Universidad Tecnológica de la Mixteca

División de Estudios de Posgrado

Desarrollo e implementación de un sistema de identificación de signos de la lengua de señas mexicana basado en redes neuronales y el sistema embebido Jetson Nano

Tesis

Para obtener el grado de:

Maestro en Electrónica, Opción: Sistemas Inteligentes Aplicados

Presenta:

Ing. Alfredo Antonio Estévez Acosta

Director de tesis:

Dr. Rosebet Miranda Luna

Co-director de tesis:

Dr. José Anibal Arias Aguilar

Huajuapan de León, Oaxaca. Abril de 2022

Dedicatoria

A mis padres Rubén Estévez y Antonia Acosta, por su apoyo en todas las etapas de mi vida, por animarme en cada dificultad y por sus incansables oraciones. A mi hermana Sandra Estévez, por estar al pendiente de cada paso. A mi abuelita Asunción Martínez y mi tía María Martínez por sus cuidados y atenciones en todo momento. Al Dios y creador eterno, por rescatar mi vida y darme una nueva visión y un nuevo propósito.

Agradecimientos

Mi profundo agradecimiento a la Universidad Tecnológica de la Mixteca y a la División de Estudios de Posgrado, a sus profesores, personal administrativo y de limpieza, por generar un espacio de excelencia académica en todos los sentidos.

Agradezco a mi director de tesis, Dr. Rosebet Miranda Luna por su dedicación, atención y correcciones en éste trabajo de tesis, así como al Dr. José Anibal Arias Aguilar quien fungió como co-director de tesis brindando apoyo oportuno. De igual manera, a los sinodales asignados, cuyas aportaciones guiaron el desarrollo de éste proyecto.

Así mismo al Consejo Nacional de Ciencia y Tecnología por el apoyo económico brindado durante mi estancia en el posgrado.

Por último y no menos importante a todos aquellos familiares, amigos y líderes que hicieron posible el poder llegar a la culminación de ésta etapa, proveyéndome de necesarios consejos, apoyo físico, emocional y económico. Mi agradecimiento siempre será con mi Dios y con ustedes.

Índice general

Índice de figuras

Índice de tablas

Resúmen

En el presente trabajo de tesis se muestran las etapas para el desarrollo de una red neuronal convolucional (RNC) la cual permite clasificar en tiempo real signos de la lengua de señas Mexicana. Se pueden identificar cuatro partes principales en esta investigación; la recolección, limpieza y tratamiento de las imágenes de la base de datos, el desarrollo de la RNC en la plataforma *Google Colaboratory*, la obtención de un modelo compatible con la tarjeta Jetson Nano y la ejecución en tiempo real del modelo de clasificación el cual muestra un 94.85 % de exactitud ante un conjunto de pruebas de 2,100 imágenes, dicho modelo fue obtenido a través de la optimización de una RNC propuesta variando el número de filtros, capas de convolución y valor del optimizador para 10 modelos de RNC basados en la RNC propuesta. La base de datos en general contiene 10,500 imágenes en escala de grises la cual es una integración de bases de datos de la lengua de señas americana, bengalí y colombiana, así como imágenes propias adquiridas. Se presentan además los algoritmos necesarios para transformar un modelo de RNC a un archivo el cual puede ser ejecutado por la tarjeta Jetson Nano. Finalmente, se muestran las pruebas realizadas en un entorno físico y con voluntarios con las cuales se comprobó el funcionamiento de este sistema permitiendo observar el reconocimiento en tiempo real para cada signo estático de la LSM realizado por los usuarios.

Capítulo 1

Introducción

La lengua de signos o de señas es un lenguaje natural de expresión y configuración gestoespacial y percepción visual, gracias a la cual las personas sordas pueden establecer un canal de comunicación con su entorno social, ya sea entre otras personas sordas o cualquier persona que conozca la lengua de señas empleada, en este caso, la lengua de señas mexicana (LSM), ya que al contrario de lo que comúnmente se piensa, no existe una única lengua de signos, de hecho, cada país cuenta con su propia lengua de signos, en la que tanto las señas empleadas para representar las letras como las empleadas para representar palabras completas son distintas según el idioma y el país en el que uno se encuentre. De hecho, según la Federación Mundial de Sordos existen alrededor de 300 lenguas de signos distintas en todo el mundo (Voisard (2015)). El propósito de las lenguas de señas en cada país y, por tanto, de la lengua de señas mexicana, es permitir la inclusión de un sector de la población cuya limitación es la auditiva, más no la intelectual y/o social.

1.1. Estado del arte

La identificación de signos de la lengua de señas, según Matallana (2019), se realiza principalmente en tres etapas:

- Detección de las manos.
- Adquisición y procesamiento de la imagen y,
- Reconocimiento del signo.

La detección de las manos y su posición se puede realizar utilizando guantes electrónicos, sensores de Leap Motion, sensores Kinect o haciendo uso únicamente de

una cámara digital.

Por su parte el uso de guantes (ver Figura 1.1) presentan la ventaja de que es fácil obtener información sobre el grado de flexión de los dedos y la posición expresada en tres dimensiones, así como de la orientación de la mano que lleva el guante, por lo que exige menos poder de cómputo para el sistema en el que se implementa, siendo así más sencillo lograr la traducción en tiempo real. Sin embargo, presentan también varias desventajas como es el hecho de que estos guantes son bastante caros, alrededor de 600 euros según Fernández (2007) o alrededor de 2500 pesos mexicanos según los costos reportados por el proyecto de Espinosa Aguilar y Pogo León (2013), y aunque existen algunos que son más baratos, estos comúnmente utilizan sensores de bajo costo los cuales suelen ser muy susceptibles al ruido según menciona Matallana (2019).

Figura 1.1. Guantes electrónicos presentados por Espinosa Aguilar y Pogo León (2013).

Otra alternativa para la detección de las manos es mediante el uso del sensor *Leap Motion* (ver Figura 1.2), el cual es un dispositivo que se conecta a una computadora de forma análoga a un mouse, pero no requiere contacto con las manos. Posee dos cámaras con lentes de tipo biconvexas y un sistema de coordenadas propio el cual permite obtener en tiempo real el valor de la posición de los dedos y manos, sin embargo, la precisión, el espacio de operación y fidelidad de la detección de las manos sigue siendo una debilidad de este tipo de sensor como reporta Weichert y cols. (2013), a la fecha este dispositivo ya no cuenta con soporte técnico con lo cual éstas desventajas siguen estando presentes en este sensor como menciona Matney (2019).

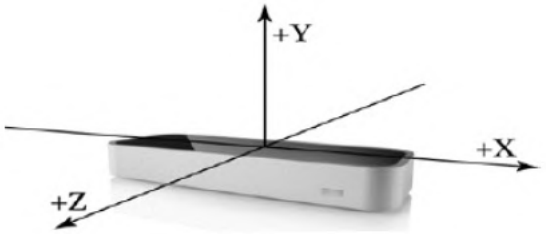


Figura 1.2. Sensor Leap Motion, Weichert y cols. (2013).

Así mismo se ha hecho uso del sensor *Kinect*, el cual, como menciona Murillo (2012) está integrado con un sensor de profundidad, una cámara RGB, un arreglo de micrófonos y un sensor de infrarrojos que actúa de emisor y de receptor con lo cual, se puede capturar, reconocer y posicionar el esqueleto humano en el plano (ver Figura 1.3).

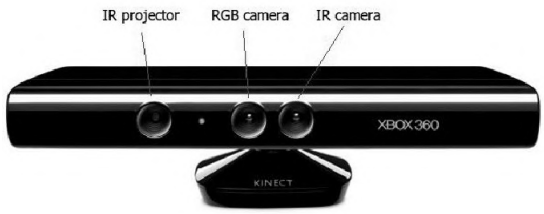


Figura 1.3. Sensor kinect, Murillo (2012).

Por último, se ha hecho uso de la cámara digital la cual surge como una forma de evitar usar hardware adicional y más complejo como los mencionados anteriormente, sin embargo, debido a que la obtención del posicionamiento y las coordenadas de la mano no es automática, es necesario por tanto realizar una segmentación de la imagen para aislar la mano del resto del cuerpo. Como ejemplo de esto se encuentra la tesis realizada por Matallana (2019) titulada "Traductor automático de la lengua de signos española a texto mediante visión por computador y redes neuronales".

El primer paso que realizaron consistió en detectar la piel como se muestra en la Figura 1.4.

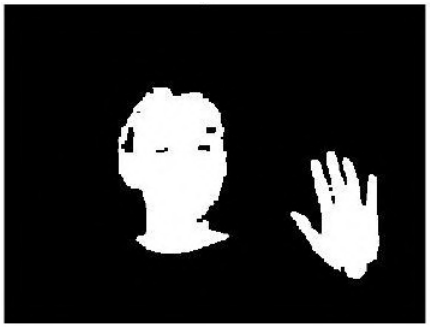


Figura 1.4. Sistema desarrollado por Matallana (2019), primera parte.

El siguiente paso consistió en la detección del rostro para posteriormente ser eliminado (ver Figura 1.5).

Figura 1.5. Sistema desarrollado por Matallana (2019), segunda parte.

Los algoritmos utilizados para aislar la mano del resto del cuerpo y obtener una imagen en blanco y negro fueron basados en técnicas de procesamiento de imágenes y siendo auxiliados de las bibliotecas disponibles en Visual Studio como lo es OpenCV.

Pérez y cols. (2017) presentan una metodología para el reconocimiento de signos LSM estáticos valiéndose de técnicas de visión artificial y clustering (ver Figura 1.6). Para la segmentación de las imágenes utiliza un algoritmo llamado Fuzzy C Means (FCM), el cual permite asociar cada dato de entrada a múltiples grupos teniendo en cada uno diferentes grados de membresía.

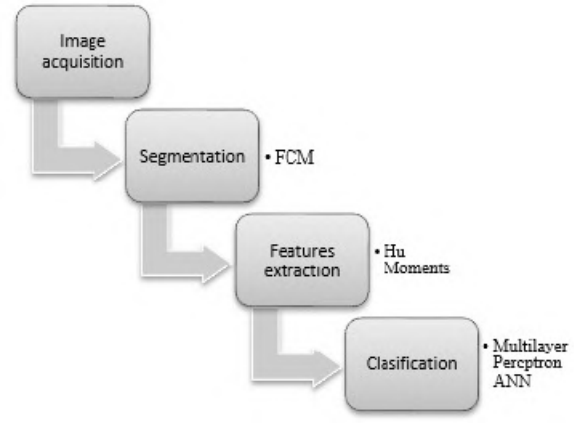


Figura 1.6. Diagrama de bloques propuesto por Pérez y cols. (2017).

Para la extracción de las características se usan descriptores geométricos conocidos como momentos de Hu los cuales son una herramienta matemática que proveen siete momentos para identificar las características geométricas de los objetos.

Las características geométricas obtenidas por estos siete momentos son usadas como entradas en un clasificador, dicho clasificador se basa en una red neuronal multicapa tipo perceptrón. Este algoritmo permite identificar de forma aceptable las letras (A, B, C, D, E, F, G, H, I, L, M, N, O, P, S, T, U, V, W, Y) y con un desempeño menor la letra R.

Para entender el desempeño del clasificador se utilizan matrices de confusión las cuales muestran de forma general un 91 % de aciertos, excepto en caso de la letra R en el cual el porcentaje se ve reducido a un 70 %.

Se escogieron esas letras del abecedario ya que son las que se representan de forma estática sin hacer ningún movimiento de manos.

Figura 1.7. Imagen segmentada con el algoritmo FCM (Pérez y cols. (2017)).

Jimenez y cols. (2017) realizaron el reconocimiento de las manos (ver Figura 1.7) mediante un dispositivo Kinect para las señas (A, B, C, D, E, 1, 2, 3, 4, 5). Emplearon información de características tipo Haar 3D de imágenes a profundidad capturadas con el sensor Kinect. Las características 3D propuestas fueron clasificadas con el algoritmo de aprendizaje Adaboost. Para analizar los resultados se hizo un promedio de las medidas de verosimilitud: sensibilidad (tasa de positivos verdaderos), especificidad (tasa de negativos verdaderos) y el F1score, con lo cual de forma general se aproximó la eficiencia en un 95 %. En la Figura 1.8 se muestra la arquitectura construida antes mencionada.

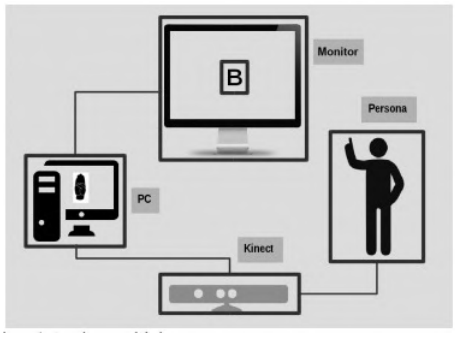


Figura 1.8. Arquitectura del sistema propuesto por Jimenez y cols. (2017).

Los errores obtenidos en la clasificación se deben principalmente a fallos en el proceso de captura tal como reporta Jimenez y cols. (2017) y como se muestra en la figura (1.9).

Figura 1.9. Errores de clasificación en el sistema desarrollado por Jimenez y cols. (2017).

Montaño y Rodríguez-Aguilar (2011) presentan un sistema de traducción automático de la Lengua de Señas Mexicana (LSM) a texto. El sistema se basa en reconocimiento de patrones sin hacer uso de redes neuronales. Este reconocimiento se aplica continuamente usando una cámara digital como dispositivo de captura y el IDE Visual Studio, específicamente la librería OpenCV.

El primer paso consiste en hacer un procesamiento de la imagen, en el cual el color RGB (rojo, verde, azul) original se transforma en HSV (matiz, saturación, brillo), además de aplicar diversos filtros para eliminar el ruido. Después es implementado el algoritmo Pavlidis con el cual se obtienen las coordenadas del borde entre los pixeles blancos y negros. Estas coordenadas se almacenan como vectores de puntos. Con ayuda de otros dos algoritmos (*convex hull* y *minimun enclosing circle*) que se implementan de igual forma haciendo uso de la librería OpenCV, se detectan las puntas de los dedos, la palma de la mano y otros puntos clave para determinar posteriormente el signo como lo muestra la figura (1.10).

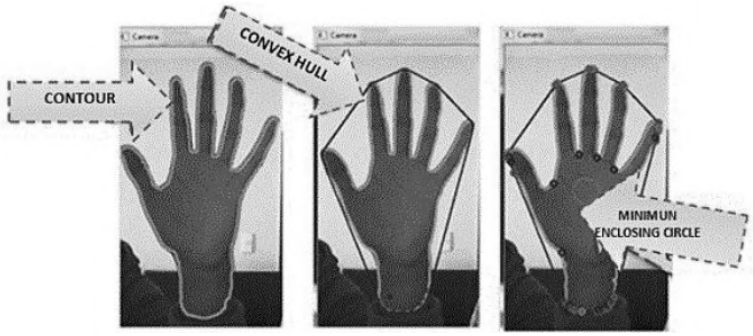


Figura 1.10. Algoritmo Pavlidis, *convex hull* y *minimun enclosing circle* desarrollado por Montaño y Rodríguez-Aguilar (2011).

Los resultados están basados en el promedio de aciertos y muestran una exactitud del 80 % reconociendo números (1, 2, 3, 4, 5, 6, 7, 8), 68 % cuando se trata de letras (A, B, C, D, E, F, G, H, I, L, M, N, O, P, R, S, T, U, V, W, Y) y 54 % de exactitud cuando se trata de reconocer palabras (ABAJO, ADENTRO, CEREZA, CISNE, CONSUEGRO, COPA, HOGAR, JIRAFA, MADRE, MANO, MENOS, MESA, MUCHO, MUÑECA, OTOÑO, PESOS, PISTOLA, RESTA, SOLO, SUMA, TORTA, UÑA, VACA, YO).

Nájera y cols. (2016) presentan un artículo llamado "Reconocimiento de lengua de señas mexicana a través de un controlador del tipo Leap Motion". En este trabajo solo se presenta el gesto obtenido de una palabra que se muestra en la Figura 1.11 dejando para trabajos futuros la implementación de algoritmos que permitan el reconocimiento de patrones y el uso de una red neuronal multicapa tipo Perceptrón.

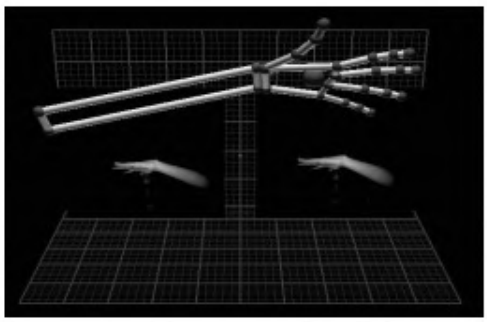


Figura 1.11. Gesto obtenido por el controlador Leap Motion, Nájera y cols. (2016).

Solís y cols. (2016) presentan un sistema que utiliza una cámara web como dispositivo de captura, para el procesamiento de las imágenes implementa momentos normalizados y en la etapa de reconocimiento hace uso de una red neuronal multicapa tipo Perceptrón. Para eliminar en la mayor medida posible el ruido presente en el fondo de la imagen al momento de ser capturada, se utilizan reflectores LED y un fondo uniforme contrastante con la piel para detectar de una mejor forma la seña realizada, ver Figuras 1.12 y 1.13.

Figura 1.12. Hardware adaptado por Solís y cols. (2016).

Figura 1.13. Fondo uniforme contrastante en color verde, Solís y cols. (2016).

Una vez capturada la imagen, el fondo es eliminado para establecer una región de interés, además de convertir la imagen RGB a una escala de grises, esto con el fin de reducir el costo computacional. Se calculan 42 momentos normalizados para cada imagen, estos momentos son usados para generar descriptores para cada una, los cuales son introducidos en una red neuronal multicapa tipo perceptrón para reconocer el signo.

Este sistema permite el reconocimiento del signo con una exactitud de hasta el 93 % de aciertos cuando se captura la imagen con un fondo casi completamente aislado del ruido, si esto no sucede, los resultados distan mucho de ser los esperados tal como concluye Solís y cols. (2016), la interfaz de dicho sistema implementado se muestra en la Figura 1.14.

Figura 1.14. Interfaz y gesto obtenido por Solís y cols. (2016).

Chen y cols. (2019) presenta un artículo titulado "Reconocimiento de números escritos a mano usando redes neuronales convolucionales implementados en la tarjeta NVIDIA Jetson Nano" en donde hace uso de una base de datos conocida como MNIST (*Modified National Institute of Standars and Technology*), ver Figura 1.15, la cual se compone de un conjunto de 70,000 imágenes de 28x28 pixeles, para cada imagen existe un vector de 785 elementos, donde 784 de estos elementos representan los valores de grises de la imagen y un elemento del vector que va del 0 al 9 representa la clase a la cual pertenece.

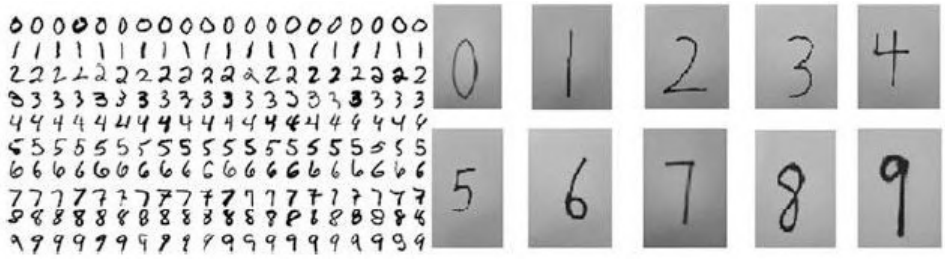


Figura 1.15. Ejemplo del conjunto de datos MNIST mostrado por Chen y cols. (2019).

En este trabajo se utiliza la librería Tensor Flow para generar el modelo de la red neuronal convolucional y se desarrolla directamente en la plataforma Jetson Nano tal como lo muestra en el siguiente diagrama de la Figura 1.16:

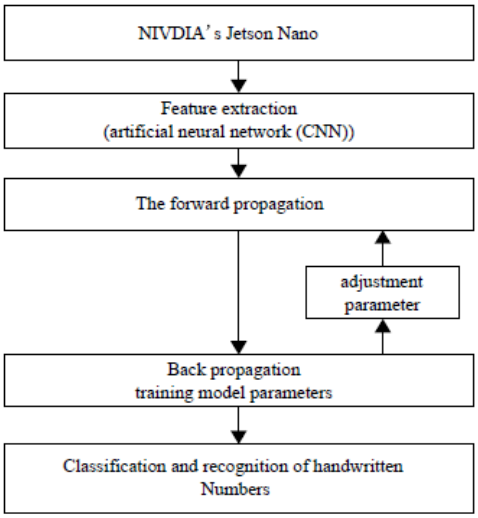


Figura 1.16. Diagrama de flujo del modelo desarrollado por Chen y cols. (2019).

Con el modelo desarrollado Chen reporta una precisión en la clasificación del 98.25 % cuando utiliza el optimizador Adam y 98.29 % cuando utiliza el optimizador RMSProp; sin embargo, recomienda el uso del optimizador Adam debido a que este presenta una convergencia más rápida.

Zhang y cols. (2019) también utilizan el sistema Jetson Nano como una plataforma para desarrollar un modelo de reconocimiento de imágenes basado en una red neuronal convolucional. La base de datos que se utiliza es *ImageNet* (ver Figura 1.17), la cual es utilizada para realizar problemas de clasificación, las imágenes son de 227 x 227 píxeles y cada una puede contener más de una sola clase, simulando entornos de la vida real.

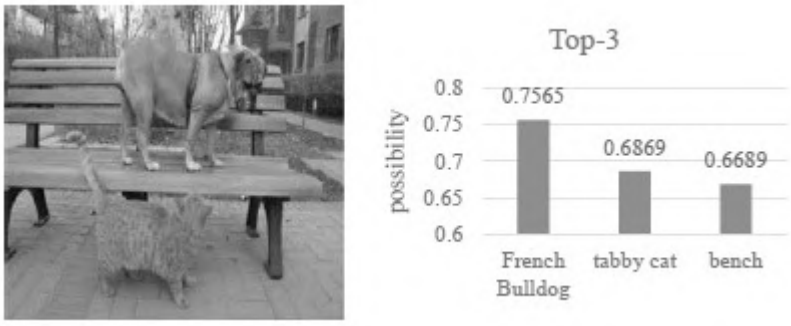


Figura 1.17. Ejemplo de imagen multiclase y respuesta obtenida por el modelo de Zhang y cols. (2019).

1.2. Planteamiento del problema

El entorno social de las personas sordas se ve reducido debido a que solamente pueden establecer canales de comunicación con individuos que conozcan su misma lengua de señas, LSM en este caso. Por lo cual, las interacciones sociales son casi nulas con personas que desconocen esta forma de comunicación, y es agravante cuando no solo se trata de convivencia social, si no cuando también se trata de comunicaciones necesarias para realizar trámites, impartir o recibir educación y demás actividades que son comunes para las personas que no presentan esta discapacidad.

Diversos autores han propuesto la realización de un traductor de lengua de señas a texto o incluso a voz. Se ha observado que comúnmente presentan tres etapas: Detección de las manos, adquisición y procesamiento de la imagen y reconocimiento del signo.

En la detección de las manos y su posición se ha observado que portar o hacer uso de hardware adicional resulta estorboso además de elevar los costos del proyecto.

En la segunda y tercera etapa también se ha visto que cualquier variación en el fondo de la imagen al momento de la captura del signo, conlleva a resultados drásticos y no deseados, como la reducción del porcentaje de efectividad de las redes neuronales empleadas.

También se ha visto que los sistemas desarrollados suelen estar en una estación fija, lo cual no permite la movilidad del mismo, por lo cual los usuarios solo pueden comunicarse en el lugar en el que se encuentre el sistema.

1.3. Justificación

Este tema es relevante debido a que como informa el Instituto Nacional de Estadística y Geografía (INEGI), de las 5, 739, 270 personas con discapacidad que viven en México, 12.1 % son sordas en algún grado, es decir 694,451 habitantes del país padecen deficiencia auditiva (INEGI (2012)).
Como señala Vidal-Espinoza y Cornejo Valderrama (2016) las personas con discapacidad auditiva representan un gasto extra en sus hogares de entre 40 % – 188 %. Su educación es sin duda el mayor reto, ya que 24 % son analfabetas y 55 % cursan nivel básico debido a la limitación para encontrar escuelas acordes a sus necesidades de aprendizaje, maestros capacitados y la orientación educativa oportuna.
De la misma forma la Encuesta Nacional de Percepción de Discapacidad en la Población Mexicana (ENPD (2012)), dan cuenta de su desigualdad social: solamente el 25 % de este grupo está ocupada, percibiendo entre 1-2 salarios mínimos mensuales.
Es verdad, como afirma Rodríguez (2005) que la comunidad sorda es silenciosa, pero también es silenciada. Por tanto, lo que se busca en este proyecto es aportar una alternativa que contribuya a romper, en la medida de lo posible este aislamiento social de manera que, mediante una cámara digital se registren los signos con los que se comunica el usuario con deficiencia auditiva en cuestión y muestre por algún medio de visualización el signo identificado.

1.4. Hipótesis

Es posible desarrollar un sistema que identifique en tiempo real signos estáticos del abecedario dactilológico de la Lengua de Señas Mexicana basado en redes neuronales convolucionales e implementado en la plataforma Jetson Nano.

1.5. Objetivos

1.5.1. Objetivo general

Desarrollar e implementar en la plataforma Jetson Nano un sistema para la identificación en tiempo real de signos estáticos correspondientes al abecedario dactilológico de la lengua de señas mexicana basado en una red neuronal convolucional.

1.5.2. Objetivos específicos

- Compilar una base de datos balanceada de signos estáticos para la LSM.
- Estandarizar la información de las imágenes de la base de datos.
- Diseñar e implementar en una computadora, una red neuronal convolucional para el reconocimiento de los signos de la base de datos.
- Implementar una red neuronal convolucional para el reconocimiento de los signos de la base de datos en el sistema Jetson Nano.
- Determinar el sistema de captura para la adquisición de las imágenes.
- Integrar el sistema para el reconocimiento en tiempo real de los signos realizados por el usuario.

1.6. Metas

- Obtención de 3 bases de datos cuyos signos sean semejantes a los signos de la lengua de señas mexicana.
- Balanceo de la cantidad de imágenes en cada clase de la base de datos para tener al menos 500 imágenes por clase.
- Estandarización del tamaño de todas las imágenes de la base de datos balanceada.
- Obtención de la representación en vectores lineales y los niveles de gris de las imágenes de la base de datos balanceada.
- Selección de una estructura (y/o la plataforma) adecuada para la RNC.
- Entrenamiento del sistema con la base de datos propia.
- Pruebas y validaciones del sistema de reconocimiento de signos implementado.
- Implementación del software y hardware necesarios para la integración del dispositivo de captura de imágenes y el sistema Jetson Nano.
- Implementación de un algoritmo para la captura del signo realizado.
- Implementación de un algoritmo para el procesamiento de las imágenes.
- Visualización mediante alguna interfaz el resultado del reconocimiento.
- Validación del dispositivo implementado.

1.7. Limitaciones

Únicamente se trabajará con los signos estáticos del abecedario dactilológico de la lengua de señas mexicana que son: A, B, C, D, E, F, G, H, I, L, M, N, O, P, R, S, T, U, V, W, Y. Se omiten por el momento las letras J, K, Ñ, Q, X, Z ya que requieren movimientos con una o ambas manos para su interpretación. El tamaño de la base de datos final será de acuerdo a las que existan en repositorios en línea y sean útiles para este proyecto. Los recursos en cuanto al CPU y GPU para generar la red neuronal serán los disponibles al momento de realizar este trabajo y según la versión del sistema Jetson Nano adquirido.

1.8. Metodología

Se propone la siguiente metodología mostrada en la Figura 1.18, esta es una adaptación a la metodología presentada por Suárez (2015).

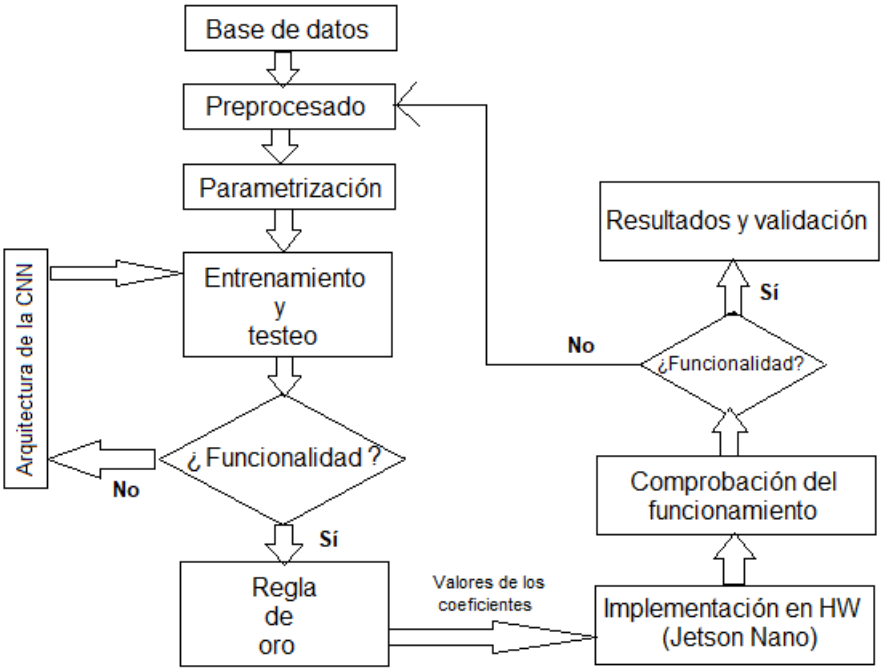


Figura 1.18. Metodología propuesta adaptada de (Suárez (2015))

Capítulo 2

Fundamentos teóricos

2.1. Inteligencia artificial

La inteligencia artificial, como menciona Lazcano (2020), se refiere al estudio, al desarrollo y a la aplicación de técnicas informáticas que permiten a las computadoras adquirir ciertas habilidades propias de la inteligencia humana.

Estas pueden ser, por mencionar algunas:

- Analizar y resolver problemas.
- Reconocer patrones.
- Aprender a realizar nuevas tareas.
- Comprender el lenguaje natural.
- Reconocer imágenes.

Si hay algo que distingue a la Inteligencia Artificial de otras ramas de la informática es precisamente una de sus áreas clave: el Aprendizaje Automático.

2.2. Aprendizaje automático

El Aprendizaje Automático (ML, del inglés, Machine Learning) es una rama de la Inteligencia Artificial que pretende aprender una función o proceso, el cual se da a partir de casos conocidos que cuentan con una solución de un problema en particular, aunque no se conozca dicha solución para todos los casos posibles como señalan Alpaydin (2020) y Burch y cols. (2001).

El Aprendizaje Automático proporciona mecanismos mediante los cuales la computadora es capaz de aprender por sí misma a resolver un problema. En estos casos, el programador se encarga de diseñar un algoritmo de aprendizaje que resulte adecuado para el problema que se pretende resolver, pero es la computadora la que resuelve el problema según Berzal (2019).

La tarea por tanto del Aprendizaje Automático consiste en detectar patrones, que al ser usados permitan reconocer procesos o realizar predicciones.

El Aprendizaje Automático ha tenido éxito en problemas de clasificación donde no se cuenta con un algoritmo específico para tareas como son: reconocimiento de voz para la transcripción de conversaciones como puede verse en Evermann y cols. (2005), clasificación de documentos como los mostrados por JingHua y cols. (2012) o detección de signos escritos a mano tal como el reportado en Chen y cols. (2019), por citar algunos.

2.2.1. Clasificación de los métodos de aprendizaje

Los tipos en los que el proceso de Aprendizaje Automático puede ser dividido son tres:

2.2.1.1. Aprendizaje supervisado

Durante este proceso los algoritmos son entrenados con diferentes ejemplos en los que la solución es conocida según Harrington (2012). En este tipo de aprendizaje se busca inducir modelos capaces de predecir el valor de ciertas variables dependientes a partir de variables independientes como describe F. A. González (2015).
Un ejemplo de problema de aprendizaje supervisado es el problema de clasificación, en el cual la variable dependiente o entrada corresponde a un atributo o salida que indica a qué clase (por ejemplo, enfermo o no enfermo en el caso de un problema de diagnóstico médico) pertenece una muestra particular.
En caso de que las salidas se asocien a valores continuos, se trata de un proceso conocido como regresión.
En las Figuras 2.1 y 2.2 se pueden ver ejemplos del uso de técnicas de aprendizaje supervisado para clasificación y regresión.

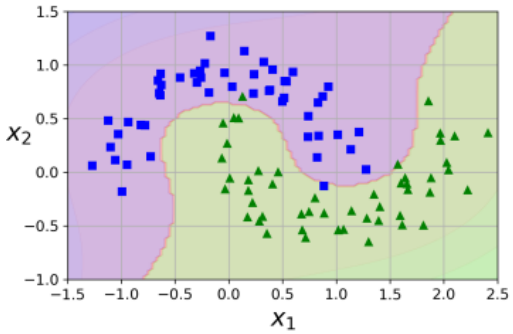


Figura 2.1. Aprendizaje supervisado: Clasificación, según ejemplifica Géron (2019).

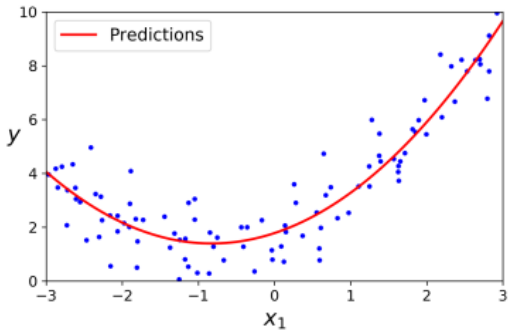


Figura 2.2. Aprendizaje supervisado: Regresión, según ejemplifica Géron (2019).

Algunos enfoques importantes del aprendizaje supervisado son:

- Redes Neuronales Artificiales.
- Máquinas de soporte vectorial.
- Árboles de decisión.

2.2.1.2. Aprendizaje no supervisado

En los modelos de aprendizaje no supervisado no hay una distinción entre variables dependientes e independientes, las salidas no son conocidas, por lo tanto, el objetivo de estos algoritmos es encontrar regularidades y relaciones entre los datos de entrada que expliquen la estructura de los datos. Al no haber datos de salida no existe una señal que retroalimente el modelo o evalúe una solución potencial.

Un caso de este modelo de aprendizaje es el análisis de agrupamientos (clusters). En este tipo de aprendizaje se encuentran algoritmos como:

- K-means.
- Agrupamiento jerárquico.
- Reducción de la dimensionalidad.

2.2.1.3. Aprendizaje semi-supervisado

Seif (2018) menciona que este tipo de aprendizaje utiliza técnicas del aprendizaje supervisado y aprendizaje no supervisado, se emplea usualmente cuando se tiene gran cantidad de datos de entrada y pocos datos etiquetados a la salida. La importancia del aprendizaje semi-supervisado es que puede realizar tareas de aprendizaje supervisado o no supervisado a partir de pocos datos etiquetados.
Los ejemplos más claros del uso de este tipo de aprendizaje son en texto e imágenes, en donde existe una gran cantidad de texto e imágenes no etiquetadas afirma Morales y cols. (2009).

2.2.2. Redes Neuronales Artificiales

Las redes neuronales artificiales forman parte del aprendizaje supervisado y proporcionan uno de los mecanismos mediante los que se puede conseguir que la computadora simule el proceso de aprendizaje humano y, a diferencia de los modelos que se representan de forma explícita mediante una función y reglas lógicas, las redes neuronales se representan de forma implícita, a través de los pesos con los que se modelan las conexiones entre neuronas.

Estas redes neuronales se inspiran en el funcionamiento del cerebro humano, el cual es un sistema altamente complejo, no lineal y paralelo. Como mencionan Izaurieta y Saavedra (2000) las redes neuronales se caracterizan principalmente por:

- Tener una inclinación natural a adquirir el conocimiento a través de la experiencia, el cual es almacenado, al igual que en el cerebro, en el peso relativo de las conexiones interneuronales.
- Tener una altísima plasticidad y gran adaptabilidad, siendo capaces de cambiar dinámicamente junto con el medio.
- Poseer un alto nivel de tolerancia a fallas, es decir, pueden sufrir un daño considerable y continuar teniendo un buen comportamiento, al igual como ocurre en los sistemas biológicos.

- Tener un comportamiento no-lineal, lo que les permite procesar información procedente de otros sistemas no-lineales.

Las redes neuronales artificiales están basadas en el funcionamiento de las redes de neuronas biológicas. Estas últimas están compuestas de dendritas, soma y axón: las dendritas se encargan de captar los impulsos nerviosos que emiten otras neuronas. Estos impulsos, se integran en el soma y si su valor sobrepasa un umbral de activación, se genera un impulso (potencial de acción) que se transmite a través del axón que emite un impulso nervioso hacia las neuronas contiguas. Por su parte, en el caso de las neuronas artificiales, la suma de las entradas multiplicadas por sus pesos asociados determina el impulso que recibe la neurona. Este valor, se procesa en el interior de la neurona mediante una función de activación que devuelve un valor que se envía como salida de la neurona.

En la Figura 2.3 se pueden observar las partes generales de una neurona biológica:

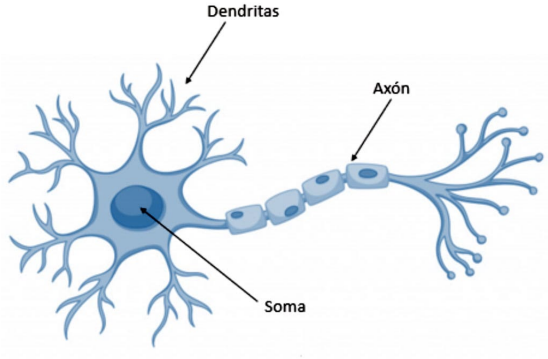


Figura 2.3. Partes generales de una neurona biológica como se muestra en Garcia (2019).

La Figura 2.4 muestra un modelo simplificado de neurona que sirve como base para desarrollar las redes neuronales artificiales:

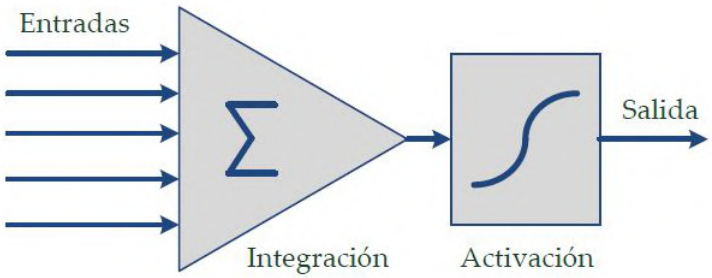


Figura 2.4. Modelo estándar de neurona artificial con función de activación no lineal como describe Berzal (2019).

Generalmente se requieren funciones de activación no lineales debido a que una red formada de neuronas lineales sólo podrá representar funciones lineales, sin embargo, la mayor parte de los problemas reales incorporan algún tipo de no linealidad e interesa que la red neuronal artificial generada sea capaz de representar esas no linealidades.

Del mismo modo que nuestro cerebro está compuesto por neuronas interconectadas, una red neuronal artificial está formada por neuronas artificiales conectadas entre sí y agrupadas en diferentes niveles que denominamos capas. Una capa es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son la entrada de una capa posterior.

2.2.2.1. Redes Neuronales Artificiales Unicapa

Una red unicapa (ver Figura 2.5) es la forma más simple de red neuronal, la capa es estrictamente del tipo hacia adelante (*feed forward*). Este tipo de red sólo puede resolver problemas linealmente separables (Figura 2.6). La única capa que se considera se refiere a la capa de salida de las neuronas, no se cuenta la capa de entrada debido a que no se realiza ningún cálculo en esta.

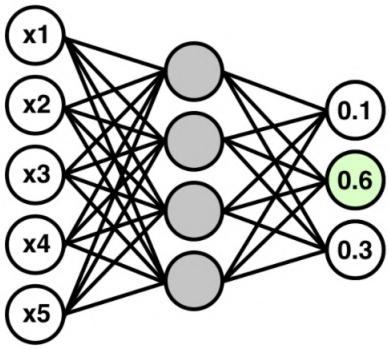


Figura 2.5. Red Neuronal Unicapa, Wong (2017).

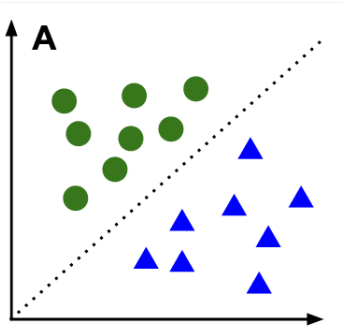


Figura 2.6. Ejemplo de problema linealmente separable según se muestra en Palomera (2015).

2.2.2.2. Redes Neuronales Artificiales Multicapa

Las redes multicapa son una segunda clase de redes neuronales del tipo hacia adelante. Este tipo de redes neuronales según Haykin y cols. (2009) se caracterizan por tener una o más capas ocultas que realizan algún cómputo que influye en la salida, el término capas ocultas se refiere al hecho de que estas capas no son visibles directamente desde la capa de entrada o la capa de salida.

Usualmente, las capas están ordenadas por el orden en que reciben la señal desde la entrada hasta la salida y están unidas en ese orden (Figura 2.7). Ese tipo de conexiones se denominan conexiones *feedforward* o hacia adelante sostiene Ballesteros (2015).

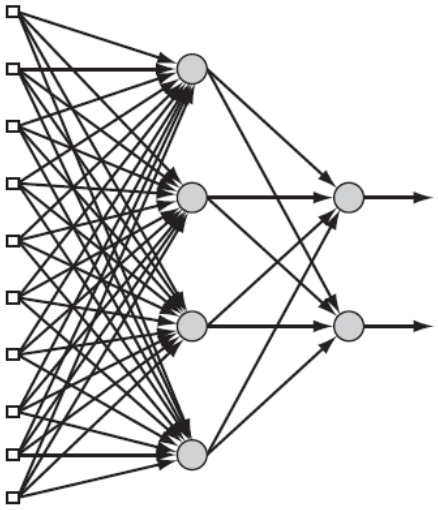


Figura 2.7. Red neuronal multicapa, de izquierda a derecha, capa de entrada, capa oculta y capa de salida como describe Haykin y cols. (2009).

Este tipo de configuraciones pueden ser utilizadas para resolver problemas los cuales no son linealmente separables como reconocimiento de patrones, clasificación no lineal (ver Figura 2.8), predicciones y aproximaciones.

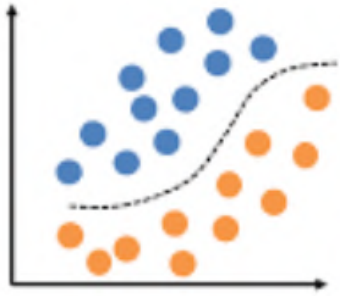


Figura 2.8. Ejemplo de problema linealmente no separable según se muestra en Palomera (2015).

2.2.2.3. Redes Neuronales Artificiales Profundas

Una red neuronal profunda es una red neuronal artificial con varias capas ocultas entre las capas de entrada y salida capaces de modelar relaciones no lineales complejas según menciona Nielsen (2015). Estas arquitecturas permiten aprender representaciones de los datos con diferentes niveles de abstracción.

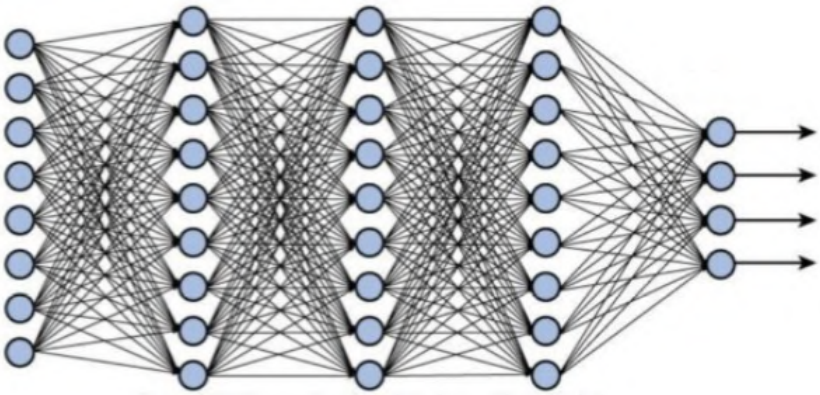


Figura 2.9. Red Neuronal Profunda con múltiples capas como describe Nielsen (2015).

En la Figura 2.9 se puede observar una red neuronal artificial que cuenta con una capa de entrada, tres capas ocultas y una capa de salida. Esta es sólo una representación básica ya que en el aprendizaje profundo el número de capas ocultas, en su mayoría no lineales puede ser aún más grande.

2.2.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son una clase de redes neuronales artificiales profundas que son comúnmente aplicadas a tareas de procesamiento de imágenes tales como detección e identificación de objetos.

La arquitectura de este tipo de redes se divide en dos partes, una **etapa convolucional** y una **etapa de clasificación** .

El objetivo de la etapa convolucional es extraer características propias de cada imagen, esta extracción de características se logra al aplicar diferentes filtros o *kernels* a cada imagen a través de la así llamada operación de convolución. Este proceso suele estar acompañado de algún método de submuestreo para reducir el costo computacional del aprendizaje de la RNC debido a la gran cantidad de características obtenidas durante las fases anteriores. Así mismo, se hace uso de la técnica de aplanado de datos antes de introducir los datos a la etapa de clasificación

La etapa de clasificación consiste en algún tipo de red neuronal multicapa

convencional en donde el total de neuronas de la capa de salida coincide con el número de clases.

2.2.3.1. Etapa convolucional de las RNC

Retomando de Berzal (2019) la convolución es una operación matemática que se realiza sobre dos funciones para producir una tercera que se suele interpretar como una versión modificada (filtrada) de una de las funciones originales.

La convolución de las matrices I de dimensiones n, m y k de dimensiones a, b que se suele denotar mediante un asterisco $*$ se representa matemáticamente de la siguiente manera:

$$[I * k](i, j) = S(i, j) = \sum_{n} \sum_{m} I(n, m) \cdot k(i - n, j - m) \tag{2.1}$$

Donde S es la matriz de salida, I la matriz de entrada y k es el kernel de convolución. Básicamente lo que representa la ecuación 2.1 es la inversión de un kernel de convolución k y su desplazamiento a través de los elementos de la matriz de entrada I a la vez que se realiza el producto punto de los elementos que se solapan. En la Figura 2.10 puede apreciarse un ejemplo de dicho proceso en donde **I** que es la matriz de entrada representa una imagen a procesar, **k** representa un kernel de tamaño 3 x 3 y **S** la matriz e imagen de salida, en esta matriz de salida es calculado el elemento $1, 1$ o píxel $1, 1$.

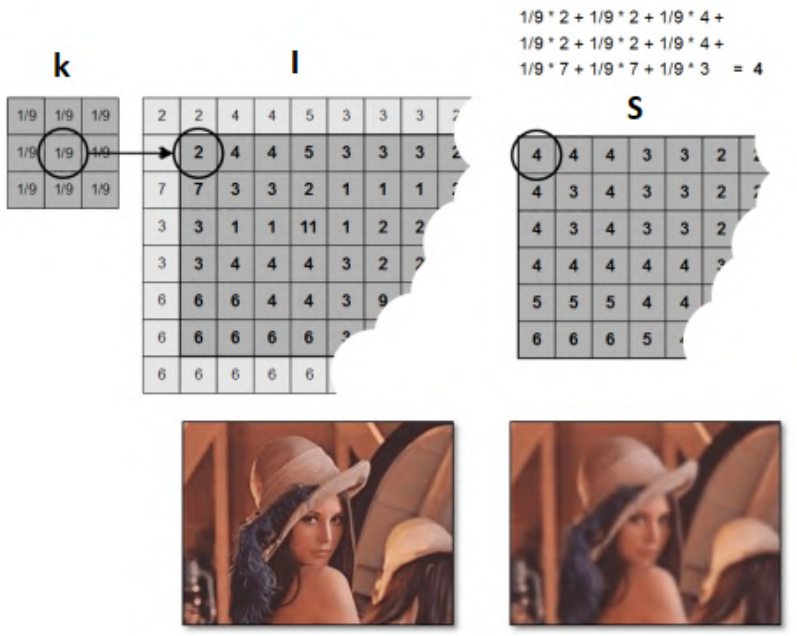


Figura 2.10. Obtención del elemento $1,1$ de la matriz de salida S al aplicarse el kernel de convolución k al elemento $1,1$ de la matriz de entrada I, obtenido de Quirós (2019)

Esta nueva matriz e imagen de salida S es conocida también como versión filtrada de la imagen o mapa de características. Los filtros convolucionales pueden tanto acentuar como atenuar características específicas en las imágenes de entrada, como curvas, bordes o colores. Diferentes filtros convolucionales extraen diferentes características y es la combinación de los mapas de características resultantes lo que potencia las predicciones de las redes neuronales convolucionales. En la figura 2.11 se observan en la parte de la derecha 16 mapas de características extraídos para la imagen de la izquierda.

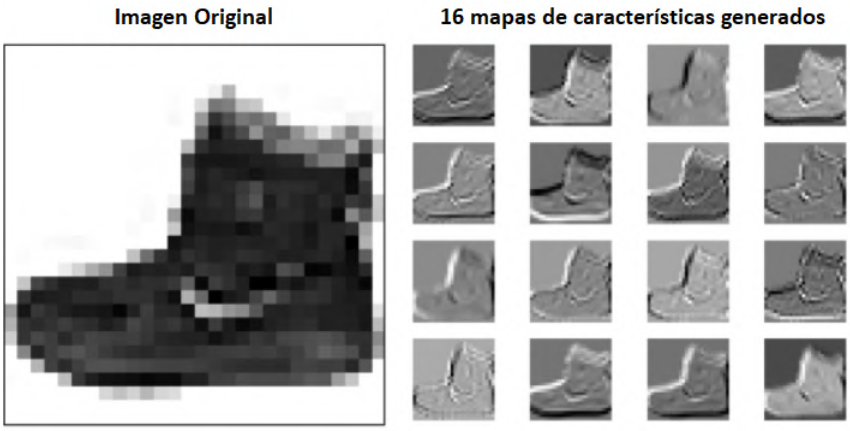


Figura 2.11. Ejemplo de 16 mapas de características generados a partir de la imagen de la izquierda.

En la práctica, los filtros convolucionales son un conjunto de pesos que se aplican a los valores de píxel en cada imagen de entrada. Estos pesos se aprenden y refinan durante la fase de entrenamiento.

Capa de Pooling

La idea principal detrás de la capa de Pooling es "acumular" características de los mapas generados por el filtro de convolución sobre la imagen. Formalmente su función es reducir progresivamente el tamaño espacial de la representación para reducir la cantidad de parámetros y la complejidad computacional de la red, así como prevenir el sobreajuste de la red neuronal artificial enfatiza Granda Cárdenas (2020).

La forma más común de Pooling es "Max-Pooling", el cual aplica un filtro "Max" a una subregión de la imagen, donde obtiene el pixel con el máximo valor de dicha región, tal como se muestra en la Figura 2.12.

Figura 2.12. Max-Pooling aplicado a una imagen dividida en subregiones como muestra Granda Cárdenas (2020)

Aplanado de los datos

El aplanado o "flattening" se realiza para que los datos de los mapas de características una vez que hayan pasado por una reducción de su dimensionalidad con la capa de Max-Pooling se les acomode en un vector unidimensional, esto con el fin de prepararlos para ser los datos de la capa de entrada de la futura red neuronal artificial. Un ejemplo de este procedimiento se muestra en la Figura 2.13.

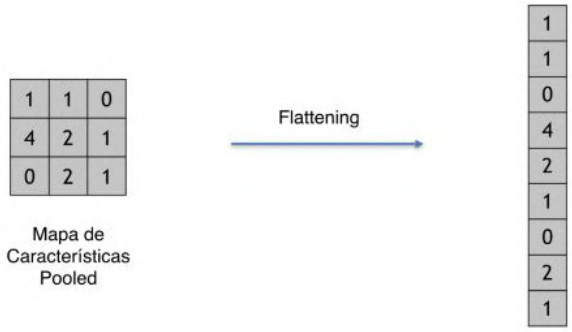


Figura 2.13. Ejemplo de "aplanado" o "flattening" de una imagen como menciona Gomilla (2019)

2.2.3.2. Etapa de clasificación de las RNC

Por último, el vector de datos unidimensional que se obtuvo mediante el método de "aplanado" se introduce a la capa de entrada de una red neuronal artificial multicapa para realizar la clasificación de la imagen de entrada. En este caso la cantidad de neuronas de la capa final de la red neuronal artificial corresponderá con las clases a identificar.

La Figura 2.14 que propone Gomilla (2019) resume los pasos en los que una imagen es tratada en una red neuronal convolucional:

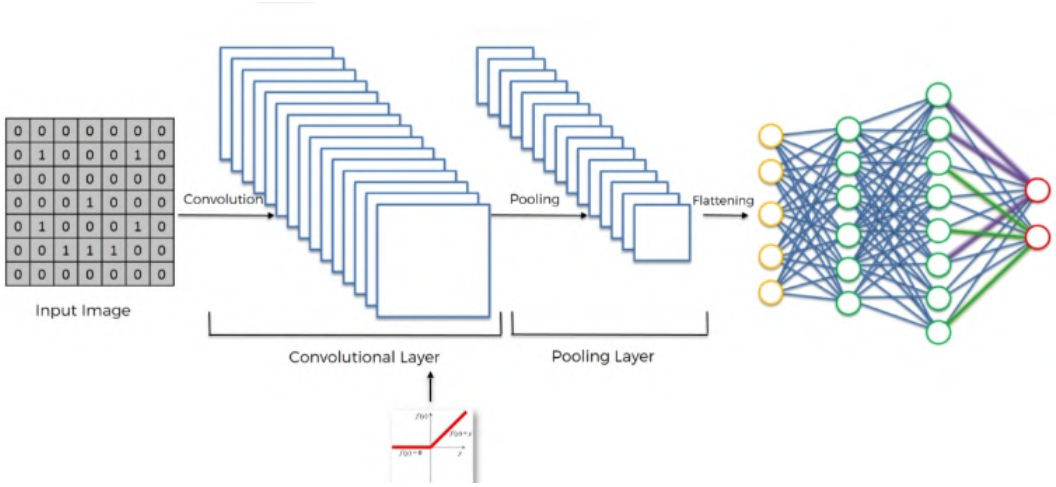


Figura 2.14. Resumen de pasos en una Red Neuronal Convolucional propuestos por Gomilla (2019)

2.3. Descripción del hardware utilizado

2.3.1. Tarjeta de desarrollo Jetson Nano

La tarjeta que se muestra en la Figura 2.15 y se describe en NVIDIA (2019), es una computadora potente y pequeña que permite ejecutar múltiples redes neuronales en paralelo para aplicaciones en Inteligencia Artificial, como clasificación de imágenes, detección de objetos, segmentación y procesamiento de voz.

Figura 2.15. Kit de Desarrollo NVIDIA Jetson Nano mostrado en NVIDIA (2019)

La tarjeta de desarrollo NVIDIA Jetson Nano contiene un procesador Quad-core ARM A57 @ 1.43 GHz, una memoria RAM de 4 GB 64-bit tipo LPDDR4, maneja una unidad gráfica (GPU) 128-core Maxwell, además de contar con conexiones GPIO, I2C, I2S, SPI, UART.
Para su alimentación se requieren 5 Volts y es capaz de suministrar hasta 2 Amperes. Así mismo maneja dos modos de potencia, a 5 Watts y 10 Watts.

2.3.2. Cámara de video Raspberry Pi

La cámara de video utilizada en este proyecto es la **Raspberry Camera v2**, este dispositivo es una cámara de 8 mega pixeles de resolución, además la comunicación con la tarjeta Jetson Nano es por medio de la interfaz serial CSI. Este periférico puede observarse en la figura 2.16.

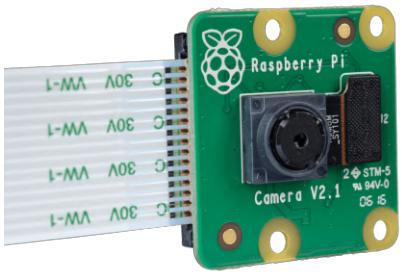


Figura 2.16. Cámara de video Raspberry V2

2.4. Descripción del software utilizado

2.4.1. OpenCV

OpenCV (*Open Source Computer Vision*) es una biblioteca gratuita orientada a tareas de visión artificial y que permite la manipulación de imágenes y videos para una gran variedad de tareas, desde únicamente desplegar cuadros (*frames*) de video de una cámara de video como la *Raspberry Camera v2*, hasta establecer el flujo de trabajo de un sistema de reconocimiento de objetos en tiempo real, Howse (2020).

2.4.2. TensorFlow y Keras

TensorFlow es una biblioteca de código abierto usada para entrenar y desarrollar modelos de aprendizaje automático (*machine learning*). Más específicamente, es una biblioteca matemática simbólica según describe Wolfe (2021).

Keras por su parte es una biblioteca especializada en el desarrollo de modelos de aprendizaje profundo (*deep learning*) como lo son las redes neuronales artificiales. Esta biblioteca está creada para ejecutarse sobre otras como lo es TensorFlow.

2.5. Dactilología de la Lengua de Señas Mexicana

El proceso de aprendizaje de la Lengua de Señas Mexicana inicia con la dactilología como menciona G. C. González y cols. (2020), después se enseñan campos semánticos, narración de cuentos y conversaciones.

La dactilología, según describe Vilches Vilela (2005), es la representación manual de cada una de las letras que componen el alfabeto. A través de ella se puede transmitir a la persona con deficiencia auditiva cualquier palabra que se desee comunicar, por complicada que esta sea.
Este deletreo manual es usado más adelante como base para construir campos semánticos, los cuales permiten una mayor fluidez y naturalidad en la comunicación.

La lengua de señas mexicana (LSM) cuenta con un abecedario dactilológico de 27 posiciones diferentes que representan cada una de las letras del alfabeto como se muestra en la Figura 2.17.

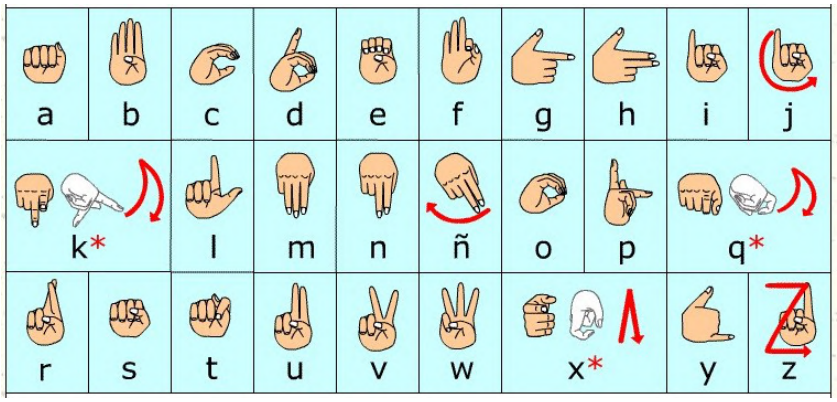


Figura 2.17. Abecedario dactilológico de la LSM, obtenido de Ramos (2018)

Capítulo 3

Desarrollo Metodológico

En esta sección se muestra el desarrollo del proyecto basado en la metodología propuesta y adaptada de Suárez (2015). El problema a analizar pertenece a un tipo de aprendizaje supervisado, dado que se conoce de antemano a que clase pertenece cada elemento. El objetivo en este capítulo ha sido seleccionar un modelo de red neuronal convolucional a partir de dos distintas propuestas de modelos; uno creado como un modelo base y otro tomando la configuración de una red conocida como AlexNet.

3.1. Integración de la base de datos

La base de datos utilizada está constituida por 21 clases, con 500 imágenes por clase, dando un total de 10,500 imágenes. Las 21 clases corresponden a las letras : A, B, C, D, E, F, G, H, I, L, M, N, O, P, R, S, T, U, V, W, Y. Estas clases fueron seleccionadas debido a que sus correspondientes signos dactitológicos en la lengua de señas mexicana se expresan de forma estática, sin ningún tipo de movimiento como afirman Mendoza y Jackson-Maldonado (2020) y Hernández (2010).

De las 10,500 imágenes, 6,493 se obtuvieron de tres bases de datos con signos semejantes a los del abecedario de la LSM en el repositorio público *Kaggle*. Estas fueron: Lengua de Señas Americana (ASL) cuyas imágenes son de dimensiones 200 x 200 píxeles, Lengua de Señas Colombiana (CSL) con elementos de dimensiones de 4,608 x 2,592 píxeles y Lengua de Señas Bengalí (BSL) cuyas fotografías son de 224 x 244 píxeles, todas las anteriores imágenes se encontraban en formato jpg. Las 4,007 imágenes restantes se adquirieron con una cámara web en formato jpg y resolución de

176 x 144 px, ver Tabla 3.1.

Tabla 3.1. Bases de datos y su aporte a la LSM.

Signo	A	B	C	D	E	F	G	H	I	L	M	N	O	P	R	S	T	U	V	W	Y
ASL	0	500	500	500	0	0	0	0	500	500	0	0	500	0	500	500	0	500	500	500	0
CSL	304	0	0	0	229	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BSL	165	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	295	0	0	0	0
Por Capturar	31	0	0	0	271	500	500	500	0	0	500	500	0	500	0	0	205	0	0	0	500

3.2. Estandarización de las imágenes

Las imágenes de la base de datos se estandarizaron a un tamaño de 72 píxeles de alto x 88 píxeles de ancho, es decir se tienen 6,336 píxeles o características por imagen, cada una de estas se representó en niveles de gris y con una profundidad de pixel de 8 bits. Por tanto, en esta primera etapa de desarrollo, el algoritmo encargado de la etapa de integración de la base de datos y preprocesamiento de los mismos realiza lo siguiente:

1. Recorre 21 carpetas de imágenes (una por cada clase de letra) y genera una nueva carpeta con las imágenes redimensionadas a 72 x 88 píxeles cada una.
2. Las imágenes con tres capas de color son promediadas usando el estándar ITU-R 601-2 para así generar una nueva imagen en escala de grises.
3. Se genera un vector el cual contiene las etiquetas de cada imagen y clase correspondiente.
4. Se crea un archivo de texto el cual contiene una matriz donde 10,500 filas representan el total de imágenes de la base de datos, 6,336 columnas contienen cada valor de píxel y una última columna con sus correspondientes etiquetas de clase.

En la figura 3.1 se muestra una imagen de cada una de las 21 clases después de haberles sido aplicado el procedimiento antes mencionado.

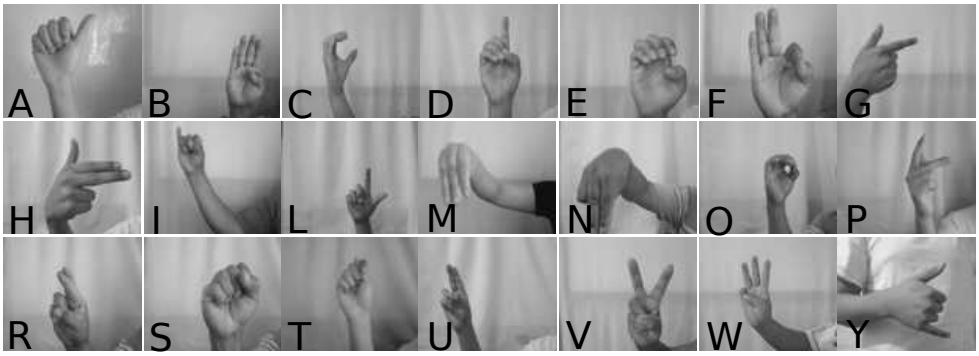


Figura 3.1. Ejemplo de elementos del abecedario de la base de datos generada.

3.2.1. Preparación de los conjunto de entrenamiento, pruebas y validación

El conjunto de pruebas consta del 20 % del total de las imágenes de la base de datos, es decir 2,100 elementos, el 80 % restante se divide en 85 % para entrenamiento y 15 % para validación o dicho de otra forma, 7,140 imágenes se usaron para la fase de entrenamiento y 1,260 para validación. La Figura 3.2 exhibe dicha división de la base de datos.

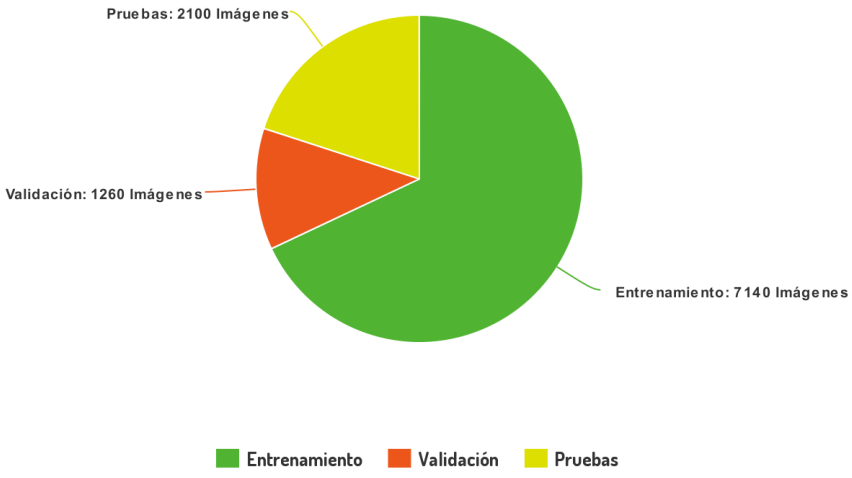


Figura 3.2. Distribución de las imágenes de la base de datos en los conjuntos de entrenamiento, pruebas y validación.

Por otro lado el vector de clases contiene 21 elementos del [0,20] los cuales representan las 21 clases de letras, comúnmente los modelos de redes neuronales

requieren que cada elemento de entrada tenga un valor de '1' en su correspondiente clase y '0' en las demás, para ello se implementa una función conocida como codificación *one hot* con la cual se obtuvo una matriz de 21 columnas y 10,500 filas (el código se muestra en el Anexo A).

3.3. Desarrollo de los modelos de red neuronal convolucional

Para realizar el reconocimiento de los 21 signos estáticos del abecedario dactilológico de la LSM, se propuso en una primera etapa entrenar un modelo de RNC basado en los hiperparámetros originales de la arquitectura desarrollada por Krizhevsky y cols. (2012), mejor conocida como AlexNet. AlexNet fué una de las primeras arquitecturas de RNC que demostraron una alta efectividad en las tareas de clasificación de imágenes, se considera que es una de las RNC más influyentes y mayormente citadas en visión artificial. Originalmente fué propuesta para la clasificación de imágenes con tres capas de color (RGB), pero para los propósitos de este trabajo la primera capa de entrada se ha configurado para aceptar una única capa de escala de grises así como la capa de salida del clasificador se ha adaptado para tener 21 neuronas a la salida, esto es, el total de clases para la clasificación de la LSM. Las características de dicha arquitectura se muestran a continuación.

3.3.1. Arquitectura AlexNet

La arquitectura AlexNet implementada es como se describe a continuación. De forma general, la etapa de convolución consiste en 5 capas de convolución, 3 capas *max-pooling* y 5 capas de normalización por lotes, así como la función de activación $ReLU$ presente al final de cada capa convolucional. En la etapa de clasificación se construyeron 2 capas densas y una capa de salida la cual se cambió de originalmente 1,000 neuronas a 21 neuronas para coincidir con el número de clases de este proyecto.

- Primera capa oculta: Con un tamaño de kernel de 11 x 11 y un paso *(stride)* de 4 píxeles se generaron 96 mapas de características con un tamaño de 18 x 22. Enseguida una capa de normalización por lotes es aplicada. Más adelante, una

ventana de pooling de 2 x 2 es usada para la operación de submuestreo, por lo que la dimensión de los mapas de características se reducen a un tamaño de 9 x 11.

- Segunda capa oculta: Kernels de tamaño 5 x 5 y un paso de 1 se usaron para la operación de convolución y 256 mapas de características fueron generados con un tamaño de 9 x 11. Enseguida una capa de normalización por lotes es aplicada. Más adelante, una ventana de pooling de 2 x 2 es usada para la operación de submuestreo, por lo que los 256 mapas de características se reducen a un tamaño de 5 x 6.

- Tercera capa oculta: Kernels de tamaño 3 x 3 y un paso de 1 se usaron para la operación de convolución y 384 mapas de características son generados con un tamaño de 5 x 6. Enseguida una capa de normalización por lotes es aplicada.

- Cuarta capa oculta: De la misma manera, kernels de tamaño 3 x 3 y un paso de 1 se usaron para la operación de convolución y 384 mapas de características son generados con un tamaño de 5 x 6. Enseguida una capa de normalización por lotes es aplicada.

- Quinta capa oculta: Con kernels de tamaño 3 x 3 y un paso de 1 se obtuvieron 256 mapas de características con un tamaño de 5 x 6. Enseguida una capa de normalización por lotes es aplicada. Más adelante, una ventana de *pooling* de 3 x 3 es usada para la operación de submuestreo, por lo que los 256 mapas de características se reducen a un tamaño de 3 x 3. Se aplica una capa de aplanado la cual genera un vector unidimensional de 2,304 elementos.

Más adelante, estos elementos son la entrada de un clasificador con dos capas ocultas de 4,096 neuronas cada una. Debido a que se tienen 21 clases distintas de signos, la capa de salida contiene 21 neuronas a las cuales es aplicada la función de activación *softmax*. El tamaño de bloques propuesto originalmente en esta arquitectura fue de 128, se utiliza la entropía cruzada como función de costo y el optimizador Adam se elige con una taza de aprendizaje por default de 0.001. El total de parámetros entrenables para esta arquitectura son $34,085,759$.

La Figura 3.3 resume de una forma gráfica las capas antes mencionadas correspondientes a las etapas de convolución y clasificación, en donde cada recuadro representa algún procesamiento realizado a la imagen de entrada ya sea convolución,

max-pooling, aplanamiento o clasificación. Es fácil ver cómo las dimensiones de la imagen se reducen en ancho y alto, pero la profundidad aumenta cada vez que se generan nuevos mapas de características.

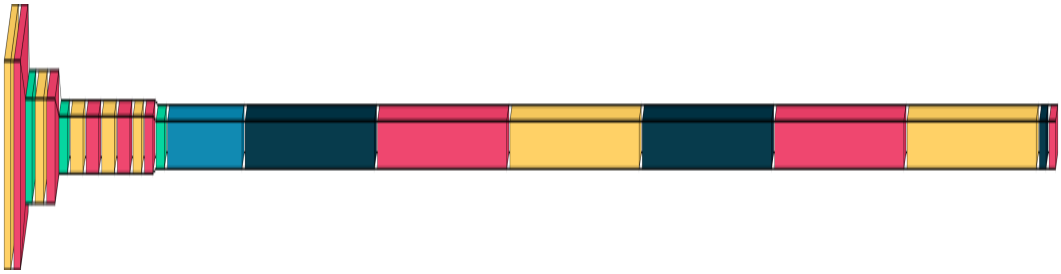


Figura 3.3. Visualización gráfica de la arquitectura construida basada en el modelo original AlexNet.

3.3.2. Arquitectura de RNC propuesta

El modelo de RNC propuesto en su etapa convolucional contiene tres capas de convolución en las cuales se aplican kernels de tamaño 4 x 4 y pasos de 1 así como la función de activación $ReLU$ a cada capa, por su parte la etapa de clasificación requiere un vector unidimensional por lo que se realiza el aplanamiento de los mapas de características. Así mismo dos capas densas son propuestas, la primera con 256 neuronas y la segunda con 21 neuronas la cual corresponde con la capa de salida en la cual se utliza la función de activación *softmax*. Dicho modelo de forma más específica es implementado como se muestra:

- Primera capa oculta: Con los kernels y pasos antes mencionados 32 mapas de características son generados con un tamaño de 72 x 88. Más adelante, una ventana de pooling de 2 x 2 es usada para la operación de submuestreo, por lo que los 32 mapas de características se reducen a un tamaño de 36 x 44.

- Segunda capa oculta: En esta etapa se obtienen 64 mapas de características de dimensiones 33 x 41 (a diferencia de la capa anterior no se usa una técnica de *padding* para conservar las mismas dimensiones de entrada).

- Tercera capa oculta: Nuevamente con kernels de tamaño 4 x 4 se generaron 64 mapas de características con dimensiones 30 x 38. Finalmente, se aplica una capa de aplanado la cual genera un vector unidimensional de 72,960 elementos.

Más adelante, estos elementos son la entrada de un clasificador con una capa oculta de 256 neuronas y, debido a que se tienen 21 clases distintas de signos, la capa de salida contiene 21 neuronas conectadas a una función de activación de tipo *softmax*. Se elige un tamaño de bloques de 128, se utiliza la entropía cruzada como función de costo y el optimizador Adam se elige con una taza de aprendizaje por default de 0.001. Esta arquitectura contiene un total de parámetros entrenables de $18,782,389$.

De la misma manera que con la arquitectura Alexnet, la Figura 3.4 resume de una forma gráfica las capas antes mencionadas. En este gráfico podemos apreciar las capas convolucionales en amarillo, la capa de max-pooling en rojo, el vector resultante del aplanamiento en azul cielo, las capas correspondientes de clasificación en azul marino seguidas de las funciones de activación de nuevo en color rojo.

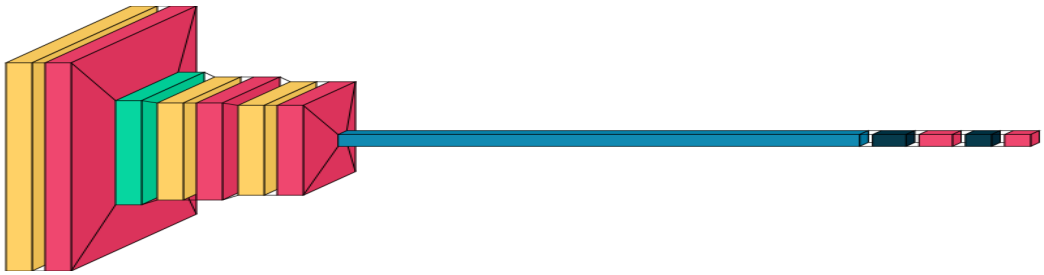


Figura 3.4. Visualización gráfica de la arquitectura construida basada en el modelo propuesto.

3.3.3. Optimización de la RNC propuesta

A continuación se muestra el proceso de optimización de los valores de la taza de aprendizaje del optimizador Adam, número de kernels de convolución por capa y número de capas ocultas de convolución.

Es posible agregar aún más hiperparámetros para su optimización, sin embargo, cada hiperparámetro representa un grado de complejidad mayor debido a que aumenta el número posible de combinaciones y aún más cuando cada hiperparámetro varía en un rango de valores.

Para realizar dicho proceso se hace uso de la biblioteca **keras-tuner** con la cual se entrenaron 10 diferentes modelos de RNC mismos que tienen como base la arquitectura

de RNC propuesta, pero el número de capas de convolución así como la cantidad de kernels y la taza de aprendizaje varía entre una y otra de forma aleatoria.

El primer paso consistió en redefinir la RNC propuesta y establecer un rango de valores de filtros *(kernels)* de convolución con los cuáles fueron entrenados los 10 modelos distintos. Una porción de esta instrucción se muestra enseguida:

```
model.add(Conv2D(hp.Int('n_kernels_conv_0',min_value=16,
                        max_value=128, step=16) ...
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Como puede observarse, el parámetro variable es el número de kernels y varía entre valores de 16 y 128 con un paso de 16, esto es, para evitar probar todo el rango de valores enteros entre 16 y 128, el uso de cierto número de kernels en alguna RNC no prohibe el uso de este valor nuevamente en posteriores redes. Esta primera capa de convoulución es la capa 0 ya que permanece fija, el número de capas de convolución a calcular fueron las siguientes que, como se muestra más adelante, varían de 1 a 3 capas, denominadas capas ocultas de convolución. Nótese también que la capa de *MaxPooling* no es afectada.

Por tanto, el proceso de selección del número de capas ocultas de convolución adecuado se realiza con el siguiente ciclo en donde son probadas diferentes cantidades de filtros para cada RNC, de 1 a 3 posibles capas:

```
for i in range(hp.Int('n_capas_conv',1,3)):
    model.add(Conv2D(hp.Int(f'n_kernels_conv_{i}',
    min_value=16, max_value=128, step=16)...)
```

Por último, la taza de aprendizaje es establecida para tomar tres distintos valores: 0,01 , 0,001 ó 0,0001.

En la Tabla 3.2 se muestra una descripción resumida de la arquitectura de los diez modelos que se entrenaron, recordar que se ha tomado como base la RNC propuesta donde:

- **n_kernels_conv_0**: Cantidad de filtros aplicados a la capa fija de convolución (capa 0).

- **n_capas_conv**: Número de capas ocultas de convolución aplicadas al modelo.
- **n_kernels_conv_i**: Cantidad de filtros aplicados a la capa n-esima de convolución.
- **learning_rate**: Valor de la taza de aprendizaje.

Tabla 3.2. Descripción de los modelos basados en la RNC propuesta.

Modelo número	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
n_kernels_conv0	64	96	80	96	32	16	80	16	80	96
n_capas_conv	2	2	2	2	3	1	2	2	3	1
n_kernels_conv1	112	48	80	32	80	16	96	96	80	80
n_kernels_conv2	112	96	80	80	64	n/a	16	64	64	n/a
n_kernels_conv3	n/a	n/a	n/a	n/a	112	n/a	n/a	n/a	80	n/a
learning_rate	0.001	0.0001	0.01	0.01	0.01	0.01	0.001	0.001	0.01	0.001

En general, cada ejecución de entrenamiento de las redes neuronales convolucionales puede conllevar a una ligera variación en los resultados si la configuración inicial de los pesos de la red se deja de forma aleatoria. Por ello es necesario seleccionar una *semilla* aleatoria constante para que al repetirse estos experimentos la única variación en cuanto al resultado sea producida por la configuración de los hiperparámetros. La semilla escogida para los modelos de RNC es la que se muestra a continuación:

```
import numpy as np
np.random.seed(1000)
```

La capa de aplanado de los datos como se ha mencionado previamente, permite el ingreso de los datos a las capas densas de clasificación, sin embargo este arreglo unidimensional también puede lograrse haciendo un redimensionamiento (*reshape*) de los datos. En el proceso de entrenamiento y pruebas de una RNC en una computadora personal elegir cualquier instrucción resulta indistinto, sin embargo sólo la segunda instrucción es válida al intentar generar una RNC compatible con la tarjeta de desarrollo Jetson Nano. Ambas instrucciones se muestran a continuación:

```
# Instruccion que genera el aplanado de los datos:
model.add(Flatten())
# Instruccion similar, pero compatible con
# la tarjeta de desarrollo Jetson Nano:
# model.add(Reshape((1536,)))
```

El código completo se puede consultar en los anexos B y C.

3.4. Generación de un modelo exportable para la tarjeta de desarrollo Jetson Nano

Usando la librería Keras, misma que se usó para desarrollar el modelo de RNC, se almacenan los pesos de la red en un archivo con extensión **h5**, dicha instrucción se muestra a continuación:

```
model.save('modelo_seleccionado.h5')
```

La extensión corresponde a un formato de datos jerárquicos (HDF) por sus siglas en inglés, este tipo de archivo permite almacenar arreglos multidimensionales de datos. Sin embargo, el mismo no puede ser leído por la tarjeta Jetson Nano, tiene que ser transformado en un formato tipo **pb**, el cual contiene un grafo 'congelado', el cual es un tipo de archivo que, además de contener los pesos del modelo, es más compacto, estructurado y rápido, ideal para ejecutarse en una plataforma como la tarjeta Jetson Nano.

La función principal con la que se obtiene el formato **pb** recibe cuatro parámetros como se muestra a continuación (el código completo se muestra en el Anexo D).

Este proceso también es conocido como *congelar* una sesión, es por ello el nombre de la función.

```
def freeze_session(session, keep_var_names=None,
                   output_names=None, clear_devices=True)
```

- **session:** Este parámetro se refiere a una sesión creada en la librería Keras, la cual permite ejecutar grafos o parte de grafos (la RNC en este caso), con ello se alojan recursos y se almacenan valores.
- **keep_var_names:** Sólo se utiliza si se desea que algunas variables no formen parte de este proceso, es por ello que en esta aplicación se designa como None.

- **output_names:** Es una lista con los nombres de las operaciones que producen las salidas deseadas del modelo de la RNC.
- **clear_devices:** Elimina variables del programa para hacer el grafo más portable.

Una vez obtenido el archivo portable en extensión **.pb** que representa la RNC, este se copia a la tarjeta Jetson Nano, se abre una terminal en la ubicación del archivo y se ejecuta el programa *ejecutar_rnc.py* que realiza la captura, inferencia y visualización de los signos introducidos por el usuario en tiempo real, dicho programa se encuentra en el Anexo E.

Carga del modelo y creación de etiquetas de clase

Por su parte, en el Anexo E se puede consultar el código completo el cual es ejecutado en la tarjeta Jetson Nano, a continuación se muestran las porciones de código más representativas de este proceso:

```
net = cv2.dnn.readNetFromTensorflow('modelo_RNC.pb')

labels=["A","B","C","D","E","F","G","H","I","L","M",
        "N","O","P","R","S","T","U","V","W","Y"]
```

La instrucción *readNetFromTensorflow* pertenece a la biblioteca *OpenCV* descrita previamente en el capítulo 2. En esta instrucción se recibe como parámetro el archivo portable con extensión **.pb** generado en el apartado anterior con la función *freeze_session*.

Captura de *frames* de video

```
cap = cv2.VideoCapture(cam_set)
```

VideoCapture es una instrucción en la cual se define un objeto de tipo *captura de video*, este objeto está condicionado por *cam_set*, el cual es una cadena de parámetros con la configuración de la cámara de video utilizada que en este caso fué la **Raspberry Camera v2**.

Una vez definido este objeto tipo *captura de video*, se ejecuta dentro de un cliclo *while* con el cual se capturan e infieren de forma continua frames de video a una taza de

refresco de aproximadamente 40 fps dependiendo de la carga del procesador, memoria RAM y GPU de la tarjeta Jetson Nano para cada inferencia.

En la siguiente porción de código se observa el objeto tipo cámara construido previamente con el que se obtiene en cada ciclo un frame o imagen, esta imagen se reescala a dimensiones de 72 x 88 píxeles, tal como las dimensiones de las imágenes de la base de datos, así mismo la imagen debe ser transformada a escala de grises.

```
ret, frame = cap.read()
oriFrame = frame
frame = cv2.resize(frame, dsize=(88, 72))
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

A continuación esta imagen transformada se convierte en un vector de dimensión 6336, el cual además se normaliza en el rango [0,1] dividiendo cada valor entre 255, que corresponde al valor máximo que puede presentar cada pixel.

```
blob = cv2.dnn.blobFromImage(frame)
blob /= 255
```

Las instrucciones subsecuentes corresponden al proceso mediante el cual se toma el vector de datos generado y se multiplica con los pesos de la RNC para de esta forma tener como salida un vector de 21 elementos, en donde cada elemento representa a una clase, es decir, un signo estático.

```
net.setInput(blob)
out = net.forward()
```

Una vez obtenido este vector es posible utilizar el método *argmax* con el cual se obtiene el índice del máximo valor en el vector correspondiente. Por último, con este índice se obtiene la etiqueta de clase almacenada previamente en el vector *labels* y se proyecta en la imagen de salida, la cual es el frame original antes de ser procesado. Dicho proceso se lleva a cabo con las siguientes líneas de código:

```
cv2.putText(oriFrame, labels[np.argmax(out)],
            (50,50), 0, 1, 255)
cv2.imshow("frame", oriFrame)
```

Capítulo 4

Pruebas y Resultados

En esta sección se muestran los resultados obtenidos del entrenamiento de la RNC basada en la arquitectura AlexNet así como la RNC propuesta, de la misma manera se muestra una comparación entre ambas arquitecturas basándose en la evolución de la función de pérdida y la exactitud en la etapa de entrenamiento con el conjunto de validación. Además se presenta la optimización de los hiperparámetros de la RNC propuesta al ejecutar 10 diferentes versiones de la misma y seleccionar aquella que presenta el mejor comportamiento en cuanto a la exactitud con el conjunto de entrenamiento y validación. Todos los entrenamientos y pruebas se realizaron con las imágenes de la base de datos integrada que se menciona en el capítulo 3. Por último la mejor versión de la RNC propuesta es transformada al formato adecuado para ser ejecutada por la tarjeta Jetson Nano y realizar la detección de signos en tiempo real.

4.1. Resultados del entrenamiento del modelo basado en AlexNet

El modelo basado en la arquitectura AlexNet exhibe un comportamiento caótico en las curvas de pérdida y exactitud con el conjunto de validación tal como se aprecia en la Figura 4.1, esto indica que se presenta un claro sobre ajuste del sistema. El modelo se entrenó con 20 épocas y el tiempo total de entrenamiento fue de 144.60 segundos

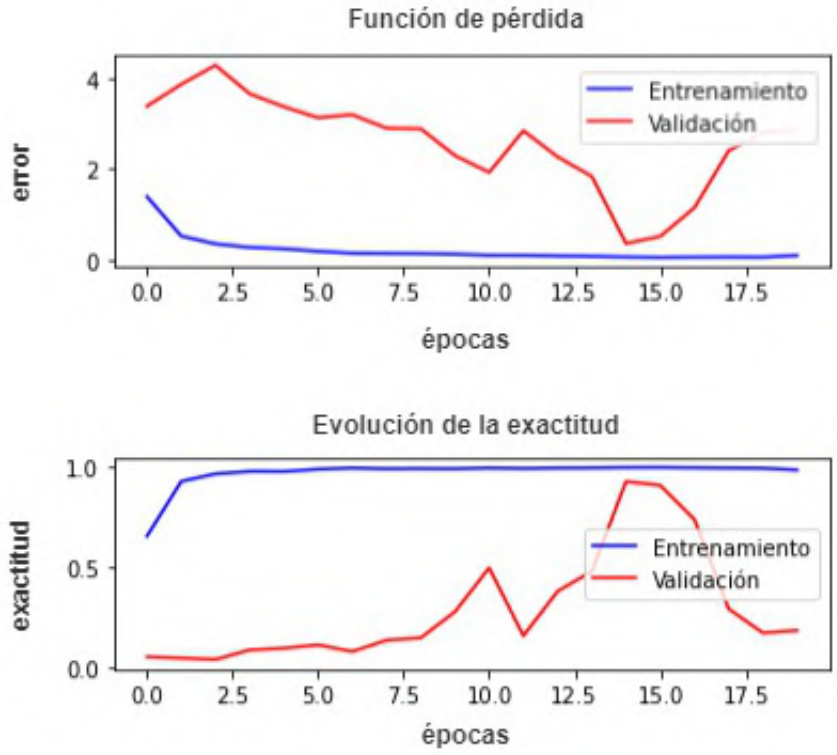


Figura 4.1. Respuestas del modelo basado en la arquitectura AlexNet; Arriba, comportamiento de la función de pérdida ante los conjuntos de entrenamiento y validación. Abajo, evolución del valor de la exactitud ante los mismos conjuntos.

El comportamiento de las gráficas anteriores corresponde al modelo utilizando los conjuntos de entrenamiento y validación. Una vez concluido este proceso y obtenidos los pesos de dicho modelo es puesto a evaluación con el conjunto de pruebas con lo cual el valor de la exactitud con este conjunto fue de 30,23 % reafirmando el hecho de que esta arquitectura sobreajusta el conjunto de entrenamiento lo cual no permite generalizar de una manera eficiente imágenes no vistas previamente por el modelo. Esto puede deberse a que el modelo fue concebido originalmente para trabajar con imágenes a color (RGB), como parte de las pruebas en este proyecto se sugirió utilizar dicho modelo sin embargo la arquitectura diseñada para identificar óptimamente características de las imágenes de entrada, esta no puede ser aplicada con los mismos hiperparámetros a imágenes a escala de grises sin sufrir un sobre ajuste con el conjunto de entrenamiento como lo visto anteriormente.

Una forma gráfica de observar el desempeño de la red es mediante una matriz de

confusión donde cada columna de la matriz representa una clase en particular, mientras que en cada fila se pueden ver el total de imágenes asignadas a dicha clase. En la Figura 4.2 se visualiza en un mapa de calor esta matriz de confusión donde colores más claros representan menor cantidad de datos y colores más obscuros una mayor cantidad de estos, es deseable por tanto, que la mayor cantidad de datos se encuentren en la diagonal principal.

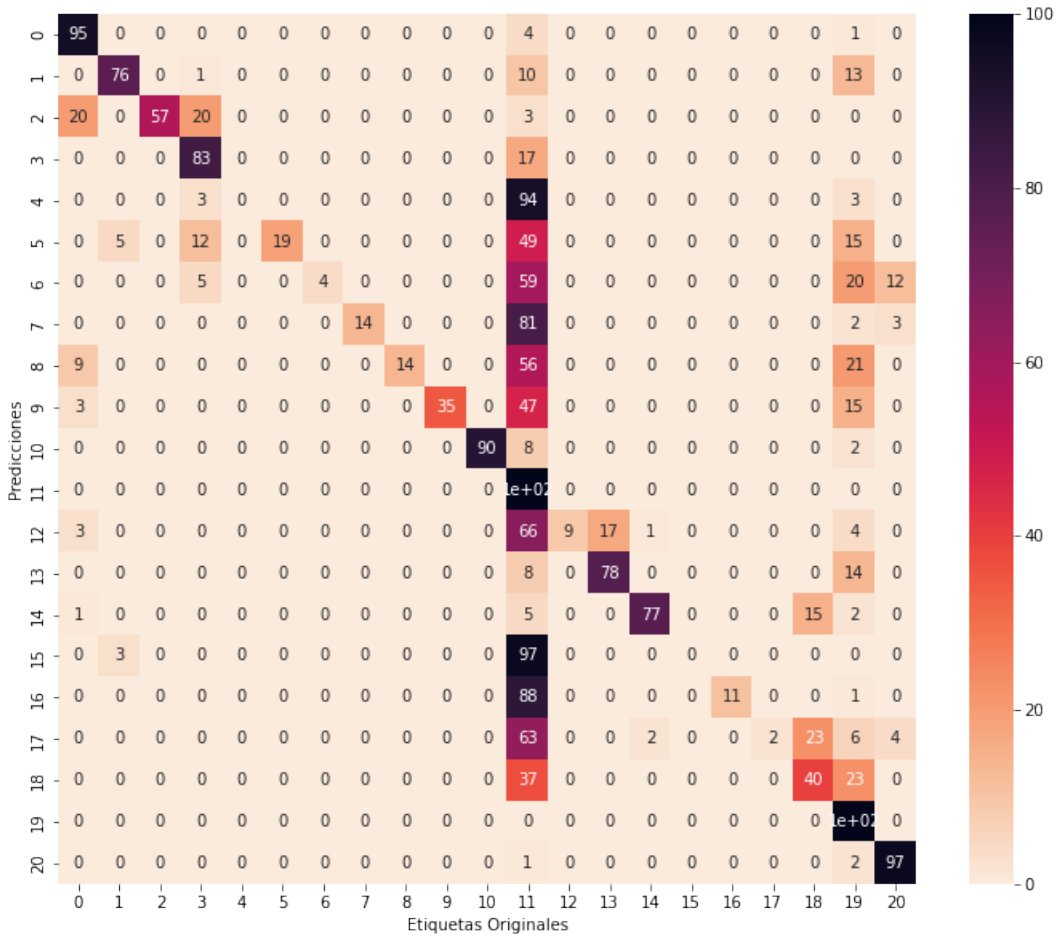


Figura 4.2. Matriz de confusión del modelo basado en la arquitectura AlexNet.

4.2. Resultados del entrenamiento del modelo con la arquitectura propuesta

La duración del entrenamiento sobre 20 épocas fue de un total de 142.63 segundos. La evolución del mismo durante esta fase se puede ver en la Figura 4.3.

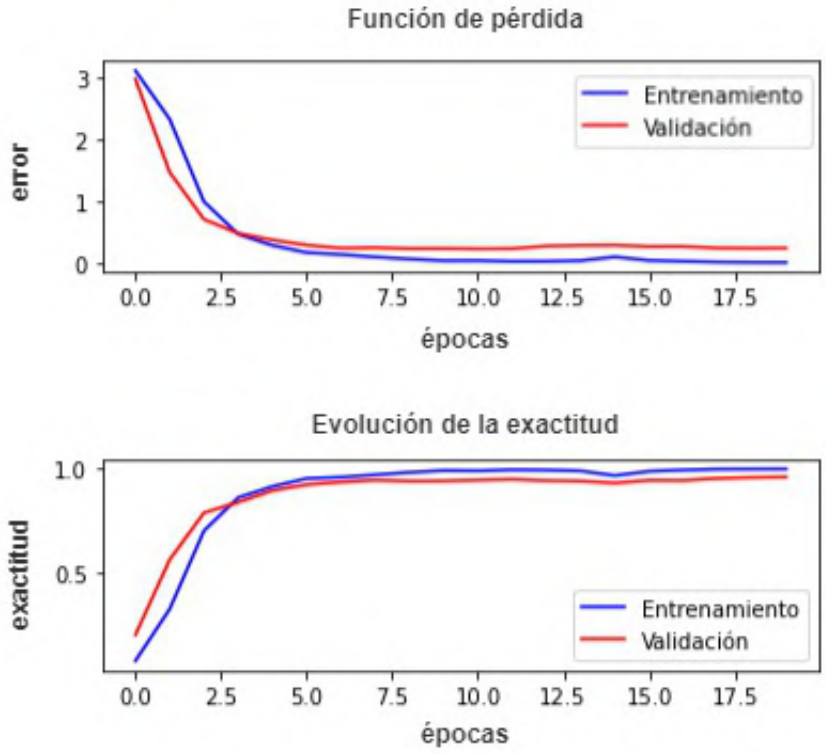


Figura 4.3. Respuestas del modelo de RNC propuesto; Arriba, comportamiento de la función de pérdida ante los conjuntos de entrenamiento y validación. Abajo, evolución del valor de la exactitud ante los mismos conjuntos.

El modelo parece alcanzar los mejores valores de exactitud y pérdida a partir de las épocas 7 a 10. Por su parte al ser evaluado este modelo con el conjunto de pruebas se obtiene una exactitud del 95,52 %.

La matriz de confusión de este modelo se muestra en la Figura 4.4.

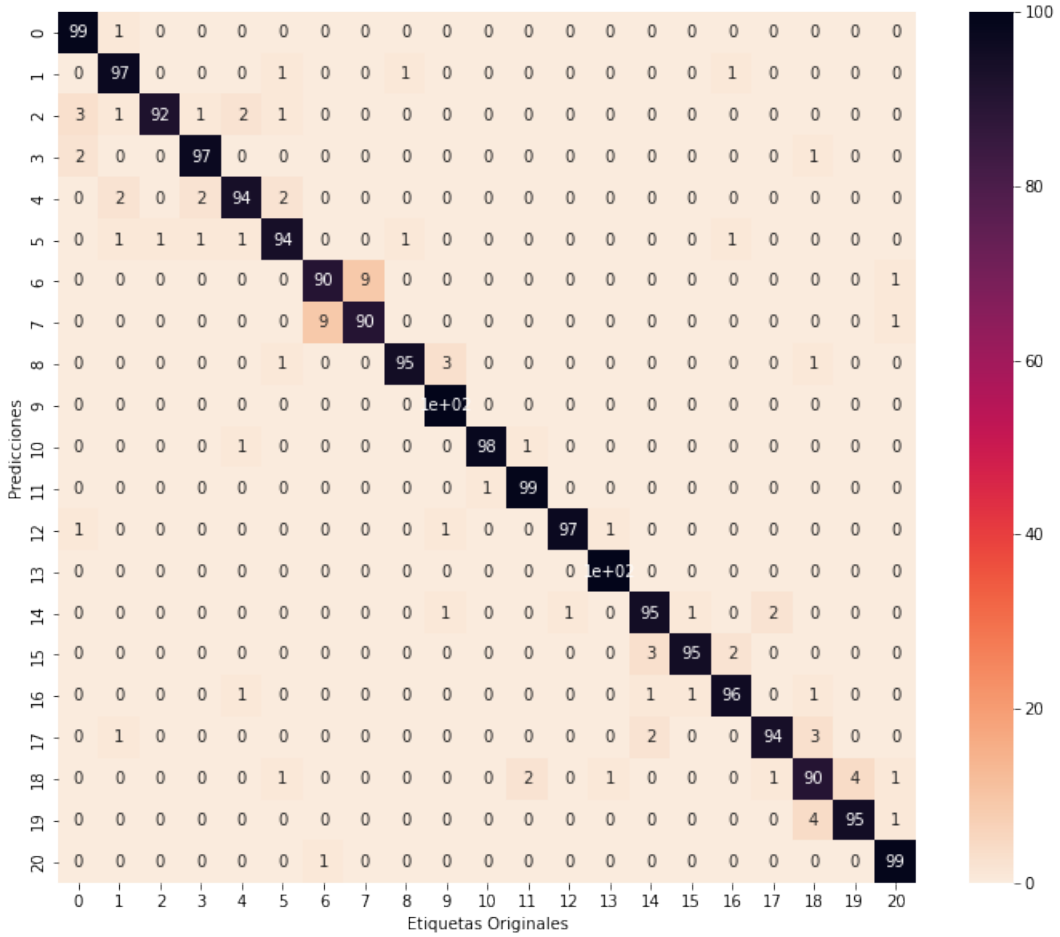


Figura 4.4. Matriz de confusión del modelo basado en la arquitectura propuesta.

Cada clase en el conjunto de pruebas contiene 100 imágenes, dicho esto se observa que en general hay una buena respuesta del modelo ante imágenes no vistas, ya que los valores en la diagonal principal son cercanos a este número 100. Por otro lado en las clases donde se logra observar una leve disminución de este valor son las correspondientes a las clases 6 y 7, es decir, las letras **H** y **G**. Esto podría deberse a la similitud que muestran ambos signos tal como puede observarse en la Figura 3.1

En el estado del arte se encontraron los dos mejores resultados al realizar sistemas de reconocimiento de signos de la LSM, los cuales fueron los obtenidos por Pérez y cols. (2017) en donde se obtuvo un 95,80 % de exactitud y el presentado por Solís y cols. (2016) en donde se reporta un valor para la misma métrica de 93,00 %. Estos resultados sólo se mencionan en este proyecto de investigación y no se hace una comparación con

ellos debido a la naturaleza distinta de los trabajos reportados y los modelos aquí desarrollados.

4.3. Resultados de la optimización de la RNC propuesta

El tiempo de entrenamiento y exactitud máxima de los modelos de RNC descritos en la Tabla 3.2 con respecto al conjunto de validación se muestran en la siguiente Tabla 4.1.

Tabla 4.1. Resultados del entrenamiento de los modelos basados en la RNC propuesta

Modelo número	**Tiempo de entrenamiento (segundos)**	**Exactitud con el conjunto de validación**
1	266	0.9420
2	200	0.9309
3	196	0.0539
4	142	0.0484
5	140	0.0523
6	41	0.0484
7	202	0.9420
8	146	0.9349
9	266	0.0523
10	202	0.9126

Los resultados anteriores muestran que los modelos 1 y 7 son los que exhiben la mayor exactitud con el conjunto de validación, sin embargo se eligió al modelo número 7 como el mejor debido a que en la segunda y tercera capa de convolución el modelo 1 se ha entrenado con una cantidad considerablemente mayor de filtros a los usados por el modelo número 7, esto se traduce en una RNC con mayor número de parámetros entrenables.

Las gráficas de pérdida y exactitud para este modelo número 7 se muestran en la Figura 4.5.

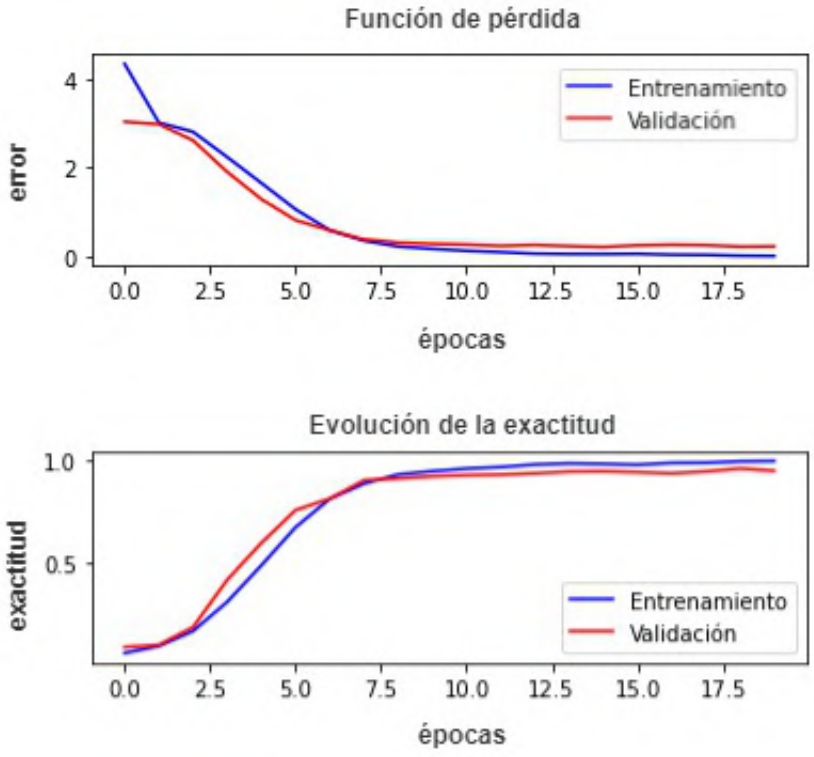


Figura 4.5. Respuestas del modelo optimizado; Arriba, comportamiento de la función de pérdida ante los conjuntos de entrenamiento y validación. Abajo, evolución del valor de la exactitud ante los mismos conjuntos.

Así mismo, al ser evaluado con el conjunto de pruebas es obtenida una exactitud del 94,85 %. El mapa de calor que describe la matriz de confusión para este modelo se muestra en la Figura 4.6.

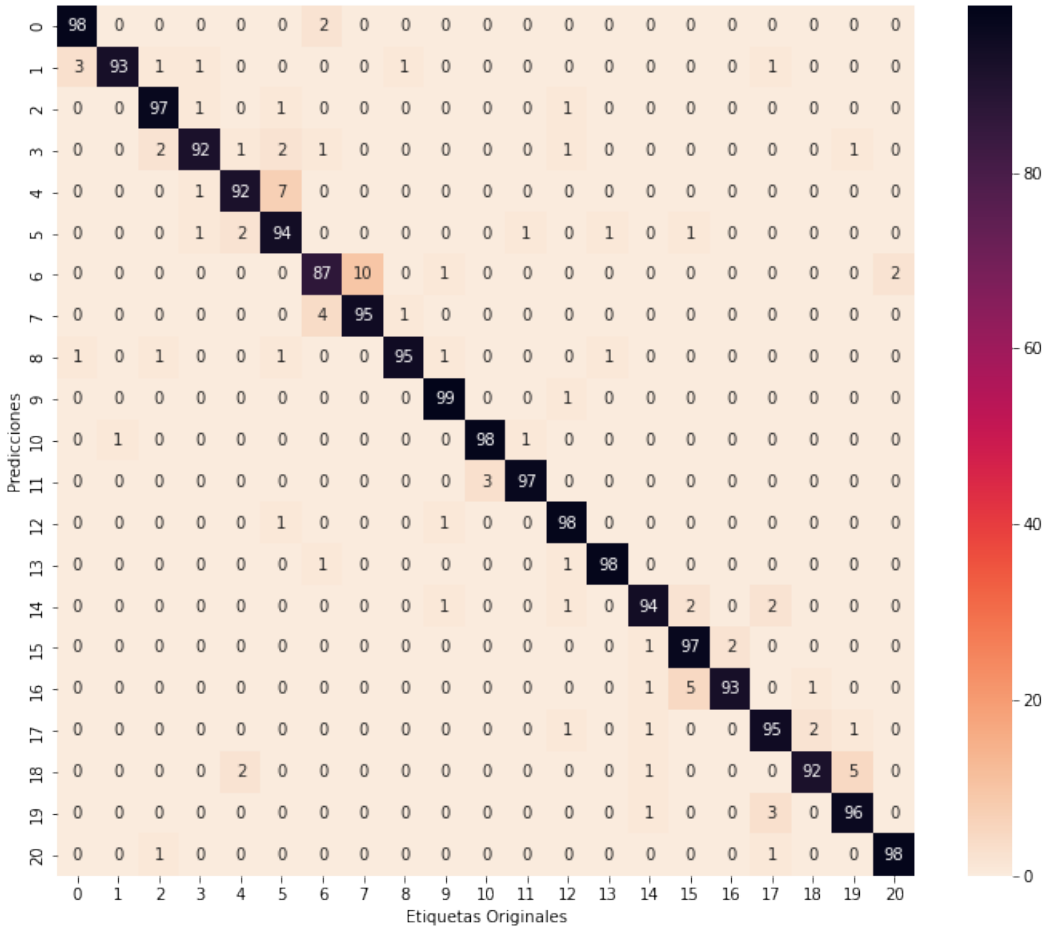


Figura 4.6. Matriz de confusión del modelo basado en la arquitectura propuesta.

Se puede notar que esta matriz de confusión luce muy similar a la obtenida por el modelo de RNC propuesto, ver Figura 4.4. De hecho la diferencia entre ambas exactitudes es mínima 94,85 % vs 95,52 %. Sin embargo con el proceso de optimización de la RNC propuesta se pudo obtener una RNC con un comportamiento similar y con una reducción de parámetros entrenables bastante significativa; $18,782,389$ para la RNC propuesta contra $6,642,645$ de la RNC optimizada. Y como menciona Zhu (2019), una RNC con mayor cantidad de parámetros implica un procesamiento más lento, así como una tendencia a sobreajustar los datos. Por tanto, este modelo número 7 obtenido del proceso de optimización es la RNC que se exportó a la tarjeta Jetson Nano para hacer detecciones en tiempo real, dichos resultados se muestran a continuación.

4.4. Pruebas experimentales con la RNC optimizada en la tarjeta Jetson Nano

En la Figura 4.7 se puede ver el entorno físico de pruebas, incluyendo la tarjeta Jetson Nano utilizada en este proyecto.

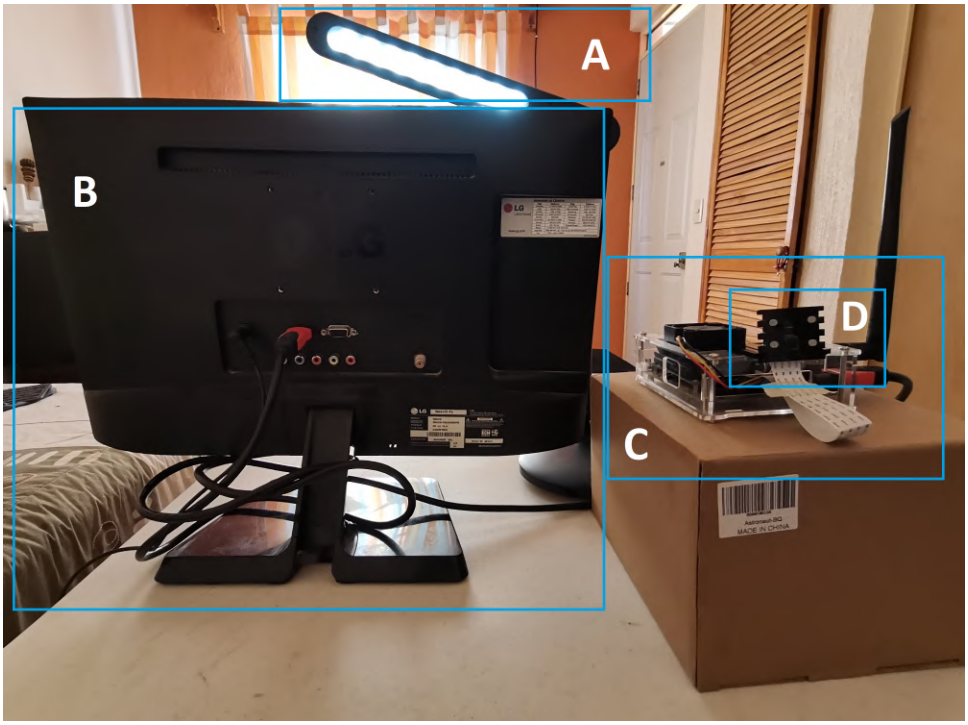


Figura 4.7. Entorno físico de pruebas, **A**: Luz artificial auxiliar, **B**: Monitor para visualización en el lado contrario del usuario, **C**: Tarjeta Jetson Nano y **D**: cámara Raspberry v2.

El usuario se coloca enfrente de la cámara Jetson Nano y detrás del monitor el cual mostrará el signo detectado por el sistema como se muestra en la Figura 4.8

Figura 4.8. Posición inicial del usuario situado en el lugar de pruebas antes de empezar el reconocimiento de signos.

Para empezar el proceso de reconocimiento de signos a través de la cámara de video, es necesario ejecutar el programa de python *ejecutar_rnc.py* en la tarjeta Jetson Nano y en el directorio en donde previamente fue guardado el modelo con extensión **.pb**. En la Figura 4.9 se observan tres usuarios realizando un signo frente a la cámara de video, y al sistema Jetson Nano realizando la detección del mismo.

Figura 4.9. Usuarios utilizando el sistema de reconocimiento de signos de la LSM, se puede apreciar en pantalla de izquierda a derecha el reconocimiento de los signos: **U**, **A** y **B**.

A manera de ejemplo considérese la Figura 4.10, la cual es una captura del modelo ejecutándose en tiempo real.

Figura 4.10. Captura y resultado de la predicción en tiempo real de la RNC en la tarjeta Jetson Nano.

La tabla 4.2 muestra el vector de probabilidades generado por la capa de salida de la RNC optimizada. Es posible ver que el mayor valor se encuentra en la clase **0**, es decir, la

clase correspondiente a la letra **A**. Es de esta manera como el sistema captura un signo realizado por el usuario y despliega la letra de la clase correspondiente.

Tabla 4.2. Vector obtenido al ser detectado el signo de la Figura 4.10.

Índice	**Valor**	**Letra correspondiente**
1	**7.30441272e-01**	A
2	1.11885434e-04	B
3	4.84441221e-03	C
4	1.24311191e-05	D
5	3.98626924e-03	E
6	2.36740409e-04	F
7	1.04747456e-03	G
8	1.39740124e-01	H
9	7.07302010e-04	I
10	1.11015759e-01	L
11	1.02730455e-05	M
12	4.33803980e-05	N
13	5.05478913e-03	O
14	7.82295305e-04	P
15	1.87354578e-06	R
16	1.59264903e-03	S
17	3.66651744e-04	T
18	3.10538724e-08	U
19	1.05809590e-07	V
20	4.04192679e-06	W
21	7.05386896e-08	Y

En la Figura 4.11 se puede observar un ejemplo de cada uno de los 21 signos dactilológicos realizados por un usuario así como la clasificación correspondiente indicada mediante una etiqueta de la letra correspondiente al signo predicho.

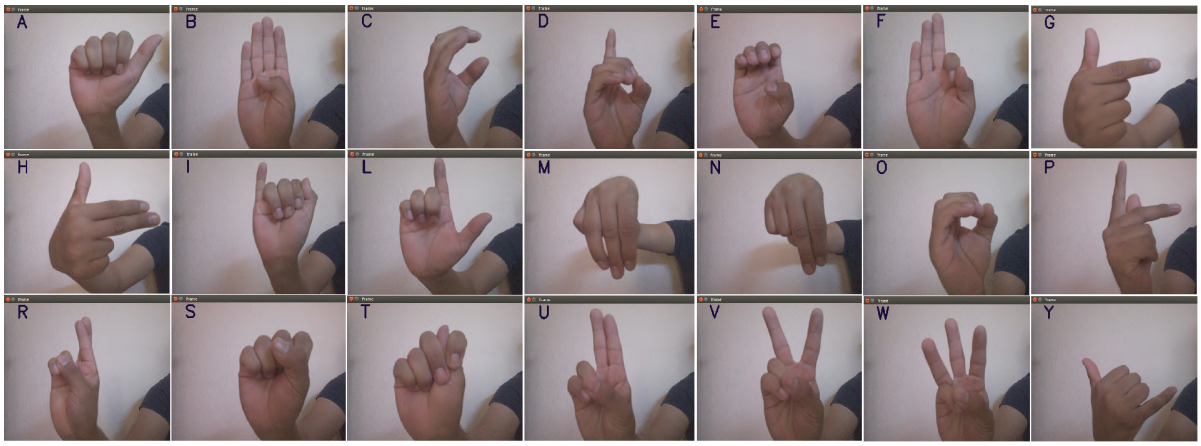


Figura 4.11. Capturas de pantalla del resultado de la clasificación de las 21 clases de signos de la LSM con la tarjeta Jetson Nano y la RNC optimizada.

Discusión

Es necesario discutir algunos aspectos relevantes obtenidos a lo largo de este trabajo de investigación. En primer lugar, se pudo comprobar que el sistema tiene la capacidad de converger a la clase ejecutada por el usuario cada vez que este realiza un signo enfrente del sistema de captura. Se notó también que la capacidad de clasificación, o dicho de otra forma, la robustez del sistema depende de factores como iluminación del entorno, variaciones en el fondo y la distancia a la cual la mano es acercada o alejada de la cámara. No fue posible hallar constantes de estas variables que permitieran la clasificación fluida para todos los signos. Recordar que las imágenes sólo fueron escaladas y transformadas a escala de grises, un posible tratamiento adicional de las imágenes a futuro podría mejorar la robustez de este sistema.

Además, se pudo notar que una arquitectura de RNC como AlexNet sobre ajusta las imágenes de entrenamiento y no permite una buena generalización de las mismas, esto se debe probablemente a que este modelo fue concebido originalmente para ser entrenado y probado con imágenes con tres capas de color, y el uso en imágenes en escala de grises con la misma eficacia no puede ser asumido.

Por su parte la arquitectura de RNC propuesta mostró una buena capacidad de generalización ante el conjunto de pruebas y, además pudo ser optimizada con lo cual se logró una exactitud similar reduciendo la complejidad original del modelo, dicha exactitud fue del 94.85 %.

Al analizar la matriz de confusión de la RNC propuesta y optimizada, pudo notarse que había una ligera variación en la exactitud de las predicciones de los modelos para las clases 6 y 7, estas clases correspondientes a signos similares en ejecución (G y H) permiten entender la necesidad futura de realizar transformaciones en las imágenes de

la base de datos y de esta manera el modelo pueda ser capaz de distinguir de mejor manera características propias de ambas clases.

Al recurrir a la literatura se encontró que los dos sistemas desarrollados para el reconocimiento de signos de la LSM con mejor comportamiento exhiben un 95.80 % y un 93.00 % de exactitud, sin embargo sólo el primero de ellos indica que se trabajó con una base de datos de 1,680 imágenes mientras la otra investigación no reporta dicha información. Por tanto, los resultados obtenidos en este proyecto de tesis no pueden ser directamente comparados con los anteriores debido a la naturaleza distinta de los proyectos. Sin embargo, la presente investigación es una contribución al estado del arte y además puede ser motivo de comparación en trabajos futuros debido al aporte que se realiza de la base de datos de 10,500 imágenes.

Conclusiones

En este proyecto de tesis se demostró que fue posible desarrollar un sistema que clasifique en tiempo real los signos estáticos del abecedario dactilológico de la Lengua de Señas Mexicana, dicha clasificación se llevó a cabo por medio de una red neuronal convolucional misma que se ejecutó en la tarjeta de desarrollo Jetson Nano.

Para completar el objetivo de tener 500 imágenes por clase se añadieron a la base de datos imágenes de adquisición propia ya que la suma de las imágenes de las bases de datos de otras lenguas de señas semejantes a la LSM no lograron completar este objetivo en primera instancia. Después del tratamiento de las imágenes se logró completar una base de datos de 10,500 imágenes la cual podrá servir como comparación de futuros modelos y con ello profundizar en el estado del arte de la clasificación de signos de la LSM.

Los pasos que describen la generación de un modelo exportable para la tarjeta Jetson Nano, así como las instrucciones de ejecución del programa de clasificación de signos permitieron crear un flujo de trabajo acorde a la metodología propuesta, en el cual se establece el proceso para entrenar un modelo en un ambiente externo a la tarjeta Jetson Nano, generar un modelo compatible con el mismo y ejecutarlo en tiempo real en dicha tarjeta.

Los resultados en un ambiente físico con voluntarios mostraron que el sistema es capaz de clasificar los signos ejecutados por los usuarios en tiempo real, sin embargo la robustez del mismo podría mejorarse al realizar algoritmos de tratamiento de imágenes a la base de datos como se mencionó en la sección de discusión, esto se deja como trabajo futuro.

Así mismo, se propone como trabajo futuro el análisis del tiempo de inferencia entre el modelo de RNC ejecutado en una computadora personal y en el dispositivo embebido Jetson Nano y de esta forma obtener métricas de desempeño entre ambos dispositivos.

Anexos

Anexo A

Acceso a la base de datos y configuración del entorno

A continuación se presenta el código necesario para acceder a la base de datos desde **Google Colaboratory**, instalar las bibliotecas indispensables, crear el vector de clases y desplegar una imagen aleatoria para corroborar el funcionamiento.

Las siguientes importaciones de bibliotecas son las necesarias para ejecutar el modelo de RNC.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow import keras

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense,
Flatten, Dropout, Reshape
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import BatchNormalization

from kerastuner import RandomSearch
from kerastuner.engine.hyperparameters \
import HyperParameters
```

La base de datos se encuentra alojada en **Google Drive**, las siguientes bibliotecas

se utilizan para la conexión con la misma.

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

Se establecen las variables globales.

```
classes = 21
epochs = 20
batch_size = 128
```

Conexión con la base de datos.

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get
_application_default()
drive = GoogleDrive(gauth)
```

Descarga del archivo que contiene la base de datos

```
downloaded = drive.CreateFile({'id':'1FyF_
Ut7WUxJ56FarHsxYbpY_DtN3HNLB'})
downloaded.GetContentFile('LSM_reduced.csv')
data = pd.read_csv('LSM_reduced.csv')
```

Codigo para generar las etiquetas de las 21 clases

```
target= np.full((500,1),30)
for i in np.arange(21):
    aux = np.full((500,1),i)
    target = np.vstack((target,aux))


target = np.delete(target,slice(0,500),0)
target=target.reshape(10500)
```

Normalizacion de las imagenes

```
X_train=X_train/255
X_test=X_test/255
```

Etiquetas de clase.

```
labels=["A","B","C","D","E","F","G","H","I","L",
"M","N","O","P","R","S","T","U","V","W","Y"]
```

Codigo para mostrar una imagen aleatoria de la base de datos.

```
indx=np.random.randint(low=0,high=X_train.shape[0])
img=X_train[indx,:].reshape(72,88)
plt.imshow(img,cmap='gray')
label=y_train[indx]
plt.title(labels[label])
```

Anexo B

Desarrollo de las arquitecturas de RNC

División de la base de datos en conjuntos de entrenamiento y pruebas

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split
(data, target,test_size=0.2,random_state=0,
stratify=target)

X_train=np.array(X_train,dtype='int')
X_test=np.array(X_test,dtype='int')
```

Codificacion One-Hot

```
def one_hot(x, n_clases=21):
  r=np.zeros((x.shape[0],n_clases))
  for i in np.arange(n_clases):
    r[:,i]=(x==i)*1
  return r

y_orig=y_test
y_test=one_hot(y_test)
y_train=one_hot(y_train)
```

Arquitectura AlexNet, desarrollo y entrenamiento

```
np.random.seed(1000)

AlexNet = Sequential()
```

```
AlexNet.add(Conv2D(filters=96, input_shape=(72,88,1),
kernel_size=(11,11), strides=(4,4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),
strides=(2,2),
padding='same'))

AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5),
strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),
strides=(2,2),
padding='same'))

AlexNet.add(Conv2D(filters=384, kernel_size=(3,3),
strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Conv2D(filters=384, kernel_size=(3,3),
strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Conv2D(filters=256, kernel_size=(3,3),
strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),
strides=(2,2),
padding='same'))

AlexNet.add(Flatten())

AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Dropout(0.4))
```

```
AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Dropout(0.4))

AlexNet.add(Dense(1000))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Dropout(0.4))

AlexNet.add(Dense(21))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('softmax'))

AlexNet.compile(loss = keras.losses.categorical\
_crossentropy,
optimizer= 'adam', metrics=['accuracy'])

import time
start = time.time()
history_alexNet = AlexNet.fit(X_train, y_train,
                    validation_data=(X_validate,
                    y_validate),
                    epochs=epochs,
                    batch_size=batch_size)
training_time = time.time() - start
print("Tiempo para entrenar el modelo: ",
training_time)
```

Arquitectura del modelo propuesto, desarrollo y entrenamiento

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(4,4),
activation='relu',
strides=1, input_shape=(72, 88, 1), padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(4,4),
activation='relu', strides=1))
model.add(Conv2D(64, kernel_size=(4,4),
```

```
activation='relu', strides=1))

model.add(Flatten())
#model.add(Reshape((1536,)))
model.add(Dense(256, activation='relu'))
model.add(Dense(classes, activation='softmax'))

model.compile(loss="categorical_crossentropy",
optimizer=Adam(lr=0.001), metrics=['accuracy'])


import time
start = time.time()
history = model.fit(X_train, y_train,
                    validation_data=(X_validate,
                    y_validate),
                    epochs=epochs,
                    batch_size=batch_size)
training_time = time.time() - start
print("Tiempo para entrenar el modelo: ",
training_time)
```

Anexo C

Optimización de la RNC

En la siguiente función se redefine el modelo de RNC propuesto.

```
def build_model(hp):
  model = keras.models.Sequential()
  model.add(Conv2D(hp.Int('input_units',
  min_value=16,
  max_value=128, step=16),(4,4),
  input_shape=(72, 88, 1),
  padding='same'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  for i in range(hp.Int("number_conv_layers",1,3)):
    model.add(Conv2D(hp.Int(f'conv_{i}_units',
    min_value=16,
    max_value=128, step=16),(4,4), padding='same'))
  model.add(Flatten())

  model.add(Dense(256, activation='relu'))
  model.add(Dense(classes, activation='softmax'))

  model.compile(loss="categorical_crossentropy",
  optimizer=keras.optimizers.Adam(hp.Choice
  ('learning_rate',
  values=[1e-2,1e-3,1e-4])) , metrics=['accuracy'])
  return model
```

El entrenamiento de los 10 modelos se realiza con la siguiente instrucción.

```
tuner = RandomSearch(
    build_model,
    objective = "val_accuracy",
```

```
    max_trials = 10,
    executions_per_trial = 1,
    directory = 'output',
    project_name='lsm_selection'
)

tuner.search(x=X_train,
             y=y_train,
             epochs=10,
             validation_data=(X_validate, y_validate))
```

Y el mejor modelo se obtiene de la siguiente manera:

```
best_model=tuner.get_best_models(num_models=1)[0]

model = best_model
model.save('modelo_seleccionado.h5')
```

Anexo D

Modelo exportable para la tarjeta de desarrollo Jetson Nano

A continuación se presenta el código para transformar un modelo **.h5** y obtener un archivo de tipo **.pb** compatible con la tarjeta Jetson Nano.

```
from keras import backend as K
K.set_learning_phase(0)

from keras.models import load_model
kera_model_path = 'cnn_test.h5'
model = load_model(kera_model_path.encode())

from keras import backend as K
import tensorflow as tf

def freeze_session(session, keep_var_names=None,
output_names=None, clear_devices=True):

    from tensorflow.python.framework.graph_util import
    convert_variables_to_constants
    graph = session.graph
    with graph.as_default():
        freeze_var_names = list(set(v.op.name for v in
        tf.global_variables()).difference
        (keep_var_names or []))
        output_names = output_names or []
        output_names += [v.op.name for v in
        tf.global_variables()]
```

```
        input_graph_def = graph.as_graph_def()
        if clear_devices:
            for node in input_graph_def.node:
                node.device = ""
        frozen_graph = convert_variables_to_constants
        (session, input_graph_def,
        output_names, freeze_var_names)
            return frozen_graph


frozen_graph = freeze_session(K.get_session(),
                              output_names=[out.op.name
                              for out in model.outputs])


tf.train.write_graph(frozen_graph, "modelo",
"modelo_rnc.pb", as_text=False)
```

Anexo E

Ejecución del modelo en tiempo real

En el siguiente código se muestran las instrucciones que permiten la carga del modelo y la ejecución en tiempo real en la tarjeta Jetson Nano. El nombre de este archivo es: *ejecutar_rnc.py*

```
import numpy as np
import cv2


if __name__ == '__main__':

    labels=["A","B","C","D","Epyt","F","G","H","I","L","M",
    "N","O","P","R","S","T","U","V","W","Y"]

    # cargar el modelo
    net = cv2.dnn.readNetFromTensorflow('tf_model.pb')

    # correr el modelo en tiempo real
    dispW=640
    dispH=480
    flip=2
    cam_set = cv2.VideoCapture(cam_set)

    if not cap.isOpened():
        print("Camara no detectada")

    while True:
```

```
ret, frame = cap.read()

oriFrame = frame
# redimensionar el frame y normalizar
frame = cv2.resize(frame, dsize=(88, 72))
frame =  cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
blob = cv2.dnn.blobFromImage(frame)
blob /= 255

net.setInput(blob)

# datos introducidos al modelo
out = net.forward()

oriFrame = cv2.cvtColor(oriFrame,
cv2.COLOR_BGR2GRAY)

# imprimir etiqueta correspondiente
cv2.putText(oriFrame, labels[np.argmax(out)],
(50,50), 0, 1, 255)
```

Anexo F

Artículo aceptado en el congreso SOMI XXXV

SOMIXXXV
CONGRESO DE INSTRUMENTACIÓN
Y 1er. SIMPOSIO NACIONAL DE BIOSENSORES
27 AL 29 DE OCTUBRE DE 2021
Año 07, No 01, octubre 2021 ISSN 2395-8499

Implementación y comparación de los métodos KNN y CNN para el reconocimiento de signos estáticos de la lengua de señas mexicana

Alfredo Estévez-Acosta, Rosebet Miranda-Luna
División de estudios de Posgrado, Universidad Tecnológica de la Mixteca
69000 Huajuapan de León, Oaxaca, Mexico
alfredo.estevez.acosta@gs.utm.mx; rmiranda@mixteco.utm.mx

RESUMEN

En este artículo se presenta el reconocimiento automático de signos estáticos pertenecientes al abecedario dactilológico de la Lengua de Señas Mexicana (LSM) utilizando dos técnicas distintas, una red neuronal convolucional (CNN) y un clasificador basado en el método de los k vecinos más cercanos (KNN). Estos se entrenaron y probaron con una base de datos compuesta de 10500 imágenes en escala de grises que corresponden a los signos realizados por distintas personas con la mano derecha y cuyas dimensiones son de 88 píxeles de ancho y 72 píxeles de alto. La CNN propuesta muestra una exactitud de 97.85%, y se compara con una arquitectura profunda conocida como Alex Net, la cual exhibe un sobreajuste sobre el conjunto de entrenamiento, lo cual demuestra que una implementación de esta arquitectura con los hiper parámetros originales no es adecuada para esta aplicación. Por otro lado el clasificador KNN se sugiere utilizar un valor de vecinos cercanos K=3, mismo que es obtenido a partir de analizar el comportamiento de K con valores impares de [1,39] y puesto a prueba con los conjuntos de entrenamiento y validación.

PALABRAS CLAVE: Redes Neuronales Convolucionales - K vecinos más cercanos - Lengua de Señas Mexicana - Abecedario dactilológico.

1 INTRODUCCIÓN

La lengua de señas permite a las personas sordas comunicarse con otras personas sordas o no, que conozcan la lengua de señas empleada.

El abecedario dactilológico, es decir, las configuraciones manuales del alfabeto, en este caso la lengua de señas mexicana está compuesto por 27 signos realizados mediante la posición y movimiento de flexión de los dedos de una sola mano, ya sea derecha o izquierda, 21 de los cuales se representan de forma estática y 6 de forma dinámica.

Los sistemas de reconocimiento automático de señas de la LSM se han orientado en su mayoría a la clasificación de signos estáticos y realizados con una mano [1]. Dentro de estos sistemas

Año 07, No 01, octubre 2021 ISSN 2395-8499

encontramos los que se basan en la adquisición y procesamiento de imágenes de la mano. Dichos sistemas se basan en cámaras de video convencionales para la adquisición de las imágenes y

diferentes técnicas de clasificación como Fuzzy C Means (FCM), y red neuronal multicapa tipo perceptrón, para el reconocimiento de los signos [2]. El empleo de sistemas de iluminación, así como un fondo uniforme que contrasta con la piel, son estrategias que han sido utilizadas para disminuir el efecto negativo del ruido en las imágenes y mejorar el desempeño de los sistemas [3].

Los mejores desempeños reportados en el estado del arte para un sistema de reconocimiento de signos dactilológicos de la LSM han sido de 95.8% [2] y 93% [3].

En este trabajo se presenta la implementación de dos métodos de clasificación para el reconocimiento de signos dactilológicos estáticos de la LSM, el primero basado en el método de los K vecinos más cercanos (KNN), y el segundo en una red neuronal convolucional (CNN). La arquitectura propuesta en este trabajo para la CNN se compara con una arquitectura profunda conocida como Alex Net. En la siguiente sección se presenta la obtención y procesamiento de la base de datos, así como el desarrollo de los dos métodos mostrados en este trabajo. En la sección 3 se ubican los resultados y comparativas entre estos métodos bajo distintas métricas, por último en la sección de conclusiones se retoman los resultados aquí obtenidos con los consultados en el estado del arte así como una propuesta de trabajo futuro.

2 MATERIALES Y MÉTODOS

2.1 Base de datos

Los clasificadores expuestos en este artículo se entrenaron y probaron con una base de datos compuesta de 10500 fotografías pertenecientes a los 21 signos estáticos de la LSM, los cuales son: A, B, C, D, E, F, G, H, I, L, M, N, O, P, R, S, T, U, V, W, Y.

Estas imágenes fueron obtenidas por una cámara web, en un fondo liso, con iluminación natural semejante y con capturas fotográficas de una resolución de 176 píxeles de anchura y 144 píxeles de altura a una distancia de 12-35 cm, correspondientes a seis voluntarios. Todas las imágenes capturadas son realizadas con la mano derecha.

A continuación las imágenes fueron escaladas a dimensiones de 72 píxeles de alto y 88 píxeles de ancho, enseguida se transformaron a escala de grises ya que originalmente se obtuvieron con tres capas de color: RGB, para ello se utilizó el estándar ITU-R 601-1 mismo que se encuentra en la biblioteca pillow, y su implementación se realizó en el lenguaje python. Finalmente se normalizaron los valores de los píxeles de cada imagen en el rango de [0,1].

En la Figura 1 se muestran ejemplos de las imágenes contenidas en la base de datos. En resumen, se tienen 10500 imágenes distribuidas balanceadamente en 21 clases (500 imágenes por clase), cada imagen se describe con un vector de 6337 valores, 6336 correspondientes a los píxeles de

Año 07, No 01, octubre 2021 ISSN 2395-8499

la imagen y el último valor es su etiqueta correspondiente. Los conjuntos de entrenamiento y pruebas se obtienen dividiendo el conjunto de datos en 80% y 20% respectivamente.

Figura 1. Base de datos de la LSM.

2.2 Red Neuronal Convolucional

Para enfrentar este problema de clasificación mediante redes neuronales convolucionales, se ha propuesto la arquitectura mostrada en la Figura 2.

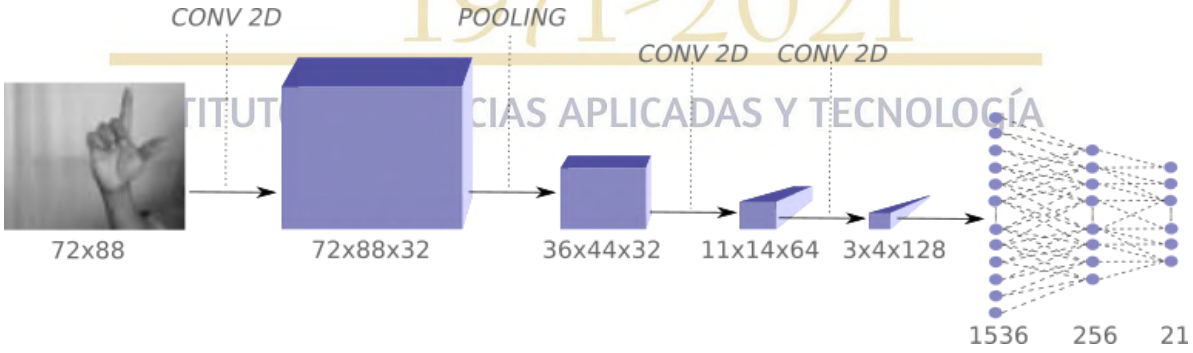


Figura 2. Arquitectura implementada para la RNC.

Año 07, No 01, octubre 2021 ISSN 2395-8499

Este modelo presenta las siguientes características:

- Primera capa de convolución: Con un tamaño de kernel de 4 x 4 y un paso de 1 se obtienen 32 mapas de características con dimensiones de 72 x 88 píxeles (px). Una capa de dropout es aplicada para prevenir el sobreajuste la cual retiene el 75% de los datos, seguido de una ventana de pooling de 2 x 2 la cual es usada para la operación de submuestreo reduciendo el tamaño de los 32 mapas de características a dimensiones de 36 x 44 px.
- Segunda capa de convolución: Con las mismas dimensiones de kernel y paso son generados 64 mapas de características de dimensiones 11 x 14 px, enseguida una capa de dropout es aplicada de la misma forma, una capa de pooling no es requerida.
- Tercera capa de convolución: 128 mapas de características son generados con dimensiones de 3 x 4 px seguido de una capa de dropout, la capa de pooling de la misma forma no es considerada.

Más adelante, estos elementos son la entrada de un clasificador con una capa oculta de 256 neuronas. Debido a que se tienen 21 clases distintas de signos, la capa de salida contiene 21 neuronas. Se eligió un tamaño de bloques de 128, así como la entropía cruzada como función de costo y el optimizador Adam con una taza de aprendizaje de 0.001.

SOMIXXXV
CONGRESO DE INSTRUMENTACIÓN
Y 1er. SIMPOSIO NACIONAL DE BIOSENSORES
BIOSENSORES
27 AL 29 DE OCTUBRE DE 2021
Año 07, No 01, octubre 2021 ISSN 2395-8499

2.3 K vecinos más cercanos

El algoritmo implementado se puede resumir en los siguientes pasos, el cual se repite para cada imagen nueva a clasificar:

- Se utilizó la distancia Euclidiana para determinar la similaridad entre la imagen a clasificar y el resto de las imágenes de la base de datos. Sean $A = (x_1, x_2, \ldots, x_m)$ y $B = (y_1, y_2, \ldots, y_m)$ dos vectores de dimensiones m, donde $m = 6336$ y cuyo valor representa la cantidad de píxeles o características para cada imagen. La distancia entre Ay Bse calcula como:

$$dist\ (A, B) = \sqrt{\frac{\sum_{i=1}^{m} (x_i - y_i^2)}{m}} \quad (1)$$

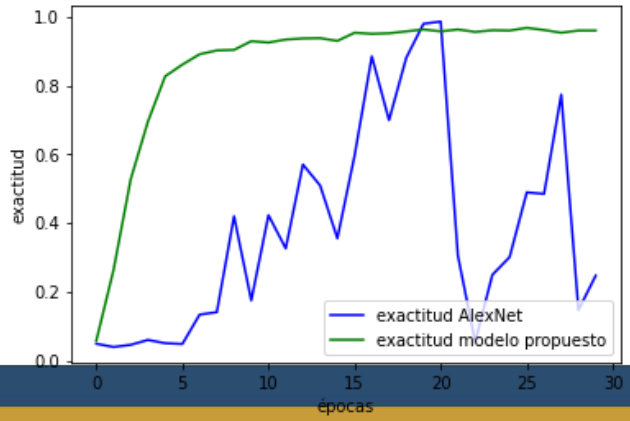
- Una vez calculadas todas las distancias de las imágenes de la base de datos con respecto a la imagen a clasificar se seleccionaron las 'K' menores distancias y sus correspondientes etiquetas de clase.
- De esta vecindad de tamaño K, se obtiene la clase mayoritaria y esta se le asigna a la imagen a clasificar.

3 RESULTADOS

3.1 Redes Neuronales Convolucionales

El comportamiento de la arquitectura propuesta se compara con otra arquitectura profunda conocida como Alex Net [4], la cual se compone de 5 capas de convolución y tres capas ocultas para la clasificación, los hiper parámetros se dejan como en el modelo original, ésta arquitectura se implementó y entrenó con la base de datos antes mencionada.

Para analizar ambos comportamientos se utiliza un conjunto de validación y un conjunto de entrenamiento generados a partir del 15% y 85% del conjunto de entrenamiento original, el comportamiento de la exactitud y la función de costo se muestra en la Figura 3 y 4.



SOMIXXXV
CONGRESO DE INSTRUMENTACIÓN
Y 1er. SIMPOSIO NACIONAL DE BIOSENSORES
BIOSENSORES
27 AL 29 DE OCTUBRE DE 2021
Año 07, No 01, octubre 2021 ISSN 2395-8499

Figura 3. Exactitud del modelo propuesto y la arquitectura Alex Net con los conjuntos de entrenamiento y validación.

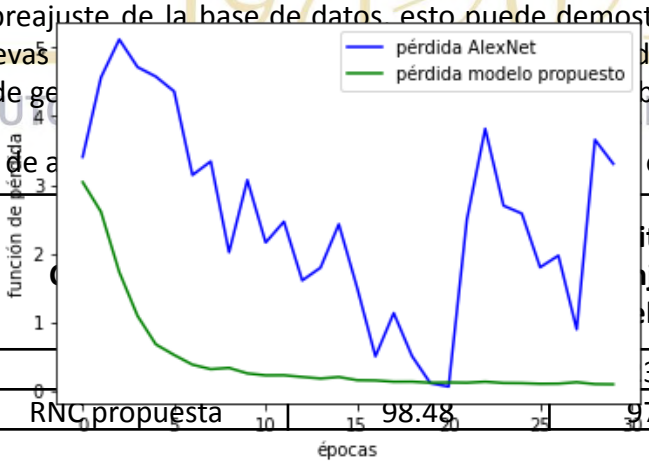


Figura 4. Comportamiento de la función de pérdida de ambos modelos con los conjuntos de entrenamiento y validación.

La arquitectura propuesta ayuda a evitar comportamientos caóticos en las curvas de exactitud y pérdida respecto a Alex Net, esto debido a que la configuración de la red neuronal convolucional propuesta evita el sobreajuste de la base de datos, esto puede demostrarse al observar la exactitud para las imágenes nuevas [illegible] decir, la RNC propuesta exhibe una buena capacidad de ge[illegible]bla 1.

Tabla 1. Exactitud de a[illegible] entrenamiento y pruebas

		itud ante ojunto de ebas (%)
		3.19
RNC propuesta	98.48	97.85

Año 07, No 01, octubre 2021 ISSN 2395-8499

3.2 K vecinos más cercanos

El comportamiento de la exactitud con los conjuntos de prueba y validación para diferentes valores impares y enteros de K en el rango [1,39] se muestra en la Figura 5.

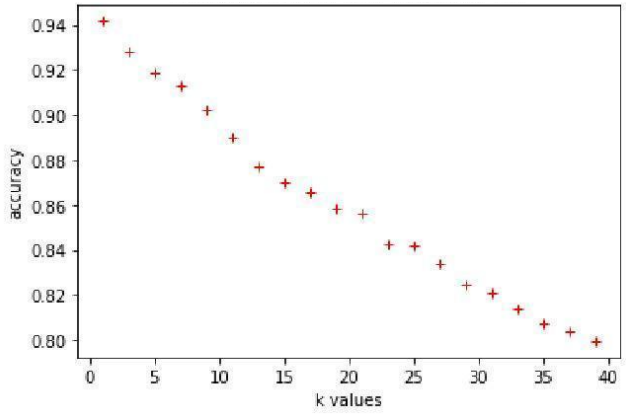


Figura 5. Evolución de la exactitud de KNN con el conjunto de validación

A pesar de que el mejor resultado se halla con K=1, no es recomendable tomar un valor de K=1 debido a que el clasificador se vuelve muy sensible al ruido como lo es el cambio en la iluminación de la imagen. Por tanto, se eligió un valor de K=3, con este valor se entrenó de nuevo al sistema con el

80% de datos originales y un 20% para pruebas, la exactitud de clasificación que exhibe este clasificador con dichos parámetros es del 94.71%.

3.3 Comparativas

Los porcentajes en cuanto a la exactitud de clasificación de los clasificadores KNN y RNC propuestos son muy cercanos, por ello para tener una mejor perspectiva del comportamiento de cada uno se propone analizarlos bajo otras métricas. La diferencia más amplia se encuentra cuando se calcula la precisión de ambos clasificadores, 98.67% que exhibe la RNC contra 94.78% que alcanza KNN, esta métrica se refiere a la proporción de asignaciones positivas correctas, es decir, para este caso, de los signos que el algoritmo clasificó a una clase como correctos, cuántos verdaderamente pertenecen al signo que corresponde. Por otro lado, observando los puntajes en cuanto a la sensibilidad se concluye que la RNC clasifica correctamente los signos un 3.12% mejor que KNN. Una forma simple de combinar precisión y sensitividad y de ésta forma comparar dos clasificadores, es por medio de la métrica llamada F1-score, dicho esto se obtiene un puntaje de 94.21% para KNN y 97.87% para la RNC, es decir que la RNC se presenta como un mejor clasificador a comparación de KNN, siendo su mayor ventaja la precisión, aún así KNN se mantiene como un algoritmo sencillo, con un buen rendimiento así como una buena capacidad de generalización.

Año 07, No 01, octubre 2021 ISSN 2395-8499

Estos algoritmos presentados a lo largo de este trabajo se comparan con los encontrados en el estado del arte y que muestran los dos mejores resultados en cuanto a la exactitud de clasificación de sus modelos, dicha comparación se observa en la Tabla 2.

Tabla 2. Comparativa entre los clasificadores mostrados y los presentados por [2] y [3]

Clasificador	**RNC Propuesta**	**KNN**	**Alex Net**	**FCM** [2]	**Algoritmo Pavlidis** [3]
Exactitud (Porcentaje)	97.85	94.71	23.19	95.80	93.00
Tamaño de la base de datos	10500	10500	10500	1680	No indicado
Número de signos a clasificar	21	21	21	21	21
Tipo de signos	Estáticos	Estáticos	Estáticos	Estáticos	Estáticos

4 CONCLUSIONES

Comparativas entre clasificadores con metodologías distintas fueron presentadas para el reconocimiento automático de signos estáticos del abecedario dactilológico de la lengua de señas mexicana. Los resultados experimentales muestran que la arquitectura Alex Net no debería ser utilizada con los mismos parámetros originales de la red debido a que causa un claro sobreajuste, sino que se debe adaptar a través de variaciones en su estructura. Con lo cual el modelo propuesto presenta un mejor comportamiento de generalización. Por otro lado, este modelo también presenta un mejor comportamiento en cuanto a la exactitud, precisión y sensibilidad de clasificación en comparación con el clasificador KNN. Sin embargo KNN se presentó como un método sencillo y con una buena respuesta de clasificación ante los elementos del conjunto de pruebas, estos métodos comparados con los dos mejores obtenidos en el estado del arte presentan un mejor comportamiento en cuanto la exactitud.

Como trabajo futuro se sugiere implementar dichos algoritmos presentados; red neuronal convolucional y K vecinos más cercanos, en tiempo real y con un dispositivo de captura como una cámara web, y de esta forma observar su comportamiento ante diferentes entornos de iluminación.

Año 07, No 01, octubre 2021 ISSN 2395-8499

REFERENCIAS

[1] Wadhawan, Ankita, and Parteek Kumar. :"Sign Language Recognition Systems: A Decade Systematic Literature Review." Archives of Computational Methods in Engineering, 2019; 1-29.

[2] Pérez, L. M., Rosales, A. J., Gallegos, F. J., y Barba, A. V.: Lsm static signs recognition using image processing. **14th international conference on electrical engineering, computing science and automatic control (cce)** , pp. 1–5, 2017.

[3] Solis, F., Martınez, D., y Espinoza: Automatic Mexican sign language recognition using normalized moments and artificial neural networks. Engineering, 8 (10), 2016; 733–740.

[4] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012, vol. 25, p. 1097-1105.

Bibliografía

Alpaydin, E. (2020). *Introduction to machine learning.* MIT press.

Ballesteros, A. (2015). *Las redes neuronales multicapa.* http://www.redes-neuronales.com.es/tutorial-redes-neuronales/Las-redes-neuronales-multicapa.htm.

Berzal, F. (2019). *Redes neuronales & deep learning: Volumen ii.* Independently published.

Burch, C., y cols. (2001). A survey of machine learning. *A survey for the Pennsylvania Governor's School for the Sciences*.

Chen, H., Liu, S., Zhang, H., y Cheng, W. (2019). Handwriting numerals recognition using convolutional neural network implemented on nvidia's jetson nano. En *International conference in communications, signal processing, and systems* (pp. 529–535).

ENPD. (2012). Encuesta nacional sobre discriminación en méxico 2010. *Espiral (Guadalajara)*, *19*(54), 261–270.

Espinosa Aguilar, P. A., y Pogo León, H. A. (2013). *Diseño y construcción de un guante prototipo electrónico capaz de traducir el lenguaje de señas de una persona sordomuda al lenguaje de letras* (B.S. thesis).

Evermann, G., Chan, H. Y., Gales, M. J., Jia, B., Mrva, D., Woodland, P. C., y Yu, K. (2005). Training lvcsr systems on thousands of hours of data. En *Proceedings.(icassp'05). ieee international conference on acoustics, speech, and signal processing, 2005.* (Vol. 1, pp. I–209).

Fernández, S. (2007). *MS Windows NT kernel description.* Descargado 2020-02-30, de `https://www.xatakamovil.com/movil-y-sociedad/ya-existen-los-guantes-que-haran-que-los-sordomudos-puedan-hablar-a-traves-del-movil`

Garcia, O. (2019). *Redes neuronales artificiales: Qué son y cómo se entrenan.* https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i.

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media.

Gomilla, J. (2019). *Deep learning de a a z: redes neuronales en python desde cero.* España: Udemy.

González, F. A. (2015). *Modelos de aprendizaje computacional en reumatología.* Elsevier.

González, G. C., alberto Fernández, C., Cisneros, J. A., y cols. (2020). Hacia un sistema de software basado en ihc para el apoyo de niños con capacidades auditivas diferentes. *ReCIBE, Revista electrónica de Computación, Informática, Biomédica y Electrónica*, *9*(1), 5–12.

Granda Cárdenas, A. A. (2020). *Comparativa de extractores de características para clasificación de rostros* (B.S. thesis). Quito, 2020.

Harrington, P. (2012). *Machine learning in action.* Manning Publications Co.

Haykin, S. S., y cols. (2009). *Neural networks and learning machines/simon haykin.* New York: Prentice Hall,.

Hernández, C. (2010). Diccionario español-lengua de señas mexicana (dielseme), estudio introductorio al léxico de la lsm. *Secretaría de Educación Pública, Dirección de Educación Especial*.

Howse, . M. J., J. (2020). *Learning opencv 4 computer vision with python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning.* Packt Publishing Ltd.

INEGI. (2012). *Estadísticas a propósito del día internacional de las personas con discapacidad.* INEGI México.

Izaurieta, F., y Saavedra, C. (2000). Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*.

Jimenez, J., Martin, A., Uc, V., y Espinosa, A. (2017). Mexican sign language alphanumerical gestures recognition using 3d haar-like features. *IEEE Latin America Transactions*, *15*(10), 2000–2005.

JingHua, B., Xian, Z. X., ZhiXin, L., y XiaoPing, L. (2012). Mixture models for web page classification. *Physics Procedia*, *25*, 499–505.

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, 1097–1105.

Lazcano. (2020). *Definiciones de la inteligencia artificial.* https://blog.enzymeadvisinggroup.com/inteligencia-artificial-machine-learning.

Matallana, L. B. (2019). Traductor automático de la lengua de signos española a español mediante visión por computador y redes neuronales.

Matney, L. (2019). *Once poised to kill the mouse and keyboard, leap motion plays its final hand.* https://techcrunch.com.

Mendoza, E., y Jackson-Maldonado, D. (2020). Lectura de palabras por personas sordas usuarias de lengua de señas mexicana. *Revista de Logopedia, Foniatría y Audiología*, *40*(1), 4–11.

Montaño, L. A. F., y Rodríguez-Aguilar, R. M. (2011). Automatic translation system from mexican sign language to text. *Advances in Computational Linguistics*, 53.

Morales, B., Muñoz, A., Morales, E., Zaragoza, M. J. C. H., Gómez, M. J. I. V., Govea, B. A. V., ... others (2009). Aprendizaje cooperativo de conceptos para sistemas multi-robot.

Murillo, A. (2012). Kinect para developers. *Obtenido de Kinect for developers: http://www. kinectfordevelopers. com/es/2012/11/06/que-es-el-dispositivo-kinect*.

Nájera, L. O. R., Sánchez, M. L., Serna, J. G. G., Tapia, R. P., y Llanes, J. Y. A. (2016). Recognition of mexican sign language through the leap motion controller. En *Proceedings of the international conference on scientific computing (csc)* (p. 147).

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 2018). Determination press San Francisco, CA.

NVIDIA. (2019). *Jetson nano developer kit.* https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

Palomera, D. (2015). *Aprendizaje semi-supervisado para la detección de respuestas informativas en preguntas procedurales* (Tesis Doctoral no publicada). UNIVERSIDAD ANDRÉS BELLO.

Pérez, L. M., Rosales, A. J., Gallegos, F. J., y Barba, A. V. (2017). Lsm static signs recognition using image processing. En *2017 14th international conference on electrical engineering, computing science and automatic control (cce)* (pp. 1–5).

Quirós, J. P. S. (2019). *Convolución matricial aplicado al procesamiento de imágenes.* https://www.tec.ac.cr/.

Ramos, E. B. (2018). La lengua de señas mexicana (lsm) como mediador entre el sordo y la matemática.

Rodríguez, L. (2005). Estudi sobre la identitat cultural a la comunitat sorda. *Revista d'etnologia de Catalunya*, 154–157.

Seif, G. (2018). *an easy introduction to unsupervised learning with 4 basic techniques.* Récupéré sur https://towardsdatascience. com/an-easy-introduction ....

Solís, F., Martínez, D., y Espinoza, O. (2016). Automatic mexican sign language recognition using normalized moments and artificial neural networks. *Engineering*, *8*(10), 733–740.

Suárez, S. T. P. (2015). *Metodologías de diseño de redes neuronales sobre dispositivos digitales programables para procesado de señales en tiempo real* (Tesis Doctoral no publicada). Universidad de Las Palmas de Gran Canaria.

Vidal-Espinoza, R., y Cornejo Valderrama, C. G. (2016). Trabajo y discapacidad: una mirada critica a la inclusion al empleo.

Vilches Vilela, M. (2005). La dactilología,¿ qué, cómo, cuándo. *Universidad de Magisterio "Sagrado Corazón" de Córdoba.[en línea]*.

Voisard, A. (2015). *Día internacional de las lenguas de señas, 23 de septiembre.* https://www.un.org/es/observances/sign-languages-day.

Weichert, F., Bachmann, D., Rudak, B., y Fisseler, D. (2013). Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, *13*(5), 6380–6393.

Wolfe, M. (2021). *Tensorflow vs keras: A comparison.* https://towardsdatascience.com/tensorflow-vs-keras-d51f2d68fdfc.

Wong, A. (2017). *Coding up a neural network classifier from scratch.* https://towardsdatascience.com/coding-up-a-neural-network-classifier-from-scratch-977d235d8a24.

Zhang, H., Liu, S., Chen, H., y Cheng, W. (2019). Implementation of image recognition on embedded systems. En *International conference in communications, signal processing, and systems* (pp. 536–543).

Zhu, C. (2019). *The "less is more" of machine learning.* https://towardsdatascience.com/the-less-is-more-of-machine-learning-1f571c0d4481.

Labor et sapiens libertas

Escrito por Alfredo Estevez
Heróica Ciudad de Huajuapan de León
Oaxaca, México.