UNIVERSIDAD DEL BÍO-BÍO, CHILE

FACULTAD DE CIENCIAS EMPRESARIALES

Departamento de Sistemas de Información

RECONOCIMIENTO EN TIEMPO REAL DEL ALFABETO DE LENGUA DE SEÑAS CHILENA EMPLEANDO APRENDIZAJE AUTOMÁTICO

PROYECTO DE TÍTULO PRESENTADO POR LUIS BENÍTEZ ANDRADE Y RICARDO CATRIL CARRILLO DE LA CARRERA INGENIERÍA CIVIL INFORMÁTICA

DIRIGIDA POR CLEMENTE RUBIO MANZANO

2022

Agradecimientos

Primeramente dar gracias a Dios, pues gracias a él he llegado hasta donde estoy. A mi familia, que siempre me apoyó y me dieron la ayuda que necesité. A mis amigos, Franco, Edgar y mi compañero de proyecto de título Ricardo, que muchas veces me ayudaron y motivaron cuando lo necesité. Finalmente agradecer a mi profesor guía Clemente y demás profesores de la universidad, por no sólo el conocimiento entregado sino por los valores y enseñanzas de vida que aprendí de ellos.

Luis Benítez A.

En primer lugar, quiero expresar mi agradecimiento a mi padre y madre, quienes siempre me dieron su apoyo moral desde que tengo recuerdo. También quiero dar gracias a mis amigos con quienes compartí lindos recuerdos, haciendo amena esta etapa universitaria llena de altibajos. Al profesor Clemente Rubio y Luis Benítez por acompañarme en este proyecto lleno de enseñanzas y trabajo arduo, que sin duda alguna darán frutos para mi crecimiento personal y profesional.

Ricardo Catril C.

Resumen

En Chile, existen 712.005 personas con discapacidad que tienen algún grado de pérdida de audición, de ellas se estima que 179.268 personas tendrían sordera total. La comunidad sordomuda ha sufrido problemas de inclusión debido al desconocimiento de las personas sobre las limitaciones a las cuales se pueden llegar a enfrentar. Una de las iniciativas para mejorar la integración de comunidad sordomuda ha sido la promulgación de la ley 21.303 en el año 2021, que en su artículo 26 señala la lengua de señas chilena como el lenguaje natural, originario y patrimonio intangible de las personas sordas, así como también el elemento esencial de su cultura e identidad individual y colectiva. El gobierno también ha implementado cursos de lengua de señas para estudiantes de enseñanza básica y media como plan piloto.

Por otro lado, se han impulsado propuestas tecnológicas que tratan de disminuir ésta brecha. En particular, se está investigando el uso de inteligencia artificial y del aprendizaje automático para aumentar de forma significativa la masificación del conocimiento de lengua de señas y disminuir la brecha comunicacional existente entre una persona que conozca lengua de señas y otra que no. Si bien ya existen herramientas que permiten la traducción automática de lengua de señas de forma eficaz, la mayoría de ellas no se han enfocado a la lengua de señas chilena.

El objetivo de este proyecto ha sido la implementación de un reconocedor automático del alfabeto de la lengua de señas chilena mediante la generación de modelo de aprendizaje automático que se ha incorporado a una aplicación móvil también desarrollada para este fin y que permite reconocer en tiempo real los gestos realizados por las personas a la cámara. Para el desarrollo del módulo de clasificación se utilizó transferencia de aprendizaje, recolectando nuestro propio conjunto de datos junto a otros obtenidos desde distintos repositorios. Logrando finalmente el desarrollo de una aplicación móvil que implementa el módulo de clasificación y logra reconocer las letras de lengua de señas chilena en tiempo real con una precisión que va desde el 80% hasta el 99% por letra.

Palabras clave: Inteligencia Artificial, Aprendizaje Automático, Transferencia de Aprendizaje, Clasificación de Imágenes, Lengua de Señas.

Índice general

Estructura y organización de la memoria

La presente memoria está organizada en cinco capítulos de la siguiente manera:

- En el **capítulo 1** se realiza una breve introducción a la problemática de las personas sordas en Chile, sus dificultades y a la importancia de desarrollar herramientas que permitan mejorar su integración en la sociedad. Se describen ejemplos de soluciones software recientes y se motiva la creación de una herramienta inteligente sobre dispositivo móvil con aprendizaje automático que permita interpretar en tiempo real las palabras de lengua de señas realizadas por una persona frente a la cámara.
- El **capítulo 2** corresponde al marco teórico de esta investigación. En este capítulo se explican brevemente conceptos claves para la lectura de esta investigación, introduciendo conceptos tales como inteligencia artificial, aprendizaje automático y aprendizaje profundo, estando estos directamente relacionados con el reconocimiento automático de señas.
- El **capítulo 3** corresponde al estado del arte, se analizarán trabajos similares y las principales diferencias con este trabajo. Además, se consideran en este capítulo algunas de las herramientas tecnológicas recurrentes para la solución de problemas similares, que puedan servir para el problema de la clasificación de letras del alfabeto de lengua chilena en tiempo real.
- El **capítulo 4** aborda lo que es el estudio y la solución a la problemática, en este capítulo se explica cómo se desarrolló un prototipo funcional con las letras del alfabeto de lengua de señas chileno. Considerando así los procesos desde la etapa de recolección de datos, entrenamiento y validación del modelo IA, y finalmente el desarrollo de la aplicación android que utiliza el modelo previamente creado. También contiene el código utilizado en el desarrollo de esta investigación e integración de tecnologías. Lo compone un primer código desarrollado en lenguaje Python, y el código Android para el desarrollo de la interfaz de la aplicación desarrollada en Java.

- El **capítulo 5** presenta las conclusiones de esta investigación. Se abordan distintas miradas desde un punto de vista de integración de tecnologías, también se consideran las limitaciones de esta investigación y propuestas para poder seguir desarrollando éste tema en un futuro.

Índice de figuras

Índice de Tablas

Capítulo 1.- Introducción.

Actualmente, según los datos de la Organización Mundial de la Salud (OMS) (*Sordera Y Pérdida De La Audición*, 2021), la comunidad sorda corresponde a más del 5% de la población mundial. Esto equivale a 466 millones de personas, de las cuales 432 millones son adultos y 34 millones son niños. Una persona sufre pérdida de audición cuando no es capaz de oír bien, es decir, aquellas personas cuyo umbral de audición es menor que 20 dB. La pérdida de audición se clasifica en leve, moderada, grave o profunda. En Chile, según el Segundo Estudio Nacional de la Discapacidad (*Servicio Nacional De La Discapacidad*, 2021), existen 712.005 personas con discapacidad que tienen algún grado de pérdida de audición, de ellas se estima que 179.268 personas tendrían sordera total. En el año 2021 a través de la promulgación de la ley N° 21.303 el Estado de Chile reconoce a la lengua de señas como la lengua oficial de las personas sordas, además, se establece que el Estado reconoce y se obliga a promover y respetar los derechos culturales y lingüísticos de las personas sordas, asegurándoles el acceso a los servicios públicos y privados, a la educación, y mercado laboral.

Uno de los problemas más importantes respecto a esta investigación, es que muchas personas no conocen la lengua de señas y esto puede ser un gran obstáculo a la hora de comunicarse con las personas sordomudas produciendo aislamiento social, soledad y problemas de estigma en estas personas. Por ejemplo, en el ámbito de la educación y empleo, en los países en desarrollo los niños con pérdida de audición pocas veces son escolarizados. En los adultos la tasa de desempleo es alta. Entre los que tienen un trabajo, no logran ocupar un buen puesto de trabajo. Desde un punto de vista social y económico la OMS calcula que los casos desatendidos de pérdida de audición representan un coste anual de 980.000 millones de dólares. Por estas razones es importante contar con iniciativas que traten de disminuir estos problemas. Algunos desarrolladores como David Lee, Abdullah, Electrocode, entre otros, han desarrollado modelos de clasificación de imágenes del

alfabeto de lengua de señas en tiempo real, pero ninguno de ellos está disponible para la clasificación en tiempo real de lengua chilena. En este trabajo se desarrollará un sistema de reconocimiento en tiempo real para el alfabeto de la lengua de señas chilena (LSCh) mediante técnicas de aprendizaje automático. Al tratar de solucionar este problema mediante una aplicación móvil, se podría establecer un medio de comunicación con una persona que no conoce de esta lengua, trayendo consigo beneficios que además de poder comunicarse, se abre un abanico de posibilidades para estas personas, atacando directamente los problemas antes mencionados, pudiendo generar nuevas fuentes de empleo, comunicación efectiva entre docente y alumno en la educación, y en general una inclusión en la sociedad para estas personas en la barrera del lenguaje.

Este trabajo es una continuación de la investigación llevada a cabo en la Universidad del Bio-Bio (Navarrete, 2022) cuyo título es: *Evaluación y comparación de modelos de aprendizaje automático supervisados y no supervisados para la clasificación de imágenes enfocado al alfabeto de lengua de señas chilena*. En este trabajo se compararon distintos modelos existentes para la clasificación de imágenes de lengua de señas de manos para el alfabeto chileno. Para ésta tarea, se comparan técnicas y algoritmos de aprendizaje supervisado como de aprendizaje no supervisado. En el aprendizaje no supervisado, se evaluó la Red de Hopfield, Autoencoder y K-means. En el aprendizaje supervisado, se evaluó el desempeño de Máquina de soporte de vectores, Redes neuronales convolucionales, y transferencia de aprendizaje.

	Porcentaje de aciertos
Hopfield	No cumple
K-means	No cumple
Encoder	82,9 %
Autoencoder	89,2 %
SVM	86,1 %
CNN	95,1 %
CNN con transferencia de aprendizaje	98,2 %

Tabla 1. Tabla con el porcentaje de aciertos de clasificación de imágenes (Navarrete, 2022)

Los resultados de esta investigación fueron que los algoritmos de aprendizaje supervisado son los que presentan mayor cantidad de aciertos en la tarea de clasificación de imágenes. Los algoritmos de aprendizaje no supervisado no logran superar el 90% de aciertos, mientras que los algoritmos de aprendizaje supervisado logran superar el 95% de aciertos. Otras investigaciones han demostrado resultados similares. En el artículo *Hands-On Guide To Sign Language Classification Using CNN* (Kumar, 2020), los resultados muestran que el puntaje de precisión del modelo con redes neuronales convolucionales es superior al 96%. *Deep Learning for Sign Language Recognition: Current Techniques, Benchmarks, and Open Issues.* (Al-Qurishi et al., 2021) indica en sus conclusiones que gran parte de los trabajos para clasificación de imágenes de lengua de señas utiliza redes neuronales convolucionales. *Sign Language Recognition: A Deep Survey* (Rastgoo & Escalera, 2021) concluye que a pesar de que muchos modelos han sido propuestos en los últimos años para el reconocimiento de imágenes de lengua de señas, la mayoría de estos modelos utilizan redes neuronales convolucionales .

Para ésta investigación, se ha utilizado la metodología de desarrollo evolutivo. Puesto que primero nos centraremos en el desarrollo de un prototipo inicial (modelo de prototipos) que pueda responder a las necesidades de este proyecto. Posteriormente se mejoró el modelo conforme concurrió el transcurso del proyecto para así estar al tanto de descubrimientos nuevos y cambios en el desarrollo del proyecto, es decir, mejoras en el algoritmo, resultados no esperados, y situaciones afines.

1.1 Objetivo general

El objetivo general de este proyecto ha sido desarrollar una aplicación móvil que permita reconocer en tiempo real el alfabeto de la lengua de señas chilena a partir de los gestos realizados a la cámara empleando herramientas de aprendizaje automático.

1.2 Objetivos específicos

- Para ello, hemos investigado los principales trabajos sobre localización y etiquetado automático de objetos en tiempo real, así como de modelos de aprendizaje automático empleados para este tipo de problema.

- Proponemos una arquitectura software que pueda trabajar con modelos de localización y etiquetado de objetos en tiempo real, así como con modelos de aprendizaje automático sobre dispositivos móviles.

- Recolectamos y etiquetamos un conjunto de datos para los entrenamientos de los modelos. Se ha programado un módulo de localización y etiquetado para el reconocimiento de imágenes de letras del lenguaje de señas chileno e incorporar el módulo IA creado previamente en una aplicación móvil básica con una interfaz sencilla.

- Finalmente hemos evaluado el desempeño del módulo IA creado.

Capítulo 2.- Marco Teórico.

El marco teórico de esta investigación está compuesto por conceptos netamente relacionados con la inteligencia artificial. Hablar de inteligencia artificial va más allá de combinar algoritmos. La inteligencia artificial (IA) como disciplina al menos está formada por tres áreas, ciencias de la cognición, robótica, y sistemas inteligentes de computación. Dentro de los sistemas inteligentes computacionales hay dos paradigmas: el conexionista y el simbólico. En los simbólicos que suelen ser *"top down"* el experto modela el sistema para que resuelva un problema como lo haría un experto. En el conexionista se crean modelos que resuelven problemas en base a los datos suelen ser un diseño *"bottom up"*. El sistema de este proyecto es conexionista ya que se entrena y luego se incorpora a un proceso de desarrollo clásico.

2.1 Inteligencia Artificial

La Inteligencia Artificial (IA) es un área multidisciplinar de la que se pueden encontrar distintas definiciones de diferentes autores importantes. Por ejemplo, John McCarthy, considerado como uno de los padres de la IA, en 1956 en la conferencia de Dartmouth dijo *"Es la ciencia e ingeniería para construir máquinas inteligentes, especialmente, programas de computación inteligentes. Así como, lo relativo a la tarea de usar computadoras para entender la inteligencia humana, pero no limitada a métodos observables biológicamente." Por otro lado,* Marvin Minsky en 1990 explica: *"Aún cuando todavía no conocemos cómo los cerebros realizan sus habilidades mentales, podemos trabajar hacia el objetivo de hacer máquinas que hagan lo mismo. La 'Inteligencia Artificial' es simplemente el nombre que dimos a esta investigación." Otro autor como Russell* en 2003 *define la IA como: "Un sistema inteligente es aquel cuya expectativa de utilidad es la más alta que se puede alcanzar por cualquier otro sistema con las mismas limitaciones computacionales". También* (Rouhiainen, 2018) dice: *''La IA es la capacidad de las máquinas para usar algoritmos, aprender de los datos y utilizar lo aprendido en la toma de decisiones tal y como lo haría un ser humano.". Por último,* (François Chollet, 2018) establece *que* ''*El esfuerzo de automatizar las tareas intelectuales que normalmente*

realizan los humanos.".

Actualmente la IA se puede definir como la combinación de áreas que proveen de capacidades a las máquinas para que tengan sentido común y una capacidad eficiente de aprender, resolver problemas y afrontar situaciones como lo haría un ser humano. Las ventajas de esta tecnología es que puede analizar grandes cantidades de información, sumado a que el porcentaje de error es significativamente menor en comparación a los humanos que realizan la misma tarea. La IA es una disciplina que trata de crear sistemas artificiales (electrónicos o biológicos) que permitan resolver problemas como lo hacen el hombre o los sistemas biológicos. La IA al menos está formada por tres áreas: Ciencias de la Cognición, Robótica y Inteligencia Computacional. Dentro de la inteligencia computacional hay dos paradigmas: el conexionista y el simbólico. En los simbólicos suelen ser del tipo top down donde el experto modela e implementa el sistema para que resuelva un problema como lo haría un experto. El resultado es lo que se han llamado Sistemas expertos. En el conexionista se crean modelos que resuelven problemas en base a los datos suelen ser un diseño bottom up. El sistema de este proyecto es conexionista ya que se entrena y luego se incorpora a un proceso de desarrollo clásico. Son sistemas de aprendizaje automático y es el paradigma del sistema que presentamos en esta propuesta.

2.2 Aprendizaje automático

El aprendizaje automático se considera como una subrama de la IA. En esta subrama, se utilizan algoritmos para encontrar patrones comunes en los datos. Es decir, a través del aprendizaje automático un sistema o programa tendrá la capacidad de aprender o mejorar el rendimiento a partir de un conjunto de datos (en su mayoría grandes y complejos). Estos algoritmos identificarán los patrones encontrando una estructura estadística que generará reglas en el sistema para así poder generar predicciones o bien tomar decisiones de forma inteligente, sin haber estado programado previamente para tomar dicha decisión, sino que surge netamente como resultado del aprendizaje automático.

Tom Mitchell (Tom Mitchell, 1997) define el aprendizaje automático como: *"El estudio de algoritmos de computación que mejoran automáticamente su rendimiento gracias a la experiencia. Se dice que un programa informático aprende sobre un conjunto de tareas, gracias a la experiencia y usando una medida de rendimiento, si su desempeño en estas*

tareas mejora con la experiencia."

Por otro lado, Russell (Stuart Russell, 1996) lo define: *"El subcampo de la inteligencia artificial (IA) relacionado con los sistemas inteligentes que pueden aprender."*

Para realizar aprendizaje automático se necesitan:

- **Datos de entrada:** Los distintos algoritmos de aprendizaje automático necesitan de datos de entrada (etiquetados en el caso de ser supervisados) para poder aprender de ellos.
- **Ejemplos de resultados esperados:** El cómo debería responder el modelo frente a una determinada entrada.
- **Una forma de medir si el algoritmo está haciendo un buen trabajo:** A través de métricas establecidas para poder validar que tan bueno o malo es un determinado modelo en la tarea de predicción.

Un modelo de aprendizaje automático transforma los datos de entrada en salidas significativas, en este proceso se aprende de los distintos ejemplos de datos de entrada y su salida con su resultado esperado. (François Chollet, 2018). Existen dos clases de aprendizaje automático:

- **Aprendizaje supervisado:** en estos algoritmos se necesitan datos que hayan sido etiquetados u organizados para poder indicar cómo tendría que ser categorizada la nueva información. Incluyen algoritmos tales como regresión lineal y logística, clasificación multiclase y máquinas de vectores de soporte.

- **Aprendizaje no supervisado:** los datos en este algoritmo no están etiquetados u organizados de forma previa para indicar cómo tendría que ser categorizada la nueva información, sino que el mismo algoritmo tiene que ver la manera de cómo clasificarla. Estos algoritmos buscan agrupar los datos, asignando a cada grupo una etiqueta nueva.

2.3 Redes neuronales artificiales

Una red neuronal artificial es un conjunto de neuronas artificiales que conectadas entre sí forman una red. Las redes neuronales artificiales se inspiran y basan en las redes neurológicas biológicas de los humanos. En estas redes, cada neurona se conecta con otra neurona o más de una, y se conectan a través de enlaces. Cada neurona transmite una señal con la neurona que está conectada. Una red neuronal está compuesta por una o más capas de neuronas.

2.3.1 Capas

Las capas son la estructura de datos fundamental en las redes neuronales. Definiremos una capa como un módulo de procesamiento de datos que toma como entrada uno o más tensores y que genera uno o más tensores. Dependiendo del tipo de tensores, es que se utiliza un tipo de capa u otra.

Los datos de imagen, generalmente utilizan capas de convolución (Conv2D). Dentro de una capa las neuronas forman grupos neuronales. El conjunto de una o más capas dará por resultado una red neuronal (Ver figura 1).

- **Capa de entrada:** Reciben los datos de entrada desde el entorno.
- **Capa de salida**: Capa que otorga la respuesta de la red neuronal.
- **Capa oculta:** capa intermedia que no tiene relación con el entorno, estas capas ocultas pueden ser variadas en número en una red neuronal.

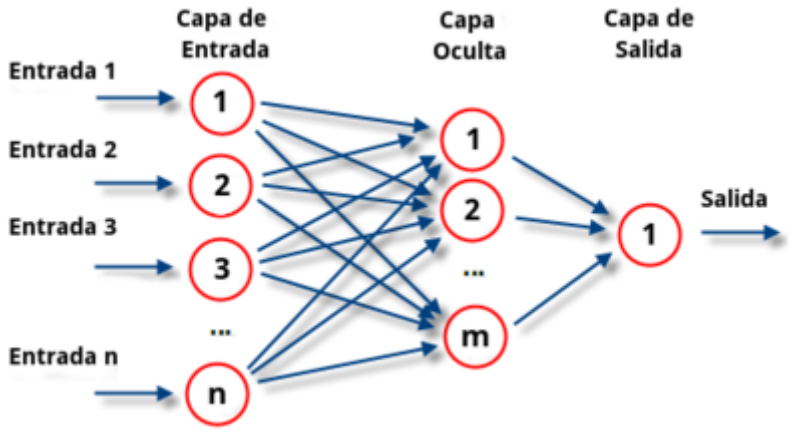

Figura 1. Red neuronal de 3 capas con n entradas, m neuronas en la capa oculta y una neurona en la salida.

2.3.2 Tensores

La mayor parte de los sistemas de aprendizaje automático utilizan tensores como estructura de los datos principales. Definiremos un tensor como un contenedor de datos para el aprendizaje automático (Chollet, 2017). En base a lo anterior, dependiendo del tipo de problema que se quiere resolver se utilizará uno u otro tipo de tensor. Algunos de ellos son los siguientes:

- Tensores 0D: Tensor que contiene un sólo número, también denominado tensor escalar o tensor de dimensión 0.
- Tensores 1D: Llamados tensores vectoriales, son de un eje.
- Tensored 2D: Llamados tensores matrices, son de dos ejes, tiene entradas tanto en su primer eje (filas) como en su segundo eje (columnas).
- Tensores 3D: Este tipo de tensor es de una dimensión superior, se puede interpretar como un cubo de números. Tienen tres ejes y por tanto tendrán tres entradas de datos, en sus tres ejes.

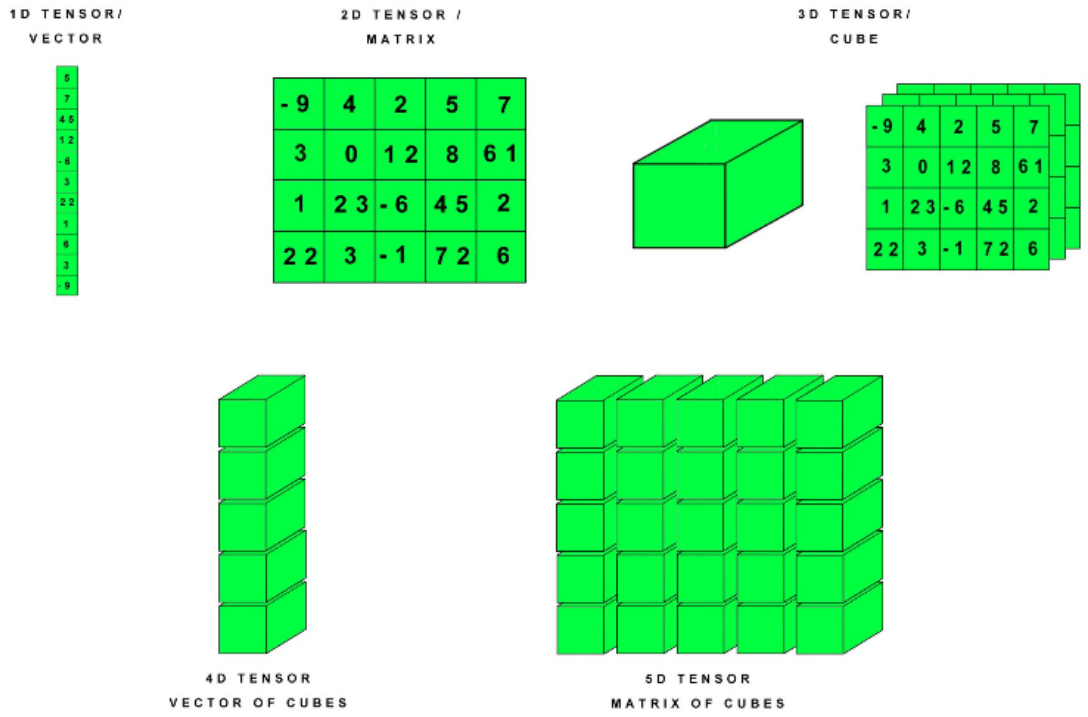

Figura 2. Representación gráfica de tensores de 1 a 5 dimensiones (Recuero, 2019).

En general, podemos encontrarnos con tensores de 1 a 5 dimensiones (Figura 2), esto dependiendo del tipo de problema y el tipo de dato de entrada que se desea procesar. Para la problemática de procesar datos de imagen, estas suelen tener 3 dimensiones (largo, ancho y profundidad). Al tener 3 dimensiones podemos almacenar un lote de imágenes en un tensor 4D. Para los datos de video se necesitarán tensores 5D. Los videos están compuestos por fotogramas, cada fotograma se definirá como una imagen en color. Es por esto que presentan una capa más, entonces en base a esto podemos almacenar un lote de videos en un tensor 5D.

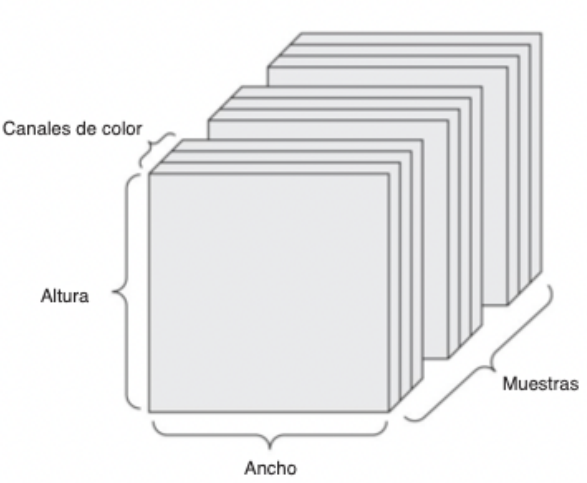

Figura 3. Tensor de datos de imágen 4D (Deep learning with python, 2017)

2.3.3 Pesos

Los pesos corresponden a coeficientes que pueden adaptarse dentro de la red que determinan la intensidad o importancia de la señal de entrada registrada por la neurona artificial (*Advanced Tech Computing Group UTPL*, 2007). A veces, estos se denominan los parámetros de una capa. La especificación de lo que hace una capa con sus datos de entrada se almacena en los pesos de la capa. Básicamente, en una red neuronal los pesos se tienen que ir ajustando hasta reducir los valores de la función de pérdida. Este valor de función de pérdida se utiliza a modo de retroalimentación para así ajustar el valor de los pesos y optimizar el modelo.

2.3.4 Funciones de pérdida y optimizadores

Las funciones de pérdida sirven para evaluar qué tan bien los pesos consiguen ajustarse para resolver el problema que se quiere automatizar. Mientras más pequeño es el valor que adquiera la función de pérdida el modelo es mejor. En cambio, cuando el valor arrojado por la función de pérdida es muy grande, significa que las predicciones del modelo se alejan demasiado de los resultados reales. La función de optimización tendrá por objetivo aprender a reducir el error en la predicción, mejorando así cada vez más el modelo, la red se actualizará a sí misma en función de los datos que ve y su función de pérdida.

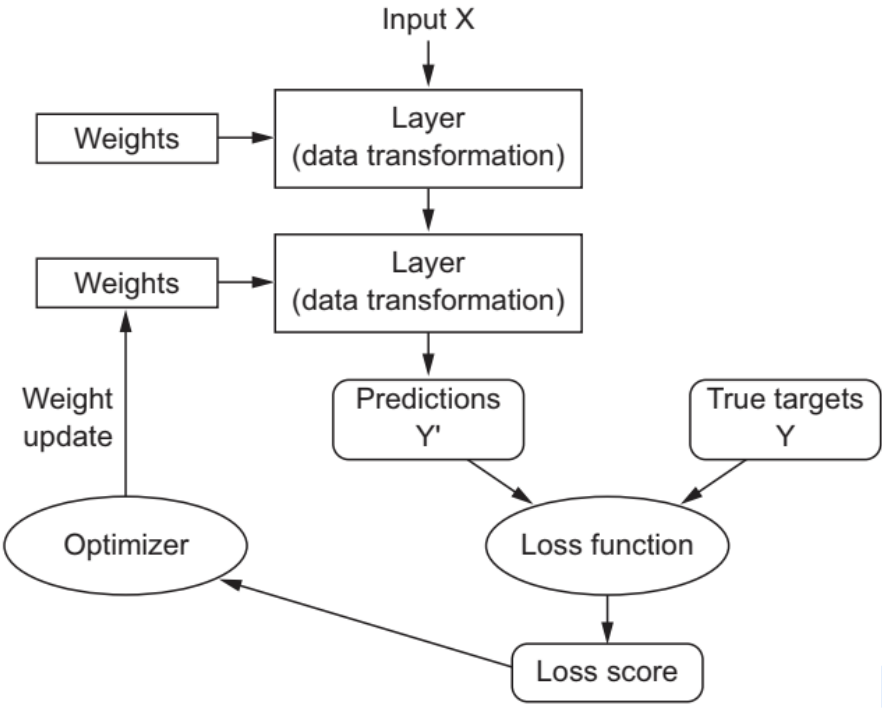

Figura 4. Relación entre la red, las capas, la función de pérdida y el optimizador (François Chollet, 2018).

2.4. Aprendizaje profundo

El aprendizaje profundo (Deep learning) es un subconjunto del aprendizaje automático. Los algoritmos de aprendizaje profundo permiten un aprendizaje progresivo mejorado. Tienen la particularidad de que mejoran la capacidad de clasificación, reconocimiento y detección de objetos. En el aprendizaje profundo, surge la idea de tener modelos con una multitud de capas cada vez más numerosas. Estas representaciones de muchas capas en el aprendizaje profundo se realizan a través de las redes neuronales. Algunos tipos de estructuras de redes neuronales que se consideran para realizar aprendizaje profundo son:

- Redes neuronales convolucionales (CNN)
- Perceptrones multicapa (MLP)
- Redes de función de base radial (RBFN)

- Redes neuronales recurrentes (RNN)
- Redes de memoria a largo plazo (LSTM)
- Autoenconder.

2.5. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales corresponden a uno de los tipos de redes más utilizadas para el problema de clasificación de imágenes. El problema que tiene este tipo de reconocimiento es que en la actualidad las resoluciones utilizadas en los dispositivos electrónicos, son muy altas, y tratar estas imágenes en crudo hace que la tarea de entrenamiento y reconocimiento, sea muy exigente para el sistema. Por lo que el objetivo de esta red es reducir ésta carga, mediante el uso de capas que permiten quedarse con las características y patrones de las imágenes, permitiendo a la red neuronal identificar curvas, líneas e incluso formas más elaboradas, como lo pueden ser rostros, siluetas y expresiones, que es una tarea mucho menos demandante para el sistema y que permite clasificar una imagen de forma rápida sin perder precisión.

Las CNN tienen una capa de entrada, una capa de salida (capas completamente conectadas), y numerosas capas ocultas (capas convolucionales y capas de agrupación), que finalmente llevan al modelo a generar una clasificación.

2.5.1 Capa de convolución

La capa convolucional contiene una serie de operaciones que permite reducir la carga computacional del sistema, ésta serie de operaciones son conocidas como convoluciones, que consisten en aplicar un filtro a una imagen de entrada con el fin de generar un mapa de características. Ésta capa consigue que cada neurona esté conectada a un subconjunto de elementos de la imagen, permitiéndole detectar el mismo rasgo en cualquier parte de la imagen. Logrando reducir el número de conexiones y parámetros al momento de entrenar, entre las neuronas de la capa oculta y los elementos de la imagen de entrada.

La entrada de una capa convolucional es una imagen de m x m x r (m: altura y el ancho de la imagen, r: número de canales) y un filtro de dimensiones n x n x q (n: tamaño del filtro, q:

profundidad), obteniendo una matriz de características de tamaño (m - n + 1) x (m - n +1) x p (p: cantidad de filtros)

El proceso de convolución consiste en recorrer la imagen de entrada con un filtro, cada vez que el filtro es aplicado a un segmento de la imagen se calcula la convolución, siendo el resultado almacenado en la matriz de activación. El recorrido del filtro se realiza de izquierda a derecha, y se baja una unidad cuando llega al borde de la imagen. Cuando el proceso es completado, la matriz de activación contiene las características que se buscan de la imagen, a partir del filtro aplicado, proceso que se puede ver gráficamente en la figura 5 (Durán & Torres, 2017).

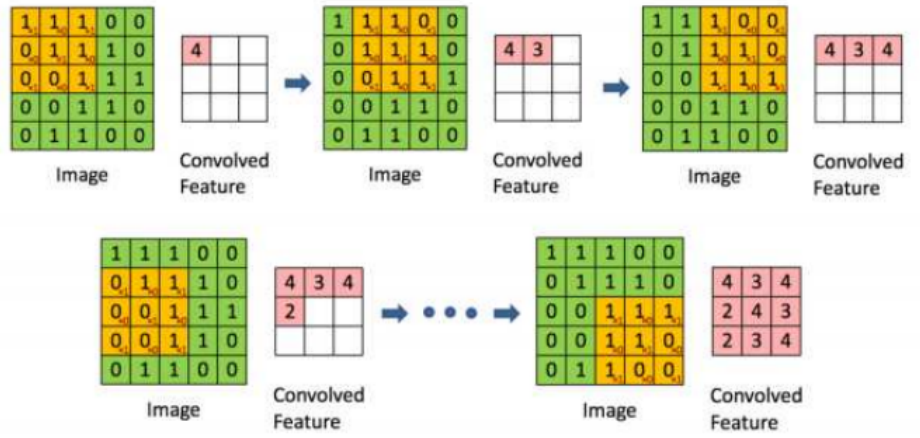

Figura 5. Obtención de matriz de activación (Durán & Torres, 2017).

2.5.2 Capa de agrupación

También conocida como capa de pooling, tiene como objetivo disminuir aún más el uso de recursos de un sistema, y también ayuda a caracterizar y localizar rasgos de una imagen de forma certera. Existen dos tipos de pooling: average-pooling o mean-pooling y max-pooling.

El proceso de la capa de agrupación es el siguiente: se recorre la matriz de activación de izquierda a derecha, y se selecciona una submatriz de tamaño p x p, y dependiendo del tipo de pooling, los valores de la matriz de salida cambian. En el caso del tipo average-pooling o mean-pooling, de los elementos contenidos en la submatriz, se calcula la media y el resultado es almacenado en la matriz de salida. En cambio el tipo de pooling max-pooling, de los

elementos contenidos en la submatriz, selecciona el valor mayor, y lo guarda en la matriz de salida (Figura 6).

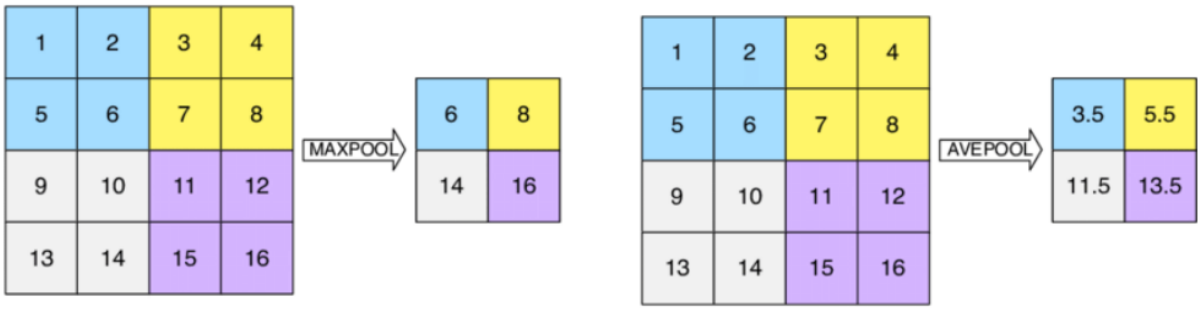

Figura 6. Operación de max-pooling y de average-pooling (Durán & Torres, 2017).

2.5.3 Capa completamente conectada

La capa completamente conectada o capa full-connected corresponde a la última capa de una red neuronal convolucional, y es la encargada de determinar a qué clase corresponde la imagen de entrada, mediante la agrupación de información que se ha obtenido hasta el momento. Se caracteriza por su arquitectura totalmente conectada, debido a que todas sus neuronas están conectadas a todas las neuronas de la capa anterior, por lo que tiene un gran número de conexiones, a diferencia de las capas convolucionales que se encuentran conectadas a una sola región local de la entrada y que muchas de las neuronas de dicha capa comparten parámetros (Bonilla Carrión & Pino Mejías, 2020).

En las redes convolucionales, se suelen ocupar múltiples capas completamente conectadas una tras otra, siendo la última de estas compuesta por una cantidad de neuronas igual a la cantidad de clases que contiene el conjunto de datos, y éstas tendrán como salida la probabilidad que tiene de ser determinada clase.

2.5.4 Transferencia de aprendizaje

La tarea de entrenamiento se hace mucho más fácil cuando una red neuronal no se entrena desde cero, es por ello que en la mayoría de redes neuronales se utiliza una arquitectura base, con parámetros de un modelo pre-entrenado con millones de imágenes y miles de etiquetas distintas, otorgándole a la nueva red neuronal la capacidad de reconocer

muchas cosas desde el comienzo, traduciéndose en un tiempo de entrenamiento menor. Para obtener una nueva red neuronal que sea capaz de reconocer imágenes en específico y transferir el aprendizaje del modelo pre-entrenado, es necesario un conjunto de datos y etiquetas que apunten a la tarea de reconocimiento en específico. Con esta alternativa de entrenamiento se pueden obtener mejores resultados y que se alcancen mucho antes.

La transferencia de aprendizaje consiste en quitar la capa de salida del modelo pre-entrenado y añadir capas adicionales con el nuevo conjunto de datos, efectuándose retoques en los parámetros que permitirán adaptar el modelo a nuestros requerimientos.

2.6. Reconocimiento automático

Las redes convolucionales son un tipo de red neuronal que fueron diseñadas especialmente para el reconocimiento y clasificación de imágenes, ya que estas neuronas buscan trabajar tal y como lo hace la corteza visual primaria. Estas redes tienen una estructura de 3 capas: Convolutional Layer, Pooling Layer y Fulling-Connected Layer, que en conjunto conectan una serie de capas para obtener la salida deseada, la cual corresponde al número de clases (Figura 7).

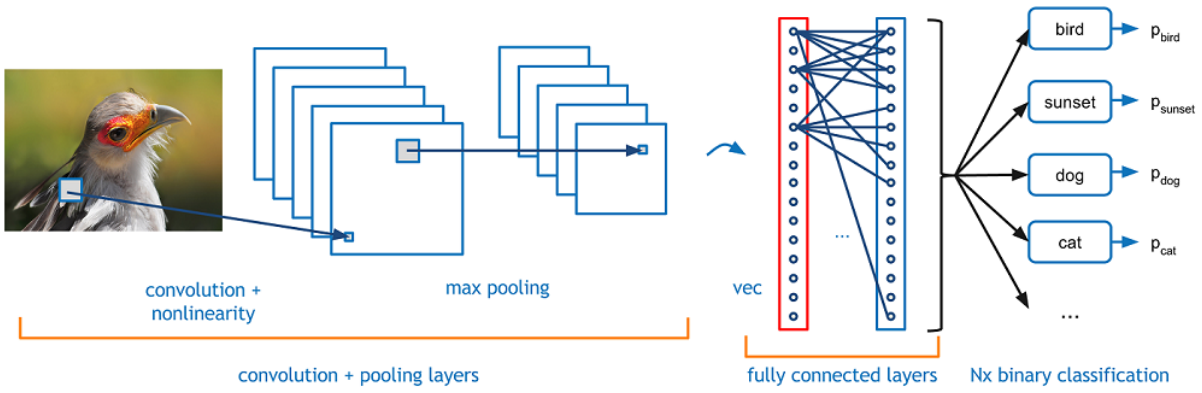

Figura 7. Estructura de una red neuronal convolucional para la clasificación de una imagen.

2.7 Utilización de redes neuronales

Resolver el problema de clasificación de imágenes del alfabeto de lengua de señas chilena, requerirá hacer uso de redes neuronales. Para entender a simple modo cómo funciona una red neuronal, a continuación mencionaremos algunos ejemplos que puedan servir de ayuda:

2.7.1 Red neuronal simple (Una sola capa)

En el siguiente ejemplo, necesitamos saber cómo poder calcular la equivalencia de una temperatura, pasar de grados celsius a fahrenheit. Para ello, sólo le proporcionamos como datos de entrada a nuestra red neuronal los valores de los respectivos grados celsius y su equivalencia en grados fahrenheit. El parámetro "units" hace referencia a que nuestra red tendrá sólo una neurona por capa y el parámetro "input_shape" a que sólo tendrá una entrada como neurona.

Gráficamente la red neuronal se vería de la siguiente manera (Figura 8):

Figura 8. Visualización gráfica red neuronal simple (Fuente propia).

Luego de entrenar la red neuronal por alrededor de 800 épocas, nos damos cuenta que en la época 600 aproximadamente, la red neuronal ya no evoluciona en su aprendizaje, por tanto basta con 600 épocas aproximadamente para tener resultados óptimos en la predicción (Figura 9).

```
import tensorflow as tf
import numpy as np

celsius = np.array([-40, -10, 0, 8, 15, 22, 38], dtype=float)
fahrenheit = np.array([-40, 14, 32, 46, 59, 72, 100], dtype=float)

capa = tf.keras.layers.Dense(units=1, input_shape=[1])
modelo = tf.keras.Sequential([capa])

modelo.compile(
    optimizer=tf.keras.optimizers.Adam(0.1),
    loss='mean_squared_error'
)

historial = modelo.fit(celsius, fahrenheit, epochs=800, verbose=False)

import matplotlib.pyplot as plt
plt.xlabel("# Epoca")
plt.ylabel("Magnitud de pérdida")
plt.plot(historial.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7fb368b79d50>]
```

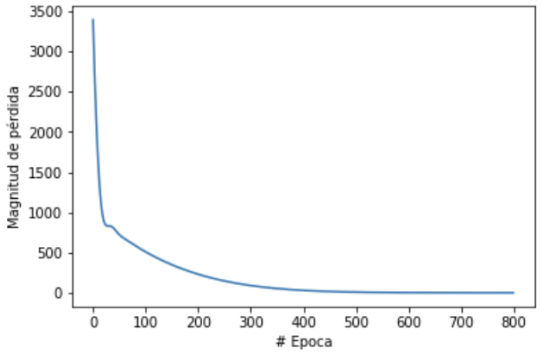

Figura 9. Codificación red neuronal simple y gráfica de magnitud de pérdida (Fuente propia).

2.7.2 Red neuronal con 2 capas ocultas

Al crear una red neuronal con 2 capas ocultas de tres neuronas cada capa, y proveer sólo una entrada y una salida a la red neuronal (Figura 10), tendríamos algo así:

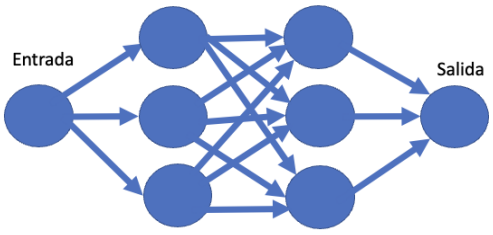

Figura 10. Gráfica red neuronal con dos capas ocultas y 3 neuronas (Fuente propia).

Como podemos apreciar, luego de 65 o 75 épocas de entrenamiento, el resultado de la red neuronal deja de mejorar, pero concluimos que en comparación al caso anterior, al agregar dos capas ocultas con más neuronas, la precisión de resultados correctos aumenta con menos épocas de entrenamiento (Figura 11).

```
import tensorflow as tf
import numpy as np

celsius = np.array([-40, -10, 0, 8, 15, 22, 38], dtype=float)
fahrenheit = np.array([-40, 14, 32, 46, 59, 72, 100], dtype=float)

oculta1 = tf.keras.layers.Dense(units=3, input_shape=[1])
oculta2 = tf.keras.layers.Dense(units=3)
salida = tf.keras.layers.Dense(units=1)
modelo = tf.keras.Sequential([oculta1, oculta2, salida])

modelo.compile(
    optimizer=tf.keras.optimizers.Adam(0.1),
    loss='mean_squared_error'
)

historial = modelo.fit(celsius, fahrenheit, epochs=200, verbose=False)

import matplotlib.pyplot as plt
plt.xlabel("# Epoca")
plt.ylabel("Magnitud de pérdida")
plt.plot(historial.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7fb36792c590>]
```

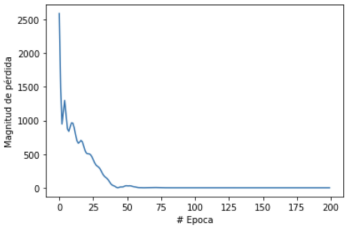

Figura 11. Codificación red neuronal con dos capas ocultas de 3 neuronas y gráfica de magnitud de pérdida (Fuente propia).

2.7.3 Red neuronal con 2 capas ocultas, 784 entradas, capas ocultas con 50 neuronas y 10 salidas posibles.

Cuando en el proceso de predecir una clase existen más de una salida posible, todo el proceso se torna más complejo, en el caso de una imagen que necesitamos saber a qué tipo de clase corresponde, lo primero que se realiza es ajustar el tamaño de la imagen a una imagen más pequeña para que esta pueda ser procesada. En el siguiente ejemplo, se reduce el tamaño de la imagen a un tamaño de 28 x 28 píxeles, teniendo 784 píxeles en total. Cada uno de estos píxeles serán una entrada, y cada una de estas entradas tendrán que ser procesadas por una neurona. Por tanto, en la capa de entrada tendríamos 784 neuronas y entradas. Cada uno de estos píxeles tomará un valor numérico, es decir, desde 0 hasta 255,

siendo el número 0 un píxel de color totalmente negro y el 255 para totalmente blanco (Figura 12).

Figura 12. Ejemplo de imagen a procesar en red neuronal y funcionamiento con píxeles (Ringa Tech, 2021).

Para realizar una buena predicción, es necesaria una red más compleja, para este caso a modo de ejemplo, lo que se realiza son dos capas ocultas (densas), cada una con 50 neuronas, estas están completamente conectadas entre sí en forma secuencial y finalmente se producen 10 salidas posibles, debido a que existen 10 clases posibles de clasificación.

La red neuronal se vería se la siguiente manera (Figura 13):

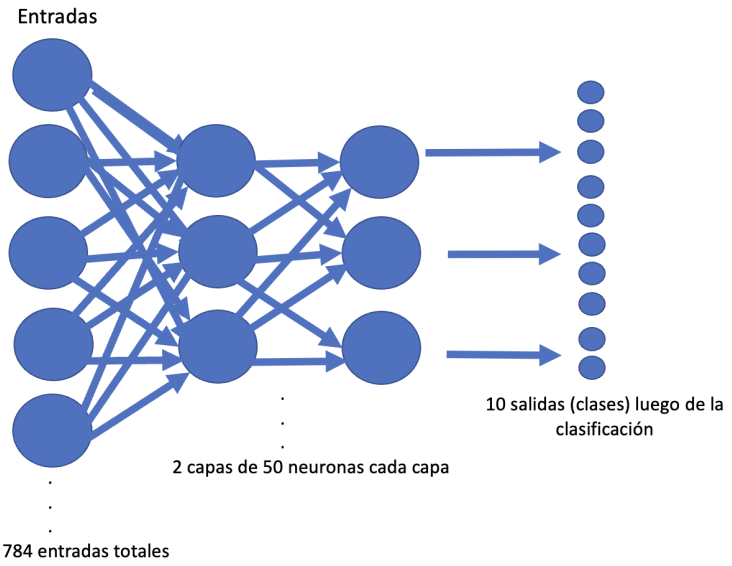

Figura 13. Ejemplo de red neuronal con 784 entradas (píxeles), 2 capas ocultas de 50.

neuronales y 10 posibles salidas (Fuente propia).

Luego de 300 épocas de entrenamiento, la red neuronal se comporta de la siguiente manera, en base a estos resultados, es muy probable que con más capas ocultas y más neuronas se podrían obtener muchos mejores resultados (Figura 14).

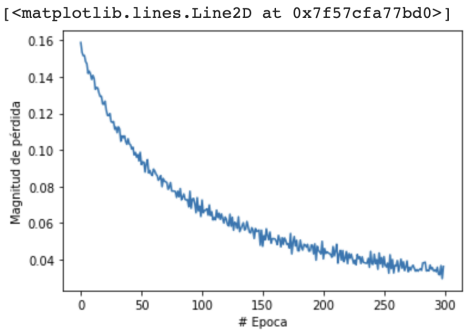

Figura 14. Comportamiento gráfico de función de pérdida luego de 300 épocas de entrenamiento (Fuente propia).

2.8. Dataset

Para llevar a cabo el reconocimiento y clasificación de imágenes, es necesario tener la información que se quiere analizar, caracterizándose por tener una cantidad de muestras que permitan estudiar los patrones de los datos. Entre los tipos de aprendizaje automático, el supervisado es el mejor en las tareas de reconocimiento, sin embargo en su contra parte, el etiquetado de cada uno de sus datos supone un alto costo de tiempo y producción.

Hay varios conjuntos de datos gratuitos subidos a internet que pueden ser aprovechados para aplicarlos a las redes convolucionales, entre los más populares MNIST, CIFAR-10, ImageNet, pudiendo servir de complemento para el entrenamiento de los modelos.

2.9. Data Augmentation

Data Augmentation es una técnica que permite aumentar artificialmente la cantidad de datos, realizando modificaciones ya existentes en el Dataset, estas alteraciones pueden ser la aplicación de desenfoque en la imagen, cambio de escala de grises, modificación de

tamaño, rotación de la imagen, cambio de color, brillo o contraste (Figura 15). Es una de las técnicas más populares para crear una IA robusta sin la necesidad de gastar muchos recursos.

Figura 15. Cambio de color como técnica de *Augmentation*. (Shorten, C., Khoshgoftaar, 2015*)*.

2.10 Detección de objetos

Con el pasar de los años la visión computacional ha avanzado a pasos agigantados, a tal punto que permite identificar objetos, lugares y personas con un grado de precisión muy elevado. Métodos que hacen uso de Deep Learning destacan en el área de la detección de objetos, ya que no es necesario codificar las características de los objetos, sino más bien entrenar un modelo de redes neuronales en base a un set de datos.

El lenguaje de programación Python es uno de los lenguajes más utilizados para la creación de aplicaciones de aprendizaje automática ya que disponen de una gran cantidad de librerías y frameworks, que facilitan el proceso de desarrollo, entre ellos destacan TensorFlow, Yolo, Scitik-learn.

2.11 TensorFlow Lite

Como bien lo define su página oficial, *TensorFlow Lite es un conjunto de herramientas que ayuda a los desarrolladores a ejecutar sus modelos en dispositivos incorporados, móviles o de IoT, y les permite implementar el aprendizaje automático integrado en el dispositivo.* Esto permite reducir el tamaño del modelo, acomodándose a las limitaciones existentes de memoria y recursos que hay en los dispositivos IoT. La aparición de TensorFlow Lite permite optimizar el aprendizaje automático, destacándose en los siguientes aspectos:

- Latencia: se evita el intercambio de información de ida y vuelta a servidores para llevar a cabo las inferencias.
- Privacidad: al realizarse la detección en el mismo dispositivo, no es necesario que salga información del dispositivo.
- Consumo de energía: reduce en gran medida el consumo de energía el hecho de realizar la inferencia en el dispositivo sin la necesidad de conectarse a internet.
- *Optimizado para el aprendizaje automático integrado en el dispositivo*, ya que aborda 5 limitaciones clave: latencia (no hay ida y vuelta con un servidor), privacidad (ningún dato personal sale del dispositivo), conectividad (no es necesaria una conexión a Internet), tamaño (tamaño reducido del modelo y de los objetos binarios) y consumo de energía (inferencia de alta eficiencia sin necesidad de conexiones de red)
- *Compatibilidad con múltiples plataformas*, lo que incluye dispositivos iOS y Android, Linux incorporado y microcontroladores.
- *Compatibilidad con diversos lenguajes*, entre los que se incluyen Java, Swift, Objective-C, C++ y Python
- *Alto rendimiento*, con aceleración de hardware y optimización de modelos
- Ejemplos d*e extremo a extremo* de tareas comunes de aprendizaje automático, como clasificación de imágenes, detección de objetos, estimación de poses, respuestas a preguntas, clasificación de texto, etc. en múltiples plataformas

Capítulo 3.- Estado del arte

La aplicación de técnicas de aprendizaje automático para problemas de reconocimiento de lengua de señas es muy habitual en la actualidad. Un punto en común es que la mayoría de las investigaciones que resuelven el problema de clasificación de imágenes de alfabetos de lengua de señas, utilizan redes neuronales convolucionales, pues han sido estas las que han dado los mejores resultados en las investigaciones. Autores como Lee, Jorviader y Abdullah entre los años 2019 y 2022, utilizando redes neuronales convolucionales han desarrollado trabajos de clasificación de imágenes para las letras de lengua de señas. El primer problema que se presenta en muchos de estos trabajos es que hay poco desarrollo de clasificadores de imágenes en tiempo real. El segundo problema, es que no se encontró un clasificador en tiempo real y específicamente alguna aplicación móvil que pueda realizar esta tarea. Otro inconveniente, siendo la principal propuesta de valor de esta investigación, es que no se encontraron conjuntos de datos de lengua de señas chilena. Existen muchos conjuntos de datos del alfabeto americano, y uno que otro de otros países, como el árabe, pero no se encontró ningún conjunto de datos del alfabeto chileno. A continuación se explican de forma resumida algunos de los artículos y resultados de investigación para el desarrollo de esta problemática.

3.1 YOLO

Según el artículo *You Only Look Once: Unified, Real-Time Object Detection (Redmon, 2016),* YOLO es un nuevo enfoque para la detección de objetos, básicamente como lo dice su nombre, con tan sólo mirar una vez la imagen ya podemos clasificar y realizar una clasificación de una clase (Figura 16).

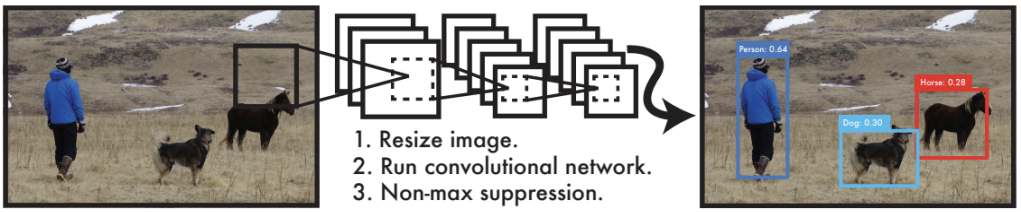

Figura 16. Funcionamiento del sistema de detección de YOLO (Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. 2016).

Antes de YOLO, los sistemas de clasificación utilizaban la transferencia de aprendizaje y reutilizaban el "conocimiento" de la red para poder realizar una tarea de clasificación. Una de las limitaciones de esto es muchas veces la complejidad de la red y lo pesado que finalmente termina siendo todo este proceso. Con YOLO, una sola red neuronal convolucional predice simultáneamente múltiples cuadros delimitadores y probabilidades de clase para estos cuadros. YOLO entrena con imágenes completas y optimiza directamente el rendimiento de detección.

Dentro de sus beneficios encontramos:

- Es muy rápido: El procesamiento de video para clasificación de objetos en tiempo real presenta menos de 25 milisegundos de latencia.
- Mejor precisión: logra mejor precisión en tiempo real que otros modelos de clasificación.
- Comete menos errores: YOLO ve la imagen completa durante el entrenamiento y el tiempo de prueba, por lo que codifica implícitamente información contextual sobre las clases, así como su apariencia.

La red de YOLO como mencionamos anteriormente utiliza características de toda la imagen para poder predecir cada cuadro delimitador. Además, predice todos los cuadros delimitadores en todas las clases para una imagen simultáneamente. Su diseño, permite un entrenamiento de extremo a extremo y que las velocidades en tiempo real sean altas manteniendo una precisión promedio alta.

YOLO divide la imagen en una cuadrícula de tamaño S x S (Figura 17), si el centro de un objeto cae en una celda de la cuadrícula, esa celda de la cuadrícula es la responsable de detectar ese objeto. En este proceso, cada celda de la cuadrícula predice cuadros delimitadores y puntuaciones de confianza para los cuadros. Cada cuadro delimitador consta de 5 predicciones, "x, y, w, h y confianza". X e Y corresponden a las coordenadas del centro de la imagen, y W y H el ancho y la altura de los límites de la celda, la confianza es qué tan probable es que el objeto encapsulado pueda ser o no una clase.

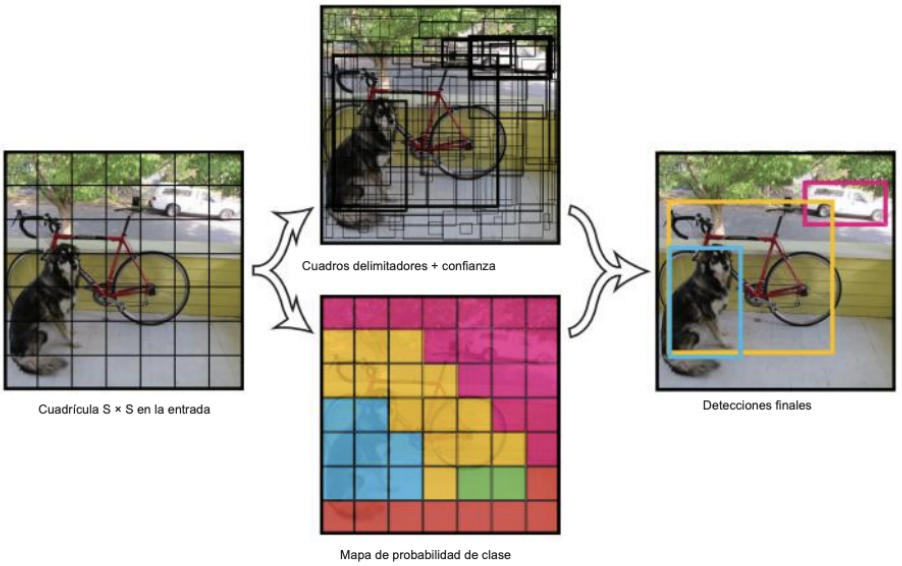

Figura 17. Esquema del algoritmo básico de YOLO (Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. 2016).

YOLO utiliza la métrica Intersection over Union (IoU). Esta métrica mide la precisión de un detector de objetos en un conjunto de datos en particular. Se utiliza en los cuadros delimitadores para evitar que en una imagen un mismo objeto pueda ser delimitado por múltiples cuadros. Es básicamente una fórmula, que divide el área de superposición por el área de unión de la imagen. El cuadro delimitador que mejor represente a la imagen original delimitada, será el cuadro que se verá reflejado. Generalmente, una puntuación de intersección sobre unión mayor a 0.5 se considera una buena predicción.

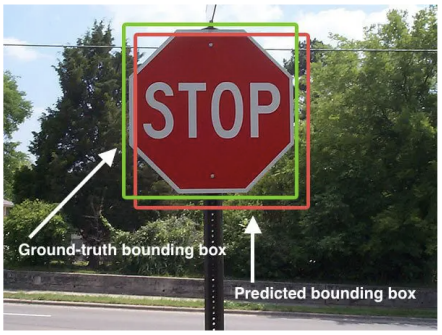

Figura 18. Ejemplo de cuadro delimitador programado por usuario (verde) vs cuadro detectado por la inteligencia (rojo) (Rosebrock, 2016).

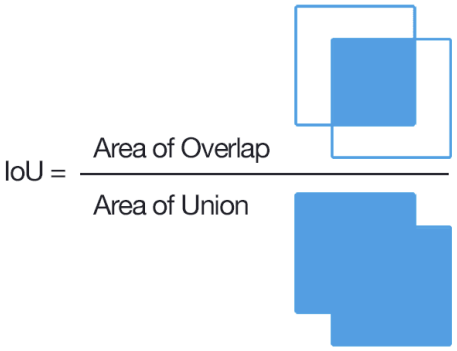

Figura 19. Fórmula de obtención IoU (Rosebrock, 2016).

Figura 20. Ejemplo de clasificación de valores IoU (Pobre, bueno, excelente) (Rosebrock, 2016).

YOLO en sus primeras versiones está implementado con una red convolucional, las capas convolucionales iniciales de la red extraen características de la imagen, mientras que las capas completamente conectadas predicen las probabilidades y coordenadas de salida. La arquitectura base está inspirada en el modelo de GoogleNET para la clasificación de imágenes. La red original posee 24 capas convolucionales seguidas de 2 capas completamente conectadas, pero, también existe una versión más rápida de YOLO diseñada para mejorar los tiempos de clasificación que utiliza 9 capas convolucionales en lugar de 24.

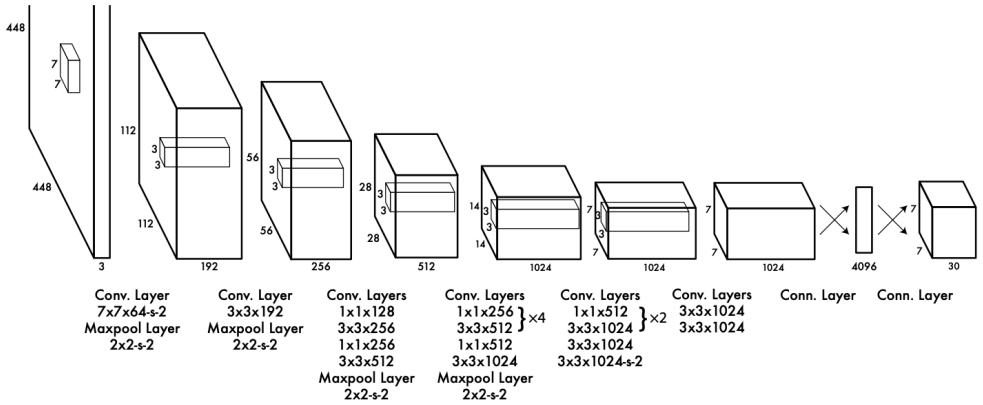

Figura 21. Arquitectura YOLO (Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. 2016).

3.2 Using Computer Vision in Helping the Deaf and Hard of Hearing Communities with Yolo V5 (Lee, 2020)

En el artículo *Using Computer Vision in Helping the Deaf and Hard of Hearing Communities with Yolo V5* (Lee, 2020), se crea un modelo a través del uso de redes convolucionales con transferencia de aprendizaje para la clasificación de imágenes en tiempo real. En este proyecto se creó un conjunto de datos desde cero de aproximadamente 1800 imágenes que posteriormente son etiquetadas.

El modelo de Lee funciona bastante bien, pero presenta problemas a la hora de predecir algunas letras, estas son:

- Letra D predicha como F.
- Letra E predicha como T.
- Letra P predicha como Q.
- Letra R predicha como U.

Este modelo presenta una precisión promedio de 95% (Figura 22), un resultado bastante alto para esta investigación considerando el pequeño conjunto de datos disponible para la investigación. Dentro de las limitaciones además de las letras anteriormente mencionadas, se encuentra la distancia en la cual se realiza la predicción, a mayor distancia menor y/o nula la precisión del modelo.

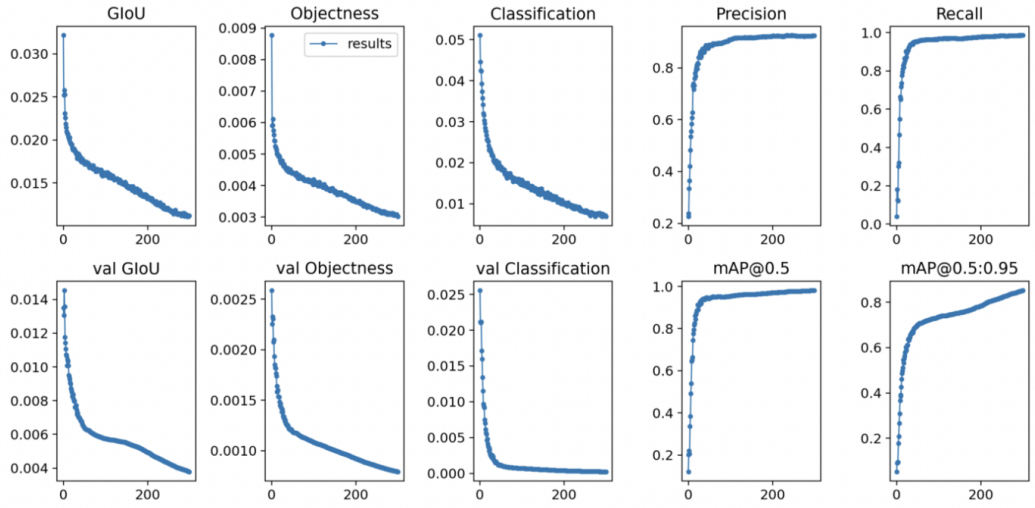

Figura 22. Resultados y evaluación del modelo (Lee, 2020).

Según el autor, el principal desafío de este proyecto es aumentar el conjunto de datos para así mejorar aún más la predicción del modelo.

3.3 VGG19 Sign Language Translator Python Opencv (Jorviader, 2019).

VGG19 consiste en una red convolucional pre entrenada, esta red ha sido entrenada sobre el dataset ImageNet. Uno de los trabajos desarrollados con este modelo VGG19 desarrollado en el año 2019 (*Ecabestadistica/sign-Language-Translator-Python-Opencv*), trabaja con imágenes de lengua de señas en tiempo real, el problema de este trabajo es que sólo funciona cuando la mano se pone en un área de la cámara web. Cuando la mano se pone en una parte externa que no sea el rectángulo visualizado en el video el modelo no funciona.

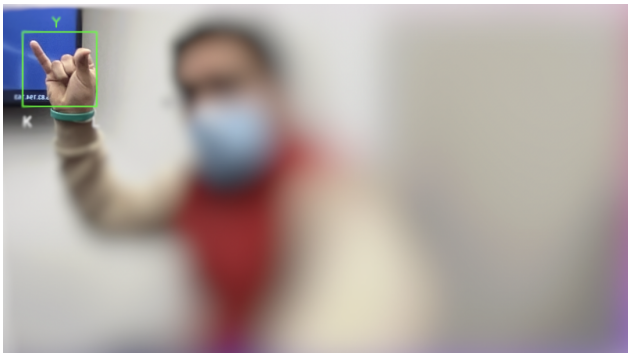

Figura 23. Reconocimiento de señas en tiempo real, sólo en el recuadro verde (Fuente propia).

3.4 American Sign Language Detection Android App using SSD_Mobilenet

Otro desarrollo similar es realizado por Deepak Raj (codePerfectPlus/ASL: American Sign Language Detection Android App using SSD_Mobilenet | Google Colab (github.com)). Él crea una aplicación de cámara que detecta continuamente la lengua de señas (delimitados por recuadros y su clase correspondiente a la seña) en los fotogramas vistos por la cámara trasera de su dispositivo.

El modelo que utiliza es entrenado en base al dataset del anterior nombrado David Lee en conjunto de un modelo pre entrenado (SSD MobileNet) y da un paso hacia la plataforma

móvil, exportando el modelo a TensorFlow Lite. Si bien exportar el modelo a este formato reduce su rendimiento, sigue arrojando buenos resultados reconociendo letras, concluyendo que es una excelente forma de construir un modelo personalizado. Sin embargo sigue acarreando los problemas del proyecto realizado por David Lee, donde algunas letras en específico (D, E, P, R) las reconoce como otras y además no reconoce señas que estén un poco alejadas de la cámara.

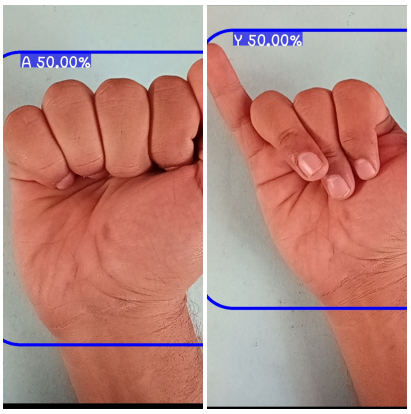

Figura 24. Aplicación Android de reconocimiento de lengua de señas americano (Deepak Raj, 2021).

Una ventaja notable que tiene el proyecto de Deepak Raj, es que utiliza recursos oficiales de Tensor Flow para entrenar modelos customizables, que permite construir una IA que reconozca objetos que el desarrollador necesite, siempre y cuando se tenga un dataset y su respectivo etiquetado de imágenes. Además TensorFlow proporciona un repositorio que permite probar el modelo en un dispositivo Android, repositorio que fue utilizado para desarrollar y probar esta aplicación.

3.5 Sign Language Recognition Android App, TensorFlow Lite (GPU), OpenCV, High Accuracy. (ElectroCode, 2021).

En 2021, el canal ElectroCode en la plataforma de Youtube, sube una serie de videos en el cual muestra el desarrollo de una aplicación móvil que traduce lengua de señas en tiempo real, obteniendo resultados muy buenos en cuanto a seguimiento de mano y porcentajes de acierto en la detección de la letra.

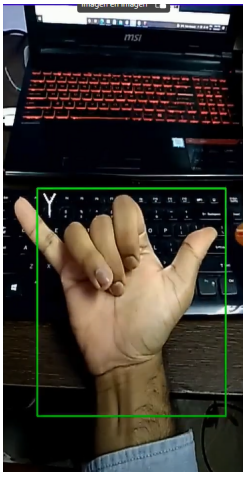

Figura 25. Aplicación móvil de reconocimiento de señas en tiempo real (ElectroCode, 2021).

Para el desarrollo de su proyecto entrena dos modelos:

- El primero que es encargado del seguimiento de la mano, es entrenado en base a una sola clase (Mano), con un robusto dataset de imágenes, para posteriormente ser utilizado en la aplicación con ayuda de la librería OpenCV para la detección de manos.
- El segundo modelo se encarga de reconocer la letra que encasilla el primer modelo, de esta manera la IA que hace la tarea de reconocimiento, no es necesario que sea entrenada con un robusto dataset que contenga multitudes de escenarios, sino que se constituye de imágenes recortadas que contengan sólo la mano la seña de la letra.

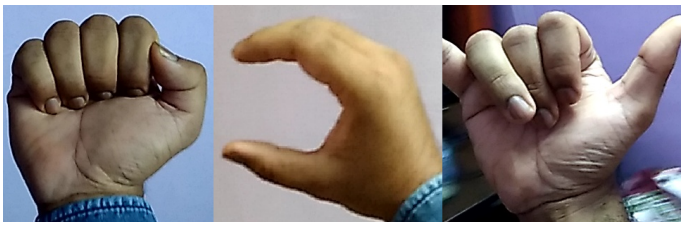

Figura 26. Ejemplos de imágenes utilizadas para el entrenamiento del modelo de reconocimiento de señas (Electro Code, 2021).

Ha sido un gran hallazgo encontrar este proyecto ya que es novedosa su forma de desarrollo y supone un gran avance al modularizar la clasificación de imagen en tiempo real. Por su contraparte, la aplicación desarrollada de esta forma, demanda muchos recursos, cerrándose repentinamente en algunos dispositivos, además el tamaño de la aplicación es 4 veces más

grande que aplicaciones que utilizan un solo modelo. Sumando a esto, existen pocos proyectos que se hayan realizado de esta forma, añadiendo complejidad en la tarea de desarrollo.

3.6 Custom Object Detection Training using YOLOv5. (Abdullah, 2022).

En la actualidad, decir que un estudiante con algunas semanas de estudio en el aprendizaje profundo, pueda entrenar una red neuronal, no es nada descabellado. Tanto así se ha reducido la complejidad de desarrollar redes neuronales, que es posible entrenar modelos con datos personalizados con tan solo 20 líneas de código.

Esta publicación sirve para el entrenamiento de modelos, con un conjunto de datos personalizados, asegurando que si el desarrollador tiene el conjunto de datos listo, puede comenzar a entrenar en 2 minutos.

Se describen 5 tipos de modelos que puede tener YOLOv5 (Tabla 2), cada una apuntando al sector tecnológico que tiene sus limitaciones de hardware. Por ejemplo en modelo YOLOv5n es el más rápido y liviano, siendo ideal para dispositivos móviles ya que una de sus principales desventajas es su capacidad de cómputo de estos aparatos, por su contraparte no tiene la misma precisión que el modelo YOLOv5x, que es el más pesado y preciso de todos los modelos, éste, siendo destinado para detecciones de objetos de última generación con GPU en la nube.

Model Name	Params (Million)	Accuracy (mAP 0.5)	CPU Time (ms)	GPU Time (ms)
YOLOv5n	1.9	45.7	45	6.3
YOLOv5s	7.2	56.8	98	6.4
YOLOv5m	21.2	64.1	224	8.2
YOLOv5l	46.5	67.3	430	10.1
YOLOv5x	86.7	68.9	766	12.1

Tabla 2. Modelos disponibles en YOLOv5 (Abdullah, 2022).

A su vez YOLOv5 proporciona un registro de Tensorboard que permite ver métricas, pérdidas e imágenes junto con todas sus predicciones de validación, sirviendo para monitorizar el rendimiento del entrenamiento.

Se muestra además una técnica de Augmentation, para mejorar los resultados de entrenamiento, que es llamada Mosaic Augmentation, ésta, además de aumentar la cantidad de imágenes del dataset, permite que el modelo pueda aprender a lidiar con un conjunto de imágenes variado y difícil.

YOLOv5 es la versión más reciente, que supuso un gran avance con respecto a las versión YOLOv3, en la Tabla 3, se puede ver la comparación entre los modelos Ultralytics YOLOv5, los cuales funcionan mucho mejor que los de YOLOv3.

Model	Image Resolution	AP@0.50	AP@0.50:0.95
Darknet YOLOv3	608	58.2	33.1
Darknet YOLOv3-tiny		35.4	16.6
Darknet YOLOv3-SPP		60.7	37.0
Ultralytics YOLOv3-SPP	608	62.8	43.1
Ultralytics YOLOv5n	608	45.7	28.0
Ultralytics YOLOv5s		56.8	37.4
Ultralytics YOLOv5m		64.1	45.4
Ultralytics YOLOv5l		67.3	49.0
Ultralytics YOLOv5x		68.9	50.7

Tabla 3. Comparación entre YOLOv5 y YOLOv3. (Abdullah, 2022).

En dicha publicación se entrenan tres modelos distintos, el YOLOv5s, YOLOv5m y finalmente un método de congelamiento de capas del modelo YOLOv5m, con el fin de compararlos y ver el rendimiento que tiene cada uno. El modelo está enfocado en detección de objetos personalizado que contendrá 5 clases de vehículos (coche, autobús, motocicleta, camión y ambulancia), utilizando un conjunto de datos de la plataforma Roboflow, que contiene 439 imágenes de entrenamiento (train), 125 de validación (valid) y 65 de pruebas (test).

El conjunto de datos debe seguir la siguiente estructura (Figura 20), las carpetas que contienen los conjuntos de datos de pruebas (test), entrenamiento (train), validación (valid), y sus subcarpetas que contienen las imágenes (images) y el etiquetado de cada una de estas (labels)

```
├── test
│   ├── images
│   └── labels
├── train
│   ├── images
│   ├── labels
│   └── labels.cache
├── valid
│   ├── images
│   ├── labels
│   └── labels.cache
├── data.yaml
├── README.dataset.txt
└── README.roboflow.txt
```

Figura 27. Estructura de directorios del conjunto de datos. (Abdullah, 2022).

Otra propiedad importante que nombra es el archivo YAML del conjunto de datos. Archivo que contiene las rutas de acceso a los datos de entrenamiento y validación, junto con los nombres de clases (Figura 21).

```yaml
train: ../train/images
val: ../valid/images
nc: 5
names: ['Ambulance', 'Bus', 'Car', 'Motorcycle', 'Truck']
```

Figura 28. Archivo YAML. (Abdullah, 2022).

El siguiente paso es ir de lleno al entrenamiento de los modelos utilizando los algoritmos disponibles que ofrece el repositorio oficial de YOLOv5. Este recurso contiene un archivo que permite entrenar un modelo con un simple comando, y dependerá del hardware utilizado, el tiempo que demorará éste proceso, recomendándo disponer de una GPU para realizar el entrenamiento, ya que acorta sustancialmente el tiempo de ejecución. Cabe aclarar que todos fueron entrenados con 25 epochs.

Durante el entrenamiento se guardan todas las predicciones de validación en un directorio y es posible visualizarlas luego de que el proceso termine. Esta funcionalidad permite mostrar que:

1. El modelo YOLOv5s pudo detectar los objetos en todas las imágenes de entrenamiento, a excepción de una.

Luego de que el entrenamiento es completado con éxito, es posible ver la precisión de detección a continuación.

1. El modelo YOLOv5s fue capaz de lograr un mAP de 58.8% a 0.5 IoU y 41% a 0.5:0.95 IoU.

2. El modelo YOLOv5m fue capaz de lograr un mAP de 64% a 0.5 IoU y 45.6% a 0.5:0.95 IoU.

3. El modelo YOLOv5m por congelamiento de capas, fue capaz de lograr un mAP de 64% a 0.5 IoU y 45.6% a 0.5:0.95 IoU.

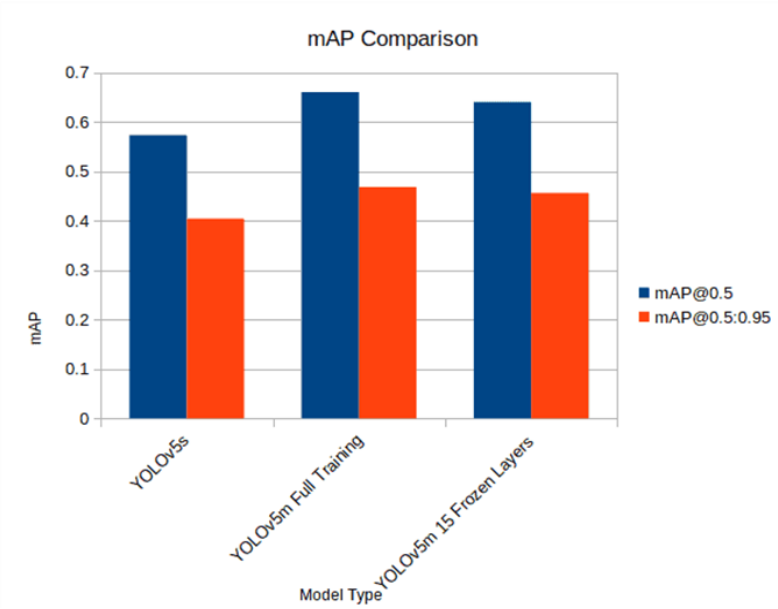

Figura 29. Gráfica comparativa de mAP (Abdullah, 2022).

Otra funcionalidad disponible es la inferencia de imágenes que nos permite evaluar lo que el modelo ha aprendido y qué tan preciso es. Comparando los distintos modelos se obtuvieron los siguientes resultados:

1. YOLOv5s: solo fue capaz de detectar un automóvil en una imagen que contaba con más de 10 tipos de vehículos y además detecta los autobuses erróneamente como ambulancia.

2. YOLOv5m: Es capaz de detectar más vehículos y con más confianza. Pero todavía se detectan algunos de los autobuses y camiones como ambulancia.

3. YOLOV5m con congelamiento de capas: para algunos de los objetos los resultados son mucho mejores en comparación con las dos inferencias anteriores, pero en algunas pruebas no logra detectar la totalidad de vehículos en la imagen.

Model Type	GPU Inference Speed in ms (FPS)	CPU Inference Speed in ms (FPS)
YOLOv5s	8.0 ms (125 FPS)	55 ms (18 FPS)
YOLOv5m Full Training	16 ms (62 FPS)	133 ms (7.5 FPS)
YOLOv5m Frozen Layers	16 ms (62 FPS)	133 ms (7.5 FPS)

Tabla 4. Tabla comparativa de velocidad de inferencia. (Abdullah, 2022).

Capítulo 4.- Estudio del problema y solución propuesta

Resolver ésta problemática ha sido un proceso de continuo desarrollo y aprendizaje, en base a pruebas y experimentaciones. Los algoritmos y herramientas utilizadas están disponibles de forma pública, el desafío está en saber utilizar efectivamente estas herramientas. Un buen conjunto de datos será la clave para el desarrollo exitoso de nuestro modelo de reconocimiento. Una vez creado el modelo se implementó en una aplicación móvil para dispositivos android que es capaz de reconocer en tiempo real las letras del alfabeto LSCh.

La forma de desarrollo de nuestra aplicación utilizó una metodología de desarrollo evolutivo. Primeramente nos centraremos en el desarrollo de un prototipo inicial (modelo de prototipos) que pueda responder a las necesidades de este proyecto. Posteriormente se mejoró el modelo conforme concurrió el transcurso del proyecto. Las tres primeras etapas (Figura 30) son las más significativas durante el transcurso de la investigación. Puesto que si en la evaluación del modelo descubrimos que la precisión de clasificación del modelo no es buena, entonces es necesario volver a preparar o corregir el conjunto de datos para nuevamente volver a entrenar el modelo y posteriormente generar su evaluación.

Finalmente con el modelo ya evaluado y considerando que ya es capaz de realizar la tarea de clasificación en tiempo real, se procede a desarrollar la aplicación Android que pueda utilizar el modelo creado.

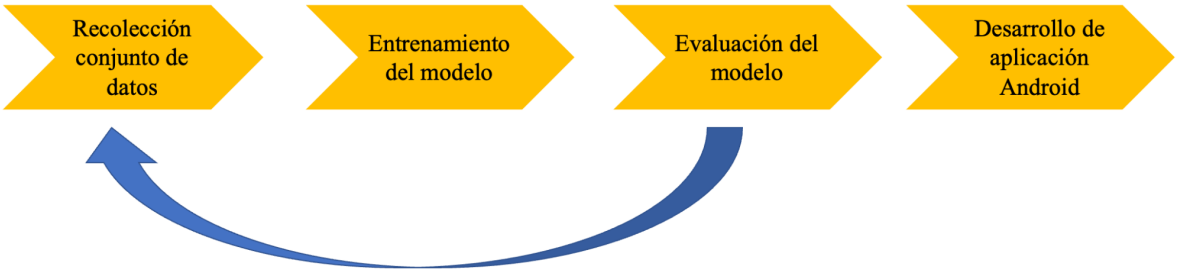

Figura 30. Diagrama de desarrollo de la investigación (Fuente propia).

4.1 Dataset

En esta investigación se utilizan 3 distintos datasets conforme avanzó la investigación. Se utilizan tres tipos de datasets distintos por varios motivos. El primer problema es que los datasets de LSCH no se encuentran de forma pública. En contraste, se puede acceder al conjunto de datos del alfabeto americano. Los datasets ASL presentan el problema de que tienen 19 letras en común con el alfabeto de lengua de señas chileno, es decir, faltarían 9 letras. Siendo esta la principal propuesta de valor de la investigación ya que se tuvo que recolectar un nuevo dataset desde cero, crearlo y etiquetar imagen por imagen. El segundo problema que presentan los datasets ASL, es que si bien tienen muchas imágenes para poder entrenar los modelos, es que las imágenes en sí se enfocan todas casi en la misma distancia y los mismos fondos, por tanto, cuando se trata de utilizar el clasificador en tiempo real, funciona mal. Por estos motivos, se utilizan sólo algunas imágenes de los datasets ASL y el resto se complementa con nuestro propio dataset obtenido de forma manual.

Los siguientes son los datasets utilizados:

4.1.1 Alfabeto ASL (Akash, Kaggle)

El dataset Alfabeto ASL contiene un conjunto de datos de entrenamiento con 87.000 imágenes de 200 x 200 píxeles. Hay 29 clases, de las cuales 26 son para las letras AZ y 3 clases para poder realizar otras acciones. El conjunto de datos de prueba contiene solo 29 imágenes, para fomentar el uso de imágenes de prueba del mundo real.

Acceso al dataset de Kaggle:

https://www.kaggle.com/datasets/grassknoted/asl-alphabet

4.1.2 American Sign Language Letters Dataset (David Lee)

Este conjunto de datos es un conjunto de datos de detección de objetos de cada letra ASL con un cuadro delimitador. Este conjunto de datos consta de 1728 imágenes distribuidas en 26 clases en total. Tiene la ventaja de que las imágenes ya tienen el cuadro delimitador, ahorrando bastante tiempo a la hora previa de tener que entrenar el modelo de clasificación de imágenes en tiempo real.

Acceso al dataset de David Lee:

https://public.roboflow.com/object-detection/american-sign-language-letters/1/download

4.1.3 Dataset recolectado manualmente

Este dataset consta de 663 imágenes tomadas manualmente por nosotros en distintos escenarios posibles, de las distintas letras del alfabeto de lengua de señas chilena. Principalmente lo creamos para suplir las ocho letras (E, G, M, N, Ñ, Q, T, U) que son distintas en comparación al alfabeto americano. También cumple el objetivo de mejorar los datasets anteriores, tomando imágenes desde distintas distancias, fondos y posiciones de las otras 18 letras del alfabeto.

4.1.4 Roboflow

Roboflow es una plataforma para desarrolladores que busquen crear aplicaciones de visión artificial, la cual proporciona todas las herramientas necesarias para desplegar un modelo de visión artificial e implementarlo en investigaciones o aplicaciones. Se puede utilizar para:

- Etiquetar imágenes de forma colaborativa o cargar etiquetas existentes.
- Convertir anotaciones XML de COV existentes a anotaciones JSON de COCO.
- Preprocesar imágenes.
- Aplicar técnicas de *data augmentation*.
- Generar formatos de etiquetados como TFRecords e implementaciones de YOLO.
- Compartir conjuntos de datos con equipos de trabajo.
- Entrenar modelos.
- Implementar los modelos en APIs.
- Obtener y compartir conjuntos de datos públicamente.

La completitud de éste sistema lo ha llevado a tener un impacto positivo en el área de la visión computacional, llegando a ser utilizada por grandes y pequeñas empresas como Walmart, Amgen, USG, Getaround, etc. Tan bueno ha sido el recibimiento de la comunidad de Machine Learning, que hasta la fecha se han realizado 66 millones de etiquetado de imágenes, 90 mil conjuntos de datos creados, 14 millones de inferencias realizadas en APIs

públicas y 7 mil modelos pre entrenados.

Sumado a lo anterior, las razones por las que utilizamos este sistema, es porque permite crear conjuntos de datos de datos personalizados y exportarlos en el formato necesario para entrenar los modelos con YOLOv5. Para llevar a cabo ésto, en primer lugar realizamos la subida y etiquetado de imágenes: éste proceso nos ayuda a generar de forma automática las clases que tendrá el dataset. Posteriormente realizamos uso de la separación del conjunto de datos en conjuntos de entrenamiento, validación y pruebas. Luego empleamos el uso de técnicas de preprocesamiento de imágenes y de *data augmentation*. Y finalmente la exportación del dataset para entrenar un modelo.

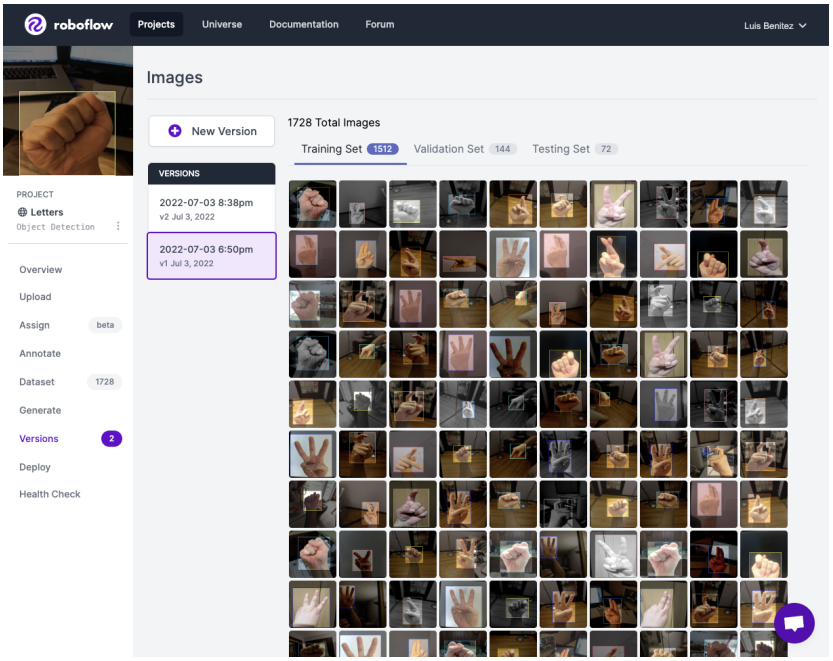

Figura 31. Etiquetado de imágenes a través de Roboflow (Fuente propia).

4.2 Experimentación

En un principio se realizaron entrenamiento de pruebas con YOLOv5, para comprobar e investigar los conceptos del aprendizaje profundo llevados a la práctica. Para todos los entrenamientos se utilizaron los pesos de un modelo pre entrenado COCO. Además se utilizó

un batch size de 16, debido a que la cantidad de datos es poca y sirve para optimizar los tiempos de entrenamiento.

Los concepto de *overfitting* (sobreajuste) y *underfitting* (sub-ajuste) se ve contrarrestado por la forma de entrenar del algoritmo de YOLOv5, debido a que siempre guarda el mejor modelo del entrenamiento realizado, evitando que se realicen más *epochs* de los necesarios que provocan un overfitting o menos de los *epochs* necesarios provocando *underfitting*, sin embargo puede llegar a presentarse y se va a comprobar.

Es por ello que nos planteamos las siguientes preguntas para desarrollar los entrenamientos de prueba.

1. ¿Qué ocurre cuando hay más imágenes de una letra que otra?

Para llevar a cabo este experimento se creó un dataset compuesto de dos clases similares para ver el comportamiento del modelo entrenado, es por ello que se eligieron las señas N y M. Para la letra N se utilizaron 14 imágenes de entrenamiento, 10 de validación y 10 de prueba, mientras que para la letra M se utilizaron 34 imágenes de entrenamiento, 10 de validación y 10 de prueba.

El entrenamiento dejó de obtener mejoras a partir de la pasada 150, obteniendo los siguientes resultados:

Clases	P	R	mAP 50	mAP 50-95
Total	0.929	0.941	0.956	0.625
M	0.898	0.983	0.962	0.699
N	0.961	0.9	0.951	0.551

Tabla 5. Resumen de modelo entrenado con cantidad de imágenes distinta (Fuente propia).

Si bien los resultados obtenidos de las métricas son bastante positivos y proponen una precisión alta en el modelo, cuando éste es llevado a la práctica sus predicciones no son del todo correctas, de las 19 imágenes de prueba 3 no fueron acertadas.

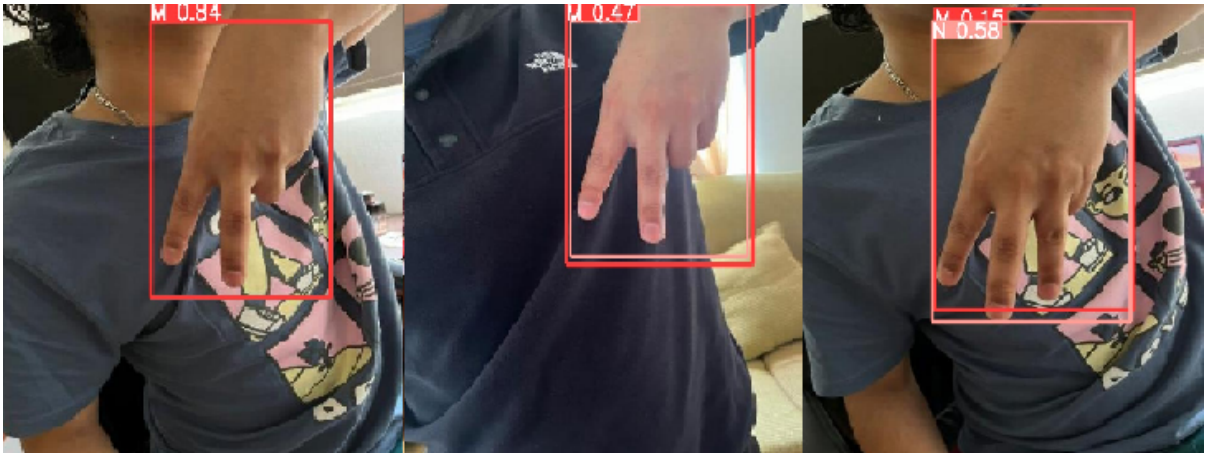

Figura 32. Inferencias fallidas de modelo entrenado con cantidad de imágenes distinta (Fuente propia).

Se realizó el mismo experimento pero con la misma cantidad de datos para ambas clases, 25 imágenes de entrenamiento, 11 imágenes de validación y 10 imágenes de pruebas para cada una de las señas, obteniendo los siguientes resultados:

Clases	P	R	mAP 50	mAP 50-95
Total	0.877	0.831	0.916	0.664
M	0.754	1	0.908	0.673
N	1	0.662	0.924	0.654

Tabla 6. Resumen de modelo entrenado con cantidad de imágenes igual (Fuente propia).

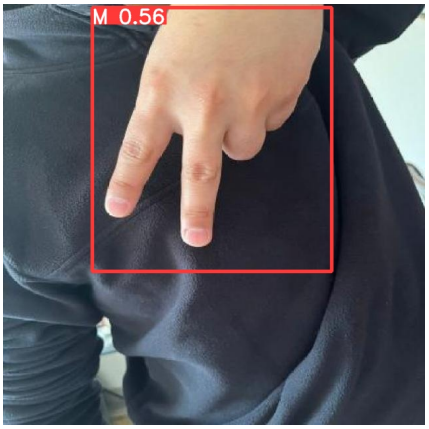

Figura 33. Inferencia fallida del modelo entrenado con cantidad de imágenes iguales (Fuente propia).

Es muy importante que la cantidad de imágenes por clase sea equitativa, ya que la carencia de imágenes se puede ver traducido en un sub-ajuste del modelo, impidiendo que el modelo pueda adquirir la cantidad de mapas de características necesarias para identificar correctamente las señas de forma proporcional. Por su contraparte, se puede ver una disminución en la tasa de recuperación y precisión del modelo, que hacen que el modelo en teoría empeore; se entiende como recuperación cuando el modelo es capaz de arrojar un resultado (correcto o incorrecto) al pasarle un dato, y precisión cuando el modelo acierta una predicción. Sin embargo en la práctica, el modelo que fue entrenado con la misma cantidad de imágenes para cada seña tiende a equivocarse mucho menos, en este caso al realizar inferencia con las 20 imágenes de prueba, el modelo solo se equivoca en 1 predicción, a diferencia del modelo con cantidad de imágenes desigual se equivocó en 3 predicciones.

2. Comparación entre modelo sin *data augmentation* y con *data augmentation.*

Para el siguiente experimento se utilizó el dataset American Sign Language Letters de David Lee, el cual contiene 720 imágenes y 26 clases, este conjunto de datos hace referencia a la lengua de señas americana, y está muy bien construido y permite entrenar un modelo bastante preciso.En primer lugar se entrenó un modelo sin *data augmentation* y los resultados fueron los siguientes:

- Los mejores resultados se alcanzaron en el epoch 342.
- De las 72 predicciones realizadas 10 fallaron.

Clases	P	R	mAP 50	mAP 50-95
Total	0.964	0.982	0.992	0.751
A	0.99	1	0.995	0.829
B	0.981	1	0.995	0.801
C	0.991	1	0.995	0.768
D	0.925	1	0.995	0.848
E	1	0.939	0.995	0.766
F	0.963	1	0.995	0.82
G	0.895	0.946	0.984	0.647
H	0.958	1	0.995	0.71
I	0.967	1	0.995	0.705
J	1	0.981	0.995	0.726
K	0.959	1	0.995	0.832
L	0.914	1	0.963	0.703
M	0.905	1	0.995	0.729
N	1	0.852	0.995	0.793
O	1	0.985	0.995	0.831
P	0.959	1	0.995	0.649
Q	0.978	1	0.995	0.682
R	0.991	1	0.995	0.773
S	0.964	1	0.995	0.728
T	1	0.947	0.995	0.744
U	0.969	1	0.995	0.764
V	1	0.937	0.995	0.756
W	0.837	1	0.972	0.697
X	1	0.934	0.995	0.801
Y	0.935	1	0.995	0.693
Z	0.974	1	0.995	0.724

Tabla 7. Resumen de modelo entrenado sin data augmentation (Fuente propia).

Posteriormente se aplicó data augmentation al conjunto de datos, aumentando a 1728 la cantidad de imágenes, es decir se agregaron 1008 imágenes adicionales.

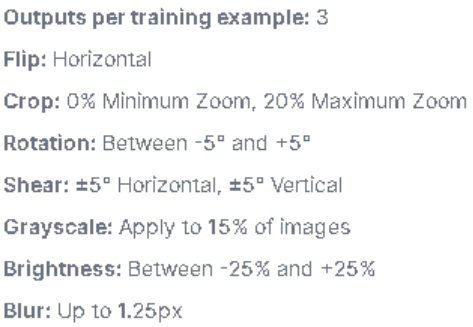

Figura 34. Técnicas utilizadas para aumentar la cantidad de imágenes (Fuente propia).

Para el modelo que utilizó *data augmentation*, los resultados fueron los siguientes:

- Los mejores resultados se alcanzaron en el epoch 280.
- De las 72 predicciones realizadas 7 fallaron.

Clases	P	R	mAP 50	mAP 50-95
Total	0.901	0.883	0.964	0.785
A	1	0.657	0.962	0.745
B	0.991	0.778	0.973	0.829
C	1	0.71	0.995	0.852
D	1	0.96	0.995	0.859
E	0.953	1	0.995	0.809
F	0.885	1	0.995	0.804
G	0.816	1	0.962	0.769
H	0.989	0.889	0.961	0.757
I	0.902	0.5	0.745	0.646
J	0.989	1	0.995	0.61
K	0.973	0.883	0.955	0.792

L	0.763	1	0.995	0.803
M	0.977	0.875	0.971	0.778
N	0.664	1	0.912	0.784
O	1	0.881	0.995	0.754
P	1	0.727	0.93	0.734
Q	0.786	1	0.995	0.796
R	0.986	1	0.995	0.818
S	0.963	1	0.995	0.838
T	0.803	0.682	0.948	0.79
U	1	0.992	0.995	0.82
V	0.744	0.8	0.818	0.647
W	0.943	1	0.995	0.929
X	0.328	1	0.995	0.895
Y	1	0.672	0.995	0.721
Z	0.964	1	0.995	0.829

Tabla 8. Resumen de modelo entrenado con data augmentation. (Fuente propia).

Analizando los resultados obtenidos del experimento, se puede ver que al efectuar *data augmentation* al conjunto de datos, la cantidad de épocas necesarias para entrenar el modelo son mucho menos, sin embargo el tiempo necesario para realizar cada una es mucho mayor, ya que la cantidad de datos es más grande. Además se observa una mejoría en el rendimiento del modelo al aplicar *data augmentation*, ya que pasó de equivocarse en 10 predicciones a 7 predicciones, esto se puede atribuir a que el modelo se entrenó con mucho más ejemplos. No obstante empeoró mucho la precisión del modelo al reconocer la letra X pasando de 1 a 0.328, esto se puede dar porque el conjunto de imágenes de validación y pruebas no cambia, y la *data augmentation* solo se emplea en el set de entrenamiento, lo que no es beneficioso para los indicadores, pero sí para el modelo en la práctica ya que el modelo es entrenado con imágenes que presentan modificaciones.

4.3 Desarrollo del modelo

En el siguiente apartado se describen las consideraciones que se tuvieron para generar el modelo, para ello las fotografías fueron capturarlas desde distintos ángulos, variar escenarios de fondo, y cambio de ropa de la persona que realizaba las señas. El etiquetado, la aumentación de datos y exportación del dataset se realizó con la ayuda de la plataforma roboflow.

Otras observaciones:

- Se utilizaron imágenes del conjunto de datos de David Lee, y se ocuparon sólo imágenes de las letras A, B, L, R, W, Y, cumpliendo que las imágenes a reutilizar fueran de la perspectiva en la cual la cámara apunta al sujeto que realiza las señas. (añadir imagen de ejemplo)
- Para las señas donde se realizan gestos para representar las letras, se capturaron imágenes de la última posición de la mano al realizarla.
- Se tomaron en cuenta los experimentos realizados con anterioridad, por lo tanto la cantidad de imágenes para cada letra del LSCh es de 26 imágenes para cada una, a excepción de las reutilizadas del repositorio anteriormente nombrado y se aplicaron las técnicas de *data augmentation*, para que sea entrenado con un volumen mayor de datos y sea capaz de identificar una variedad más grande de datos de entrada.
- Por el momento la letra Ñ no será añadida al conjunto, ya que el gesto realizado para representarla, es similar a la N al finalizar el gesto y similar a la H al empezar el gesto.

Para este entrenamiento se añadieron 663 imágenes al dataset, siendo aproximadamente 26 imágenes adicionales por cada clase del conjunto de datos, quedando distribuidas de la siguiente forma:

- 769 imágenes para el set de entrenamiento.
- 222 imágenes para el set de validación.
- 144 imágenes para el set de prueba.

Y aplicando *data augmentation* la cantidad de imágenes para el set de entrenamiento aumenta a 2300 imágenes.

El entrenamiento dejó de presentar mejoras a partir de la época 163, obteniendo los siguientes resultados:

Clases	P	R	mAP 50	mAP 50-95
Total	0.916	0.914	0.973	0.892
A	1	0.874	0.995	0..967
B	1	0.903	0.995	0..906
C	0.968	1	0.995	0..929
D	0.845	1	0.972	0..929
E	0.961	1	0.995	0..97
F	1	0.935	0.995	0.974
G	0.981	1	0.995	0.914
H	0.988	1	0.995	0.937
I	0.533	0.5	0.62	0.608
J	0.991	1	0.995	0.706
K	0.856	1	0.995	0.926
L	0.767	1	0.995	0.945
M	0.991	0.875	0.971	0.861
N	0.625	1	0.945	0.945
O	1	0.89	0.995	0.904
P	0.991	0.857	0.964	0.772
Q	0.785	1	0.995	0.821
R	1	0.886	0.995	0.915
S	0.941	1	0.995	0.964
T	1	1	0.995	0.914
U	1	0.894	0.995	0.84
V	1	0.706	0.938	0.842
W	0.712	1	0.995	0.929
X	0.902	1	0.995	0.995
Y	1	0.843	0.995	0.847
Z	0.969	1	0.995	0.933

Tabla 9. Resultado de entrenamiento con nuevo dataset (Fuente propia).

Ya entrenado el modelo, se probó la predicción del modelo con las 114 imágenes del set de pruebas (*inferencia*), fallando una sola predicción en el reconocimiento de la seña (Figura 33). En cuanto a los indicadores que arrojó el entrenamiento del modelo se obtienen buenos resultados en todos, con una precisión de 91,6%, tasa de recuperación de un 91,4%, una precisión media de 97,3% arriba del 50% de precisión y finalmente un 89,2% de precisión media para predicciones con un 95% de precisión. Sin embargo quedan por mejorar la precisión de las letras I, L, N, Q y W, que tienen una precisión de 53,3%, 78,5%, 62,5%, 78,5 y 71,2% respectivamente.

Figura 35. Inferencia fallida del primer modelo prototipo (Fuente propia).

4.4 Desarrollo aplicación móvil

Para probar el modelo en dispositivos móviles, primero se exportó el modelo en formato tflite y se utilizó código fuente del repositorio: zldrobit/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > iOS (github.com), el cual contiene un proyecto en lenguaje Java que con ayuda del IDE Android Studio se logró montar la aplicación en un dispositivo móvil. A continuación se presenta el procedimiento realizado desde la exportación e implementación del módulo IA, hasta el diseño y funcionamiento de la aplicación móvil. Para utilizar el modelo los pasos son los siguientes:

1. **Convertir a un archivo tflite:**

El procedimiento para poder exportar el modelo de YOLOV5S a formato tflite requiere ejecutar las líneas de código de la figura 36. Se exporta el modelo a formato tflite debido a la necesidad de tener que utilizar este modelo en un dispositivo móvil.

```
!python export.py --weights runs/train/exp/weights/best.pt --include tflite --int8 --img 320 --data {dataset.location}/data.yaml
!python detect.py --weights runs/train/exp/weights/best-int8.tflite --img 320
```

Figura 36. Código para exportar a modelo tflite (Fuente propia).

Este formato está optimizado y tiene una baja latencia además de un alto rendimiento que consume pocos recursos del dispositivo. TensorFlow en android toma los datos, los procesa y genera una predicción basada en la lógica del modelo utilizado. Los datos se pasan al modelo en formato de tensores. Cuando un modelo procesa los datos, lo que se conoce como ejecutar una *inferencia*, genera resultados de predicción como nuevos tensores y luego los pasa a la aplicación android para que pueda tomar medidas, como mostrar el resultado de la clasificación de la imagen en tiempo real.

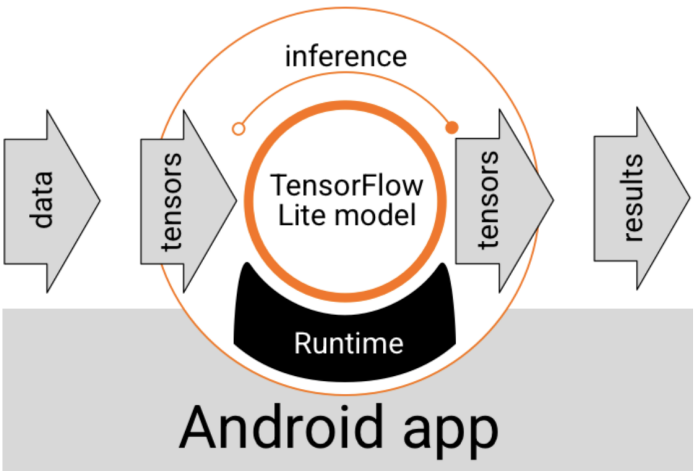

Figura 37. Flujo de ejecución funcional para modelos TensorFlow Lite en aplicaciones de

Android (Tensorflow, 2022).

2. **Utilizar archivo tflite en proyecto android:**

Como mencionamos anteriormente en la introducción, el proyecto android base para la aplicación móvil está desarrollado sobre el código de fuente del repositorio:

zldrobit/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > iOS (github.com)

La estructura de este proyecto android es la siguiente:

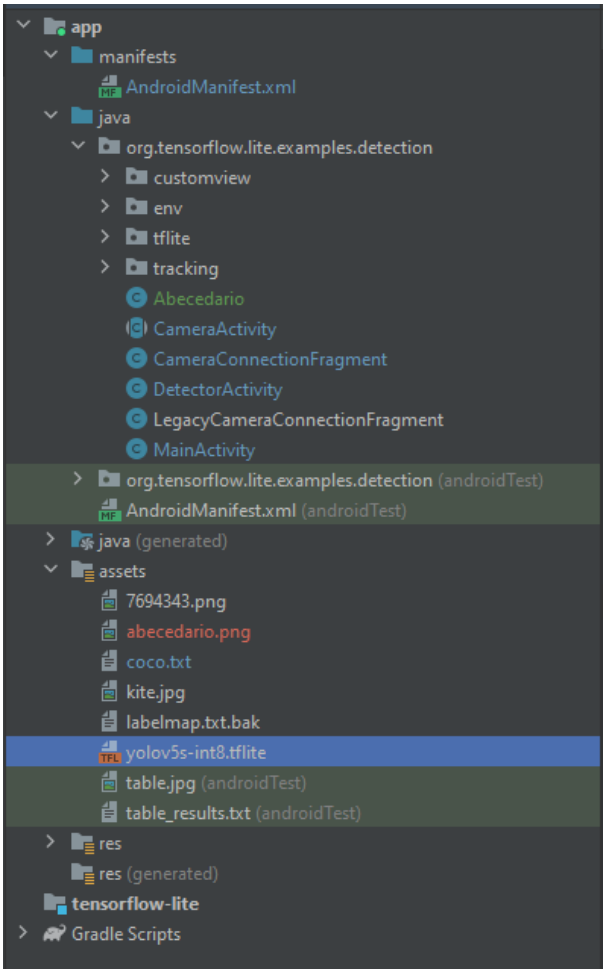

Figura 38. Estructura de proyecto Android (Fuente propia).

La estructura del proyecto se segmenta básicamente en 4 carpetas, la primera carpeta "customview" contiene las clases encargadas de la vista de la aplicación, la segunda llamada "env" contiene clases igualmente encargadas de aspectos visuales de la aplicación, la tercera carpeta "tflite" contiene las clases encargadas de interactuar con el modelo tflite, y la carpeta "tracking" que contiene las clases encargadas del seguimiento en tiempo real de los objetos.

Dentro de la carpeta assets se encuentra nuestro modelo de YOLO en formato tflite a utilizar, también se encuentra un archivo llamado coco.txt que contiene las etiquetas de nuestras clases, del abecedario de lengua de señas chilena. Los demás archivos dentro de la carpeta assets son utilizados como recursos visuales dentro de nuestra aplicación.

3. Diseño de la interfaz móvil y desarrollo de la aplicación:

Comenzamos construyendo un diseño de cómo se verá la aplicación móvil, llegando al siguiente prototipo de diseño:

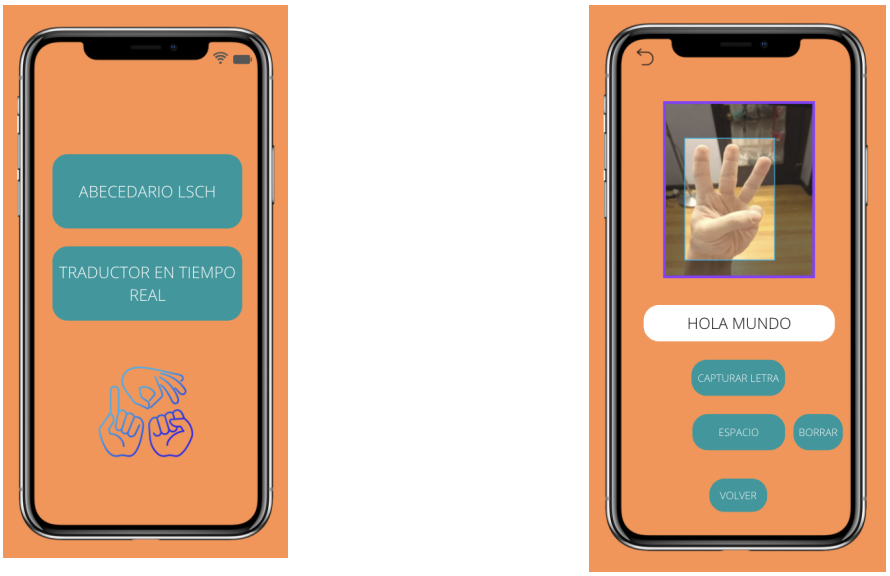

Figura 39. Diseño de propósito de la aplicación móvil (Fuente propia).

Posteriormente procedemos a programar la aplicación, modificando algunas clases dentro del proyecto. Dentro de la carpeta tflite se encuentra la clase “YoloV5Classifier”, dentro de esta clase se trabajó modificando funciones para así permitirnos la tarea objetivo que es al pulsar el botón “capturar letra” podamos obtener la letra actual la cual se está detectando en ese preciso instante a través del modelo de reconocimiento.

```
288     for (int k = 0; k < labels.size(); k++) {
289         //1.find max confidence per class
290         PriorityQueue<Recognition> pq =
291                 new PriorityQueue<~>(
292                         initialCapacity: 50,
293                         new Comparator<Recognition>() {
294                             @Override
295                             public int compare(final Recognition lhs, final Recognition rhs) {
296                                 // Intentionally reversed to put high confidence at the head of the queue.
297                                 return Float.compare(rhs.getConfidence(), lhs.getConfidence());
298                             }
299                         });
300
301         for (int i = 0; i < list.size(); ++i) {
302             if (list.get(i).getDetectedClass() == k) {
303                 pq.add(list.get(i));
304             }
305         }
306
307         //2.do non maximum suppression
308         while (pq.size() > 0) {
309             //insert detection with max confidence
310             Recognition[] a = new Recognition[pq.size()];
311             Recognition[] detections = pq.toArray(a);
312             Recognition max = detections[0];
313             nmsList.add(max);
314             pq.clear();
315
316             for (int j = 1; j < detections.length; j++) {
317                 Recognition detection = detections[j];
318                 RectF b = detection.getLocation();
319                 setLetraActual(detection.getTitle());
320                 if (box_iou(max.getLocation(), b) < mNmsThresh) {
321                     pq.add(detection);
322                 }
323             }
324         }
325     }
326
```

Figura 40. Clase “YoloV5Classifier” encargada de detectar y reconocer la letra actual enfocada en tiempo real (Fuente propia).

Las vistas de la aplicación se programaron en base al diseño previamente establecido, modificando y creando los archivos respectivos de diseño, dentro de la carpeta “layout”.

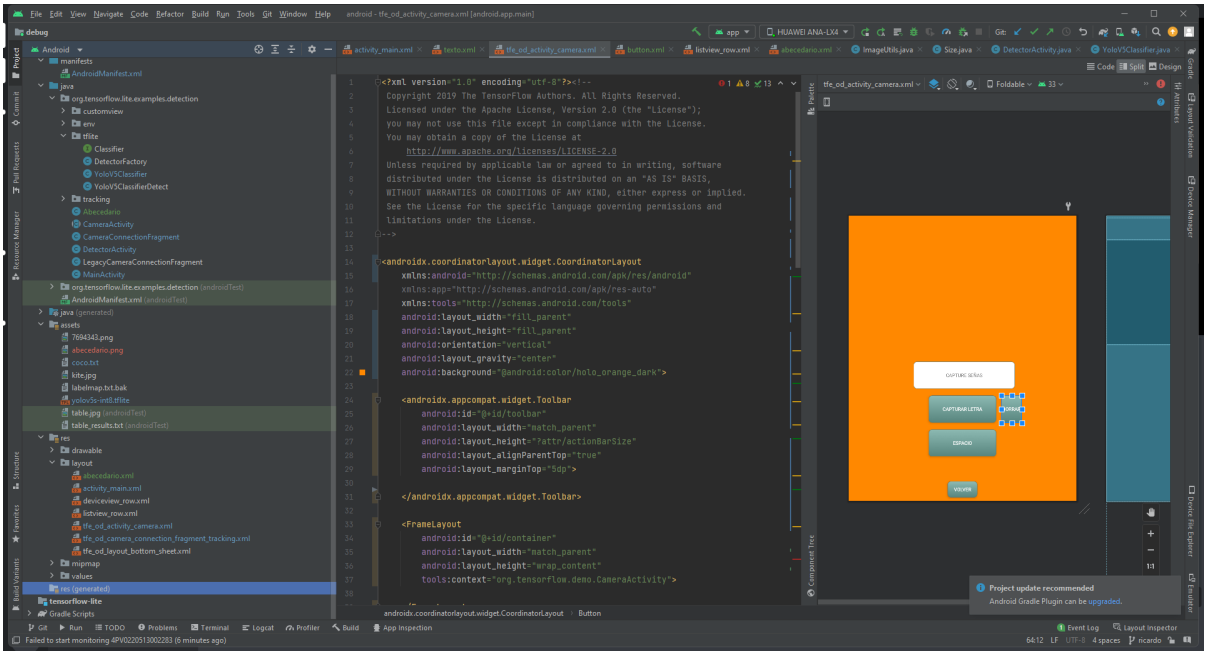

Figura 41. Programación de módulos de diseño de la aplicación móvil (Fuente propia).

4. **Compilar la aplicación.**

Finalmente compilamos la aplicación, teniendo un resultado exitoso. En su pantalla de inicio (Figura 41) cuenta con dos botones. El botón “ABECEDARIO LSCH” nos dirige a una vista con una imagen del alfabeto LSCh.

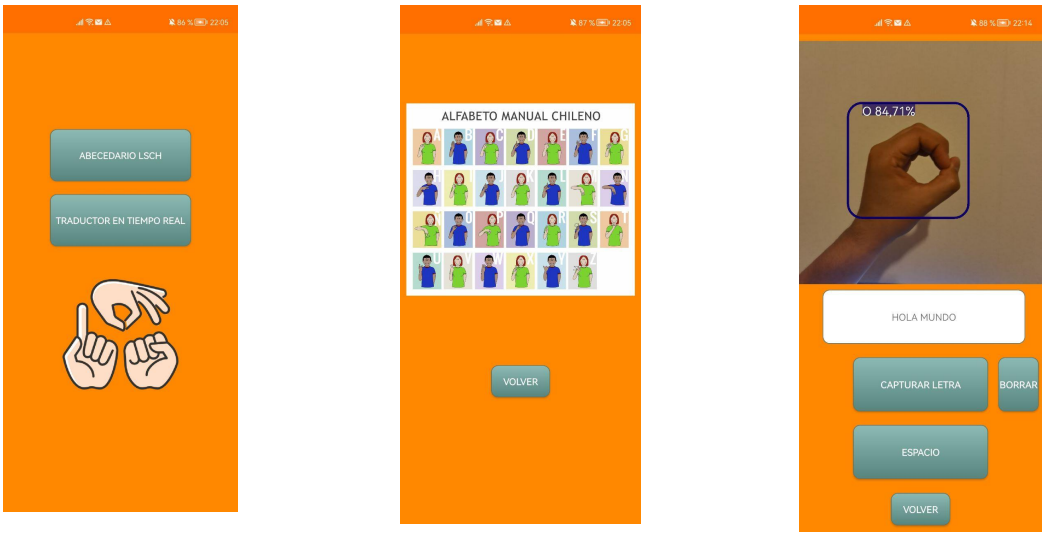

Figura 42. Pantalla de inicio y vista de botón "ABECEDARIO LSCH" (Fuente propia).

El segundo botón llamado "TRADUCTOR EN TIEMPO REAL" nos dirige al reconocedor en tiempo real de las letras del alfabeto LSCh (Figura 41).

El botón "CAPTURAR LETRA" tiene la función de captar la letra actual que está reconociendo el sistema de inteligencia, el botón "BORRAR" tiene la función de eliminar una letra dentro del cuadro de texto, el botón "ESPACIO" la función de generar un espacio entre una letra y otra, y un último botón "VOLVER" encargado de volver al menú principal de la aplicación.

4.5 Hardware y herramientas utilizadas

A continuación se mencionan las herramientas y el hardware utilizados durante el desarrollo de esta investigación. Cabe mencionar que para el entrenamiento del modelo de inteligencia artificial el equipo no tiene gran relevancia, ya que se utilizaron los servicios de google colab, en un principio el plan gratuito y luego el plan pro de google (De pago).

Hardware:

-Equipo Android Huawei P40.
-Notebook Asus TUF F15.
-Macbook Air M1.

Plataformas y entornos de desarrollo:

-Roboflow.
-Google Colab.
-Android Studio.

Lenguajes de programación:

-Python.
-Java.

4.6 Algoritmos y código

A continuación se presenta y se dispone el acceso a los distintos repositorios, para quienes deseen seguir esta línea de investigación se facilita el acceso al conjunto de datos utilizado, código Java de la aplicación android y el cuaderno de Google Colab con acceso al código python del entrenamiento del modelo con YOLO.

4.6.1 Repositorio código fuente aplicación Android

Código fuente de aplicación android: ktrilu/android-yolov5 (github.com)

Está compartido de forma pública y contiene todos los archivos necesarios para montar la aplicación en un dispositivo Android, entre ellos el modelo entrenado llamado yolov5s-int8.tflite, que en caso de futuras investigaciones se plantea utilizar éste repositorio para realizar pruebas a un modelo entrenado, solo basta con reemplazar el archivo con el mismo nombre anterior, pero con el modelo actualizado (la aplicación solo soporta modelos en formato tflite).

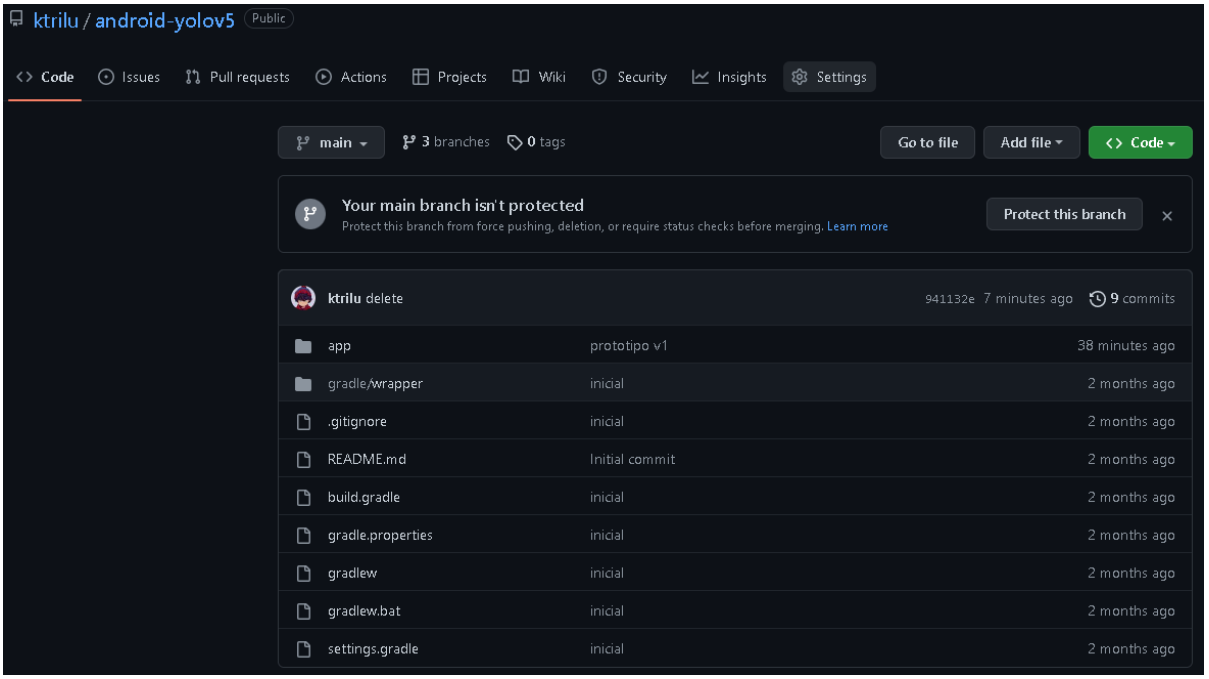

Figura 43. Repositorio Android con código Java de la aplicación móvil (Fuente propia).

4.6.2 Repositorio del conjunto de datos.

Conjunto de datos utilizado: Lengua de señas chilena 26 clases Dataset > Overview (roboflow.com)

Contiene las 26 clases utilizadas para este proyecto y puede ser exportado directamente desde enlace adjunto a diferentes formatos (YOLO, COCOJSON, TFRecord, Pascal VOC, etc.). También contiene una versión base del conjunto de datos, el cual puede ser descargado para aplicar las técnicas de aumentación de datos que requiera, y también puede servir como complemento para la creación de otro conjunto de datos con más imágenes.

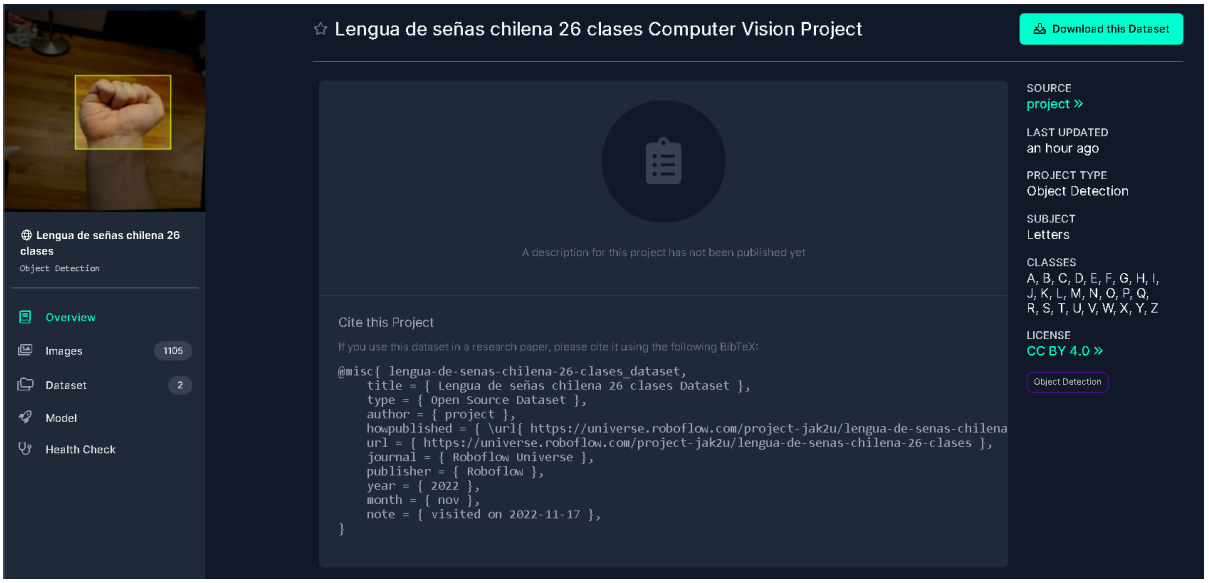

Figura 44. Repositorio del conjunto de datos utilizado (Fuente propia).

4.6.3 Cuaderno Google Colab con código del modelo

Este cuaderno de programación contiene la preparación del conjunto de datos para su utilización y el posterior entrenamiento de un modelo personalizado con YOLOv5, además contiene código para evaluar su desempeño mediante inferencias realizadas al modelo y gráficas que evalúan los indicadores de precisión, recuperaciones y media de precisión.

Enlace de acceso a cuaderno de programación utilizado:

https://colab.research.google.com/drive/1k1XbIhoF2Zj_01tFpHN-usInPZsHBpAf

Capítulo 5.- Conclusiones.

El exitoso resultado del funcionamiento de la aplicación móvil, logra reconocer las distintas letras del alfabeto de lengua de señas chilena, se concluye que gracias a la transferencia de aprendizaje, el resultado de entrenar un modelo de clasificación de imágenes resulta una tarea más sencilla, eficaz y eficiente. Pero esta eficacia del modelo dependerá netamente de tener un buen conjunto de datos. Un conjunto de datos con pocas imágenes, o bien similares entre sí, con el mismo fondo, la misma distancia del objeto a enfocar, e incluso la misma persona con la misma mano, provocará que el modelo funcione mal. En caso contrario, cuando se recolecta un buen conjunto de datos, con distintas manos, distancias, fondos, y cantidad de imágenes, se logra mejorar bastante y de forma impresionante el resultado del reconocimiento de letras del alfabeto LSCh en tiempo real.

La técnica data augmentation, empeora la precisión del modelo bajando de 0.964 a 0.901, esto debido a que aumenta la cantidad de imágenes del entrenamiento pero no los de validación, más que empeorarlo, concluimos que nos da una visión más crítica y real de la verdadera precisión del modelo. En contraste, como consecuencia de aumentar la cantidad de datos de imágenes provoca que el modelo se comporte mucho mejor en tiempo real.

Para el desarrollo del modelo final empleado en nuestra aplicación, se utilizaron imágenes de David Lee y otras 663 imágenes capturadas por nosotros, logramos mejorar la precisión del modelo de 0.901 a 0.916 y además aumenta considerablemente el mAP 0.5-0.95 de 0.785 a 0.89. Además, es el modelo que mejor se comporta en tiempo real alcanzando predicciones sobre el 90% de éxito. En base a estos resultados, ratificamos que el aumentar y preparar un buen conjunto de datos, mejorará el desempeño del modelo.

Dentro de las limitaciones de este estudio se encuentran principalmente dos limitaciones, la primera es haber conseguido un conjunto de datos bastante más amplio y la segunda es técnica, que nos impidió probar otras versiones de YOLO u algoritmos a utilizar, esto debido a que los entrenamiento de Google Colab son limitados. Los posibles trabajos futuros de esta investigación podrían dirigirse por dos líneas. La primera es en caso de que se disponga con los requerimientos técnicos para poder comparar la eficiencia y la precisión de la versión de YOLO utilizada en esta investigación con otras versiones u algoritmos disponibles, y si es posible poder mejorar el conjunto de datos. La segunda posible investigación es utilizando visión artificial y posiblemente OpenCV, es desarrollar un clasificador pero que en vez de letras pueda clasificar gestos de personas sordomudas.

Referencias

- Abdullah, K. (2022, April 19). *Custom Object Detection Training using YOLOv5 Custom Object Detection Training using YOLOv5*. LearnOpenCV. Retrieved August 3, 2022, from https://learnopencv.com/custom-object-detection-training-using-yolov5/

- Al-Qurishi, M., Khalid, T., & Souissi, R. (2021). Deep Learning for Sign Language Recognition: Current Techniques, Benchmarks, and Open Issues. *IEEE Radar Conference*. https://ieeexplore.ieee.org/abstract/document/9530569

- Bonilla Carrión, C., & Pino Mejías, R. (2020). *REDES CONVOLUCIONALES*. idUS. Retrieved October 12, 2022, from https://idus.us.es/bitstream/handle/11441/115221/TFG%20DGMyE%20Bonilla%20Carri%c3%b3n%2c%20Carmelo.pdf?sequence=1&isAllowed=y

- Chollet, F. (2017). *Deep learning with python*.

- Durán, J., & Torres, A. (2017). *Redes Neuronales Convo Redes Neuronales Convolucionales en R volucionales en R*. Universidad de Sevilla. Retrieved October 12, 2022, from https://idus.us.es/bitstream/handle/11441/69564/TFG_Jaime%20Dur%c3%a1n%20Su%c3%a1rez.pdf?sequence=1&isAllowed=y

- *ecabestadistica/sign-language-translator-python-opencv*. (n.d.). GitHub. Retrieved June 11, 2022, from https://github.com/ecabestadistica/sign-language-translator-python-opencv

- *ELEMENTOS BÁSICOS DE UNA RED NEURONAL ARTIFICIAL | Advanced Tech Computing Group UTPL*. (2007, August 31). Advanced Tech Computing Group UTPL. Retrieved August 4, 2022, from https://advancedtech.wordpress.com/2007/08/31/elementos-basicos-de-una-red-neuronal-artificial/

- Joseph Redmon, Santosh Divvala, Ross Girshick, & Ali Farhadi. (n.d.). You Only Look Once: Unified, Real-Time Object Detection.

- Kumar, V. (2020, June 10). *Hands-On Guide To Sign Language Classification Using CNN*. Analytics India Magazine. Retrieved June 11, 2022, from https://analyticsindiamag.com/hands-on-guide-to-sign-language-classification-using-cnn/

- Lee, D. (2020, October 15). *Using Computer Vision in Helping the Deaf and Hard of Hearing Communities with Yolov5*. David Lee. Retrieved June 11, 2022, from https://daviddaeshinlee.medium.com/using-computer-vision-in-helping-the-deaf-and-hard-of-hearing-communities-with-yolov5-7d764c2eb614

- Li, Y., Zhang, J., Hu, Y., Zhao, Y., & Cao, Y. (2022). Real-time Safety Helmet-wearing Detection Based on Improved YOLOv5. *Tech Science Press*. 10.32604/csse.2022.028224

- Rastgoo, R., & Escalera, S. (2021). Sign Language Recognition: A Deep Survey. *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2020.113794

- Recuero, P. (2019, January 15). *Deep Learning para todos los públicos: ¿Qué son los tensores? ¿Qué es TensorFlow?* Think Big Empresas. Retrieved August 4, 2022, from https://empresas.blogthinkbig.com/deep-learning-para-todos-los-publicos/

- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (n.d.). *CVPR 2016 Open Access Repository*. CVPR 2016 Open Access Repository. Retrieved June 11, 2022, from https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html

- *Servicio Nacional de la Discapacidad*. (2021, January 26). Servicio Nacional de la Discapacidad. Retrieved May 24, 2022, from https://www.senadis.gob.cl/region/valparaiso/d/noticias/8431/ministerio-de-desarrollo-social-y-familia-destaca-ley-que-reconoce-a-la-lengua-de-senas-como-lengua-oficial-de-las-personas-sordas

- *Sordera y pérdida de la audición*. (2021, March 2). WHO | World Health Organization. Retrieved May 24, 2022, from https://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss

- McCarthy, John, What Is Artificial Intelligence (2007, November 11)

- Russell, S. (2003) Stuart Russell on the Future of Artificial Intelligence.

- Minsky, M. (1990), The Age of Intelligent Machines: Thoughts About Artificial Intelligence.

- Mitchell, Tom (1997). Machine Learning.

- Stuart Russell (1997). Artificial Intelligence.