

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAISO - CHILE



“ANÁLISIS Y DESARROLLO DE UN MODELO DE RED
NEURONAL CON *TRANSFER LEARNING* PARA LA
INTERPRETACIÓN DE LA LENGUA DE SEÑAS CHILENA”

ISMAEL ANDRÉS CRUZ PEÑARANDA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Carlos Valle Vidal
Profesor Correferente: Ricardo Ñanculef

Agosto - 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: "ANÁLISIS Y DESARROLLO DE UN MODELO DE RED NEURONAL CON TRANSFER LEARNING PARA LA INTERPRETACIÓN DE LA LENGUA DE SEÑAS CHILENA"

Nombre del candidato(a): Ismael Andrés Cruz Peñaranda

Carrera / Grado: Ingeniería Civil Informática

Campus: Casa Central Valparaíso ; **Departamento:** Departamento de Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Carlos Antonio Valle Vidal, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 29/07/2025

Firma:

Estudiante o Candidato(a):

Fecha: 17/07/2025

Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

Este trabajo está dedicado a mis padres María Lorena Peñaranda y Jorge Cruz Jaramillo, quienes siempre me han estado apoyando e impulsando a seguir mis sueños, no podría haber llegado a este punto de mi vida sin ellos.

AGRADECIMIENTOS

Quiero partir agradeciéndole a mi familia por todo el apoyo, amor, comprensión y guía que me han entregado a lo largo de mi vida y me ha permitido desarrollarme en la persona que soy actualmente.

Quiero también agradecer al profesor Carlos Valle por no solo haberme entregado las herramientas y el conocimiento necesario durante sus clases para la elaboración de este proyecto, pero además por su disposición y guía durante el desarrollo de este.

En particular, quiero agradecerles a mis amigos Adolfo Solar, Bruno Rojas, Miguel Vergara, Claudio Rodriguez, Daniel Roco, y a mi mamá Lorena Peñaranda, por su ayuda en el desarrollo del *dataset* utilizado en este proyecto, sin su ayuda no hubiese sido capaz de desarrollar este trabajo.

Y por último, a todos los profesores de la universidad Santa María por brindarme la educación que me permitió llegar a este punto de mi vida académica.

RESUMEN

Resumen— Las lenguas de señas son sistemas completos de comunicación visual utilizados por personas en situación de discapacidad auditiva en todo el mundo. Debido a esto ha surgido un creciente interés en el desarrollo de sistemas automáticos para la interpretación de estas. A pesar de aquello existe una ausencia de estos sistemas para la lengua de señas Chilena. En este trabajo se aborda el desarrollo de un modelo de aprendizaje automático utilizando técnicas de *transfer learning* sobre los modelos *VGG16*, *ResNet50* y *InceptionV3* y métodos de optimización de hiperparámetros, aumento de datos y *fine tuning*. A pesar de las limitaciones del conjunto de datos utilizado, el modelo resultante presenta una precisión por sobre el 90 % en data nueva.

Palabras Clave— Aprendizaje Automático, Redes Neuronales, Reconocimiento de lengua de señas, Clasificación de Imágenes, Transferencia de Aprendizaje.

ABSTRACT

Abstract— Sign languages are fully formed systems of visual communication used around the world by people with hearing disabilities. Due to this the development of machine learning systems for their interpretation has been a growing field of research throughout the past few decades. Despite this there is an absence of these types of systems designed for the Chilean sign language. This work addresses the development of a machine learning model for recognizing the chilean sign language alphabet utilizing transfer learning techniques on the models *VGG16*, *ResNet50* and *InceptionV3* alongside the usage of techniques such as hyperparameter optimization and fine tuning. Despite the limitations in our dataset, the results show that the model is capable of predicting the correct sign with a greater than 90 % accuracy on completely new data.

Keywords— Machine Learning, Neural Networks, Sign Language Recognition, Image Classification, Transfer Learning.

GLOSARIO

ASL: *American Sign Language*, es la lengua de señas oficial utilizada en Estados Unidos.

ASOCH: Asociación de sordos de Chile.

CNN: *Convolutional Neural Network*.

ILSVRC: *ImageNet Large Scale Visual Recognition Challenge*.

LSA: Lengua de Señas Argentina.

LSCh: Lengua de Señas Chilena.

LSE: Lengua de Señas Española.

NNA: Niños, Niñas y Adolescentes.

RNA: Red Neuronal Artificial.

UTFSM: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Árbol de Problema	4
1.2 Impacto inicial de la solución	4
1.3 Objetivo General	5
1.4 Objetivos Específicos	5
1.5 Alcance	5
CAPÍTULO 2: MARCO CONCEPTUAL	6
2.1 Aprendizaje Automático	6
2.1.1 Aprendizaje Supervisado	7
2.1.2 Aprendizaje no Supervisado	7
2.1.3 Aprendizaje Semi-Supervisado	8
2.1.4 Aprendizaje reforzado	8
2.2 Redes Neuronales Artificiales	8
2.3 Aprendizaje Profundo (Deep Learning)	10
2.4 Redes Neuronales Convolucionales (CNN)	11
2.4.1 Capas de Convolución	11
2.4.2 Capas de Agrupación	12
2.4.3 Capas completamente conectadas	13
2.5 Problemas de Clasificación	14
2.5.1 Métricas de evaluación	14
2.5.2 Underfitting and Overfitting	15
2.6 Transfer Learning	16
2.6.1 Feature Extracion	16
2.6.2 Fine Tuning	17
2.6.3 Data Augmentation	18
2.7 Optimización de Hiperparámetros	19
2.8 Ablation Studies	20
2.9 Estado del arte	20
2.9.1 Reconocimiento en Tiempo Real del Alfabeto de Lengua de Señas Chilena Empleando Aprendizaje Automático [Benítez y Catril, 2022]	20

2.9.2	<i>American Sign Language Recognition Based on Transfer Learning Algorithms</i> [Mohsin et al., 2023]	21
2.9.3	<i>Isolated sign language recognition using hidden transfer learning</i> [Mia et al., 2023]	21
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN		23
3.1	Desarrollo de <i>dataset</i>	23
3.1.1	Pre-procesamiento de datos	24
3.2	Entrenamiento de modelo	25
3.2.1	Selección de modelo base candidato	25
3.2.2	Entrenamiento y selección de modelo	31
3.2.3	Selección de hiperparámetros para optimizador	32
3.3	<i>Ablation Study</i> propuesto	34
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		37
4.1	Herramientas	37
4.1.1	<i>Hardware</i>	37
4.1.2	<i>Software</i>	37
4.2	<i>Datasets</i>	37
4.2.1	Lengua de Señas Chilena 26 Clases (Beitez, Catril)	38
4.2.2	<i>American Sign Language Dataset</i> (Ayush Thakur)	38
4.2.3	<i>Dataset</i> recolectado manualmente	38
4.3	Selección de modelo base	41
4.3.1	Resultados VGG16	44
4.3.2	Resultados ResNet50	45
4.3.3	Resultados InceptionV3	47
4.4	Resultados de <i>Ablation Studies</i> propuestos	50
CAPÍTULO 5: CONCLUSIONES		54
5.1	Conclusiones Generales	54
5.2	Cumplimiento de los objetivos	55
5.3	Trabajo Futuro	55
REFERENCIAS BIBLIOGRÁFICAS		57

ÍNDICE DE FIGURAS

1	Árbol de problemas	4
2	Tipos de aprendizaje automático.	7
3	Red neuronal artificial básica de tres capas.	9
4	Ejemplos de funciones de activación comúnmente usadas.	10
5	Ejemplo de red neuronal tradicional en comparación una red neuronal profunda.	11
6	Ejemplo de una operación convolucional sobre una matriz	12
7	Ejemplo de <i>Max Pooling</i> y <i>Average Pooling</i>	13
8	Diagrama de una red neuronal convolucional básica	14
9	Visualización de subajuste y sobreajuste	16
10	Ejemplo de <i>feature extraction</i> en <i>transfer learning</i> .	17
11	Ilustración de proceso de <i>fine tuning</i>	18
12	Ejemplos de técnicas de aumento de imágenes.	19
13	Distintas representaciones para la letra Y en la LSCh	24
14	Arquitectura de red VGG16.	25
15	Tabla de arquitectura de modelos ResNet.	27
16	Bloque <i>Bottleneck</i> de ResNet.	28
17	Arquitectura de la red InceptionV3.	29
18	Módulo <i>Inception</i> con convoluciones $n \times n$ factorizadas.	30
19	Proceso de entrenamiento y selección de modelo.	32
20	Proceso de <i>fine tuning</i> propuesto.	35
21	Representaciones utilizadas en el <i>dataset</i> de LSCh.	39
22	Estructura de las carpetas del <i>dataset</i> .	40
23	Matriz de confusión de la prueba.	45
24	Matriz de confusión de mejor modelo ResNet50.	47
25	Matriz de confusión de mejor modelo InceptionV3.	49
26	Matriz de confusión de modelo sin <i>data augmentation</i> .	51
27	Matriz de confusión modelo con todas las capas descongeladas.	52

ÍNDICE DE TABLAS

1	Grilla de hiperparámetros.	33
2	Técnicas de <i>data augmentation</i> utilizadas.	33
3	<i>Callbacks</i> durante selección de modelo	34
4	Parámetros de mejor modelo VGG16 encontrado.	42
5	Parámetros de mejor modelo ResNet50 encontrado.	42
6	Parámetros de mejor modelo InceptionV3 encontrado.	42
7	Resultados de modelos tras ajuste de parámetros.	43
8	Reporte de clasificación de modelo basado en VGG16.	44
9	Reporte de clasificación de modelo basado en ResNet50.	46
10	Reporte de clasificación de modelo basado en InceptionV3.	48

11	Parámetros de mejor modelo VGG16 encontrado.	50
12	Resultados de <i>fine tuning</i> sobre la red.	52
13	Reporte de clasificación de modelo con todos los pesos descongelados.	53

INTRODUCCIÓN

Las lenguas de señas son sistemas completos y naturales de comunicación visual utilizados principalmente por personas con discapacidad auditiva en todo el mundo, permitiéndoles comunicarse tanto entre sí como con el resto de la sociedad. A diferencia de lo que se podría pensar, no existe una lengua de señas universal, de hecho, se estima que hay más de 300 lenguas de señas distintas a nivel global. Cada una de ellas es única y posee su propia gramática, sintaxis, vocabulario y alfabeto manual, lo que refleja la diversidad cultural y lingüística de las comunidades sordas en diferentes regiones y países.

Por ejemplo, la Lengua de Señas Argentina (LSA) y la Lengua de Señas Española (LSE), aunque ambas provienen de países hispanohablantes, presentan diferencias significativas en su estructura lingüística. Esto significa que una persona usuaria de la LSA puede tener dificultades para comprender la LSE, debido a las variaciones regionales propias de cada lengua de señas.

En el contexto de nuestro país, existe poco conocimiento sobre esta lengua, más encima, los sistemas públicamente disponibles actualmente se enfocan más en el área de la enseñanza que de la interpretación, por lo que su alcance es aun más limitado ya que va a depender de que una persona quiera aprender una nueva lengua.

En la presente memoria se propone el desarrollo de un modelo de una red neuronal artificial mediante el uso de técnicas de *transfer learning* para la interpretación del alfabeto de la lengua de señas Chilena (LSCh). Para esto se estudian los avances que se han desarrollado en esta área en los últimos años, se construye un nuevo conjunto de datos para el alfabeto de la LSCh y se desarrolla un modelo tomando como base otro modelo previamente existente que ha sido entrenado para cumplir con una tarea similar.

La estructura de este trabajo se compone en los siguientes capítulos:

- **Definición del problema:** Se describe el contexto del problema en Chile, el impacto que se espera que tenga la solución propuesta y se describen los objetivos que se desean cumplir en esta memoria.
- **Marco conceptual:** Se contextualiza el área de la memoria detallando conceptos importantes del aprendizaje automático junto con una sección sobre el estado del arte donde se discuten tres trabajos sobre este problema.
- **Propuesta de la solución:** Se presentan los procesos a seguir para desarrollar la solución al problema, describiendo desde los requerimientos para el nuevo conjunto de datos a desarrollar, los distintos modelos sobre los cuales vamos a trabajar, hasta por último un *ablation study* sobre el modelo desarrollado.
- **Validación de la solución:** Se presentan los resultados del desarrollo planteado en capítulo anterior, desde la estructura del conjunto de datos desarrollado, los resultados

del entrenamiento del modelo seleccionado y los resultados de los experimentos propuestos en el *ablation study*.

- **Conclusiones:** Se expone el aprendizaje obtenido de los capítulos previos, evaluando el cumplimiento de los objetivos planteados y posibles trabajos futuros en base a esta memoria.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

En los últimos años, los avances en la computación han permitido el desarrollo de sistemas más efectivos para la interpretación de lenguas de señas. Un estudio de Óscar Koller sobre el estado del arte en el reconocimiento de lenguas de señas [Koller, 2020] muestra que, entre 2015 y 2020, la cantidad de modelos publicados para la interpretación de lenguas de señas aisladas¹ casi se duplicó en comparación con el periodo 2010-2015. La mayoría de estos modelos han sido entrenados para interpretar la lengua de señas americana (ASL), destacándose su popularidad en el ámbito de la investigación. Sin embargo, es importante señalar la ausencia de modelos específicamente diseñados para la lengua de señas chilena, lo que sugiere un área de investigación poco explorada.

En Chile, con la aprobación de la ley 21.303 el año 2021 [Congreso Nacional de Chile, 2021], la lengua de señas oficialmente reconocida es la Lengua de Señas Chilena (LSCh), la cual constituye el principal medio de comunicación para muchas personas sordas en el país. La LSCh no solo es fundamental para la interacción diaria de la comunidad sorda chilena, sino que también representa un elemento clave para su inclusión social, acceso a la educación, servicios públicos y participación en la vida cívica.

A pesar de los avances en el desarrollo de modelos de reconocimiento de lenguas de señas en el ámbito internacional, existe una notable ausencia de sistemas desarrollados para la interpretación de LSCh. Esta ausencia se convierte en un problema cuando toma en consideración el tamaño de la población que podría beneficiarse de estos avances en Chile.

¹Aislada: Especifica que la data sobre la lengua de señas se compone por señas únicas y no frases u oraciones enteras

1.1. Árbol de Problema

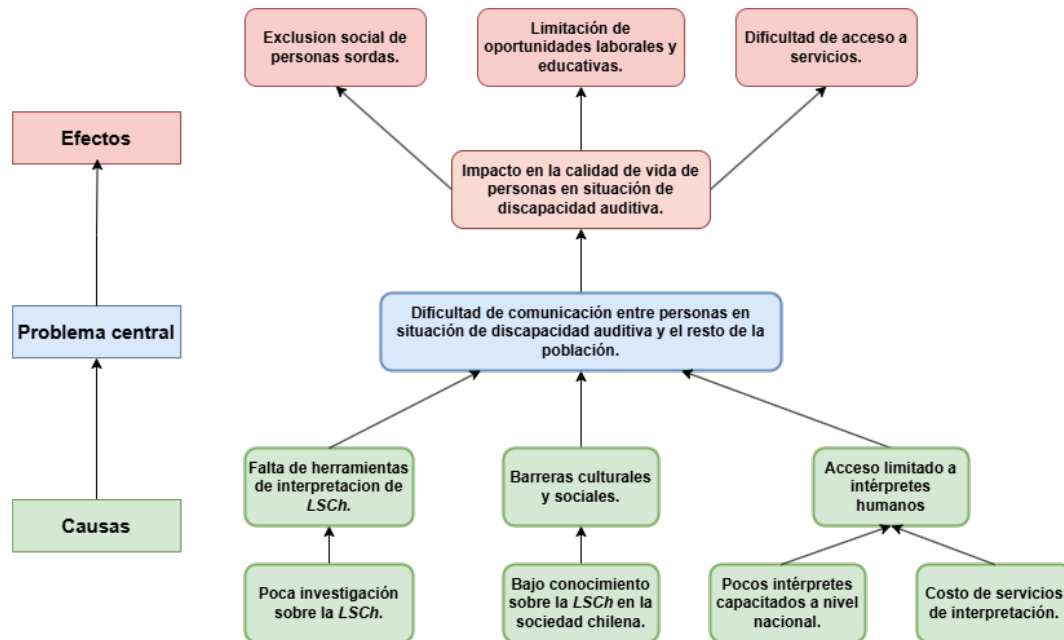


Figura 1: Árbol de problemas
. Fuente: Elaboración propia.

Es importante considerar que el problema trabajado en este escrito es considerado complejo dentro de la ingeniería. Esto se debe a múltiples factores, desde la naturaleza de la información con la que se trabaja hasta la ausencia de una solución única. Además, a medida que surgen nuevas tecnologías, el estado del arte en esta área está en constante evolución y cambio.

1.2. Impacto inicial de la solución

Según el Tercer Estudio Nacional de la Discapacidad (ENDISC) [Servicio Nacional de la Discapacidad (SENADIS), 2022] realizado el 2022; el 17.6 % de la población adulta chilena se encuentra en situación de discapacidad, de este grupo un 18 % presenta pérdida de audición de algún tipo y 4 % presenta sordera total, mientras que 14.7 % de la población NNA² en situación de discapacidad, 3.9 % presenta pérdida de audición y 0.5 % presenta sordera total. Estas condiciones limitan la capacidad para poder comunicarse de manera efectiva en contextos donde la lengua oral es la forma principal de interacción. La falta de sistemas automáticos para la interpretación de la LSCh en conjunto con el bajo conocimiento de esta lengua a nivel nacional afecta a esta población, no solo

²NNA: Niños, Niñas y Adolescentes

en el contexto de comunicación, sino que a sus oportunidades de integración en áreas cruciales como la educación, trabajo y acceso a servicios públicos. En comparación con otras lenguas de señas más estudiadas, tales como ASL, la LSCh ha recibido poca atención de la comunidad científica y tecnológica. Esto deja a un grupo importante de personas sin acceso a herramientas de comunicación que podría significar una mejora substancial en su calidad de vida e inclusión en la sociedad chilena.

No se espera que el impacto que tenga el desarrollo de esta solución solo sea un aporte a este grupo de nuestra población, sino que también aumente el conocimiento formal respecto a este problema y sea una herramienta que otros investigadores puedan utilizar para el desarrollo de proyectos futuros.

1.3. Objetivo General

Desarrollar una red neuronal para la interpretación del alfabeto manual de la lengua de señas Chilena (LSCh) mediante el uso de técnicas de *transfer learning* y *fine tuning* para adaptar un modelo pre-entrenado ya existente a esta nueva tarea.

1.4. Objetivos Específicos

- Adaptar un modelo pre-entrenado mediante métodos de *Transfer Learning* y *Fine Tuning* para la interpretación del alfabeto de la lengua de señas Chilena.
- Recolectar y pre-procesar un conjunto de datos representativo de señas del alfabeto de la lengua de señas chilena, incluyendo variaciones visuales para mejorar la generalización del modelo.
- Evaluar y optimizar el rendimiento del modelo seleccionado utilizando técnicas de *fine tuning*, ajuste de hiperparámetros y aumento de datos, con el fin de maximizar la precisión en la clasificación de las señas del alfabeto LSCh.

1.5. Alcance

Este trabajo de memoria tiene como objetivo el desarrollo de un modelo de red neuronal. Debido a la falta de disponibilidad de *datasets* sobre las señas del alfabeto de la lengua de señas chilena de buena calidad, se tuvo que expandir el alcance para incluir el desarrollo y procesamiento de un nuevo *dataset* para la elaboración de este trabajo. Debido a la necesidad de volver a iniciar el desarrollo durante varias etapas de este trabajo y por restricciones de tiempo la aplicación de métodos de *fine tuning* se limitaron a un *Ablation Study* que mide su impacto en el rendimiento del modelo desarrollado.

CAPÍTULO 2

MARCO CONCEPTUAL

A continuación se presenta el marco conceptual que fundamentara los conocimientos clave para el desarrollo del presente informe. Este marco está compuesto por los siguientes temas principales:

- Aprendizaje Automático.
- Redes Neuronales Artificiales.
- Aprendizaje Profundo.
- Redes Neuronales Convolucionales.
- Problemas de clasificación.
- *Transfer Learning*.
- *Optimización de Hiperparámetros*.
- *Ablation Studies*.
- Estado del Arte.

2.1. Aprendizaje Automático

El Aprendizaje Automático (*Machine Learning*) es un campo de la Inteligencia Artificial centrado en desarrollar algoritmos que aprenden directamente de la información (datos). Su objetivo es que los sistemas reconozcan patrones, realicen predicciones o tomen decisiones, volviéndose más precisos y eficientes a medida que analizan mayor volumen de datos, sin requerir una programación específica para cada tarea.

Tom M. Mitchell, uno de los pioneros en el área de aprendizaje automático, define este de la siguiente manera: "se dice que un sistema computacional aprende desde la experiencia para un conjunto de tareas utilizando una métrica de rendimiento si el rendimiento del sistema en estas tareas mejora con la experiencia. [Mitchell, 1997]"

En base a esto, se puede definir que un sistema de aprendizaje automático requiere de:

- **Tarea:** Es el objetivo que tiene el sistema, puede ser generar predicciones a base de datos disponibles o la clasificación de imágenes de un conjunto de datos.

- **Experiencia:** Constituye el conjunto de datos (o información) a partir del cual el algoritmo aprende. Es a través de esta experiencia que el sistema descubre patrones y ajusta su funcionamiento.
- **Métricas de rendimiento:** Es el criterio utilizado para medir la eficacia del programa al ejecutar la tarea. Esta métrica permite determinar si el aprendizaje está ocurriendo y si el sistema está mejorando. Ejemplos de estas serían la **precisión**, **recall** y **F1-Score**.

El aprendizaje automático se puede categorizar en distintos tipos de aprendizaje. [Sarker, 2021] presenta las siguientes 4 categorías de aprendizaje automático:

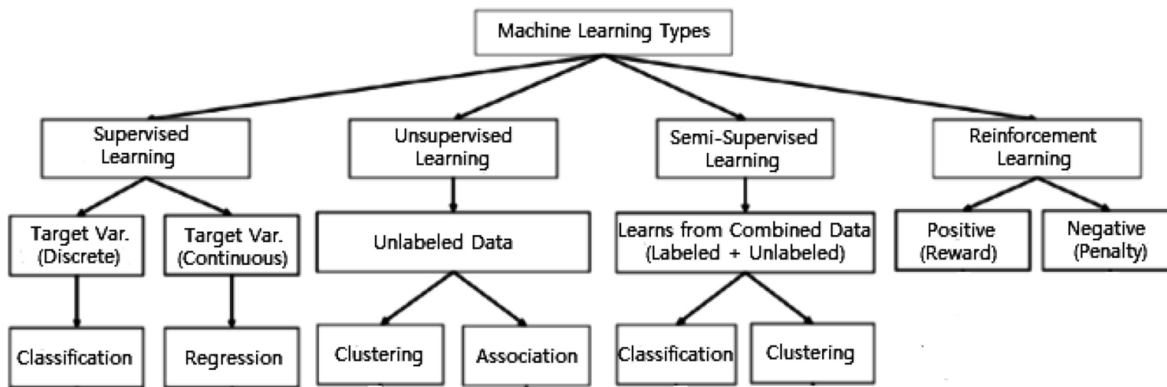


Figura 2: Tipos de aprendizaje automático.

Fuente: [Sarker, 2021]

2.1.1. Aprendizaje Supervisado

El aprendizaje supervisado o *supervised learning* se caracteriza por el entrenamiento con datos que ya están etiquetados, es decir, se le proporciona la entrada y la salida correcta esperada. A partir de estos ejemplos, aprende a mapear entradas a salidas para hacer predicciones en datos nuevos y no vistos.

2.1.2. Aprendizaje no Supervisado

El aprendizaje no supervisado o *unsupervised learning*, a diferencia del aprendizaje supervisado, se caracteriza por el uso de datos no etiquetados para entrenar al sistema a cumplir con alguna tarea específica. Esta restricción genera que el sistema tenga que aprender patrones, estructuras o relaciones ocultas entre los datos por sí mismo sin la necesidad de interferencia humana, lo cual hace este tipo de aprendizaje ideal para desarrollar análisis exploratorio sobre conjuntos de datos y para problemas de reconocimiento de imágenes y patrones.

2.1.3. Aprendizaje Semi-Supervisado

El aprendizaje semi-supervisado o *semi supervised* es un método que combina el aprendizaje supervisado con el no supervisado y se caracteriza por el uso de datos etiquetados y no etiquetados para el entrenamiento de un sistema. Este tipo de aprendizaje tiende a ser útil cuando no se dispone de suficiente data etiquetada o la obtención de esta es prohibitivamente difícil o costoso, pero se tiene acceso a grandes cantidades de datos no etiquetados. El sistema aprende de los datos etiquetados y luego utiliza esa información para inferir o predecir etiquetas para los datos sin etiquetar, refinando continuamente su conocimiento.

2.1.4. Aprendizaje reforzado

El aprendizaje por refuerzo (*Reinforced learning*) es un tipo de aprendizaje que le permite a una máquina decidir de forma automática el comportamiento óptimo que le permitirá mejorar su rendimiento para una tarea especificada. El proceso que le permite a una máquina tomar estas decisiones se basa en un sistema de recompensas y penalizaciones que le entregan o quitan puntos dependiendo si la acción tomada mejora o empeora su rendimiento para la tarea.

2.2. Redes Neuronales Artificiales

Las redes neuronales artificiales (RNA) son sistemas computacionales inspirados en la estructura y el funcionamiento del cerebro humano. Estos sistemas se componen de conjuntos de neuronas artificiales conectadas entre sí en estructuras denominadas como capas. Toda red neuronal consiste en 3 o más capas neuronales interconectadas que transmiten información a la siguiente capa; esta información tiene asociada a sí un **peso** que determina su importancia respecto a la tarea que se desea cumplir. Al igual que el cerebro humano, estas redes son capaces de aprender y identificar patrones dentro de la data y ajustar sus conexiones internas con el fin de mejorar su rendimiento para una tarea específica.

Toda red neuronal está compuesta por los siguientes 3 componentes (Ver figura 3):

- **Capa de Entrada:** También conocida como la *input layer*, es la primera capa de toda red neuronal. Esta se encarga de recibir los datos de entrada y propagarlos a través de la red. En esta capa no se realizan ningún tipo de cálculo sobre la información recibida y el número de neuronas que conforman esta capa dependerán de la naturaleza de los datos de entrada, dado a que cada neurona representa una característica o variable del conjunto de entrada.
- **Capa oculta:** También conocida como *hidden layer*, es una o varias capas de neuronas (o nodos) que se encuentran entre la capa de entrada y la capa de salida. Su función

principal es procesar y transformar la información que reciben de la capa anterior. A diferencia de las capas de entrada y salida (que interactúan directamente con los datos externos), las capas ocultas realizan cálculos internos complejos para encontrar patrones y características relevantes en los datos.

- **Capa de Salida:** También se conoce como *output layer*, es la última capa de una red neuronal artificial y su rol es proporcionar el resultado de los datos después de haber sido procesados por el resto de la red. Esta capa cumple el rol de la tarea que se le asigna, tal como puede ser generar predicciones o clasificar data, y la cantidad de neuronas que componen a esta red están determinadas por el tipo de tarea que se le asigne.

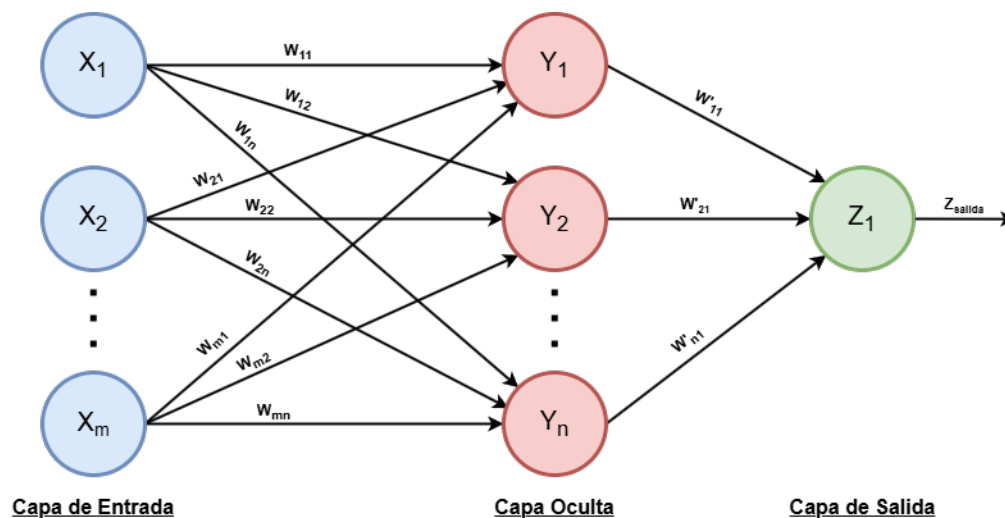


Figura 3: Red neuronal artificial básica de tres capas.

Fuente: Elaboración Propia.

Una red no es simplemente capaz de aprender si lo único que hace es pasar los valores de un nodo a otro dado que los patrones o características de la información no son igualmente importantes. Es por esto que para determinar la importancia de alguna característica las redes hacen uso de **pesos, funciones de activación y optimizadores**.

- **Pesos:** Los pesos de una red, en su forma más simple, representan la importancia de algún patrón o característica extraída. Mientras más alto es su valor, más importante o relevante es la característica para la tarea específica.
- **Funciones de activación:** Una función de activación es una operación matemática no lineal que se aplica a la salida de cada neurona en una red neuronal. Determina si una neurona debería activarse o no y la intensidad de esa activación, ejemplos de estas funciones serían *ReLU*, *Softmax* y *Tanh*.

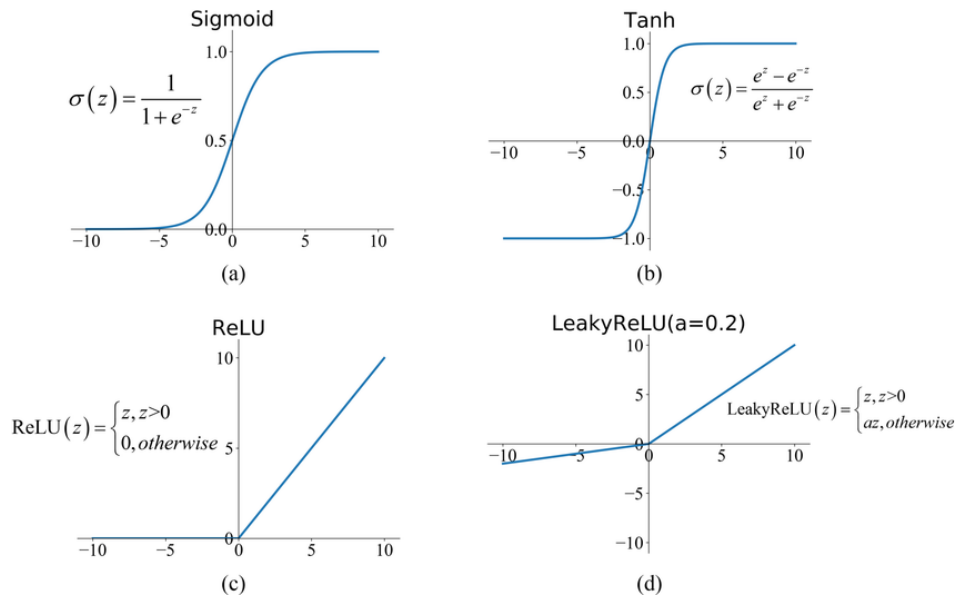


Figura 4: Ejemplos de funciones de activación comúnmente usadas.

Fuente: [Feng et al., 2019]

- **Optimizadores:** Son algoritmos que ajustan los pesos y sesgos del modelo. Su función principal es minimizar la función de pérdida (el error de la red) durante el entrenamiento, determinando la dirección y magnitud de cada actualización para mejorar el aprendizaje de la red.

2.3. Aprendizaje Profundo (Deep Learning)

El Aprendizaje Profundo (*Deep Learning*) es una subrama del Aprendizaje Automático que se basa en el desarrollo de redes neuronales con múltiples capas ocultas que le permite crear una jerarquía de conceptos, donde cada uno se define en base a su relación con conceptos más simples. Esencialmente, estas redes aprenden a identificar patrones complejos basándose en patrones más simples [Goodfellow et al., 2016]. En la figura 5 se puede ver la diferencia entre una RNN tradicional (a) y una red neuronal profunda (b).

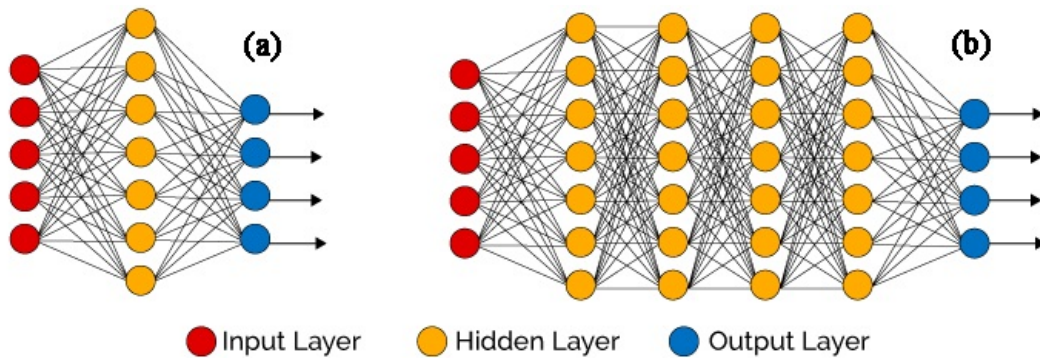


Figura 5: Ejemplo de red neuronal tradicional en comparación una red neuronal profunda.

Fuente: [Azawii et al., 2019]

Este tipo de estructura se ve presente en varios tipos de arquitecturas de redes neuronales, tales como en las siguientes:

- Redes Neuronales Convolucionales.
- Redes Neuronales Recurrentes.
- Modelos de Difusión.
- Redes Generativas Antagónicas.
- *Transformers*.
- Autocodificadores.

2.4. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales (CNN) son un tipo de red neuronal artificial principalmente compuesta por 3 capas, las capas convolucionales, capas de *pooling* y capas completamente conectadas. Estos tipos de redes se caracterizan por tener buen rendimiento procesando información visual, tal como imágenes y video, tanto como señales de audio y voz. A continuación, se explica cómo funcionan las capas de una red convolución y el rol que cumplen dentro de la red:

2.4.1. Capas de Convolución

Las capas convolucionales son el foco de las CNN, una convolución es una operacional matricial que consiste en la aplicación de un filtro (*kernel*) sobre la matriz de píxeles de una imagen

para poder enfatizar ciertos rasgos, tales como pueden ser los bordes, líneas verticales y horizontales, entre otros. La matriz resultante de esta operación se denomina *feature map*, y su dimensión va a estar determinada por el tamaño del *kernel*, el *stride* seleccionado y el tipo de *padding* aplicado como se explica a continuación:

- **Kernel** : Un kernel es una matriz que se utiliza para detectar patrones dentro de una imagen, desde características simples a más complejas a lo largo de la red.
- **Stride** : El *kernel* se desliza a través de la matriz una cierta cantidad de píxeles a la vez, el *stride* es el parámetro que determina cuantos píxeles se desplaza por iteración.
- **Padding** : Al aplicar una convolución se está reduciendo la dimensión de la matriz resultante, el *padding* permite mantener la dimensionalidad al rellenar el espacio vacío con un valor, generalmente este es 0.
- **Feature Map** : Es la matriz resultante al aplicarle un filtro (*kernel*) a una imagen, su dimensionalidad tiende a ser menor que antes de ser procesada y esta resalta la característica detectada por el filtro.

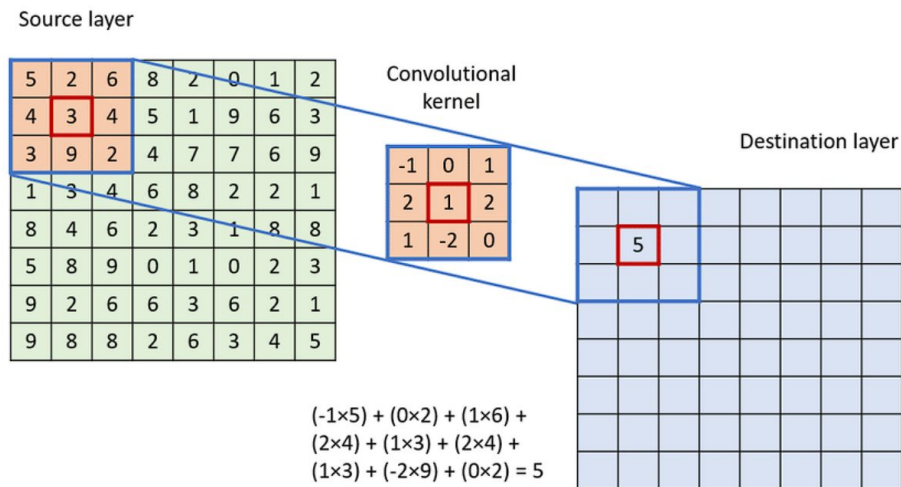


Figura 6: Ejemplo de una operación convolucional sobre una matriz

Fuente: [Podareanu et al., 2019]

2.4.2. Capas de Agrupación

Las capas de agrupación (*pooling*) se usan en las CNN para reducir el tamaño de los mapas de características. Esto disminuye la cantidad de parámetros de la red y ayuda a evitar el sobreajuste (*overfitting*).

Existen dos tipos de funciones de *pooling*:

- **Max Pooling:** Selecciona el valor más alto de una región, destacando las características más fuertes y haciendo la red robusta a pequeñas variaciones.
- **Average Pooling:** Calcula el promedio de una región, lo que suaviza la información y mantiene una representación más general.

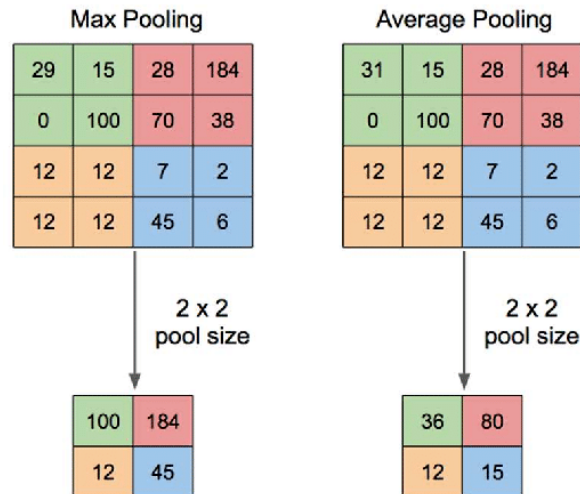


Figura 7: Ejemplo de *Max Pooling* y *Average Pooling*

Fuente: [Yani et al., 2019]

2.4.3. Capas completamente conectadas

Son las últimas capas de una *CNN* y realizan la tarea de clasificación en base a las características extraídas en las capas previas. Los nodos de esta capa siempre van a estar directamente conectados con un nodo de la capa anterior. A diferencia de las capas convolucionales y de agrupación (*pooling*) que utilizan *ReLU* como su función de activación, esta capa suele usar una función de activación *softmax* para clasificar adecuadamente las entradas y generar una probabilidad entre 0 y 1.

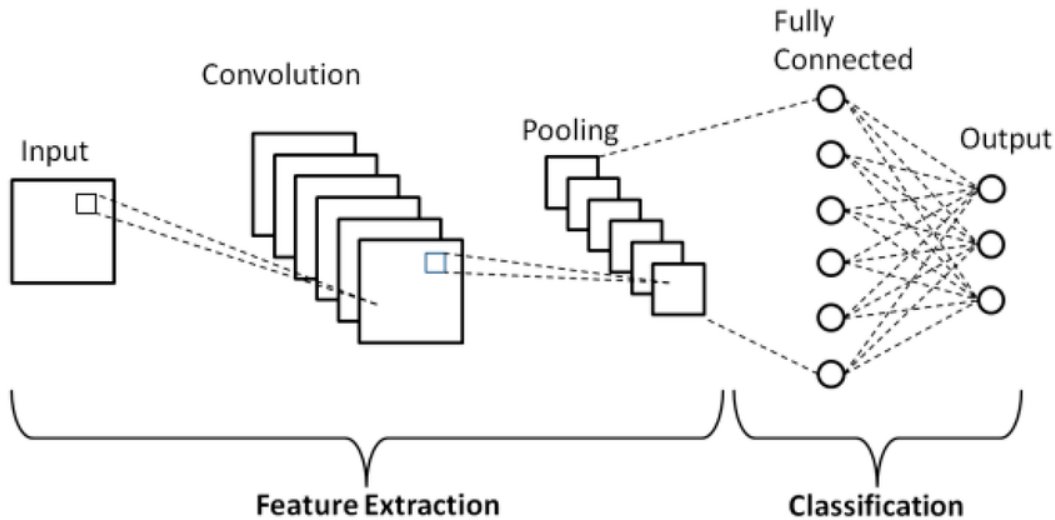


Figura 8: Diagrama de una red neuronal convolucional básica

Fuente: [Phung y Rhee, 2019]

2.5. Problemas de Clasificación

Los problemas de clasificación en redes neuronales consisten en categorizar datos de entrada en una de varias clases predefinidas. La red aprende a identificar patrones para asignar correctamente cada dato a su etiqueta correspondiente.

Existen distintos tipos de problemas de clasificación, los más clásicos son los problemas de clasificación binaria, en donde se busca clasificar la data de entrada a una de dos clases mutuamente excluyentes, por ejemplo si un correo es *spam* o no.

Pero este no es el único tipo de problema de clasificación disponible, el otro tipo de problema y el enfoque principal de este trabajo son los problemas de clasificación de múltiples clases, donde el objetivo es determinar a cual de las N clases corresponde un cierto dato de entrada, donde a diferencia de la clasificación binaria es posible que un dato corresponda a más de una clase simultáneamente. Ejemplos de este tipo de problemas incluyen el reconocimiento facial, la identificación de objetos y la interpretación de imágenes médicas [Simonyan y Zisserman, 2015].

2.5.1. Métricas de evaluación

Clasificar es el objetivo principal de este tipo de problemas, pero para poder verificar el rendimiento de un modelo clasificador es necesario entender que son las métricas de evaluación y que nos dicen cada una.

- **Matriz de confusión:** Es una tabla que resume el rendimiento de un modelo de clasificación, muestra donde el modelo hizo predicciones correctas e incorrectas respecto al valor real del resultado.
- **Precisión:** Corresponde a la proporción de predicciones correctas sobre el total de predicciones.
- **Recall:** Métrica que mide la proporción de instancias positivas reales que fueron correctamente identificadas por el modelo.
- **F1-Score :** Es la media armónica de la **Precisión** y el **Recall**. Es un equilibrio entre ambas métricas y nos permite evaluar el rendimiento general del clasificador en situaciones donde los falsos positivos y los falsos negativos son importantes.

Las métricas previamente mencionadas nos permiten evaluar el rendimiento final de un modelo tras ser entrenado, ¿pero qué sucede si el modelo presenta una precisión muy alta, pero una vez que se pruebe con data nueva la precisión disminuye drásticamente? En situaciones como estas nos estamos enfrentando a lo que se llama *Overfitting*. La definición del cual y de *Underfitting* se explicaran en el siguiente sección.

2.5.2. *Underfitting and Overfitting*

Tal como se mencionó anteriormente, un buen resultado de precisión durante el entrenamiento no es suficiente para poder determinar si un modelo va a ser bueno o no, para esto es necesario saber si el modelo esta siendo afectado por sobreajuste (*overfitting*) o subajuste (*underfitting*)

- **Overfitting:** Es cuando un modelo no es capaz de generalizar y se ajusta demasiado cerca de la data de entrenamiento, por lo general se dice que se memoriza esta data en vez de aprender de ella. Esto se ve cuando el modelo es capaz de predecir con alta precisión los resultados de la data de entrada pero pierde esta precisión cuando se enfrenta a data nunca antes vista.
- **Underfitting:** En el otro caso, cuando un modelo no es capaz de generar relaciones entre la data de entrada y la de salida, se dice que esta haciendo *underfitting*, lo que se puede ver cuando las predicciones de dicho modelo son imprecisas tanto durante el entrenamiento como con data nueva.

En la figura 9 se pueden ver 3 grafos que permiten visualizar los casos de subajuste, un ajuste adecuado y de sobreajuste de los datos.

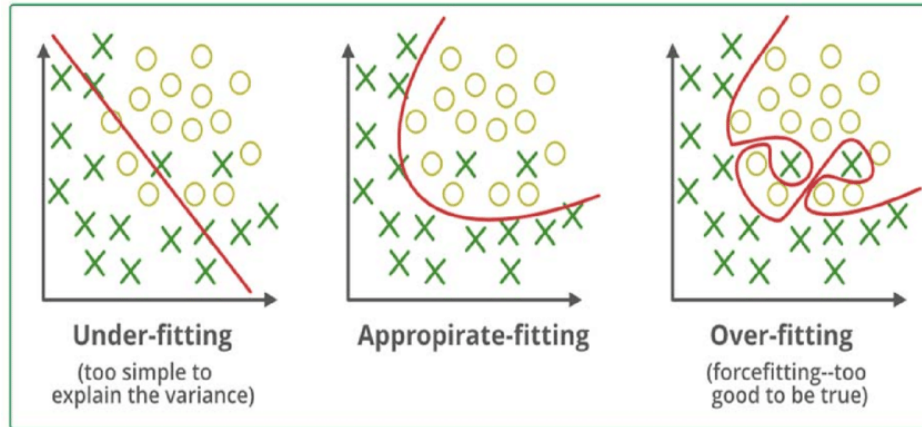


Figura 9: Visualización de subajuste y sobreajuste

Fuente: [Kolluri et al., 2020]

Estos dos fenómenos suceden bajo varias distintas condiciones, entre estas están la calidad de los datos de entrada, donde esta data puede tener demasiado ruido para poder aprender patrones verdaderamente importantes, o que el modelo se entrene con data que tenga información que no estaría disponible en los casos de prueba reales. Otro de los factores influyentes sobre si un modelo tiende a sobreajustar o subajustar es la cantidad de tiempo por el que se entrena; o para ser más específico, la cantidad de *epochs*³ por el que se entrena, si el numero es muy bajo el modelo puede tender a subajustar, por el contrario, si el número es muy grande, el modelo puede tender al sobreajuste. Es por esto que la calidad de los datos de un *dataset* y un balance entre entrenar poco y entrenar demasiado son factores cruciales que afectan el rendimiento final de un modelo.

2.6. Transfer Learning

2.6.1. Feature Extracion

Feature extraction es un proceso que consiste en tomar un modelo que ha sido previamente entrenado para una tarea en particular, congelar los pesos que ha adquirido a través del entrenamiento, sustituir las capas finales por nuevas capas finales y entrenarlo para la tarea deseada; en efecto solo se están entrenando las capas finales de el nuevo modelo.

En la figura 10 se presenta un ejemplo de el uso de *feature extraction* para la detección de COVID-19.

³Epoch: Corresponde a una pasada completa por el conjunto de datos de entrenamiento.

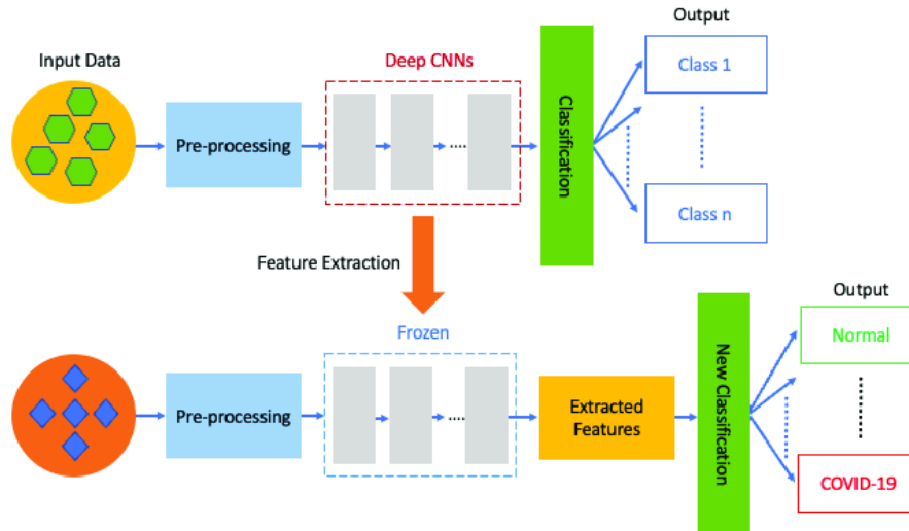


Figura 10: Ejemplo de *feature extraction* en *transfer learning*.

Fuente: [Alahmari et al., 2022]

2.6.2. Fine Tuning

Fine tuning es un proceso de entrenamiento que se separa en dos pasos [Raghu y Schmidt, 2020], el primero de estos se denomina el paso de **pre-entrenamiento** o *pretraining step*, el cual consiste en seleccionar una red o modelo con parámetros inicializados aleatoriamente y entrenarla con un conjunto de datos amplio y general para cumplir con una tarea. Un conjunto de datos comúnmente usado para el esta fase es *ImageNet* [Deng et al., 2009], el cual consiste de más de 14 millones de imágenes para entrenar una red para clasificación de imágenes, detección de objetos y localización de objetos.

Una vez que se ha completado el pre-entrenamiento se tendrá un modelo que ha aprendido relaciones que nos serán útiles para el siguiente paso, el cual se le llama ajuste fino (o *fine-tuning*), en este paso se utiliza el modelo previamente entrenado y se le vuelve a entrenar para aprender el conjunto de datos que queremos utilizar.

En la figura 11 se ilustra el proceso de *fine tuning* para entrenar un modelo.

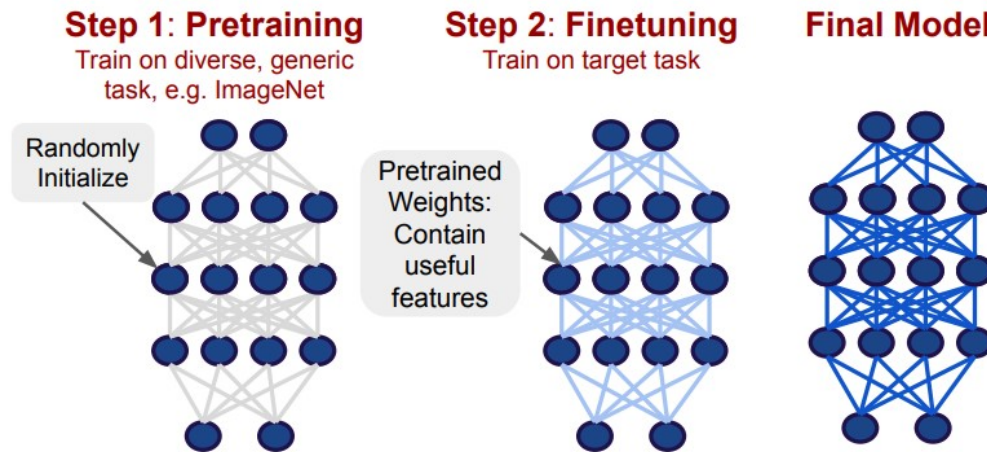


Figura 11: Ilustración de proceso de *fine tuning*

Fuente: [Szegedy et al., 2014]

2.6.3. Data Augmentation

En la gran mayoría de los casos *transfer learning* se utiliza dado a que la cantidad de datos que tenemos disponibles no serian suficientes para entrenar bien a un modelo, adicionalmente si los datos a los cuales tenemos acceso no tienen suficiente variedad entonces el modelo final puede tender al sobreajuste. En situaciones como estas podemos aplicar aumento de datos (*data augmentation*).

El aumento de datos consiste en utilizar data pre-existente para generar nueva data al aplicarle modificaciones, tal como desenfocando la imagen, aplicando rotación o cortando la imagen, aplicándole cambios de escalas de color, brillo o hasta contraste (Ver figura 12).

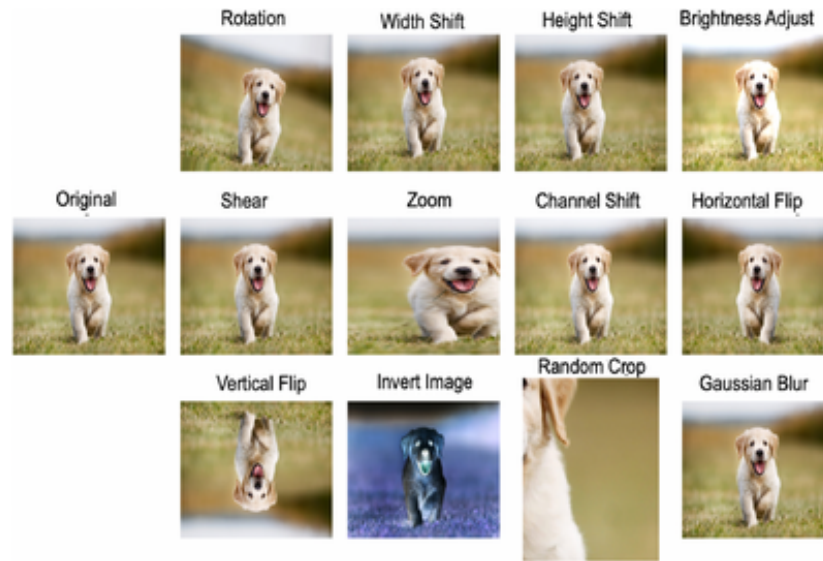


Figura 12: Ejemplos de técnicas de aumento de imágenes.

Fuente: [Singh et al., 2025]

Es importante que estos métodos se apliquen solamente al conjunto de datos que se usarán para entrenar, no sobre el conjunto de datos en su totalidad ya que esto puede introducir *data leakage*⁴ al conjunto de prueba.

2.7. Optimización de Hiperparámetros

Es el proceso de encontrar la mejor configuración de los parámetros externos de un modelo de aprendizaje automático (o red neuronal) para que este logre el mejor rendimiento posible. A diferencia de los pesos y sesgos, que son aprendidos por el modelo durante el entrenamiento, los hiperparámetros deben ser establecidos antes de que comience el entrenamiento.

Métodos comunes de optimización:

- **Búsqueda de Grilla:** Prueba sistemáticamente todas las combinaciones posibles de hiperparámetros dentro de un rango predefinido.
- **Búsqueda Aleatoria:** Selecciona aleatoriamente combinaciones de hiperparámetros, a menudo más eficiente que la búsqueda en grilla para encontrar buenos resultados.

⁴Data Leakage: Suceso que ocurre cuando un modelo incorpora información durante su entrenamiento a la cual no tendría acceso a la hora de generar predicciones.

- **Successive Halving:** Busca identificar rápidamente las configuraciones prometedoras y descartar las menos eficaces de forma temprana en el proceso de entrenamiento. Es la base de algoritmos de optimización avanzados como *Hyperband* [Li et al., 2018].

2.8. Ablation Studies

El término *Ablation* proviene del campo de la medicina, donde se define como el proceso de remover o destruir una parte, tejido o función del cuerpo. En el estudio de las redes neuronales, su utilización fue propuesta inicialmente por Allen Newell para describir la acción de remover o reemplazar componentes de sistemas neuronales y de esta manera, medir el efecto que esta acción tiene en el rendimiento del sistema.

En el área de las redes neuronales, particularmente las redes profundas, la realización de *ablation studies* es una herramienta fundamental para el análisis de nuestros resultados. Nos permite determinar la utilidad y el valor de ciertos componentes en el rendimiento final de nuestro sistema. De este modo, se puede identificar qué parámetros o componentes son cruciales, cuáles son útiles y cuáles podrían ser redundantes. Esta información es de gran importancia para priorizar procesos y guiar el trabajo futuro en la optimización y diseño de modelos.

2.9. Estado del arte

2.9.1. Reconocimiento en Tiempo Real del Alfabeto de Lengua de Señas Chilena Empleando Aprendizaje Automático [Benítez y Catril, 2022]

En el trabajo de título *Reconocimiento en Tiempo Real del Alfabeto de Lengua de Señas Chilena Empleando Aprendizaje Automático* [Benítez y Catril, 2022] de los estudiantes de la universidad del Bio-Bio, se crea una aplicación para la interpretación del alfabeto de la lengua de señas chilena en tiempo real a través del uso de una red YOLO V5 utilizando transferencia de aprendizaje.

Los autores tuvieron que desarrollar un conjunto de datos del alfabeto de la lengua de señas chilena desde cero, debido a la ausencia de *datasets* de esta área disponibles públicamente. El modelo que desarrollaron utilizó como base un modelo YOLO V5 con pesos de COCO y una nueva cabeza clasificadora de 26 nodos correspondientes a las 26 clases de la data.

El modelo fue luego exportado en formato *tfLite*⁵, el cual utilizan en una aplicación móvil para *Android* utilizando como base un proyecto de fuente abierta disponible en *GitHub*, esta aplicación les permite utilizar la cámara de un dispositivo móvil para capturar una representación

⁵tfLite: TensorFlow Lite.

de la *LSCh* y clasificarla con el modelo que desarrollaron.

El modelo que desarrollaron presenta buenos resultados en el conjunto de datos de prueba, obteniendo una precisión de 91 % sobre estos datos, pero presenta problemas a la hora de predecir las letras I, L, N, Q y W, donde los resultados presentados por los autores para estas son de 53, 3 %, 78, 5 %, 62, 5 %, 78, 5 %, 71, 2 % respectivamente para las señas.

Los autores concluyen que el entrenar un modelo de clasificación de imágenes con métodos como *transfer learning* simplifica el proceso de entrenamiento de un modelo, pero la eficacia de este modelo va a depender principalmente en la calidad del conjunto de datos, sea esto por el tamaño reducido de este, poca variedad visual o por casos de muchas señas siendo representadas por la misma persona.

2.9.2. *American Sign Language Recognition Based on Transfer Learning Algorithms* [Mohsin et al., 2023]

Este trabajo estudia la efectividad del uso de sistemas de aprendizaje profundo mediante *transfer learning* para la identificación e interpretación de las representaciones de letras y números que componen al alfabeto de ASL.

Para esto utilizan un *dataset* de ASL del *Modified National Institute of Standards and Technology (MNIST)* y lo utilizan para entrenar y hacer *fine tuning* sobre 4 modelos pre-existentes, para adaptarlos a la interpretación de esta nueva tarea. Los modelos que seleccionaron fueron *VGG16*, *InceptionV3*, *ResNet50* y *MobileNetV2*, estos modelos fueron entrenados, ajustados y finalmente los evalúan y comparan sus rendimientos.

Con la data que ofrecen los investigadores, el modelo que presentó las mejores métricas de rendimiento fue *InceptionV3*, el cual alcanza un *accuracy* del 96 % y un *F1 Score* de 95 %, con el modelo *VGG16* obteniendo segundo lugar.

Los investigadores concluyen que la utilización de *transfer learning* sobre modelos de aprendizaje profundo previamente entrenados, es una buena estrategia para el entrenamiento de sistemas para la tarea de reconocimiento de lenguas de señas, siempre y cuando la calidad del conjunto de datos utilizado para este proceso sea buena, ya que los investigadores concluyen que, más que el modelo, la calidad de los datos tienen la mayor influencia en los resultados finales.

2.9.3. *Isolated sign language recognition using hidden transfer learning* [Mia et al., 2023]

En este trabajo los investigadores proponen un sistema de detección de gestos de manos y de interpretación de lenguas de señas mediante el uso de una red convolucional profunda

que les permita interpretar las letras y números del alfabeto de ASL más ciertas palabras.

Para el desarrollo del sistema proponen 3 redes, la primera de estas redes es una red convolucional profunda desarrollada por los investigadores, la segunda de estas es la red VGG16 y la última de estas redes esta construida utilizando como *feature extractor* a la red VGG16 con sus pesos congelados a la que se le agrega la red convolucional profunda desarrollada para la clasificación final.

La red convolucional profunda que se propone consiste de 4 capas convolucionales y 2 capas *flatten*, utilizando *kernels* de tamaño 2x2 y *ReLU* como su función de activación.

Los resultados muestran que la red profunda que utiliza *transfer learning* con VGG16 obtiene un *accuracy* de 0.9505 en la data de entrenamiento y a pesar de la diferencia alta entre la perdida en el entrenamiento y la validación el modelo resultante, presenta muy buenos resultados para data nueva. Los otros modelos lograron resultados de 0.9447 con la red inicial y 0.9113 con VGG16. Los investigadores concluyen que la utilización de herramientas como *transfer learning* permiten mejorar la capacidad de redes para problemas como la interpretación de lenguas de señas.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

En este capítulo se presenta la estructura seguida para el desarrollo de la solución propuesta, la cual tal como se menciona en el capítulo 1, consiste en el desarrollo de un modelo de red neuronal aplicando técnicas de *transfer learning* sobre un modelo pre-entrenado para la clasificación de imágenes de señas correspondientes al alfabeto de lengua de señas chilena.

La propuesta de la solución se va a separar en 3 fases, dentro de las cuales se presentará tanto del proceso de desarrollo planteado y el razonamiento tras las decisiones tomadas. Las fases de este capítulo serán las siguientes:

1. Desarrollo de *dataset*.
2. Entrenamiento y selección de modelo.
3. *Ablation Study* propuesto.

3.1. Desarrollo de *dataset*

En el proceso de desarrollo de el trabajo presente se investigó sobre la disponibilidad de *datasets* del alfabeto de la lengua de señas chilena. Plataformas como RoboFlow⁶ contienen proyectos públicamente disponibles con *datasets* modelados para la lengua de señas chilena. A pesar de que los *datasets* disponibles se ven prometedores, será necesario crear un nuevo *dataset* manualmente.

El alfabeto de la LSCh esta compuesto por 27 señas correspondientes a cada letra del alfabeto chileno/español, estas representaciones se caracterizan por los siguiente:

- **Movimiento:** Las letras G, J, Ñ, S, X, Z en el alfabeto de la LSCh son representaciones que requieren de que el emisor mueva su mano.
- **Estáticas:** Las 21 señas restantes no requieren de hacer un movimiento con la mano, y de estas 19 representaciones se comparten con el alfabeto ASL.
- **Variaciones:** No obstante, las representaciones de las letras no son únicas, existen variaciones de las representaciones que son consideradas válidas, por ejemplo, la letra Y tiene dos representaciones, una que orienta el dorso de la mano al receptor y otra que orienta la palma.

⁶RoboFlow: <https://roboflow.com/>



Figura 13: Distintas representaciones para la letra Y en la LSCh

Fuente: Elaboración Propia.

En base a estas características se plantea desarrollar un nuevo *dataset* que cumpla con los siguientes requerimientos:

1. Las representaciones que utilizan movimiento van a ser representadas con la posición final de la mano al hacer el movimiento. Esto se debe a que se comparten más posiciones iniciales con otras letras que las posiciones finales.
2. Para el enfoque de este trabajo se va a optar por no incluir la letra Ñ en nuestras representaciones, esto se debe a que en su movimiento la posición inicial de esta es la misma que de la letra N, y su posición final es demasiado cercana a la representación de la letra H.
3. Para las representaciones que se pueden hacer con el dorso o palma de la mano orientada hacia el receptor, se optó por utilizar una representación que muestre la palma de la mano; esta representación le permitirá al modelo que se desea entrenar poder identificar más diferencias entre representaciones similares.
4. Para señas con movimiento que tienen la misma posición final se optó por aplicar una leve exageración a su movimiento permitiendo distinguirlas entre si; este es el caso particular de la representación de las letras S y X.

3.1.1. Pre-procesamiento de datos

Se propone la utilización de herramientas que permitan detectar una mano en una imagen, aplicarle un *bounding box* y recortar la imagen para solamente mostrar la mano de la persona. Para esto se utilizarán las librerías de *python*⁷: *mediapipe*⁸ y *OpenCV*⁹ para la detección y procesamiento de las representaciones.

⁷Python: <https://www.python.org/>

⁸MediaPipe: <https://pypi.org/project/mediapipe/>

⁹OpenCV: <https://opencv.org/>

3.2. Entrenamiento de modelo

3.2.1. Selección de modelo base candidato

■ VGG16

VGG16 [Simonyan y Zisserman, 2015] es una red neuronal profunda utilizando una arquitectura de red convolucional propuesta por Simonyan y Zisserman, miembros del equipo *Visual Geometry Group* de la universidad de Oxford. Este modelo se destacó al obtener el primer lugar en las categorías de localización y clasificación de la competencia *ILSVRC* el año 2014 [ImageNet, 2014].

Arquitectura

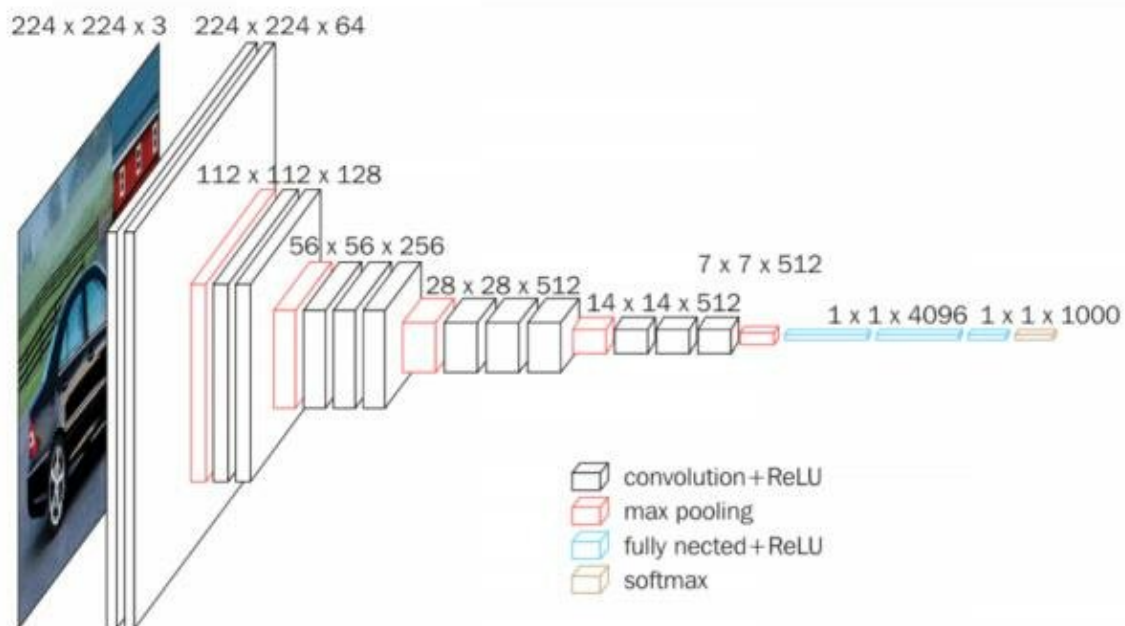


Figura 14: Arquitectura de red VGG16.

Fuente: [Kamal y Ez-Zahraouy, 2023].

Esta red se diferencia en el hecho de que es bastante simple, siendo esta compuesta por una serie de bloques convolucionales, compuestas por 13 capas convolucionales y 5 capas de agrupación totales, y tres capas completamente conectadas en la cabeza del modelo.

Su simpleza proviene de la estructura de los bloques convolucionales, utilizando 2 a 3 capas convolucionales con filtros de 3x3 agrupados y por último una capa de agrupación de tipo *MaxPooling* con un *stride* de 2. Al acoplar las capas convolucionales le permiten obtener el mismo campo receptivo que al utilizar una capa convolucional con un filtro más grande pero requiriendo menos parámetros, por lo que los bloques

de 2 capas generan el mismo campo receptivo que uno de una sola capa con un filtro 5x5, y similarmente los bloques de 3 capas generan el mismo campo que un filtro de 7x7.

Este modelo y su versión de 19 capas (VGG19) han sido utilizados en varias áreas de desarrollo, desde la detección y clasificación de tumores [Sajjad *et al.*, 2019], sistemas de reconocimiento facial [Sun *et al.*, 2015] hasta problemas de identificación y clasificación de lenguas de señas [Fowley y Ventresque, 2022]. Por estas razones fue que optamos por escoger VGG16 como uno de los modelos base para aplicar *transfer learning*.

■ ResNet50

ResNet50 [He et al., 2015] es una de las variaciones de la familia de redes residuales desarrolladas por un equipo de investigadores de *Microsoft Research Asia*. La familia de redes ResNet son redes convolucionales cuyo objetivo es mitigar el problema del gradiente desvaneciente en redes profundas, permitiendo que se puedan utilizar las ventajas de las redes profundas mitigando sus desventajas.

Arquitectura

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 15: Tabla de arquitectura de modelos ResNet.

Fuente: [He et al., 2015].

Estas redes se basan en la utilización de componentes llamados *Bloques Residuales*, estos bloques permiten que información importante aprendida durante etapas previas de la red se puedan propagar fácilmente. A continuación se va a entrar en detalle sobre la estructura de estas capas/bloques

Bottleneck Block

Los bloques *bottleneck* son la estructura que le permite al modelo ResNet su capacidad de poder utilizar su profundidad mientras que se mitigan las desventajas de las redes profundas. La manera que hace esto es a través de una combinación de extracción de características, reducción de dimensionalidad y una conexión *skip* que permite que la salida de una capa previa se agregue a otra capa posterior en la red.

A continuación se presentará un diagrama que permite visualizar la estructura de uno de estos bloques y se explicará que hace cada componente.

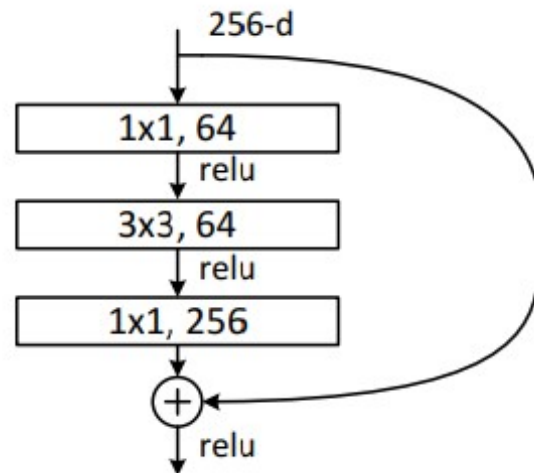


Figura 16: Bloque *Bottleneck* de ResNet.

Fuente: [He et al., 2015].

En el caso de *ResNet50*, los bloques *bottleneck* se componen de tres operaciones convolucionales, una conexión *skip* y por último, una función que agrega los valores de la conexión *skip* y de la data procesada por las capas convolucionales. Estas se describen a continuación:

Primera capa (1x1, 64): Reduce la cantidad de canales de la data de *input* y aplica compresión manteniendo la integridad de la información.

Segunda capa (3x3, 64): Capa que utiliza un filtro de 3x3 para extraer características y relaciones de la información.

Tercera capa (1x1, 256): Vuelve a restaurar la dimensión original de la data para que se pueda sumar con la data de la *skip connection* debido a que tienen que tener la misma dimensión para poder llevar a cabo esta operación.

Este modelo ha logrado producir resultados bastante buenos y desde su introducción obtuvo primer lugar en varias categorías de la competencia *ILSVRC 2015* [ImageNet, 2015]. Ha sido utilizado en varios problemas de clasificación de imágenes, *computer vision* y en problemas de interpretación/clasificación de lenguas de señas [Abu-Jamie y Abu-Naser, 2022] [Mohsin et al., 2023].

■ InceptionV3

InceptionV3 [Szegedy et al., 2015] es la tercera versión de la familia de CNN's *InceptionNet* desarrolladas por Google; estas redes se caracterizan por la utilización de una estructura llamada *inception module*, la cual va a ser explicada a mayor detalle a continuación.

La tercera versión de esta familia de redes se caracteriza principalmente por el uso de las siguientes mejoras: optimizador *RMSprop*, convoluciones factorizadas, clasificadores auxiliares y *label smoothing*. Al igual que el módulo *inception* estos van a ser discutidos a continuación.

Arquitectura

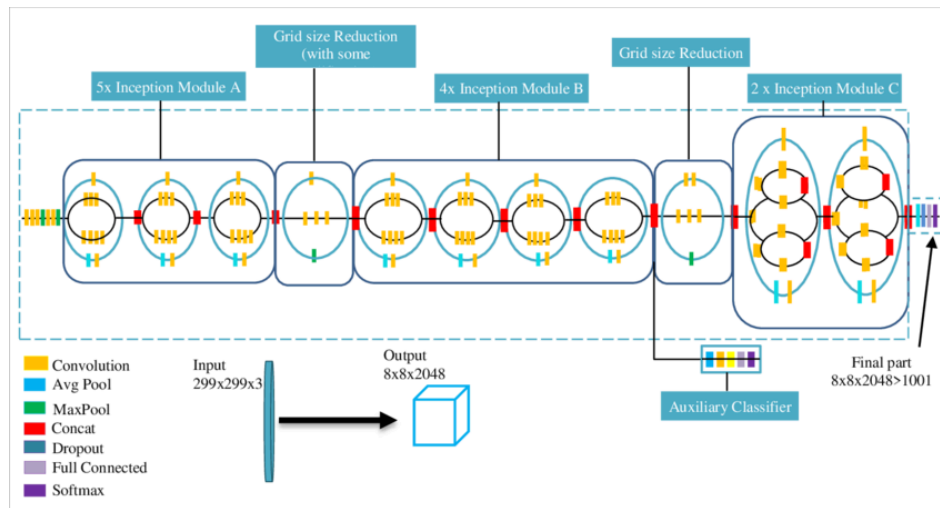


Figura 17: Arquitectura de la red *InceptionV3*.

Fuente: [parraguirre-Villanueva et al., 2022].

La red se compone de forma similar a *ResNet50* por su uso de bloques dentro de los cuales se hacen operaciones convolucionales. En el caso de esta red estos bloques se llaman *Inception Modules* y se ocupan bloques de 1x1 para reducir el tamaño de la grilla de datos entre módulos. Cada una de estas estructuras serán descritas en mayor detalle en la siguiente sección.

Módulo *Inception*

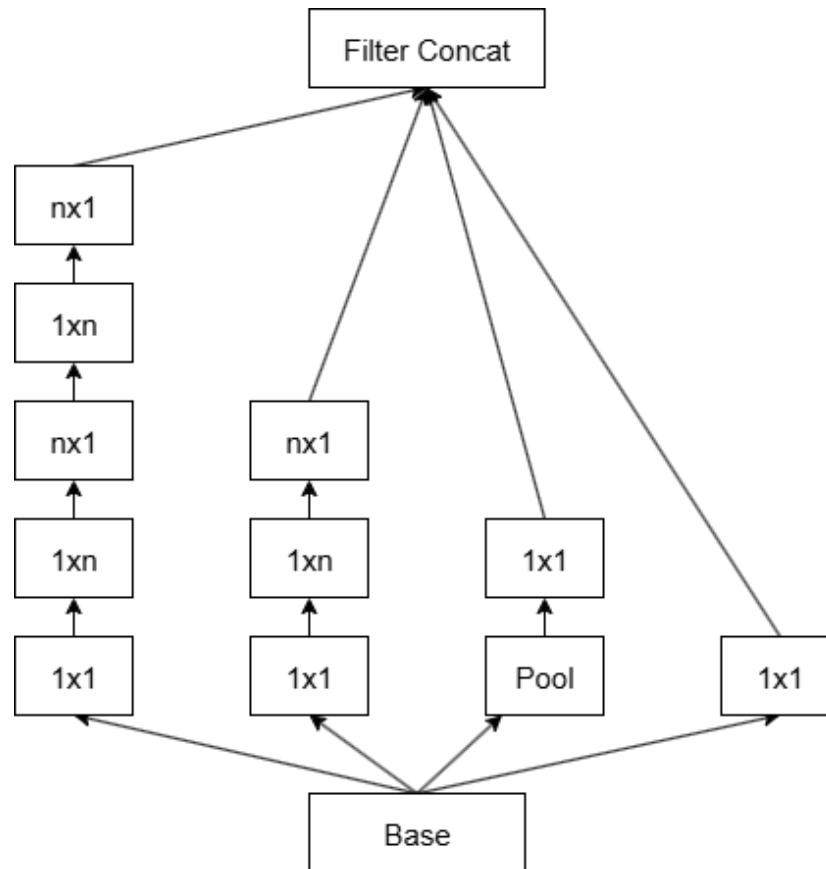


Figura 18: Módulo *Inception* con convoluciones $n \times n$ factorizadas.
Fuente: Elaboración propia.

La estructura más importante de esta red es el tal módulo *inception*. Lo que hace este módulo es que en vez de hacer solo una convolución o una serie de convoluciones de manera secuencial, se hacen varias convoluciones de distintas dimensiones en paralelo para luego concatenar los resultados. Esto le permite a la red a aprender características y patrones con múltiples dimensiones simultáneamente y hasta le permite determinar cual dimensión es la más relevante para un cierto *input*.

Convoluciones factorizadas.

El mayor cambio que propone *InceptionV3* es la factorización de operaciones convolucionales. En su reporte los autores proponen que la substitución de operaciones matriciales, por ejemplo de 5×5 por dos operaciones 3×3 en secuencia, permiten obtener resultados idénticos pero con un menor consumo de recursos por parte del computador. En base a esto proponen una reducción aun mayor utilizando convoluciones asimétricas para reemplazar las convoluciones factorizadas. Esto significa que una operación 3×3 puede ser sustituida por dos operaciones, una 1×3 seguida de otra 3×1 , lo

que resulta en una reducción de procesamiento necesario para una convolución 3×3 de hasta 33 %.

En resumen, descomponen las operaciones convolucionales en combinaciones de operaciones $1 \times n$, $n \times 1$, lo que reduce el nivel de procesamiento requerido a lo largo de la red. Cabe mencionar que se encontró que estas factorizaciones no son muy útiles en las primeras capas de la red, pero brillan en capas donde la data tiene una dimensión $M \times M$, $M \in [12, 20]$.

Reducción de dimensionalidad con 1×1

Todas las convoluciones dentro del módulo *inception* se realizan después de utilizar convoluciones 1×1 para reducir la dimensión de la data, permitiendo que las operaciones que le siguen sean menos computacionalmente costosas.

Clasificadores Auxiliares

Los clasificadores auxiliares son exactamente lo que suenan, son clasificadores que se posicionan antes de la cabeza de la red; el objetivo de estos es mitigar el problema del gradiente desvaneciente al generar clasificaciones más temprano en la red.

Label Smoothing

A diferencia de otros modelos donde las etiquetas corresponden a valores binarios (por ejemplo: $[1, 0, 0]$), en *Inception* se permite que las etiquetas tengan un valor dentro del rango de 0 y 1, lo que le permite generalizar mejor al reducir la sobreconfianza de las predicciones.

Este modelo ha tenido un gran impacto en el área de reconocimiento de imágenes desde su publicación, rápidamente convirtiéndose en un modelo estado del arte que requiere el entrenamiento de tan solo 5 millones de parámetros en comparación con redes como *AlexNet* [Krizhevsky et al., 2012] y VGG [Simonyan y Zisserman, 2015] que requieren el entrenamiento de 60 y 138 millones de parámetros respectivamente. Esto lleva a que el tiempo de inferencia del modelo sea bajo, por lo que ha sido adoptado para aplicaciones de reconocimiento en tiempo real incluyendo la interpretación de lenguas de señas.

3.2.2. Entrenamiento y selección de modelo

Para poder escoger que modelo se va a seleccionar es necesario determinar que parámetros se deben utilizar, es por esto que se propone realizar el entrenamiento del modelo utilizando un algoritmo de optimización de hiperparámetros, en este caso en particular, se va a optar por utilizar el optimizador *Hyperband* [Li et al., 2018].

El objetivo de este proceso va a ser determinar el mejor modelo y el conjunto de hiperparámetros que permiten que su rendimiento sea mejor que otros. Debido al alto costo computacional y tiempo requerido para poder explorar estas combinaciones para todos los

modelos a la vez, se va a optar por explorar estas combinaciones por modelo de forma individual y una vez encontrados los mejores modelos para *VGG16*, *ResNet50* y *InceptionV3*, se va a seleccionar el mejor según la métrica *Weighted Validation F1-Score* obtenida durante su entrenamiento.

La figura 14 presenta un resumen sobre proceso de entrenamiento que se propone.

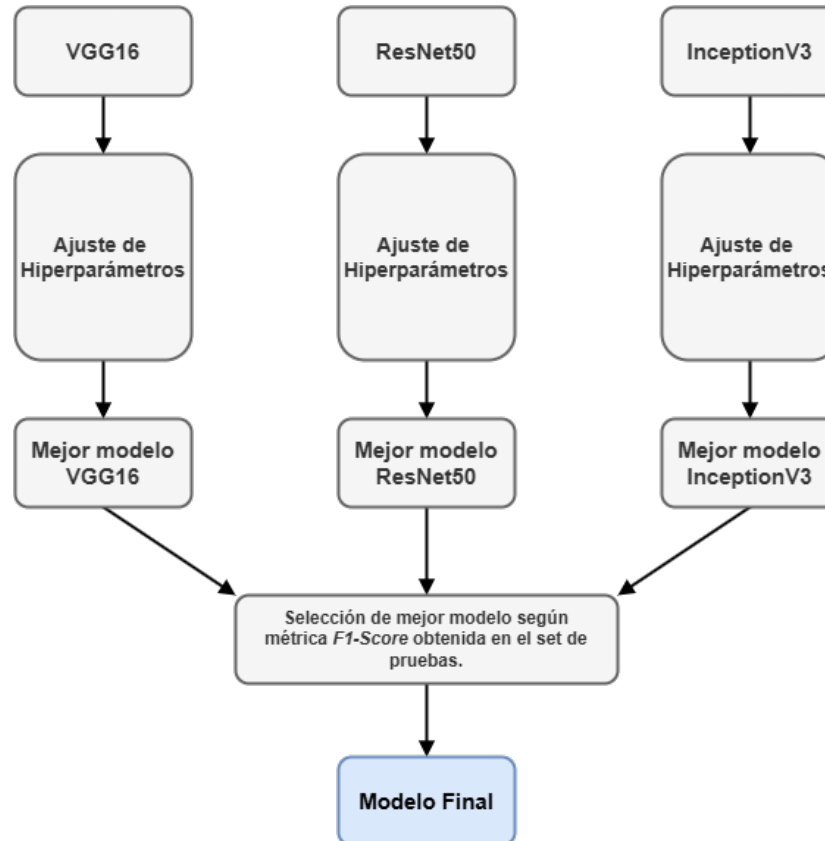


Figura 19: Proceso de entrenamiento y selección de modelo.

Fuente: Elaboración Propia.

Es importante detallar que el uso de la métrica *weighted validation f1-score* va a ser importante debido a que las clases de nuestro conjunto de datos no van a estar balanceadas, por lo que esta métrica nos permite evaluar con mayor precisión el rendimiento del modelo por sobre métricas como *precision* o el *recall*.

3.2.3. Selección de hiperparámetros para optimizador

El proceso de ajuste de hiperparámetros requiere que definamos cuales parámetros se van a poder modificar. Para esto es importante tomar en consideración que un mayor espacio de

búsqueda nos podría permitir explorar una mayor cantidad de combinaciones de parámetros y potencialmente encontrar una que sea particularmente buena para este problema. Pero a medida que crece el espacio de búsqueda, aumenta con ello el tiempo de cómputo y las demandas sobre el *hardware* necesario para poder explorar todas estas opciones.

Debido a las limitaciones de *hardware* y tiempo con la que nos enfrentamos en el desarrollo de este trabajo, se propone la siguiente grilla de hiperparámetros que van a ser ajustados y los valores que estos pueden tomar:

Tabla 1: Grilla de hiperparámetros.

Fuente: Elaboración Propia.

Parámetro	Valor
<i>Learning Rate</i>	[0.01, 0.00001] <i>Sampling: Log</i>
<i>Dropout Rate</i>	[0.0, 0.6] <i>Step = 0.05</i>
<i>Optimizer</i>	[Adam, SGD (<i>momentum</i> = 0.9), RMSprop (<i>rho</i> = 0.9)]
<i>Pooling layer</i>	[Flatten, GlobalAveragePooling2D]
<i>Data Augmentation</i>	[True, False]
<i>L2 Regularization</i>	[True, False]

Los parámetros que se van a fijar se presentan a continuación:

■ **Data Augmentation**

Al trabajar con la identificación de señas en lengua de señas, es crucial que las variaciones aplicadas a las imágenes no alteren drásticamente la representación. Un cambio excesivo podría provocar que la seña deje de corresponder a su letra original y se confunda con otra. Además, una técnica de aumento de datos demasiado agresiva podría eliminar características relevantes como los detalles de las manos, lo que dificultaría el aprendizaje del modelo y empeoraría su rendimiento.

Es por lo anterior, que se propone utilizar las siguientes modificaciones al aplicar aumento de datos a nuestro conjunto de entrenamiento:

Tabla 2: Técnicas de *data augmentation* utilizadas.

Fuente: Elaboración Propia.

Técnica de aumento	Valor
<i>Zoom</i>	[0 %, 20 %]
<i>Rotation Range</i>	$\pm 5^\circ$
<i>Shear range</i>	$\pm 5^\circ$ <i>horizontal</i> , $\pm 5^\circ$ <i>vertical</i>
<i>Brightness range</i>	[-10 %, +25 %]
<i>Horizontal Flip</i>	True
<i>Fill mode</i>	Nearest

- **L2 Regularization:** Se va a fijar el factor de la regularización a 0.001.

- **Batch Size:** El *batch size* que se va a utilizar para la exploración y entrenamiento de los modelos va a ser de 32.

También se hace mención a las funciones de *callback* que se utilizaran en el algoritmo de optimización de hiperparámetros, los cuales son:

- **Early Stopping:** Regularizador que detiene el entrenamiento de una red para que esta no tienda a sobre ajustar. Utiliza un parámetro especificado por el usuario para determinar cuando se debe detener el entrenamiento tras un número de *epochs* de entrenamiento donde no muestre mejora.
- **Reduce Learning Rate on Plateau (RLRP):** Regularizador que reduce la tasa de entrenamiento por un factor tras una cierta cantidad de *epochs* donde el parámetro seleccionado por el usuario no presenta mejora.

Tabla 3: *Callbacks* durante selección de modelo
Fuente: Elaboración Propia.

Parámetro	Valor
Métrica Early Stopping	Validation Loss
Early Stopping Patience	7
Early Stopping Delta	0.001
Métrica Reduce Learning Rate	Validation Loss
Reduce Learning Rate Patience	2
Reduce Learning Rate Factor	0.5

3.3. Ablation Study propuesto

Debido a que el resultado que se va a obtener mediante el entrenamiento de los modelos con el optimizador de hiperparámetros va simplemente a presentar que parámetros se utilizaron, no tenemos el conocimiento de cuanto afectó al rendimiento la utilización de ciertos parámetros, como por ejemplo si el uso de aumento de datos mejoró de manera drástica el rendimiento o si el cambio que presenta es relativamente bajo.

Es por esto que se propone realizar una serie de *Ablation studies* para poder evaluar el impacto que tienen ciertos parámetros en el modelo. Dos de estos estudios se van a realizar modificando parámetros encontrados durante el entrenamiento, mientras que el último de estos va a estudiar el efecto que *fine tuning* puede presentar en el rendimiento del modelo.

Se presentan las siguientes preguntas que se desean responder:

- ¿Como se comporta el modelo si se le agrega o quita aumento de datos?
- ¿Cual es el rendimiento del modelo con los algoritmos de optimización distintos al que obtuvo?
- ¿Que efecto tiene en el rendimiento del modelo la aplicación de *fine tuning*?

Para poder evaluar el efecto en el rendimiento que tiene la modificación de estos parámetros, se va a volver a entrenar el modelo utilizando el resto de los parámetros encontrados utilizando el mismo *split* de datos con la *random seed*¹⁰ seleccionada, asegurando la reproducibilidad del entrenamiento del modelo.

Para evaluar el efecto que tiene en el rendimiento el cambiar los parámetros de aumento de datos y las funciones de optimización, simplemente se van a volver a entrenar los modelos con estos parámetros cambiados en su sección correspondiente.

En el caso del uso de *fine tuning*, se va a seguir el procedimiento presentado en la figura a continuación:

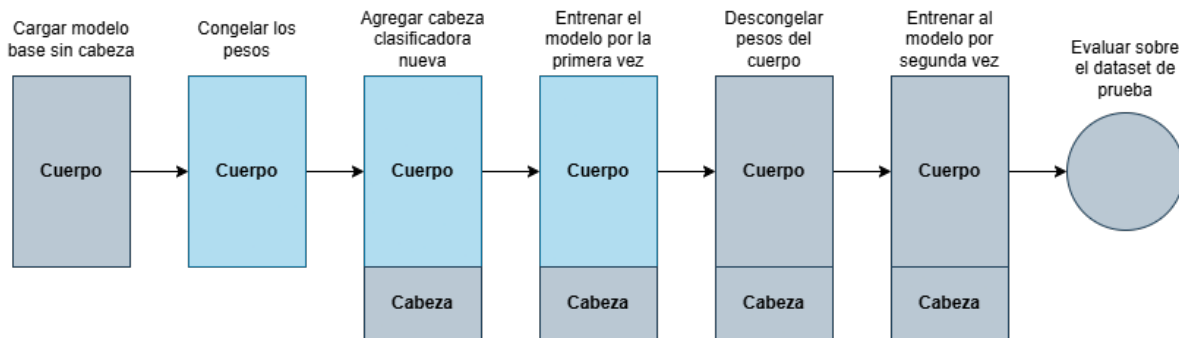


Figura 20: Proceso de *fine tuning* propuesto.
Fuente: Elaboración Propia.

Para estudiar cómo *fine tuning* influye en el rendimiento del modelo, se plantea descongelar las capas del modelo. La cantidad de capas a descongelar va a depender de que modelo sea seleccionado, esto podrá variar entre descongelar solo un par de capas hasta todas las capas del modelo.

Para medir los efectos que van a tener estas modificaciones sobre el modelo se van a evaluar sobre el *dataset* de pruebas ciegas y se medirá su *weighted F1 Score* obtenido.

El objetivo principal de desarrollar estos experimentos no solo va a ser el evaluar el impacto de estos parámetros o componentes en nuestro sistema, sino que además nos va a informar sobre si la exploración en un trabajo futuro sobre la implementación de técnicas de ajuste

¹⁰*Random seed*: Parámetro que permite la reproducibilidad del entrenamiento de un modelo.

fino en el proceso de optimización de parámetros, será suficientemente beneficiosa para justificar el mayor costo de entrenamiento.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se presentaran los resultados obtenidos tras seguir los pasos planteados en el capítulo anterior, se discutirán los razonamientos por los que se tomaron ciertas decisiones y se explicara el paso a paso que se siguió para llegar al modelo final de esta memoria.

4.1. Herramientas

En esta sección se describen las herramientas utilizadas en el desarrollo de este proyecto, las cuales son las siguientes:

4.1.1. *Hardware*

- *Nvidia RTX 4070* con 12 GB de VRAM.
- *AMD Ryzen 5600x*.
- 32 GB RAM DDR4.

4.1.2. *Software*

- *Python 3.10.16*.
- *Tensorflow 3.10* (Se uso esta versión ya que es la última que tiene soporte de GPU sin tener que usar WSL¹¹).
- *Keras* (La versión de la librería que está contenida en Tensorflow 3.10).

4.2. *Datasets*

A continuación se presentan los *datasets* que fueron utilizados en el desarrollo de este proyecto. Cabe mencionar que los primeros dos *datasets* que se listan aquí no fueron utilizados en su totalidad, sino que ejemplos de algunas representaciones fueron utilizadas en el *data-set* de entrenamiento que se discute más adelante. En particular, la razón por la que se optó

¹¹WSL: *Windows Subsystem for Linux*.

por crear un nuevo *dataset* de *LSCh* en vez de utilizar el ya existente que se presenta aquí es por que encontramos que las representaciones que se utilizan para las siguientes letras no eran suficientemente buenas: (D, I, S, U, V, W, X, Y, Z).

La razón por la que utilizamos ejemplos de representaciones de otros *datasets* en el nuestro, es que nos permiten tanto balancear las clases menos representadas en nuestro *dataset*, como además aportar con una mayor diversidad de fondos, distancias, tonos de piel y hasta leves variaciones en las formas de las manos que se consideraron que podrían permitir al modelo generalizar mejor.

4.2.1. Lengua de Señas Chilena 26 Clases (Beitez, Catril)

Dataset elaborado por Luiz Beitez y Ricardo Catril para su proyecto de título de la universidad del Bío-Bío [Benítez y Catril, 2022]. El *dataset* en su versión más reciente (v.2) se conforma de 1105 imágenes de (416x416) píxeles de resolución. Consiste de representaciones para 26 letras del alfabeto de *LSCh*, omitiendo la letra Ñ debido a que su representación sería muy similar a la letra N o a la letra H.

Su *dataset* se puede acceder mediante el siguiente vínculo:

<https://universe.roboflow.com/project-jak2u/lengua-de-senas-chilena-26-clases>

4.2.2. American Sign Language Dataset (Ayush Thakur)

Dataset de *ASL* para clasificación de imágenes compuesto por 2515 fotos pertenecientes a 36 clases, 26 de estas siendo las representaciones de las letras del alfabeto *ASL* y las 10 restantes corresponden a las representaciones de valores numéricos desde 0 a 9. Este *dataset* contiene imágenes recortadas para solo mostrar la representación de la clase en un fondo negro.

El *dataset* se puede acceder mediante el siguiente vínculo:

<https://www.kaggle.com/datasets/ayuraj/asl-dataset>

4.2.3. Dataset recolectado manualmente

Este *dataset* consta de 3136 imágenes tomadas manualmente y con el aporte de terceros quienes son nombrados en la sección de agradecimientos de este trabajo. Las imágenes son categorizadas en 26 clases pertenecientes al alfabeto de la *LSCh*, con la excepción de la letra Ñ tal como fue mencionado anteriormente.

Este *dataset* también cumple el objetivo de mejorar sobre los *datasets* anteriores al utilizar

mejores representaciones para las señas, incluir una mayor cantidad de representaciones tanto en el número de imágenes como de personas involucradas en su desarrollo, siendo en total 8 personas. Esto último es relevante ya que nos permite introducir una mayor variedad de manos, tonos de piel y formas en que las distintas personas hacen la misma seña.



Figura 21: Representaciones utilizadas en el *dataset* de LSCh.
Fuente: Elaboración Propia.

■ Estructura

El *dataset* contiene 26 carpetas cuyo nombre es la clase de la seña representada (ver figura), dentro de cada carpeta se encuentran las imágenes de las representaciones de la clase.

Name	Date modified	Type	Size
A	5/11/2025 7:10 PM	File folder	
B	5/11/2025 7:10 PM	File folder	
C	5/12/2025 8:46 PM	File folder	
D	5/11/2025 7:10 PM	File folder	
E	5/11/2025 7:10 PM	File folder	
F	5/11/2025 7:10 PM	File folder	
G	5/11/2025 7:10 PM	File folder	
H	5/11/2025 7:10 PM	File folder	
I	5/12/2025 8:46 PM	File folder	
J	5/11/2025 7:10 PM	File folder	
K	5/11/2025 7:10 PM	File folder	
L	5/11/2025 7:10 PM	File folder	
M	5/11/2025 7:10 PM	File folder	
N	5/11/2025 7:10 PM	File folder	
O	5/11/2025 7:10 PM	File folder	
P	5/11/2025 7:10 PM	File folder	
Q	5/11/2025 7:10 PM	File folder	
R	5/11/2025 7:10 PM	File folder	
S	5/11/2025 7:10 PM	File folder	
T	5/11/2025 7:10 PM	File folder	
U	5/11/2025 7:10 PM	File folder	
V	5/11/2025 7:10 PM	File folder	
W	5/11/2025 7:10 PM	File folder	
X	5/11/2025 7:10 PM	File folder	
Y	5/11/2025 7:10 PM	File folder	
Z	5/12/2025 8:46 PM	File folder	

Figura 22: Estructura de las carpetas del *dataset*.

Fuente: Elaboración Propia.

■ Validación de los datos

Un proceso importante fue la validación de los datos y las representaciones utilizadas en el *dataset* elaborado, debido a que es necesario que esta data sea representativa de como se utilizan estas señas por las personas en situación de discapacidad auditiva. Si estas representaciones no fuesen adecuadas a su uso, entonces el *dataset* no nos serviría para entrenar un modelo adecuado para esta tarea.

Es por esto que nos pusimos en contacto con Mildren Reyes, quien es una intérprete acreditada por la *ASOCH* en la región de Valparaíso. Tras una reunión con ella se validó que las representaciones utilizadas en este *dataset* son válidas para la *LSCh*.

■ Procesamiento

El procesamiento de las imágenes consistió en eliminar malos ejemplos, los que incluyen: imágenes fuera de foco; obstrucciones a la representación o mala iluminación;

recortar las imágenes para solo dejar presente la representación de la mano y por último; modificar la resolución de las imágenes a 224x224 píxeles.

El proceso de recorte de las imágenes fue realizado con el uso de herramientas para detectar y recortar imágenes en base a el *bounding box* de un objeto (en este caso la mano), y en los casos que esto fallaba, el recorte se hizo manualmente utilizando las herramientas de edición de imágenes de *Windows 10*.

■ Separación de *dataset*

Para poder evaluar el rendimiento de un modelo se optó por que el *dataset* desarrollado fuese separado en dos *datasets* distintos: un *dataset* exclusivo para el entrenamiento de los modelos y otro *dataset* utilizado exclusivamente para evaluar los modelos. Se aseguró que estos *datasets* no compartiesen representaciones hechas por la misma persona.

El *dataset* de entrenamiento se nombro *Isch_improved_v3* y consta de 2044 imágenes con representaciones de 3 personas involucradas en el desarrollo del *dataset* final, más un par de representaciones de los *datasets Lengua de Señas Chilena 26 Clases* y *American Sign Language Dataset*, resultando en un total de 2284 imágenes.

El otro *dataset*, *test_set_v1*, consta de 1107 imágenes con representaciones de las otras 5 personas involucradas en la elaboración del *dataset*.

Esta separación nos permite tener un conjunto de datos con suficiente variedad para llevar a cabo un entrenamiento robusto y un segundo conjunto de datos suficientemente grande, para realizar pruebas que nos permita una mejor idea del rendimiento del modelo en casos de uso reales.

4.3. Selección de modelo base

Siguiendo el procedimiento establecido la sección 3.1.1 del capítulo anterior, se procedió a correr el algoritmo de ajuste de hiperparámetros para los tres modelos. El conjunto que se utilizó para el entrenamiento se separó en un set de entrenamiento y un set de validación con un *split* de 80 % entrenamiento y 20 % validación.

Se llevaron a cabo dos rondas de ajuste de hiperparámetros por modelo, de tal forma que nos pudiésemos asegurar de que se explorase un mayor numero de posibles combinaciones. Por lo general estas fases de entrenamiento se demoraban cerca de 3 a 4 horas con la grilla de hiperparámetros previamente definida.

Las tablas 4, 5 y 6 a continuación presentan los conjuntos de hiperparámetros encontrados para los modelos *VGG16*, *ResNet50* y *InceptionV3* respectivamente.

Tabla 4: Parámetros de mejor modelo VGG16 encontrado.

Fuente: Elaboración Propia.

Parámetro	Valor
<i>Learning Rate</i>	0.000249
<i>Dropout Rate</i>	0.25
<i>Optimizer</i>	<i>Adam</i>
<i>Pooling Layer</i>	<i>Flatten</i>
<i>Data Augmentation</i>	<i>True</i>
<i>L2 Regularization</i>	<i>True</i>

Tabla 5: Parámetros de mejor modelo ResNet50 encontrado.

Fuente: Elaboración Propia.

Parámetro	Valor
<i>Learning Rate</i>	0.000195
<i>Dropout Rate</i>	0.3
<i>Optimizer</i>	<i>Adam</i>
<i>Pooling Layer</i>	<i>Flatten</i>
<i>Data Augmentation</i>	<i>True</i>
<i>L2 Regularization</i>	<i>False</i>

Tabla 6: Parámetros de mejor modelo InceptionV3 encontrado.

Fuente: Elaboración Propia.

Parámetro	Valor
<i>Learning Rate</i>	0.000066
<i>Dropout Rate</i>	0.00
<i>Optimizer</i>	<i>Adam</i>
<i>Pooling Layer</i>	<i>Flatten</i>
<i>Data Augmentation</i>	<i>True</i>
<i>L2 Regularization</i>	<i>False</i>

Utilizando estos parámetros somos capaces de entrenar los modelos de forma separada y evaluarlos en el conjunto de pruebas ciegas. Se utilizó el mismo *data split* para entrenamiento y validación junto a la utilización de la misma *random seed* que se usó en el optimizador de hiperparámetros, los resultados fueron los siguientes:

(Todos los resultados de esta tabla son del tipo *weighted*.)

Tabla 7: Resultados de modelos tras ajuste de parámetros.
Fuente: Elaboración Propia.

Modelo	Validation F1-Score	Accuracy	Precision	Recall	F1-Score
VGG16	0.9735	0.8952	0.9015	0.8952	0.8944
ResNet50	0.9681	0.8528	0.8689	0.8528	0.8515
InceptionV3	0.9540	0.6920	0.7024	0.6920	0.6769

Se puede ver que el modelo con el mejor rendimiento para este problema fue VGG16, logrando obtener resultados correctos con una precisión de 90 %, mientras que al *InceptionV3* le fue peor, solamente alcanzando una precisión de 69 %.

Un fenómeno muy interesante pero también preocupante son los resultados altos de *weighted validation F1-Score*. Valores tan altos como estos por lo general son indicativo de que el modelo está tendiendo hacia el sobreajuste, lo cual se puede ver reflejado en el modelo *InceptionV3*, donde la diferencia entre sus resultados en el test y su métrica de validación es tan alta que no ha aprendido a generalizar bien y está memorizando la data del set de entrenamiento y validación.

Por el contrario podemos ver que esto afectó menos a *ResNet50* y drásticamente menos a VGG16, donde los resultados de este último demuestran que ha aprendido de las relaciones y patrones inherentes de la data y puede generalizar sobre data completamente nueva con una buena precisión.

Una explicación de lo que puede estar causando esto, tal como se mencionó anteriormente, es que se está tendiendo al sobreajuste en el conjunto de validación. Esto puede ser porque la data de este conjunto es demasiado similar a la data de entrenamiento, posiblemente dado a que provienen del mismo conjunto de datos y pueden compartir la persona que hizo una representación.

Para mitigar este tipo de problemas existen distintas opciones, variando entre varias otras desde: asegurarse que la data de validación no comparta el mismo origen que la de entrenamiento, aumentando la cantidad de imágenes y aumentando la cantidad de personas haciendo las representaciones en el conjunto de datos.

Lo que nos va a permitir entender de mejor forma como está operando el modelo va a ser el reporte de clasificación y la matriz de confusión. A continuación se presentarán estas para los modelos VGG16, *ResNet50* y *InceptionV3* respectivamente.

4.3.1. Resultados VGG16

- El modelo detuvo su entrenamiento tras 54 *epochs*.
- De las 1107 predicciones, se equivocó en 116.

Tabla 8: Reporte de clasificación de modelo basado en VGG16.
Fuente: Elaboración Propia.

Class	Precision	Recall	F1-Score	Support
A	1.0000	0.9778	0.9888	45
B	0.9787	1.0000	0.9892	47
C	0.9189	0.8293	0.8718	41
D	0.9412	0.7805	0.8533	41
E	0.9756	0.9756	0.9756	41
F	0.9714	0.8095	0.8831	42
G	0.8667	0.8125	0.8387	32
H	0.8039	1.0000	0.8913	41
I	0.8846	0.6970	0.7797	33
J	0.9756	0.9091	0.9412	44
K	0.9500	0.8444	0.8941	45
L	1.0000	1.0000	1.0000	45
M	0.8654	1.0000	0.9278	45
N	0.7778	0.7955	0.7865	44
O	0.7885	0.9762	0.8723	42
P	0.9268	0.8444	0.8837	45
Q	0.9167	0.8250	0.8684	40
R	0.8478	0.9286	0.8864	42
S	0.8980	1.0000	0.9462	44
T	0.9565	1.0000	0.9778	44
U	0.8889	0.9091	0.9091	44
V	0.9091	0.9091	0.9091	43
W	0.7755	0.9500	0.8539	40
X	0.9412	0.6667	0.7805	48
Y	0.9783	1.0000	0.9890	45
Z	0.6667	0.7556	0.7083	45
Accuracy			0.8952	1107
Macro Avg	0.9001	0.8930	0.8925	1107
Weighted Avg	0.9015	0.8952	0.8944	1107

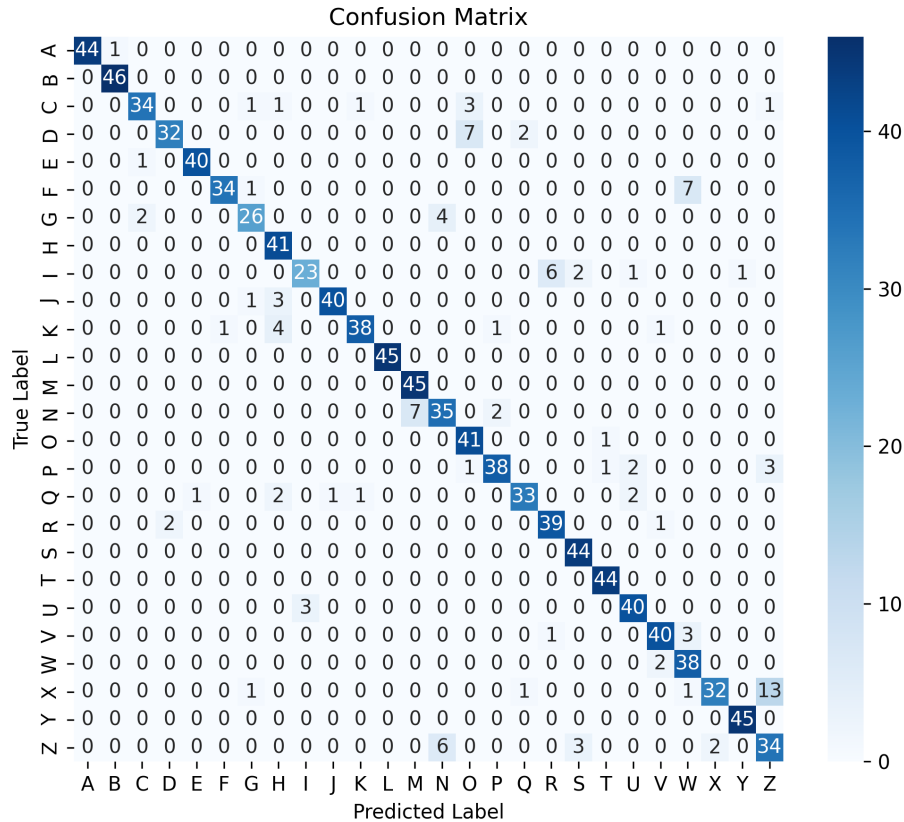


Figura 23: Matriz de confusión de la prueba.
Fuente: Elaboración Propia.

Hay un par de detalles interesantes respecto a estos resultados:

- A pesar de que el modelo parece tender al sobreajuste en el conjunto de validación, su rendimiento demuestra que ha logrado aprender de las representaciones y es capaz de generalizar bastante bien. Esto le permite obtener buenos resultados con data completamente nueva.
- A diferencia de lo que se esperaba que fuese a suceder, a este modelo no le costó diferenciar entre representaciones similares como la letra *M* y la *N*, o las letras *W* y *Y*.
- La mayor dificultad que tiene el modelo es la identificación correcta entre las letras *X* y *Z*, lo cual es de esperarse debido a que la representación de estas son bastante similares.

4.3.2. Resultados ResNet50

- El modelo detuvo su entrenamiento tras 25 *epochs*.

- De las 1107 predicciones, se equivocó en 163.

Tabla 9: Reporte de clasificación de modelo basado en ResNet50.

Fuente: Elaboración Propia.

Class	Precision	Recall	F1-Score	Support
A	1.0000	0.9333	0.9655	45
B	1.0000	0.9783	0.9890	46
C	0.8780	0.8780	0.8780	41
D	0.7692	0.7317	0.7500	41
E	0.9535	1.0000	0.9762	41
F	0.7045	0.7381	0.7209	42
G	0.7647	0.8125	0.7879	32
H	0.5541	1.0000	0.7130	41
I	0.8333	0.9091	0.8696	33
J	0.8696	0.9091	0.8889	44
K	1.0000	0.6000	0.7500	45
L	1.0000	1.0000	1.0000	45
M	0.9318	0.9111	0.9213	45
N	0.8723	0.9318	0.9011	44
O	0.6596	0.7381	0.6966	42
P	0.8800	0.9778	0.9263	45
Q	0.9211	0.8750	0.8974	40
R	0.8684	0.8750	0.8528	42
S	1.0000	0.6591	0.7945	44
T	1.0000	1.0000	1.0000	44
U	0.8444	0.8837	0.8636	43
V	0.7049	0.9773	0.8190	44
W	0.8378	0.7750	0.8052	40
X	0.7857	0.4583	0.5789	48
Y	1.0000	0.9778	0.9888	45
Z	0.8718	0.7556	0.8095	45
Accuracy			0.8528	1107
Macro Avg	0.8656	0.8537	0.8506	1107
Weighted Avg	0.8689	0.8528	0.8515	1107

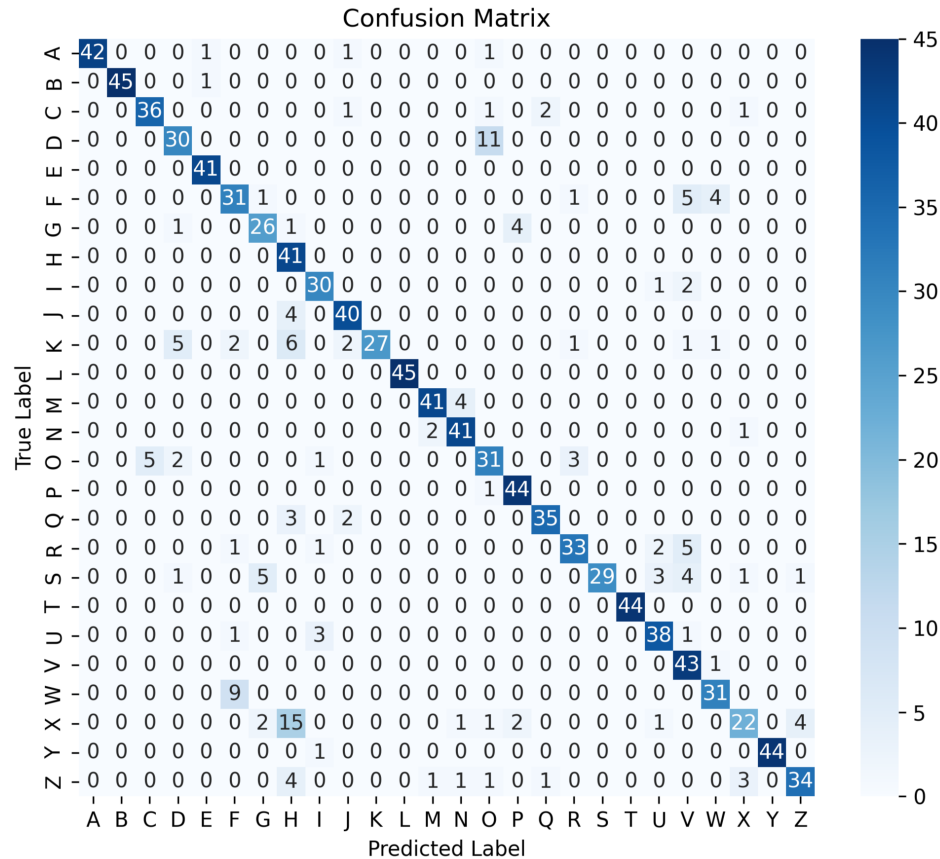


Figura 24: Matriz de confusión de mejor modelo *ResNet50*.

Fuente: Elaboración Propia.

Este modelo no es tanto peor que el modelo *VGG16*, pero el punto de falla más grande que tenemos vuelve a ser la letra X, donde el *recall* de esta nos muestra que la cantidad de predicciones verdaderas correctas fueron menores a la cantidad de predicciones totales para la letra, significando que al menos para la letra X el modelo se va a equivocar más de el 50 % de las veces.

4.3.3. Resultados *InceptionV3*

- El modelo detuvo su entrenamiento tras 26 *epochs*.
- De las 1107 predicciones, se equivocó en 341.

Tabla 10: Reporte de clasificación de modelo basado en *InceptionV3*.
Fuente: Elaboración Propia.

Class	Precision	Recall	F1-Score	Support
A	0.7321	0.9111	0.8119	45
B	0.8039	0.8913	0.8454	46
C	0.7561	0.7561	0.7561	41
D	0.7826	0.4390	0.5625	41
E	0.8444	0.9268	0.8837	41
F	0.7949	0.7381	0.7654	42
G	0.5217	0.7500	0.6154	32
H	0.5522	0.9024	0.6852	41
I	0.5000	0.5152	0.5075	33
J	0.6222	0.6364	0.6292	44
K	0.7931	0.5111	0.6216	45
L	0.7447	0.7778	0.7609	45
M	0.7347	0.8000	0.7660	45
N	0.5692	0.8409	0.6789	44
O	0.6939	0.8095	0.7473	42
P	0.7193	0.9111	0.8039	45
Q	0.8000	0.6000	0.6857	40
R	0.8889	0.5714	0.6957	42
S	0.7949	0.7045	0.7470	44
T	0.7288	0.9773	0.8350	44
U	0.6226	0.7674	0.6875	43
V	0.5909	0.5909	0.5909	44
W	0.5370	0.7250	0.6170	40
X	0.4000	0.0833	0.1379	48
Y	0.9200	0.5111	0.6571	45
Z	0.7391	0.3778	0.5000	45
Accuracy			0.6920	1107
Macro Avg	0.6995	0.6933	0.6767	1107
Weighted Avg	0.7024	0.6920	0.6769	1107

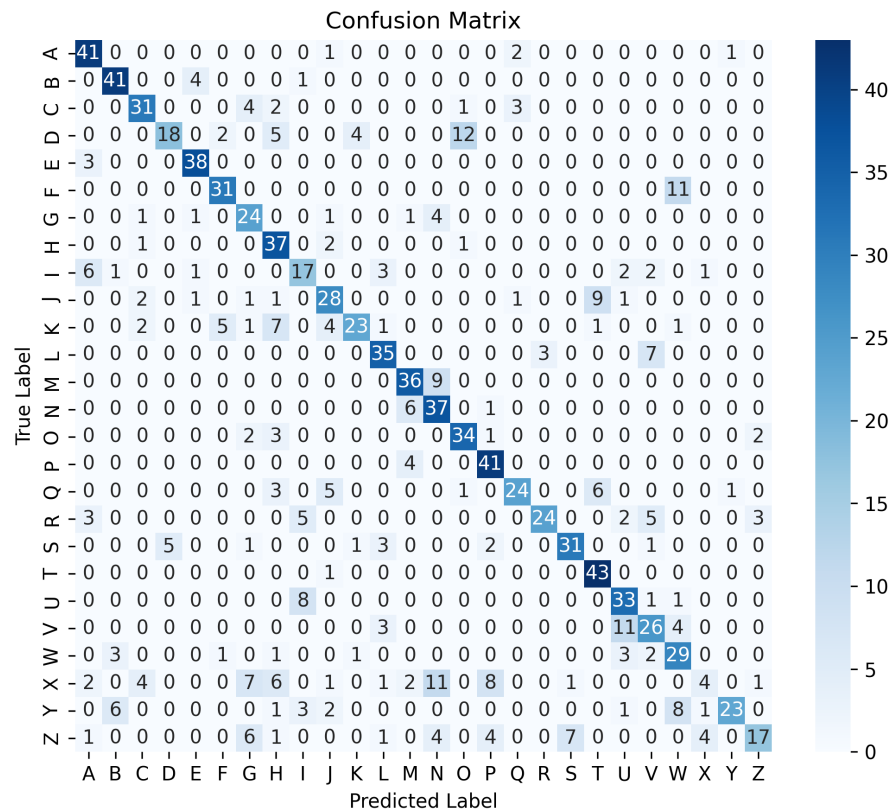


Figura 25: Matriz de confusión de mejor modelo *InceptionV3*.

Fuente: Elaboración Propia.

Al igual que todos los otros modelos que fueron entrenados este también se equivocó en la letra X, resultando en el peor *recall* que hemos observado. Además presenta el peor rendimiento general de todos los modelos. Claramente este modelo no fue bueno.

A diferencia de lo que se sugiere de otros estudios, para nuestro caso particular el modelo *InceptionV3* no es adecuado. Esto se puede deber a que la arquitectura de la cabeza clasificadora no es adecuada para este modelo o es posible que la data, en la forma que esta es entregada a los modelos tampoco lo sea. Esto lleva a los resultados que vemos presentes en las tablas.

En base a los resultados que se obtuvieron determinamos que el modelo más prometedor de los entrenados fue *VGG16*, donde los resultados sobre el conjunto de pruebas superaron a los otros dos modelos *ResNet50* y *InceptionV3* por 4 % y hasta 20 % respectivamente. El mayor punto de debilidad de todos estos modelos es el mismo, la dificultad de interpretar las letras X y Z, especialmente entre si.

4.4. Resultados de *Ablation Studies* propuestos

El siguiente paso fue la elaboración de los experimentos propuestos en la sección 3.4 del capítulo anterior. Los resultados de estos se presentaran en el formato de tablas con la información del modelo base y los modelos ajustados. Además se incluyen los *weighted F1-Score* de los modelos en el set de pruebas y la diferencia de rendimiento entre estos y el modelo base.

A continuación se presentan los resultados de los experimentos propuestos para el *ablation study* en la sección 3.3 del capítulo anterior. La tabla contiene información sobre los dos modelos con funciones de optimización distintas al modelo base y el modelo que se entrenó sin la utilización de aumento de datos. Los resultados van a incluir las métricas de *Accuracy*, *Weighted Precision*, *Weighted Recall* y *Weighted F1-Score* de sus reportes de clasificación.

Tabla 11: Parámetros de mejor modelo *VGG16* encontrado.
Fuente: Elaboración Propia.

Modelo	Data Augmentation	Optimizer	Accuracy	Precision	Recall	F1-Score
Baseline	True	Adam	0.8952	0.9015	0.8952	0.8944
Base_no_aug	False	Adam	0.8762	0.8834	0.8762	0.8745
Base_SGD	True	SGD	0.8771	0.8867	0.8771	0.8761
Base_RMSProp	True	RMSProp	0.8880	0.8951	0.8880	0.8860

De estos modelos hay un detalle interesante sobre el rendimiento de uno de estos, en particular el modelo donde se optó por no utilizar aumento de datos, y para esto debemos ver su matriz de confusión.

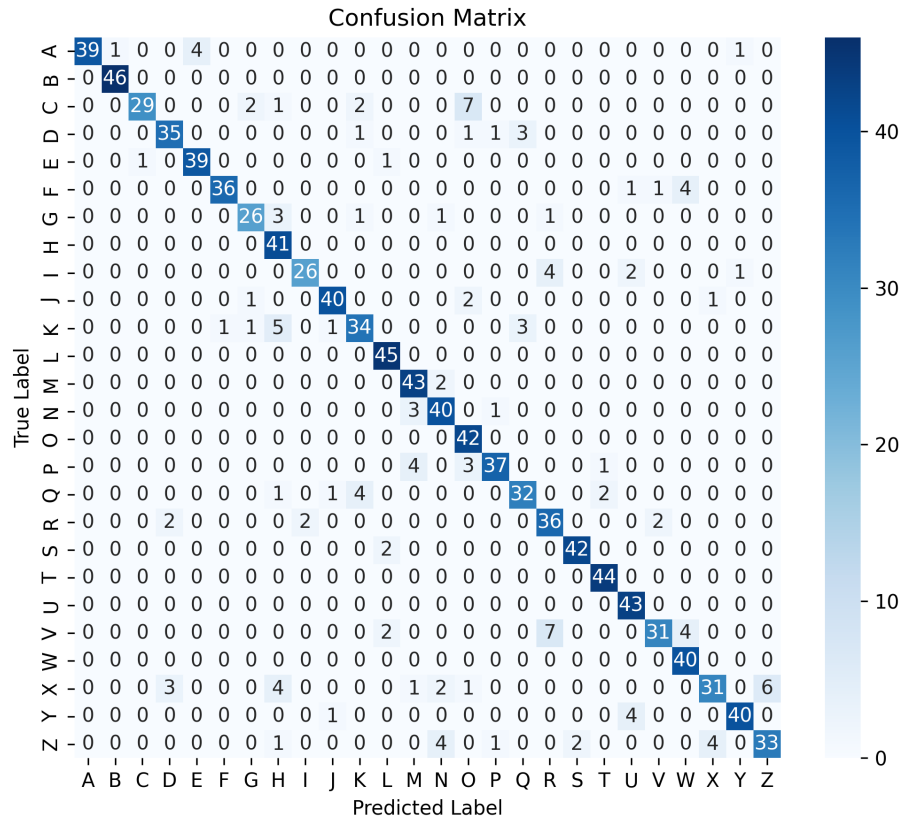


Figura 26: Matriz de confusión de modelo sin *data augmentation*.
Fuente: Elaboración Propia.

El detalle en que nos queremos enfocar es en los errores al identificar la letra X por la letra Z, ya que en este caso en comparación con nuestro modelo base la cantidad de veces que el sistema se equivoca en esta predicción es menos que la mitad del otro sistema. Este resultado nos podría indicar que el tipo de técnicas de aumento de datos que estamos utilizando pueden estar generando demasiado ruido o alterando las imágenes de tal manera que le genere confusión con representaciones suficientemente similares.

Ahora nos vamos a enfocar en los resultados de la aplicación de *fine tuning* en el modelo. Para determinar que capas descongelar se consideraron las características de la arquitectura de VGG16, por lo que se optó por descongelar bloques convolucionales en vez de capas individuales.

Ya que VGG16 se compone de 5 bloques convolucionales, decidimos experimentar descongelando bloque por bloque, partiendo con el último bloque convolucional hasta haber descongelado todos los bloques de la red. Los resultados de estos experimentos se presentan a continuación.

Tabla 12: Resultados de *fine tuning* sobre la red.
Fuente: Elaboración Propia.

Modelo	Bloques Descongelados	Accuracy	Precision	Recall	F1-Score
Baseline	Ninguna	0.8952	0.9015	0.8952	0.8944
Base_Block5	Block5_	0.8916	0.8987	0.8916	0.8898
Base_Block4	Block5_ - Block4_	0.9061	0.9120	0.9061	0.9040
Base_Block3	Block5_ - Block3_	0.9142	0.9193	0.9142	0.9129
Base_Block2	Block5_ - Block2_	0.9015	0.9075	0.9015	0.8991
Base_All	Todas las capas	0.9178	0.9223	0.9178	0.9160

Como se puede ver, la mejora sobre el rendimiento de la red al aplicar *fine tuning* no es insignificante. Con estos experimentos relativamente simples logramos encontrar una mejora de 2 % sobre los resultados de nuestro modelo inicial. A continuación se presenta la matriz de confusión y tabla de clasificación de este modelo.

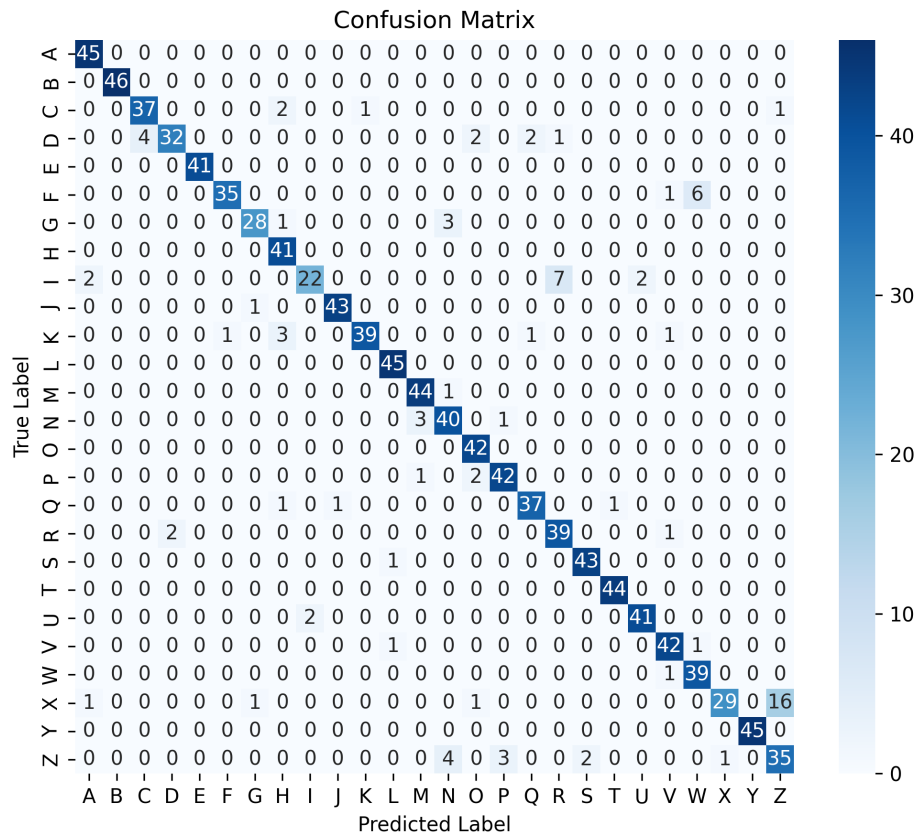


Figura 27: Matriz de confusión modelo con todas las capas descongeladas.
Fuente: Elaboración Propia.

Tabla 13: Reporte de clasificación de modelo con todos los pesos descongelados.

Fuente: Elaboración propia.

Class	Precision	Recall	F1-Score	Support
A	0.9375	1.0000	0.9677	45
B	1.0000	1.0000	1.0000	46
C	0.9024	0.9024	0.9024	41
D	0.9412	0.7805	0.8533	41
E	1.0000	1.0000	1.0000	41
F	0.9722	0.8333	0.8974	42
G	0.9333	0.8750	0.9032	32
H	0.8542	1.0000	0.9213	41
I	0.9167	0.6667	0.7719	33
J	0.9773	0.9773	0.9773	44
K	0.9750	0.8667	0.9176	45
L	0.9574	1.0000	0.9783	45
M	0.9167	0.9778	0.9462	45
N	0.8333	0.9091	0.8696	44
O	0.8936	1.0000	0.9438	42
P	0.9130	0.9333	0.9231	45
Q	0.9250	0.9250	0.9250	40
R	0.8298	0.9286	0.8764	42
S	0.9556	0.9773	0.9663	44
T	0.9778	1.0000	0.9888	44
U	0.9535	0.9535	0.9535	43
V	0.9130	0.9545	0.9333	44
W	0.8478	0.9750	0.9070	40
X	0.9667	0.6042	0.7436	48
Y	1.0000	1.0000	1.0000	45
Z	0.6731	0.7778	0.7216	45
Accuracy			0.9178	1107
Macro Avg	0.9218	0.9161	0.9150	1107
Weighted Avg	0.9223	0.9178	0.9160	1107

Es interesante que el descongelar solamente el último bloque convolucional empeoró el rendimiento del modelo, pero a medida que se iban descongelando los bloques más bajos, se veía una mejora. Se podría interpretar estos resultados como que las capas de extracción de características más simples tienen una mayor influencia en el rendimiento final del modelo, pero para poder asegurarse de esto se tendría que hacer un análisis más profundo, lo que se puede plantear para un trabajo futuro en base a este proyecto.

CAPÍTULO 5

CONCLUSIONES

5.1. Conclusiones Generales

En esta memoria se ha documentado el proceso realizado para desarrollar una red neuronal entrenada para la interpretación del alfabeto de la lengua de señas chilena. La utilización de técnicas de aprendizaje como *transfer learning* permiten que el desarrollo de modelos de clasificación de imágenes sea más simple, eficiente y factible hasta con conjuntos de datos limitados. La importancia que tiene la calidad de estos datos es crucial, porque malas representaciones o poca variedad visual en la data pueden llevar a que los modelos entrenados no sean capaces de aprender a generalizar en base a los datos.

La recolección de datos para el *dataset* que se desarrolló en esta memoria probó ser una tarea particularmente larga, que requirió el coordinar con varios terceros para aumentar la variedad de los datos, enseñarles sobre las representaciones que se utilizaron, verificar que los datos fuesen de buena calidad y validarlos con una intérprete acreditada por la ASOCH. Se considera que este paso fue crucial ya que nos permitió asegurar que la data fuese verdaderamente representativa de la realidad y se considera que presenta una mejora sobre conjuntos de datos públicamente disponibles.

El proceso de entrenar y comparar los distintos modelos base nos permitió experimentar con las distintas arquitecturas y aprender el efecto que tienen los distintos parámetros en cada una, como también encontrar posibles puntos de falla en nuestro modelo o la data. Adicionalmente el ver que los tres modelos tienden a tener problemas diferenciando la representación de la letra Z y la letra X significa que habría que optar por una representación distinta para estas señas, o que se tendrían que tomar otras decisiones respecto a la arquitectura de los modelos para mitigar este problema.

Junto con lo anterior, la separación de datos en los distintos conjuntos de entrenamiento, validación y pruebas fue de gran importancia. En particular el haber asegurado que las personas en los conjuntos de pruebas y entrenamiento fuesen mutuamente exclusivos nos dio una mejor idea del rendimiento del modelo en la realidad.

Respecto a los experimentos del *ablation study* que se llevaron a cabo, se ve que aplicar *fine tuning* permite mejorar el rendimiento de nuestro modelo por más de 2% en el mejor de los casos, y que en el peor de los casos la pérdida de rendimiento es mínima. Además las variaciones del uso de aumento de datos o de función de optimización, permiten concluir que a pesar de que estas implementaciones puedan ser computacionalmente costosas, el beneficio en el rendimiento del modelo justifican su uso.

5.2. Cumplimiento de los objetivos

A continuación se presentan de que manera se cumplieron los objetivos delineados en la subsección 1.4 de este trabajo.

1. Con respecto al primer objetivo específico planteado, en la sección 3.2.2 de la propuesta de la solución se presenta el proceso de la selección de los modelos base, los cuales son entrenados utilizando los pesos de *ImageNet*. En la sección 4.3 de la validación de la solución se presentan los resultados obtenidos tras la selección y entrenamiento del mejor modelo para la tarea de la interpretación del *dataset* del alfabeto de la lengua de señas chilena. La utilización de *fine tuning* es descrita durante el *ablation study* propuesto. Considerando que se logró adaptar el modelo VGG16 para la tarea propuesta mediante la aplicación de transferencia de aprendizaje, se considera que el objetivo fue cumplido exitosamente.
2. Respecto al segundo objetivo específico, durante la propuesta de la solución en la sección 3.1 de este trabajo se presentan los requerimientos que debería tener un nuevo *dataset* para el alfabeto de la LSCh, lo cual fue llevado a la práctica como se presenta en la sección 4.1 de la validación de la solución. La representatividad de la data fue corroborada por una intérprete acreditada por la ASOCH. El pre-procesamiento de los datos se describe inicialmente en la sección 3.1.1 y es puesto en práctica en la sección 4.2.3. Por esto se considera que el objetivo fue cumplido exitosamente.
3. Y respecto al tercer objetivo específico, inicialmente fue planteado bajo la premisa que se iba a desarrollar un modelo inicial y sobre este se iban a aplicar los métodos descritos, pero durante el transcurso de este trabajo se optó por utilizar el algoritmo de ajuste de hiperparámetros incluyendo aumento de datos como un parámetro ajustable. La aplicación de *fine tuning* se dejó para el *ablation study* propuesto y se desarrolló en la sección 4.4 de este trabajo. Lo que se quería lograr con éste objetivo se considera logrado, ya que se emplearon estas técnicas para mejorar el rendimiento del modelo.

Tomando en consideración el cumplimiento de los objetivos planteados, se considera que el objetivo general presentado en la subsección 1.3 de esta memoria fue cumplido. El modelo fue desarrollado y la utilización de *fine tuning* fue separada del proceso de entrenamiento y se llevó a cabo como parte de un *ablation study*.

5.3. Trabajo Futuro

El modelo en su forma base presenta un *accuracy* de 90 % para información nueva. Este resultado es bastante alto, pero como se presentó con el *ablation study* propuesto se puede mejorar. Por esto se pueden plantear posibles mejoras al modelo desarrollado o utilizar otras técnicas y medidas para construir un nuevo modelo que genere mejores resultados.

Uno de los posibles trabajos futuros sería la expansión de la grilla de hiperparámetros para el algoritmo de optimización de hiperparámetros, haciendo que esta incluya la técnica de *fine tuning*. Parece también posible ampliar el número de capas y neuronas en la grilla de hiperparámetros, para permitir al algoritmo un mayor espacio de búsqueda que pueda llegar a una mejor combinación de parámetros.

La selección de modelo base también puede dar a lugar a un estudio futuro. En esta memoria se trabajó con modelos cuya arquitectura corresponde a una red neuronal convolucional, pero modelos basados en arquitecturas del tipo *Vision Transformers (ViT)* han estado presentado los mejores resultados en problemas de detección de objetos y clasificación de imágenes. La implementación de estos modelos para el problema de la identificación de lenguas de señas, queda como una posible mejora sobre los sistemas previamente disponibles.

REFERENCIAS BIBLIOGRÁFICAS

- [Abu-Jamie y Abu-Naser, 2022] Abu-Jamie, T. N. y Abu-Naser, S. S. (2022). Classification of sign-language using deep learning by resnet. *International Journal of Academic Information Systems Research (IJAIRS)*, 6(8):25–34.
- [Alahmari et al., 2022] Alahmari, S., Altazi, B., Hwang, J., Hawkins, S., y Salem, T. (2022). A comprehensive review of deep learning-based methods for covid-19 detection using chest x-ray images. *IEEE Access*, PP:1–1.
- [Azawii et al., 2019] Azawii, A., Al-Janabi, S., y Al-Khateeb, B. (2019). Survey on intrusion detection systems based on deep learning. *Periodicals of Engineering and Natural Sciences (PEN)*, 7:1074–1095.
- [Benítez y Catril, 2022] Benítez y Catril, M. (2022). Reconocimiento en tiempo real del alfabeto de lengua de señas chilena empleando aprendizaje automático.
- [Congreso Nacional de Chile, 2021] Congreso Nacional de Chile (2021). Ley N.º 21.303, Modifica la ley N° 20.422, que establece normas sobre igualdad de oportunidades e inclusión social de personas con discapacidad, para promover el uso de la lengua de señas. Diario Oficial de la República de Chile. Publicada en el Diario Oficial el 22 de enero de 2021.
- [Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. En *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.
- [Feng et al., 2019] Feng, J., He, X., Teng, Q., Ren, C., Chen, H., y Li, Y. (2019). Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E*, 100.
- [Fowley y Ventresque, 2022] Fowley, F. y Ventresque, A. (2022). Sign language fingerspelling recognition using synthetic data.
- [Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- [He et al., 2015] He, K., Zhang, X., Ren, S., y Sun, J. (2015). Deep residual learning for image recognition.
- [ImageNet, 2014] ImageNet (2014). ILSVRC2014 Results. <https://image-net.org/challenges/LSVRC/2014/results#clsloc>. Accessed: Jun. 22, 2025.
- [ImageNet, 2015] ImageNet (2015). ILSVRC2015 Results. <https://image-net.org/challenges/LSVRC/2015/results.php>. Accessed: Jun. 7, 2025.

- [Iparraguirre-Villanueva *et al.*, 2022] Iparraguirre-Villanueva, O., Guevara-Ponce, V., Paredes, O., Sierra-Liñan, F., Zapata-Paulini, J., y Cabanillas-Carbonell, M. (2022). Convolutional neural networks with transfer learning for pneumonia detection. *International Journal of Advanced Computer Science and Applications*, 13.
- [Kamal y Ez-Zahraouy, 2023] Kamal, K. y Ez-Zahraouy, H. (2023). A comparison between the vgg16, vgg19 and resnet50 architecture frameworks for classification of normal and clahe processed medical images.
- [Koller, 2020] Koller, O. (2020). Quantitative survey of the state of the art in sign language recognition.
- [Kolluri *et al.*, 2020] Kolluri, J., Kumar, K., Phridviraj, M., y Razia, S. (2020). Reducing overfitting problem in machine learning using novel l1/4 regularization method. pp. 934–938.
- [Krizhevsky *et al.*, 2012] Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. En Pereira, F., Burges, C., Bottou, L., y Weinberger, K., editores, *Advances in Neural Information Processing Systems*, volumen 25. Curran Associates, Inc.
- [Li *et al.*, 2018] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., y Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization.
- [Mia *et al.*, 2023] Mia, K., Islam, T., Assaduzzaman, M., Shaha, S. P., Saha, A., Razzak, M. A., Dhar, A., Sarker, T., y Alvy, A. I. (2023). Isolated sign language recognition using hidden transfer learning. *International Journal of Computer Science and Information Technology Research*, 11(1).
- [Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York, NY, USA.
- [Mohsin *et al.*, 2023] Mohsin, S., Salim, B. W., Mohamedsaeed, A. K., Ibrahim, B. F., y Zeebaree, S. R. M. (2023). American sign language recognition based on transfer learning algorithms. *International Journal of Intelligent Systems and Applications in Engineering*, 12(5s):390–399.
- [Phung y Rhee, 2019] Phung, V. y Rhee, E. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, 9:4500.
- [Podareanu *et al.*, 2019] Podareanu, D., Codreanu, V., Aigner, S., Leeuwen, C., y Weinberg, V. (2019). Best practice guide - deep learning.
- [Raghu y Schmidt, 2020] Raghu, M. y Schmidt, E. (2020). A survey of deep learning for scientific discovery.
- [Sajjad *et al.*, 2019] Sajjad, M., Khan, S., Muhammad, K., Wu, W., Ullah, A., y Baik, S. W. (2019). Multi-grade brain tumor classification using deep cnn with extensive data augmentation. *Journal of Computational Science*, 30:174–182.

- [Sarker, 2021] Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):160.
- [Servicio Nacional de la Discapacidad (SENADIS), 2022] Servicio Nacional de la Discapacidad (SENADIS) (2022). III Estudio Nacional de la Discapacidad 2022 (III ENDISC 2022). Gobierno de Chile, Santiago, Chile. Disponible en: https://www.senadis.gob.cl/pag/693/2004/iii_estudio_nacional_de_la_discapacidad.
- [Simonyan y Zisserman, 2015] Simonyan, K. y Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.
- [Singh *et al.*, 2025] Singh, A., Ambalavanan, N., Metodiev Sirakov, N., y Nakhmani, A. (2025). Iterative data distillation and augmentation for enhancing deep learning performance with limited image training data. *IET Image Processing*, 19.
- [Sun *et al.*, 2015] Sun, Y., Liang, D., Wang, X., y Tang, X. (2015). Deepid3: Face recognition with very deep neural networks.
- [Szegedy *et al.*, 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., y Rabinovich, A. (2014). Going deeper with convolutions.
- [Szegedy *et al.*, 2015] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., y Wojna, Z. (2015). Rethinking the inception architecture for computer vision.
- [Yani *et al.*, 2019] Yani, M., Irawan, S., y Setianingsih, C. (2019). Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201:012052.