LETW Documentación

Anthony Villalobos Hidalgo, tonyiish223@gmail.com

***Abstract*—Documentación acerca del proyecto open source, llamado LETW. Este proyecto busca ayudar a los desarrolladores a crear modelos de IA para el reconocimiento de acciones de una forma sencilla, por medio de una aplicación tipo consola.**

***Index Terms*—LETW, Action Recognition, Tensorflow, Mediapipe, LESCO**

I. Introduction

Este proyecto personal nació con el objetivo de mejorar mis habilidades de programación y, al mismo tiempo, facilitar mi comprensión sobre el funcionamiento de las redes neuronales. Durante su desarrollo inicial, identifiqué la necesidad de contar con sistemas que contribuyan a mejorar la calidad de la atención y los servicios disponibles para muchas personas en nuestro país, en especial para la comunidad que utiliza LESCO como medio de comunicación.

Si bien en otros países existen proyectos similares que ya son ampliamente utilizados, en el nuestro el desarrollo de este tipo de iniciativas no es tan común, o en muchos casos no se encuentran disponibles de manera abierta al público. Por esta razón, decidí crear un proyecto que pueda ser aprovechado y fortalecido por instituciones o personas con mayores recursos en hardware y tiempo, con el propósito de generar un impacto positivo en la calidad de vida de esta comunidad.

Aunque el proyecto incluye un modelo compilado en TensorFlow, capaz de realizar detección en tiempo real, aún no está completo. Por ello, se concibe como un proyecto open source, con la intención de que más personas puedan contribuir en su desarrollo y, de esta forma, lograr un modelo más robusto que brinde un mayor apoyo a quienes utilizan LESCO.

II. Fundamentos

Visión por computadora: Podemos describir la visión por computadora como un campo de la inteligencia artificial que, a través de cálculos matemáticos, técnicas de reconocimiento de acciones y principios relacionados con la percepción, busca realizar predicciones o generar resultados en forma de descripciones o valores a partir de una imagen. Se entiende por imagen un conjunto de píxeles que contienen valores numéricos.[1] Deep Learning: Es una rama del machine learning ampliamente utilizada en modelos de visión por computadora, debido a la gran cantidad de capas y redes que emplea. Se trabaja con volúmenes masivos de datos y con un alto nivel de abstracción. Los modelos basados en deep learning utilizan métodos como la retropropagación (backpropagation), los cuales permiten ajustar los parámetros internos del modelo para mejorar su rendimiento.[2] LSTM: Las redes LSTM (Long Short-Term Memory) son un tipo de red neuronal recurrente (RNN, Recurrent Neural Networks). Su estudio comenzó en la década de los noventa. A diferencia de las RNN convencionales, las LSTM pueden aprender de sus estados anteriores a lo largo de secuencias mucho más extensas, llegando a conservar información de hasta 1,000 timesteps previos. Esto les permite superar limitaciones como la pérdida de memoria a largo plazo y facilita que las redes neuronales aprendan de manera más eficiente y autónoma.[3] Capas Dense: Las capas densas son un tipo de red neuronal que se utiliza comúnmente en modelos de reconocimiento de imágenes y en el procesamiento de datos numéricos. Suelen emplearse en combinación con otras capas, ya que establecen conexiones completas entre todas las neuronas de una capa y las de la siguiente. Esto permite realizar una compresión de la información, lo que facilita que los modelos mejoren en la detección de patrones.[4] Mediapipe: Mediapipe es un framework de código abierto desarrollado por Google, encargado de la detección de poses, rostros y manos. Su objetivo principal es facilitar el desarrollo de aplicaciones de este tipo con un bajo consumo de recursos de hardware.[5]

III. Metodología

Antes de explicar el flujo de ejecución o entrar en más detalle acerca de la arquitectura del sistema, es importante mencionar algunos aspectos relevantes. Este proyecto se basa en el trabajo de reconocimiento de señas de Nicholas Renotte, titulado Action Detection for Sign Language, desarrollado en 2021.[6]. Al inicio, como mencioné anteriormente, el proyecto nació con la idea de ayudarme a mejorar mis habilidades de programación y mi comprensión sobre machine learning y redes neuronales. Durante el desarrollo, este enfoque fue cambiando y me fui centrando en crear una herramienta que ayudara a las personas a desarrollar modelos de manera más sencilla.

Es importante recalcar que, originalmente, la lógica del proyecto estaba diseñada para ejecutarse desde un proyecto o un cuaderno de Jupyter. Se desarrolló una aplicación tipo consola en Python. La lógica original estaba pensada para obtener y extraer datos de aproximadamente 30 videos, con 30 frames por cada uno, para cada acción. Es decir, si tenemos 2 acciones, se tomarían un total de 60 videos y se extraerían 60 frames, esto desde video captado en tiempo real desde la cámara.

Durante el desarrollo, me di cuenta de que este proceso no era tan eficiente si quisiéramos trabajar con una mayor cantidad de acciones y datos, como se explica más adelante en este documento. Debido a esto, modificamos la lógica para poder trabajar con videos ya grabados, aumentando así el tamaño del dataset y creando un modelo más robusto y eficiente.

Es importante recalcar que el modelo se ejecuta usando Python 3.8, debido a la compatibilidad con ciertas librerías y,

principalmente, por el uso de Mediapipe Holistic, un modelo de Google que actualmente es legacy, y que ha sido reemplazado por Mediapipe Tasks. Más adelante, se discutirán posibles cambios en la lógica del proyecto para mejorar su funcionamiento.

Sin más por el momento, continuaremos con la explicación de la lógica del sistema.

El proyecto está desarrollado en Python, a continuación se explicará la lógica de cada clase.

A. Clase principal: App.py

Es la clase principal del sistema y se encarga de manejar el flujo de procesamiento de la aplicación. Dentro de este archivo se definen variables globales utilizadas por las demás clases del sistema, las cuales se pasan como parámetros cuando es necesario. Entre ellas se encuentran: repetitions, que especifica la cantidad de repeticiones de los videos o el total de videos que se van a procesar, frames, que indica la cantidad de fotogramas que se extraerán de cada video procesado, siendo importante tener en cuenta que, a mayor cantidad de frames, mayor será el tiempo de procesamiento y signs, un arreglo que contiene las palabras correspondientes a las señas que el modelo reconocerá. En caso de procesar un gran número de palabras, también se puede utilizar un archivo .txt que las contenga todas.

Además, se definen las variables video paths y mp path. Ambas provienen del resultado de un método definido en otra clase y obtiene la ruta donde se guardan los videos, mientras que la segunda indica la ruta donde se almacenarán los datos utilizados para el entrenamiento. La variable menu = True se utiliza para controlar el ciclo del menú, permitiendo iterar sobre las opciones hasta que el usuario decida salir.

La opción 1 ejecuta la lógica de la clase SetUp, encargada de crear las carpetas y directorios necesarios para el correcto funcionamiento del sistema. La opción 2 permite extraer los keypoints o frames de un video específico, o de varios videos definidos en la variable signs. La opción 3 realiza una tarea similar a la opción anterior, con la diferencia de que no se extraen los keypoints como arreglos de NumPy, sino que se muestra la identificación de las personas en uno o varios videos, según corresponda. Tanto la opción 2 como la 3 instancian la clase VideoBatchProcessor, encargada de ejecutar uno o varios videos dependiendo de la elección del usuario, y desde esta clase también se instancian otras clases según sea necesario.

La opción 4 crea una instancia de la clase DataLabelling, desde donde se maneja la lógica de etiquetado de los datos. La opción 5 se encarga de ejecutar la lógica necesaria para entrenar el modelo. La opción 6 ejecuta la lógica que permite realizar detecciones en tiempo real utilizando el modelo previamente creado y compilado. Finalmente, la opción 7 cierra la ejecución del sistema.

B. Clases secundarias

VideoBatchProcessor es la clase encargada de ejecutar uno o varios videos según la elección del usuario y a continuación se explica su lógica. Dentro de la función init se reciben como parámetros self, directory y repetitions y en este método se instancian otras clases según sea necesario. En este caso se crea el objeto directory que se encarga de instanciar el directorio de donde provienen los videos, extractor que instancia la clase KeypointExtractor encargada de extraer los keypoints, processor que instancia la clase ImageProcessor responsable de convertir las imágenes o frames a los modelos de representación de color necesarios, dataextractor que instancia la clase encargada de extraer los keypoints requeridos, repetitions que representa la cantidad de videos con los que se va a trabajar y finalmente counter y logger que instancian la clase Utilities y su método logger para generar registros (logs). El método run es el encargado de ejecutar un solo video y funciona únicamente si en la carpeta de videos existe un solo archivo nombrado directamente dentro de ella, en caso de tener subdirectorios o subcarpetas se producirá un error, para lo cual se crearon otros métodos dentro de la misma clase. En el método run se crean variables que permiten obtener la ruta del directorio a utilizar en el formato para el cual el código fue optimizado, también se define un counter que se va incrementando después de cada repetición o video procesado y starttime que permite iniciar un contador para estimar la duración de la ejecución de la lógica. Después de esto se inicia un ciclo for que permite iterar sobre cada uno de los videos, es decir, si tenemos una lista de 100 videos se realizarán 100 repeticiones de la lógica dentro de este ciclo. En el ciclo se crean o instancian las transformaciones que ayudan a generar un poco de data augmentation en caso de trabajar con videos duplicados, aunque esto no es del todo recomendable ya que puede aplicarse en una sola repetición pero si se realizan dos o más el modelo podría confundirse durante la etapa de entrenamiento. Posteriormente se incluye un bloque encargado de imprimir información en consola y de guardarla en los logs, a continuación se instancia la clase encargada de aplicar las detecciones mediante MediaPipe Holistic, es importante recalcar que este modelo es legacy, mientras que el modelo actual a la fecha de redacción de este documento es MediaPipe Tasks, que ofrece más funciones aunque su implementación es más extensa y un poco más tediosa. Después de realizar las detecciones se realiza una verificación, si existen resultados se indica al usuario que los keypoints han sido extraídos de manera exitosa y de lo contrario se muestran los mensajes de error correspondientes. Este método es el que se ejecuta cuando el usuario selecciona la opción 3 del menú principal y posteriormente la opción 1. Las demás funciones o métodos de esta clase se encargan de una lógica similar con la diferencia de que se ejecutan dependiendo de las opciones elegidas por el usuario. Como se explicó al inicio, la opción 3 del menú principal permite ejecutar la lógica sin extraer datos y en caso de querer modificar la lógica de las funciones que sí extraen datos se deben editar las funciones Extractsinglepath y Extractparentpath.

ImageProcessor: Esta clase se encarga de procesar la imagen con el modelo seleccionado, en este caso Mediapipe Holistic. Anteriormente se había mencionado esta clase, pero ahora se explicará con más detalle. Cuenta con una función llamada mediapipedetection, que se encarga de realizar la conversión de colores de las imágenes extraídas. Esto es necesario para

Fig. 1. Muestra gráfica de cómo se ven los landmarks o keypoints detectados por el sistema de reconocimiento de Mediapipe.

poder realizar la detección, ya que para procesar el video se utiliza OpenCV, mientras que para detectar poses, personas y otros elementos se emplea Mediapipe Holistic, que utiliza formatos de color distintos. Además, incluye otra función llamada processvideo, que se encarga de pasar los fotogramas a través de Mediapipe Holistic para realizar las detecciones de poses y superponer los keypoints correspondientes. Una vez finalizado el proceso, devuelve lastframe y lastresult, es decir, un fotograma con los keypoints y la detección realizada.

KeypointExtractor: Esta clase se utiliza en el proceso de VideoBatchProcessor. Su función principal es tomar los resultados de ImageProcessor, donde se aplicó la detección con Mediapipe. Para entender mejor esto, se puede revisar la Figura 1, que muestra cómo se ven las detecciones realizadas por Mediapipe. En esta etapa se obtienen los keypoints o puntos de referencia de las poses, es decir, las distintas partes del cuerpo de la persona que aparece en la imagen. La clase se encarga de extraer estos resultados y guardarlos como arreglos de NumPy, que luego se utilizan para entrenar el modelo. Una de las primeras funciones de la clase se dedica a normalizar los keypoints. Para esta normalización se utilizan los hombros como referencia. Para conocer la correspondencia de cada keypoint, se puede consultar la documentación de Mediapipe Holistic o del nuevo modelo Mediapipe Tasks. Esta normalización es muy importante, ya que si no se realiza, al extraer los keypoints podríamos estar usando coordenadas absolutas (x, y, z) en lugar de puntos relativos. Por ejemplo, si en un frame el sujeto tiene x = 21, y = 18, z = -2, se extraería esa posición exacta y, en tiempo real, si el sujeto no está exactamente en esas coordenadas, el modelo podría no reconocer la acción. La normalización soluciona esto tomando un punto de referencia del cuerpo y calculando todas las posiciones relativas a ese punto.[7] Según la documentación y diversos foros, esto también ayuda a que el modelo no se confunda con personas de diferentes tamaños, formas o alturas.[8]

La lógica de normalización para la pose consiste primero en verificar que haya suficientes landmarks detectados. Luego se extraen los valores de los hombros, por ejemplo, para el hombro izquierdo (landmark 11), cada landmark tiene cuatro valores: x, y, z y visibility. Para este cálculo se utilizan únicamente x, y, z, ignorando visibility, y se multiplica el índice del landmark por 4 para ubicar correctamente los valores en el arreglo. Después se calcula el centro de los hombros sumando las coordenadas del hombro izquierdo y derecho y dividiendo entre 2. Se crea un arreglo donde se guardarán los landmarks ya normalizados y se recorre cada landmark de 4 en 4, tomando las coordenadas x, y, z y aplicando la normalización centrando y escalando según la distancia entre los hombros. Finalmente, se devuelve un arreglo de NumPy con los valores normalizados. Para las manos se utiliza la muñeca como punto de referencia en lugar de los hombros y se aplica una lógica prácticamente idéntica: se centra y se escala según la distancia entre la muñeca y la punta del dedo medio. La última parte de la clase se encarga de extraer los keypoints y devolverlos. Si no hay resultados de detección, se devuelve un arreglo de ceros. Si hay resultados, se aplica la extracción de pose y manos, se convierte la lista en un vector unidimensional, se concatenan todas las partes y se devuelve un vector de tamaño fijo, listo para ser usado en entrenamiento o inferencia.

DataExtraction: La clase DataExtractor se encarga de tomar uno o varios videos y extraer sus landmarks usando MediaPipe Holistic. Su objetivo es procesar cada video, obtener los keypoints y guardarlos como arreglos de NumPy, que posteriormente se utilizan para entrenar el modelo. En el init se definen los parámetros principales: la cantidad de repeticiones que se van a realizar por cada acción, la cantidad de frames que se extraen por secuencia y la lista de acciones válidas. También se instancian el drawer, que dibuja los landmarks, y el extractor, que convierte los resultados de Mediapipe en vectores numéricos. Además, se define la ruta donde se guardan los datos y un logger para registrar todo el proceso. La función mediapipedetection convierte un frame de BGR a RGB, lo procesa con el modelo Holistic y devuelve tanto la imagen procesada como los resultados del modelo. El método principal es processvideo. Primero verifica si la ruta corresponde a un directorio o a un archivo de video. Si es un directorio, toma el nombre como acción y los videos internos como ejemplos, si es un archivo, identifica en su nombre cuál de las acciones de la lista corresponde. Luego se carga el modelo Holistic y, para cada repetición definida, se abre el video, se calcula el total de frames y se seleccionan los índices que se van a procesar de manera uniforme con NumPy. En cada frame seleccionado se realiza la detección, se dibujan los landmarks y se extraen los keypoints. Si todo funciona correctamente, se guardan como archivos .npy dentro de la carpeta correspondiente a la acción y a la secuencia actual. En caso de errores, se reportan en consola y en los logs. Una vez procesados todos los frames, se cierra el video, se incrementa la secuencia y se pasa al siguiente. Al final, OpenCV cierra todas las ventanas abiertas. En resumen, esta clase automatiza todo el flujo de procesamiento: toma los videos, extrae un número fijo de frames, aplica MediaPipe Holistic sobre ellos, normaliza y guarda los keypoints, dejándolos listos para el entrenamiento del modelo.

DataLabelling: Esta clase, en resumen, se encarga de etiquetar los datos. Toma como parámetros las señas, la cantidad de repeticiones y la cantidad de frames. Además, dentro de su método init se asigna una variable que especifica el directorio donde se obtendrán los datos, en este caso los arreglos de NumPy. A partir de ahí, se crea un arreglo que contendrá los valores de las secuencias, es decir, las coordenadas y sus respectivas etiquetas. Las etiquetas corresponden al número específico asignado a cada acción. El proceso funciona de

la siguiente manera: se recorre cada signo, para cada signo se accede a su carpeta dentro de MP Data, en esa carpeta se encuentra una cantidad de repeticiones o videos, y dentro de cada repetición, una serie de frames. Una vez hecho esto, se carga cada arreglo de NumPy en la lista window. Luego, se añade esa secuencia completa junto con la etiqueta correspondiente, que proviene del índice actual. Es decir, si la etiqueta "1" corresponde a "Árbol", todos los datos de esa secuencia se almacenan bajo esa etiqueta. Después, se concatenan todos los datos en la variable xcoordinate, que tendrá la forma (N, frames, F), donde N representa el total de secuencias, frames la cantidad de frames por secuencia (en este caso 30) y F la cantidad de puntos extraídos. Recordemos lo mencionado anteriormente: se consideran todos los puntos detectados por MediaPipe, en este caso un máximo de 1662. Finalmente, se realiza la separación de los datos en lotes o subconjuntos para test, validación y entrenamiento. En total, se reserva un 15 por ciento de los datos para test, otro 15 por ciento para validación y el 70 por ciento restante para entrenamiento. Es importante recalcar que esta clase se puede ejecutar de manera directa, pero no es necesario, ya que al ejecutar la clase training lstm, uno de los primeros pasos que realiza es ejecutar la lógica de etiquetado.

Un ejemplo de esta clase en ejecución es el que vemos en la figura 2.

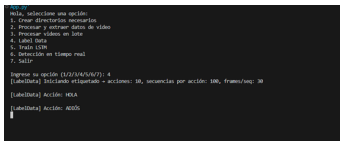

Fig. 2. Ejemplo de la clase DataLabelling en ejecución

Una vez que hemos terminado la ejecución de este proceso, podremos ver el resultado de los datos y verificarlos tanto en los logs como en la consola. Una representación de esto se muestra en las figuras 3 y 4.

Fig. 3. Resultado de la clase DataLabelling mostrado en la consola.

Fig. 4. Resultado de la clase DataLabelling registrado en los logs del programa.

Training LSTM: Esta clase de nuestro código es importante, ya que se encarga de realizar el entrenamiento del modelo. Es una clase a la que debemos prestar atención durante su ejecución, pues nos permite monitorear los resultados y el comportamiento del modelo, además de realizar cambios o ajustes si fuera necesario. En el constructor, tomamos el modelo de TensorFlow que vamos a usar, instanciamos nuestro labeller, definimos las señas y configuramos el logger. Luego, en el método build_model, encargado de construir el modelo, comenzamos especificando una carpeta donde se guardarán los logs del entrenamiento, para posteriormente visualizarlos en la aplicación web local de TensorFlow llamada TensorBoard. Dentro de esta clase, contamos con un pequeño menú diseñado para facilitar la ejecución de la lógica. La primera opción permite entrenar el modelo. Para ello, creamos la variable es (early stopping), encargada de monitorear en tiempo real los resultados del entrenamiento. Esta tiene como parámetro una paciencia de 20 epochs, si después de ese número de iteraciones no hay mejoras, el entrenamiento se detiene y se restauran los mejores pesos. (Es recomendable revisar un poco sobre redes neuronales para entender con más detalle el concepto de pesos: en pocas palabras, son los parámetros que el modelo ajusta para decidir qué neuronas activar durante el aprendizaje). Posteriormente, definimos otra variable llamada rlrop, que supervisa la validation loss (val_loss). Si esta no mejora después de 5 epochs, el learning rate se reduce a la mitad, lo cual ayuda a que el modelo continúe aprendiendo de manera más precisa. A continuación, configuramos la arquitectura de la red neuronal: especificamos la cantidad de neuronas, el tipo de capas y su número. Empezamos con tres capas LSTM. La primera tiene 64 unidades y utiliza la activación tanh, adecuada para secuencias largas. Esta capa también incluye dropout para evitar overfitting (es decir, que el modelo aprenda características irrelevantes), batch normalization y el parámetro return_sequences=True para devolver toda la secuencia. Luego pasamos a la segunda capa LSTM, con menos unidades, y finalmente a una tercera capa LSTM, que resume toda la secuencia y la envía a dos capas densas (Dense): una de 32 y otra de 16 neuronas. Estas capas tienen como función generalizar los datos y preparar la salida para la clasificación final. Es importante aclarar que, dado que trabajamos con reconocimiento de acciones y la cantidad de clases es reducida, usamos una arquitectura relativamente pequeña. Si aumentara la cantidad de acciones, sería necesario incrementar la cantidad de capas o unidades, si disminuyera, se podrían reducir. La configuración actual fue la que mejores resultados nos brindó en la práctica. También es fundamental recalcar que, aunque la arquitectura esté bien diseñada, es necesario contar con un dataset lo suficientemente grande para evitar overfitting y obtener buenos resultados. Además, se podrían probar capas GRU en lugar de LSTM, sin embargo, en la práctica, los resultados suelen ser ligeramente inferiores y los tiempos de entrenamiento mayores. Una vez realizado el entrenamiento y obtenidos resultados satisfactorios, el proceso finaliza, permitiendo al usuario ejecutar pruebas de confusión y detección. Si los resultados cumplen las expectativas, se brinda la opción de guardar el modelo en la carpeta principal, para posteriormente poder utilizarlo en detecciones en tiempo real. Al ejecutar el proceso de entrenamiento en nuestra consola y revisar los logs, podremos monitorear en tiempo real el progreso y los resultados del modelo. Un ejemplo en la siguiente figura:

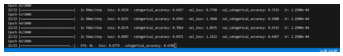

Fig. 5. Ejemplo de los resultados en consola a la hora del entrenamiento del modelo

Monitorear estas salidas es importante, ya que nos permite observar cómo avanza el aprendizaje y, además, nos ayuda a detener el proceso para evitar malgastar tiempo. Actualmente, con la cantidad de datos y redes neuronales que utiliza este modelo, los resultados del entrenamiento se sitúan entre el 10 % y el 20 %, mientras que la precisión (accuracy) del modelo se mantiene por encima del 90 %. A continuación, lo analizamos en detalle.

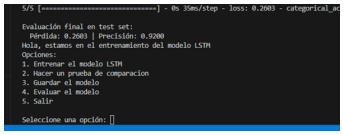

Fig. 6. Ejemplo del resultado del entrenamiento de un modelo basado en las características específicas que se detallan en este documento.

Al menos en teoría, ahora que estamos dentro de la clase training LSTM, podemos realizar pruebas de comparación y de confusión para obtener un punto de referencia (benchmark) sobre nuestros resultados. Un ejemplo se muestra a continuación:

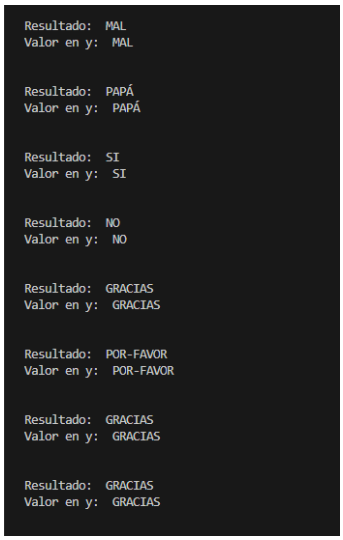

Fig. 7. Resultados de una prueba de comparación con el modelo entrenado, basado en las características específicas detalladas en este documento.

Además de una prueba de comparación como la anterior, que se encarga de brindar datos reales de las acciones y compararlos con los resultados del modelo, también contamos con pruebas de confusión, las cuales nos permiten obtener un benchmark o una mejor idea del rendimiento (performance) del modelo. A continuación, se muestra un ejemplo:

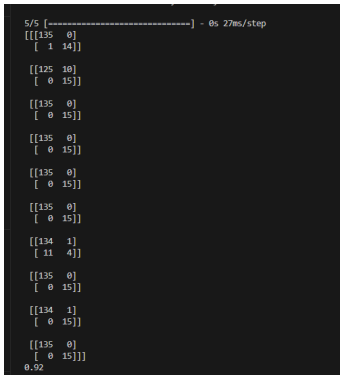

Fig. 8. Resultado de la prueba de confusión

Cada bloque, o cada matriz de 2×2 que vemos en la imagen, representa una de las acciones de nuestro modelo. Dentro de estas matrices, los valores indican lo siguiente: índice 00, verdaderos negativos, 01, falsos positivos, 10, falsos negativos, y 11, verdaderos positivos. En la siguiente imagen se presenta esto para facilitar su comprensión:

Fig. 9. Representación gráfica del significado de los datos mostrados como resultado en la figura 8.

Desglose y explicación:

- **VN:** Verdadero negativo
- **FN:** Falso negativo
- **FP:** Falso positivo
- **VP:** Verdadero positivo

Basándonos en la información de la figura 8, tomamos los datos de la primera matriz que aparece. Según estos datos, la interpretación sería que tuvimos 135 verdaderos negativos y 0 falsos positivos. Esto significa que, cuando el modelo encuentra acciones que no corresponden a la acción en cuestión, no las clasifica erróneamente como dicha acción.

Por otro lado, con los datos de la segunda fila, observamos que se detectó un falso negativo y 14 verdaderos positivos. Esto nos indica que, en una sola ocasión, el modelo clasificó incorrectamente la acción como negativa.

Utilities: Esta clase no es la principal de entrenamiento, pero sus métodos nos ayudan a manejar rutas de videos, transformaciones y aumentos de datos. Todos sus métodos son estáticos, lo que significa que podemos utilizarlos sin necesidad de crear una instancia de la clase.

El primer método estático es get_video_paths. Su función es sencilla: devuelve una lista con todas las rutas de los videos dentro de un directorio específico. Por defecto, busca archivos con extensiones .mp4, .avi o .mov. Por ejemplo, si tenemos una carpeta con 10 videos, este método nos devolverá una lista de 10 rutas, cada una apuntando a un archivo de video.

El siguiente método, get_video_by_action, es un poco más avanzado. Su función es organizar los videos por acciones.

Primero, recorre todas las carpetas dentro del directorio principal. Cada carpeta se asume que representa una acción diferente. Luego, busca los videos dentro de cada carpeta y los guarda en un diccionario. La clave del diccionario es el nombre de la acción en mayúsculas y el valor es una lista con las rutas de los videos correspondientes. Esto nos permite, por ejemplo, acceder fácilmente a todos los videos de la acción "SALTO" mediante video_dict['SALTO'].

El método flip_horizontal es muy simple: recibe un frame de video y lo devuelve volteado horizontalmente. Esto es útil cuando queremos aumentar nuestro dataset simulando que la acción ocurre de manera inversa.

El método random_augmentation se encarga de aplicar transformaciones aleatorias a un frame. Primero, elige aleatoriamente entre las opciones 'brightness' y 'none'. Si la opción es 'brightness', ajusta el brillo del frame sumando o restando un valor aleatorio al canal de brillo en el espacio HSV. Si es 'none', simplemente devuelve el frame tal cual. Esto ayuda a aumentar la variedad de datos para que el modelo no se vuelva demasiado dependiente de un solo tipo de imagen.

Finalmente, el método setup_logging configura un sistema de logs para registrar la información del proceso. Se puede especificar un archivo de log (por defecto, app.log). Define el nivel de log, en este caso INFO, y el formato del mensaje, incluyendo la fecha, el nivel de severidad y el mensaje. Esto permite monitorear de manera más organizada lo que ocurre durante la ejecución del proyecto.

IV. Limitaciones

Es importante tener en cuenta que, como proyecto personal, he intentado mejorarlo continuamente para que sea lo más robusto posible y funcione de manera óptima. Planeo actualizarlo poco a poco y espero que, en algún momento, pueda llegar a manos de personas o instituciones con mayores recursos que realmente puedan potenciar su desarrollo.

En caso de que ya se esté trabajando en él, quiero comentar algunas limitaciones y recomendaciones. Como se mencionó a lo largo de este documento, me basé en el proyecto de Nicholas Rennote sobre Action Recognition para el reconocimiento de señas.[6], Este proyecto originalmente toma los datos para entrenar el modelo mediante la cámara de la computadora en la que se ejecuta. Nosotros realizamos modificaciones, ya explicadas, para trabajar con videos que deben guardarse en una carpeta específica dentro del directorio del proyecto.

Es importante tener en cuenta que, dependiendo de la cantidad de acciones con las que trabajemos, se necesita un número considerable de videos. Por ejemplo, el modelo que entrené con la arquitectura especificada en este documento alcanzó un accuracy del 92 %. En este caso, grabé 50 videos por cada acción y apliqué una transformación para duplicar la cantidad a 100 videos por acción, trabajando con un total de 10 señas a reconocer.

Al inicio del desarrollo, entrené un modelo con 6 acciones y 25 videos por acción (50 tras la duplicación) y obtuve un accuracy de 89-90 %. Para brindar un contexto adicional, el modelo creado por Nicholas Rennote reconocía 3 acciones con un desempeño similar, basado en 30 grabaciones por acción.

Con este ejemplo quiero recalcar que la cantidad de videos necesarios para mantener un buen accuracy debe aumentar según la cantidad de acciones. Aunque esta relación pueda parecer lineal, en realidad no lo es, ya que llega un punto en el que cada video adicional aporta menos información relevante. En ese momento, será necesario ajustar o mejorar la arquitectura de la red para seguir aumentando el rendimiento del modelo.

Ahora voy a abordar otro punto importante, relacionado con los videos. A lo largo de los 9 meses de desarrollo de este proyecto y tras numerosas pruebas, me di cuenta de que se obtienen mejores resultados cuando los videos que alimentan al modelo son cortos y bastante específicos con las señas a detectar. Es decir, es más efectivo proporcionar videos de 2-3 segundos donde únicamente se ejecute la seña, en lugar de videos largos que incluyan mucho ruido o movimientos no relacionados con la acción a reconocer. Con esto me refiero a que los sujetos no deben realizar movimientos adicionales que no correspondan a la seña específica.

Con respecto a la calidad de los videos, obtenemos los mejores resultados cuando se utilizan videos sin audio, ya que esto reduce su tamaño y facilita que el hardware los procese con menor consumo de recursos. Además, es recomendable que la resolución sea igual o superior a 854×480 píxeles, es decir, 480p.

Con respecto al hardware, es importante tener en cuenta que, si vamos a realizar detecciones en tiempo real una vez desarrollado el modelo, debemos asegurarnos de que la calidad de nuestra webcam sea buena y que el ambiente de pruebas tenga una iluminación adecuada. Si la iluminación o la calidad de la webcam no son óptimas, MediaPipe Holistic tendrá dificultades para realizar el reconocimiento correctamente, y el accuracy con el que detecte objetos o personas será menor.

Para finalizar, debido a que utilizamos una versión de MediaPipe que ya se considera legacy, no es posible actualizar a versiones más recientes de Python. Por este mismo motivo, tampoco podemos aprovechar recursos como las tarjetas gráficas al trabajar desde una máquina con sistema operativo Windows, las cuales representan la mejor opción de hardware para este tipo de desarrollo según la documentación oficial de los frameworks. Esto sí sería posible al trabajar desde una distribución de Linux. En ese caso, también sería necesario actualizar o modificar la configuración de la red neuronal para aprovechar al máximo la arquitectura. Sin embargo, en nuestro caso específico, trabajamos desde Windows. Mi recomendación es actualizar la lógica del sistema para utilizar el modelo actual de Google: MediaPipe Tasks.

V. Nota final

A la fecha en que finalizo este documento, he tomado la decisión de trabajar en un modelo más robusto, aun cuando esto requiera una cantidad significativa de tiempo. Espero poder compartirlo con ustedes muy pronto.

VI. Agradecimientos

A pesar de que, como he especificado en varias ocasiones a lo largo de este documento, este proyecto no tiene ningún

vínculo con mis estudios de grado ni con mi trabajo profesional, ya que se trata principalmente de una iniciativa personal, deseo expresar mi agradecimiento al Ingeniero y Máster José Luis Medrano Cerdas por su ayuda y recomendaciones para el desarrollo del proyecto; al Ingeniero Jairo Alberto Zúñiga Gómez por su colaboración y sugerencias; al Ingeniero y Máster Luis Córdoba por sus valiosas recomendaciones; y a Mariana Camacho Ruiz por su apoyo y orientación.

REFERENCES

[1] A. Hanson, *Computer vision systems*. Elsevier, 1978.
[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[3] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019.
[4] M. Samad, "Understanding dense neural networks: A deep dive into architecture, learning, and applications," *Major Digest*, 2024.
[5] I. Grishchenko and V. Bazarevsky, "Mediapipe holistic — simultaneous face, hand and pose prediction, on device," 2020.
[6] N. Renotte, "Actiondetectionforsignlanguage," 2021.
[7] H. Hu, H. Wang, and X. Wang, "A method for standardized wear detection of masks based on key point detection," *Journal of Computational Methods in Science and Engineering*, vol. 23, no. 6, pp. 2813–2823, 2023.
[8] Crtony03, "Pregunta sobre la normalización de keypoints en mediapipe," 2025.