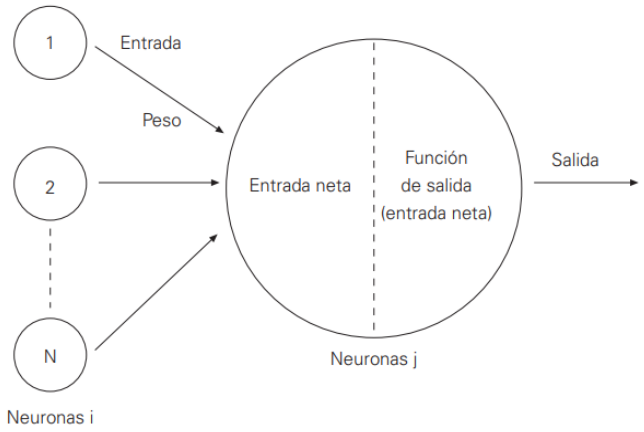
Estructura general de una neurona biológica



Fuente: Pérez, D., Hurtado. N. (2014*). Estudio difuso de isótopo de carbono 13 e isotopo de oxígeno 18 en el límite Triásico-Jurásico/Tesis*. (Trabajo de grado, Universidad Central de Venezuela). ResearchGate. https://www.researchgate.net/publication/316741449_Estudio_Neuro_difuso_de_isotopo_de_carbono_13_e_isotopo_de_oxigeno_18_en_el_limite_Triasico-Jurasico

Figura 2

Funcionamiento de una neurona artificial



Fuente: Pérez, D., Hurtado. N. (2014*). Estudio difuso de isótopo de carbono 13 e isotopo de oxígeno 18 en el límite Triásico-Jurásico/Tesis*. (Trabajo de grado, Universidad Central de Venezuela). ResearchGate. https://www.researchgate.net/publication/316741449_Estudio_Neuro_difuso_de_isotopo_de_carbono_13_e_isotopo_de_oxigeno_18_en_el_limite_Triasico-Jurasico

Python: es un lenguaje de programación fácil de leer para cualquier persona que tenga conocimientos básicos de programación. Además, su uso se ha extendido en los últimos años; algunas de sus características son:

> Es totalmente gratuito. Se trata de un lenguaje open source o de código abierto, por lo que no hay que pagar ninguna licencia para utilizarlo.
>
> Está respaldado por una enorme comunidad. Su carácter gratuito hace que continuamente se estén desarrollando nuevas librerías y aplicaciones. Es difícil pensar en algo que no haya hecho alguien. Esto es un factor multiplicativo para los programadores, puesto que cualquier duda estará resuelta en los foros.
>
> Es un lenguaje multiparadigma. Esto significa que combina propiedades de diferentes paradigmas de programación, lo que permite que sea muy flexible y fácil de aprender de manera independiente de los conocimientos del interesado.
>
> Sus aplicaciones no se limitan a un área en concreto. El hecho de que sea multiparadigma permite utilizarlo en campos aparentemente tan dispares como el diseño de aplicaciones web o la inteligencia artificial, entre muchos otros.
>
> Python es apto para todas las plataformas. Podemos ejecutarlo en diferentes sistemas operativos como Windows o Linux simplemente usando el intérprete correspondiente (Visus, 2020).

Red neuronal artificial - rna: en los contenidos brindados por Amazon Web Services (AWS) desde su página de recursos principal, se encuentran la siguiente definición, RNA es un método de la inteligencia artificial diseñado para enseñar a computadoras a procesar los datos asemejando la manera en la que lo hace un Red Neuronal Humana, esta semejanza radica en la utilización de estas neuronas interconectadas en una estructura de capas como la del cerebro humano, y así se puede realizar procesos de aprendizaje profundo teniendo un sistema adaptable el cual es utilizado por el ordenador para detectar errores y solucionarlos, podría decirse que para aprender de ellos; esto las convierte en una herramienta altamente funcional en problemas como generalizar y sacar conclusiones, analizar patrones en altas cantidades de datos, y en campos más relacionados al ser humano como: Visión artificial, reconocimiento de voz, procesamiento de lenguaje natural, etc.

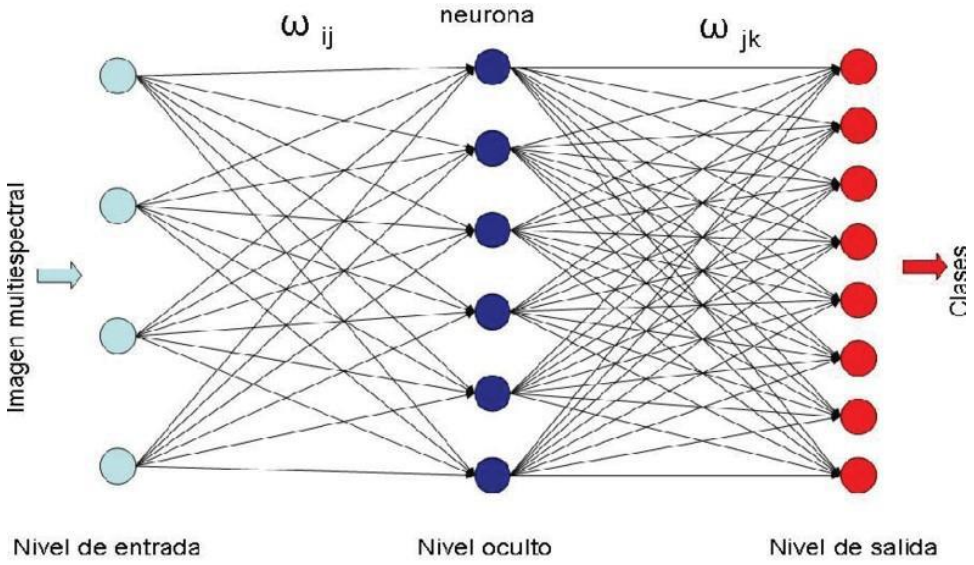
En el cerebro humano, millones de neuronas se encuentran interconectadas, lo que es altamente eficaz para que, a través de señales eléctricas envíe y reciba datos, procesando y

convirtiéndolos en información. En una Red neuronal Artificial estas neuronas son llamadas nodos y de la misma manera se encargan de trabajar en interconexión con otros miles de nodos, estas redes son programas o algoritmos los cuales en esencia realizan cálculos matemáticos en cada uno de estos miles y miles de nodos.

La arquitectura simple de una red neuronal está dada en primer lugar por una Capa de Entrada donde se recibe la información del mundo exterior se empieza con el procesamiento clasificando y pasando a las siguiente, que es la Capa Oculta, la cual está encargada de recibir la información ya sea de la capa de entrada o de otra capa oculta, procesando aún más la información y pasándola a la siguiente capa. Y, por último, se tiene la capa de salida donde se busca materializar los datos en resultados; puede ser en uno o varios nodos, esto dependerá del modelo que se maneje o del problema que se quiera resolver (Amazon Web Services, s. f). En la figura 3 se presenta un esquema de estas capas y donde se resalta la sencillez de esta arquitectura.

Figura 3

Estructura de una red neuronal de tres niveles.



Fuente: Carvajal F., Aguilar M., Agüera F., Aguilar F. (2022). *Clasificación de una imagen multiespectral del satélite de alta resolución espacial mediante redes neuronales.* https://docplayer.es/70443569-Clasificacion-de-una-imagen-multiespectral-de-satelite-de-alta-resolucion-espacial-mediante-redes-neuronales-artificiales.html

***Seña*:** Según la Real Academia Española "una seña es un indicio o gesto para dar a entender algo o venir en conocimiento de ello, aquello que de concierto está determinado entre dos o más personas para entenderse" (Real Academia Española [RAE], 2021).

***Red neuronal recurrente - rnn*:** conocida así por su traducción a ingles Recurrent Neural Network, IBM en su página principal muestra una definición para este término:

> Una red neuronal recurrente (RNN) es un tipo de red neuronal artificial que utiliza datos secuenciales o datos de series temporales. Estos algoritmos de aprendizaje profundo se usan comúnmente para problemas ordinales o temporales, como traducción de idiomas, procesamiento de lenguaje natural (nlp), reconocimiento de voz y subtítulos de imágenes; se incorporan a aplicaciones populares como Siri, búsqueda por voz y Google Translate. Al igual que las redes neuronales convolucionales y de avance (CNN), las redes neuronales recurrentes utilizan datos de entrenamiento para aprender. Se distinguen por su "memoria", ya que toman información de entradas anteriores para influir en la entrada y salida actual. Mientras que las redes neuronales profundas tradicionales asumen que las entradas y salidas son independientes entre sí, la salida de las redes neuronales recurrentes depende de los elementos previos dentro de la secuencia (IBM, 2021).

***Traductor*:** a lo largo de la historia se ha evidenciado la importancia de esta ciencia, empezando por los escritos antiguos que tenían que ser traducidos a lenguajes más actuales, y llegando a la actualidad donde existen, según Europa Press (2021), alrededor de 7.097 idiomas distintos en el mundo; es por esta magnitud que ha sido necesario la intervención de la tecnología en esta tarea tan antigua y extensa. Como ejemplo de estos programas el periódico El País habla del Traductor de Google que es utilizado por 500 millones de personas, también Bing de Microsoft que traduce alrededor de 70 idiomas (Rubio, 2020, 30 mayo). Es así como se evidencia que la función principal del traductor es tomar un texto en un idioma y traducirlo a otro idioma, de igual manera en texto (Hicheri, 2011).

Resumen

Inteligencia Artificial (IA) en la traducción de algunas expresiones corporales de la Lengua de Señas Colombiana en la atención médica a pacientes

La historia ha demostrado, como la ingeniería de sistemas trabaja a diario en la resolución de problemas, y como la rama de Inteligencia Artificial en las últimas décadas ha evolucionado permitiendo mejorar la vida de las personas; esta investigación pretende dar un aporte en la traducción de expresiones de la Lengua de Señas Colombiana. Inicialmente se verifica la necesidad de disminuir la brecha comunicacional de los médicos con personas con discapacidad auditiva, así como se sondean las expresiones más usadas en el ambiente de una consulta médica. Al mismo tiempo, se establecen las aplicaciones que otros investigadores han desarrollado en esta línea, apoyándose en medios como GitHub YouTube y Google Académico, entre otros. Posteriormente se selecciona un modelo de red neuronal recurrente, que se apoya en bibliotecas nativas de Python como Tensor Flow, Open Pose, Cv2, Matplotlib y Media Pipe con lo que se adecua un desarrollo propio que permite clasificar algunas expresiones de la Lengua de Señas Colombiana, red neuronal artificial que arrojó una precisión aceptable y un error bajo, cuya medida hizo uso de la biblioteca Tensor Board.

De esta forma, se logra alcanzar el objetivo general de entrenar un modelo de Redes Neuronales, que realice la traducción de la Lengua de Señas Colombiana a texto, enfocado en mejorar la comunicación de los estudiantes de medicina de la universidad de Boyacá y sus pacientes con discapacidad auditiva.

La metodología utilizada en este proyecto fue de tipo aplicada, ya que se hace énfasis en la solución de un problema. soportado en un enfoque mixto, ya que tiene resultados cualitativos, ya que se describe el funcionamiento de la aplicación y, así mismo, cuantitativos por que se estiman la precisión del reconocimiento de las señas y el error del modelo, representados en graficas donde se puede observar que alcanza el nivel más alto de precisión.

En el desarrollo de este proyecto se ha notado falta de información y su difusión con respecto a Lengua de Señas Colombiana (LSC), a pesar de los esfuerzos gubernamentales y de entidades como el Instituto Nacional para Sordos, así mismo, no se evidencian muchos aportes tecnológicos en la traducción de diferentes lenguas de señas, y específicamente en la traducción de la Lengua de Señas Colombiana. Con el desarrollo que se presenta en este trabajo, se ha evidenciado que la Red Neuronal Recurrente LSTM tiene una gran precisión, gracias a que permite procesar grandes cantidades de datos en la detección y seguimiento de acciones por medio de video. Finalmente, es importante que los investigadores continúen trabajando en el área de la Lengua de Señas Colombiana, explorando técnicas de Inteligencia Artificial para conseguir modelos que sean más flexibles y que generen menos carga de trabajo, contribuyendo a los procesos de inclusión en nuestro entorno.

Palabras claves: Inteligencia Artificial, Lengua de señas, Lengua de Señas Colombiana (LSC), Red Neuronal Convolucional (CNN).

Abstract

Artificial Intelligence (AI) in the translation of some body expressions of Colombian Sign Language in medical attention to patients.

History has shown how systems engineering works daily in the resolution of problems, and how the branch of Artificial Intelligence in the last decades has evolved allowing to improve people's lives; this research intends to give a contribution in the translation of expressions of the Colombian Sign Language. Initially, the need to reduce the communication gap between doctors and hearing-impaired people is verified, as well as the most used expressions in a medical office environment are surveyed. At the same time, the applications that other researchers have developed in this line are established, relying on media such as GitHub YouTube and Google Scholar, among others. Subsequently, a recurrent neural network model is selected, which is supported by native Python libraries such as Tensor Flow, Open Pose, Cv2, Matplotlib and Media Pipe, with which an own development is adapted to classify some expressions of the Colombian Sign Language, artificial neural network that yielded an acceptable accuracy and a low error, whose measurement made use of the Tensor Board library.

In this way, the general objective of training a Neural Network model, which performs the translation of Colombian Sign Language to text, focused on improving the communication of medical students of the University of Boyacá and their patients with hearing impairment, is achieved.

The methodology used in this project was of applied type, since it emphasizes the solution of a problem, supported in a mixed approach, since it has qualitative results, since it describes the operation of the application and, likewise, quantitative because it estimates the accuracy of sign recognition and the error of the model, represented in graphs where it can be observed that it reaches the highest level of accuracy.

In the development of this project, it has been noticed the lack of information and its diffusion with respect to Colombian Sign Language (CSL), in spite of the governmental efforts and entities such as the National Institute for the Deaf, likewise, there are not many technological contributions in the translation of different sign languages, and specifically in the translation of Colombian Sign Language. With the development presented in this work, it has been evidenced

that the Recurrent Neural Network LSTM has a great precision, thanks to the fact that it allows processing large amounts of data in the detection and tracking of actions by means of video. Finally, it is important that researchers continue working in the area of Colombian Sign Language, exploring Artificial Intelligence techniques to achieve models that are more flexible and generate less workload, contributing to the processes of inclusion in our environment.

Keywords: Artificial Intelligence, Sign language, Lengua de Señas Colombiana (LSC), Convolutional Neural Network (CNN).

Introducción

La Historia de la humanidad ha mostrado su evolución en diferentes campos del conocimiento y el uso de este para mejorar la calidad de vida de la especie. En este proyecto de grado se trabaja con dos ramas de esta evolución, buscando tomar herramientas de ambas y usándolas en aras de cumplir con los objetivos propuestos para el desarrollo de este proyecto.

La primera rama de investigación es la que busca soluciones y avances para la comunicación con la población discapacidad auditiva, enfocándose en un área básica de la buena calidad de vida como lo es la salud, trabajando en romper brechas comunicacionales entre el paciente y el medico en una consulta médica.

La segunda rama es en la tecnológica, en el área de la inteligencia artificial, enfocando el trabajo a investigar los avances en Machine learning, Visión artificial, y algunas otras tecnologías desarrolladas y programadas para la detección de poses humanas en video; se implementaran con distintos fines librerías como OpenPose, TensorFlow, OpenCV, CV2, las cuales ayudarán para que el computador sea capaz de traducir algunas expresiones de la Lengua de Señas Colombiana al lenguaje a castellano escrito, esto con el fin de ayudar a disminuir las brechas en la comunicación entre médicos que no cuenten con conocimientos suficientes para entablar una correcta conversación en Lengua de Señas Colombiana con los pacientes con discapacidad auditiva.

En la descripción de este documento encontraremos tres capítulos que evidencian el proceso por el que se ha pasado para culminar este prototipo: el Capítulo 1 presenta una vista técnica de los diferentes modelos investigados, así como las herramientas, librerías e interfaces de programación de aplicaciones (APIS) que ellas usan y cómo lo hacen. Basándose en este análisis y en los fines de este proyecto se recomendará o no el uso de los modelos o sus características y se presentará el modelo con el cual se desarrollará el proyecto. El Capítulo 2 mostrará la implementación del modelo definido anteriormente en el desarrollo del prototipo, la instalación de las librerías necesarias, su integración y todo lo que este proceso conlleva, como las pruebas, el manejo de errores y la documentación de estos. En el Capítulo 3 se presentan resultados apoyándose en, gráficas en imágenes que muestran los resultados obtenidos al aplicar pruebas al software. Por último, se dan unas conclusiones y recomendaciones para trabajos futuros que den continuidad a este proyecto.

Modelos de redes neuronales estudiados para este proyecto

Para este proyecto se estudiaron modelos de inteligencia artificial que aportan a solucionar el problema principal de esta investigación, como es la consolidación de una herramienta que permita traducir expresiones de la Lengua de Señas Colombiana.

En esta parte técnica se volcaron los esfuerzos a repositorios y plataformas digitales especializadas en almacenar y compartir proyectos, como: GitHub, StackOverflow, Kaggle y la Red Social de contenidos multimedia Youtube, donde se han encontrado amplias comunidades relacionadas con el desarrollo de inteligencia artificial y se puede observar y determinar que el lenguaje más propicio para la causa del proyecto es PYTHON por sus amplias librerías, flexibilidad, optimización en el área y capacidad de cálculo matemático.

Se ha orientado la búsqueda de modelos, hacia los de redes neuronales convolucionales, por ser estas las que más se asemejan al grupo de neuronas de la corteza visual primaria en el cerebro humano, ya que este modelo tienen como enfoque la identificación de objetos, la clasificación de escenas y la clasificación de imágenes en general; este modelo detecta patrones y se encarga de descomponer las imágenes en números, los cuales pueden ser analizados entrenados y, con la facultad de convolución de estas redes, se pueden convertir estos números de nuevo en imágenes, permitiendo visualizar los resultados, esta flexibilidad también permite tratar esta información con diferentes librerías (Cifuentes et al., 2018).

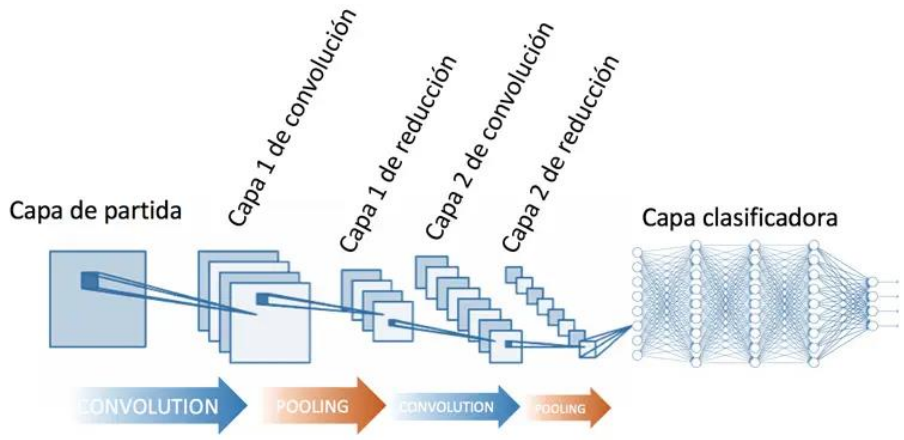
En la figura 4 se muestra de forma sencilla, cómo la Red Neuronal Convolucional, transforma la información de la capa de entrada, mediante operaciones de reducción y otras, que permiten entregar a la capa clasificadora la información pertinente de las imágenes para que realice los procesos de aprendizaje de la Red Neuronal Artificial inicialmente explicada.

En la actualidad la mayor parte de proyectos de investigación y desarrollos en producción, de IA con el enfoque de traducción de lengua de señas están basándose en el lenguaje de programación Python, siendo este el lienzo donde se comienzan a integrar diferentes librerías, las cuales su funcionamiento difiere en las técnicas de desarrollo de la red neuronal y en el entrenamiento de la misma, ya que hacen uso de diferentes librerías y técnicas para realizar el entrenamiento del modelo; unas de ellas son: tensorflow, keras y pytorch, que son los marcos más importantes utilizados por los científicos de datos en el aprendizaje profundo (Deep learning). En sí, cada uno de estos recursos son bibliotecas de software de código libre que

permiten abordar el aprendizaje automático, las redes neuronales profundas y el procesamiento de lenguaje natural (interacción entre máquina y lenguaje humano), los cuales conforman en conjunto herramientas de las cuales seleccionamos algunas de acuerdo con los requerimientos y necesidades.

Figura 4

Arquitectura de una red neuronal convolucional



Fuente: Calvo, D. (2017). *Definición de Red Neuronal Convolucional*. https://www.diegocalvo.es/red-neuronal-convolucional/

En este proceso investigativo se ha realizado una comparación entre estas herramientas, para determinar cuál usar. En el proyecto identificamos algunas diferencias primordiales entre estas bibliotecas, lo que ayudó a identificar las herramientas ideales para su implantación en el proyecto, de las tres bibliotecas en cuanto a nivel de api (interfaz de desarrollo de aplicaciones), keras es una API de alto nivel de muy fácil uso y de integración semántica muy simple y sencilla de entender, facilitando el desarrollo fluido y rápido, con capacidad de correr sobre tensorflow.

Por otro lado, tensorflow es un framework que se encarga de proveer ambos niveles de API (de bajo nivel y de alto nivel), a diferencia de pythoch que es de bajo nivel, enfocado al trabajo directo con conjuntos de expresiones de difícil entendimiento y uso; en cuanto al tamaño del dataset, tensorflow es mucho más robusto para los grandes, a diferencia de keras que por lo general usa datasets de tamaño pequeño, pytorch por otra parte permite una gran manipulación y configuración de modelos con grandes datasets que requieren una gran velocidad de

ejecución; en este aspecto keras es mucho más lento que tensorflow y pytorch y en este último es más fácil y frecuente realizar depuración de la redes neuronales, ya que cuenta con más herramientas de depuración, a diferencia de tensorflow donde realizar un proceso de depuración de las redes neuronales es muy complejo. De acuerdo con lo anterior se decide usar tensorflow ya que posee una gran adaptabilidad y buen performance en cuanto a la creación de una red neuronal y su entrenamiento, ya que el dataset es complejo de manipular y necesita una respuesta rápida en la ejecución de procesos como recolección de dataset, entrenamiento y predicción.

A continuación, se encontrarán algunos de los proyectos más relevantes que se revisaron en búsqueda de un modelo que permitiera realizar el reconocimiento de las expresiones de la manera óptima posible.

Modelo two-way sign language translator

Proyecto desarrollado en Python como aplicación de escritorio para traducir de voz a señas y viceversa haciendo uso de las siguientes librerías: Tkinter, Tensorflow, Keras, Pyaudio, Speech Recognition, PIL y OpenCV. Para realizar un test del proyecto y analizar su funcionamiento se debe realizar la instalación de la plataforma Anaconda, la cual facilita una distribución libre del lenguaje de programación Python y está enfocada en la ciencia de datos y el aprendizaje automático; se especializa en la gestión y en la implementación de paquetes de "data science". Una vez instalado Anaconda Navigator se crea un nuevo entorno de desarrollo donde se realiza el despliegue e instalación de todos los paquetes necesarios para el funcionamiento del proyecto.

En este caso el entorno de desarrollo que se usó para este proyecto cuenta con las siguientes versiones de librerías y lenguaje de desarrollo: tensorflow 2.4.1, opencv 4.6.0, keras 1.1.2 y Python 3.6.13. Muchos de estos proyectos utilizan ciertas versiones de librerías. Una vez instalados todos los requerimientos, se ejecuta el modelo desde jupyter-lab, IDE a través del cual se revisó el funcionamiento del código fuente.

El objetivo de este proyecto es realizar la traducción de una seña de alguna de las letras del abecedario, a texto y voz y viceversa; el proyecto está dividido en 5 partes: importación de librerías, lectura de los datos, implementación en keras, entrenamiento y traducción de señas a

texto y voz. El set de datos consiste en un conjunto de imágenes con su significado en la lengua de señas, con el que se entrena en keras y realiza el reconocimiento de la seña correspondiente a las letras del abecedario, para finalmente realizar una predicción de una seña a texto y voz y viceversa (Aniketdhole07, 2021).

Modelo detección y clasificación de manos

En este proyecto se realiza un detector de manos. Detecta si la mano es la derecha o la izquierda; está diseñado en Python con librerías como tensorflow y OpenCv, y realiza una recolección de datos de testeo y de entrenamiento a través de Cv2; la recolección se realiza por medio de una cámara.

Como primer paso se realiza la recolección del set de datos, el cual se divide en dos categorías el conjunto de datos de entrenamiento y el conjunto de datos de validación, este conjunto de datos se recolecta a través de una cámara web y se realiza usando la librería Cv2 para realizar captura de 300 imágenes de cada una de las manos derecha e izquierda, este conjunto de datos es almacenado en un objeto, se extraen 21 puntos de interés de cada mano, al igual que el ancho y el alto de los fotogramas, cuyas coordenadas se ubican en 3 matrices (RGB).

Una vez se obtiene el set de datos se importan las librerías correspondientes de tensorflow para realizar la creación de la red neuronal, con sus capas de convolución o filtros; para realizar el entrenamiento de la red neuronal, aplicando tres filtros de convolución; también se realiza un tratamiento de imágenes girando, haciendo acercamiento e invirtiendo las imágenes para obtener más elementos y lograr obtener un entrenamiento óptimo.

Los filtros harán que las imágenes queden de un tamaño más pequeño, sus matrices son aplanadas, es decir, quedan de una sola dimensión (vector). Finalmente, en la última capa de la red se utiliza la función softmax la cual analiza qué probabilidad hay de que esa imagen sea de una clase o de otra con una buena precisión. Por último, se guarda el modelo en un archivo para posteriormente usarlo para realizar la predicción de manos derechas e izquierdas (Aprende Ingeniería, s.f.).

Modelo ASL alphabet translation

En este proyecto se hace uso de una red convolucional neuronal en base a un modelo pre entrenado de tensorflow que permite ahorrar en tiempo al realizar el entrenamiento de nuestra red convolucional el entrenamiento de este modelo se realiza a través de tensorflow, el dataset que se usó para el proyecto son imágenes en formato .jpg el cual está conformado por 6 señas hola, no, por favor, gracias, sí, te amo (Munisht06, s.f.).

Modelo sign-language

Este modelo está estructurado en Python haciendo uso de librerías como OpenCV 3.4, h5py y pyttsx3. A diferencia de otros, hace uso de la potencia de procesamiento de la tarjeta gráfica del equipo donde se esté ejecutando, en este caso para obtener menor tiempo a hora de realizar la ejecución realiza como primer paso la recolección de un histograma de la posición de la mano como medio para la recolección de señas en formato jpg y de tipo monocromático; una vez se ha realizado la creación del dataset se procesan las imágenes dándole un giro o dando un efecto espejo en las imágenes para aumentar los elementos del dataset; una vez terminado este proceso se realiza el entrenamiento a través de la librería keras y el modelo generado es almacenado para posteriormente cargarlo con keras y ejecutar la detección de señas haciendo uso de OpenCv (EvilPort2, s.f.).

Modelo real time face expression recognition

Este proyecto está enfocado en el reconocimiento de expresiones faciales, haciendo uso de Keras, Flask and OpenCV para la identificación de las expresiones del rostro dentro de siete categorías, las cuales están definidas por estado de ánimo como enojado, feliz, triste, miedo, sorprendido, neutral, disgusto. El dataset de este proyecto está conformado por 36.000 imágenes aproximadamente, el 80% de estas imágenes conforman el conjunto de datos de entrenamiento dejando así un 20% de imágenes para validación (Rishabhjainps, s.f.).

Modelo action detection for sign language

Este proyecto se encuentra en un repositorio público de github. Su desarrollo está estructurado en Python, y hace uso de librerías o bibliotecas de código abierto y de redes neuronales. En el siguiente capítulo se hará una descripción de la implementación de este modelo, las herramientas necesarias para su uso y todo lo que envuelve el proceso de implementación, ya que fue el seleccionado para resolver el problema de esta investigación (Renotte, s.f.).

Implementación del modelo más adecuado

En este capítulo se presentan las actividades que fueron necesarias ejecutar para conseguir el funcionamiento de una aplicación, que permite entrenar 5 expresiones de la Lengua de Señas Colombiana con una precisión aceptable.

Instalación de librerías necesarias

Se realiza la instalación de Anaconda para crear un entorno de desarrollo con las librerías y paquetes necesarios para que funcione el modelo, se descarga el instalador desde la página oficial y se siguen las instrucciones para su correcta instalación. En la figura 5 se ilustra el proceso de descarga.

Figura 5

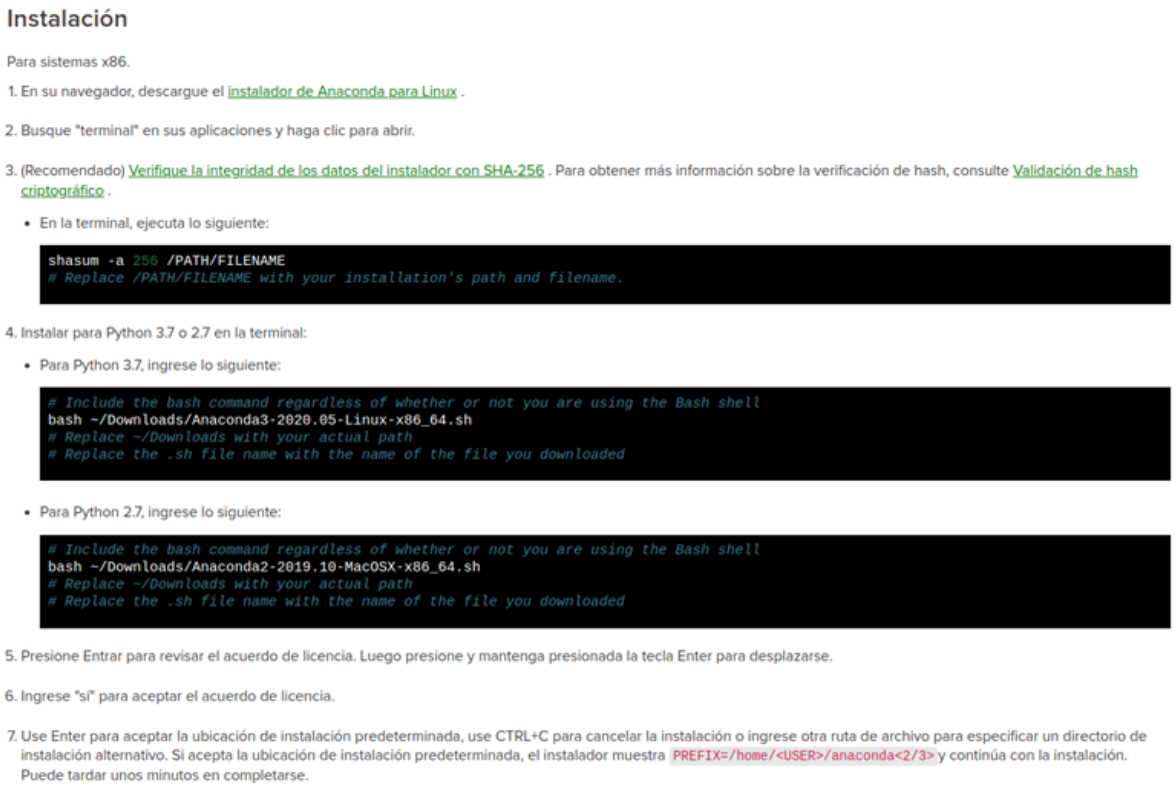
Descarga de recursos de Anaconda

Fuente: Anaconda. (s.f.). *The world's most popular data science platform.* https://www.Anaconda.com/

Las instrucciones de instalación se pueden encontrar en la siguiente página, sitio oficial de documentación Anaconda https://docs.Anaconda.com/Anaconda/install/linux/. El proceso paso a paso se describe en la figura 6.

Figura 6

Instrucciones de instalación dentro de entorno Linux



Instalación

Para sistemas x86.

1. En su navegador, descargue el instalador de Anaconda para Linux .
2. Busque "terminal" en sus aplicaciones y haga clic para abrir.
3. (Recomendado) Verifique la integridad de los datos del instalador con SHA-256 . Para obtener más información sobre la verificación de hash, consulte Validación de hash criptográfico .
   - En la terminal, ejecuta lo siguiente:

```
shasum -a 256 /PATH/FILENAME
# Replace /PATH/FILENAME with your installation's path and filename.
```

4. Instalar para Python 3.7 o 2.7 en la terminal:
   - Para Python 3.7, ingrese lo siguiente:

```
# Include the bash command regardless of whether or not you are using the Bash shell
bash ~/Downloads/Anaconda3-2020.05-Linux-x86_64.sh
# Replace ~/Downloads with your actual path
# Replace the .sh file name with the name of the file you downloaded
```

   - Para Python 2.7, ingrese lo siguiente:

```
# Include the bash command regardless of whether or not you are using the Bash shell
bash ~/Downloads/Anaconda2-2019.10-MacOSX-x86_64.sh
# Replace ~/Downloads with your actual path
# Replace the .sh file name with the name of the file you downloaded
```

5. Presione Entrar para revisar el acuerdo de licencia. Luego presione y mantenga presionada la tecla Enter para desplazarse.
6. Ingrese "sí" para aceptar el acuerdo de licencia.
7. Use Enter para aceptar la ubicación de instalación predeterminada, use CTRL+C para cancelar la instalación o ingrese otra ruta de archivo para especificar un directorio de instalación alternativo. Si acepta la ubicación de instalación predeterminada, el instalador muestra `PREFIX=/home/<USER>/anaconda<2/3>` y continúa con la instalación. Puede tardar unos minutos en completarse.

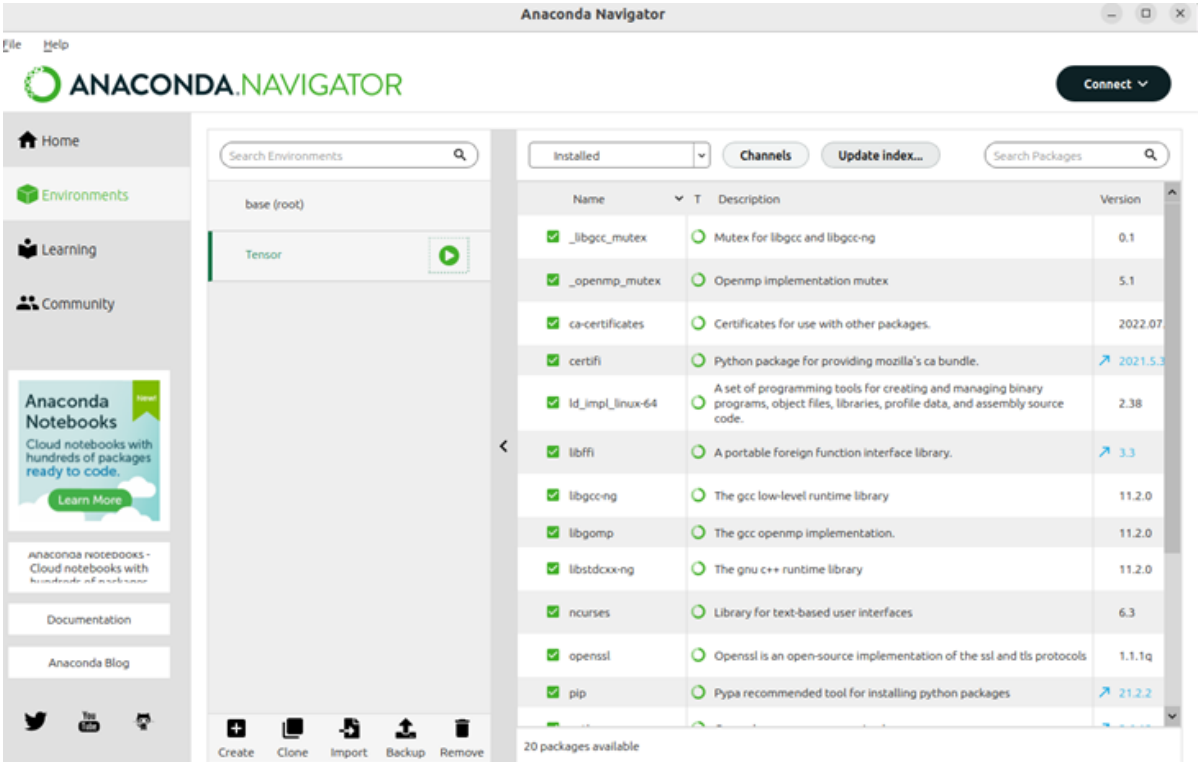
Fuente: Anaconda documentation (s.f.). *Anaconda documentation*. https://docs.Anaconda.com/

Una vez instalado el entorno de desarrollo se procede a instalar las librerías necesarias para la ejecución del proyecto; haciendo uso de Anaconda navigator se crea un nuevo entorno de desarrollo el cual se llamó Tensor y dentro de este entorno de desarrollo.

Se instalan las librerías a través de la consola de Anaconda navigator de acuerdo con las versiones requeridas como se observa en la figura 7 donde se muestra el entorno de desarrollo y las versiones de las librerías para el modelo de red neuronal LSTM RNN.

Figura 7

Menú principal Anaconda navigator



Fuente: Autores.

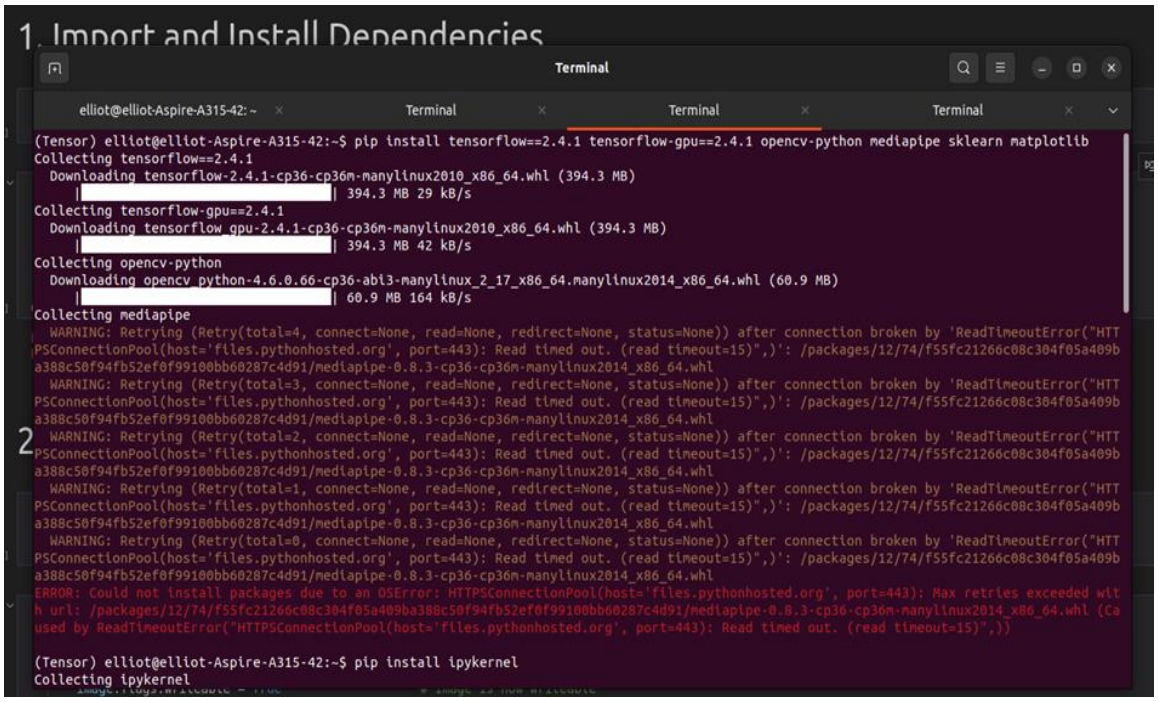
Desde Anaconda navigator y, una vez seleccionado el nuevo entorno de desarrollo, se instala las siguientes librerías con su correspondiente versión, tensorflow 2.4.1, tensorflow-gpu 2.4.1, opencv-Python, Mediapipey sklearn matplotlib; para realizar este proceso se tiene que abrir el entorno de desarrollo que se ha creado en Anaconda

Tal entorno de desarrollo se ha llamado Tensor y se abre en una terminal desde donde se instalan las dependencias.

Es importante instalar las versiones que se recomiendan de cada una de las librerías ya que puede generar conflictos y errores al momento de ejecutar este modelo, en la figura 8 se observa cómo se lleva a cabo el proceso de instalación de librerías.

Figura 8

Instalación de paquetes y librerías necesarias.



Fuente: Autores.

En este punto ya se tiene todo un entorno de desarrollo con las dependencias, paquetes y librerías que son requeridas para realizar pruebas y ejecutar dicho modelo. Para realizar el análisis del código del modelo y adecuarlo a las necesidades del proyecto se utiliza el IDE Visual Studio Code, en el cual se puede ejecutar el modelo usando el entorno de desarrollo que previamente se ha preparado.

Descripción del modelo

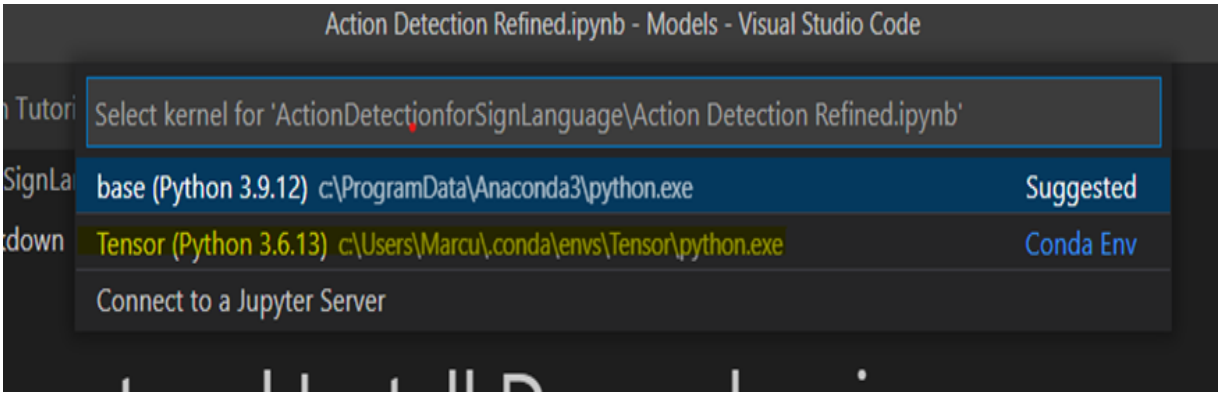
Para construir el modelo es necesario tener en cuenta etapas como: la recolección y tratamiento del set de datos, la creación de la red neuronal LSTM, el entrenamiento y almacenamiento del modelo y las pruebas. En esta subsección se muestra paso a paso la construcción del modelo.

Importación de paquetes

Como primer paso se debe abrir el proyecto desde el IDE Visual Studio Code, se selecciona el entorno de desarrollo llamado Tensor, creado en Anaconda, el cual se puede observar en la figura 9.

Figura 9

Seleccionando kernel, para el entorno de desarrollo implementación

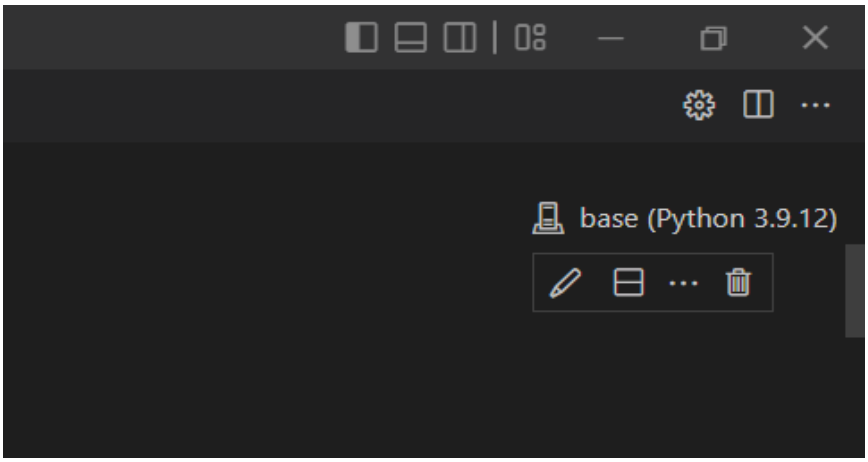


Fuente: Autores.

En la figura 10 observamos el entorno de desarrollo que ha generado Anaconda por defecto.

Figura 10

Seleccionando kernel, para entorno de desarrollo, nativo



Fuente: Autores.

La invocación de los paquetes se realiza a través de una instrucción llamada **import** que buscará los módulos que se deben importar y creará un objeto del módulo para ser utilizado dentro del código; si no se encuentra el módulo a la hora de invocarlo con la función import, arrojará un error denominado **Module Not Found Error** advirtiendo de la no existencia del módulo; para solucionar este problema de importación se debe identificar el módulo o librería que no se ha encontrado y proceder a instalarla, desde el entorno de desarrollo creado en Anaconda, desde donde se instalarán las librerías a través del gestor de paquetes de Python llamado **pip**; las librerías utilizadas se pueden observar en la figura 11.

Figura 11

Importación de librerías necesarias dentro del entorno de desarrollo

Fuente: Autores

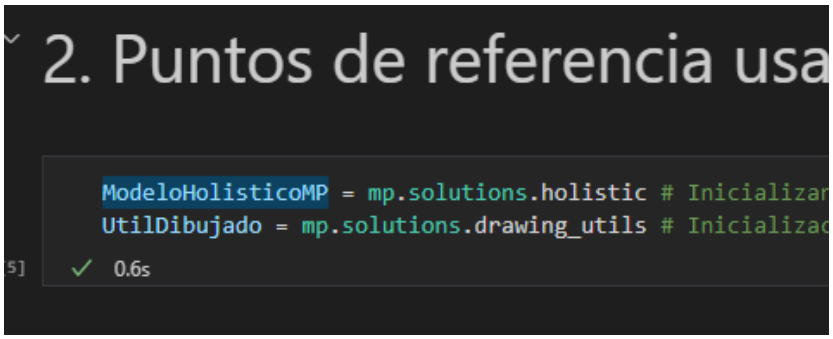
A continuación, se establecen dos variables que permitan usar el modelo holístico Mediapipe, y usar la utilidad de dibujo las cuales se usarán eventualmente: la primera inicializa el modelo para ser usado en el proyecto, Modelo Holístico MP, la segunda variable Utildibujado tiene como objetivo usar las utilidades de dibujo y representación en pantalla de los puntos de referencia que genera el modelo holístico Mediapipe, además de las expresiones que se detectan y su probabilidad en tiempo real, tal como se aprecia en la figura 12.

Inicialización del modelo mediapipe

Se define una función que recibirá dos parámetros: uno generado por el modelo y el otro será la imagen capturada por cámara, esta función permite realizar la detección además de dibujar los puntos de referencia del modelo holístico Mediapipe compuesto por tres zonas rostro, manos y la posición, esta función procesara cada uno cada uno de los frames que componen cada uno de los videos que se han registrado en el dataset para las expresiones correspondientes.

Figura 12

Establecimiento de variables para modelo



Fuente: Autores

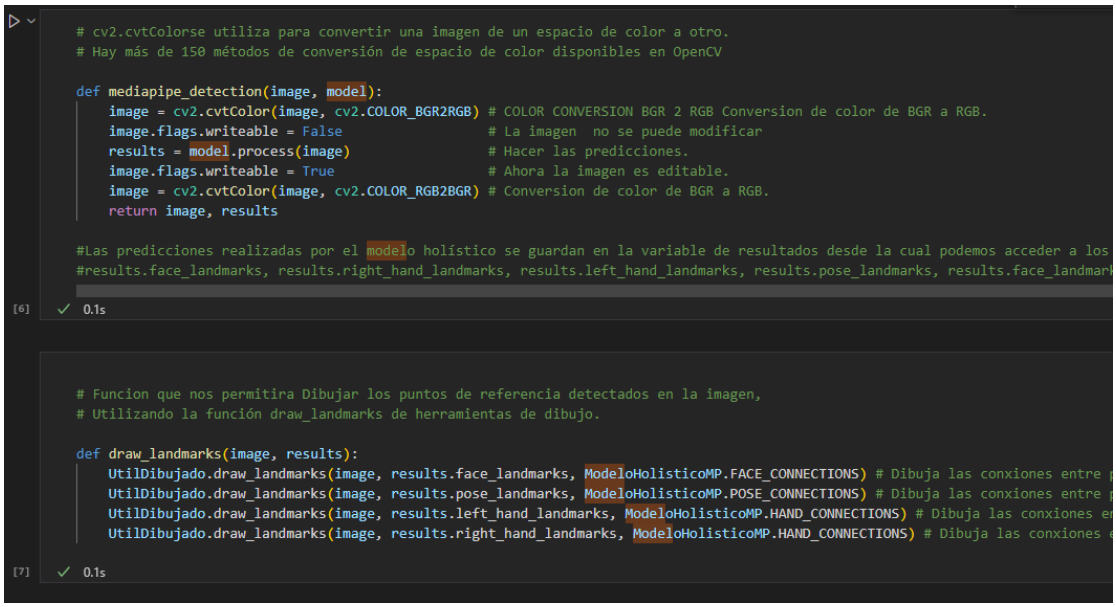
También se debe hacer un tratamiento de las imágenes que se capturan en cuanto al formato de estas, ya que Mediapipe trabaja con un formato RGB; para ello se realiza la conversión de la imagen de entrada de cada uno de los videos recolectados haciendo uso de la librería de OpenCv desde el formato origen BGR a el formato RGB para que puedan se puedan ser utilizados y procesados por el modelo, para lo cual se usa *image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR),* la función se denomina mediapipe_detection.

El proceso de tratamiento de imágenes para el set datos se puede observar en la función responsable del proceso en la figura 13.

Se define y se ajustan los parámetros correspondientes a los estilos de los puntos de referencia y de las conexiones del modelo holístico, la forma, tamaño, color y demás características de estilos para identificar y diferenciar cada una de las zonas de interés y los puntos de referencia correspondientes, haciendo uso del modelo holístico Mediapipe.

Figura 13

Funciones básicas para detección



```
# cv2.cvtColorse utiliza para convertir una imagen de un espacio de color a otro.
# Hay más de 150 métodos de conversión de espacio de color disponibles en OpenCV

def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB Conversion de color de BGR a RGB.
    image.flags.writeable = False                  # La imagen  no se puede modificar
    results = model.process(image)                 # Hacer las predicciones.
    image.flags.writeable = True                   # Ahora la imagen es editable.
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # Conversion de color de BGR a RGB.
    return image, results

#Las predicciones realizadas por el modelo holístico se guardan en la variable de resultados desde la cual podemos acceder a los
#results.face_landmarks, results.right_hand_landmarks, results.left_hand_landmarks, results.pose_landmarks, results.face_landmark
```

[6] ✓ 0.1s

```
# Funcion que nos permitira Dibujar los puntos de referencia detectados en la imagen,
# Utilizando la función draw_landmarks de herramientas de dibujo.

def draw_landmarks(image, results):
    UtilDibujado.draw_landmarks(image, results.face_landmarks, ModeloHolisticoMP.FACE_CONNECTIONS) # Dibuja las conxiones entre
    UtilDibujado.draw_landmarks(image, results.pose_landmarks, ModeloHolisticoMP.POSE_CONNECTIONS) # Dibuja las conxiones entre
    UtilDibujado.draw_landmarks(image, results.left_hand_landmarks, ModeloHolisticoMP.HAND_CONNECTIONS) # Dibuja las conxiones e
    UtilDibujado.draw_landmarks(image, results.right_hand_landmarks, ModeloHolisticoMP.HAND_CONNECTIONS) # Dibuja las conxiones
```

[7] ✓ 0.1s

Fuente: Autores

Se define la función draw_styled_landmarks que estará encargada de realizar la representación de los estilos de dibujado para cada uno de los puntos de referencia del modelo holístico Mediapipe; la cantidad total de puntos de referencia establecidos en el modelo holístico Mediapipe es 1662, los cuales están preestablecidos y distribuidos de la siguiente forma: la zona del rostro está representada por 468 puntos de los cuales cubren la totalidad del rostro identificando los ojos, nariz y boca, las manos están representadas por 21 puntos de referencia para la mano izquierda y derecha y sus conexiones Github (s.f.). Mediapipe Pose.

En la figura 14 se muestran los puntos de referencia para la mano izquierda. Los estilos utilizados para la representación de los puntos de referencia se aprecian en la figura 15.

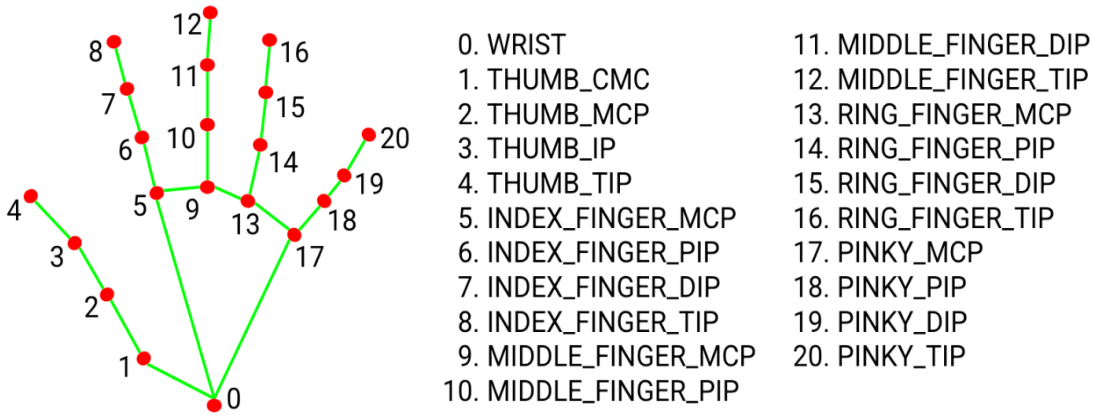
Ajuste de parámetros, estilos en puntos de referencia y conexiones

En este apartado se inicia identificando la cámara o dispositivo desde el que se va a realizar la captura de los videos, para este fin se declara Cv2.VideoCapture(0), dependiendo del dispositivo que se desee usar ya sea la cámara web que integra la computadora o una cámara

externa; el número 0 identificará la cámara integrada en el equipo y así sucesivamente para los demás dispositivos que estén conectados en el equipo.

Figura 14

Modelo puntos de referencia Mediapipe mano.



Fuente: Github (s.f.). *MediaPipe Hands*. https://google.github.io/mediapipe/solutions/hands.html

Figura 15

Puntos de referencia distribuidos

```
#Define color, tamaño, tipo, forma de representacion de conexiones y puntos de referencia.
def draw_styled_landmarks(image, results):
    # Estilos de dibujado  para conexiones y puntos de referencia  para zona rostro.
    UtilDibujado.draw_landmarks(image, results.face_landmarks, ModeloHolisticoMP.FACE_CONNECTIONS,
                                UtilDibujado.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                                UtilDibujado.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                                )
    # Estilos de dibujado  para conexiones y puntos de referencia  para zona postura.
    UtilDibujado.draw_landmarks(image, results.pose_landmarks, ModeloHolisticoMP.POSE_CONNECTIONS,
                                UtilDibujado.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                UtilDibujado.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                )
    # Estilos de dibujado  para conexiones y puntos de referencia  para zona mano izquierda.
    UtilDibujado.draw_landmarks(image, results.left_hand_landmarks, ModeloHolisticoMP.HAND_CONNECTIONS,
                                UtilDibujado.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                                UtilDibujado.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                                )
    # Estilos de dibujado  para conexiones y puntos de referencia  para zona man derecha.
    UtilDibujado.draw_landmarks(image, results.right_hand_landmarks, ModeloHolisticoMP.HAND_CONNECTIONS,
                                UtilDibujado.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                                UtilDibujado.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                )
 ✓ 0.5s
```

Fuente: Autores

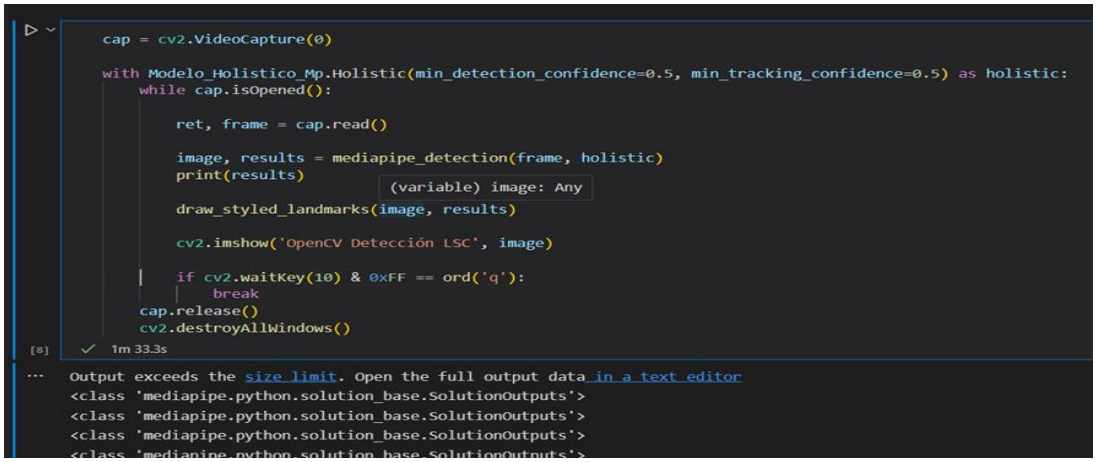
Se crea una instancia del modelo Mediapipe para empezar a usarlo, ajustando los valores para el nivel mínimo de detección de manos (min detection confidence=0.5) y los valores para el nivel mínimo de rastreo seguimiento de los puntos de referencia o landmarks (min_tracking_confidence=0.5) de las manos, que serán 21 puntos para cada mano derecha e izquierda.

Mapeo de puntos de referencia

Se inicia una captura para mapear los puntos de referencia, rostro, manos y posición con respecto a la detección y seguimiento de la persona que está realizando la seña, se realiza la predicción y se realiza el dibujado de los puntos de referencia y conexiones, posteriormente se imprime el resultado con OpenCv, proceso representado en la figura 16.

Figura 16

Configuraciones necesarias para entrenamiento



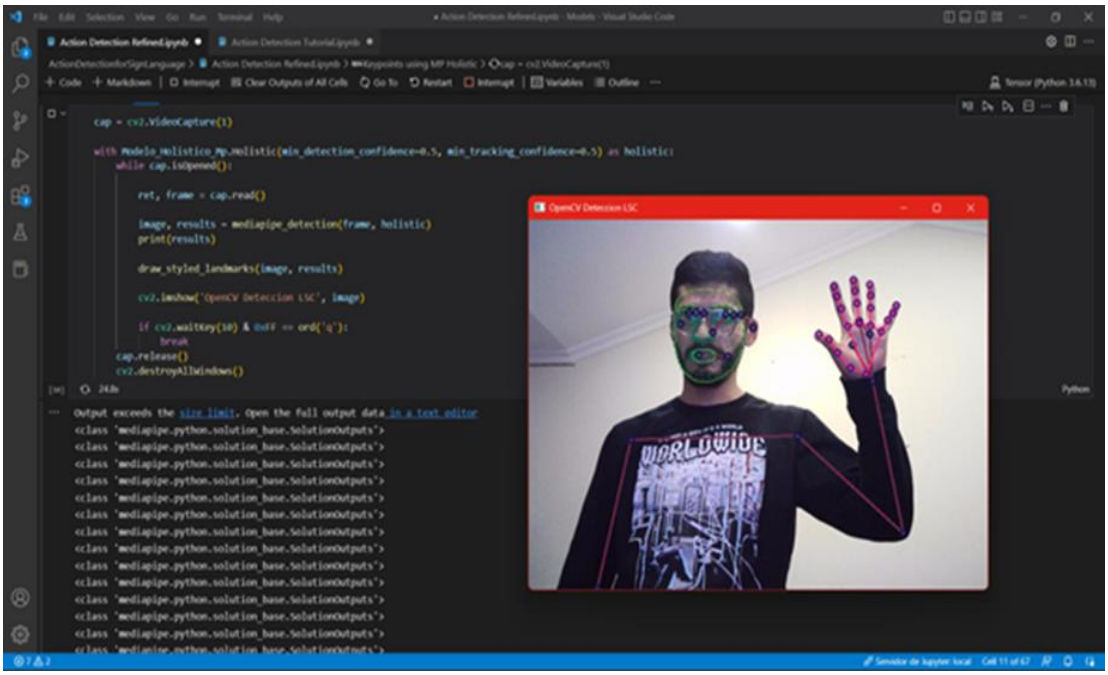
Fuente: Autores

Se comprueba los puntos de referencia que se capturaron, imprimiendo la imagen a través de OpenCv, junto con los puntos de referencia para el rostro, manos y postura y se verifica el tamaño de los puntos de referencia para la zona de la mano izquierda como.

Una muestra de la representación de los puntos de referencia del modelo holístico Mediapipe se observa en la figura 17.

Figura 17

Puntos de referencia mano izquierda



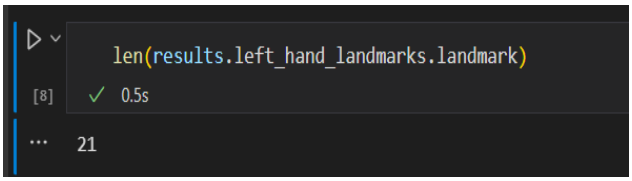
Fuente: Autores

Se verifica la extensión de los puntos de referencia que se ha mapeado, luego se imprimen el tamaño de los puntos registrados por el modelo, (Se comprueba que corresponde a los puntos de la mano izquierda).

Usando len() se verifica el tamaño del arreglo que contiene los puntos de referencia para la mano izquierda, como se observa en la figura 18.

Figura 18

Comprobando puntos mano izquierda



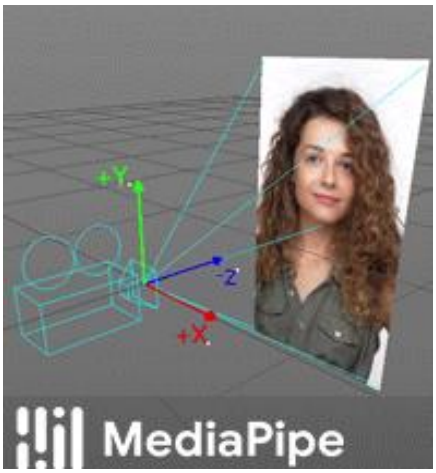
Fuente: Autores

Extracción de puntos de referencia

Se realiza la extracción de los valores correspondientes a cada uno de los puntos clave o de referencia para la postura, rostro, mano izquierda y derecha; los valores de estos puntos corresponden a coordenadas espaciales ancho, alto y profundidad (x,y,z) y un valor que corresponde a la visibilidad del punto de referencia de posición como se representa en la figura 19 y se aprecia en el modelo en la figura 20.

Figura 19

Coordenadas espaciales



Fuente: Github (s.f.). *MediaPipe Face Mesh*. https://google.github.io/mediapipe/solutions/face_mesh.html

Figura 20

Extracción valores de puntos de referencia

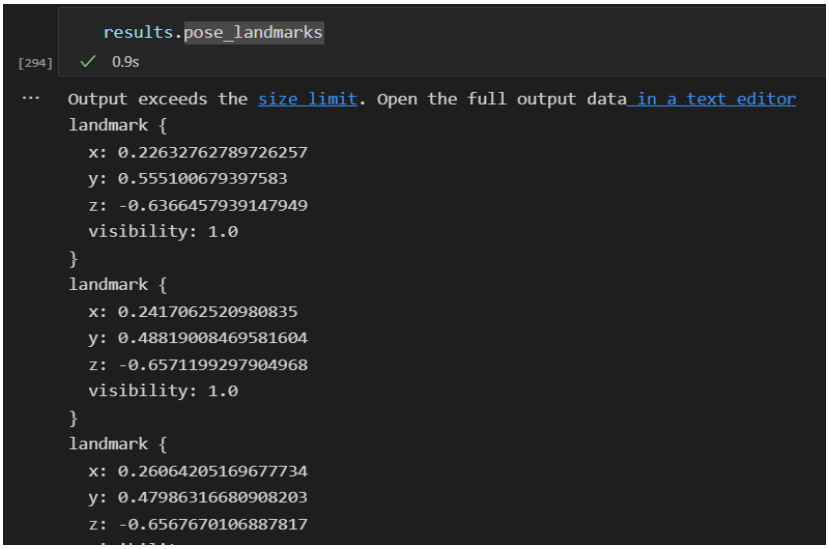
```
pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)
[14] ✓ 0.8s

len(test)
[302] ✓ 0.8s
... 4

pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(132)
face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(1404)
lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
[15] ✓ 0.8s
```

Fuente: Autores

Se almacenan los valores de los puntos de referencia en arreglos usando numpy, para posteriormente usar estos datos en la red neuronal, esto se ejecuta para cada una de las zonas de interés: posición, rostro, mano izquierda y mano derecha. Los valores son almacenados en matrices que se inician con valor cero, los cuales posteriormente serán reemplazados por los valores de cada uno de los puntos de referencia. Dichos valores se verifican en la figura 21.

Figura 21

Almacenamiento valores de puntos



Fuente: Autores

Para la zona del rostro se debe obtener 468 puntos de referencia con coordenadas en x, y, z. Si se multiplican los 468 puntos de referencia por las coordenadas 3D x, y, z, se obtendrán un total de 1404; de la misma forma para la zona de postura se debe obtener 33 puntos de referencia los cuales serán multiplicados por las coordenadas 3D x, y, z y un valor de visibilidad que indica si un punto de referencia es visible o está oculto en la imagen, este valor de visibilidad tiene un valor entre 0.0 y 1.0 (Mediapipe Pose, 2020), obteniendo un total de 132; se tiene que realizar lo anterior para la mano derecha e izquierda, para cada una de ellas se tendrá en cuenta 21 puntos de referencia y cada uno de estos puntos con coordenadas x.y, z, con lo que se obtiene un total de 63, como se puede observar en la figura 22.

Figura 22

Modelos empleados en mediapipe holistic

Fuente: Solano, G.S. (2021, 11 agosto). *Detecta 543 puntos del cuerpo con MEDIAPIPE HOLISTIC – MediaPipe – OpenCV*. https://omes-va.com/mediapipe-holistic-mediapipe-Python/

Los valores que se han obtenido en la figura 21 serán los tamaños de las matrices donde se almacenan los valores de cada punto de referencia y sus coordenadas correspondientes; a través de numpy creamos las matrices con los tamaños establecidos para cada zona: posición, rostro, mano izquierda y mano derecha matrices, con valores en cero que serán reemplazados posteriormente.

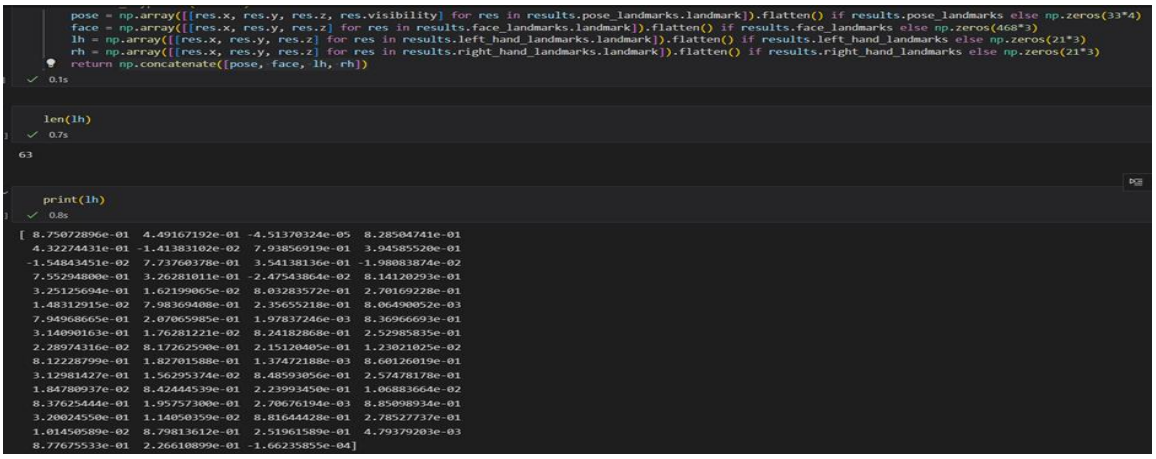
Creación de directorios y almacenamiento de valores de puntos de referencia en matrices

Una vez preparado lo anterior se devuelve un array en el cual están concatenadas las matrices para posición, rostro, mano izquierda y mano derecha, información que se almacena en un archivo para alimentar el modelo de red neuronal, como se observa en la figura 23.

Se preparan los directorios donde se realizará la exportación de los datos, se selecciona el directorio raíz del proyecto donde se crea un nuevo directorio denominado MP_Data donde se almacenan los datos contenidos en las matrices que se han creado anteriormente.

Figura 23

Almacenamiento de valores de puntos de referencia.

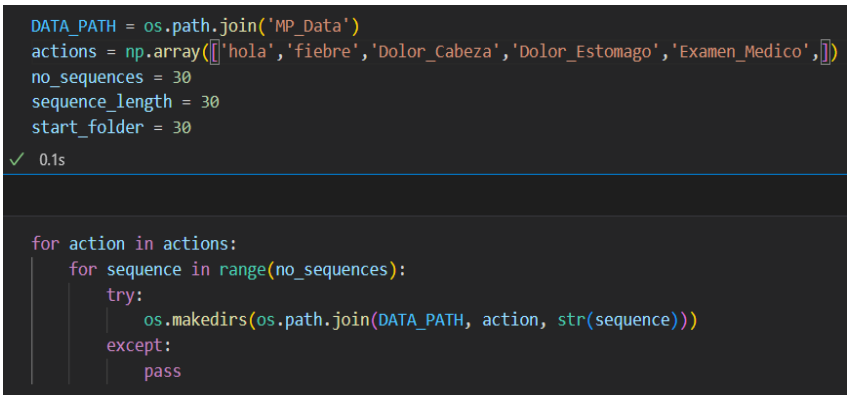


Fuente: Autores

Las expresiones que se clasificaran son, Hola, Fiebre, Dolor de cabeza, Dolor de estómago y Examen médico. Se nombran dos variables donde se establecen el número de secuencias de video que se van a almacenar no_sequences y el número de fotogramas para cada porción de video, donde se recolectan 30 secuencias de video por expresión, cada secuencia de video con un tamaño de 30 frames, y a su vez cada frame o imagen contendrá 1662 puntos de referencia en total, de las zonas de rostro, postura, manos. Se seleccionaron 30 videos con un tamaño de 30 frames ya que es el valor óptimo en cuanto a tiempo y potencia de procesamiento, como se evidencia en el código que se muestra en la figura 24.

Figura 24

Establecimiento de expresiones y directorios de almacenamiento.



Fuente: Autores

Seguidamente se realizará la recolección, extracción de los puntos de referencia, y almacenamiento de los valores de cada uno de estos puntos de referencia en los directorios correspondientes a cada expresión a clasificar. A través de OpenCv se inicia la captura de vídeo seleccionando el dispositivo a usar, configurando el ancho y alto de la captura de vídeo, se inicia la recolección, extracción y almacenamiento de los valores de los puntos de referencia proceso que se observa en la figura 25.

Figura 25

Configuración previa a la recolección.

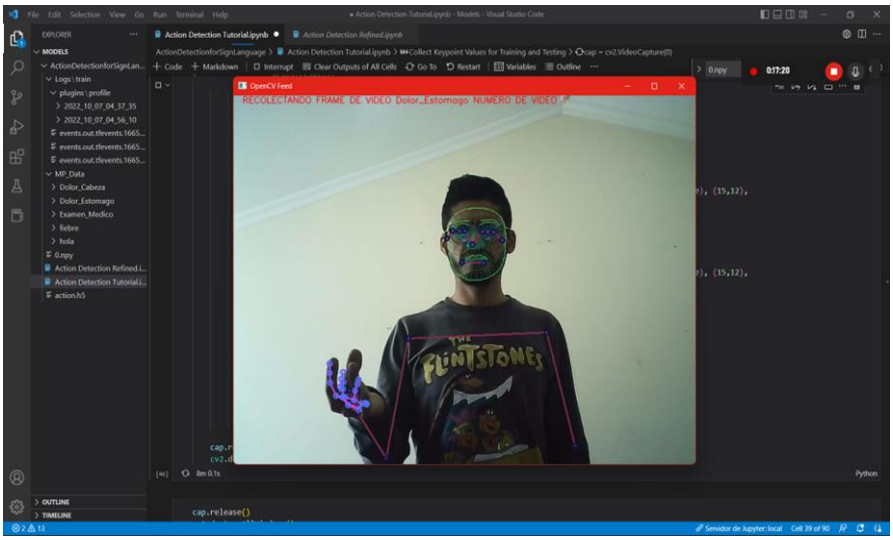
```
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1200)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1024)
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    for action in actions:
        for sequence in range(no_sequences):
            for frame_num in range(sequence_length):
                ret, frame = cap.read()
                image, results = mediapipe_detection(frame, holistic)
                draw_styled_landmarks(image, results)

                if frame_num == 0:

                    cv2.putText(image, 'INICIANDO RECOLECCION', (500, 500),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0, 255), 5, cv2.LINE_AA)
                    cv2.putText(image, 'RECOLECTANDO FRAME DE VIDEO {} NUMERO DE VIDEO {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1, cv2.LINE_AA)

                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:

                    cv2.putText(image, 'RECOLECTANDO FRAME DE VIDEO {} NUMERO DE VIDEO {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 1, cv2.LINE_AA)

                    cv2.imshow('OpenCV Feed', image)

                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

    cap.release()
    cv2.destroyAllWindows()
```

Fuente: Autores

El proceso de recolección de valores de puntos de referencia para las 5 expresiones se aprecia en las figuras 26, 27 y 28 donde se ven 3 capturas para la recolección de puntos de referencia de la expresión *dolor de estómago*.

Figura 26

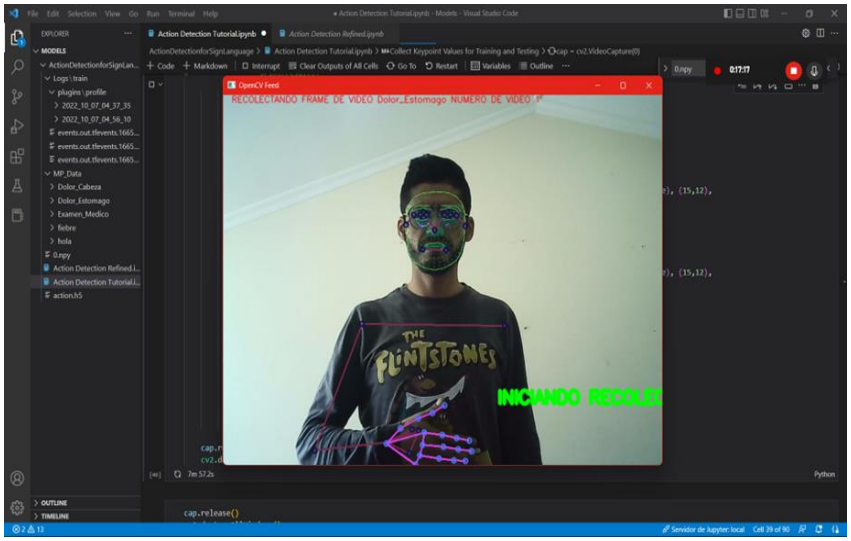
Entrenamiento del modelo - imagen 1



Fuente: Autores

Figura 27

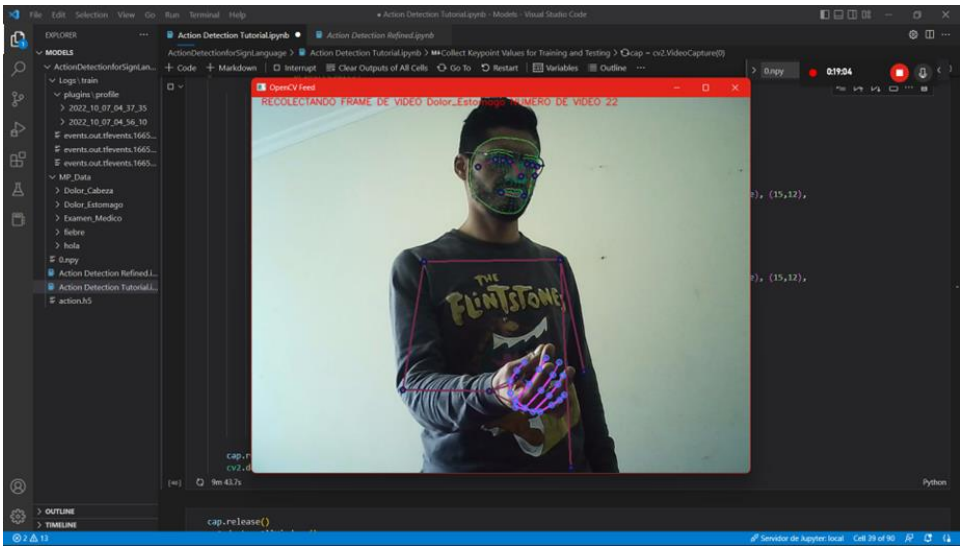
Entrenamiento del modelo - imagen 2



Fuente: Autores

Figura 28

Entrenamiento del modelo - imagen 3



Fuente: Autores

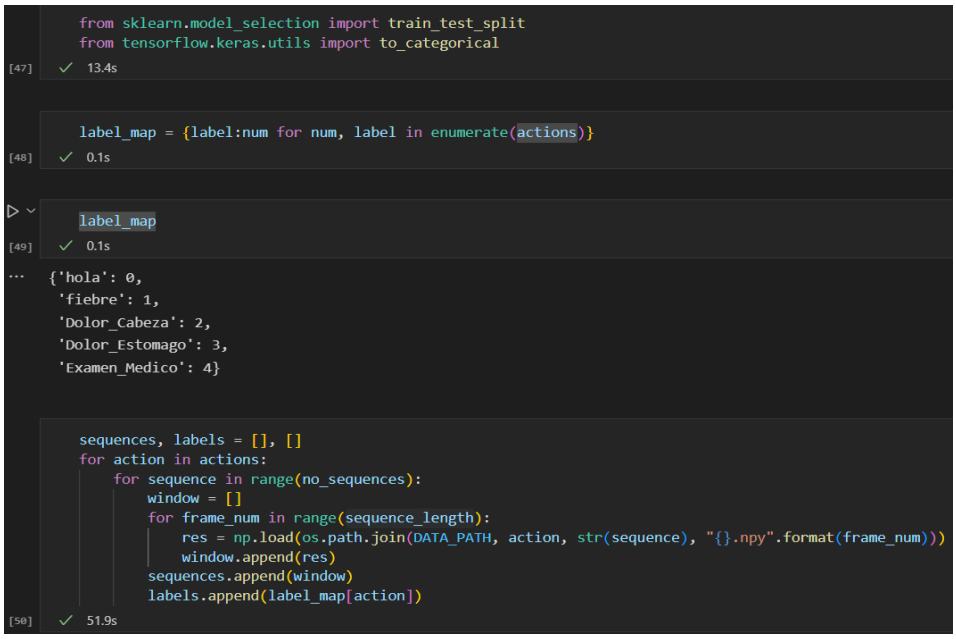
Etiquetado de datos (data labeling)

Una vez recolectados, extraídos y almacenados los puntos de referencia, se realiza el preprocesado de esta información y se realiza el proceso de etiquetado de datos y sus propiedades; este etiquetado es un paso crucial, ya que se refiere al proceso de identificar datos sin procesar, en este caso los frames de cada uno de los videos que hace parte del dataset, lo que implica agregar una etiqueta (inferencia de lo que es) que permita identificar y proporcionar un contexto al modelo de red neuronal. Un ejemplo de etiquetado es la inferencia que indica si una foto contiene un edificio o una motocicleta o en el caso de un audio o grabación la inferencia que indica qué posibles palabras se escucha en él. Amazon Web Services. (s.f.).

En este caso se realiza un etiquetado de datos en contexto a las expresiones que se establecieron para ser clasificadas, infiriendo o etiquetando el tipo de expresión (seña) que está representada en la figura 29. Este etiquetado proporciona un conjunto de datos de entrenamiento para el modelo de red neuronal y hace que esta sea aún más efectiva.

Figura 29

Etiquetado de datos para entrenamiento



```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
[47] ✓ 13.4s

label_map = {label:num for num, label in enumerate(actions)}
[48] ✓ 0.1s

label_map
[49] ✓ 0.1s

{'hola': 0,
 'fiebre': 1,
 'Dolor_Cabeza': 2,
 'Dolor_Estomago': 3,
 'Examen_Medico': 4}

sequences, labels = [], []
for action in actions:
    for sequence in range(no_sequences):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
[50] ✓ 51.9s
```

Fuente: Autores

Establecimiento de la red neuronal de tipo LSTM y ajuste de parámetros

Posteriormente al etiquetado de los datos, y con el conjunto de datos que se obtuvo, se realizó la construcción y el entrenamiento de la red neuronal tipo LSTM. El modelo de red neuronal recurrente LSTM RNN (long short-term memory recurrent neural network) es un tipo especial de red neuronal recurrente capaz de recordar un dato relevante y preservarlo a lo largo de un intervalo de tiempo, lo que significa que tiene una memoria de largo plazo así como de corto plazo, a diferencia de una red neuronal convencional la cual tiene una memoria a corto plazo, es por esta característica que el modelo LSTM RNN es una de las redes neuronales más óptimas y robustas utilizadas en diferentes aplicaciones de aprendizaje automático, reconocimiento de voz y detección de lenguaje de signos.

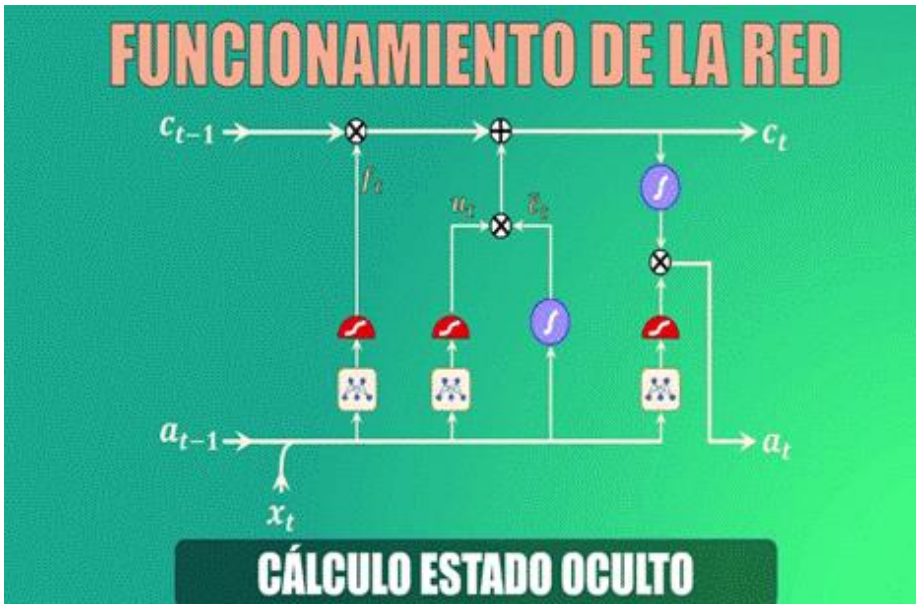
Esta red neuronal funciona de forma similar a como el cerebro humano realiza el análisis de secuencias; por ejemplo, al comprar un artículo en internet y dirigirse a la sección de valoración del artículo, para tomar la decisión de compra, se filtra la información más relevante de la reseña como: calidad, duración y resistencia, y el resto de información es descartada.

Esta es la estructura de funcionamiento de una red neuronal LSTM RNN a la cual, gracias a que posee una memoria a largo y otra a corto plazo, se le puede agregar, actualizar y eliminar información, filtrando los datos que sean relevantes de los que no y así facilitar la predicción. Si se considera la memoria a largo y corto plazo como una banda transportadora que almacenará los datos y a través del uso de compuertas, es posible agregar o eliminar información a una celda de estado dentro de la banda transportadora usando una serie de compuertas (olvidar, actualizar y salida) que permiten clasificar entre la información más importante es irrelevante.

Esta clasificación o filtración de información se realiza a través de una función sigmoidal, la cual toma valores entre 0 y 1; al tomar el valor cero descarta la información que se almacena en memoria, y si toma el valor 1 permite que la información avance hacia la memoria. Una vez se tenga filtrada la información se puede acceder a diferentes réplicas de la celda en diferentes instantes de tiempo, generando una predicción de acuerdo con los datos que se encuentran en memoria. Su funcionamiento y estructura están representadas de una forma más didáctica en la figura 30.

Figura 30

Red neuronal recurrente (LSTM)



Fuente: Sotaquira. (2019). *¿Qué son las Redes LSTM?*. https://www.codificandobits.com/img/posts/2019-07-20/actualizacion-estado-oculto.gif

Se construye la red neuronal LSTM importando las librerías necesarias como se observa en la figura 31; se realiza la construcción de las capas de la red neuronal LSTM RNN, se establecen tres capas LSTM de la siguiente forma dos de 64 unidades y una de 128 unidades más dos capas ocultas y una capa de salida final.

Figura 31

Construcción de red neuronal

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
```

✓ 0.1s

```
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
```

✓ 0.1s

```
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
```

✓ 0.8s

Fuente: Autores

Predicción, almacenamiento del modelo y evaluación

Seguidamente se compila el modelo y se realiza el entrenamiento del modelo ajustando la cantidad de etapas de entrenamiento (epoch). En cuanto a al número y tipo de capas que se utilizaran para la red neuronal son tres capas LSTM de las cuales están distribuidas en una capa de entrada con 64 unidades que recibirá como parámetro la cantidad de fotogramas multiplicado por el valor total de puntos de referencia, otras dos capas de activación LSTM una de 128 unidades y otra de 64 unidades la primera recibirá el valor de secuencias que retorna la primera capa, la segunda recibe igualmente el valor de secuencias de la segunda capa sin embargo no retornara valores para las siguientes capas, las cuales son ocultas, de 64 y 32 nodos respectivamente y una de capa final de salida de expresiones con un parámetro Softmax que devolverá valores que están dentro de una probabilidad entre 0 y 1.

En el modelo se evidencio que el número de etapas de entrenamiento es un parámetro relevante al momento de realizar las predicción y detección de las expresiones, se realizaron múltiples pruebas de entrenamiento con diferentes etapas de entrenamiento con el fin de mejorar el tiempo de entrenamiento la precisión de este y obtener el valor mínimo de pérdida, todos los datos de entrenamiento que se recolectaron previamente pasan a través de la red neuronal para que esta aprenda sobre ellos, para este caso existen 5000 ciclos y 150 arreglos para el conjunto de 5 expresiones, cada ciclo los 150 datos, pasaran por la red neuronal y esta realizara un proceso de aprendizaje.

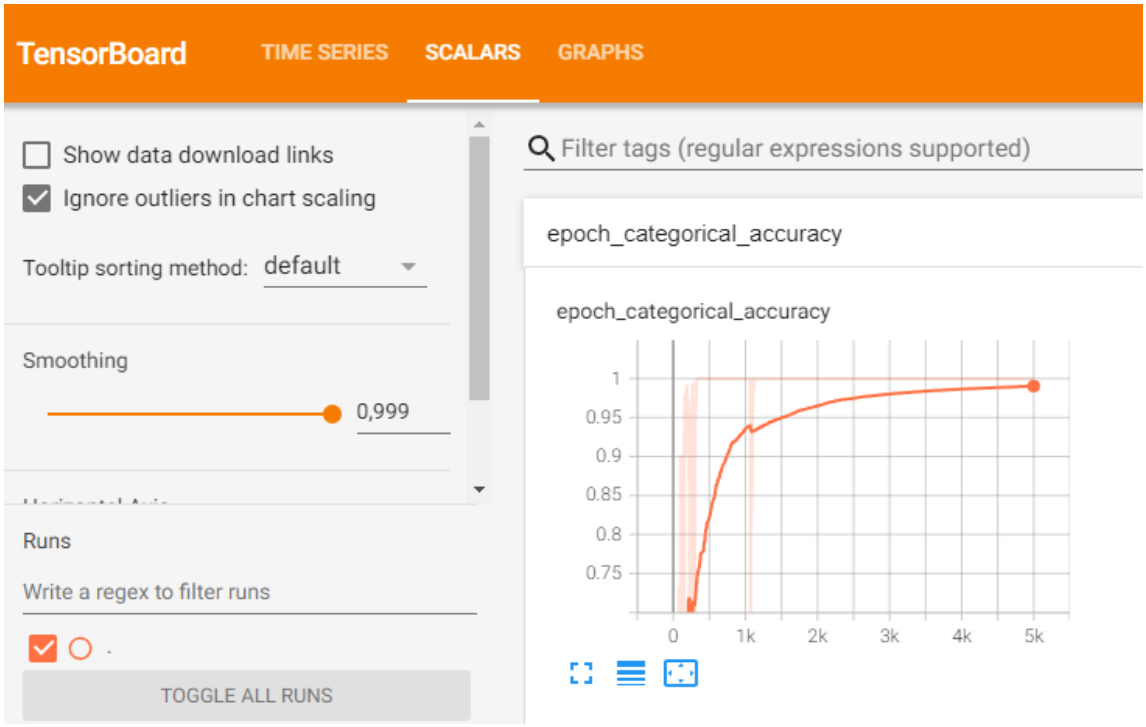
Ese tipo de red neuronal se usó por tres razones muy importantes, la primera es la cantidad de datos que necesita para realizar predicciones con un valor de precisión alto dentro del intervalo preestablecido, lo que permite ser más agiles en cuanto a tiempo y recursos de procesamiento, la segunda razón es la facilidad y rapidez de entrenamiento gracias a que es una red neuronal mucho más densa contando con más de medio millón de parámetros, y la tercera razón por la que se uso es la rapidez en la predicción de las expresiones en tiempo real.

Se evidencia que la cantidad de etapas de entrenamiento pueden afectar al modelo en cuanto a precisión y predicción de las expresiones, se pueden presentar dos fenómenos denominados Overfitting y Underfitting , el modelo es susceptible a cualquiera de los dos ya que el overfitting se presenta cuando se realiza un sobreajuste del modelo, es decir un sobre entrenamiento del modelo donde el algoritmo considerara como valores validos solo los datos

que sean idénticos a los que le proporcionamos en el conjunto de datos o set de datos, tomando así valores no validos o erróneos o comúnmente denominados ruido, que ocasionaran que el modelo sea incapaz de diferenciar las entradas de datos buenas como fiables, por el contrario el underfitting se presenta cuando no proporcionamos suficientes datos de entrada, o proporcionamos una gran cantidad de entradas de datos erróneas por lo que el algoritmo no tendrá la suficiente cantidad de datos fiables para realizar una construcción de un modelo preciso.

Figura 32

Gráfica exponencial de precisión en el entrenamiento del modelo.



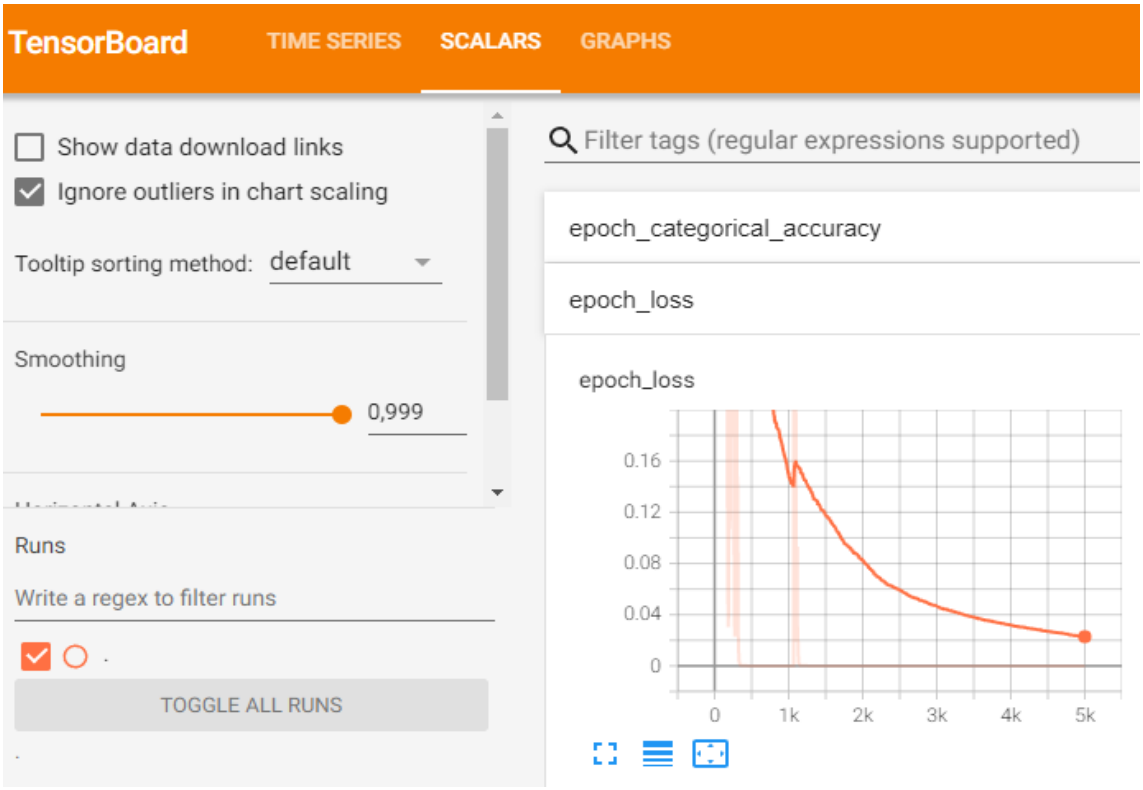
Fuente: Autores

Para evitar el underfitting es necesario tener en cuenta la complejidad del modelo, eliminar el ruido de los datos y aumentar el número de ciclos de entrenamiento para obtener resultados más óptimos, por el contrario para contrarrestar el overfitting se puede aumentar la cantidad de datos de entrenamiento, analizar el entrenamiento en tiempo real para identificar si existe un aumento sustancial en la pérdida y si es el caso detener el entrenamiento, en el caso del modelo que se usó y de acuerdo a las gráficas de precisión y pérdida que se obtuvieron, al

seguimiento del entrenamiento, de los valores de precisión y pérdida en tiempo real concluimos, que existe un equilibrio del modelo que permitirá realizar una predicción y detección de cinco expresiones de la Lengua de Señas Colombiana, con valores muy buenos de precisión dentro el intervalo preestablecido, cómo podemos observar en la figuras 32 y 33 respectivamente.

Figura 33

Gráfica exponencial de pérdida en el entrenamiento.



Fuente: Autores

Se utilizó la herramienta de análisis de datos que integra Tensorflow, llamada Tensorboard, la cual permite analizar en tiempo real el comportamiento del entrenamiento en cuanto a las variables de precisión y pérdida que previamente se establecieron en el modelo para hacer el seguimiento.

Realizando un análisis del sumario del modelo se observa que las capas de la red neuronal que se definieron previamente además de la cantidad total de parámetros usados, el total de parámetros y la cantidad de parámetros no entrenados, da cuenta de que en realidad este modelo hace uso de una cantidad de datos muy pequeña para realizar el entrenamiento lo que

en una red neuronal CNN o RNN convencional no sería posible ya que se necesitaría una cantidad de datos mucho mayor para poder realizar el entrenamiento como se observa en la tabla de la figura 34.

Figura 34

Propiedades de la red neuronal recurrente LSTM.

```
[28]  model.summary()
      ✓ 0.1s

...   Model: "sequential"
      _________________________________________________________________
      Layer (type)                 Output Shape              Param #
      =================================================================
      lstm (LSTM)                  (None, 30, 64)            442112
      _________________________________________________________________
      lstm_1 (LSTM)                (None, 30, 128)           98816
      _________________________________________________________________
      lstm_2 (LSTM)                (None, 64)                49408
      _________________________________________________________________
      dense (Dense)                (None, 64)                4160
      _________________________________________________________________
      dense_1 (Dense)              (None, 32)                2080
      _________________________________________________________________
      dense_2 (Dense)              (None, 5)                 165
      =================================================================
      Total params: 596,741
      Trainable params: 596,741
      Non-trainable params: 0
      _________________________________________________________________
```

Fuente: Autores

En este punto se guarda el modelo para posteriormente cargarlo y ser utilizado, el cual contiene los pesos que son los parámetros más relevantes dentro de la red neuronal LSTM ya que estos parámetros se multiplican por los datos de entrada y a su vez con las salidas de las neuronas de la capa anterior; cada uno de estos parámetros es diferente para cada neurona ya que es allí donde se almacenan las secuencias que cada una de las neuronas aprendió en el proceso de entrenamiento sobre los datos; para este caso se almacena el modelo generado

usando model.save('action.h5') y para cargar los pesos del modelo se usó model.load_weights('action.h5'), tal como se aprecia en la figura 35.

Figura 35

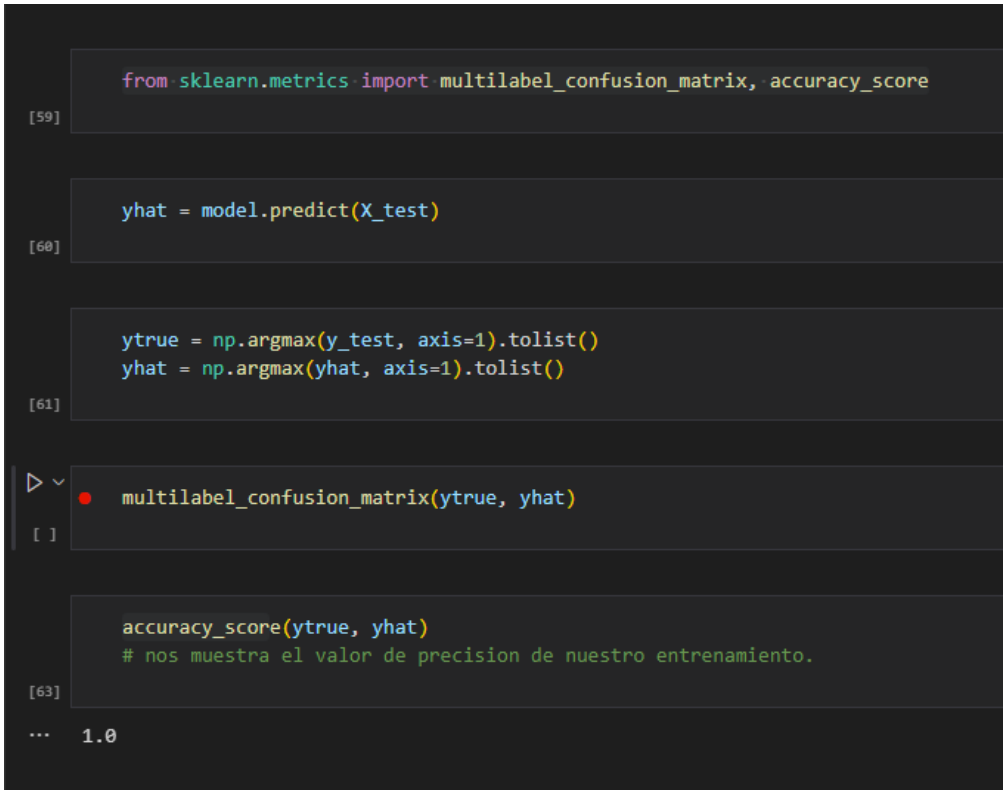
Almacenaje de pesos

Fuente: Autores

En la figura 36 donde se muestran las instrucciones que validan la precisión del modelo.

Figura 36

Evaluación del modelo



Fuente: Autores

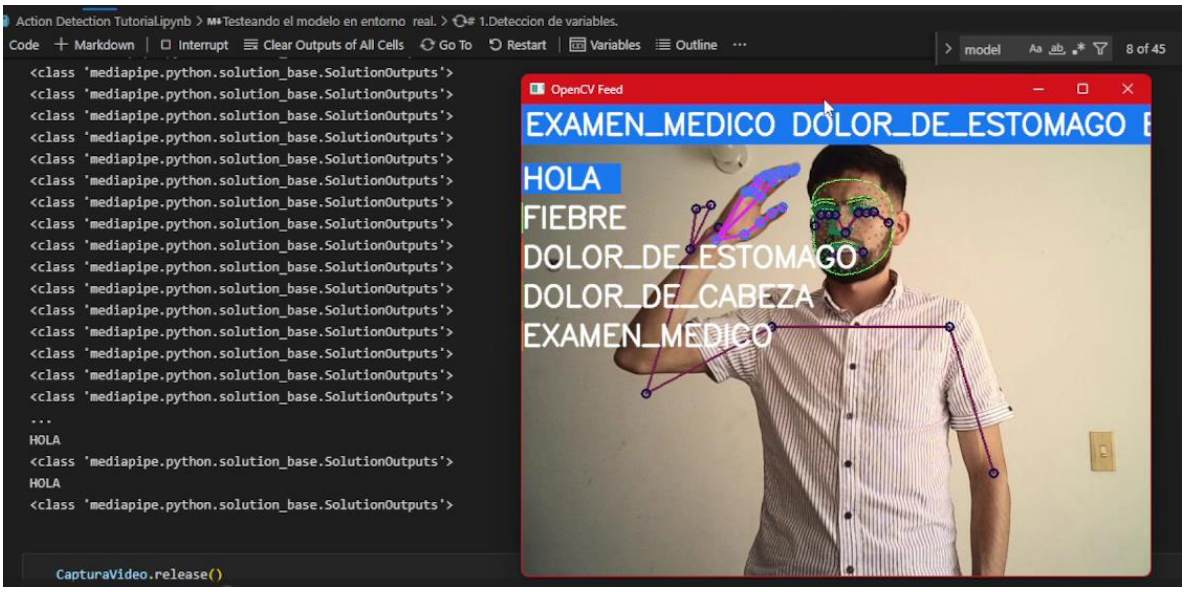
Para realizar la evaluación del modelo se hizo uso de una herramienta para el análisis predictivo de datos, usada habitualmente para realizar clasificación, regresión, agrupación, reducción de dimensionalidad y preprocesamiento de datos, la que en este caso se usó para evaluar el rendimiento del modelo, para lo cual se importa desde sklearn.metrics, una instancia de matriz de confusión y precisión la cual realiza la evaluación de la precisión de las clasificaciones que realice el modelo de acuerdo a cada una de las expresiones anteriormente establecidas.

El valor de precisión de 1.0 obtenido del modelo lo cual indica que el modelo está listo para realizar el paso final que será realizar la detección de las expresiones en tiempo real, ya que este valor es el valor máximo de precisión.

Como último paso se realiza la ejecución final donde se hace la predicción y detección de cada una de las 5 expresiones de la Lengua de Señas Colombiana, donde convergerán todos los elementos y pasos que se realizaron previamente, con el objetivo de realizar la detección y predicción de acuerdo con la probabilidad que el modelo estime de cada una de las expresiones, teniendo en cuenta los pesos que resulta para cada una.

Figura 37

Expresión 1



Fuente: Autores

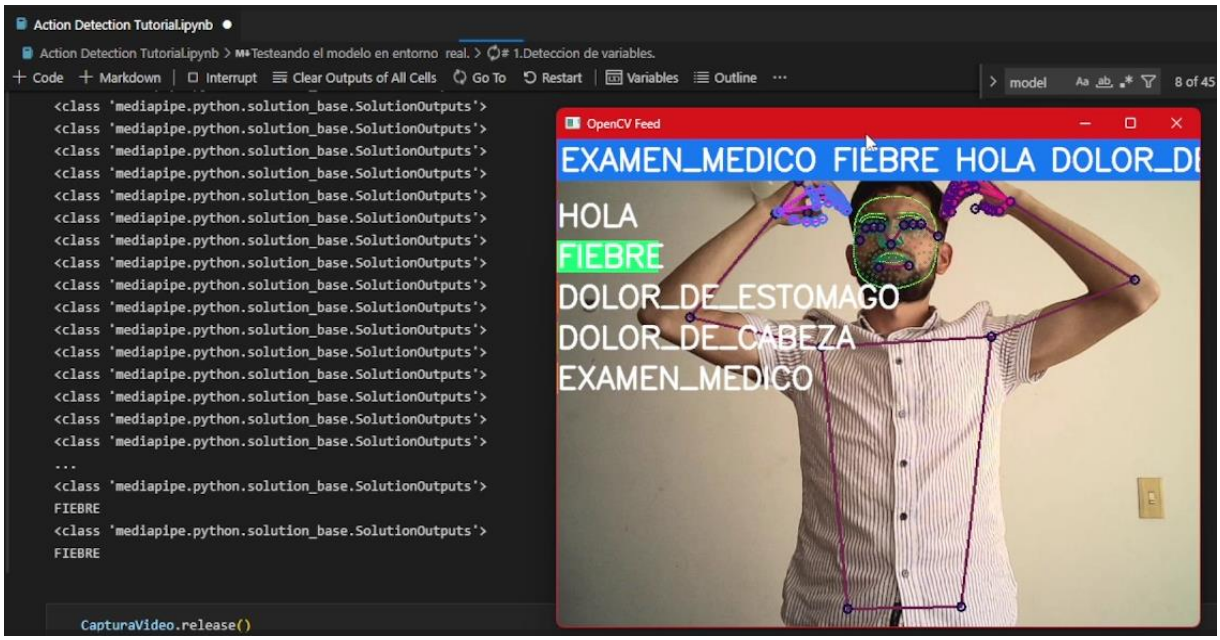
En las figuras 37, 38, 39, 40 y 41 se observa el resultado de la traducción de algunas expresiones de la Lengua de Señas Colombiana, como son: hola, fiebre, dolor de estómago, dolor de cabeza y examen médico respectivamente, señas que se enfocan a una consulta médica.

En pantalla aparecen, en texto español, las 5 expresiones. Cada una se resalta con un color diferente, para facilitar su diferencia al momento de la detección.

El usuario hace la seña frente a la cámara del dispositivo y este último muestra la palabra o expresión de la lista con la cual corresponde.

Figura 38

Expresión 2

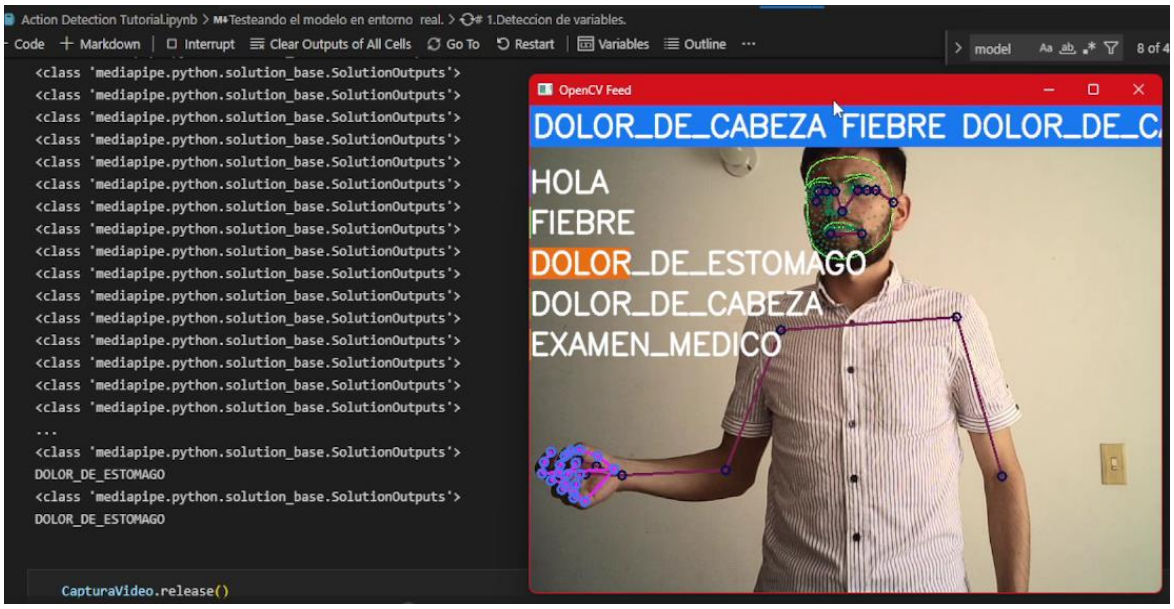


Fuente: Autores

Estas figuras permiten evidenciar que si una persona que se comunica en LSC realiza la seña frente a la pantalla del dispositivo donde se está ejecutando el modelo que se presenta en este trabajo, una persona que no conoce esta lengua puede leer su equivalencia en español y, en el caso del practicante de medicina o del médico en general, podrá identificar lo que el paciente con discapacidad auditiva le quiere comunicar.

Figura 39

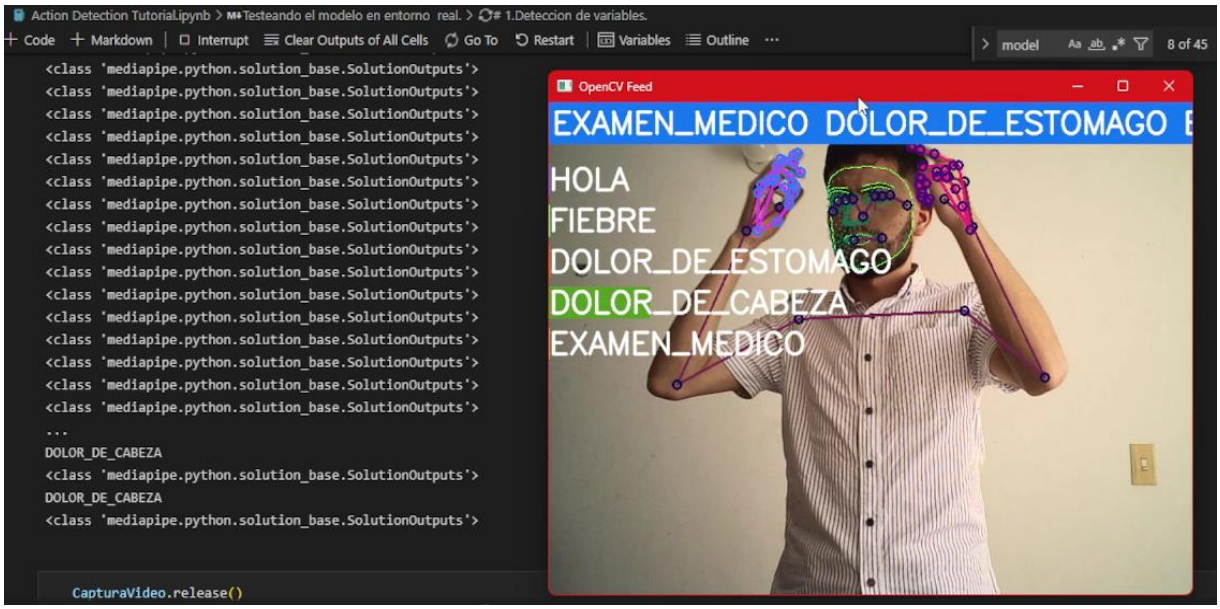
Expresión 3



Fuente: Autores

Figura 40

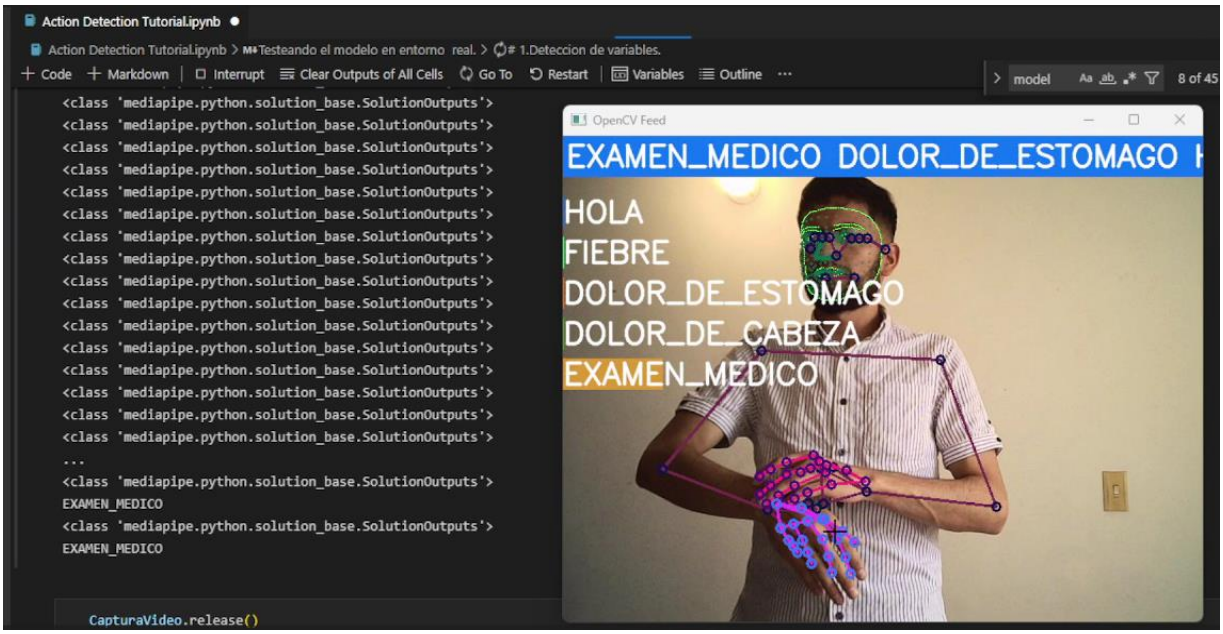
Expresión 4



Fuente: Autores

Figura 41

Expresión 5



Fuente: Autores

Conclusiones

En el desarrollo del presente proyecto se pudo evidenciar amplios avances tecnológicos en el área de la detección de acciones, análisis de datos, visión artificial, aprendizaje profundo, aprendizaje automático, detección de objetos, clasificación de imágenes, detección de expresiones faciales y, en menor cantidad, en la traducción de diferentes lenguas de señas, como la lengua de señas americana.

Se consultaron distintos proyectos y variedad de blogs, foros y demás plataformas pertinentes, donde se utilizan diferentes herramientas y técnicas que abordan el tema; se evidencia que la tecnología se ve poco aplicada a la Lengua de Señas Colombiana, por diferentes factores, tales como la necesidad una amplia capacidad de hardware, así como por la falta de conocimiento en el área, no solo en aspectos tecnológicos, sino también la lengua de señas misma y el ambiente en que se desenvuelven las persona con discapacidad auditiva.

Se ha evidenciado que el tipo de red neuronal utilizado LSTM RNN, tiene una gran precisión, gracias a que facilitan el procesamiento de secuencias de información en el tiempo; la potencia que posee la RNN para realizar el análisis y procesar grandes cantidades de datos en la detección y seguimiento de acciones por medio de video; fue el modelo más óptimo para adecuarlo a la detección de las expresiones corporales de la Lengua de Señas Colombiana ya que permite analizar en tiempo real y de forma rápida la posición del torso, las expresiones faciales y las manos.

Se ajustó el modelo de red neuronal de acuerdo con el objetivo de este proyecto, por lo que se maneja un set de datos con cinco expresiones de la Lengua de Señas Colombiana, haciendo el correspondiente ajuste en el software; igualmente se ajustó la estructura en cuanto a la cantidad de ciclos en el entrenamiento y el almacenamiento de valores de puntos de referencia del modelo, obteniendo excelentes resultados de acuerdo al intervalo de precisión y pérdida del modelo. El modelo implementado alcanzó una precisión con valores cercanos a uno y pérdidas con valores cercanos a cero, con lo que se evidencia que es una buena solución para el problema planteado.

Recomendaciones

Es importante que los investigadores continúen trabajando en el área de la Lengua de Señas Colombiana en producción de medios que capaciten al oyente en el aprendizaje de la lengua de señas o en el área de tecnología con la creación de herramientas computacionales que disminuyan la brecha comunicacional en personas oyentes y personas con discapacidad auditiva.

Es importante seguir explorando técnicas de Inteligencia Artificial para conseguir modelos que sean más flexibles y que generen menos carga de trabajo, para que se puedan ejecutar en computadoras con menos recursos y se les pueda dar un mayor aprovechamiento.

Los estudiantes de Ingeniería de Sistemas deben seguir estudiando y proponiendo soluciones computacionales para la población con discapacidad, que permita hacer de la inclusión una realidad en nuestro entorno.

Referencias

Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G., Davis A., Dean J., Devin M., Ghemawat S., Goodfellow I, Harp A., Geoffrey I., Isard M., Jia Y., Jozefowicz R., Kaiser L., Kudlur M., … Zheng X. (2015). *TensorFlow: Aprendizaje automático a gran escala en sistemas distribuidos heterogéneos*. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf

Anaconda (s.f.). *The world's most popular data science platform*. https://www.Anaconda.com/

Anaconda Documentation (s.f.). *Anaconda documentation*. https://docs.Anaconda.com/

Aniketdhole07. (2021, 25 abril). *Two-way sign language translator*. https://github.com/aniketdhole07/two-way-sign-language-translator

Aprende Ingeniería. (s.f.). *Detección y clasificación de manos*. https://github.com/AprendeIngenia/Deteccion-y-Clasificacion-de-Manos

Amazon Web Services Inc. (s. f). *¿Qué es una red neuronal?*. https://aws.amazon.com/es/what-is/neural-network/#:%7E:text=Una%20red%20neuronal%20es%20un,lo%20hace%20el%20cerebro%20humano

Amazon Web Services. (s.f.). *¿Qué es la inteligencia artificial (IA)?*. https://aws.amazon.com/es/machine-learning/what-is-ai/

Amazon Web Services. (s.f.). *¿Qué es el etiquetado de datos para el aprendizaje automático?*. https://aws.amazon.com/sagemaker/data-labeling/what-is-data-labeling/

Bobadilla, J. B. (2020). *Machine learning y deep learning: usando python, scikit y keras*. Ediciones de la U.

Calvo, D. (2017, 20 de julio). *Definición de red neuronal convolucional*. https://www.diegocalvo.es/red-neuronal-convolucional/

Carvajal F., Aguilar M., Agüera F., Aguilar F. (2022). *Clasificación de una imagen multiespectral del satélite de alta resolución espacial mediante redes neuronales*. https://docplayer.es/70443569-Clasificacion-de-una-imagen-multiespectral-de-satelite-de-alta-resolucion-espacial-mediante-redes-neuronales-artificiales.html

Cifuentes, A. C. Mendoza, E. M., Lizcano, M. L., Santrich, A. S. y Moreno, S. M. (2018). Desarrollo de una red neuronal convolucional para reconocer patrones en imágenes. *Investigación y Desarrollo en Tic, 10*(2), 7-17. https://revistas.unisimon.edu.co/index.php/identic/article/download/4007/4359

Congreso de la República de Colombia. (2002, 31 de Julio). Convención interamericana para la eliminación de todas las formas de discriminación contra las personas con discapacidad. [Ley 762 de 2002]. DO: 44.889. https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=8797

Congreso de la República de Colombia. (2005, 2 de Agosto). Por la cual se establecen normas tendientes a la equiparación de oportunidades para las personas sordas y sordociegas y se dictan otras disposiciones. [Ley 982 de 2005]. DO: 45.995. http://www.secretariasenado.gov.co/senado/basedoc/ley_0982_2005.html

Congreso de la República de Colombia. (2009, 31 de Julio). Por medio de la cual se aprueba la "convención sobre los derechos de las personas con discapacidad", adoptada por la Asamblea General de la Naciones Unidas el 13 de diciembre de 2006. [Ley 1346 de 2009]. DO: 47.427. http://www.secretariasenado.gov.co/senado/basedoc/ley_1346_2009.html

EvilPort2. (s.f.). *Sign-language*. https://github.com/EvilPort2/Sign-Language/blob/master/README.md

GitHub (s.f). *Deep learning for humans*. https://github.com/keras-team/keras

GitHub (s.f.) *Tensors and dynamic neural networks in python with strong gpu acceleration*. https://github.com/pytorch/pytorch

Github (s.f.). *Face mesh mediapipe*. https://google.github.io/mediapipe/solutions/face_mesh.html

Github (s.f.). *Mediapipe hands*. https://google.github.io/mediapipe/solutions/hands.html

Github (s.f.) *Mediapipe pose*. https://google.github.io/mediapipe/solutions/pose.html

Hicheri, L. (2011). *Traducción e interpretación en instituciones públicas*. https://cvc.cervantes.es/lengua/esletra/pdf/04/027_hicheri.pdf

IBM. (2021, 7 abril). *Recurrent neural networks*. https://www.ibm.com/cloud/learn/recurrent-neural-networks

Immune Technology Institute. (2022, 7 septiembre*). Librerías de python, ¿qué son y cuáles son las mejores?*. https://immune.institute/blog/librerias-Python-que-son/#:%7E:text=Librer%C3%ADas%20de%20Python%3A%20definici%C3%B3n&text=Es%20decir%2C%20las%20librer%C3%ADas%20de,de%20crear%20una%20interfaz%20independiente

Instituto Nacional del Cáncer. (s.f.). *Diccionario de cáncer del NCI*. https://www.cancer.gov/espanol/publicaciones/diccionarios/diccionario-cancer/def/neurona

Matplotlib (s.f.). *Visualization with python*. https://matplotlib.org/

Ministerio de Educación Nacional, Instituto Nacional para Sordos (INSOR) y Instituto Caro y Cuervo. (2006). *Diccionario básico de la Lengua de Señas Colombiana*. https://www.colombiaaprende.edu.co/sites/default/files/files_public/2022-04/Diccionario-lengua-de-senas.pdf

Ministerio de Salud y Protección Social. (2017 31 de mayo). Por medio de la cual se adopta el reglamento en cumplimiento de lo ordenado en la orden décima primera de la sentencia T-573 de 2016 de la Corte Constitucional y se dictan otras disposiciones. [Resolución número 1904 de 2017] T-573. https://www.minsalud.gov.co/sites/rid/Lists/BibliotecaDigital/RIDE/DE/DIJ/resolucion-1904-de-2017.pdf

Munisht06. (s.f.). *ASL-alphabet-recognition*. https://github.com/munisht06/ASL-Alphabet-Recognition

MediaPipe. (s.f.). *ml en vivo en cualquier lugar*. https://mediapipe.dev/

MediaPipe Pose. (2020.). *Mediapipe pose*. https://google.github.io/mediapipe/solutions/pose.html

NumPy.org (s.f.). *Manual. NumPy Documentación*. https://numpy.org/doc/stable/

OpenCV. (2020). Sobre https://opencv.org/about/

Pérez, D., Hurtado. N. (2014*). Estudio difuso de isótopo de carbono 13 e isotopo de oxígeno 18 en el límite Triásico-Jurásico/Tesis*. (Trabajo de grado, Universidad Central de Venezuela). ResearchGate. https://www.researchgate.net/publication/316741449_Estudio_Neuro_difuso_de_isotopo_de_carbono_13_e_isotopo_de_oxigeno_18_en_el_limite_Triasico-Jurasico

Renotte, N. R. (s.f.). *Action Detection for Sign Language*. https://github.com/nicknochnack/ActionDetectionforSignLanguage

Real Academia Española. (2021). *Diccionario de la lengua española*. https://dle.rae.es/se%C3%B1a

Rishabhjainps. (s.f.). *Facial expression recognition*. https://github.com/rishabhjainps/Facial-Expression-Recognition

Sotaquira, M. S. (2019, 20 julio*). ¿Qué son las redes LSTM?*. https://www.codificandobits.com/img/posts/2019-07-20/actualizacion-estado-oculto.gif

Solano, G. S. (2021, 11 agosto). *Detecta 543 puntos del cuerpo con mediapipe holistic*. https://omes-va.com/mediapipe-holistic-mediapipe-Python/

Visus, A. (2020). *¿Para qué sirve python? Razones para utilizar este lenguaje de programación*. https://www.esic.edu/rethink/tecnologia/para-que-sirve-Python#:%7E:text=El%20lenguaje%20de%20programaci%C3%B3n%20Python,aplicaciones%20empresariales%20fiables%20y%20escalables