



TECNOLÓGICO
NACIONAL DE MÉXICO®



INSTITUTO TECNOLÓGICO SUPERIOR DE TEZIUTLÁN

Tesis



"DISEÑO DE SOFTWARE PARA LA DETECCIÓN DE
LENGUAJE DE SEÑAS MEXICANO UTILIZANDO
TÉCNICA DE VISIÓN ARTIFICIAL"

PRESENTA:

VANESSA PRADO NICOLAS

CON NÚMERO DE CONTROL

19TE0681

PARA OBTENER EL TÍTULO DE:

INGENIERO MECATRONICO

CLAVE DEL PROGRAMA ACADÉMICO

IMCT-2010-229

DIRECTOR (A) DE TESIS:

M.S.C ALFREDO CARRASCO ARAOZ

TEZIUTLÁN, PUEBLA, FEBRERO 2024

"La Juventud de hoy, Tecnología del Mañana"



PRELIMINARES

AGRADECIMIENTOS

Quisiera expresar mi más franco agradecimiento a quienes han sido fundamentales en el trayecto hacia la culminación de este proyecto académico. Sus contribuciones y apoyo incondicional han sido el motor que impulsó cada página de esta tesis.

En primer lugar, a mis queridos padre José Luis y a mi madre Francisca, cuyo amor, sacrificio y constante aliento han sido una luz que me guio durante este arduo trayecto. Su dedicación y valores han sido mi guía y motivación para alcanzar cada logro.

A mi amada pareja Félix Alberto, agradezco profundamente tu paciencia, amor, comprensión y aliento constante. Tu apoyo inquebrantable ha sido mi fuente de fortaleza en los momentos desafiantes, tu presencia ha convertido cada obstáculo en una oportunidad de crecimiento y muchas gracias por siempre estar para mí.

A mi hermana, María y sobrina, Alondra Estefanía, quienes, con su alegría y cariño, han sido el equilibrio perfecto. Sus risas y abrazos han sido un recordatorio constante de la importancia de encontrar equilibrio en la vida.

A todos mis amigos y profesores que me acompañaron durante la carrera y a mis seres queridos que me han brindado su apoyo, palabras de aliento y comprensión, les agradezco con todo el corazón. Este logro no solo es mío, sino también de cada persona que ha formado parte de mi vida y me ha inspirado a seguir adelante.

En resumen, mi gratitud no tiene límites. Este proyecto no solo representa mi esfuerzo individual, sino también el resultado de un esfuerzo colectivo. Mil gracias a todos por ser parte de este viaje y por compartir este logro conmigo.

RESUMEN

El Lenguaje de Señas Mexicano (LSM) representa un medio esencial de comunicación para la comunidad de personas sordas en México, pero enfrenta desafíos notables en cuanto a su comprensión y difusión. La ausencia de herramientas efectivas y accesibles para interpretar el LSM en tiempo real genera limitaciones en la inclusión y la comunicación efectiva de las personas con discapacidad auditiva. La falta de soluciones precisas y accesibles para el reconocimiento del LSM destaca un vacío tecnológico crucial. Este estudio aborda la insuficiencia de una herramienta eficaz y precisa destinada a identificar y comprender el LSM en tiempo real.

Se propone el diseño y entrenamiento de un algoritmo mediante redes neuronales con el objetivo de detectar el lenguaje de señas, brindando un respaldo valioso a las personas sordomudas. La obtención del conjunto de datos ordenados implica el entrenamiento del modelo para lograr un porcentaje aceptable de precisión, asegurando así su aprobación. Este enfoque busca futuras mejoras en esta área, al mismo tiempo que considera la optimización del equipo, dado que este puede constituir una variable que afecta el proceso de entrenamiento.

INTRODUCCIÓN

La construcción de sociedades inclusivas y equitativas se fundamenta en la comunicación efectiva. Sin embargo, la comunidad de personas sordas en México enfrenta un desafío persistente: la barrera lingüística representada por el Lenguaje de Señas Mexicano (LSM). La carencia de herramientas accesibles y precisas para interpretar y comprender el LSM en tiempo real ha generado limitaciones en la fluidez comunicativa, creando una brecha en la interacción diaria.

Para abordar esta problemática, este estudio propone el diseño y desarrollo de un algoritmo basado en redes neuronales y técnicas de visión artificial. El objetivo primordial es lograr un reconocimiento eficaz del LSM, permitiendo la interpretación automática de las señas en imágenes capturadas en tiempo real. Al alcanzar este objetivo, se busca superar las limitaciones actuales y proporcionar a la comunidad sorda una herramienta innovadora que mejore significativamente su capacidad de comunicación.

Esta investigación no solo se centra en cerrar el vacío tecnológico en el reconocimiento del LSM, sino que también se sitúa en la intersección de la Mecatrónica, la visión artificial y la accesibilidad. Al comprender a fondo la estructura y gramática del LSM, adquirir un conjunto de datos representativo y diseñar una red neuronal adecuada, este estudio busca ofrecer una solución integral que no solo cumpla con los estándares de precisión, sino que también sea aplicable en entornos prácticos.

Mediante este enfoque multidisciplinario, se busca no solo mejorar la comunicación de las personas sordas, sino también establecer un precedente para el desarrollo y la aplicación de tecnologías asistivas en el contexto mexicano. En última instancia, este trabajo aspira a contribuir al avance de la inclusión social, promoviendo una sociedad que reconoce y valora la diversidad lingüística y cultural.

ÍNDICE GENERAL

PRELIMINARES	2
AGRADECIMIENTOS	3
RESUMEN	4
INTRODUCCIÓN	5
ÍNDICE GENERAL	6
ÍNDICE DE ILUSTRACIONES	9
CAPÍTULO I	11
1.1 DESCRIPCIÓN DE LA EMPRESA U ORGANIZACIÓN Y DEL PUESTO O ÁREA DEL TRABAJO DEL ESTUDIANTE	12
1.1.1 Datos generales de la empresa	12
1.1.2 Misión	13
1.1.3 Visión	13
1.1.4 Estructura orgánica	13
1.1.6 Infraestructura	14
1.2 DESCRIPCIÓN DEL PUESTO O ÁREA DE TRABAJO DEL ESTUDIANTE	15
1.3 PROBLEMAS DE INVESTIGACIÓN A RESOLVER, PRIORIZANDOLOS	15
1.4 PREGUNTAS DE INVESTIGACIÓN	16
1.4.1 Pregunta de investigación general	16
1.4.2 Preguntas de investigación específicas	16
1.5 OBJETIVOS	17
1.5.1 Objetivo general	17
1.5.2 Objetivos específicos	17
1.6 JUSTIFICACIÓN DE LA INVESTIGACIÓN	17
CAPÍTULO II	19
2.1 FUNDAMENTOS TEÓRICOS	20
2.1.1 Lenguaje de Señas Mexicano (LSM)	20
2.1.1.1 Gramática de LSM	20

2.1.2 Visión artificial	21
2.1.3 Procesamiento digital de imágenes	23
2.1.4 Teoría y métodos del PDI	24
2.1.5 Adquisición	25
2.1.6 Restauración y reconstrucción	26
2.1.7 Reconocimiento de patrones.....	26
2.1.8 Interpretación física	31
2.1.9 Aprendizaje profundo y Redes Neuronales Convolucionales (CNN) ...	31
2.1.10 Librerías y herramientas	36
2.1.11 Evaluación de modelo.....	38
2.1.12 Interfaces de Usuario y usabilidad en la detección de LSM.....	39
CAPÍTULO III	41
3.1 PROCEDIMIENTO Y DESCRIPCIÓN DE LAS ACTIVIDADES REALIZADAS	42
3.2 ALCANCE Y ENFOQUE DE LA INVESTIGACIÓN.....	43
3.2.1 Alcances	43
3.3 HIPÓTESIS.....	44
3.4 DISEÑO Y METODOLOGÍA DE LA INVESTIGACIÓN	44
3.5 SELECCIÓN DE LA MUESTRA	44
3.6 RECOLECCIÓN DE DATOS	44
3.6.1 SELECCIÓN DEL INSTRUMENTO	45
3.6.2 APLICACIÓN DE INSTRUMENTO	45
3.6.2.1 Revisión de literatura	45
3.6.2.2 Desarrollo de software	49
CAPÍTULO IV	63
4.1 RESULTADOS.....	64
CAPÍTULO V	71
5.1 CONCLUSIONES DEL PROYECTO, RECOMENDACIONES, EXPERIENCIA PROFESIONAL Y PERSONAL ADQUIRIDA.	72

5.2 CONCLUSIONES RELATIVAS AL OBJETIVO GENERAL	72
5.3 LIMITACIONES DEL MODELO PLANTEADO	73
5.4 RECOMENDACIONES	73
CAPÍTULO VI.....	74
6.1 COMPETENCIAS DESARROLLADAS Y/O APLICADAS	75
CAPÍTULO VII	76
7.1 FUENTES DE INFORMACIÓN.....	77
CAPÍTULO VIII.....	80
ANEXO 1:	81
ANEXO 2:	82
ANEXO 3:	83
ANEXO 4	84
ANEXO 2	87
ANEXO 3	99

Índice de ilustraciones

Ilustración 1. Estructura Orgánica ITST.....	14
Ilustración 2. Abecedario de lenguaje de señas	21
Ilustración 3. Adquisición de imagen.....	25
Ilustración 4. Deconvolución de imagen.	26
Ilustración 5. Realce de características.....	27
Ilustración 6. Transformaciones exponenciales	27
Ilustración 7. Nivel de gris en igualación	28
Ilustración 8. Eliminación de ruido	29
Ilustración 9. Niveles de gris en dos tipos de imágenes.....	29
Ilustración 10. . Niveles de gris en dos tipos de imágenes.....	30
Ilustración 11. Detección de bordes en la detección de gradientes en los niveles de gris.....	30
Ilustración 12. Detección de bordes en la detección de gradientes en los niveles de gris.....	31
Ilustración 13. Capas en una red neuronal convolucional (CNN)	33
Ilustración 14. Muestra de la toma de fotografías.	49
Ilustración 15. Etiquetado de imágenes.	50
Ilustración 16. Clasificado de imágenes.....	50
Ilustración 17. Data set en la nube.	51
Ilustración 18. Conexión de Data set con Google Colab.....	51
Ilustración 19. Código de entrenamiento de red neuronal en Google Colab.	52
Ilustración 20. Código de entrenamiento de red neuronal en Google Colab.	52
Ilustración 21. Código de entrenamiento de red neuronal en Google Colab.	53
Ilustración 22. Código de entrenamiento de red neuronal en Google Colab.	53
Ilustración 23. Código de entrenamiento de red neuronal en Google Colab.	54
Ilustración 24. Ejecución de código, muestra imágenes aleatorias.	54
Ilustración 25. Código de entrenamiento de red neuronal en Google Colab.	55
Ilustración 26. Código de entrenamiento de red neuronal en Google Colab.	55
Ilustración 27. Código de entrenamiento de red neuronal en Google Colab.	55
Ilustración 28. Código de entrenamiento de red neuronal en Google Colab.	56
Ilustración 29. Código de entrenamiento de red neuronal en Google Colab.	56
Ilustración 30. Código de entrenamiento de red neuronal en Google Colab.	57
Ilustración 31. Código de entrenamiento de red neuronal en Google Colab.	57
Ilustración 32. Código de descarga de modelo entrenado en Google Colab.	58
Ilustración 33. Código de interfaz de usuario en PyCharm Community.	58
Ilustración 34. Código de interfaz de usuario en PyCharm Community.....	59
Ilustración 35. Código de interfaz de usuario en PyCharm Community.....	59

Ilustración 36. Código de interfaz de usuario en PyCharm Community.	60
Ilustración 37. Código de interfaz de usuario en PyCharm Community.	60
Ilustración 38. Código de interfaz de usuario en PyCharm Community.	61
Ilustración 39. Código de interfaz de usuario en PyCharm Community.	61
Ilustración 40. Código de interfaz de usuario en PyCharm Community.	62
Ilustración 41. Gráfica de precisión y pérdida de entrenamiento y pruebas a 10 épocas.	64
Ilustración 42. Gráfica de precisión y pérdida de entrenamiento y pruebas a 12 épocas.	65
Ilustración 43. Gráfica de precisión y pérdida de entrenamiento y pruebas a 15 épocas.	66
Ilustración 44. Gráfica de precisión y pérdida de entrenamiento y pruebas a 25 épocas.	67
Ilustración 45. Gráfica de precisión y pérdida de entrenamiento y pruebas a 50 épocas.	67
Ilustración 46. Gráfica de precisión y pérdida de entrenamiento y pruebas a 250 épocas.	68
Ilustración 47. Gráfica de precisión y pérdida de entrenamiento y pruebas a 30 épocas.	69
Ilustración 48. Gráfica de precisión y pérdida de entrenamiento y pruebas a 100 épocas.	70
Ilustración 49. Gráfica de precisión y pérdida de entrenamiento y pruebas a 150 épocas.	70

CAPÍTULO I

GENERALIDADES DEL PROYECTO

1.1 Descripción de la empresa u organización y del puesto o área del trabajo del estudiante

Teziutlán ha desempeñado un papel histórico como centro de desarrollo económico en la región Nororiental del Estado de Puebla. Su crecimiento se ha sustentado en diversas etapas, inicialmente en la industria minera y metalúrgica, seguida por la fruticultura, la ganadería y, más recientemente, la industria de confección de ropa. La actividad industrial ha propiciado el desarrollo de otras áreas económicas, como el comercio, el transporte, los servicios financieros y, de manera destacada, la educación.

El Instituto Tecnológico Superior de Teziutlán, una entidad pública descentralizada del Gobierno del Estado de Puebla con autonomía legal y recursos independientes, se fundó mediante una resolución del Congreso del Estado de Puebla el 8 de noviembre de 1994, difundida en el Periódico Oficial de la región.

El Instituto Tecnológico Superior de Teziutlán, en sintonía con las demandas de la sociedad y los principios establecidos en la Ley de Educación del Estado de Puebla, se consolida como una institución cuyo propósito es proporcionar educación de calidad, moderna y eficaz.

1.1.1 Datos generales de la empresa

Las opciones académicas actualmente disponibles en el Instituto Tecnológico Superior de Teziutlán son:

- Licenciatura en Turismo.
- Ingeniería en Gestión Empresarial.
- Ingeniería en Industrias Alimentarias.
- Ingeniería en Sistemas Computacionales.
- Ingeniería Industrial.
- Ingeniería Informática.
- Ingeniería Mecatrónica.

- Ingeniería Animación y efectos visuales.
- Maestría en sistemas computacionales.
- Maestría en ingeniería industrial.

El instituto ofrece una educación de primer nivel, actualizada y eficaz, centrada en satisfacer las necesidades e intereses individuales, fomentando el uso efectivo y transparente de los recursos humanos, materiales y financieros disponibles, y garantizando la puntualidad en sus operaciones.

Dirección

Fracción I y II S/N, Aire Libre, Teziutlán, Puebla; C.P. 73960.

Giro de la empresa

Es una institución de educación superior pública con giro Terciario.

1.1.2 Misión

El Instituto Tecnológico Superior de Teziutlán tiene como misión formar profesionales que se contribuyan en agentes de cambio y promuevan el desarrollo integral de la sociedad mediante la implementación de procesos académicos de calidad.

1.1.3 Visión

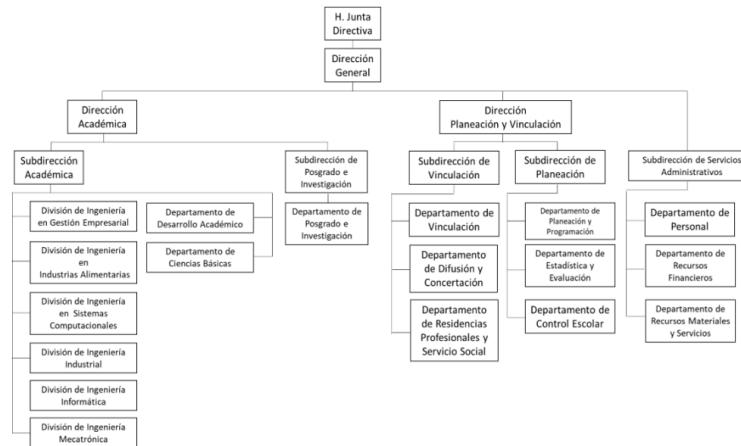
Llegar a ser la Institución de Educación Superior Tecnológica más reconocida en el Estado de Puebla que ofrezca un proceso de enseñanza – aprendizaje certificado, comprometido con la excelencia y la formación integral del alumno contribuyendo al desarrollo sustentable, económico político y social de nuestro Estado.

1.1.4 Estructura orgánica

La Dirección General del Instituto Tecnológico superior de Teziutlán se encuentra a cargo de la Mtra. Arminda Juárez Arroyo.

Como jefe de la división de ingeniería Mecatrónica y asesor externo del proyecto de residencia profesional se encuentra a cargo el M. C. Julio Cesar Camargo Santos.

Ilustración 1. Estructura Orgánica ITST



Fuente: Instituto Tecnológico Superior de Teziutlán

1.1.6 Infraestructura

El terreno asignado al Instituto abarca una superficie de 12 hectáreas, que alberga un complejo de seis edificios compuestos por:

1. Edificio de Unidad Administrativa
2. Laboratorio de Química
3. Edificio de Biblioteca, Dirección General
4. Edificio de Unidad Académica
5. Edificio Multifuncional de Talleres y Laboratorios
 - Laboratorio de Simulación
 - Laboratorio de Manufactura
 - Laboratorio de Electricidad
 - Laboratorio de Mecatrónica
 - Laboratorio de Ergonomía
 - Laboratorio de Electrónica
 - Laboratorio de Mercadotecnia
 - Laboratorio de Ciencias Básicas
6. Clases

1.2 Descripción del puesto o área de trabajo del estudiante

La División de Ingeniería Mecatrónica, liderada por un jefe de carrera y conformada por un equipo de profesores, tiene como objetivo principal orientar a los estudiantes hacia el éxito en el ámbito de la ingeniería. Un egresado de Ingeniería Mecatrónica adquirirá la capacidad de analizar, evaluar y aportar innovaciones, lo que le permitirá diseñar, planificar, construir, innovar y gestionar sistemas de control y automatización industrial mediante el uso de métodos avanzados. Este enfoque tiene como finalidad satisfacer de manera racional, eficiente, económica y sostenible las necesidades de los sectores de producción y servicios, optimizando el uso de los recursos.

1.3 PROBLEMAS DE INVESTIGACIÓN A RESOLVER, PRIORIZANDOLOS

El Lenguaje de Señas Mexicano (LSM) juega un papel fundamental en la comunicación de la comunidad de personas sordas en México. A pesar de su importancia, el LSM enfrenta desafíos en su comprensión y difusión, limitando la inclusión y la efectiva comunicación de las personas sordas en la sociedad. Un obstáculo central radica en la capacidad de interpretar y entender el LSM de manera precisa y en tiempo real, tanto para personas sordas como para aquellas no familiarizadas con este lenguaje.

Para superar esta barrera, la tecnología de reconocimiento de lenguaje de señas se presenta como una solución prometedora. Sin embargo, hasta ahora, no existen soluciones ampliamente disponibles y accesibles que sean efectivas y precisas. Esta carencia tecnológica restringe la fluidez de la comunicación en la sociedad predominada por el lenguaje hablado.

La investigación se centra en la falta de una herramienta eficaz y precisa para identificar y comprender el LSM en tiempo real. Para enfrentar este problema, se

propone desarrollar un algoritmo basado en redes neuronales y técnicas de visión artificial. Este algoritmo busca identificar el LSM en imágenes capturadas en tiempo real, con el potencial de convertir automáticamente las señas en texto, facilitando la comunicación entre personas sordas y aquellos no familiarizados con el LSM.

Los objetivos específicos incluyen profundizar en el conocimiento del LSM, adquirir datos de entrenamiento adecuados, diseñar y entrenar una red neuronal, y realizar pruebas para evaluar la precisión y eficacia del modelo propuesto. Resolver este problema beneficiaría no solo a la comunidad de personas sordas en México, sino que también tendría un impacto positivo en la sociedad en su conjunto al promover la inclusión y la comunicación efectiva entre personas sordas y oyentes.

1.4 PREGUNTAS DE INVESTIGACIÓN

1.4.1 Pregunta de investigación general

¿Cómo se puede diseñar una aplicación basada en redes neuronales y técnicas de visión artificial, haciendo uso de la librería OpenCV en el entorno de programación Python 3.10, con el propósito de facilitar la identificación del lenguaje de señas mexicano?

1.4.2 Preguntas de investigación específicas

- A. ¿Cuál es la estructura y gramática del LSM y cómo se puede representar de manera efectiva para su reconocimiento?
- B. ¿Cuál es la topología de red neuronal más adecuada para el reconocimiento de señas en imágenes en tiempo real?
- C. ¿Cómo impactará esta tecnología en la comunicación y la inclusión de la comunidad de personas sordas en México?

1.5 OBJETIVOS

1.5.1 Objetivo general

Diseñar una aplicación con redes neuronales mediante el empleo de técnicas de visión artificial utilizando la librería de OpenCV para el lenguaje de programación Python 3.10, que permita la identificación del lenguaje de señas mexicano.

1.5.2 Objetivos específicos

- Explorar y conocer el lenguaje de señas mexicano.
- Obtener la data set para el entrenamiento de la red neuronal.
- Implementar transfer learning utilizando modelo pre entrenado.
- Realizar el entrenamiento de la red neuronal utilizando el conjunto de datos.
- Diseñar una aplicación que integre el modelo de reconocimiento de señas.

1.6 JUSTIFICACIÓN DE LA INVESTIGACIÓN

El Lenguaje de Señas Mexicana (LSM) cumple una función vital en la comunicación de la comunidad de personas sordas en México. A pesar de su importancia, las barreras existentes en la comprensión y difusión del LSM limitan de manera significativa la participación y la integración de las personas sordas en la sociedad. La carencia de herramientas efectivas y accesibles para interpretar y comprender el LSM en tiempo real representa un desafío considerable que este estudio se propone abordar.

Solo en México, alrededor de un total de 2.3 millones de individuos presentan discapacidad auditiva en el país, siendo más de la mitad de ellos mayores de 60 años. Además, alrededor del 34% se encuentran en el rango de edad de 30 a 59 años, mientras que un 2% corresponde a niños y niñas. (Secretaría de Salud, 2021). La implementación de un algoritmo de reconocimiento del LSM mediante el uso de redes neuronales y visión artificial emerge como una solución prometedora para superar estas barreras de comunicación.

El propósito fundamental de este proyecto es desarrollar un algoritmo de red neuronal destinado a la identificación de la Lengua de Señas Mexicana mediante el

empleo de técnicas de visión artificial a través de la biblioteca OpenCV, en el marco del lenguaje de programación Python 3.10.

Los beneficiarios directos comprenden a las personas con discapacidad auditiva, ya que el algoritmo se erige como una herramienta efectiva de comunicación que facilita la expresión de ideas mediante gestos en la Lengua de Señas Mexicana. Asimismo, la comunidad de usuarios de lengua de signos se verá directamente favorecida al contar con tecnología avanzada que interpreta y analiza sus gestos de manera eficaz.

La escalabilidad del proyecto permite futuras aplicaciones y adaptaciones, buscando mejorar la calidad de vida y promover el desarrollo social y tecnológico en accesibilidad y comunicación inclusiva.

CAPÍTULO II

MARCO TEÓRICO

2.1 FUNDAMENTOS TEÓRICOS

2.1.1 Lenguaje de Señas Mexicano (LSM)

El término correcto es "lengua de señas", aunque comúnmente se conoce como "lenguaje de señas". Esta lengua constituye el lenguaje natural de las personas sordas, posee una historia, gramática y estructura propias y distintivas.

El Lengua de Señas Mexicana (LSM) es utilizado por la comunidad de sordos en México como un sistema de comunicación basado en gestos manuales, expresiones faciales, miradas intencionales y movimientos corporales. Este método de comunicación posee una estructura lingüística y un vocabulario tan completo y sofisticado como cualquier lengua hablada, considerándose un valioso patrimonio lingüístico de esta comunidad.

La comunicación desempeña un papel fundamental en la comprensión humana. En el ámbito educativo de personas sordas, un hito relevante se relaciona con la institución de la Escuela de Sordomudos y la posterior fundación de la Escuela Nacional de Sordomudos (ENS) fueron llevadas a cabo por decreto presidencial de Benito Juárez. Su propósito principal era capacitar maestros especializados en la instrucción de individuos con discapacidad auditiva.. El 15 de abril de 1861, el Presidente Interino Constitucional de México, Benito Juárez, emitió una ley de educación que contemplaba la creación de una escuela para personas sordas en el artículo tercero del apartado de instrucción primaria (Consejo Nacional para el Desarrollo y la Inclusión de las Personas con Discapacidad, 2019).

2.1.1.1 Gramática de LSM

Según la Real Academia Española (2014), la gramática es una parte de la lingüística que se dedica al estudio de los elementos de una lengua y cómo se organizan y combinan. También abarca la representación lingüística de los hablantes, especialmente en términos de morfología, sintaxis y ciertos aspectos del léxico. En el caso del lenguaje de señas, la gramática se fundamenta en la ubicación en el espacio y la estructura de las manos, así como en otros gestos.

La Lengua de Señas Mexicana (LSM) se organiza utilizando el cuerpo en el espacio. Dentro de este espacio definido, se establecen tres límites: uno vertical, que va desde la cintura hasta la coronilla de la cabeza; otro horizontal, que se extiende hasta la altura de los codos con los brazos doblados; y un tercero que indica la cercanía de las manos con respecto al cuerpo. Cualquier movimiento que sobrepase estas dimensiones será percibido como exagerado o enfatizado. (Cruz Aldrete y otros, 2017).

Ilustración 2. Abecedario de lenguaje de señas



Fuente: Gobierno de México

2.1.2 Visión artificial

La visión artificial, también llamada visión por computadora, es un campo de estudio que abarca todos los procesos y elementos para programar una computadora con el fin de analizar y comprender imágenes y videos. Implica la recopilación de datos de un fotograma o secuencia de video y su objetivo es transformar la información visual en datos útiles para la toma de decisiones o para generar una representación diferente de la imagen original., según exponen Alvear Puertas, Rosero Montalvo, Peluffo Ordóñez y Pijal Rojas (2017).

Las etapas en un proceso de visión artificial siguen un orden específico. La tabla (no proporcionada) muestra la secuencia en la cual se ejecuta la visión artificial, desde la obtención de la imagen hasta el procesamiento y la interpretación.

Tabla 1. Proceso de la visión artificial

Procesos	Nivel de visión	Entrada	Salida	Área
1. Captura	Bajo	Imagen	Imagen	Procesamiento de imágenes
2. Preprocesamiento				
3. Segmentación	Medio	Imagen	Grupos de pixeles en bruto(objetos o regiones)	Análisis de imágenes
4. Descripción		Objetos o regiones	Información cuantitativa de los objetos o regiones	
5. Reconocimiento (clasificación)		Información cuantitativa	Objetos clasificados en categorías	
6. Interpretación	Alto	Objetos clasificados en categorías	Comprensión de la escena	Visión por computador

Fuente: <http://scielo.senescyt.gob.ec/pdf/enfoqueute/v8s1/1390-6542-enfoqueute-8-s1-00244.pdf>

De acuerdo a (García Santillán & Caranqui Sánchez, 2015) cada etapa consiste en lo siguiente:

1. Captura: En esta etapa, se adquiere una representación visual digital mediante dispositivos tales como cámaras o escáneres., telescopios, satélites, entre otros.
2. Pre-procesamiento: En esta fase, se aplican técnicas destinadas a mejorar la calidad de la imagen. Estas técnicas pueden abordar aspectos tales como la

eliminación de ruido, la modificación del contraste, la claridad, el resaltado de características y la recuperación de la imagen.

3. Segmentación: Este paso, la imagen se divide en objetos o regiones de interés. Esta etapa es crucial para el éxito de la aplicación, ya que define la forma en que se separan los elementos en la imagen.
4. Descripción: Durante este proceso, se extraen características relevantes de los objetos identificados. Estas propiedades pueden manifestarse como atributos externos, como la forma y el tamaño, o como características internas, tales como el área, el centro de masa, texturas y atributos de color.
5. Reconocimiento: El reconocimiento implica clasificar los objetos en categorías utilizando las características obtenidas en la etapa de descripción. Los objetos con características similares se agrupan en la misma categoría, a menudo utilizando técnicas avanzadas como estadísticas, redes neuronales o algoritmos genéticos.
6. Interpretación: En esta última etapa, se da sentido a las categorías de objetos reconocidos, lo que permite comprender la escena. La interpretación puede variar según la aplicación específica de la visión artificial y puede involucrar técnicas cognitivas para la toma de decisiones.

Estos procesos suelen seguir un orden secuencial, y los resultados de una etapa se utilizan como entrada para la siguiente. Sin embargo, no todos los procesos son necesarios en todos los casos, ya que su aplicación depende de la complejidad del problema a resolver.

2.1.3 Procesamiento digital de imágenes

El procesamiento digital de imágenes (PDI) engloba un conjunto de métodos y procesos utilizados para detectar y destacar información contenida en una imagen, siendo la computadora el recurso primordial en esta tarea. En la actualidad, el procesamiento digital de imágenes (PDI) constituye un campo de investigación altamente especializado dentro del ámbito informático y guarda una estrecha relación con el procesamiento digital de señales. Esta relación se fundamenta en el

hecho de que, esencialmente, el PDI se presenta como una forma específica de procesamiento digital de señales, solo que aplicado a datos visuales en dos o tres dimensiones. (Torres Domínguez, 1996).

En la actualidad, el Procesamiento Digital de Imágenes (PDI) tiene diversas aplicaciones, y a continuación se presentan algunos de sus usos principales:

- Cartografía y Sistemas de Información Geográfica (SIG): Los satélites capturan imágenes de la Tierra, y al someter estas imágenes a procesamiento, se genera información detallada y precisa acerca de nuestro planeta.
- Aplicaciones en la Medicina: Profesionales en radiología y laboratorios médicos utilizan imágenes digitales generadas por tecnologías como la Tomografía Asistida por Computadora (TAC), resonancia magnética y métodos de medicina nuclear.
- Visualización en las Ciencias: Dentro de este ámbito, se exploran disciplinas como la aerodinámica, la dinámica de fluidos, la modelación molecular y otras áreas de estudio relacionadas.
- Otras Aplicaciones: El PDI experimenta un crecimiento constante gracias a la accesibilidad de equipos (software y hardware) que no son tan costosos. Esto permite explorar nuevas disciplinas como grabado, metalurgia, reconocimiento de objetos, microscopía, percepción remota, seguridad fotográfica, impresión de huellas digitales, metamorfosis de imágenes, lectura de documentos, crecimiento de baterías, modelación geológica, navegación autónoma, astronomía, mecánica de fluidos, mapas climatológicos, CAD, y transferencia y comparación de documentos.

2.1.4 Teoría y métodos del PDI

El procesamiento digital de imágenes (PDI) se vale de diversas técnicas provenientes de las ramas de las matemáticas puras y aplicadas. Las áreas matemáticas fundamentales en este contexto abarcan principalmente el álgebra lineal, cálculo en una y múltiples dimensiones, ecuaciones diferenciales ordinarias y parciales,

ecuaciones integrales (incluyendo transformadas integrales), así como probabilidad y estadística. Estos campos matemáticos se emplean de diversas formas para abordar tres problemas fundamentales comunes en muchas áreas de aplicación del procesamiento digital de imágenes:

1. Adquisición
2. Restauración y reconstrucción de imágenes.
3. Reconocimiento de patrones.
4. Interpretación física.

2.1.5 Adquisición

En la actualidad, es posible adquirir imágenes digitales a través de diferentes medios, siendo los principales el escáner y las cámaras digitales. Ambos sistemas se basan en el uso de un componente conocido como CCD (Coupled Charge Device, por sus siglas en inglés). Este dispositivo recibe la luz de la imagen, ya sea por reflexión o transmisión, e integra la cantidad de luz que llega a él en un tiempo definido. Al formar un arreglo de CCD, es posible realizar la digitalización de la imagen por renglones o en su totalidad (Martín Ortiz, 2013). Mismo ejemplo se muestra en la imagen de a continuación.

Ilustración 3. Adquisición de imagen

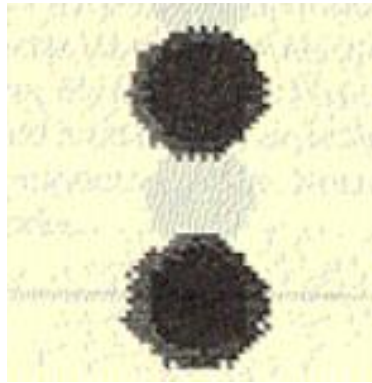


Fuente: [Redalyc.Procesamiento digital de imágenes](#)

2.1.6 Restauración y reconstrucción

Este método, como su nombre sugiere, implica la generación de una imagen de tal manera que la discrepancia entre esta y la imagen original sea mínima, en términos de mínimos cuadrados. Para ejecutar esta deconvolución de forma eficiente desde una perspectiva matemática y computacional, se emplea el algoritmo conocido como transformada rápida de Fourier.

Ilustración 4. Deconvolución de imagen.



Fuente: [Redalyc.Procesamiento digital de imágenes](#)

2.1.7 Reconocimiento de patrones

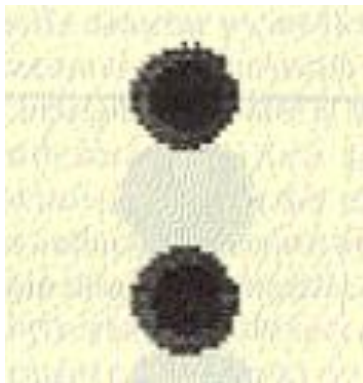
En el procesamiento digital de imágenes (PDI), la fase de reconocimiento de patrones se enfoca en la modificación de la imagen obtenida tras el proceso de deconvolución. Durante este procedimiento, se ajustan los niveles de gris con el fin de mejorar la legibilidad de la imagen y detectar características que puedan ser físicamente relevantes. Esta manipulación se lleva a cabo mediante tres subprocesos: realce de la imagen, reducción de ruido y segmentación y detección de bordes. Este proceso es bastante complejo y guarda similitudes con la ilustración 4.

El resalta miento de imágenes tiene como objetivo mejorar visualmente la apariencia de la imagen. No obstante, este proceso es subjetivo, ya que la mejora visual depende de los detalles y contrastes que el usuario busca obtener. Estas características, a su vez, están vinculadas a las propiedades físicas de la imagen y

al conocimiento previo que el usuario pueda tener sobre ciertas características de la misma. Existen diversas técnicas para resaltar imágenes, seleccionadas según los resultados deseados. Dos de estas técnicas son:

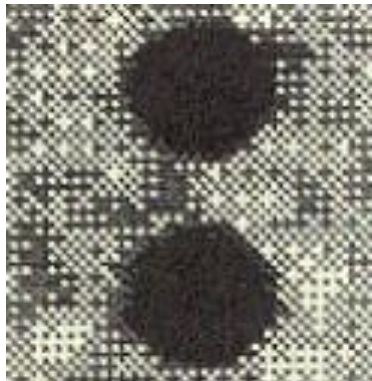
- Transformaciones sencillas: se aplican con el propósito de realzar características en una imagen que se encuentran en áreas más oscuras o más luminosas. Estas incluyen transformaciones logarítmicas, que destacan zonas oscuras (ver figura 5), y transformaciones exponenciales, empleadas para el proceso opuesto (ver figura 6).

Ilustración 5. Realce de características.



Fuente: [Redalyc.Procesamiento digital de imágenes](#)

Ilustración 6. Transformaciones exponenciales



Fuente: [Redalyc.Procesamiento digital de imágenes](#)

- Igualación de Histogramas: La igualación de histogramas tiene como objetivo generar una imagen en la cual el histograma sea uniforme, es decir, que cada

nivel de gris aparezca con la misma frecuencia. Este proceso se denomina "igualación" debido a la distribución uniforme resultante. El efecto es un aumento en el rango dinámico de los niveles de gris. Este método tiene un impacto significativo en la apariencia visual de la imagen al resaltar los detalles en las regiones tanto oscuras como luminosas. Su principal función es mejorar la calidad visual mediante la uniformización del histograma.

- Reducción de Ruido: El objetivo principal de la reducción de ruido es optimizar la calidad visual de una imagen al eliminar perturbaciones aleatorias no correlacionadas. En la práctica, el ruido tiende a afectar las frecuencias altas en los datos, donde la amplitud normalmente es baja. Por lo tanto, la reducción de ruido implica atenuar las componentes de altas frecuencias en un rango seleccionado y ajustado por el usuario. Esto se traduce en una mejora visual al eliminar distracciones no deseadas y mejorar la claridad de la imagen.

Ilustración 7. Nivel de gris en igualación



Fuente: Redalyc.Procesamiento digital de imágenes

Ilustración 8. Eliminación de ruido

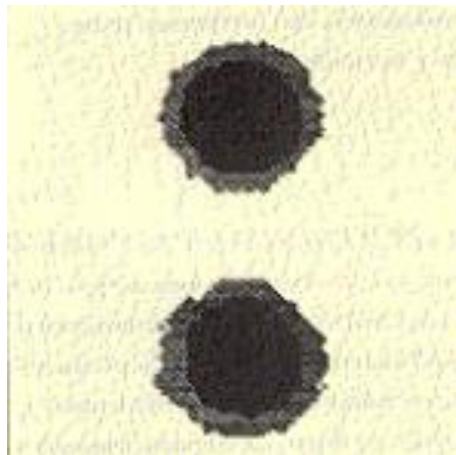


Fuente: [Redalyc.Procesamiento digital de imágenes](#)

La segmentación conlleva la partición de una imagen en segmentos o regiones que pueden ser manipuladas. Este proceso se emplea en el análisis de imágenes para distinguir propiedades o áreas específicas dentro de una textura dada. La segmentación es esencial para la clasificación de imágenes y representa el primer paso en el análisis automático de imágenes y el reconocimiento de patrones. Se basa en una de dos características principales de la imagen:

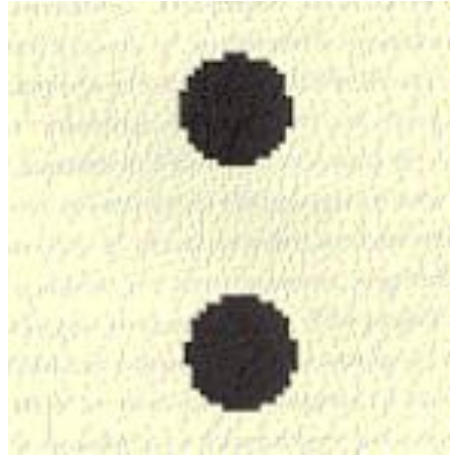
- Similitud: Divide la imagen en áreas cuyos niveles de gris se encuentran dentro de un intervalo predefinido.

Ilustración 9. Niveles de gris en dos tipos de imágenes.



Fuente: [Redalyc.Procesamiento digital de imágenes](#)

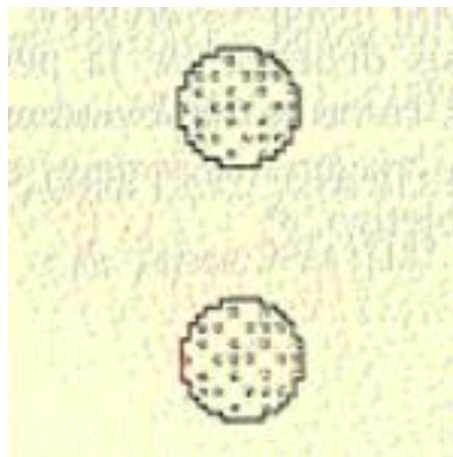
Ilustración 10. . Niveles de gris en dos tipos de imágenes.



Fuente: [Redalyc.Procesamiento digital de imágenes](#)

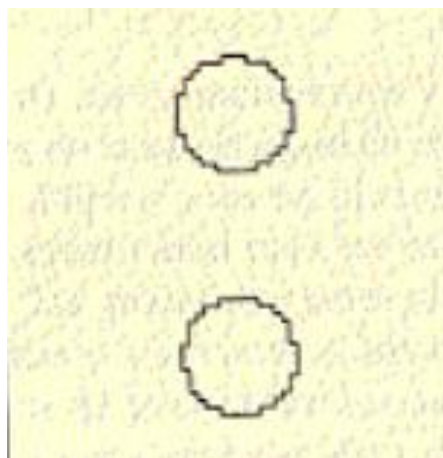
- **Discontinuidad:** La detección de bordes en el procesamiento digital de imágenes consiste en separar la imagen en áreas de discontinuidad donde se produce una transición abrupta en los valores de los niveles de gris. Este método se utiliza para identificar los límites o bordes en la imagen. En términos simples, la detección de bordes emplea operadores diferenciales para identificar variaciones en los gradientes de los niveles de gris.

Ilustración 11. Detección de bordes en la detección de gradientes en los niveles de gris.



Fuente: [Redalyc.Procesamiento digital de imágenes](#)

*Ilustración 12. Detección de bordes en la detección de gradientes
en los niveles de gris.*



Fuente: [Redalyc.Procesamiento digital de imágenes](#)

2.1.8 Interpretación física

En resumen, el reconocimiento de patrones tiene como objetivo identificar las características en una imagen que tienen un significado físico. La interpretación física de una imagen puede ser difícil de cuantificar, ya que los objetivos y expectativas de este aspecto del procesamiento digital de imágenes están estrechamente relacionados con el diseño y la aplicación de un sistema de manipulación de imágenes. En esencia, el objetivo principal es interpretar una imagen en función de las propiedades físicas del campo de radiación electromagnética del cual se deriva la imagen, para luego deducir información sobre la estructura y propiedades del material de los objetos capturados en la imagen.

2.1.9 Aprendizaje profundo y Redes Neuronales Convolucionales (CNN)

El aprendizaje profundo es una sub disciplina del aprendizaje automático (machine learning, ML) en la cual las redes neuronales artificiales, diseñadas para emular el funcionamiento del cerebro humano, aprenden a partir de grandes volúmenes de datos. (Oracle, 2023).

El aprendizaje profundo se fundamenta en capas de redes neuronales, algoritmos que se inspiran vagamente en el funcionamiento del cerebro humano. La configuración de las neuronas en la red neuronal se logra mediante el entrenamiento con extensos conjuntos de datos, lo que resulta en un modelo de aprendizaje profundo capaz de procesar nuevos datos una vez que ha sido entrenado. Estos modelos son capaces de recopilar información de diversas fuentes y analizar datos en tiempo real sin la necesidad de intervención humana. En el aprendizaje profundo, las unidades de procesamiento gráfico (GPU) son optimizadas para el entrenamiento de modelos al llevar a cabo múltiples cálculos simultáneamente. En términos simples, el aprendizaje profundo se refiere a redes neuronales con múltiples capas. Estas redes transfieren datos, como imágenes o sonido, a través de nodos interconectados en capas. En cada capa, los nodos realizan operaciones simples en los datos y transfieren los resultados selectivamente a otros nodos.

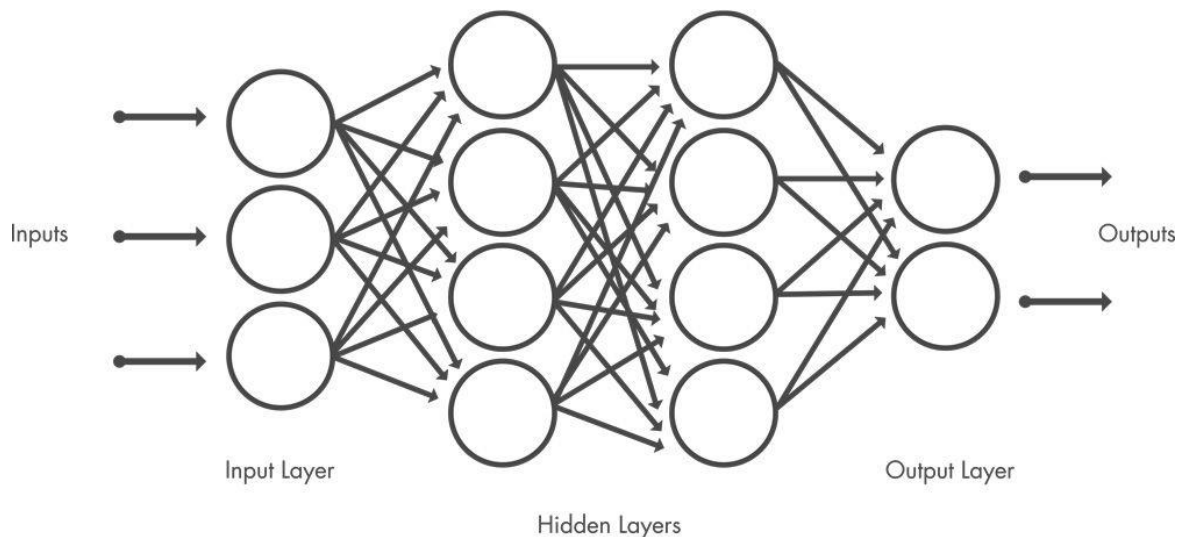
Las redes neuronales básicas pueden tener una o dos capas ocultas entre la entrada y la salida. En contraste, el aprendizaje profundo se distingue por tener muchas capas ocultas, a veces decenas o incluso cientos. Aumentar el número de capas y nodos puede mejorar la precisión, pero también implica más parámetros y recursos computacionales.

Durante el entrenamiento de un modelo de aprendizaje profundo, se procesan las entradas a través de capas ocultas utilizando pesos que se ajustan iterativamente durante el proceso de entrenamiento. Estos pesos representan la fuerza de la conexión entre las entradas, y el modelo los ajusta en función de los datos de entrenamiento para mejorar las predicciones. Debido al procesamiento intensivo, el entrenamiento de modelos de aprendizaje profundo requiere recursos informáticos de alto rendimiento.

En el aprendizaje profundo, se pueden utilizar diferentes tipos de redes neuronales según la aplicación. Las redes de propagación hacia adelante, también conocidas como perceptrones multicapa (MLP), son ampliamente utilizadas y están compuestas por capas de entrada, capas ocultas y capas de salida. Además, las redes neuronales

convolucionales (CNN) son particularmente eficaces para tareas como el reconocimiento de imágenes, el reconocimiento de patrones y la visión artificial. Estas redes emplean filtros para detectar características en diferentes resoluciones de una imagen, y a medida que avanzan en las capas, se especializan cada vez más en identificar características específicas del objeto en cuestión. (MathWorks, s.f.; IBM, 2021).

Ilustración 13. Capas en una red neuronal convolucional (CNN)



Fuente: [¿Qué es una red neuronal convolucional? / 3 cosas que debe saber - MATLAB & Simulink \(mathworks.com\)](#)

Estas capas realizan operaciones diseñadas para transformar los datos con el fin de comprender sus características específicas. Las tres capas más usuales, según (IBM, ¿Qué son las redes neuronales convolucionales?, s.f.), son: la capa de convolución, la capa de agrupamiento y la capa totalmente conectada (FC).

- **Capa convolucional:** La capa convolucional es fundamental en una Convolutional Neural Network (CNN) y constituye la etapa central donde se realiza la mayor parte del cálculo. Requiere ciertos elementos, como datos de entrada, un filtro y un mapa de características. El detector de características, también denominado kernel o filtro, se mueve a través de campos receptivos en la imagen para detectar la presencia de características, en un proceso conocido como convolución.

El detector de características es una matriz bidimensional (2-D) de pesos que representa una parte de la imagen. Por lo general, el tamaño del filtro es una matriz de 3×3 , lo que determina el tamaño del campo receptivo. El filtro se aplica a un área de la imagen y se calcula el producto punto entre los píxeles de entrada y el filtro. El resultado se introduce en una matriz de salida, y el filtro se desplaza con pasos para cubrir toda la imagen. El resultado final se conoce como mapa de características, mapa de activación o característica convolucionada.

Después de cada operación de convolución, la CNN aplica una función de activación de Unidad Lineal Rectificada (ReLU) al mapa de características, lo que introduce no linealidad en el modelo. Al agregar otra capa de convolución después de la inicial, la CNN adopta una estructura jerárquica, permitiendo que las capas posteriores examinen los píxeles dentro de los campos receptivos de las capas anteriores, proporcionando una visión más compleja y detallada de las características en la imagen.

La capa convolucional es esencial en una Convolutional Neural Network (CNN) y desempeña un papel central en la realización de la mayor parte del cálculo. Esta capa necesita elementos específicos, como datos de entrada, un filtro y un mapa de características. El detector de características, también llamado kernel o filtro, se mueve a través de campos receptivos en la imagen para verificar la presencia de características, en un proceso conocido como convolución.

El detector de características es una matriz bidimensional (2-D) de pesos que representa una parte de la imagen. Normalmente, el tamaño del filtro es una matriz de 3×3 , lo que determina el tamaño del campo receptivo. El filtro se aplica a un área de la imagen y se calcula el producto punto entre los píxeles de entrada y el filtro. El resultado se introduce en una matriz de salida, y el filtro se desplaza con pasos para cubrir toda la imagen. El producto final de esta serie de operaciones se conoce como mapa de características, mapa de activación o característica convolucionada.

Tras cada operación de convolución, la CNN aplica una transformación de Unidad Lineal Rectificada (ReLU) al mapa de características, introduciendo no linealidad en el modelo. Al agregar otra capa de convolución después de la capa inicial, la CNN adopta una estructura jerárquica. Esto permite que las capas posteriores examinen los píxeles dentro de los campos receptivos de las capas anteriores, proporcionando una visión más compleja y detallada de las características en la imagen.

- **Capa de agrupamiento:** La capa de agrupamiento, también conocida como reducción de muestreo, tiene como objetivo disminuir la dimensionalidad y, por ende, la cantidad de parámetros en la entrada. Hay dos tipos principales de agrupación.

1. **Agrupación máxima:** A medida que el filtro se mueve sobre la entrada, selecciona el píxel con el valor máximo para incluirlo en la matriz de salida. Este enfoque se utiliza con mayor frecuencia en comparación con la agrupación promedio.

2. **Agrupación promedio:** A medida que el filtro se desplaza sobre la entrada, calcula el valor promedio dentro del campo receptivo y lo asigna a la matriz de salida.

Aunque la capa de agrupamiento conlleva la pérdida de información, presenta varios beneficios para la CNN. Contribuye a reducir la complejidad del modelo, mejora la eficiencia y mitiga el riesgo de sobreajuste.

- **Capa totalmente conectada:** Cada nodo en la capa de salida establece una conexión directa con un nodo en la capa anterior. Esta capa desempeña la función de clasificación, utilizando las características extraídas por las capas anteriores y sus diversos filtros. Mientras que las capas convolucionales y de agrupamiento generalmente emplean funciones ReLU, las capas totalmente conectadas (FC) suelen utilizar una función de activación softmax para clasificar las entradas de manera adecuada, generando probabilidades en un rango de 0 a 1.

Las CNN (Convolutional Neural Networks) son fundamentales para el reconocimiento de imágenes y las aplicaciones de visión artificial. La visión artificial, dentro del ámbito de la inteligencia artificial (IA), posibilita a los sistemas y equipos extraer

información valiosa a partir de imágenes digitales, videos y otras entradas visuales. Basándose en esta información, estos sistemas pueden tomar acciones específicas. La capacidad de ofrecer recomendaciones distingue a la visión artificial de las simples tareas de reconocimiento de imágenes.

2.1.10 Librerías y herramientas

Python, concebido por Guido Van Rossum e introducido por primera vez en 1991, fue creado con la intención de ser un lenguaje de programación fácil de leer y aprender. Van Rossum se inspiró en lenguajes como ABC y Modula-3. Desde su lanzamiento inicial, Python ha experimentado una evolución constante y ha ganado gran popularidad en la comunidad de programadores debido a su énfasis en la legibilidad y la productividad.

Python es un lenguaje de programación orientado a objetos de alto nivel y fácil de interpretar, con una sintaxis clara y fácilmente legible. Es ideal para la creación de prototipos y la realización de tareas ad hoc, y se emplea ampliamente en campos como la computación científica, el desarrollo web y la automatización. Reconocido como un lenguaje apto tanto para principiantes como para usuarios avanzados, Python cuenta con el respaldo de numerosos científicos de la computación y desarrolladores de aplicaciones en todo el mundo. (Oracle, ¿Qué es Python?, 2022).

La distintiva sintaxis clara y fácilmente legible de Python simplifica tanto la redacción como la comprensión del código. Este lenguaje se destaca por su versatilidad, adaptándose con facilidad a diversas necesidades y situaciones.

En comparación con la mayoría de los otros lenguajes de programación, en Python, la estructura del código depende de la sangría. Es necesario agregar espacios de manera predecible y específica a las líneas de código dentro de un bloque determinado. Seguir esta convención es esencial para que el código se ejecute correctamente.

Las bibliotecas de paquetes son esenciales en cualquier lenguaje de programación moderno, ya que proporcionan funcionalidades comunes y rutinarias, permitiendo a

los programadores aprovechar soluciones existentes. Aquí se presentan algunas de las principales bibliotecas utilizadas en este proyecto y escritas en Python:

1. OpenCV: OpenCV es una biblioteca de software de visión artificial y aprendizaje automático de código abierto. Su objetivo principal es proporcionar una infraestructura común para aplicaciones de visión artificial y promover la utilización de la percepción de máquinas en productos comerciales. OpenCV incluye más de 2500 algoritmos optimizados, abarcando desde algoritmos clásicos hasta los más avanzados en visión por computadora y aprendizaje automático.

- Licencia: Apache 2

2. NumPy: NumPy es el paquete fundamental para la computación científica en Python. Proporciona un objeto de matriz multidimensional, varios objetos derivados y una variedad de rutinas para operaciones rápidas en matrices, incluyendo operaciones matemáticas, lógicas, manipulación de formas, clasificación, selección, E/S, transformadas discretas de Fourier, álgebra lineal básica, estadísticas y más.

- Licencia: BSD

3. TensorFlow: TensorFlow es una biblioteca matemática simbólica que se fundamenta en el flujo de datos y la programación diferenciable. Principalmente, se emplea en el ámbito del aprendizaje profundo.

- Licencia: Apache 2.0

4. Matplotlib: Matplotlib es una biblioteca integral diseñada para la creación de gráficos estáticos, animaciones y visualizaciones interactivas en el entorno de programación Python.

- Licencia: Matplotlib License

5. Tkinter: Tkinter es la interfaz estándar de Python para el kit de herramientas de interfaz gráfica de usuario de Tcl/Tk. Ofrece un conjunto moderno de componentes de interfaz de usuario temáticos, así como una nueva API para utilizarlos.

- Licencia: Python Software Foundation License

En cuanto al entrenamiento de modelos neuronales, Google Colab se presenta como un entorno de codificación en la nube. Este entorno permite la utilización de Jupyter Notebooks y Python, constituyendo una herramienta gratuita proporcionada por Google para facilitar tanto la investigación como el desarrollo de aplicaciones relacionadas con el aprendizaje automático e inteligencia artificial. Google Colab ofrece acceso a las unidades de procesamiento gráfico (GPU) y a las unidades de procesamiento tensorial (TPU) de Google. Además, brinda acceso a librerías especializadas como TensorFlow, Keras y OpenCV. También es posible abrir archivos externos, almacenar los cuadernos en Google Drive y compartirlos con otros usuarios.

2.1.11 Evaluación de modelo

En la evaluación del modelo, se utilizan métricas de clasificación como indicadores para demostrar qué tan bien se adapta al modelo de entrenamiento. Comprender estas métricas requiere familiaridad con los siguientes conceptos (Abril Cruz, s.f.):

- Verdadero Positivo (TP): Se refiere a la predicción correcta de la clase positiva por parte del modelo.
- Verdadero Negativo (TN): Se refiere a la predicción correcta de una clase negativa por parte del modelo.
- Falso Positivo (FP): Se refiere a la predicción incorrecta de una clase positiva por parte del modelo.
- Falso Negativo (FN): Se refiere a la predicción incorrecta de una clase negativa por parte del modelo.

Para la medición de la exactitud es la fracción de la cuenta de las predicciones correctas del modelo sobre el total de predicciones y se representa como se muestra a continuación:

$$exactitud = \frac{\text{predicciones correctas}}{\text{Número total de predicciones}} = \frac{TP + TN}{FP + FN}$$

Ahora para la determinación de la precisión del modelo para no etiquetar una muestra como positiva cuando es negativa. Adelante se encuentra su representación:

$$precisión = \frac{TP}{TP + FP}$$

La Exhaustividad del modelo se define como la habilidad del modelo para detectar todas las muestras como positivas. Su representación es la siguiente manera:

$$Exhaustividad = \frac{TP}{TP + FN}$$

2.1.12 Interfaces de Usuario y usabilidad en la detección de LSM

La interfaz de usuario (IU o UI, por sus siglas en inglés) sirve como el medio de comunicación entre un usuario y un dispositivo, máquina o computadora, abarcando todos los puntos de interacción entre el usuario y el sistema.

Generalmente, se diseñan con el objetivo de ser comprensibles y de fácil manejo. Sin embargo, en el ámbito informático, es más apropiado describirlas como "usables", "amigables e intuitivas" debido a su complejidad. La interfaz constituye la parte visible de las aplicaciones, representando lo que se percibe de ellas. Por lo tanto, se le concede una importancia creciente y se presta mayor atención en su desarrollo.

El diseño de interfaces de usuario ha experimentado una notable evolución en los últimos años y continúa haciéndolo en la actualidad. Puede argumentarse que la interfaz de usuario constituye la primera impresión que un usuario forma de una aplicación, y si no cumple con la calidad adecuada, puede generar rechazo. Además, la interfaz puede imponer limitaciones en la comunicación entre la máquina y el usuario, ya que todo lo que no se pueda expresar a través de la interfaz corre el riesgo de perderse.

La usabilidad, definida como la medida de uso de un producto por parte de usuarios específicos para lograr objetivos de eficiencia, efectividad y satisfacción en un contexto de uso particular, es crucial. En este sentido, la interfaz de usuario debe ser usable, lo que implica que el usuario pueda realizar las tareas de manera rápida y sencilla, y se sienta cómodo con la aplicación (Universidad Ramón Areces, s.f.).

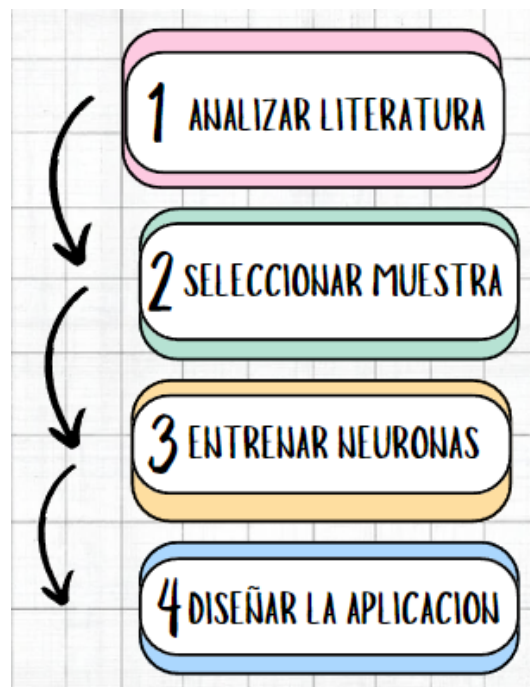
CAPÍTULO III

DESARROLLO Y METODOLOGÍA

3.1 PROCEDIMIENTO Y DESCRIPCIÓN DE LAS ACTIVIDADES REALIZADAS

Este proyecto de investigación se basará principalmente en una metodología aplicada, con el objetivo general de diseñar y entrenar un algoritmo de red neuronal para la identificación del lenguaje de señas mexicano. Para enriquecer el análisis, se incorporarán elementos cualitativos centrados en evaluar la usabilidad y las experiencias con literaturas previas, como se muestra en la siguiente imagen.

Ilustración 14. Diagrama de etapas de investigación.



Fuente: autoría propia.

En una fase inicial de investigación exploratoria, se llevará a cabo una revisión exhaustiva de la literatura relacionada con el análisis del lenguaje de señas y la aplicación de técnicas de visión artificial. Se explorarán proyectos previos que hayan utilizado OpenCV para analizar imágenes en el contexto del lenguaje de señas.

El desarrollo del modelo se realizará mediante la implementación de un algoritmo de red neuronal utilizando TensorFlow y OpenCV, específicamente en el entorno de programación Python 3.10. y Google Colab.

La evaluación del modelo abarcará métricas cuantitativas, como precisión y velocidad de procesamiento, para evaluar el rendimiento del algoritmo en la identificación de gestos.

Para la validación empírica, se dividirá el conjunto de datos en subconjuntos de entrenamiento y prueba con el fin de validar la eficacia del modelo diseñado.

Consideraciones éticas serán integradas en todo el proceso, evaluando aspectos como la privacidad de los datos y la inclusión de la perspectiva de la comunidad de usuarios, especialmente aquellos con discapacidad auditiva.

La documentación detallada del diseño y entrenamiento del algoritmo, así como de las decisiones clave tomadas durante el proceso, se presentará junto con los resultados obtenidos en una presentación clara que destaque las implicaciones prácticas del sistema propuesto.

Esta metodología integral permitirá el desarrollo de un algoritmo efectivo para la identificación del lenguaje de señas mexicano, integrando técnicas de visión artificial así como considerando las necesidades y experiencias de los usuarios finales.

3.2 ALCANCE Y ENFOQUE DE LA INVESTIGACIÓN

Esta investigación se enfocará en el diseño de una aplicación que utilice un modelo entrenado de redes neuronales para traducir el lenguaje de señas. El objetivo principal es proporcionar una herramienta para facilitar la comunicación entre personas sordas que utilizan el lenguaje de señas y aquellas que no lo comprenden.

3.2.1 Alcances

- Análisis de requisitos para capturar y procesar abecedario estático del lenguaje de señas.
- El desarrollo del entrenamiento del modelo de red neuronal.
- Implementación del modelo entrenado en una interfaz de usuario.

3.3 HIPÓTESIS

La aplicación diseñada, posee una predicción de imágenes con un 80% de certeza en la detección.

3.4 DISEÑO Y METODOLOGÍA DE LA INVESTIGACIÓN

Para realizar el proyecto expuesto en este documento, se plantea la adopción de un enfoque de investigación exploratoria y de desarrollo de tecnología, ya que por medio del análisis de literatura se busca encontrar cualidades a tomar en cuenta para el desarrollo del diseño de la aplicación, con ello proponer una aplicación que sea capaz de cubrir las necesidades encontradas por las tecnologías analizadas.

Para la parte del entrenamiento de las neuronas por medio de imágenes capturadas como muestra, se realizarán pruebas de funcionamiento para obtener cierta veracidad en la detección. Este enfoque implica el análisis de imágenes con el respaldo de bibliotecas como OpenCV, que está disponible para el lenguaje de programación Python. La finalidad es desarrollar una aplicación capaz de analizar e interpretar el lenguaje de señas mexicano a través de esta metodología experimental.

3.5 SELECCIÓN DE LA MUESTRA

De primera mano se seleccionaron algunas investigaciones con respaldo científico desarrollado en un lapso de tiempo no mayor a 5 años.

Para la parte del análisis de imágenes se optó por tres tipos diferentes de manos, que cuentan con diferentes características, esto para brindar un rango más amplio de análisis, aunque en aplicaciones similares e investigaciones previas se utilizan entre 1 y 5 tipos diferentes de muestras para el análisis de imágenes.

3.6 RECOLECCIÓN DE DATOS

Durante la fase de recolección de datos, se optó por un enfoque que implicaba la captura de fotografías de gestos en lengua de señas mexicana. Se estableció un procedimiento estructurado que definía gestos específicos para ser fotografiados.

Las imágenes fueron tomadas en un entorno controlado, con condiciones de iluminación uniformes y un fondo neutral, asegurando así la calidad de las imágenes. Se utilizó un dispositivo de alta resolución para capturar a los voluntarios realizando ciertos gestos desde diversos ángulos. Este proceso generó un conjunto de datos diverso y representativo, aspecto crucial para el entrenamiento de modelos. Se garantizó la ética, obteniendo el consentimiento informado de los participantes y protegiendo los datos al excluir identidades de las imágenes grabadas.

Posteriormente, las imágenes fueron sometidas a un proceso de preprocesamiento, que incluyó la normalización del tamaño y la corrección de la iluminación. Esto se llevó a cabo para asegurar la coherencia y calidad de los datos antes de ser utilizados en el entrenamiento del modelo. Este método de recopilación de datos proporcionó un conjunto sólido y representativo, fundamental para el desarrollo y entrenamiento de un algoritmo de reconocimiento de la Lengua de Señas Mexicana.

3.6.1 SELECCIÓN DEL INSTRUMENTO

Como instrumento de trabajo se seleccionó una revisión de literatura que nos permita analizar cómo es que previamente y en la actualidad, se ha tratado con la problemática abordada en esta investigación, así, cualitativamente poder obtener información que puede llegar mejorar u optimizar la aplicación desarrollada.

De la mano de lo anterior, el desarrollo del software, es decir de la aplicación que por medio de pruebas se buscara cubrir la hipótesis y los objetivos del presente proyecto de investigación.

3.6.2 APLICACIÓN DE INSTRUMENTO

3.6.2.1 Revisión de literatura

Si bien se encontraron varias aplicaciones desarrolladas para apoyar a las personas con la comprensión del LSM, existen otras diseñadas para enseñar este lenguaje, pero en esta ocasión fueron seleccionadas literaturas y o aplicaciones con la finalidad de analizar y detectar el lenguaje de señas, mismas que se muestran a continuación:

- Redes neuronales multicapa y convolucionales para el reconocimiento del lenguaje de señas boliviano: una evaluación empírica. (Rodríguez Villarroel, Juan Pablo, Ponce de León Espinoza, Nicolás, & Arteaga Sabja, Wendoline. (2021). Multilayer and convolutional neural networks for Bolivian Sign Language recognition: an empirical evaluation. *Acta Nova*, 10(1), 22-41. Recuperado en 09 de febrero de 2024)

En este artículo se describe una investigación empírica que se centra en el uso de redes neuronales convolucionales y multicapa para traducir la Lengua de Señas Boliviana al español. Se utiliza una muestra de fotogramas de imágenes recopilados de un vídeo como entrada para el proceso de traducción. Para seleccionar estos fotogramas de imagen, se emplea una técnica de selección basada en movimiento, capturando cada cuadro donde se observa un cambio notable en la dirección de la mano en el vídeo. El algoritmo compara cada fotograma y selecciona aquellos en los que el movimiento de la persona experimenta un cambio brusco de dirección. Posteriormente, se aplica un algoritmo de detección de bordes propuesto por Canny et al. (1986) a todos los fotogramas seleccionados. Finalmente, se analiza cada gesto representativo como una clase para la red neuronal.

En el experimento realizado, se evaluaron dos tipos de redes neuronales para analizar 60 vídeos que contenían cuatro frases básicas de Lengua de Señas Boliviana. Los resultados del experimento revelan que la red neuronal multicapa (MLP) tiene una precisión que varía entre el 65% y el 88%, mientras que la red neuronal convolucional (CNN) tiene una precisión que oscila entre el 95% y el 99% para los signos considerados en el estudio.

- Detección y clasificación de objetos aplicado a un área de investigación: reconocimiento de la lengua de señas mexicana. Jaime-Gaytán, C. E. (2023, mayo) Detección y clasificación de objetos a un área de investigación: reconocimiento de la lengua de señas mexicana. *Revista de Investigación*).

En este proyecto, se llevó a cabo una investigación detallada para evaluar y comparar dos arquitecturas de redes neuronales convolucionales, VGG16 y ResNet50, en el ámbito de la detección de puntos clave en la Lengua de Señas Mexicana. Se establecieron objetivos claros y se utilizó una metodología sólida que abarcó desde la recopilación de datos hasta la implementación y evaluación de los modelos.

Uno de los principales descubrimientos fue que la arquitectura ResNet50 superó de manera consistente a VGG16 en términos de precisión en la predicción de coordenadas y la pérdida asociada, resaltando su eficacia en la tarea de localizar puntos clave en las imágenes de señas. Ambas arquitecturas demostraron una eficacia excepcional al predecir las categorías de las imágenes, logrando una precisión del 100%.

En cuanto a las fortalezas identificadas, se determinó que ResNet50 destacó en precisión y pérdida en coordenadas, lo que la convierte en la preferida para tareas de detección de puntos clave. Sin embargo, también se observó que esta arquitectura demanda un tiempo de entrenamiento más extenso en comparación con VGG16, lo que implica un equilibrio necesario entre precisión y eficiencia temporal.

Entre las limitaciones identificadas se encuentran el tamaño limitado del conjunto de datos y la variabilidad en las señas. Además, se identificaron áreas para futuras investigaciones, como la adaptación de los modelos a situaciones del mundo real y la mejora de la recopilación de datos.

- Sistema de detección automático de una lengua de señas específica por medio de inteligencia artificial. (Peña Pedraza, D. (2023). Sistema de detección automático de una lengua de señas específica por medio de inteligencia artificial. Universitat Politècnica de Catalunya,.)

Durante el desarrollo de este proyecto, se logró establecer una base de datos de manera exitosa, un hito significativo en la investigación. Sin embargo, al analizar el

proceso, se identificó la necesidad de un control más riguroso de las condiciones ambientales para futuras investigaciones. Además, se dedicó un esfuerzo meticuloso en la preparación y procesamiento de los datos, una fase crucial en la construcción del modelo de aprendizaje automático. Se abordaron tres modalidades de datos que desempeñaron un papel esencial en la implementación del primer modelo.

A pesar de los esfuerzos, el primer modelo no alcanzó los resultados esperados debido a ciertas limitaciones en la implementación de las modalidades y a la antigüedad de las librerías utilizadas, lo que impactó en la robustez de la base de datos. Sin embargo, esta experiencia no careció de méritos sustanciales. Permitió adquirir un mayor conocimiento sobre diversas técnicas de procesamiento, destacando la habilidad para adaptar los datos de entrada según su tamaño. Además, se realizaron ajustes en los códigos, lo que permitió configurarlos y vincularlos con datos provenientes directamente de una plataforma en la nube, como Amazon S3.

El uso de aprendizaje por transferencia demostró ser valioso en el desarrollo del proyecto, aunque se subraya la importancia de gestionar proactivamente las dependencias tecnológicas debido a la rápida obsolescencia de las librerías y paquetes. A pesar de los desafíos, el proceso de aprendizaje resultó altamente productivo, proporcionando una comprensión más profunda de las complejidades de las señas. Esto llevó a reorientar la investigación hacia la complejidad inherente del problema.

La incorporación de una base de datos complementaria con señas dinámicas de una sola mano en el segundo modelo condujo a mejoras significativas en los resultados. Esto subraya la importancia de contar con conjuntos de datos robustos y llevó a centrarse en señas realizadas con una sola mano y de naturaleza dinámica, una elección estratégica efectiva en el segundo modelo. Finalmente, el segundo modelo, con las adaptaciones mencionadas en su arquitectura, arrojó resultados prometedores. Se logró una exactitud global del 74.18%, lo que representa una

mejora significativa con respecto a la implementación inicial del segundo modelo, que tuvo una exactitud global del 60.3%.

Si bien de las aplicaciones encontradas, varias de ellas se enfocan en el análisis de lenguajes de señas de otros países, la finalidad es la misma a la del proyecto realizado. Muchas de ellas por medio del entrenamiento de redes neuronales, desarrollaron aplicaciones y o algoritmos capaces de predecir imágenes capturadas de personas comunicándose por LDS, así como otros más por video o en tiempo real.

Todas ellas usaron de diferentes arquitecturas tanto para el entrenamiento, como para el desarrollo de sus aplicaciones, con ello podemos decir que, si bien existen varios proyectos que ya se adentraron en el análisis de esta problemática, con la metodología propuesta y con ayuda de Python, así como de google colab, ha sido poco explotada.

3.6.2.2 Desarrollo de software

Para construir el modelo de entrenamiento, se requiere recopilar un conjunto de datos ordenados. En este proyecto, se recolectó un conjunto de imágenes que representan cada letra estática del abecedario LSM. Con el objetivo de garantizar un entrenamiento efectivo, se capturaron quinientas imágenes por cada letra. Cada una de estas imágenes tiene dimensiones uniformes de 100 x 81 píxeles. Este enfoque asegura una cantidad significativa de datos para el entrenamiento del modelo y establece una consistencia en las dimensiones de las imágenes, facilitando así el proceso de análisis y procesamiento.

Ilustración 15. Muestra de la toma de fotografías.

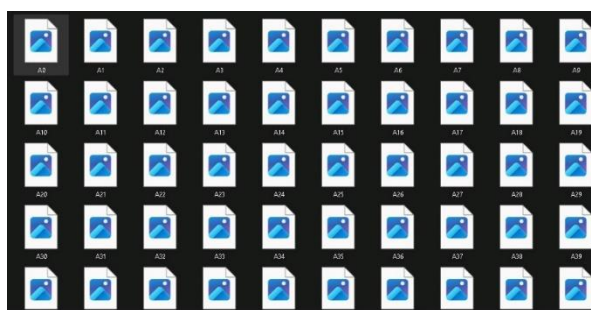


Fuente: autoría propia.

Cada imagen fue etiquetada para facilitar la cuantificación y clasificación de las mismas. Este etiquetado permite identificar y categorizar cada imagen según la letra del abecedario LSM que representa. Además de facilitar la organización de los datos, el etiquetado es esencial para el proceso de entrenamiento del modelo, ya que proporciona información crucial sobre las clases a las que pertenecen las imágenes.

En el caso de que se supere la cantidad de imágenes por cada letra en el futuro, se pueden implementar soluciones para manejar este problema

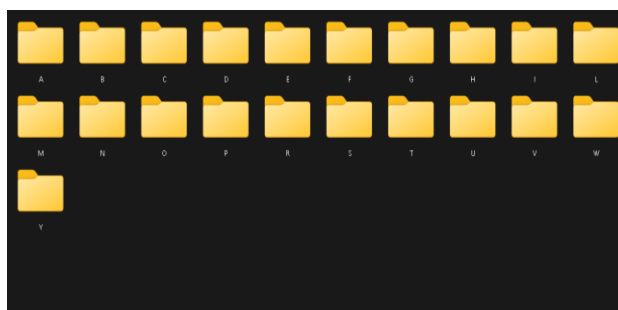
Ilustración 16. Etiquetado de imágenes.



Fuente: autoría propia.

Clasificar las imágenes etiquetadas en diferentes carpetas es una práctica organizativa fundamental. Esta clasificación permite estructurar y ordenar el conjunto de datos de manera coherente y comprensible. Al agrupar las imágenes por categorías o etiquetas en carpetas separadas, se facilita enormemente el proceso de análisis posterior.

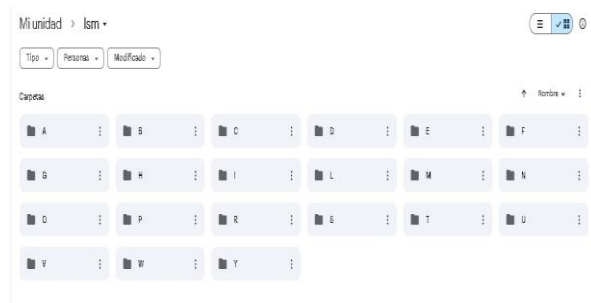
Ilustración 17. Clasificado de imágenes.



Fuente: autoría propia.

Subir la carpeta del conjunto de datos a Google Drive facilita el manejo y acceso a los datos durante el proceso de creación y entrenamiento de la red neuronal en Google Colab, lo que contribuye a una experiencia más fluida y eficiente en el desarrollo del proyecto. Además, al utilizar Google Drive, no existirá problema por la pérdida de datos, ya que tus archivos estarán almacenados de forma segura en la nube.

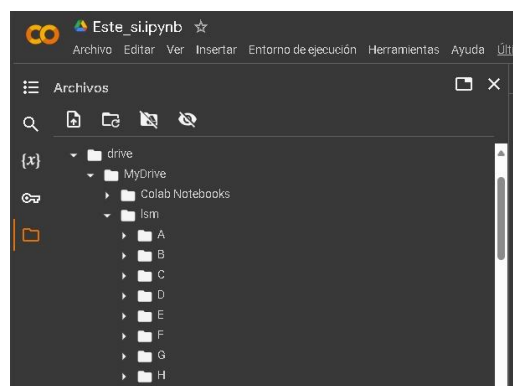
Ilustración 18. Data set en la nube.



Fuente: autoría propia.

Exactamente, para acceder al conjunto de datos desde Google Colab, simplemente se necesita dar acceso a la unidad de Google Drive. Esto se puede lograr montando la unidad de Google Drive en Google Colab. Esto te permite leer, escribir y manipular los datos directamente desde Google Colab como si estuvieran almacenados en la computadora.

Ilustración 19. Conexión de Data set con Google Colab



Fuente: autoría propia.

Una de las primeras tareas al iniciar el entrenamiento en Google Colab es vincular la carpeta donde se encuentra el conjunto de datos. Esto se hace utilizando comandos específicos en Python para montar la unidad de Google Drive en Colab.

Además, deseamos visualizar las imágenes contenidas en una de las carpetas para confirmar que están correctamente clasificadas.

Ilustración 20. Código de entrenamiento de red neuronal en Google Colab.

```
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

plt.figure(figsize=(15,15))

carpeta = '/content/drive/MyDrive/lsm/V'
imagenes = os.listdir(carpeta)

for i, name_img in enumerate(imagenes[:25]):
    plt.subplot(5,5,i+1)
    imagen = mpimg.imread(carpeta + '/' + name_img)
    plt.imshow(imagen)
```

Fuente: autoría propia.

Dentro del entorno de Colab, se crearán carpetas para alojar el conjunto de datos. Cada letra tendrá su propia carpeta virtual asignada. Con el propósito de manipular los datos durante el entrenamiento.

Ilustración 21. Código de entrenamiento de red neuronal en Google Colab.

```
[ ] #Crear carpetas para hacer el set de datos

mkdir dataset
mkdir dataset/A
mkdir dataset/B
mkdir dataset/C
mkdir dataset/D
mkdir dataset/E
mkdir dataset/F
mkdir dataset/G
mkdir dataset/H
mkdir dataset/I
mkdir dataset/L
mkdir dataset/M
mkdir dataset/N
mkdir dataset/O
mkdir dataset/P
```

Fuente: autoría propia.

Se transferirán las imágenes desde la unidad de Drive a cada carpeta creada en el entorno de Google Colab. Además, se limitará la cantidad de imágenes transferidas a un máximo de quinientas por carpeta. Este proceso se llevará a cabo para cada letra estática en el abecedario LSM.

Ilustración 22. Código de entrenamiento de red neuronal en Google Colab.

```
#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/A'
carpeta_destino = '/content/dataset/A'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino + '/' + nombreimg)
```

Fuente: autoría propia.

Se aplicarán transformaciones aleatorias al conjunto de datos, como estirar, mover o girar las imágenes, con el fin de diversificar el conjunto de datos. Para llevar a cabo este proceso, utilizaremos la clase ImageDataGenerator. Además, especificaremos que utilizaremos el 20 por ciento del conjunto de datos para pruebas, reservando el restante para el entrenamiento. También redimensionaremos las imágenes a un tamaño de 224 x 224 píxeles.

Ilustración 23. Código de entrenamiento de red neuronal en Google Colab.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

#Crear el dataset generador
datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range = 30,
    width_shift_range = 0.25,
    height_shift_range = 0.25,
    shear_range = 15,
    zoom_range = [0.5, 1.5], no
    validation_split=0.2 #20% para pruebas
)

#Generadores para sets de entrenamiento y pruebas
data_gen_entrenamiento = datagen.flow_from_directory('/content/dataset', target_size=(224,224),
    batch_size=32, shuffle=True, subset='training')
data_gen_pruebas = datagen.flow_from_directory('/content/dataset', target_size=(224,224),
    batch_size=32, shuffle=True, subset='validation')
```

Fuente: autoría propia.

Una vez realizado lo anterior, imprimiremos las imágenes para observar los cambios aleatorios efectuados. De esta manera, podremos apreciar las modificaciones en las imágenes, las cuales estarán dispuestas de manera aleatoria para su visualización.

Ilustración 24. Código de entrenamiento de red neuronal en Google Colab.

```
#Imprimir 10 imágenes del generador de entrenamiento
for imagen, etiqueta in data_gen_entrenamiento:
    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(imagen[i])
        break
plt.show()
```

Fuente: autoría propia.

Durante la ejecución, se observará la cantidad de imágenes utilizadas para el entrenamiento y la cantidad de imágenes utilizadas para pruebas.

Ilustración 25. Ejecución de código, muestra imágenes aleatorias.



Fuente: autoría propia.

Se buscará un modelo pre entrenado en TensorFlowHub, donde existen diversas opciones disponibles. El modelo que mejor se adapta a nuestras necesidades es el de MobileNet_v2. Exportaremos este modelo a Google Colab para poder descargarlo, junto con la información sobre la forma de entrada con la que trabaja el modelo.

Ilustración 26. Código de entrenamiento de red neuronal en Google Colab.

```
import tensorflow as tf
import tensorflow_hub as hub

url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
mobilenetv2 = hub.KerasLayer(url, input_shape=(224,224,3))
```

Fuente: autoría propia.

Una parte crucial es congelar los parámetros del modelo utilizando la propiedad "trainable", lo que incluye sus pesos, sesgos, entre otros. Esto se debe a que estos parámetros no cambian en esta sección y deben permanecer fijos durante el proceso de entrenamiento o inferencia del modelo.

Ilustración 27. Código de entrenamiento de red neuronal en Google Colab.

```
#Congelar el modelo descargado
mobilenetv2.trainable = False
```

Fuente: autoría propia.

El siguiente paso es crear el modelo de entrenamiento de tipo secuencial. Primero, incorporamos el modelo pre-entrenado y lo configuramos para categorizarlo en las 21 clases correspondientes.

Ilustración 28. Código de entrenamiento de red neuronal en Google Colab.

```
modelo = tf.keras.Sequential([
    mobilenetv2,
    tf.keras.layers.Dense(21, activation='softmax')
])
```

Fuente: autoría propia.

Luego de configurar el modelo de entrenamiento, procedemos a compilarlo para guardar todas las modificaciones realizadas.

Ilustración 29. Código de entrenamiento de red neuronal en Google Colab.

```
#Compilar como siempre
modelo.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Fuente: autoría propia.

Para entrenar el modelo, especificamos el número de épocas que debe realizar, en este caso, se establece en 150. Además, indicamos que utilice tanto los datos de entrenamiento como los de prueba. Es importante tener en cuenta que este proceso puede demorar debido a la cantidad de épocas demandadas, ya que cada época implica un ciclo completo de entrenamiento sobre todo el conjunto de datos. Por lo tanto, es posible que el tiempo de entrenamiento aumente considerablemente con un mayor número de épocas.

Ilustración 30. Código de entrenamiento de red neuronal en Google Colab.

```
#Entrenar el modelo
EPOCHAS = 150

historial = modelo.fit(
    data_gen_entrenamiento, epochs=EPOCHAS, batch_size=32,
    validation_data=data_gen_pruebas
)

Epoch 1/150
263/263 [=====] - 124s 427ms/step -
Epoch 2/150
263/263 [=====] - 115s 439ms/step -
Epoch 3/150
```

Fuente: autoría propia.

Una vez finalizado el entrenamiento, es crucial corroborar en las gráficas de precisión qué tan confiable es nuestro modelo entrenado. Estas gráficas nos brindarán

información sobre el rendimiento del modelo durante el proceso de entrenamiento, lo que nos ayudará a evaluar su eficacia y precisión en la clasificación de los datos.

Ilustración 31. Código de entrenamiento de red neuronal en Google Colab.

```
#Graficas de precisión
acc = historial.history['accuracy']
val_acc = historial.history['val_accuracy']

loss = historial.history['loss']
val_loss = historial.history['val_loss']

rango_epocas = range(150)

plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(rango_epocas, acc, label='Precisión Entrenamiento')
plt.plot(rango_epocas, val_acc, label='Precisión Pruebas')
plt.legend(loc='lower right')
plt.title('Precisión de entrenamiento y pruebas')

plt.subplot(1,2,2)
plt.plot(rango_epocas, loss, label='Pérdida de entrenamiento')
plt.plot(rango_epocas, val_loss, label='Pérdida de pruebas')
plt.legend(loc='upper right')
plt.title('Pérdida de entrenamiento y pruebas')
plt.show()
```

Fuente: autoría propia.

Para evaluar la funcionalidad del modelo, es crucial corroborar su desempeño utilizando una imagen. En este caso, se seleccionó una imagen de Internet y se le aplicaron diversas modificaciones para que el modelo pueda clasificarla correctamente. Este proceso nos permitirá verificar si el modelo puede generalizar su aprendizaje y realizar predicciones precisas sobre nuevas imágenes.

Ilustración 32. Código de entrenamiento de red neuronal en Google Colab.

```
#Categorizar una imagen de internet
from PIL import Image
import requests
from io import BytesIO
import cv2

def categorizar(url):
    respuesta = requests.get(url)
    img = Image.open(BytesIO(respuesta.content))
    img = np.array(img).astype(float)/255

    img = cv2.resize(img, (224,224))
    prediccion = modelo.predict(img.reshape(-1, 224, 224, 3))
    return np.argmax(prediccion[0], axis=-1)
```

Fuente: autoría propia.

Una vez que el modelo ha sido probado y evaluado satisfactoriamente, se procede a guardar el modelo para utilizarlo en una interfaz de usuario. Este paso garantiza que podamos utilizar el modelo entrenado en aplicaciones y sistemas interactivos, como una interfaz de usuario, para realizar predicciones en tiempo real.

Ilustración 33. Código de descarga de modelo entrenado en Google Colab.

```
#Crear la carpeta para exportarla a TF Serving
!mkdir -p carpeta_salida/modelo_lsm/1

#Guardar el modelo en formato SavedModel
modelo.save('carpeta_salida/modelo_lsm/1')

#Hacerlo un zip para bajarlo y usarlo en otro lado
!zip -r modelo_lsm.zip /content/carpetas_salida/modelo_lsm/
```

Fuente: autoría propia.

Ahora trabajaremos con la aplicación PyCharm Community Edition 2023.2.1. Guardaremos nuestro modelo entrenado en una de las carpetas de PyCharm. Además, necesitaremos descargar las bibliotecas necesarias para poder crear la interfaz de usuario. Crearemos un nuevo archivo en el cual importaremos las bibliotecas que necesitamos. Esto nos permitirá comenzar a trabajar en el desarrollo de la interfaz de usuario.

Ilustración 34. Código de interfaz de usuario en PyCharm Community.

```
import cv2
from tkinter import *
from PIL import Image
from PIL import ImageTk
import json
import numpy as np
import tensorflow as tf
from tensorflow import keras
import tensorflow_hub as hub
```

Fuente: autoría propia.

A continuación, abriremos nuestro modelo de entrenamiento para poder utilizarlo en la interfaz de usuario. También inicializaremos nuestras variables que contendrán valores vacíos, lo cual nos permitirá preparar la interfaz para recibir datos y realizar predicciones.

Ilustración 35. Código de interfaz de usuario en PyCharm Community.

```
'''abrimos el modelo'''
with open('model.json', 'r') as archivo_lsm:
    datos_lsm = archivo_lsm.read()

modelo_lsm = keras.models.model_from_json(json.dumps(datos_lsm))

'''variables iniciadas'''
frame_procesar = None
resultado = None
respuesta = None
```

Fuente: autoría propia.

Siguiendo con el programa, comenzamos definiendo las clases que procesarán las imágenes y permitirán su detección. Estas clases serán fundamentales para el funcionamiento de nuestra interfaz de usuario y nos permitirán interactuar con las imágenes de manera efectiva.

Ilustración 36. Código de interfaz de usuario en PyCharm Community.

```
def procss_lsm():
    global selec, frame_procesar

    selec = False

    def inicio():
        global selec, frame_procesar

        visualizar()
        botonDetectar.config(state='normal')
        selec = False
        return
```

Fuente: autoría propia.

La siguiente clase muestra la visualización en tiempo real de la mano, la cual estará contenida dentro de un rectángulo con las dimensiones necesarias para su uso con

el modelo de entrenamiento. Esta clase es la encargada de capturar la imagen que será utilizada para la predicción.

Ilustración 37. Código de interfaz de usuario en PyCharm Community.

```
42 def visualizar():
43     global selec
44
45     if captura is not None:
46         ret, frame = captura.read()
47         cv2.rectangle(frame, (100, 50), (324, 274), (0, 255, 0), 2)
48         if ret == True:
49             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
50             im = Image.fromarray(frame)
51             img = ImageTk.PhotoImage(image=im)
52             etiquetaVideo.configure(image=img)
53             etiquetaVideo.image = img
54             etiquetaVideo.after(ms=10, visualizar)
55             if selec == True:
56                 frame_procesar = frame[50:274, 100:324]
57                 cv2.imshow('Procesamiento', frame_procesar)
58                 selec = False
59                 return frame_procesar
60             else:
61                 etiquetaVideo.image = ""
62                 Button.configure(state="disabled")
63                 captura.release()
```

Fuente: autoría propia.

En esta clase se realiza la clasificación de la imagen, detectando qué letra está mostrando la mano y la clasifica en una de las letras correspondientes. Este proceso es fundamental para la funcionalidad de la interfaz de usuario, ya que permite identificar y categorizar la letra que se muestra en la imagen capturada.

Ilustración 38. Código de interfaz de usuario en PyCharm Community.

```
66 def deteccion_lsm():
67     '''metodo en donde se haga el procesamiento de imagen'''
68     global selec, frame_procesar, resultado, respuesta
69
70     selec = True
71
72     if frame_procesar is not None:
73         #procesar imagen
74         frame_procesar = cv2.cvtColor(frame_procesar, cv2.COLOR_BGR2GRAY)
75         frame_procesar = frame_procesar / 255.0
76         #hacer la predicción
77         tensor = np.expand_dims(frame_procesar, axis=0)
78         tensor = np.expand_dims(frame_procesar, axis=3)
79         resultado = modelo_lsm.predict(tensor).flatten()
80         print('el valor de la predicción es ', resultado)
81         if resultado == 0:
82             respuesta = "A"
83         if resultado == 1:
84             respuesta = "B"
```

Fuente: autoría propia.

Por último, la clase "Finalizar y Limpiar" permite cerrar y liberar la memoria utilizada por la interfaz de usuario. Esto es importante para garantizar una gestión adecuada de los recursos del sistema y evitar posibles fugas de memoria.

Ilustración 39. Código de interfaz de usuario en PyCharm Community.

```
def finalizar_limpiar():  
    etiquetaVideo.image = ""  
    captura.release()  
    ventanaPrin.destroy()
```

Fuente: autoría propia.

Para la creación de la interfaz de usuario, se emplearon las siguientes características, como el tamaño, el color y el nombre de la aplicación. Además, se inicializó la cámara para permitir la captura de imágenes en tiempo real. Estos elementos son fundamentales para diseñar una interfaz atractiva y funcional para los usuarios.

Ilustración 40. Código de interfaz de usuario en PyCharm Community.

```
134     ventanaImagen = Toplevel(ventanaPrin)  
135     ventanaImagen.geometry('730x700')  
136     ventanaImagen.title('Lenguaje de señas mexicano')  
137     ventanaImagen.geometry('+0+0') # coloca la ventana en la esquina superior derecha  
138     ventanaImagen.configure(bg='#E5CCFF') # color de la ventana  
139     ventanaImagen.focus_set()  
140     ventanaImagen.grab_set()  
141     ventanaImagen.transient(master=ventanaPrin)  
142  
143     imagen_frame = TK.Frame(ventanaPrin)  
144     imagen_frame = Frame(ventanaPrin, highlightcolor='#0080FF', highlightthickness=2)  
145     imagen_frame.place(x=20, y=20)  
146     imagen_frame.config(width=690, height=525, bg='white', relief='flat')  
147  
148     canvas = Canvas(imagen_frame, bg='white')  
149     canvas.place(x=20, y=20)  
150  
151     '''variables'''  
152     resultado = IntVar()
```

Fuente: autoría propia.

Además, como parte de la interfaz de usuario, se agregaron etiquetas, títulos y botones que están vinculados a las funciones correspondientes. Estos elementos

permiten al usuario interactuar con la aplicación, ya sea para iniciar o finalizar la detección de la mano y la clasificación de las letras.

Ilustración 41. Código de interfaz de usuario en PyCharm Community.

```
etiquetaVideo = Label(ventanaImagen) # etiqueta en la cual se coloca la camara
etiquetaVideo.place(x=50, y=20) # posicon de la camara
Label(ventanaImagen, textvariable=respuesta, font=('Arial Black', 12)).place(x=360, y=580)
botonInciar = Button(ventanaImagen, text='Iniciar', font=('Arial Black', 12), width=18, height=1, bg='#99CCFF', command=
botonInciar.place(x=20, y=630)
botonDetectar = Button(ventanaImagen, text='Detectar', font=('Arial Black', 12), width=18, height=1, bg='#99CCFF', command=
botonDetectar.place(x=250, y=630)
botonDetectar.config(state='disable')
Button(ventanaImagen, text='Salir y limpiar', font=('Arial Black', 12), width=18, height=1, bg='#99CCFF',
command=finalizar_limpiar).place(x=500, y=630)
'''Iniciamos la caprura de video'''
captura = cv2.VideoCapture(0, cv2.CAP_DSHOW)

return

'''pantalla principal'''
ventanaPrin = Tk()
pross_lsm()
ventanaPrin.mainloop()
```

Fuente: autoría propia.

CAPÍTULO IV

RESULTADOS

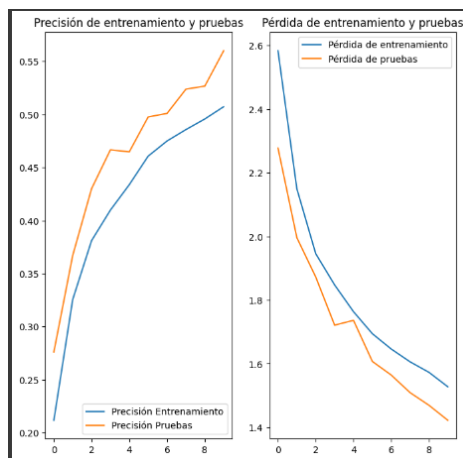
4.1 Resultados

Durante el entrenamiento de la red neuronal, se realizaron varios ciclos completos de entrenamiento, conocidos como "épocas", en los cuales la red neuronal recibió un conjunto de datos y se ajustó a él. Se utilizó un data-set que contenía 500 imágenes (de 20x16 píxeles respectivamente) por cada letra estática del abecedario de señas.

A continuación, se muestra la gráfica generada por el programa, la cual representa el comportamiento del entrenamiento.

- Numero de épocas es igual a 10. Se puede observar que no se alcanza un porcentaje aceptable en el entrenamiento, y que hay una pérdida significativa durante las pruebas de entrenamiento. Esto sugiere que el modelo no está aprendiendo de manera efectiva los patrones en los datos de entrenamiento, lo que puede deberse a diversos factores, como la complejidad del problema, la cantidad y calidad de los datos, así como la arquitectura y los hiperparámetros del modelo. Es necesario realizar un análisis más detallado para identificar y abordar las posibles causas de este bajo rendimiento.

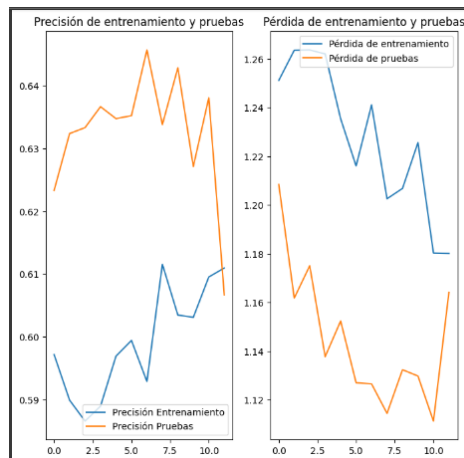
Ilustración 42. Grafica de precisión y perdida de entrenamiento y pruebas a 10 épocas.



Fuente: autoría propia.

- Numero de épocas es igual a 12. Con 12 épocas de entrenamiento, se hace más evidente la pérdida de precisión tanto en los datos de entrenamiento como en los de prueba, ya que se observa una caída repentina en la precisión de los datos de entrenamiento. Este comportamiento sugiere que el modelo puede estar experimentando dificultades para generalizar adecuadamente a partir de los datos de entrenamiento, lo que puede ser indicativo de un sobreajuste u otros problemas en la configuración del modelo.

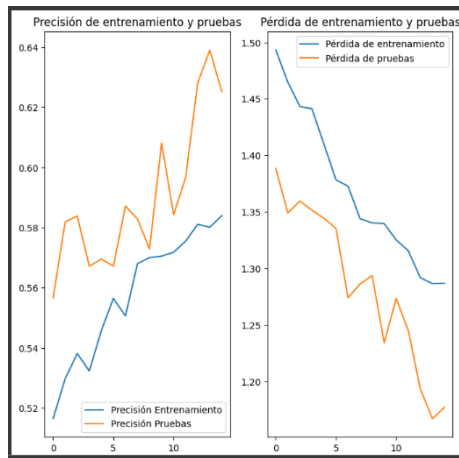
Ilustración 43. Grafica de precisión y perdida de entrenamiento y pruebas a 12 épocas.



Fuente: autoría propia.

- Numero de épocas es igual a 15. Después de realizar el entrenamiento con 15 épocas, se observó un ligero aumento en el porcentaje en comparación con el entrenamiento anterior. Sin embargo, el resultado aún no era aceptable, ya que persistía una baja precisión. Además, se evidenció una caída en la precisión de las pruebas.

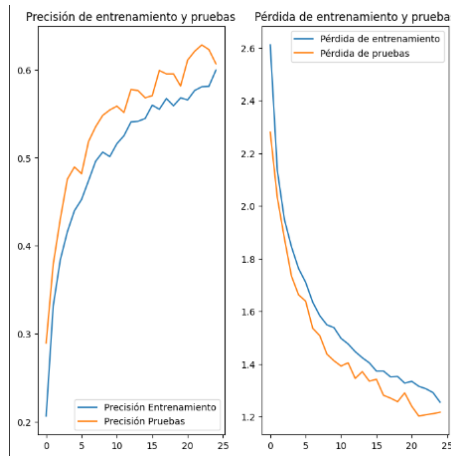
Ilustración 44. Grafica de precisión y perdida de entrenamiento y pruebas a 15 épocas.



Fuente: autoría propia.

- Numero de épocas es igual a 25. En este entrenamiento de 25 épocas, se observa que tanto la precisión como la pérdida de datos muestran menos desajuste en comparación con entrenamientos anteriores. Sin embargo, a pesar de esta mejora, el porcentaje de precisión sigue siendo insuficiente, rondando alrededor del 60 por ciento. Esto sugiere que, aunque se han realizado mejoras en el ajuste del modelo, aún queda trabajo por hacer para mejorar su rendimiento y aumentar su precisión. Es necesario seguir explorando diferentes enfoques y ajustes en la configuración del modelo para alcanzar un nivel de precisión aceptable.

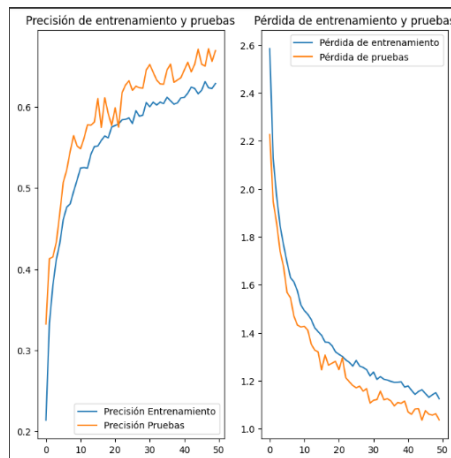
Ilustración 45. Grafica de precisión y perdida de entrenamiento y pruebas a 25 épocas.



Fuente: autoría propia.

- Numero de época es igual a 50. Con 50 épocas de entrenamiento, aunque se observa un ligero aumento en el porcentaje de precisión en comparación con entrenamientos anteriores, aún no alcanza un nivel aceptable, ya que se mantiene por encima del 60%. Esto indica que el modelo aún no está logrando capturar de manera efectiva los patrones en los datos de entrenamiento para realizar predicciones precisas.

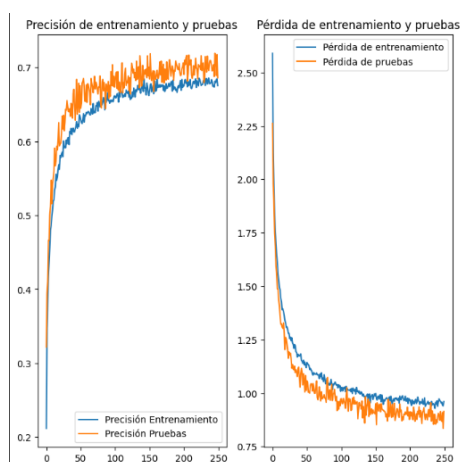
Ilustración 46. Grafica de precisión y perdida de entrenamiento y pruebas a 50 épocas.



Fuente: autoría propia.

- Numero de épocas es igual a 250. Después de incrementar el número de épocas a 250 en un intento por mejorar el porcentaje de aceptación del modelo, se observó que el modelo aún no alcanzaba el nivel deseado de precisión. A pesar de este aumento en el número de épocas, el porcentaje de precisión obtenido solo alcanzó el 70 por ciento. Este resultado sugiere que simplemente aumentar el número de épocas no es suficiente para mejorar significativamente el rendimiento del modelo. Es probable que se necesiten otros ajustes en la arquitectura del modelo

Ilustración 47. Grafica de precisión y perdida de entrenamiento y pruebas a 250 épocas.

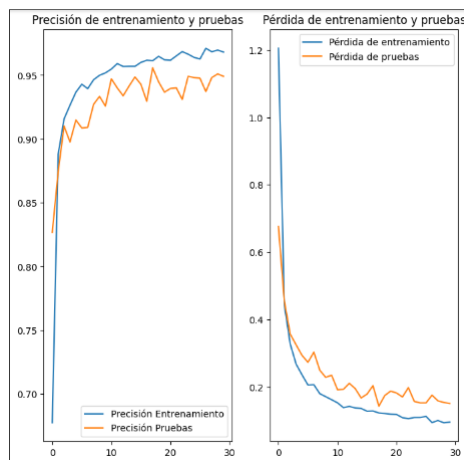


Fuente: autoría propia.

Se implementó la creación de un nuevo conjunto de datos que contiene imágenes con dimensiones diferentes a las anteriores, específicamente de 100x81 píxeles. Esta decisión surgió tras detectar que el tamaño de las imágenes estaba afectando el proceso de entrenamiento. Se observó que, al redimensionar las imágenes en una parte del programa, se producían distorsiones que afectaban la estabilidad tanto del conjunto de datos de entrenamiento como del conjunto de datos de pruebas. Por lo tanto, el cambio en las dimensiones de las imágenes se realizó con el objetivo de mejorar la calidad y la estabilidad del conjunto de datos, lo que a su vez se espera que contribuya a un entrenamiento más efectivo y preciso del modelo de detección del lenguaje de señas mexicano.

- Numero de épocas es igual a 50. Es evidente una mejora significativa en el entrenamiento, ya que se logró alcanzar un porcentaje de precisión aceptable, que supera el 95%. Sin embargo, se observa una pérdida en la precisión de las pruebas, lo que indica que el modelo puede estar sobreajustado a los datos de entrenamiento y no generaliza bien a nuevos datos. A pesar de este inconveniente en las pruebas, se puede concluir que el cambio en el tamaño de las imágenes fue favorable para el entrenamiento, ya que permitió alcanzar un alto nivel de precisión en el conjunto de datos de entrenamiento.

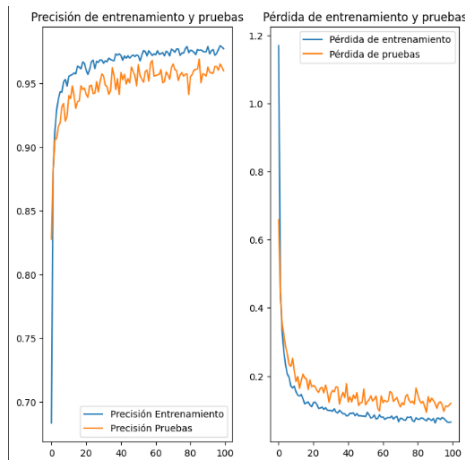
Ilustración 48. Grafica de precisión y perdida de entrenamiento y pruebas a 30 épocas.



Fuente: autoría propia.

- Numero de épocas es igual a 100. Se logró alcanzar un porcentaje aceptable tanto en la precisión de los datos de entrenamiento como en los datos de prueba con la cantidad de 100 épocas. Esta observación indica que el modelo ha logrado aprender de manera efectiva los patrones presentes en los datos de entrenamiento y puede generalizar bien a datos nuevos. Este hito es significativo, ya que demuestra que el modelo es capaz de detectar el lenguaje de señas mexicano con una precisión aceptable después de un entrenamiento más prolongado.

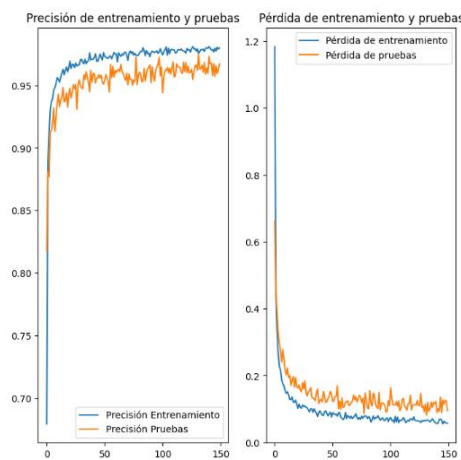
Ilustración 49. Grafica de precisión y perdida de entrenamiento y pruebas a 100 épocas.



Fuente: autoría propia.

- Numero de épocas es igual a 150. El último entrenamiento realizado con 150 épocas mostró resultados significativamente mejores, con un mayor porcentaje de precisión y una mínima pérdida en los datos. Esto indica que el modelo ha alcanzado su mejor desempeño hasta ahora y ofrece una mayor confianza en su capacidad para detectar el lenguaje de señas mexicano. Por lo tanto, este modelo se elige como el modelo final debido a su excelente rendimiento.

Ilustración 50. Grafica de precisión y perdida de entrenamiento y pruebas a 150 épocas.



Fuente: autoría propia.

CAPÍTULO V

CONCLUSIONES

5.1 Conclusiones del proyecto, recomendaciones, experiencia profesional y personal adquirida.

Durante el desarrollo del proyecto, se aplicaron diversas competencias adquiridas en la carrera de ingeniería en Mecatrónica, lo que permitió llevar a cabo una investigación sobre los desafíos que enfrentan las personas con discapacidades, como los sordomudos, al comunicarse con la sociedad. A pesar de los avances en inclusión, estas personas aún encuentran obstáculos en su interacción diaria. La ingeniería ofrece soluciones a través de herramientas como el análisis de imágenes.

Tradicionalmente, el análisis de imágenes se ha asociado con la edición de fotografías, utilizando filtros para ajustar aspectos como el color, el tono y la nitidez. Sin embargo, su aplicación va más allá. Con el uso de bibliotecas como OpenCV en Python, es posible realizar un análisis más profundo de las imágenes. Esta herramienta permite filtrar colores en las imágenes para generar vectores y matrices que pueden ser interpretados digitalmente por un computador. Así, es posible detectar y comparar imágenes similares basadas en sus características.

Con comandos simples, esta información obtenida del análisis de imágenes puede ser utilizada para desarrollar una aplicación ejecutable capaz de detectar y analizar imágenes según las necesidades del usuario. En el contexto de este proyecto, se enfoca en la detección del lenguaje de señas mexicano, ofreciendo una solución práctica para facilitar la comunicación de las personas con discapacidad auditiva.

5.2 Conclusiones relativas al objetivo general

Durante el desarrollo de la investigación, se persiguió el cumplimiento del objetivo general, a pesar de los desafíos planteados por la cantidad de épocas necesarias para alcanzar el 80% de predicción deseado. Sin embargo, como se mencionó anteriormente, con un entrenamiento que abarcó desde 50 hasta 150 épocas, se logró alcanzar el umbral requerido para cumplir con el objetivo general de la investigación. Se tomó en consideración la necesidad de evitar un sobre

entrenamiento de la red, evitando excederse en el número de épocas, lo que podría haber generado resultados sesgados.

5.3 Limitaciones del modelo planteado

La investigación está limitada ya que solo detecta las letras que son estáticas, en el abecedario de lengua de señas mexicano (LSM) existen ciertas letras que necesitan hacer un movimiento para poder diferenciarse de otra. Al igual se tiene la limitación del sistema de entrenamiento de google colab, ya que, si bien este funcionaba, el tiempo de carga era bastante largo, por lo que el sistema podía llegar a fallar por déficit de potencia de equipo.

5.4 Recomendaciones

Para el desarrollo del proyecto, se pueden aplicar diferentes tipos de análisis de imagen, que pueden ayudar a mejorar la detección a futuro, de la misma manera, implementar un banco de imágenes más amplio y con un mayor número de manos con diferentes características, cuidando ser imágenes de mejor calidad para realizar el análisis sería adecuado si se busca una mejor detección. También cabe mencionar que para el procesamiento de las imágenes muestra, el tener un equipo con mayor potencia puede ayudar a dar mejores y más rápidos resultados. Así como la implementación de otro lenguaje de programación como HTML puede ayudar si se consideran otras aplicaciones a las planteadas para esta investigación, como la implementación de una página web o una aplicación móvil para portátiles, que pueda generar un mayor impacto en la sociedad que necesita este servicio.

CAPÍTULO VI

COMPETENCIAS DESARROLLADAS

6.1 competencias desarrolladas y/o aplicadas

- Capacidad de aplicación de conocimientos en la práctica.
- Conocimientos básicos de la carrera ingeniería Mecatrónica.
- Habilidades de investigación.
- Capacidad de síntesis y análisis.
- Habilidad de manejo de software de ingeniería.
- Habilidad para búsqueda y análisis de información proveniente de fuentes diversas.
- Solución de problemas.
- Conoce y analiza el lenguaje de señas mexicano.

CAPÍTULO VII

FUENTES DE INFORMACIÓN

7.1 Fuentes de información

- Abril Cruz, N. (s.f.). *Repositorio*. Obtenido de Repositorio Unidades : <https://repositorio.uniandes.edu.co/flip/?pdf=https://repositorio.uniandes.edu.co/server/api/core/bitstreams/ea53fe19-2ac8-4038-a812-a72634a6acaa/content>
- Alvear Puertas, V., Rosero Montalvo, P., Peluffo Ordóñez, D., & Pijal Rojas, J. (febrero de 2017). *Internet de las Cosas y Visión Artificial, Funcionamiento y de Literatura*. Obtenido de Scielo: <http://scielo.senescyt.gob.ec/pdf/enfoqueute/v8s1/1390-6542-enfoqueute-8-s1-00244.pdf>
- Azure. (2023). *¿Qué es el aprendizaje profundo?* Obtenido de Azure: <https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-deep-learning/>
- Consejo Nacional para el Desarrollo y la Inclusión de las Personas con Discapacidad. (10 de junio de 2019). *Día nacional de la Lengua de Señas Mexicana (LSM)*. Obtenido de Gobierno de México: <https://www.gob.mx/conadis/articulos/dia-nacional-de-la-lengua-de-senas-mexicana-lsm-203888?idiom=es>
- Cruz Aldrete , M., Pool Westgaard, M., Mercader Flores, C. A., Escobar De, Escobar Dellamary, L., & Ramirez Barba, M. G. (25 de septiembre de 2017). *Diccionario de Lengua de Señas Mexicana Ciudad de México*. Obtenido de ACADEMIA: https://www.academia.edu/38402018/DICCIONARIO_LSM
- García Santillán, I. D., & Caranqui Sánchez, V. M. (2015). LA VISIÓN ARTIFICIAL Y LOS CAMPOS DE APLICACIÓN. En I. D. García Santillán, & V. M. Caranqui Sánchez, *LA VISIÓN ARTIFICIAL Y LOS CAMPOS DE APLICACIÓN* (págs. 98-108). Tulcán, Ecuador: Tierra Infinita.
- González Marcos, A., Martínez de Pisán Ascacibar, F. J., Pernía Espinoza, A. V., Alba Elías, F., Castejón Limas, J., & Vergara González, E. (2006). TÉCNICAS Y ALGORITMOS BÁSICOS DE VISIÓN ARTIFICIAL. En A. González Marcos, F. J. Martínez de Pisán Ascacibar, A. V. Pernía Espinoza, F. Alba Elías, J. Castejón Limas, & E. Vergara González, *TÉCNICAS Y ALGORITMOS BÁSICOS DE VISIÓN ARTIFICIAL* (págs. 11-16). España: Universidad de La Rioja, Servicio de Publicaciones.
- IBM. (2021). *¿Qué son las redes neuronales?* Obtenido de IBM: <https://www.ibm.com/es-es/topics/neural-networks>

- IBM. (s.f.). *¿Qué son las redes neuronales convolucionales?* Obtenido de IBM: <https://www.ibm.com/mx-es/topics/convolutional-neural-networks>
- Martín Ortiz, M. (enero de 2013). *Procesamiento digital de imagenes*. Obtenido de Cs Buap: <http://www.cs.buap.mx/~mmartin/notas/PDI-MM-Rev.2013.pdf>
- MathWorks. (s.f.). *Redes neuronales convolucionales*. Obtenido de MathWorks: <https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html#:~:text=Una%20red%20neuronal%20convolucional%20%28CNN%20o%20ConvNet%29%20es,el%20fin%20de%20reconocer%20objetos%2C%20clases%20y%20categorias%3%ADas>.
- Numpy Developers. (2022). *What is NumPy?* Obtenido de Numpy: <https://numpy.org/doc/stable/user/whatisnumpy.html>
- OpenCV. (2023). *About*. Obtenido de OpenCV: <https://opencv.org/about/>
- Oracle. (31 de mayo de 2022). *¿Qué es Python?* Obtenido de Artículos técnicos : <https://developer.oracle.com/es/learn/technical-articles/what-is-python>
- Oracle. (4 de mayo de 2022). *Qué es TensorFlow: una guía completa*. Obtenido de Artículos técnicos: <https://developer.oracle.com/es/learn/technical-articles/what-is-tensorflow>
- Oracle. (2023). *¿Qué es el aprendizaje profundo?* Obtenido de Oracle: <https://www.oracle.com/mx/artificial-intelligence/machine-learning/what-is-deep-learning/>
- Python Software Foundation. (2023). *Interfaz de Python a Tcl/Tk*. Obtenido de Python: <https://docs.python.org/3/library/tkinter.html>
- REAL ACADEMIA ESPAÑOLA. (2014). *REAL ACADEMIA ESPAÑOLA*. Obtenido de REAL ACADEMIA ESPAÑOLA: <https://dle.rae.es>
- Secretaría de salud. (28 de noviembre de 2021). *530. Con discapacidad auditiva, 2.3 millones de personas: Instituto Nacional de Rehabilitación*. Obtenido de Gobierno de México: <https://www.gob.mx/salud/prensa/530-con-discapacidad-auditiva-2-3-millones-de-personas-instituto-nacional-de-rehabilitacion?idiom=es>
- Torres Domínguez, A. (abril de 1996). *Procesamiento digital de imagenes*. Obtenido de Redalyc.
- Universidad Ramón Areces . (s.f.). *Interfaces de Usuario*. Obtenido de Ceresa: <https://www.ceresa.es/media/areces/files/book-attachment-3027.pdf>
- Rodríguez Villarroel, Juan Pablo, Ponce de León Espinoza, Nicolás, & Arteaga Sabja, Wendoline. (2021). Multilayer and convolutional neural networks for Bolivian Sign

Language recognition: an empirical evaluation. *Acta Nova*, 10(1), 22-41. Recuperado en 09 de febrero de 2024, de http://www.scielo.org.bo/scielo.php?script=sci_arttext&pid=S1683-07892021000100003&lng=es&tlng=en

CAPÍTULO VIII

ANEXOS

Anexo 1. Asignación de Asesor(a), Comisión Revisora, Entrega de Trabajo Profesional y Dictamen

**EDUCACIÓN**
SECRETARÍA DE EDUCACIÓN PÚBLICA



**TECNOLÓGICO**
NACIONAL DE MÉXICO

Instituto Tecnológico Superior
de Teziutlán

Asunto: **Asignación de Asesor(a), Comisión Revisora, Entrega de Trabajo Profesional y Dictamen**

Teziutlán, Puebla, **15/enero/2024**

Asesor(a): **ALFREDO CARRASCO ARAOZ**
Integrante de Comisión Revisora: **GABRIEL ANGEL RAMIREZ VICENTE**
Integrante de Comisión Revisora: **MONICA ADRIANA ANASTACIO MENDOZA**
Presentes

Por este medio me permito informar que ha sido asignado como asesor(a) y comisión revisora del trabajo profesional que se convertirá en Tesis de:

Alumno(a): **PRADO NICOLAS VANESSA**
Apellido paterno/materno/nombre (s)

Número de Control: **19TE0681** Licenciatura o Posgrado: **INGENIERÍA MECATRONICA**

Plan: **2010** Correo Electrónico: **L19TE0681@TEZIUTLAN.TECNM.MX**

Cuyo tema es: **DISEÑO DE SOFTWARE PARA LA DETECCIÓN DE LENGUAJE DE SEÑAS MEXICANO UTILIZANDO TÉCNICA DE VISIÓN ARTIFICIAL**

25 palabras (máximo)

Se ha enviado a su correo institucional el trabajo profesional o de grado, por lo cual la comisión revisora tendrá 5 días hábiles para realizar las observaciones al alumnado, el(la) interesado(a) tendrá igualmente **5 días** para corregir y las enviará al correo electrónico institucional de la comisión revisora, agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de licenciatura o de posgrado de nuestro alumnado egresado.

Dictamen de Comisión Revisora, Aprobación de Impresión o Grabación y Autorización para subirla al Repositorio del TecNM

Siendo el día: 19 de febrero de 2024 se reunieron los miembros de la comisión para revisar el trabajo asignado y una vez analizado se decidió liberarlo y aprobarlo para su grabación y programación de examen profesional.


ALFREDO CARRASCO ARAOZ
Nombre y Firma del(la) Asesor(a)


GABRIEL ANGEL RAMIREZ VICENTE
Nombre y Firma de integrante de la Comisión Revisora


MONICA ADRIANA ANASTACIO MENDOZA
Nombre y Firma de integrante de la Comisión Revisora


MYRIAM SANCHEZ PEREZ
Subdirección Académica

R08/06/2023
Folio:44

**GOBIERNO DEL ESTADO**
S.E.P.
INSTITUTO TECNOLÓGICO
SUPERIOR DE TEZIUTLÁN
SUBDIRECCIÓN
ACADÉMICA

F-SAC-18



Fracción I y II s/n Aire Libre, Teziutlán, Puebla, C.P. 73960 Tels. 231 311 4000 / 4001 / 4002 / 4003
e-mail: itsteziutlan@hotmail.com | www.teziutlan.tecnm.mx

**2024**
Felipe Carrillo
PUERTO
GOBIERNO DEL ESTADO DE PUEBLA

Fuente: Subdirección académica, Instituto Tecnológico Superior de Teziutlán, 2024.

Anexo 2. Licencia de uso.

LICENCIA DE USO OTORGADA POR Vanessa Prado Nicolás, de nacionalidad Mexicana, mayor de edad, con domicilio ubicado en la calle Av. Nautla 360, novena del Carmen, 73820 Teziutlán, Puebla, en mi calidad de titular de los derechos patrimoniales y morales y autor(a) de la tesis denominada "Diseño de software para la detección de lenguaje de señas mexicano utilizando técnica de visión artificial" en adelante "LA OBRA" quien para todos los fines del presente documento se denominará "EL(LA) AUTOR(A) Y/O EL(LA) TITULAR", a favor del Instituto Tecnológico Superior de Teziutlán del Tecnológico Nacional de México, la cual se registrará por las cláusulas siguientes:

PRIMERA –OBJETO: "EL(LA) AUTOR(A) Y/O EL(LA) TITULAR", mediante el presente documento otorga al Instituto Tecnológico Superior de Teziutlán del Tecnológico Nacional de México, licencia de uso gratuita e indefinida respecto de "LA OBRA", para almacenar, preservar, publicar, reproducir y/o divulgar la misma, con fines académicos, por cualquier medio en forma física y a través del repositorio institucional y del repositorio nacional, éste último consultable en la página: (<https://www.repositorionacionalcti.mx/>).

SEGUNDA - TERRITORIO: La presente licencia se otorga, de manera no exclusiva, sin limitación geográfica o territorial alguna, de manera gratuita e indefinida.

TERCERA -ALCANCE: La presente licencia contempla la autorización para formato uso de "LA OBRA" en cualquier formato o soporte material y se extiende a la utilización, de manera enunciativa más no limitativa a los siguientes medios: óptico, magnético, electrónico, virtual (en red), mensaje de datos o similar, conocido o por conocerse.

CUARTA – EXCLUSIVIDAD: La presente licencia de uso aquí establecida no implica exclusividad en favor del Instituto Tecnológico Superior de Teziutlán; por lo tanto, "EL(LA) AUTOR(A) Y/O EL(LA) TITULAR" conserva los derechos patrimoniales y morales de "LA OBRA", objeto del presente documento.

QUINTA – CRÉDITOS: El Instituto Tecnológico Superior de Teziutlán y/o el Tecnológico Nacional de México reconoce que "EL(LA) AUTOR(A) Y/O EL(LA) TITULAR" es el(la) único(a), primigenio(a) y perpetuo(a) titular de los derechos morales sobre "LA OBRA"; por lo tanto, siempre deberá otorgarle los créditos correspondientes por la autoría de la misma.

SEXTA – AUTORIA: "EL(LA) AUTOR(A) Y/O EL(LA) TITULAR" manifiesta ser el(la) único(a) titular de los derechos de autor que derivan de "LA OBRA" y declara que el material objeto del presente fue realizado por él(ella), sin violentar o usurpar derechos de propiedad intelectual de terceros; por lo tanto, en caso de controversia sobre los mismos, se obliga a ser el(la) único(a) responsable. Dado en la Ciudad de Teziutlán, Puebla, a los diecinueve días del mes febrero de 2024.

"EL(LA) AUTOR(A) Y/O
EL(LA) TITULAR",



Vanessa Prado Nicolás
Nombre y Firma

"EL INSTITUTO TECNOLÓGICO
SUPERIOR DE TEZIUTLÁN" S.E.P.

INSTITUTO TECNOLÓGICO
SUPERIOR DE TEZIUTLÁN
DIRECCIÓN GENERAL

Arminda Juárez Arroyo
Directora General
Nombre, Firma y Sello

Fuente: Subdirección académica, Instituto Tecnológico Superior de Teziutlán, 2024.

Anexo 3. Carta de autorización del(a) autor(a) para la consulta y publicación electrónica del trabajo de investigación.

Tecnológico Nacional de México
Instituto Tecnológico Superior de Teziutlán

CARTA DE AUTORIZACIÓN DEL(LA) AUTOR(A) PARA LA CONSULTA Y PUBLICACIÓN ELECTRÓNICA DEL TRABAJO DE INVESTIGACIÓN

El que suscribe:

VANESSA

PRADO

NICOLAS

Con Número de
Control

19TE0681

Perteneciente al
Programa Educativo

INGENIERÍA MECATRÓNICA

Por este conducto me permito informar que he dado mi autorización para la consulta y publicación electrónica del trabajo de investigación en los repositorios académicos.

Registrado con el
producto:

TESIS

Cuyo Tema es:

**DISEÑO DE SOFTWARE PARA LA DETECCIÓN DE LENGUAJE DE SEÑAS MEXICANO UTILIZANDO
TECNICA DE VISION ARTIFICIAL**

Correspondiente al periodo:

AGOSTO-DICIEMBRE 2023

Y cuyo(a) director(a) de tesis es:

M.S.C ALFREDO CARRASCO ARAOZ

ATENTAMENTE



VANESSA PRADO NICOLAS

Nombre y firma

Fecha de emisión: **19/02/2024**
c.c.p. Subdirección Académica

Fuente: Subdirección académica, Instituto Tecnológico Superior de Teziutlán, 2024.

Anexo 4. Código de entrenamiento en el Google Colab.

```
# !/usr/bin/env python
# -- coding: utf-8 --
# -- coding: iso-8859-1 -
# "/dev/ttyUSB0"

import cv2
from tkinter import *
from PIL import Image
from PIL import ImageTk
import json
import numpy as np
import tensorflow as tf
from tensorflow import keras
import tensorflow_hub as hub

'''abrimos el modelo'''
with open('model.json', 'r') as archivo_lsm:
    datos_lsm = archivo_lsm.read()

modelo_lsm = keras.models.model_from_json(json.dumps(datos_lsm))

'''variables iniciadas'''
frame_procesar = None
resultado = None
respuesta = None

'''programa principal'''
def procc_lsm():

    global selec, frame_procesar

    selec = False

    def inicio():
        global selec, frame_procesar

        visualizar()
        botonDetectar.config(state='normal')
        selec = False
        return

    def visualizar():
        global selec

        if captura is not None:
            ret, frame = captura.read()
            cv2.rectangle(frame, (100, 50), (324, 274), (0, 255, 0),
2)

            if ret == True:
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                im = Image.fromarray(frame)
                img = ImageTk.PhotoImage(image=im)
                etiquetaVideo.configure(image=img)
                etiquetaVideo.image = img
```

```

        etiquetaVideo.after(10, visualizar)
        if selec == True:
            frame_procesar = frame [50:274, 100:324]
            cv2.imshow('Procesamiento', frame_procesar)
            selec = False
            return frame_procesar
    else:
        etiquetaVideo.image = ""
        Button.configure(state="disabled")
        captura.release()

def deteccion_lsm():
    '''metodo en donde se hara el procesamiento de imagen '''
    global selec, frame_procesar, resultado, respuesta

    selec = True

    if frame_procesar is not None:
        #procesar imagen
        frame_procesar = cv2.cvtColor(frame_procesar,
cv2.COLOR_BGR2GRAY)
        frame_procesar = frame_procesar / 255.0
        #hacer la predicción
        tensor = np.expand_dims(frame_procesar, axis=0)
        tensor = np.expand_dims(frame_procesar, axis=3)
        resultado = modelo_lsm.predict(tensor).flatten()
        print('el valor de la predicción es ', resultado)

    if resultado == 0:
        respuesta = "A"
    elif resultado == 1:
        respuesta = "B"
    elif resultado == 2:
        respuesta = "C"
    elif resultado == 3:
        respuesta = "D"
    elif resultado == 4:
        respuesta = "E"
    elif resultado == 5:
        respuesta = "F"
    elif resultado == 6:
        respuesta = "G"
    elif resultado == 7:
        respuesta = "H"
    elif resultado == 8:
        respuesta = "I"
    elif resultado == 9:
        respuesta = "L"
    elif resultado == 10:
        respuesta = "M"
    elif resultado == 11:
        respuesta = "N"
    elif resultado == 12:
        respuesta = "O"
    elif resultado == 13:

```

```

        respuesta = "P"
    elif resultado == 14:
        respuesta = "R"
    elif resultado == 15:
        respuesta = "S"
    elif resultado == 16:
        respuesta = "T"
    elif resultado == 17:
        respuesta = "U"
    elif resultado == 18:
        respuesta = "V"
    elif resultado == 19:
        respuesta = "W"
    elif resultado == 20:
        respuesta = "Y"
    return respuesta
else:
    respuesta = "No se detecto"
    return respuesta

def finalizar_limpiar():
    etiquetaVideo.image = ""
    captura.release()
    ventanaPrin.destroy()

ventanaImagen = Toplevel(ventanaPrin)
ventanaImagen.geometry('730x700')
ventanaImagen.title('Lenguaje de señas mexicano ')
ventanaImagen.geometry('+0+0') # coloca la ventana en la esquina
superior derecha
ventanaImagen.configure(bg='#E5CCFF') # color de la ventana
ventanaImagen.focus_set()
ventanaImagen.grab_set()
ventanaImagen.transient(master=ventanaPrin)

imagen_frame = Tk.Frame(ventanaPrin)
imagen_frame = Frame(ventanaPrin, highlightcolor='#0080FF',
highlightthickness=2)
imagen_frame.place(x=20, y=20)
imagen_frame.config(width=690, height=525, bg='white', relief='flat')

canvas = Canvas(imagen_frame, bg='white')
canvas.place(x=20, y=20)

'''variables'''
resultado = IntVar()

etiquetaVideo = Label(ventanaImagen) # etiqueta en la cual se coloca
la camara
etiquetaVideo.place(x=50, y=20) # posicon de la camara
Label(ventanaImagen, textvariable=respuesta, font=('Arial Black',
12)).place(x=360, y=580)
botonInciar = Button(ventanaImagen, text='Iniciar', font=('Arial
Black', 12), width=18, height=1, bg='#99CCFF',command=inicio)

```

```

    botonIniciar.place(x=20, y=630)
    botonDetectar = Button(ventanaImagen, text='Detectar', font=('Arial
Black', 12), width=18, height=1, bg='#99CCFF', command=deteccion_lsm)
    botonDetectar.place(x=250, y=630)
    botonDetectar.config(state='disable')
    Button(ventanaImagen, text='Salir y limpiar', font=('Arial Black',
12), width=18, height=1, bg='#99CCFF',
    command=finalizar_limpiar).place(x=500, y=630)

    '''Iniciamos la caprura de video'''
    captura = cv2.VideoCapture(0, cv2.CAP_DSHOW)

    return

'''pantalla principal'''
ventanaPrin = Tk()

pross_lsm()

ventanaPrin.mainloop()

```

Anexo 2. Codigo de diseño de interfaz de usuario.

```

import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

plt.figure(figsize=(15,15))

carpeta = '/content/drive/MyDrive/lsm/V'
imagenes = os.listdir(carpeta)

for i, name_img in enumerate(imagenes[:25]):
    plt.subplot(5,5,i+1)
    imagen = mpimg.imread(carpeta + '/' + name_img)
    plt.imshow(imagen)
#Crear carpetas para hacer el set de datos

!mkdir dataset
!mkdir dataset/A
!mkdir dataset/B
!mkdir dataset/C
!mkdir dataset/D
!mkdir dataset/E
!mkdir dataset/F
!mkdir dataset/G
!mkdir dataset/H
!mkdir dataset/I

```

```

!mkdir dataset/L
!mkdir dataset/M
!mkdir dataset/N
!mkdir dataset/O
!mkdir dataset/P
!mkdir dataset/R
!mkdir dataset/S
!mkdir dataset/T
!mkdir dataset/U
!mkdir dataset/V
!mkdir dataset/W
!mkdir dataset/Y

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/A'
carpeta_destino = '/content/dataset/A'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + n

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/B'
carpeta_destino = '/content/dataset/B'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + no
#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

```

```

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/C'
carpeta_destino = '/content/dataset/C'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/D'
carpeta_destino = '/content/dataset/D'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/E'
carpeta_destino = '/content/dataset/E'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set

```

```

#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/F'
carpeta_destino = '/content/dataset/F'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/G'
carpeta_destino = '/content/dataset/G'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/H'
carpeta_destino = '/content/dataset/H'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

```

```

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/I'
carpeta_destino = '/content/dataset/I'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/L'
carpeta_destino = '/content/dataset/L'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/M'
carpeta_destino = '/content/dataset/M'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:

```

```

    #Copia de la carpeta fuente a la destino
    shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
    '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/N'
carpeta_destino = '/content/dataset/N'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/O'
carpeta_destino = '/content/dataset/O'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/P'
carpeta_destino = '/content/dataset/P'

imagenes = os.listdir(carpeta_fuente)

```

```

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/R'
carpeta_destino = '/content/dataset/R'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/S'
carpeta_destino = '/content/dataset/S'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/T'

```

```

carpeta_destino = '/content/dataset/T'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/U'
carpeta_destino = '/content/dataset/U'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/V'
carpeta_destino = '/content/dataset/V'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set

```

```

#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/W'
carpeta_destino = '/content/dataset/W'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#cargar las imagenes en las carpetas creadas del data set
#litar las carpetas con u contenido de 500 imagenes

import shutil
carpeta_fuente = '/content/drive/MyDrive/lsm/Y'
carpeta_destino = '/content/dataset/Y'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
    if i < 500:
        #Copia de la carpeta fuente a la destino
        shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
        '/' + nombreimg)

#Mostrar cuantas imagenes tengo de cada categoria en el dataset
!ls /content/dataset/A | wc -l
!ls /content/dataset/B | wc -l
!ls /content/dataset/C | wc -l
!ls /content/dataset/D | wc -l
!ls /content/dataset/E | wc -l
!ls /content/dataset/F | wc -l
!ls /content/dataset/G | wc -l
!ls /content/dataset/H | wc -l
!ls /content/dataset/I | wc -l
!ls /content/dataset/L | wc -l
!ls /content/dataset/M | wc -l
!ls /content/dataset/N | wc -l
!ls /content/dataset/O | wc -l

```

```

!ls /content/dataset/P | wc -l
!ls /content/dataset/R | wc -l
!ls /content/dataset/S | wc -l
!ls /content/dataset/T | wc -l
!ls /content/dataset/U | wc -l
!ls /content/dataset/V | wc -l
!ls /content/dataset/W | wc -l
!ls /content/dataset/Y | wc -l

#Aumento de datos con ImageDataGenerator
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

#Crear el dataset generador
datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range = 30,
    width_shift_range = 0.25,
    height_shift_range = 0.25,
    shear_range = 15,
    zoom_range = [0.5, 1.5], no
    validation_split=0.2 #20% para pruebas
)

#Generadores para sets de entrenamiento y pruebas
data_gen_entrenamiento =
datagen.flow_from_directory('/content/dataset', target_size=(224,224),
                           batch_size=32,
shuffle=True, subset='training')
data_gen_pruebas = datagen.flow_from_directory('/content/dataset',
target_size=(224,224),
                           batch_size=32,
shuffle=True, subset='validation')

#Imprimir 10 imagenes del generador de entrenamiento
for imagen, etiqueta in data_gen_entrenamiento:
    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(imagen[i])
    break
plt.show()

```

```

import tensorflow as tf
import tensorflow_hub as hub

url = "https://tfhub.dev/google/tf2-
preview/mobilenet_v2/feature_vector/4"
mobilenetv2 = hub.KerasLayer(url, input_shape=(224,224,3))

#Congelar el modelo descargado
mobilenetv2.trainable = False

modelo = tf.keras.Sequential([
    mobilenetv2,
    tf.keras.layers.Dense(21, activation='softmax')
])

#imprimir el resumen del modelo
modelo.summary()

#Compilar como siempre
modelo.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

#Entrenar el modelo
EPOCAS = 150
historial = modelo.fit(
    data_gen_entrenamiento, epochs=EPOCAS, batch_size=32,
    validation_data=data_gen_pruebas
)

#Graficas de precisión
acc = historial.history['accuracy']
val_acc = historial.history['val_accuracy']

loss = historial.history['loss']
val_loss = historial.history['val_loss']

```

```

rango_epocas = range(150)

plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(rango_epocas, acc, label='Precisión Entrenamiento')
plt.plot(rango_epocas, val_acc, label='Precisión Pruebas')
plt.legend(loc='lower right')
plt.title('Precisión de entrenamiento y pruebas')

plt.subplot(1,2,2)
plt.plot(rango_epocas, loss, label='Pérdida de entrenamiento')
plt.plot(rango_epocas, val_loss, label='Pérdida de pruebas')
plt.legend(loc='upper right')
plt.title('Pérdida de entrenamiento y pruebas')
plt.show()

#Categorizar una imagen de internet
from PIL import Image
import requests
from io import BytesIO
import cv2

def categorizar(url):
    respuesta = requests.get(url)
    img = Image.open(BytesIO(respuesta.content))
    img = np.array(img).astype(float)/255

    img = cv2.resize(img, (224,224))
    prediccion = modelo.predict(img.reshape(-1, 224, 224, 3))
    return np.argmax(prediccion[0], axis=-1)

#a=0  b=1  c=2 d=3  e=4  f=5  g=6  h=7  i=8
L=9  m=10  n=11  o=12  p=13  r=14  s=15  t=16  u=17  v=18  w=19  y=
20
url =
'https://th.bing.com/th/id/OIP.H0IICr_VPHkZ0FY2bo8z5QHaLH?pid=ImgDet&w
=184&h=276&c=7'
prediccion = categorizar (url)
print(prediccion)

#Crear la carpeta para exportarla a TF Serving
!mkdir -p carpeta_salida/modelo_lsm/1

```

```

#Guardar el modelo en formato SavedModel
modelo.save('carpeta_salida/modelo_lsm/1')

#Hacerlo un zip para bajarlo y usarlo en otro lado
!zip -r modelo_lsm.zip /content/carpeta_salida/modelo_lsm/

#guardar el modelo como h5
modelo.save('modelo_exportado.h5')

#Instalar tensorflowjs para convertir el h5 a un modelo que pueda
cargar tensorflowjs en un explorador
!pip install tensorflowjs

!mkdir tfjs_target_dir
!tensorflowjs_converter --input_format keras modelo_exportado.h5
tfjs_target_dir

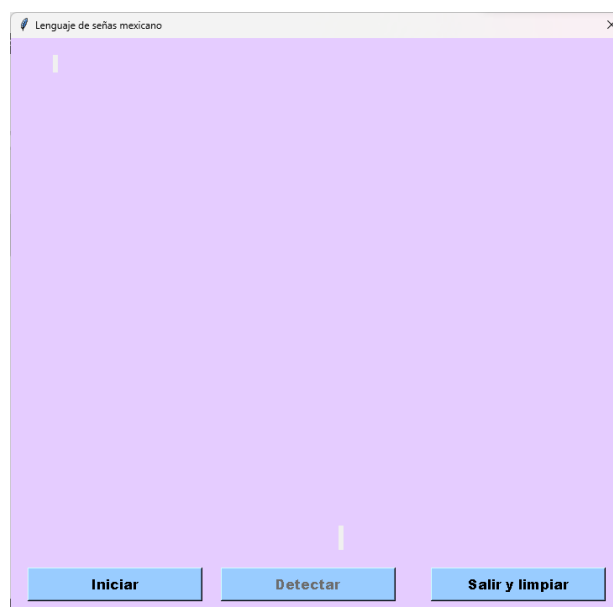
```

Anexo 3

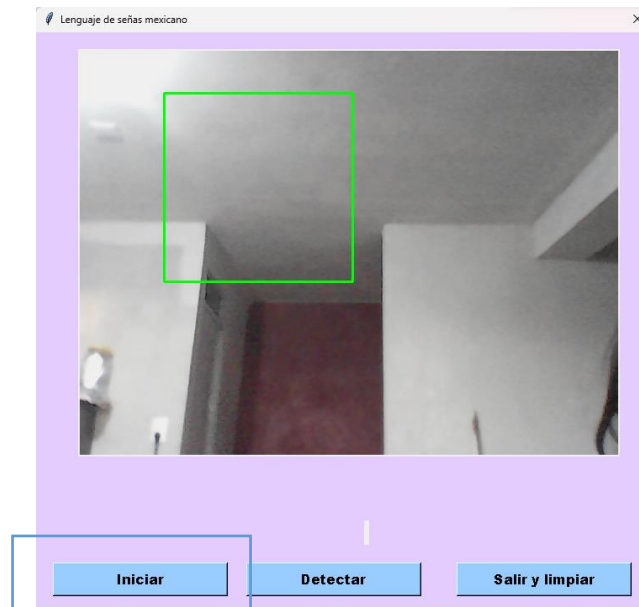
A continuación, se muestra el manual de usuario para la utilización del software.

Manual de usuario

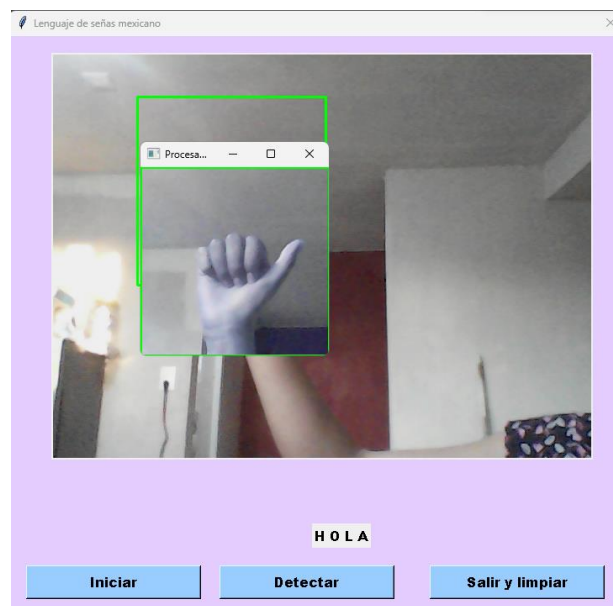
Al iniciar el programa se mostrara la siguiente interfaz de usuario.



Con el boton *Iniciar* se inicializa la camara.



Ya con la camara inicializada el bonton *Detectar* se activa, con la finalidad de seguir con la detección. Es de vital imprtancia hacer las señas dentro del recuadro marcado en la camara. En la parte inferior se muestra una etiqueta en la cual mestra el mensaje formado por la lengua de señas.



Por ultimo el bontón *Salir y limpiar*, cierra el programa.