TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Nogales

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

DESARROLLO DE UN INTÉRPRETE BÁSICO DEL LENGUAJE DE SEÑAS PARA DACTILOLOGÍA EMPLEANDO INTELIGENCIA ARTIFICIAL

TESIS

QUE PARA OBTENER EL GRADO DE
MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA
PEDRO EZEQUIEL RIVAS PEREA

DIRECTOR
M.C. OMAR VELARDE ANAYA

CO-DIRECTOR
Ph.D. PABLO RIVAS PEREA

H. NOGALES, SONORA, MÉXICO. JUNIO DE 2019.

Dictamen de Impresión

Autorización de impresión

DESARROLLO DE UN INTÉRPRETE BÁSICO DEL LENGUAJE DE SEÑAS PARA DACTILOLOGÍA EMPLEANDO INTELIGENCIA ARTIFICIAL

RESUMEN

En nuestros días ha crecido aceleradamente el desarrollo de la inteligencia artificial. Existen aplicaciones en el área de medicina, biometría, procesos de manufactura y controles de calidad. Estas disciplinas se han visto aventajadas con la implementación de estos sistemas, que llevan a la perfección estos métodos, ocasionando grandes saltos inimaginables en poco tiempo.

El presente estudio muestra una metodología para el desarrollo de un sistema intérprete de lenguaje de señas básico empleando inteligencia artificial, utilizando diferentes algoritmos en MATLAB para el procesamiento de imágenes y desarrollo de código para las capturas de imágenes y evaluación de resultados en C#.

Primeramente se realiza el análisis del algoritmo para el reconocimiento de patrones. Se entrena una red neuronal capaz de reconocer imágenes de lenguaje de señas, posteriormente se ejecuta el mismo análisis pero con imágenes obtenidas con el sensor Kinect y finalmente, se integra el diseño del algoritmo de la red neuronal con el sensor Kinect para formar un sistema de visión artificial.

La consecuencia de aplicar los conocimientos sobre lenguaje de señas, procesamiento de imágenes y visión artificial da como resultado un producto de alta calidad con muy buenos resultados, alcanzando un 98% de mejor rendimiento en el reconocimiento de las señas. Al aplicar los conocimientos adquiridos y estudiar las investigaciones elaboradas, se demuestra que el sistema cumple con los requisitos para la interpretación de lenguaje de señas por medio de sistemas de visión artificial en comparación con otros sistemas sin inteligencia.

ABSTRACT

In our days the artificial intelligence development has grown rapidly. There are applications in the area of medicine, biometrics, manufacturing processes and quality controls. These disciplines have been advantaged with the implementation of these systems, which lead to the perfection of these methods causing great unimaginable leaps in a short time.

The present study shows a methodology to develop a basic interpreter system of sign language with artificial intelligence, using different algorithms in MATLAB for image processing, and code development for image captures and evaluation of results in C #.

First, it is performed the algorithm analysis for patterns recognition. A neural network capable is trained to recognize images of sign language, then the same analysis is executed but with images obtained with the Kinect sensor, and finally, the design of the neural network algorithm is integrated with the Kinect sensor to build a artificial vision system.

The importance to apply the knowledge of sign language, image processing and artificial vision impact in a high quality product with very good results, reaching 98% of best performance in sign recognition. By applying the acquired knowledge and studying scientific researchs, it is demonstrated that the system fulfills the requirements for the interpretation of sign language by means of artificial vision systems in comparison with other systems without intelligence.

AGRADECIMIENTOS

Agradezco especialmente a mis amados padres Sra. Librada Perea Castro y el Sr. Matías Pedro Rivas Bórquez, por sus sabios consejos, desde mis primero pasos hasta el día de hoy. Sus vidas me influenciado grandemente al tomar decisiones importantes, sobre qué carrera estudiar, con quién me voy a casar y la eternidad divina. Sin duda siempre estaré agradecido con Dios por permitirme vivir esta etapa de mi vida.

También agradezco a mi amada esposa Julissa Elizabeth Campos Castro, quien me acompañado en desvelos, enojos, alegrías y bendiciones, me ha motivado a seguir adelante, cuando el camino se ve largo o los días transcurren demasiado rápido, he aprendido a tener paciencia, gracias a su amor incomparable.

Además a mis hermanos Pablo Rivas Perea y Zheila Rivas Perea, que desde que era pequeño han sido de buen ejemplo en los deportes, en lo académico, en lo profesional y sobre todo en lo espiritual. Me han enseñado que la administración del tiempo es de vital importancia para alcanzar nuestros sueños.

Por último, quiero agradecer a los maestros de la Maestría en Sistema Computacionales, que han sido pacientes y atentos a mi desarrollo durante mi estancia, en especial, al M.C. Omar Velarde Anaya por transmitirme la motivación para desarrollar esta tesis y por compartir sus conocimientos profesionales en la dirección de este proyecto.

DEDICATORIA

Quiero dedicar esta tesis a las personas que luchan diariamente en la vida. Espero que este trabajo sea de gran ayuda para mejorar los sistemas de comunicación con las personas de pérdida auditiva

También quiero dedicar este trabajo a mi hijo esperado Mateo Ezequiel Rivas Campos. Todas las palabras escritas en este documento fueron pensadas en él, para que aun pasado los años, mi amor eterno hacia ti sea recordado.

Índice

RESUMEN ..... **I**

ABSTRACT ..... **II**

AGRADECIMIENTOS ..... **III**

DEDICATORIA ..... **IV**

ÍNDICE DE TABLAS ..... **VII**

ÍNDICE DE FIGURAS ..... **VIII**

CAPÍTULO I. INTRODUCCIÓN **1**

1.1 Contenido de los capítulos 1
1.2 Antecedentes 1
1.3 Planteamiento del Problema 2
1.4 Objetivo general 3
1.5 Objetivos específicos 3
1.6 Justificación 3
1.7 Hipótesis 4
1.8 Limitaciones y delimitaciones 4
1.9 Metodología 4
1.10 Estado del Arte 5

CAPÍTULO II. MARCO TEÓRICO **9**

2.1 Las células de la cóclea 9
2.2 Causas de la pérdida de audición y la sordera 10
2.3 Lengua de Señas Mexicana (LSM) 10
2.2.1 Breve historia de la LSM *11*
2.2.2 Dactilología e ideogramas *11*
2.4 Kinect 17
2.3.1 Sensor Kinect 1.0 ..... *18*
2.3.2 Sensor Kinect 2.0 ..... *20*
2.5 Aprendizaje profundo (AP) 22
2.6 Redes Neuronales Artificiales (RNA) 23
2.7 Principales métodos de Aprendizaje Profundo 24
2.8 Autocodificadores (AE) 24
2.7.1 Autocodificadores dispersos (SAE) *26*

CAPÍTULO III. DESARROLLO **28**

3.1 Introducción 28
3.2 Especificación técnica del sensor de profundidad Kinect 28
3.3 Processing 29
3.4 Procedimiento para la instalación de drivers del sensor Kinect en OS X 29
3.5 Ambiente de desarrollo Processing utilizando sensor Kinect 30
3.6 Limitaciones en ambiente Processing 32
3.7 *Kinect for Windows Software Development Kit* (SDK) 2.0 33

3.7.1 Instalación de controlador del sensor Kinect en MS Windows *33*
3.7.2 Verificación de la instalación del controlador *34*
3.8 COMPONENTES DEL SENSOR KINECT 36
3.8.1 Esqueleto humano *36*
3.8.2 Emisor infrarrojo (IR) *37*
3.8.3 Cámara de profundidad *37*
3.9 PATRONES DE RECONOCIMIENTO 39
3.10 *ROI* 41
3.11 CAPTURA DE IMÁGENES PARA ENTRENAMIENTO DE LA RED AE 41
3.12 CONJUNTO DE DATOS DE ENTRADA 42
3.13 ENTRENAMIENTO DE LA RED NEURONAL AE 43
3.8.4 Definir parámetros de entrada *43*
3.8.5 Entrenamiento de primer AE *45*
3.8.6 Entrenamiento del segundo AE *47*
3.8.7 Entrenamiento de red Softmax *48*
3.8.8 Apilar las capas AEs y Softmax *49*
3.8.9 Fine tuning con el conjunto de entrenamiento *49*
3.8.10 Generar métricos de entrenamiento *50*
3.14 AGREGAR DLL DE MATLAB EN VISUAL STUDIO 51
3.15 IMPLEMENTACIÓN DE LA INTERFAZ INTÉRPRETE 51
3.16 DISEÑÓ DEL ALGORITMO (DIAGRAMA DE FLUJO) 53
3.17 ARQUITECTURA DEL SISTEMA 53

CAPÍTULO IV. EVALUACIÓN DE RESULTADOS Y CONCLUSIONES 55

4.1 EVALUACIÓN DE RESULTADOS 55
4.1.1 Resultados del entrenamiento, prueba y validación cruzada de la red AE con imágenes del sensor Kinect *55*
4.1.2 Resultados del entrenamiento de la red AE con imágenes de la cámara Creative y sensor Kinect *58*
4.1.3 Resultados de la validación del sistema *60*
4.2 CONCLUSIONES 65
4.3 MEJORAS Y TRABAJOS FUTUROS 66

ANEXOS 67

CODIGO C# 67
CODIGO MATLAB 76

REFERENCIAS 82

Índice de Tablas

TABLA 2.1. COMPARACIÓN DE LAS PRINCIPALES CARACTERÍSTICAS ENTRE KINECT 1.0 Y KINECT 2.0 21
TABLA 4.1. MÉTRICOS DE LOS RESULTADOS DEL ENTRENAMIENTO DEL AE 55
TABLA 4.2. MÉTRICOS DE DESEMPEÑO DE ENTRENAMIENTO CON IMÁGENES DE CÁMARA *CREATIVE* Y SENSOR *KINECT* 58
TABLA 4.3. TABLA DEL DISEÑO DE EXPERIMENTO 61
TABLA 4.4. TABLA DE RESULTADO DEL EXPERIMENTO 62

Índice de Figuras

FIGURA 2.1. REPRESENTACIÓN SISTEMA AUDITIVO. 9
FIGURA 2.2. FAMILIA FRANCESA DE LENGUA DE SEÑAS. ADAPTADA DE STOKOE (1974) POR SMITH-STARK(1990). LAS LINEAS PUNTEADAS INDICAN LA INCLUSIÓN DE LAS LENGUAS DE SEÑAS DE BRASIL Y MÉXICO 11
FIGURA 2.3. REPRESENTACIÓN DACTILOLÓGICA DE LA PALABRA MAMÁ 12
FIGURA 2.4. REPRESENTACIÓN IDEOGRAMA 12
FIGURA 2.5. ABECEDARIO LSM DE LA A – F 13
FIGURA 2.6. ABECEDARIO LSM DE LA G – L 14
FIGURA 2.7. ABECEDARIO LSM DE LA M – Q 15
FIGURA 2.8. ABECEDARIO LSM DE LA R – W 16
FIGURA 2.9. ABECEDARIO LSM DE LA X – Z 17
FIGURA 2.10. SENSOR MICROSOFT KINECT. (A) EL SENSOR KINECT PARA XBOX 360. (B) EL PROYECTOR INFRARROJO (IR), CÁMARA IR Y DENTRO DEL SENSOR KINECT UNA CÁMARA RGB 18
FIGURA 2.11. LOS PUNTOS INFRARROJOS IR VISTOS POR LA CAMARA IR. LA IMAGEN EN LA IZQUIERDA MUESTRA UN ACERCAMIENTO DEL AREA CON PUNTOS ROJOS 19
FIGURA 2.12. LOCALIZACIÓN DE COMPONENTES, KINECT 2.0 20
FIGURA 2.13. IMAGEN RGB DE KINECT 1.0 (A) COMPARADO CON LA IMAGEN RGB DE KINECT 2.0; LAS DOS IMÁGENES HAN SIDO CAPTURADAS DEL MISMO PUNTO 20
FIGURA 2.14. FOV DE CÁMARAS RGB E IR DE KINECT 2.0. EN AZUL ESTÁ REPRESENTADO EL FOV DE LA CÁMARA IR, MIENTRAS QUE EN VERDE ESTÁ REPRESENTADO EL FOV DE LA CÁMARA RGB 21
FIGURA 2.15 NEURONA ARTIFICIAL 23
FIGURA 2.16. ESQUEMA DE AGRUPACIÓN DE LOS MÉTODOS DE APRENDIZAJE PROFUNDO 24
FIGURA 2.17. LA ESTRUCTURA GENERAL DE UN AUTOENCODER, MAPEANDO UNA ENTRADA X A UN SALIDA (LLAMADO RECONSTRUCCIÓN) R A TRAVÉS DE UNA REPRESENTACIÓN INTERNA O CODIFICADO H. EL AE TIENE DOS COMPONENTES: EL CODIFICADOR F (MAPEO DE X A H) Y EL DECODIFICADOR G (MAPEO DE H A R). 25
FIGURA 2.18. REPRESENTACIÓN DE LOS PESOS DE CADA NEURONA 26
FIGURA 3.1. (A) DISTANCIA EN RANGO DE OPERACIÓN, (B) ANGULO DE VISIÓN 28
FIGURA 3.2. DETALLES DE LA INSTALACIÓN DE LOS DRIVERS 30
FIGURA 3.3. CÓDIGO EN PROCESSING 31
FIGURA 3.4. IMÁGENES: RGB, PROFUNDIDAD, IR Y COMBINADO 31
FIGURA 3.5. IMAGEN DE PROFUNDIDAD 32
FIGURA 3.6. IMAGEN DE PROFUNDIDAD, EN CIERTO RANGO DE DISTANCIA 33
FIGURA 3.7 SERVICIO DE "KINECTSERVICE" EN EJECUCIÓN 34
FIGURA 3.8. LEDS ROJOS ILUMINADOS EN SENSOR KINECT 35
FIGURA 3.9. CONTROLADOR DEL KINECT IDENTIFICADO EN EL ADMINISTRADOR DE DISPOSITIVOS 36
FIGURA 3.10. CAPTURA DE IMAGEN DE PROFUNDIDAD, RESALTANDO ESQUELETO DEL CUERPO 37
FIGURA 3.11. CAPTURA DE IMAGEN INFRARROJA 37
FIGURA 3.12. PATRON DE PUNTOS INFRARROJOS EMITIDOS POR SENSOR KINECT 38
FIGURA 3.13. CAPTURA DE IMAGEN DE PROFUNDIDAD 38
FIGURA 3.14. CÓDIGO DRAWHAND() 39
FIGURA 3.15. CÓDIGO PARA IDENTIFICAR LA ARTICULACIÓN MANO DERECHA E IZQUIERDA 39
FIGURA 3.16. CÓDIGO, SECCIÓN QUE IDENTIFICA LA ARTICULACIÓN DEL CUERPO 40
FIGURA 3.17. CAPTURAS KINECT, REPRESENTACIÓN DE SEÑAS DEL NÚMERO 1 AL 5 40
FIGURA 3.18. DISEÑO DE *ROI* EN C# CON SENSOR *KINECT* 41
FIGURA 3.19. SECCIÓN DE CÓDIGO PARA CREAR IMAGEN PNG EN C# CON SENSOR KINECT 42
FIGURA 3.20. EJEMPLOS DE IMÁGENES CAPTURADAS CON EL SENSOR KINECT, REPRESENTANDO 10 SEÑAS DEL LSM 42

FIGURA 3.21. DIVISIÓN CONJUNTO DE DATOS DE ENTRADA 43
FIGURA 3.22. REPRESENTACIÓN GRÁFICA DE LA MATRIZ DE ENTRADA DE LA PARTE CODIFICADORA DEL AE (SOLO 20 ELEMENTOS) 44
FIGURA 3.23. VARIABLES DE ENTRENAMIENTO 44
FIGURA 3.24. MATRIZ DE LA VARIABLE *Y_TARGET* 45
FIGURA 3.25. ENTRENAMIENTO DEL PRIMER AE 46
FIGURA 3.26. REPRESENTACIÓN GRÁFICA DE LA MATRIZ DE LA RECONSTRUCCIÓN DE LA PARTE DECODIFICADORA DEL AE 46
FIGURA 3.27. REPRESENTACIÓN DE LOS PESOS (W) DEL PRIMER AE 47
FIGURA 3.28. ENTRENAMIENTO DEL SEGUNDO AE 47
FIGURA 3.29. REPRESENTACIÓN DE LOS PESOS (W) DEL SEGUNDO AE 48
FIGURA 3.30. ENTRENAMIENTO DE LA CAPA SOFTMAX 48
FIGURA 3.31. ENSAMBLE DE RED PROFUNDA 49
FIGURA 3.32. MODELO DE ENTRENAMIENTO DEL AE 49
FIGURA 3.33. MÉTODO *TRAIN()* PARA REALIZAR EL AJUSTE FINO (*FINE-TUNING)* 50
FIGURA 3.34. INSTRUCCIONES PARA OBTENER MÉTRICOS 50
FIGURA 3.35. REFERENCIA DE MATLAB AGREGADO EN EL PROYECTO DE C# 51
FIGURA 3.36. SECCIÓN DE CÓDIGO EN C# QUE INTERPRETA LA IMAGEN DE ENTRADA COMO SEÑA 52
FIGURA 3.37. IMAGEN RECONOCIDA POR EL SISTEMA INTERPRETE COMO LA LETRA “L” 52
FIGURA 3.38. ETAPAS DEL SISTEMA INTÉRPRETE EMPLEANDO INTELIGENCIA ARTIFICIAL 53
FIGURA 3.39. ARQUITECTURA DEL SISTEMA INTÉRPRETE 54
FIGURA 4.1. COMPARACIÓN DE TASA DE ERROR EN ENTRENAMIENTO, PRUEBA Y VALIDACIÓN CRUZADA 56
FIGURA 4.2. COMPARACIÓN DE EXACTITUD EN ENTRENAMIENTO, PRUEBA Y VALIDACIÓN CRUZADA 57
FIGURA 4.3 COMPARACIÓN DE TASA DE ERROR EN VALIDACIÓN CRUZADA, *CREATIVE VS KINECT* 58
FIGURA 4.4. COMPARACIÓN DE EXACTITUD EN ENTRENAMIENTO, *CREATIVE VS KINECT* 59
FIGURA 4.5. COMPARACIÓN DE TASA DE ERROR BALANCEADO EN ENTRENAMIENTO, *CREATIVE VS KINECT* 59
FIGURA 4.6. COMPARACIÓN DE PRECISIÓN Y EXHAUSTIVIDAD EN ENTRENAMIENTO, *CREATIVE VS KINECT* 60
FIGURA 4.7. RESULTADOS GENERALES DEL SISTEMA CLASIFICADOR 62
FIGURA 4.8. RESULTADO DE EXACTITUD POR CADA LETRA 63
FIGURA 4.9. TOTAL DE FALSOS POSITIVOS Y FALSOS NEGATIVOS POR LETRA 63
FIGURA 4.10. RESULTADOS DEL SISTEMA DE RECONOCIMIENTO DE LA LETRA “I” 64
FIGURA 4.11. RENDIMIENTO DEL SISTEMA POR SEÑA 64

CAPÍTULO I. INTRODUCCIÓN

1.1 Contenido de los capítulos

En el **CAPÍTULO I. INTRODUCCIÓN**, aborda la información que sustenta el objetivo de estudio, que incluye un enfoque general sobre la importancia de utilizar sistemas de inteligencia artificial para la inclusión de personas con discapacidad auditiva.

El **CAPÍTULO II. MARCO TEÓRICO**, está centrado en indagar las bases, definiciones y términos que sustentan la investigación. Se fundamenta a profundidad en cada una de las áreas que conforman el objeto de estudio, tomando como referente el lenguaje de señas, redes neuronales artificiales, autocodificadores y sensores de imagen de profundidad.

En el **CAPÍTULO III. DESARROLLO**, se unen los capítulos previamente analizados, donde se utiliza la metodología para cumplir con el objetivo propuesto al inicio de la investigación, y que lleva al desarrollo del intérprete requerido para identificar señas, para poder establecer la base de una comunicación entre dos individuos.

En el **CAPÍTULO IV. EVALUACIÓN DE RESULTADOS Y CONCLUSIONES**, se presenta un análisis de los resultados obtenidos a través del sistema intérprete implementado, que incluye las aportaciones realizadas, las conclusiones que se formulan al término de la investigación, así como trabajos futuros a partir del presente proyecto.

1.2 Antecedentes.

La vista es una sentido vital en la organismo humano, que interactúa constantemente con el cerebro humano, enviando demasiada información para ser procesada. Tiene la capacidad de interpretar el ambiente por medio de la luz que entra a los ojos. Una interpretación puede ser un objeto a cierta distancia, esto requiere el aprendizaje del objeto y la adaptación de la retina para realizar el enfoque.

Con la evolución de la tecnología se han desarrollado técnicas que emulan la visión del ser humano, esto es la inteligencia artificial. Donde la combinación de hardware y software con inteligencia constituyen la visión artificial. Este sistema consiste en la

captación de imágenes, aprendizaje de la información, procesamiento e interpretación de resultados (Sotolongo & Guzman, 2009).

La visión artificial es una fuerte herramienta con diferentes aplicaciones comúnmente utilizada para diagnóstico médico, sistemas de inspección en la industria, seguridad privada, apoyo didáctico, entre otros.

Las lenguas de señas son lenguas naturales de producción gestual y percepción visual que tienen estructuras gramaticales perfectamente definidas y distintas de las lenguas orales con las que coexisten (Tovar L., 1999). Gracias a este método las personas sordas pueden llegar a establecer una vía de comunicación en la sociedad, ya sea conformado por otros sordos o personas que conozcan el lenguaje. Mientras que con el lenguaje oral la comunicación se establece en un canal vocal-auditivo, la lengua de señas lo hace por un canal gesto-viso-espacial.

Según datos de la investigación (Humphries, et al., 2012) realizada a nivel global en el año 2012, la pérdida de audición neurosensorial es más común en el nacimiento, que ocurre en 2 a 3 de cada 1000 nacimientos en países desarrollados y mucho más altos en países subdesarrollados, tal como en Nigeria, donde se encuentran 28 por cada 1000. En las causas pos-natales de pérdida auditiva neurosensorial aumenta la cifra, para la edad escolar, en 6 a 7 de 1000 niños tienen pérdida auditiva permanente. El número de niños sordos que son afectados es bastante grande y de especial cuidado.

1.3 Planteamiento del Problema

Actualmente, las personas con discapacidad auditiva tienen dificultades o les es imposible desarrollar una comunicación a nivel básico con personas que desconocen el lenguaje de señas, lo cual les dificulta su inclusión en la sociedad. Para esto, se han desarrollado diferentes sistemas; por decir, el uso de una mano electromagnética conectada a una máquina de escribir (Laenger, McFarland) o el uso de guantes con galgas de presión para interpretar los movimientos de los dedos (Kramer, Lindener & George) no han podido interpretar de manera eficiente las señas. En el caso del guante es poco ergonómico, ya que para interpretar una palabra o letras requiere cierta

flexibilidad en el movimiento de los dedos. Por otro lado los sistemas de visión requieren de cierta iluminación controlada y fondos de pantalla, construyendo un sistema complejo y sensible a perturbaciones. En estos sistemas, la iluminación es un factor importante para la captura de imágenes, si el sistema no se tiene encapsulado, es necesario realizar ajustes cada vez que el sistema sea colocado en diferentes superficies.

1.4 Objetivo general

Desarrollar un intérprete básico de lenguaje a señas Mexicana para dactilología empleando inteligencia artificial en un sistema de visión mediante el uso del sensor Kinect 2.0 y redes neuronales autoencoders, el cual sea la base para implementar un sistema interprete de la Lengua de Señas Mexicana (LSM), que pueda ser utilizado de una manera más natural.

1.5 Objetivos específicos

- Conocer el alfabeto dactilológico de la lengua de señas mexicana.
- Crear un conjunto de datos de entrenamiento para la red neuronal.
- Seleccionar una cámara capaz de extraer imágenes con características del lenguaje de señas para dactilología.
- Validar el algoritmo con inteligencia artificial para el reconocimiento de imágenes.
- Desarrollar una interfaz gráfica que muestre los resultados en tiempo real.
- Verificar la funcionalidad y compatibilidad de la cámara con el sistema operativo.
- Integrar hardware y software.

1.6 Justificación

El desarrollo de este proyecto es importante, ya que permitirá en un futuro contribuir a que las personas con discapacidad auditiva puedan tener una comunicación básica con individuos que no cuentan con algún conocimiento sobre el lenguaje de señas, en un ambiente menos controlado (sin conexión de guantes e iluminación).

Igualmente este proyecto utiliza una cámara novedosa de una consola de juegos paro lo que fue creada, pero después de conocer sus versatilidad, funcionalidad y desempeño ha sido utilizada por desarrolladores e investigadores.

Otro punto a destacar, es la implementación de la inteligencia artificial en el sistema de visión. El tipo de redes neuronales *autoencoders* es una arquitectura basada en simplificar una gran cantidad de datos (como lo son las imágenes) para que su entrenamiento sea de manera eficiente y con un desempeño óptimo.

1.7 Hipótesis

El empleo de inteligencia artificial en un traductor de lenguaje de señas basado en un sistema de visión, contribuirá al desarrollo de un sistema más robusto y menos sensible a perturbaciones en un 90%, lo cual contribuirá a desarrollar un sistema que permita la comunicación a nivel básico con personas que desconocen el lenguaje de señas.

1.8 Limitaciones y delimitaciones

Este proyecto está limitado a la detección de imágenes de la lengua de señas mexicana para dactilología, las cuales se obtienen de una cámara de profundidad

El sistema de visión solo interpretará las siguientes señas del alfabeto dactilológico mexicano: A, B, C, D, E, F, G, H, I y L

El sistema de visión no interpretará las siguientes señas del alfabeto dactilológico mexicano: J, K, M, N, O. P, Q, R, S, T, U, V, X, Y y Z.

De igual manera, no será capaz de interpretar ideogramas, los cuales se realizan con los movimientos de las manos y gestos de la cara

1.9 Metodología

En el presente documento se pretende utilizar una cámara capaz de mostrar imágenes de profundidad que funcione dentro de un ambiente de luz no controlado, que por su funcionalidad y capacidad pueda ser utilizado para el desarrollo de *software*.

De igual manera se busca utilizar un *software* que pueda realizar capturas de las imágenes de profundidad, también, es importante definir una región de interés que será el área de trabajo. Asimismo debe ser competente para mostrar los resultados esperados en la interfaz de usuario.

Es de vital importancia realizar el diseño del algoritmo para el procesamiento de imágenes de profundidad dentro de una red neuronal artificial y evaluar el desempeño de esta misma con un conjunto de imágenes similar a las capturas obtenidas con la cámara de profundidad.

1.10 Estado del Arte

En Latinoamérica, la oferta de herramientas tecnológicas para la sociedad es limitada, además y sobre todo costosas, lo que resulta más difícil para la población sorda mejorar su nivel de inclusión social, a través del impacto del desarrollo en el ámbito laboral y académico, generando una brecha muy notable que no permite a los servicios públicos y programas de apoyo para discapacidad, tener importantes avances en esta materia (Alonso, A. & Díaz, E, 2008).

Hoy en día, se han realizado varios proyectos de investigación que tienen como finalidad dar solución a la problemática descrita a través de las tecnologías de la información y la comunicación (TIC) (Hernández, M.R, Rodríguez, V.M, Parra, F.J., & Vel, 2014). Uno de los primeros proyectos, en el año de 1976, fue el desarrollo de la patente US4074444 por Laenger (Laenger, McFarland, R., & Peel, 1976) consiste en una mano electromagnética con interfaz de teclado, la cual respondía el signo correspondiente a la letra digitada por el usuario, lo que limita a tener en una sola posición la mano y depender de un sistema mecánico que en poco tiempo se deteriora. Otro interesante trabajo, en el año de1988, fue la patente US5047952 elaborada por Kramer, Lindener y George (Kramer, Lindener, & George, 1988), la cual usa un sistema compuesto por un guante que interpreta los movimientos de los dedos por medio de galgas de presión colocadas en las articulaciones, las cuales al detectar una variación en la presión de estas, transmite los datos a un computador, el cual mediante un algoritmo de reconocimiento de patrones interpreta las señales y muestra la información

en pantalla. Algunas desventajas de utilizar galgas extensiométricas es que la señal emitida es baja, no lo suficiente buena por lo que requiere un circuito amplificador. Ademas es sensible a vibraciones por lo que se requiere un soporte aceptable.

La patente 4378215 de Sparks (Sparks, 1981), desarrollado en el año de 1981, presenta un enfoque hacia el aprendizaje del alfabeto manual, consiste en el manejo de figuras de manos, cada una de ellas muestra las señas del alfabeto manual para sordos. Por otra parte, en el año 1980, la patente US4368459 de Sapora (Sapora, 1980), muestra un dispositivo de señalización para administrar via visual a estudiantes con discapacidad auditiva, en un ambiente de clases. Los mensajes del maestro son presentados como imágenes o como lenguaje de señas. Finalmente, en el año de 1987, para la enseñanza a niños con discapacidad auditiva la patente US4799889 de Jockey (Jockey, 1989), utiliza un oso de peluche que tiene brazos tubulares huecos por los cuales el maestro puede pasar sus brazos por los tubos del oso de peluche, de esta forma el profesor puede realizar el lenguaje de señas utilizando un objeto familiar conocido por los niños. El enfoque es bueno porque da la solución para el aprendizaje por medio de la lengua de señas, pero carece de un sistema autónomo que puede mantenerse por bastantes años.

Para el año 2006, se presenta el desarrollo y aplicación de un guante instrumentado (Leybon Ibarra, Ramírez Barba, & Taboada Picazo, 2006) con dispositivos foto-eléctricos para diferenciar cuatro posiciones durante el movimiento flexor de los dedos de las manos, distinguiendo los queiremas[1] que un signante realiza al comunicarse por medio de la lengua de señas mexicana (LSM). Los dispositivos desarrollados solo pueden detectar el doblez de los dedos, pero no su movimiento aproximación/separación, por lo que es necesario incluir otro tipo de dispositivos que permitan perfeccionar esta característica de los movimientos de la mano.

Oikonomidis (Oikonomidis, Kyriazis, Padeleris, & Argyros, 2011) presenta una solución novedosa en el año de 2011, al problema de la recuperación y seguimiento de la posición 3D, la orientación y la articulación completa de una mano humana a partir de

[1] *Queirema. Forma o configuración de la mano.*

observaciones visuales sin marcas obtenidos por un sensor Kinect. Se trata de un problema de optimización, que busca los parámetros del modelo de mano que reduzcan al mínimo la discrepancia entre la apariencia y la estructura 3D de casos hipotéticos de un modelo de mano y las observaciones de la mano real. El problema de optimización se resuelve usando una variante mejorada por enjambre de partículas (*PSO*) (Clerc, 2010). Los resultados experimentales extensivos demuestran que el seguimiento de mano 3D preciso y robusto es alcanzable en 15 Hz. Por lo tanto, se demuestra que basado en el modelo de seguimiento de articulación en la evidencia es casi factible en tiempo real. Da como resultado un método robusto y eficiente para el seguimiento de la postura completa de una mano en la compleja articulación. Este sistema podría ser servir como base para el reconocimiento de lenguaje de señas ya que puede rastrear la mano en tiempo real con el sensor Kinect

En el mismo año, Díaz desarrolla una investigación sobre Kinect. Este se encuentra en una plataforma móvil, el vehículo se está moviendo en una superficie verde, buscando una bola roja. El Kinect capta las imágenes RGB y los de profundidad que se envían a la tableta mediante un puerto USB. Los resultados obtenidos demuestran que no es completamente eficiente la navegación del robot móvil usando solo visión estéreo ya que las condiciones del entorno (luminosidad, características de color del objeto, entre otros) hacen que el algoritmo de navegación deba ser ajustado constantemente y es por eso que decide iniciar el estudio de procesamiento digital de imágenes (PDI) (Diaz Celis & Romero Molano, 2012). Este sistema tiene como deficiencia la inteligencia artificial, ya que el robot móvil controla sus movimientos en base a cierta cantidad de pixeles en la imagen capturada, por lo que no realiza un entrenamiento con imágenes como se espera que funcione para una red neuronal.

Para el 2015 como paso inicial, se desarrolló un sistema automático de reconocimiento de letras realizado con la mano (*fingerspelling*) utilizando redes neuronales convolucionales (*CNNs*) y mapas de profundidad. El mejor resultado alcanzado, con diferentes configuraciones de aprendizaje, con o sin validación, es de 99.99% de exactitud para los interpretes observados y 83.58% a 85.49% de exactitud para los nuevos intérpretes. La captura de estas imágenes se realizó con la cámara

Creative Senz3D. Tiene una resolución de 320 x 240 pixeles (Kang, Tripathi, & Nguyen, 2015). Esta investigación ha sido fundamental para el desarrollo de este proyecto, ya que cuenta con las características de imagen y resultados de entrenamiento, una de las principales desventajas es que no se encuentra validada su implementación como un sistema de visión y solo se hace un análisis de las imágenes.

Actualmente se han desarrollado prototipos para el reconocimiento de señas utilizando la dactilología por medio de un guante con sensores flexibles (Álvarez Torres, Vázquez Cuevas, & Velarda Anaya, 2017). El sistema ha sido diseñado en base a sensores resistivos flexibles y un microcontrolador *Arduino,* los cuales interpretan las señales resistivas cuando el guante flexiona algún dedo (Escamilla Bouchan & Luna Soto, 2018). Se obtuvieron resultados buenos respecto al reconocimiento de lenguaje de señas mexicana, pero depende de sensores flexibles lo que ocasionaría que si deja de funcionar tan solo uno, la lectura y la variación de resultados serían afectadas, ya que tiene un algoritmo estadístico.

CAPÍTULO II. MARCO TEÓRICO

En el presente capítulo se abordarán varios temas. Primero se explica las partes y funciones de los componentes del oído, así como, las causas de pérdida de audición. Posteriormente se hace referencia a la lengua de señas mexicana; su historia y sus componentes (ideogramas y dactilología). Después se describe la funcionalidad y características del sensor Kinect. Y por último temas relacionados a inteligencia artificial, tales como: Redes neuronales, aprendizaje profundo y *autoencoders*.

2.1 Las células de la cóclea

El caracol o cóclea (Pedemonte & Narins, 1999) es un órgano del oído interno con forma de tubo enrollado en espiral, constituido por un laberinto óseo tapizado de estructuras celulares que forman un laberinto membranoso (Figura 2.1)

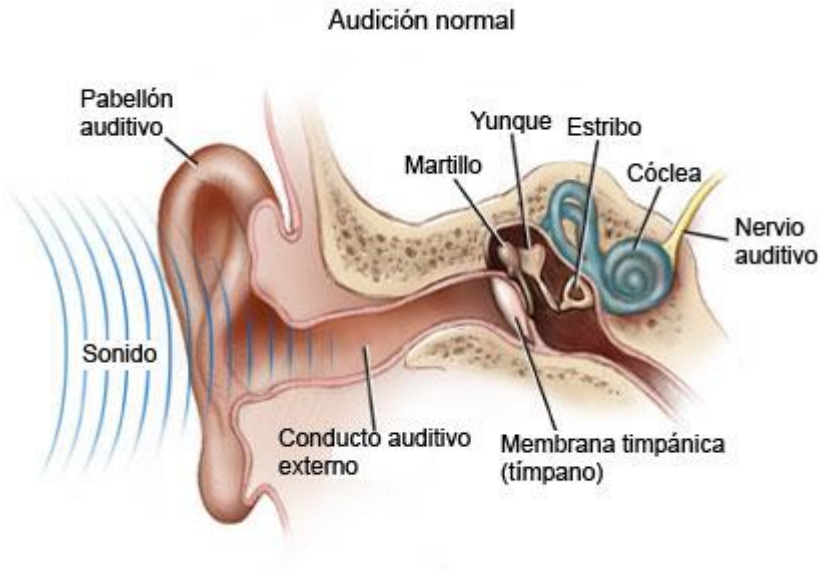


Figura 2.1. Representación Sistema Auditivo.

El eje central de la cóclea, el modiolo, es un tubo óseo dentro del cual corren las fibras del nervio auditivo. La espiral ósea enrollada alrededor del modiolo está separada en 3 compartimentos: la escala vestibular, la escala media y la escala timpánica.

La escala timpánica se separa de las otras dos por la membrana basilar; la membrana de *Reissner*, compuesta por dos capas celulares, separa la escala media de la vestibular. Las escalas timpánicas y vestibular -rellenas de *perilinfa*- se comunican en el vértice de la cóclea por un orificio, el *helicotrema*. En la base de la cóclea la escala timpánica termina en la ventana redonda. Un pequeño tubo, el acueducto coclear, conecta la *perilinfa* con el líquido cefalorraquídeo. La escala vestibular, en la base de la cóclea, se corresponde con la ventana oval en el oído medio donde se articula el estribo.

Las escalas media y vestibular se continúan en la base de la cóclea con la porción vestibular del oído interno. La escala media conecta con el sáculo a través del *ductus reuniens* y la endolinfa contenida en ella está en comunicación con el saco endolinfático

Estos receptores son sensibles a sonidos dentro de una gama de frecuencias entre unas decenas de Hertz (Hz) hasta unos 20 kHz en el humano joven.

2.2 Causas de la pérdida de audición y la sordera

Las causas de pérdida de audición y sordera pueden ser congénitas o adquiridas. Las causas congénitas se determinan en el nacimiento o poco después. También puede ser relacionado a elementos hereditarios o no hereditarios, o complicaciones durante el embarazo o parto (Sordera y pérdida de la audición, 2019).

Las causas adquiridas como su nombre lo dice, son obtenidas a cualquier edad. Existen algunas causas relacionadas a enfermedades como; la meningitis, el sarampión y la parotiditis, y otras relacionadas a eventos como; infección crónica en el oído, presencia de líquido en el oído y la exposición a sonidos muy elevados.

2.3 Lengua de Señas Mexicana (LSM)

La historia argumentada de las lenguas de señas (Aldrete, 2009) en el continente americano comienza con la emigración de maestros sordos europeos hacia América y con ellos las técnicas para la enseñanza de las personas sordas a través de la lengua de señas. Esto implicaría que el comienzo de las lenguas de señas en América comienza en Europa a partir de los métodos de enseñanza para el sordo, que incluían el uso de señas, gestos, y dactilología. Se empleaba esta metodología con el propósito de que los

sordos aprendieran la gramática de la lengua dominante, así como la posibilidad de que se oralizarán en la mayoría de los casos.

Esta forma de instrucción a través del uso de señas, signos metódicos, gestos, alfabeto manual, etcétera, fue fundamental en la formación de las lenguas de señas europeas, y por ello se puede suponer que también de las lenguas de señas de América.

2.2.1 Breve historia de la LSM

Al abordar la genealogía de la LSM se parte de reconocer la fundación de la Escuela Nacional de Sordomudos en 1867 en México. Se retoma la idea del trabajo realizado por Smith-Stark (1990) para el establecimiento de las relaciones de la LSM con la familia francesa de lenguas de señas. En la Figura 2.2, se muestra el esquema presentado por este autor.

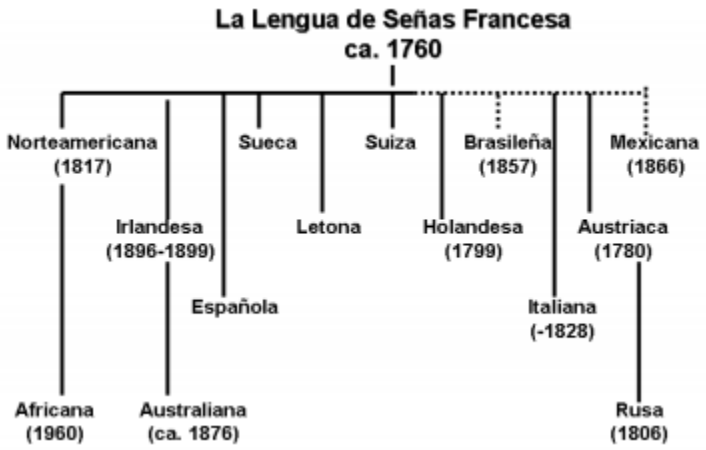


Figura 2.2. Familia francesa de lengua de señas. Adaptada de Stokoe (1974) por Smith-Stark(1990). Las lineas punteadas indican la inclusión de las lenguas de señas de Brasil y México

2.2.2 Dactilología e ideogramas

La Lengua de Señas Mexicana (LSM) está compuesta de la dactilología y los ideogramas (Serafín de Fleischmann & González Pérez, 2011). Se conoce como dactilología a lo que bien podría ser el deletreo en la lengua oral, y está representada en este diccionario sobre todo con el abecedario (Figura 2.5 a 2.9). Cada palabra se puede representar con la articulación de mano correspondiente de cada letra que la conforma.

De acuerdo con esto, “mamá” puede representarse con cada una de sus letras, como se muestra en la Figura 2.3.

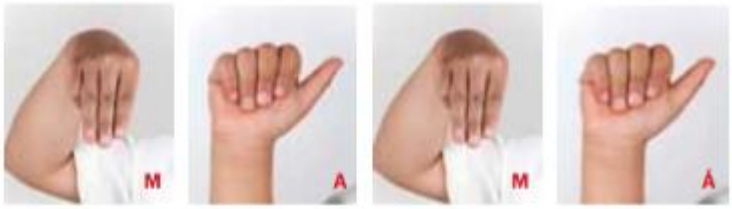
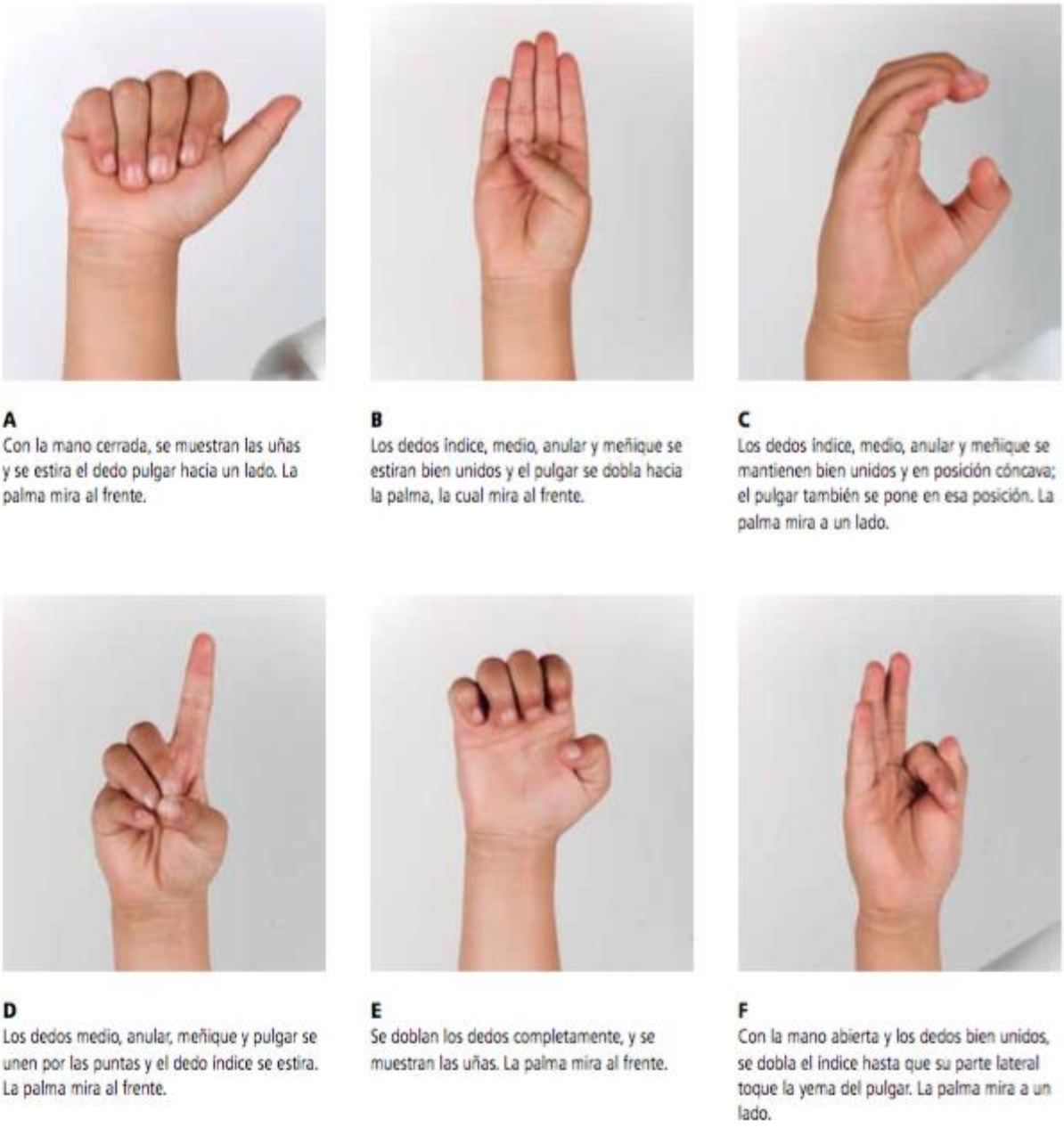


Figura 2.3. Representación dactilológica de la palabra mamá

Los ideogramas representan una palabra con una o varias configuraciones de mano. En este caso, “mamá” se articula con la letra “m” de ideograma sobre los labios, con la que se golpean varias veces los labios. (Figura 2.4)

Figura 2.4. Representación ideograma

Hay distintas metodologías para la comunicación con las personas sordas. La más básica y sencilla es seguir el orden sintáctico del español hablado al realizar las señas de las palabras para formar las expresiones que se deseen comunicar.



A
Con la mano cerrada, se muestran las uñas y se estira el dedo pulgar hacia un lado. La palma mira al frente.

B
Los dedos índice, medio, anular y meñique se estiran bien unidos y el pulgar se dobla hacia la palma, la cual mira al frente.

C
Los dedos índice, medio, anular y meñique se mantienen bien unidos y en posición cóncava; el pulgar también se pone en esa posición. La palma mira a un lado.

D
Los dedos medio, anular, meñique y pulgar se unen por las puntas y el dedo índice se estira. La palma mira al frente.

E
Se doblan los dedos completamente, y se muestran las uñas. La palma mira al frente.

F
Con la mano abierta y los dedos bien unidos, se dobla el índice hasta que su parte lateral toque la yema del pulgar. La palma mira a un lado.

Figura 2.5. Abecedario LSM de la A – F

G
Se cierra la mano y los dedos índice y pulgar se estiran. La palma mira hacia usted.

H
Con la mano cerrada y los dedos índice y medio bien estirados y unidos, se extiende el dedo pulgar señalando hacia arriba. La palma mira hacia usted.

I
Con la mano cerrada, el dedo meñique se estira señalando hacia arriba. La palma se pone de lado.

J
Con la mano cerrada, el dedo meñique bien estirado señalando hacia arriba y la palma a un lado dibuja una *j* en el aire.

K
Se cierra la mano con los dedos índice, medio y pulgar estirados. La yema del pulgar se pone entre el índice y el medio. Se mueve la muñeca hacia arriba.

L
Con la mano cerrada y los dedos índice y pulgar estirados, se forma una l. La palma mira al frente.

Figura 2.6. Abecedario LSM de la G – L

M
Con la mano cerrada, se ponen los dedos índice, medio y anular sobre el pulgar.

N
Con la mano cerrada, se ponen los dedos índice y medio sobre el pulgar.

Ñ
Con la mano cerrada, se ponen los dedos índice y medio sobre el pulgar. Se mueve la muñeca a los lados.

O
Con la mano se forma una letra o. Todos los dedos se tocan por las puntas.

P
Con la mano cerrada y los dedos índice, medio y pulgar estirados, se pone la yema del pulgar entre el índice y el medio.

Q
Con la mano cerrada, se ponen los dedos índice y pulgar en posición de garra. La palma mira hacia abajo, y se mueve la muñeca hacia los lados.

Figura 2.7. Abecedario LSM de la M – Q

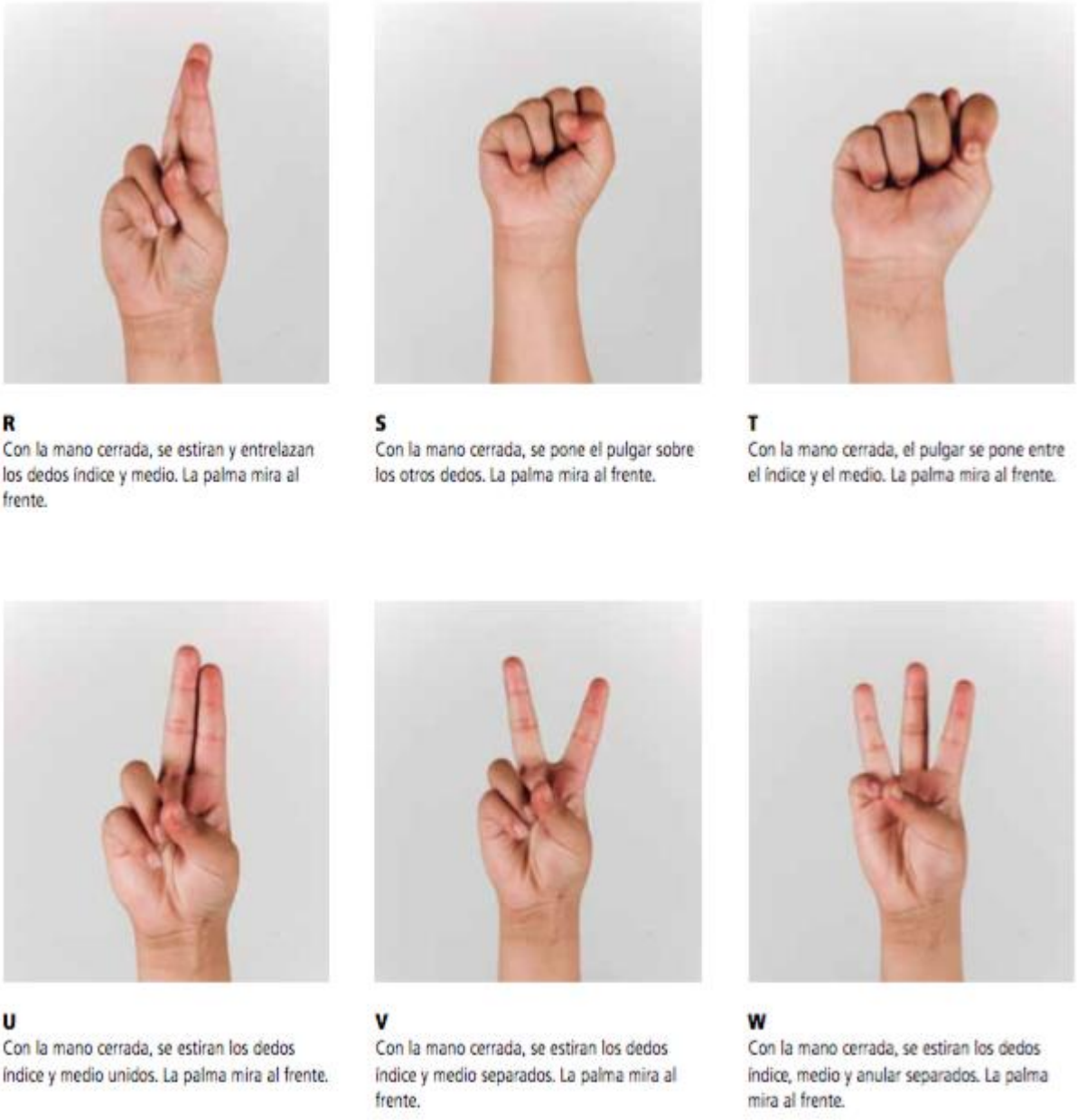


Figura 2.8. Abecedario LSM de la R – W

Figura 2.9. Abecedario LSM de la X – Z

2.4 Kinect

El Kinect fue lanzado el 4 de Noviembre del 2010 como un accesorio para la consola Xbox 360. Fue desarrollado por Microsoft y la compañía Israelí PrimeSense e introducida al Record Mundial Guiness como el más rápido dispositivo del consumidor vendido, con 8 millones de unidades vendidas durante los primeros 60 días en el mercado.

El Kinect permite nuevas interacciones durante los juegos, basado en el uso de gestos y voz. Desde su presentación ha sido atractivo para los investigadores de diferentes campos, desde robótica (El-Iaithy, Jidong, & Yeh, 2012) hasta la ingeniería biomédica (Alnowami, Alnwaimi, Tahavori, Copland, & Wells, 2012) y los sistemas de visión (Schindhelm, 2012). Tempranamente después del lanzamiento de Kinect, el dispositivo fue hackeado y creado un Kit para Desarrollo de Software (SDK) por comunidades terceras que han proliferado en toda la Internet, permitiendo ser usado el sensor no solo para el juego sino también como sistema de medición. En Junio 16, del 2011, fue lanzado oficialmente el Microsoft SDK. En marzo 2013, las librerías del Kinect Fusion fueron liberadas. Ellas permitían reconstruir la escena 3D en tiempo-real de manera simple sosteniendo en mano el Kinect y moviéndolo en el espacio. En el verano

del 2014, fue introducido al mercado la nueva generación Kinect 2.0. Este nuevo sensor es más preciso y es basado en la tecnología tiempo-de-vuelo (Gokturk, Yalcin, & Bamji, 2005).

2.3.1 Sensor Kinect 1.0

Kinect fue construido para revolucionar la modalidad de juego de las personas y cómo ellos se entretienen experimentando (Zhang , 2012). Con Kinect, la gente es capaz de interactuar con los juegos por medio de su cuerpo de una manera natural. La clave de esta tecnología está en permitir el entendimiento del lenguaje-corporal humano que un usuario está haciendo antes de poderlo responder.

El sensor Kinect incorpora significativos avances de hardware de sensado (Tashev, 2011). Notablemente, contiene un sensor de profundidad, una cámara de color y un arreglo de cuatro micrófonos que proporciona la captura completa del cuerpo en movimiento 3D, reconocimiento facial y capacidades para reconocimiento de voz (Figura 2.10a). La Figura 2.10b, muestra un arreglo de proyección infrarrojos, la cámara de color y la cámara IR. El sensor de profundidad consiste del proyector IR en combinación con la cámara IR, el cual es un sensor semiconductor metal-oxido complementario monocromático (CMOS).

Figura 2.10. Sensor Microsoft Kinect. (a) El sensor Kinect para Xbox 360. (b) El proyector infrarrojo (IR), cámara IR y dentro del sensor Kinect una cámara RGB

La tecnología sensado-profundidad esta licenciada por una compañía Israelí "PrimeSense[2]". A pesar de que la tecnología exacta no se ha dado a conocer, ésta se encuentra basada en el principio luz estructurada. El proyecto IR, es un láser IR que pasa a través de la rejilla difracción y se convierte en un conjunto de puntos IR. La Figura 2.11 muestra los puntos IR vistos por la cámara IR.

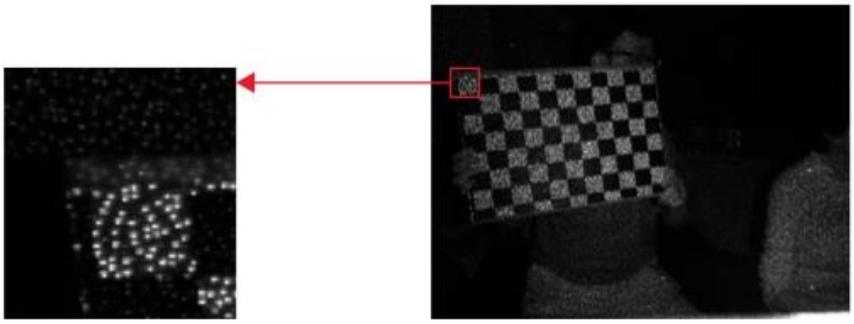


Figura 2.11. Los puntos infrarrojos IR vistos por la camara IR. La imagen en la izquierda muestra un acercamiento del area con puntos rojos

[2] www.primesense.com

2.3.2 Sensor Kinect 2.0

El sensor Kinect 2.0 tiene el mismo número de sensores (Pagliari & Pinto, 2015) como el Kinect 1.0 (Figura 2.12).

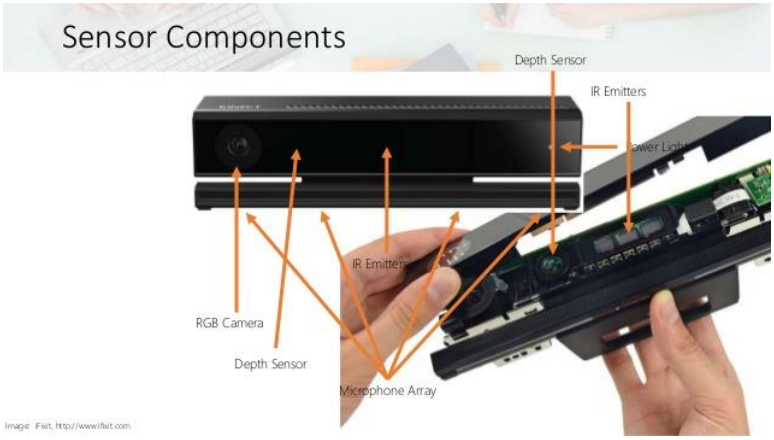


Figura 2.12. Localización de componentes, Kinect 2.0

Sin embargo, la profundidad es medida con un principio de medición completamente diferente. Las imágenes RGB son adquiridas en Alta Definición (en inglés HD). Para realizar una comparación directa entre las imágenes RGB entregadas por las dos generaciones de los dispositivos Kinect, observe la Figura 2.13.

Figura 2.13. Imagen RGB de Kinect 1.0 (a) comparado con la imagen RGB de Kinect 2.0; Las dos imágenes han sido capturadas del mismo punto

Las imágenes RGB e IR adquiridas por Kinect 2.0 están parcialmente traslapadas, desde la cámara el nuevo color tiene una cobertura horizontal amplio en lo que llamamos

Campo de Visión (en inglés Field of View, *FOV*), mientras que la nueva cámara IR tiene una cobertura verticalmente muy amplia en FOV (mirar Figura 2.14).

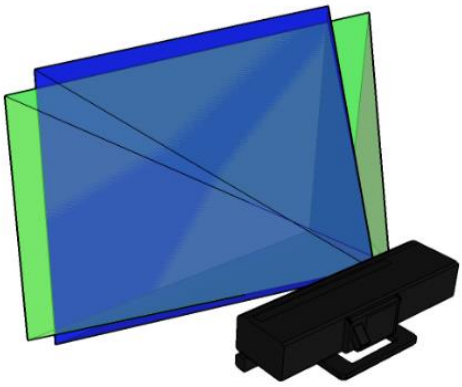


Figura 2.14. FOV de cámaras RGB e IR de Kinect 2.0. En azul está representado el FOV de la cámara IR, mientras que en verde está representado el FOV de la cámara RGB

Kinect 2.0 está definido por *Microsoft* como una camara tiempo-de-vuelo (en inglés *Time of Flight, ToF*). Actualmente, la cantidad observada está en una fase medición, lo que no es completamente correcto para definirlo como un sensor tiempo-de-vuelo (Gokturk, Yalcin, & Bamji, 2005). Sin embargo, se ha decido mantener esta terminología como literatura que describe el sensor. Las principales características se describen en la Tabla 2.1.

Tabla 2.1. Comparación de las principales características entre Kinect 1.0 y Kinect 2.0

	Kinect 1.0	**Kinect 2.0**
RGB camera (pixel)	1280 × 1024 or 640 × 480	1920 × 1080
Depth camera (pixel)	640 × 480	512 × 424
Max depth distance (m)	4.0	4.5
Min depth distance (m)	0.8	0.5
Horizontal FOV (degrees)	57	70
Vertical FOV (degrees)	43	60
Tilt motor	Yes	No
Skeleton joint define	20	26
Full skeleton tracking	2	6
USB	2.0	3.0
Price (€)	80	199

2.5 Aprendizaje profundo (AP)

El aprendizaje profundo como el aprendizaje máquina, son dos representaciones de Inteligencia Artificial. El aprendizaje profundo, a diferencia del segundo, se dice que es el desarrollo de sistemas autónomos capaces de cambiar su comportamiento en base a su experiencia, por lo que, puede aprender de datos proporcionados.

El AP permite que los modelos computacionales que se componen de varias capas de procesamiento, aprendan representaciones de datos con múltiples niveles de abstracción (LeCun, Bengio, & Hinton, 2015). Estos métodos han mejorado rápidamente el estado de la técnica en reconocimiento de voz, reconocimiento de objetos visuales, detección de objetos y muchos otros dominios como el descubrimiento de fármacos y la genómica[3].

Un métrico de desempeño del algoritmo, es la precisión, que hace referencia a la proporción del modelo que predice la salida de manera correcta. El otro métrico es la tase de error, que es la proporción de salidas predichas incorrectamente. Para evaluar el desempeño del algoritmo correctamente lo conveniente es emplear un conjunto de datos diferente al usado para entrenarlo (Perez Perez, 2017).

Los algoritmos de aprendizaje pueden clasificarse como supervisados y no supervisados. El no supervisado maneja un conjunto de datos con muchas características del cual aprende propiedades para agrupar esos datos, esto sin tener algún tipo de ayuda o guía. En cambio, el aprendizaje supervisado emplea un conjunto de datos formado por ejemplos, asociados con una etiqueta. Así es como el algoritmo aprende a clasificar en base a etiquetas, como si tuviera un profesor que le ensañara al sistema las decisiones que debe tomar.

La base para que el algoritmo aprenda, utiliza un porcentaje mayor de los datos proporcionados, este se llama conjunto de datos de entrenamiento. Después de entrenar el modelo, se utiliza un conjunto de prueba, que sirve para indicar lo bien que funciona el algoritmo. Como se mencionaba anteriormente, se puede comparar el métrico de error

[3] Es un campo interdisciplinario de la ciencia dentro del campo de la biología molecular

de entrenamiento y el métrico de prueba, los cuales tienden a reducirse al máximo en un cierto tiempo y número de iteraciones.

2.6 Redes Neuronales Artificiales (RNA)

Las redes neuronales son una implementación muy sencilla de un comportamiento local, observado en el cerebro. El cerebro está compuesto de neuronas, las cuales son elementos individuales de procesamiento. La información viaja entre las neuronas que basado en la arquitectura y ganancia de los conectores neuronales, la red se comporta de forma diferente (Ledesma, 2006). Cada neurona está conectada con otra neurona por medio de un peso de ajuste representado por la letra w. Una neurona artificial (Figura 2.15) está formada por un sumador y una función de activación, representada por f(x).

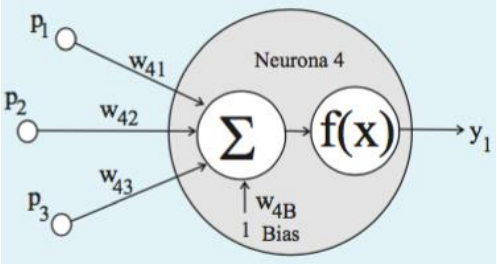


Figura 2.15 Neurona artificial

Una de las funciones de activación que se utiliza para la clasificación de imágenes es la denominada *softmax*, que es la extensión de la función sigmoide para más de dos clases (multiclase). Su papel es de importancia central en muchos modelos probabilísticos no lineales. En particular, en modelos bien conocidos que tratan con datos discretos y continuos (Bouchard, 2008).

La ecuación de la función *softmax* está dada por (Mikolov, 2010)

$$g(Z_m) = \frac{e^{Z_m}}{\sum_k e^{Z_k}} \qquad \text{Ec.1}$$

Para asegurar que la probabilidad de distribución sea válida, $e^{Z_m} > 0$ donde m representa las imágenes y $\sum_k e^{Z_k} = 1$.

En este trabajo se manejó la RNA Autocodificador Disperso, 2 Autocodificadores apilados y una capa con una función de activación *softmax*. Esta función de activación es usada para resolver problemas de función de transferencia no lineal (exponencial) en lugar de logísticas, normalizando las salidas para que su suma sea 1 (Munoz Perez, 2012). Se utilizó esta red con 10 salidas que corresponde a cada seña.

2.7 Principales métodos de Aprendizaje Profundo

Debido al creciente interés en el campo de Aprendizaje Profundo, al paso del tiempo existen más tipos de algoritmos, que nacen como modificadores de otros para objetivos determinados y concretos (Perez Perez, 2017).

Los algoritmos más comunes se agrupan en las cuatro categorías siguientes: 1) Redes Neuronales Convolucionales, 2) Máquinas de *Bolztmann* restringidas, 3) Autocodificadores y 4) Codificación dispersa, las cuales se muestran de manera más clara en la Figura 2.16, así como sus variantes.

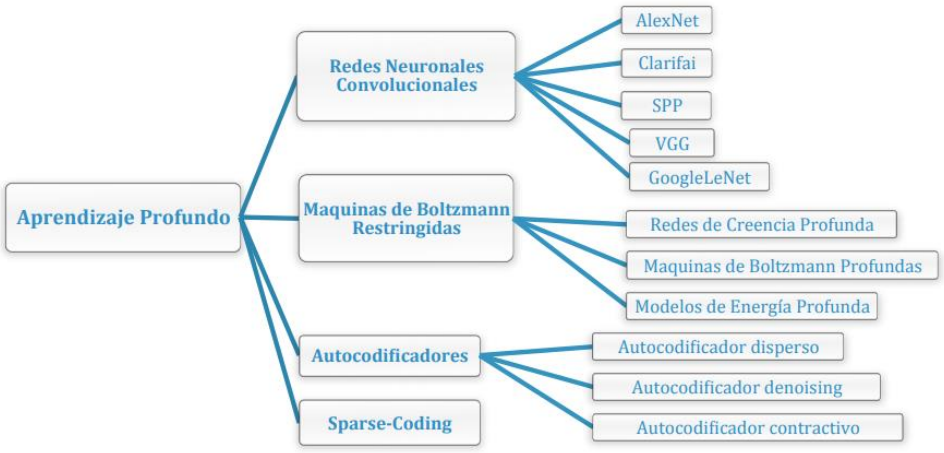


Figura 2.16. Esquema de agrupación de los métodos de Aprendizaje Profundo

2.8 Autocodificadores (AE)

Un AE es una RNA que se entrena para intentar copiar su entrada a su salida. Tiene un algoritmo de aprendizaje no supervisado que aplica retro-propagación, estableciendo valores objetivos iguales a los de entrada. El AE intenta aprender una aproximación a la función de identidad, de modo que la salida “r” es similar a “x” (Coates,

Lee, & Ng, 2010). Internamente, tiene capas ocultas “h” que describen el código utilizado para representar la entrada. La red puede ser vista en dos partes: una función codificador h=f(x) y un decodificador que produce una reconstrucción r=g(h). La arquitectura es representada por la Figura 2.17. Los AEs podrían ser tomados como un caso especial de redes de realimentación (*feed-forward*) y podría ser entrenado de igual manera con todas las técnicas, típicamente un gradiente descendiente miniatura, siguiendo el gradiente calculado por la propagación hacia atrás (*back-propagation*).

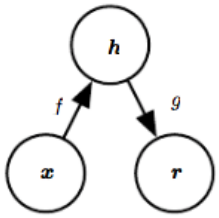


Figura 2.17. La estructura general de un autoencoder, mapeando una entrada x a un salida (llamado reconstrucción) r a través de una representación interna o codificado h. El AE tiene dos componentes: el codificador f (mapeo de x a h) y el decodificador g (mapeo de h a r).

El mapeo aprendido de la parte del codificador de un AE puede ser muy útil para extraer características de la imagen. Cada neurona en la sección codificadora tiene un vector de pesos asociado al cual será configurado (afinación) para responder a una característica visual en particular. Entrenar el AE usando señales produce resultados como se muestra en la Figura 2.18 en la capa escondida.

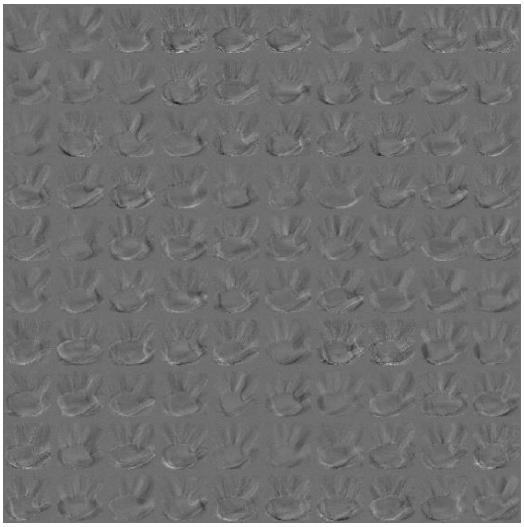


Figura 2.18. Representación de los pesos de cada neurona

Como se puede observar, los AEs son capaces de capturar características importantes y discriminantes de las imágenes.

Las RNAs son populares en tareas de reconocimiento de patrones de imágenes (Verma & Blumenstein, 2008). Han sido de gran interés para la comunidad científica, ya que emular el principal sentido de los seres humanos (la vista) es de suma importancia para poder realizar ciertas tareas automáticamente que en la actualidad requieren un sistema de visión, y que son actividades muy repetitivas, por ejemplo, la verificación de firmas, reconocimiento de caras, reconocimiento de objetos (Jaramillo González, 2009), etc. Generalmente se aplican con otros métodos de extracción de características.

2.7.1 Autocodificadores dispersos (SAE)

Como se mencionaba en el sección anterior, la función de un AE es intentar aprender una aproximación de la función identidad, lo que significa que la salida sea similar a la entrada. Existen restricciones que se pueden aplicar al algoritmo con la finalidad de descubrir estructuras interesantes sobre los datos, es la de aplicar en las neuronas ocultas un término *sparsity* (dispersión). Esto da lugar a los autocodificadores dispersos (SAE) (Martínez Álvarez, 2017).

Comúnmente se piensa que una neurona se activa si su valor de salida es cercano a 1, o que una neurona esta inactiva cuando su valor es cercano a 0. El propósito es limitar a las neuronas inactivas durante el entrenamiento.

Pensando que a_j expresa la activación de la neurona oculta *j* dada una determinada entrada *x* será:

$$\widehat{p_j} = \frac{1}{m}\sum_{i=1}^{m}\left[a_j(x^{(i)})\right] \qquad \text{Ec. 2}$$

La activación media de la neurona oculta *j* (media sobre el conjunto de entrenamiento). El propósito es cumplir, aproximadamente, la restricción siguiente:

$$\hat{p} = p$$

Donde *p* es el parámetro de *sparsity*, habitualmente cercano a 0 (*p* = 0.05). Esto quiere decir que la activación media de cada neurona oculta *j* sea próxima a 0.05. Después, para compensar esta restricción, la mayoría de las activaciones de las neuronas ocultas deben estar cercanas a 0.

Esto evita que los *SAE* utilicen todas las neuronas ocultas a la vez y obliga a usar solo un número reducido de neuronas ocultas. A medida que se activan e inactivan las neuronas ocultas, extrae una característica del conjunto de datos (Ng, 2011).

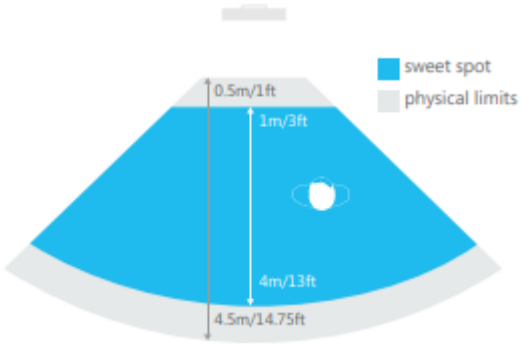
CAPÍTULO III. DESARROLLO

3.1 Introducción

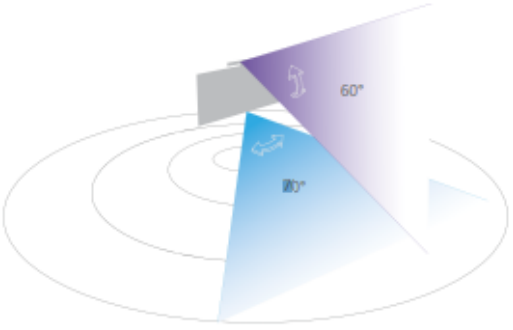
En este capítulo se observará el diseño, los experimentos, entrenamiento de la RNA, conjunto de datos e imágenes de profundidad, que conforman un sistema de visión con inteligencia artificial. Este sistema puede ser de gran ayuda para maestros y alumnos en escuelas para la inclusión de personas con deficiencia auditiva en el desarrollo intelectual y personal.

3.2 Especificación técnica del sensor de profundidad Kinect

La herramienta que se utilizará para capturar imágenes, es el sensor Kinect 2.0. Tiene una resolución 512 x424, un rango operativo de medición de 0.5m a 4.5m (Figura 3.1 (a)). Angulo de visión en Horizontal 70 grados y en Vertical 60 grados (Figura 3.1 (b)). (Lachat, Macher, Landes, & Grussenmeyer, 2015)



(a)



(b)

Figura 3.1. (a) Distancia en rango de operación, (b) Angulo de visión

3.3 Processing

Durante el desarrollo de esta investigación, se ha utilizado la herramienta *Processing.* Es un software flexible y un lenguaje de programación dentro de un contexto de artes visuales. Desde el 2001, Processing ha promovido la literatura visual dentro de la tecnología.

A continuación, se enlistan algunas de las características:

- Descarga gratuita y código abierto.
- Programas interactivos con 2D, 3D o salida PDF.
- Integración OpenGL para aceleradores 2D y 3D.
- Para GNU/Linux, Mac OS X, Windows, Android, and ARM.
- Más de 100 librerías extendidas para core software.

3.4 Procedimiento para la instalación de drivers del sensor Kinect en OS X

Para poder utilizar las funciones del sensor Kinect 2.0 con el sistema operativo OS X de Apple, se debe realizar una serie de pasos para instalar los controladores del dispositivo. En comparación con el .NET *Framework* de Windows, es un procedimiento inusual. Se requiere una secuencia de pasos (McPherson, n.d.), como se muestra a continuación (Figura 3.2):

- Instalar y actualizar Xcode.
- Instalar software adicional, (MacPorts).
- Instalar dependencias (Libtool, LibUSB).
- Instalar OpenNI SDK (Figura 3.2).
- Instalar Sensor Kinect.
- Instalar NiTE.

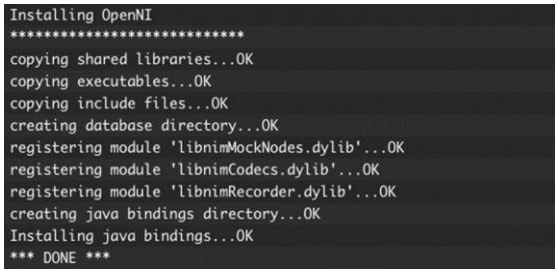


Figura 3.2. Detalles de la instalación de los drivers

3.5 Ambiente de desarrollo Processing utilizando sensor Kinect

En la Figura 3.3 se observa una sección de código donde se configura una imagen de salida de 1024 x 848 pixeles. Existen dos métodos en este código, el Setup() y el Draw(). En Setup() se inicializan los componentes de video; y en Draw(), se muestra la imagen de salida del kinect. En la sección Setup(), se crea el objeto de tipo Kinect, el cual invoca los métodos: initVideo(), para inicializar imagen de RGB; initDepth(), para inicializar imagen de profundidad, initIR(), para inicializar imagen infrarroja y initRegistered() que es una combinación de las anterior mencionadas. En la sección Draw(), la función background(0) permite refrescar la pantalla. La función Image() permite visualizar el video de acuerdo a la característica seleccionada y el cuadrante donde se desea mostrar.

```
RGBDepthTest2
// Daniel Shiffman
// All features test

// https://github.com/shiffman/OpenKinect-for-Processing
// http://shiffman.net/p5/kinect/

import org.openkinect.freenect.*;
import org.openkinect.freenect2.*;
import org.openkinect.processing.*;

Kinect2 kinect2;

void setup() {
  size(1024, 848, P2D);

  kinect2 = new Kinect2(this);
  kinect2.initVideo();
  kinect2.initDepth();
  kinect2.initIR();
  kinect2.initRegistered();
  // Start all data
  kinect2.initDevice();
}


void draw() {
  background(0);
  image(kinect2.getVideoImage(), 0, 0, kinect2.colorWidth*0.267, kinect2.colorHeight*0.267);
  image(kinect2.getDepthImage(), kinect2.depthWidth, 0);
  image(kinect2.getIrImage(), 0, kinect2.depthHeight);

  image(kinect2.getRegisteredImage(), kinect2.depthWidth, kinect2.depthHeight);
  fill(255);
  text("Framerate: " + (int)(frameRate), 10, 515);
}
```

Figura 3.3. Código en Processing

El sensor interpreta 4 tipos de imagen: RGB, profundidad, infrarroja y la cuarta es una combinación de las tres primeras antes mencionadas (Figura 3.4).

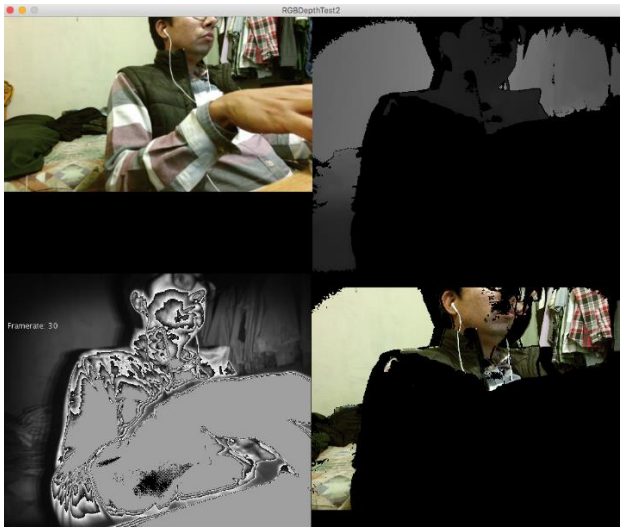


Figura 3.4. Imágenes: RGB, profundidad, IR y combinado

3.6 Limitaciones en ambiente Processing

A continuación, se describe algunas de las limitaciones durante el desarrollo de capturas de imágenes con el software Processing utilizando el sensor Kinect:

- Es una imagen borrosa (Figura 3.5).
- Filtros no definidos en imagen de profundidad (Figura 3.6).
- Resolución de salida de video no modificable.
- Filtros para extracción de características no desarrollados.
- No acepta DLL elaborados en .NET.
- Amplio conocimiento en JAVA.
- Máquina virtual no reconoce los controladores nativos.

Figura 3.5. Imagen de profundidad

Figura 3.6. Imagen de profundidad, en cierto rango de distancia

3.7 *Kinect for Windows Software Development Kit* (SDK) 2.0

Una herramienta de programación alternativa para la manipulación del sensor Kinect 2.0 es *Kinect for Windows SDK 2.0*. Este permite a los desarrolladores crear aplicaciones que soporten reconocimiento de gesto y voz, usando la tecnología del sensor Kinect en computadoras con sistema operativo *Windows 8, Windows 8.1 y Windows Embedded Standard 8*. Las características que debe tener la computadora, son las siguientes:

- Procesador de 64-bit (x64).
- Memoria de 4 GB (o mayor).
- Doble-núcleo físico de 3.1 GHz (2 núcleos lógicos por cada físico).
- Controlador dedicado USB 3.0.

3.7.1 Instalación de controlador del sensor *Kinect* en *MS Windows*

Para poder utilizar el sensor Kinect 2.0 con el propósito de desarrollar software, se deben realizar los siguientes pasos, para la correcta instalación de los controladores, descritas a continuación:

- Descargar y ejecutar el instalador “KinectSDK-v2.0_1409-Setup” de la página web de Microsoft.

- Asegurar que el sensor Kinect esté conectado a la fuente alimentación. Conecte el cable USB al puerto 3.0 de la computadora. La instalación del controlador se ejecutara automáticamente.

- Esperar a que la instalación se complete. Se puede verificar la instalación en el administrador de dispositivos, identifique en la lista el controlador con el nombre “KinectSensor Device”.

3.7.2 Verificación de la instalación del controlador

Una vez instalado el controlador, es requerido verificar la instalación, en seguida se muestran los pasos:

- Asegure que el “*KinectService*” está en ejecución (Figura 3.7).

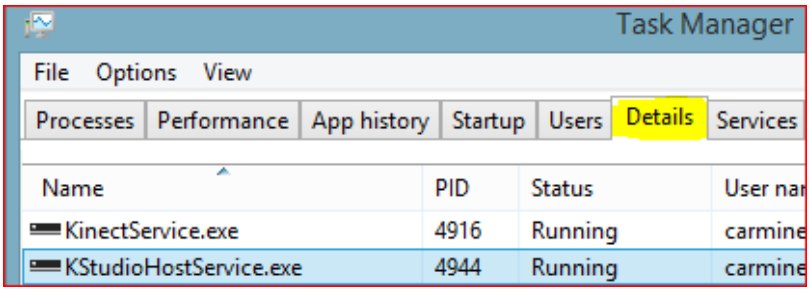


Figura 3.7 Servicio de “KinectService” en ejecución

Los tres LEDs rojos en el sensor Kinect deben estar iluminados (Figura 3.8).

Figura 3.8. LEDs rojos iluminados en sensor Kinect

El sensor Kinect debería aparecer en el Administrador de dispositivos con el siguiente nombre: "*Audio inputs and outputs: Microphone Array (Xbox NUI Sensor), KinectSensor Device: WDF KinectSensor Interface 0*" (Figura 3.9).

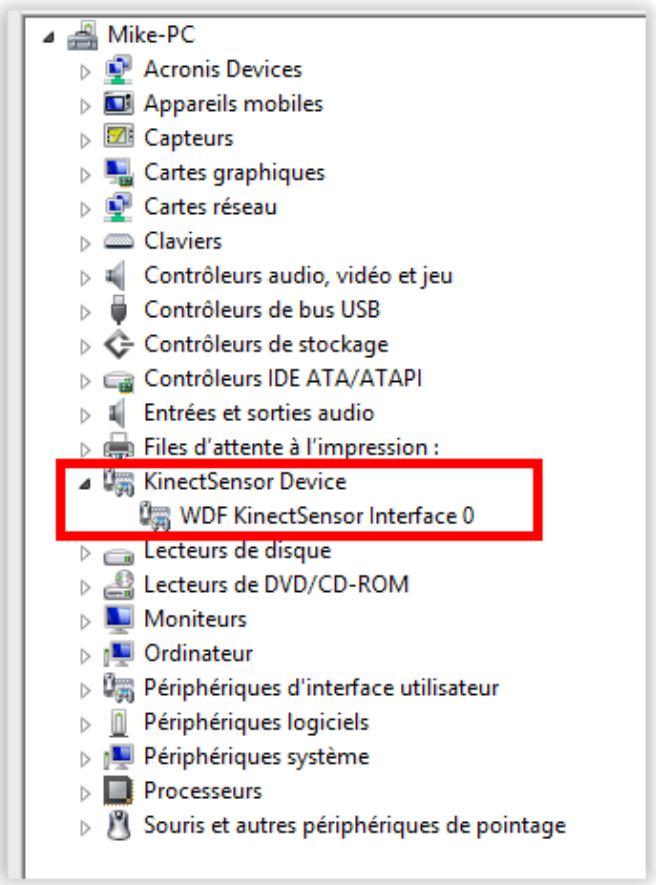


Figura 3.9. Controlador del Kinect identificado en el administrador de dispositivos

3.8 Componentes del sensor Kinect

De los principales componentes de interés, que satisface el sensor Kinect para el desarrollo de este proyecto, son: Esqueleto humano, emisor infrarrojo y cámara de profundidad.

3.8.1 Esqueleto humano

Otra característica de utilizar sensor Kinect, es que puede dibujar de manera representativa el esqueleto de la persona. Dentro del ambiente de C# existen algunas funciones que proporcionan esta información, así como identificar la posición de la mano a partir de las articulaciones del cuerpo. En la Figura 3.10, se puede observar una imagen de profundidad realizado con Kinect, donde la mano está abierta y como se representa de manera gráfica el esqueleto.

Figura 3.10. Captura de imagen de profundidad, resaltando esqueleto del cuerpo

3.8.2 Emisor infrarrojo (IR)

El emisor de infrarrojos es capaz de proyectar una luz infrarroja en una habitación. A medida que, la luz incide en la superficie del objeto, el patrón se distorsiona. Esta distorsión es leída a través de otro componente, una cámara de profundidad. (Figura 3.11)

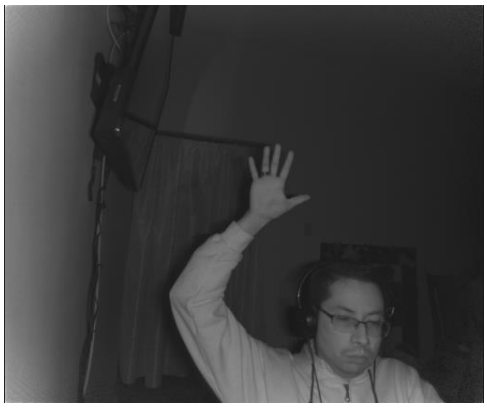


Figura 3.11. Captura de imagen infrarroja

3.8.3 Cámara de profundidad

Un sensor de profundidad utiliza luz infrarroja estructurada como base para su funcionamiento. Se establece una especie de triangulación entre la emisión de patrones de luz infrarroja y un sensor monocromático infrarrojo de CMOS (Córdova Lucero, Diciembre 2002). El sensor calcula los puntos reflejados y deduce la profundad en base a la intensidad y la deformación de los puntos. Microsoft es una empresa que fabricó este

sensor y hasta el momento nadie ha podido descifrar el algoritmo para el funcionamiento de este dispositivo. Según algunos investigadores han podido observar los puntos blancos y negros (Figura 3.12) para hacer una correlación de datos, en lo que compara el pixel actual con el vecino.

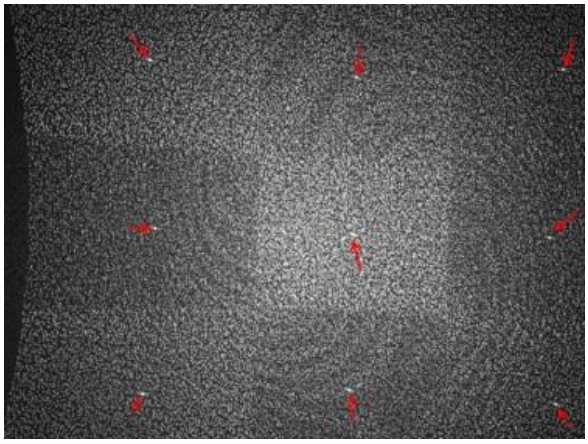


Figura 3.12. Patron de puntos infrarrojos emitidos por sensor Kinect

Esta cámara analiza los patrones infrarrojos emitidos por el emisor y es capaz de construir un mapa 3D de la habitación, así como de todos los objetos y personas que se encuentren en ella (Figura 3.13).

Figura 3.13. Captura de imagen de profundidad

3.9 Patrones de reconocimiento

Dentro de los patrones de reconocimiento de imágenes, está la función DrawHand(), propia de Kinect en Windows SDK (Figura 3.14). Ésta identifica el estado que encuentra la mano (abierta, cerrada o un solo dedo) y la posición.

```
/// <summary>
/// Draws a hand symbol if the hand is tracked: red circle = closed, green circle = opened; blue circle = lasso
/// </summary>
/// <param name="handState">state of the hand</param>
/// <param name="handPosition">position of the hand</param>
/// <param name="drawingContext">drawing context to draw to</param>
private void DrawHand(HandState handState, Point handPosition, DrawingContext drawingContext)
{
    switch (handState)
    {
        case HandState.Closed:
            drawingContext.DrawEllipse(this.handClosedBrush, null, handPosition, HandSize, HandSize);
            break;

        case HandState.Open:
            drawingContext.DrawEllipse(this.handOpenBrush, null, handPosition, HandSize, HandSize);
            break;

        case HandState.Lasso:
            drawingContext.DrawEllipse(this.handLassoBrush, null, handPosition, HandSize, HandSize);
            break;
    }
}
```

Figura 3.14. Código DrawHand()

Otra de las características de este SDK, es su capacidad de identificar qué unión o articulación corresponde a la mano derecha y mano izquierda. Esto es muy útil, ya que se obtienen los valores en X y en Y para fines específicos dentro del plano de referencia de la imagen (Figura 3.15).

```
20 references
void ProcessJointInformation(int joint, Joint j, JointOrientation jo, double time, int pointScale, UdpWriter osc
{
    if (j == null) return;
    if (jo == null) return;
    SendJointMessage(joint,j.Position.X, j.Position.Y, j.Position.Z,JointToConfidenceValue(j), time,
        pointScale, osc, fileWriter);
    if (joint == 9)
        System.Threading.Thread.Sleep(10);
    if (joint == 15)
        System.Threading.Thread.Sleep(10);
}

1 reference
```

Figura 3.15. Código para identificar la articulación mano derecha e izquierda

Esta función procesa la información cuando el primer *frame* es recibido (Figura 3.16). Si se añade demasiadas funciones en este proceso, puede ocasionar que el despliegue de información y captura sea muy lento.

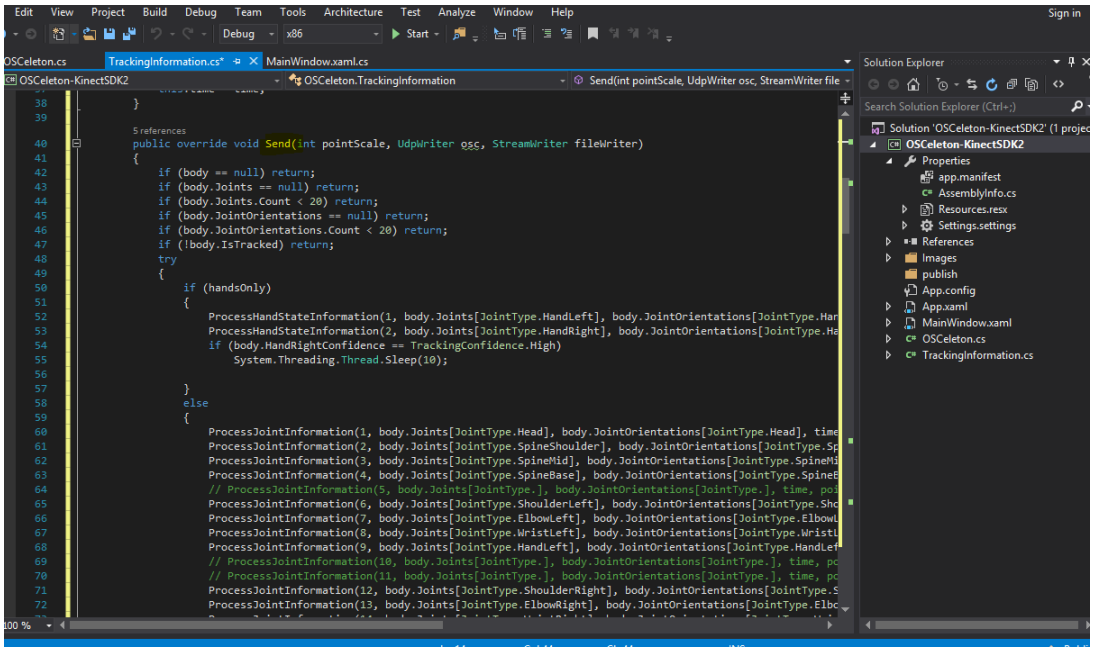


Figura 3.16. Código, sección que identifica la articulación del cuerpo

Utilizando las funciones propias del SDK de Kinect, se pueden obtener las características del contorno de la mano. En la siguiente imagen (Figura 3.17) se observan capturas del borde de la mano con el sensor de profundidad Kinect 2.0, lo cual representa los números del 1 al 5 del lenguaje de señas.

Figura 3.17. Capturas Kinect, representación de señas del número 1 al 5

3.10 *ROI*

Una región-de-interés (ROI) es muy útil al momento de utilizar procesamiento de imágenes. En este proyecto es indispensable implementar una herramienta de este tipo con la finalidad de solo extraer ciertas características de la salida completa del sensor Kinect. La *ROI* básicamente es el contorno de un segmento cuya función orgánica es evaluar al medir la cantidad de actividad que atraviesa dicha región (Núñez, 2008).

En la Figura 3.18 se observa cómo se extraen características de la imagen original para establecer una nueva.

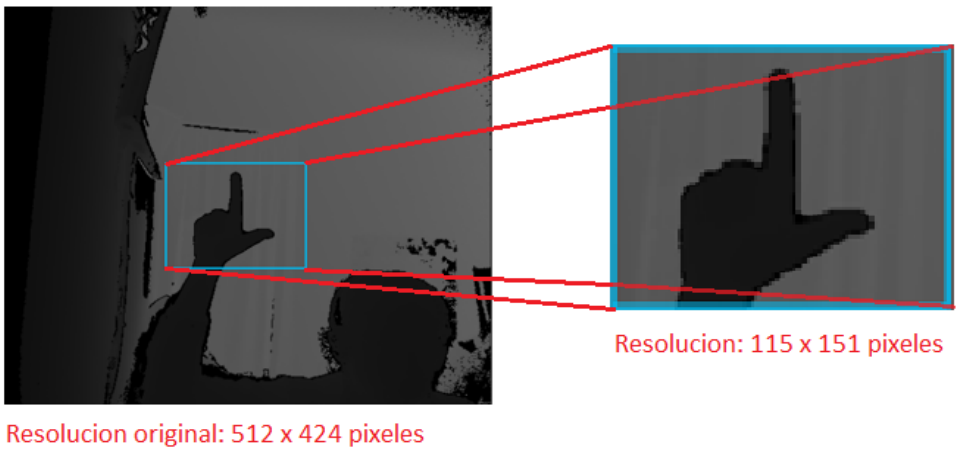


Figura 3.18. Diseño de *ROI* en C# con sensor *Kinect*

Uno de los objetivos principales de realizar esta segmentación, es facilitar el entrenamiento de la RNA, ya que conforme la imagen de entrada sea de una resolución grande, como ésta tiene mucha información, el tiempo de entrenamiento podría ser infinito, por lo que el sistema no tendría un desempeño adecuado.

3.11 Captura de imágenes para entrenamiento de la red AE

El requisito primordial de todo este proyecto, definitivamente son las imágenes. El código desarrollado en C# es capaz de guardar estas imágenes con ciertas instrucciones, como se muestran en la Figura 3.19. El formato de la imagen es png (*Portable Network Graphics*) de 32bit.

```
198             // create a png bitmap encoder which knows how to save a .png file
199             BitmapEncoder encoder = new PngBitmapEncoder();
200
201             CroppedBitmap cropImg = new CroppedBitmap(this.depthBitmap.Clone(), new Int32Rect(Left - 75, Top, Width, Height));
202             // create frame from the writable bitmap and add to encoder
203             encoder.Frames.Add(BitmapFrame.Create(cropImg));
204
205             // uncomment for save pictures for training
206             string myPhotos = "C:\\Users\\Ezequiel\\Documents\\MATLAB\\PRP\\Kinect_depthCaptures_Mar292019";
207             string path = Path.Combine(myPhotos, "S01_C10_" + inc.ToString() + ".png");
```

Figura 3.19. Sección de código para crear imagen png en C# con sensor Kinect

Para crear el conjunto de datos que se utilizará en el entrenamiento de la RNA, se eligieron 10 señas del diccionario de LSM (Figura 3.20). El ejercicio es capturar 300 de cada seña, para crear un total de 3000 imágenes de entrenamiento.

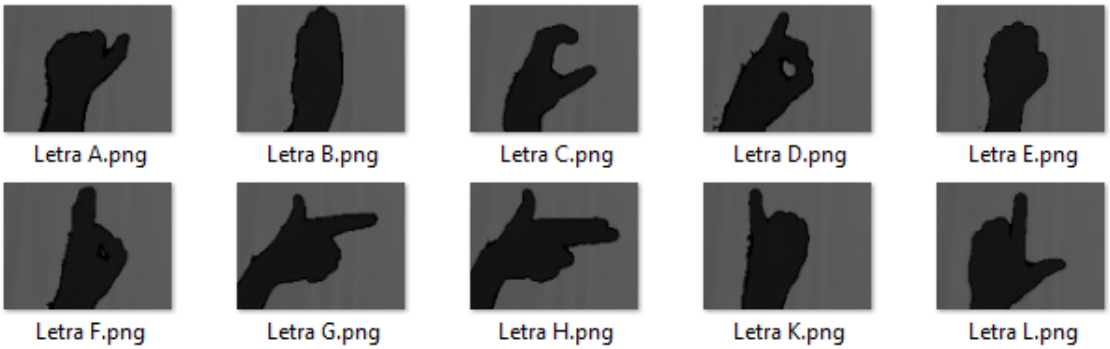


Figura 3.20. Ejemplos de imágenes capturadas con el sensor Kinect, representando 10 señas del LSM

3.12 Conjunto de datos de entrada

Se distinguen dos grupos de datos principales, el de entrenamiento y el de prueba. La primera parte como conjunto de entrenamiento, se utiliza para determinar los parámetros del clasificador y el conjunto de prueba se utiliza para estimar el error de generalización, lo más cercano a cero. El conjunto de entrenamiento suele dividirse en entrenamiento y validación para ajustar el modelo (García Cambronero & Gómez Moreno, 2006).

El conjunto de datos de entrada utilizado es el creado anteriormente con el sensor Kinect. El repositorio de estas imágenes se encuentra en el siguiente link https://github.com/pablorp80/rivas-dataset.git. Este conjunto está compuesto de imágenes de letras del lenguaje de señas mexicana (LSM) que están divididos en matrices de 115x151 píxeles. Esto significa que la dimensión del conjunto de datos de

entrada es 17365. El conjunto de datos entrada se divide a su vez en tres conjuntos diferentes: entrenamiento, validación y test. El conjunto de entrenamiento consta de 1500, el de validación de 750 y el de prueba de 750. Es decir, del conjunto total de entrada se han tomado 3000 muestras y se ha aplicado el 50%, 25% y 25% para cada uno de los conjuntos respectivamente, así como se muestra en la Figura 3.21.

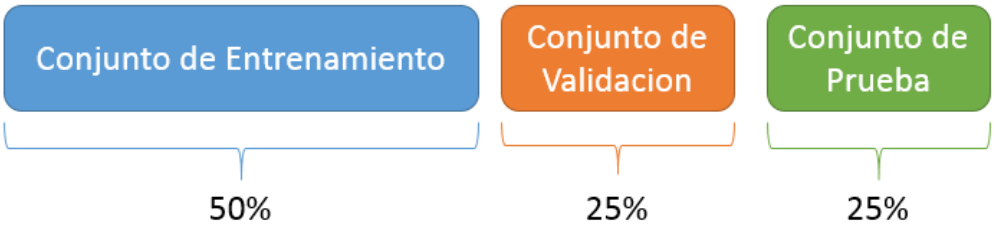


Figura 3.21. División conjunto de datos de Entrada

3.13 Entrenamiento de la red neuronal AE

La imagen de profundidad capturada con el sensor Kinect, es de una resolución grande, con ruido y variación en los datos de la imagen, por lo que, uno de los primeros pasos es normalizar la imagen, para esto se utiliza la herramienta de MATLAB. Este software es creado para analizar y diseñar sistemas que transforman nuestro mundo. Las áreas donde se utiliza con mayor frecuencia son: Aprendizaje profundo (*Deep learning*), procesamiento de señales, procesamiento de imágenes, visión artificial, comunicaciones, finanzas computacionales, diseño de control, robótica y algunos otros campos (MathWorks, n.d.).

Los pasos a seguir para el entrenamiento integral de la red profunda son: Definir variables de entrada, entrenamiento de primer AE, entrenamiento de segundo AE, entrenamiento de red softmax, apilar capas de red AE y softmax, *fine-tuning* con el conjunto de entrenamiento y generar métricos de entrenamiento.

3.8.4 Definir parámetros de entrada

En la Figura 3.22 se muestran las imágenes de entradas para el entrenamiento de la red AE

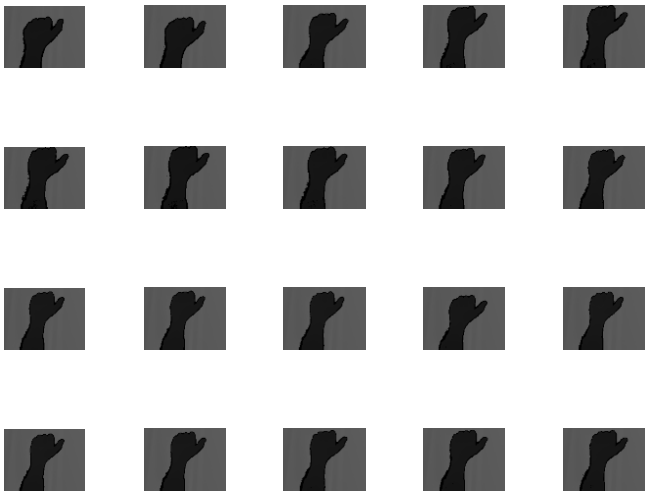


Figura 3.22. Representación gráfica de la matriz de entrada de la parte codificadora del AE (solo 20 elementos)

En la primera etapa se definen 3 variables con el *xTrainImages* con el 50% de las imágenes para el entrenamiento. *xTestImages* con el 25% para imágenes de prueba y *xValImages* con el 25% para validación cruzada (Figura 3.23).

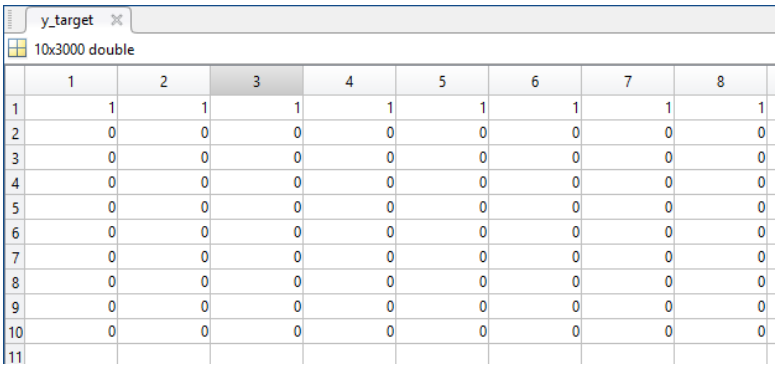
```
46 -    [idxTrain, idxRest] = crossvalind('HoldOut', length(X_input), 0.5); % 50%
47 -    xTrainImages = X_input(idxTrain);  tTrain = y_target(:, idxTrain);
48 -    X_rest = X_input(idxRest); y_rest = y_target(:, idxRest);
49
50 -    [idxTest, idxVal] = crossvalind('HoldOut', length(X_rest), 0.5);
51 -    xTestImages = X_rest(idxTest); tTest = y_rest(:, idxTest); % 25%
52 -    xValImages = X_rest(idxVal); tVal = y_rest(:, idxVal); % 25%
53 -    clear X_rest y_rest idx*
```

Figura 3.23. Variables de entrenamiento

La variable *y_target* es una matriz de 10 x 3000 donde cada columna de un elemento en particular será un 1 para indicar la clase de la seña que pertenece y el otro elemento, dentro de la misma columna, será un cero.

En la Figura 3.24 se muestra la representación gráfica de la variable y_target, donde cada columna es una imagen y cada renglón es una seña deseada. El objetivo de

definir esta variable es para realizar un aprendizaje supervisado, donde le indicamos a la red que cada entrada corresponde a cierta seña.



y_target

10x3000 double

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
11								

Figura 3.24. Matriz de la variable *y_target*

3.8.5 Entrenamiento de primer AE

Como segundo paso del entrenamiento, se utiliza la función *trainAutonencoder()*, donde *autoenc1* es la estructura de la red SAE (Figura 3.25). *xTrainImages* es el conjunto de entrenamiento; *hiddenSize1* es el número de neuronas ocultas, *L2WeightRegularization* controla el impacto de un regularizador L2 para los pesos de la red (y no los sesgos). Esto debería ser bastante pequeño (Regularización para lograr dispersión: Regularización L_1, 2018); *SparsityRegularization* controla el impacto de un regularizador de dispersión, que intenta imponer una restricción en la dispersión de la salida de la capa oculta; *SparsityProportion* es un parámetro del regularizador *sparsity*. Controla la dispersión de la salida de la capa oculta. Cuando se le da un valor bajo, por lo general, cada neurona de la capa oculta se "especializa" al dar solo un alto rendimiento para un pequeño número de conjuntos de entrenamiento. Por ejemplo, si se establece en 0.1, esto equivale a decir que cada neurona en la capa oculta debe tener una salida promedio de 0.1 sobre los ejemplos de entrenamiento. Este valor debe estar entre 0 y 1. El valor ideal varía según la naturaleza del problema.

```
121 -    fprintf('Training autoencoder1\n');
122 -    autoenc1 = trainAutoencoder(xTrainImages,hiddenSize1, ...
123          'MaxEpochs',500, ...
124          'L2WeightRegularization',0.004, ...
125          'SparsityRegularization',4, ...
126          'SparsityProportion',0.15, ...
127          'ScaleData', false, ...
128          'ShowProgressWindow',false);
```

Figura 3.25. Entrenamiento del primer AE

Después de realizar el entrenamiento del primer AE, se realiza una reconstrucción en base al conjunto de entrenamiento. En la Figura 3.26 se observan las características extraídas, donde las imágenes son muy similares a las de entrada. En la Figura 3.27 se representa de manera gráfica los valores de los pesos del autocodificador.

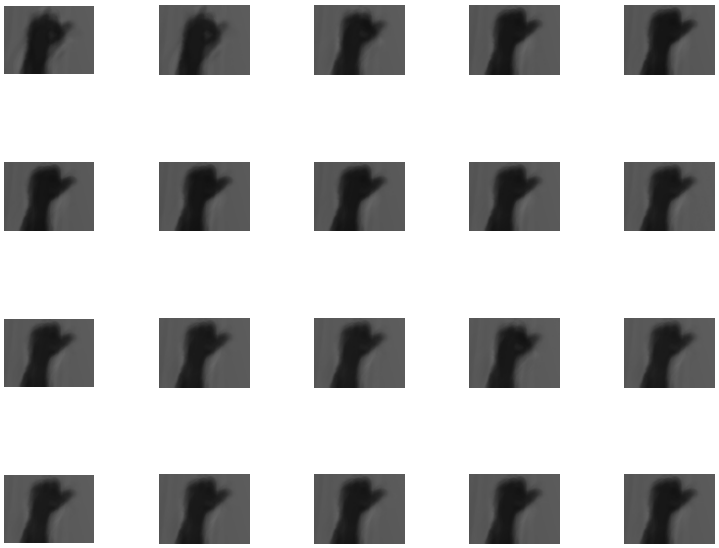


Figura 3.26. Representación gráfica de la matriz de la reconstrucción de la parte decodificadora del AE

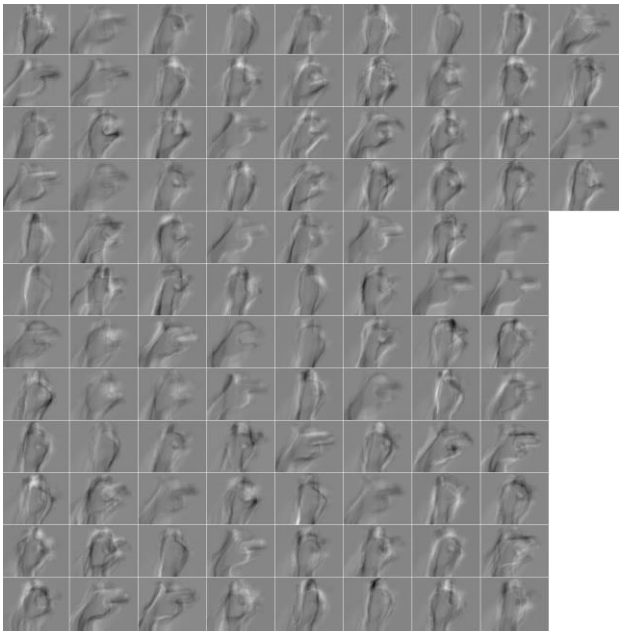


Figura 3.27. Representación de los pesos (W) del primer AE

3.8.6 Entrenamiento del segundo AE

Después del entrenamiento del primer AE, se entrenó el segundo de una manera similar. En la Figura 3.28 se puede ver la principal diferencia es que se utilizan características del primer AE. Solo cambia el tamaño de la representación de la capa oculta a 50 neuronas y el máximo de épocas 100. En la Figura 3.29 se representa; de manera gráfica los valores de los pesos del autocodificador.

```
167 -    fprintf('Training autoencoder2\n');
168 -    hiddenSize2 = 50;
169 -    autoenc2 = trainAutoencoder(feat1,hiddenSize2, ...
170          'MaxEpochs',100, ...
171          'L2WeightRegularization',0.002, ...
172          'SparsityRegularization',4, ...
173          'SparsityProportion',0.1, ...
174          'ScaleData', false, ...
175          'ShowProgressWindow',false);
176
```

Figura 3.28. Entrenamiento del segundo AE

Figura 3.29. Representación de los pesos (W) del segundo AE

3.8.7 Entrenamiento de red Softmax

En este paso (Figura 3.30) se entrena una capa de softmax para clasificar los vectores de entidades de 50 dimensiones. A diferencia de los autocodificadores, la capa de softmax se entrena en un entorno supervisado utilizando etiquetas para los datos de entrenamiento. El parámetro *feat2* representa las características del segundo AE. El *tTrain* es una matriz de salida, con valores deseados (etiquetas). Para este trabajo, la salida *softnet* de la red es entrenada para clasificar 10 clases. Cada clase representa una seña.

```
202 -    fprintf('Training softmax network\n');
203 -    softnet = trainSoftmaxLayer(feat2,tTrain,'MaxEpochs',500);
204
```

Figura 3.30. Entrenamiento de la capa Softmax

3.8.8 Apilar las capas AEs y Softmax

En esta etapa se realiza un ensamble de los codificadores de los autocodificadores junto con la capa de softmax para formar una red profunda. Prácticamente se utiliza una función *stack()* con los tres elementos que se desean apilar (Figura 3.31).

```
224
225 -    deepnet = stack(autoenc1,autoenc2,softnet);
226
```

Figura 3.31. Ensamble de red profunda

Como fue explicado anteriormente, los AEs se usaron para extraer características de las imágenes muy representativas. Se pueden apilar AEs junto con la capa de salida *softmax* para formar una RNA de aprendizaje profundo (Figura 3.32). Donde *Input* representa la matriz de imágenes de entrada, el primer *Encoder* con 100 neuronas ocultas, el segundo *Encoder* con 50 neuronas, y por último la capa *Softmax* con 10 clases. *Output* es la salida que representa la seña reconocida del sistema entrenado.

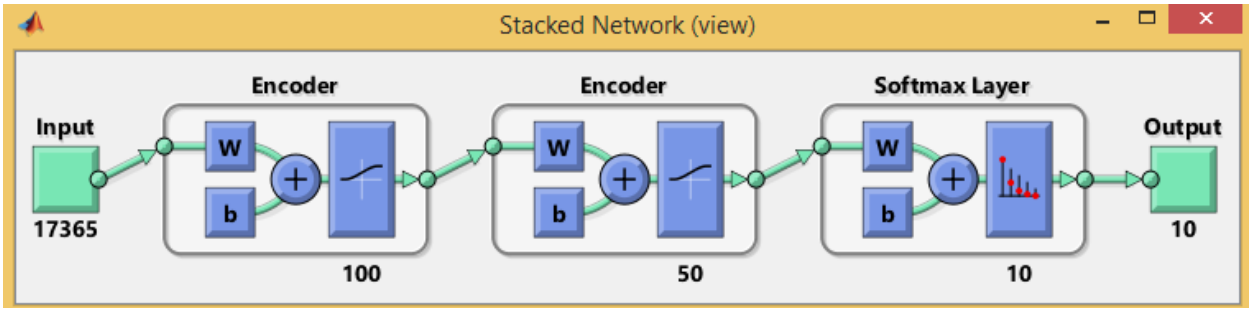


Figura 3.32. Modelo de entrenamiento del AE

3.8.9 *Fine tuning* con el conjunto de entrenamiento

Se puede volver a ajustar la red entrenando con los datos de entrenamiento de forma supervisada. En la Figura 3.33 se muestra la función *train()* utilizada para realizar el ajuste fino. Antes de poder hacer esto, se debe cambiar la forma de las imágenes de entrenamiento en una matriz *xTrain*. El parámetro *tTrain* representa los valores deseados.

```
278 -    [deepnet,tr] = train(deepnet,xTrain,tTrain);
```

Figura 3.33. Método *train()* para realizar el ajuste fino (*fine-tuning)*

3.8.10 Generar métricos de entrenamiento

En esta sección se menciona como se obtienen los métricos de regresión para evaluar el desempeño de la red en base los conjuntos de entrenamiento, prueba y validación cruzada. En la línea 285 de la Figura 3.34 se introducen a la red profunda "*deepnet*" los valores de entrenamiento *xTrain* para validar el desempeño con respecto a la clasificación. Estos resultados se almacenan en *y*. Para obtener lo métricos, se utiliza el método "prpGetMetrics()", donde *y* son los resultados obtenidos y *tTrain* son los datos esperados.

```
285 -    y = deepnet(xTrain);
286 -    fprintf('Training Error\n');
287 -    prpGetMetrics(y, tTrain);
288 -    plotconfusion(tTrain,y);
289 -    save Train_sub1 y tTrain
```

Figura 3.34. Instrucciones para obtener métricos

3.14 Agregar DLL de MATLAB en Visual Studio

Para realizar la conexión entre MATLAB y C#, recibir y mandar información a través de estos programas, es necesario agregar una instancia en el programa de C#. Se abre un proyecto. En el menú "*Project*", se selecciona "*Add Reference*". Después se selecciona la ficha COM en el cuadro de diálogo "Agregar referencia". Por último, de la lista se selecciona la aplicación MATLAB. En la Figura 3.35 aparece agregado la herramienta de MATLAB como "MLApp" en el grupo de referencias.

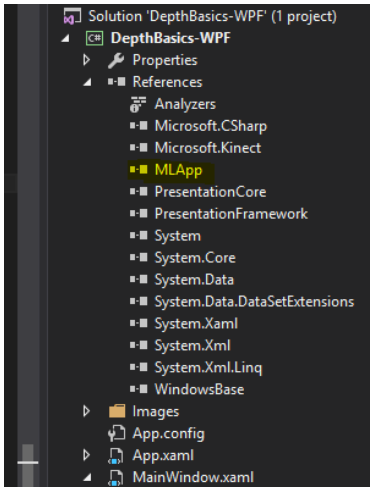


Figura 3.35. Referencia de MATLAB agregado en el proyecto de C#

3.15 Implementación de la interfaz intérprete

Finalizado el entrenamiento del RNA AE correctamente, se procede con la evaluación automática de resultados. Por medio de la interfaz de C# donde se está ejecutando en tiempo real el video de profundidad emitida por el sensor Kinect, se toma una captura, creando una imagen nueva. Esta imagen se envía a través del método *Feval()* (Linea 242 de la Figura 3.36), donde los argumentos son: "prpEvaluate" es el script que procesa la imagen en MATLAB, "1" cantidad de elementos, "out result" almacena los resultados y "S01" es el nombre de la imagen (sujeto).

```
235                     // Change to the directory where the function is located
236                     matlab.Execute(@"cd C:\Users\Ezequiel\Documents\MATLAB\PRP");
237
238                     // Define the output
239                     object result = null;
240
241                     // Call the MATLAB function myfunc
242                     matlab.Feval("prpEvaluate", 1, out result, "S01");
243
244                     // Display result
245                     //object[] res =  result as object[];
246
247                     object[] res = (object[])result;
248
249                     var x = (double[,])res[0];
250
251                     if (x[0, 0] >= 0.95)
252                         MessageBox.Show("Letter #A");// Letter #1
253                     else if (x[1, 0] >= 0.95)
254                         MessageBox.Show("Letter #B");// Letter #2
255                     else if (x[2, 0] >= 0.95)
256                         MessageBox.Show("Letter #C");// Letter #3
257                     else if (x[3, 0] >= 0.95)
258                         MessageBox.Show("Letter #D");// Letter #4
259                     else if (x[4, 0] >= 0.95)
260                         MessageBox.Show("Letter #E");// Letter #5
261                     else if (x[5, 0] >= 0.95)
262                         MessageBox.Show("Letter #F");// Letter #2
263                     else if (x[6, 0] >= 0.95)
264                         MessageBox.Show("Letter #G");// Letter #3
265                     else if (x[7, 0] >= 0.95)
266                         MessageBox.Show("Letter #H");// Letter #4
267                     else if (x[8, 0] >= 0.95)
268                         MessageBox.Show("Letter #I");// Letter #5
269                     else if (x[9, 0] >= 0.95)
270                         MessageBox.Show("Letter #L");// Letter #4
271
```

Figura 3.36. Sección de código en C# que interpreta la imagen de entrada como seña

En base a los resultados obtenidos por el método *Feval()*, se realiza una sencilla evaluación, donde la variable “res” contiene el resultado de la clasificación de la RNA a la que pertenece la imagen introducida desde C#. Si su valor es muy cercano a 1, quiere decir que realizó la clasificación de una seña, por lo que el sistema lo interpreta y muestra en un *MessageBox()* como se muestra en la Figura 3.37.

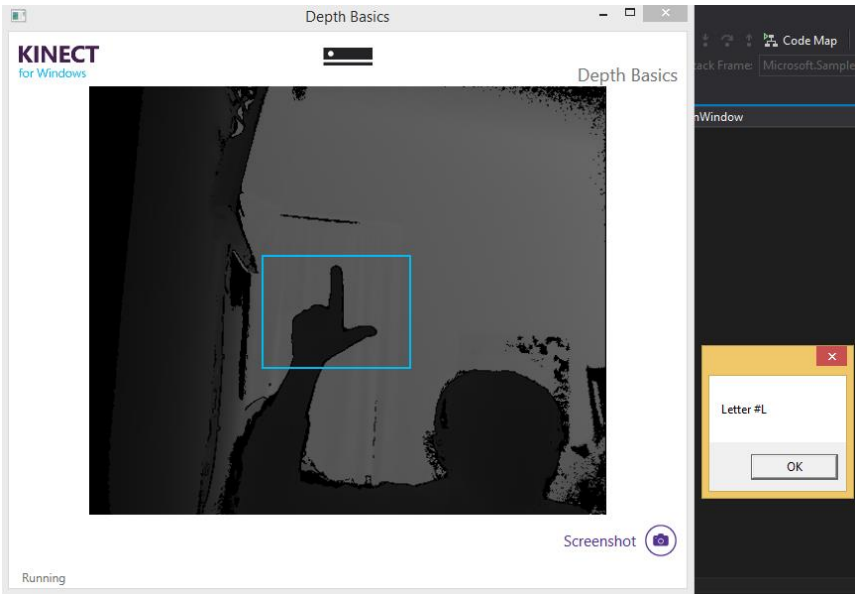


Figura 3.37. Imagen reconocida por el sistema interprete como la letra “L”

3.16 Diseñó del algoritmo (diagrama de flujo)

En la Figura 3.38 se muestra en un diagrama a bloques el funcionamiento del algoritmo. Primero se realiza la adquisición de imágenes de profundidad con el sensor Kinect. Posteriormente se define una región de interés y las imágenes se transforman en datos normalizados. Después se realiza el entrenamiento de las imágenes con las señas correspondientes o si ya fue entrenado, solo se activan las salidas de acuerdo a las entradas utilizando la red de arquitectura profunda. Por último, en base a los datos obtenidos de la red *autoencoders,* son interpretados como seña, proporcionando la letra del alfabeto correspondiente a LSM en texto visual.

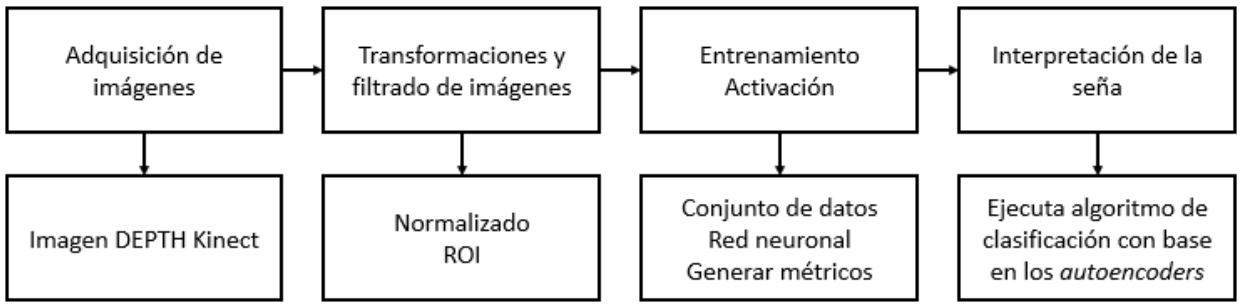


Figura 3.38. Etapas del sistema intérprete empleando inteligencia artificial

3.17 Arquitectura del sistema

A continuación se muestra en la Figura 3.39 la arquitectura del sistema intérprete. En el bloque del lado izquierdo corresponde al *Hardware* del sensor *Kinect.* Para este proyecto se utiliza la funcionalidad del proyector IR para obtener las imágenes de profundidad. Este se conecta por un puerto USB al *Hardware* de la PC. En la parte superior se encuentra lo relacionado a *Visual C#*, el cual es la interfaz entre el sensor *Kinect* y el procesamiento de la red neuronal en MATLAB. Para que el programa pueda manejar el sensor Kinect es necesario instalar el controlador *KinectSensor Device*. Para realizar la comunicación con MATLAB desde C# se debe agregar la referencia al programa de "*MLApp.dll"*. En MATLAB se procesa el conjunto de imágenes, se entrena

la red neuronal, se guardan los datos de entrenamiento y se validan los datos de entrada. Finalmente, la información proporcionada de la red neuronal, se interpreta en C# de manera visual con un mensaje de texto.

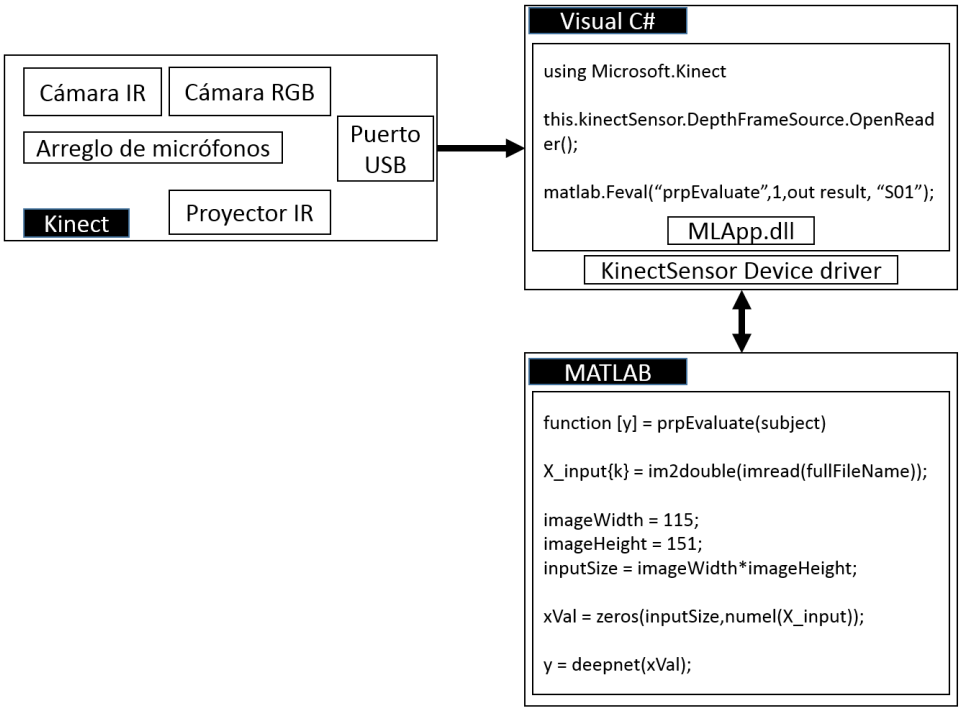


Figura 3.39. Arquitectura del sistema intérprete

CAPÍTULO IV. EVALUACIÓN DE RESULTADOS Y CONCLUSIONES

4.1 Evaluación de resultados

En esta sección se muestran los resultados obtenidos referente al a) entrenamiento de la red AE y su rendimiento con los conjuntos de entrenamiento, prueba y validación, así mismo, el b) entrenamiento de la red AE con imágenes obtenidas con cámara *Creative* y sensor *Kinect* y por último, la c) validación de la implementación del sistema interprete.

4.1.1 Resultados del entrenamiento, prueba y validación cruzada de la red AE con imágenes del sensor Kinect

En la Tabla 4.1 se muestran los métricos (Rivas Perea, 2011) para evaluar el rendimiento del entrenamiento de la red AE, durante las etapas de entrenamiento, prueba y validación.

Tabla 4.1. Métricos de los resultados del entrenamiento del AE

Metric	Training	Test	Cross Validation
Mean Absolute Error, MAE	0.00000	0.01241	0.02862
Root Mean Squared Error, RMSE	0.00002	0.27212	0.40511
Normalized Root Mean Squared Error, NRMSE	0.00001	0.09344	0.14424
Sum of Squared Error, SSE	0.00000	55.53740	123.08590
Absolute Error Mean + Error Std. Deviation, STAT	0.00003	0.28259	0.41549
Error Std. Deviation, RSTD	0.00002	0.27210	0.40525
Mean Error, RMEA	(0.00000)	(0.01049)	0.01023
True Positives Count, TP	1,500	747	744
True Negatives Count, TN	13,500	6,723	6,696
False Positives Count ,FP	-	3	6
False Negatives Count ,FN	-	27	54
True Positives Rate ,TPR	1.00000	0.96512	0.93233
False Positives Rate, FPR	-	0.00045	0.00090
Accuracy, ACC	1.00000	0.99600	0.99200
Specificity, SPC	1.00000	0.99955	0.99910
Positive Predictive Value, PPV	1.00000	0.99600	0.99200
Negative Predictive Value, NPV	1.00000	0.99600	0.99200
False Discovery Rate, FDR	-	0.00400	0.00800
Matthews Correlation Coefficient, MCC	1.00000	0.97824	0.95736
F1-Score, F1S	1.00000	0.98031	0.96124
Balanced Error Rate, BER	-	0.00449	0.00758

- Error de raíz cuadrático medio normalizado (NRMSE), donde y_i es un vector de N predicciones y d_i es un vector de valores deseados.

$$NRMSE = \frac{1}{\sigma}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - d_i)^2} \quad \text{EC. 3}$$

El desempeño obtenido en base a la tasa de error, tuvo un inicio excelente en la fase entrenamiento. Se nota una separación en relación a la etapa de prueba y validación, no obstante, sigue siendo aceptable (Figura 4.1).

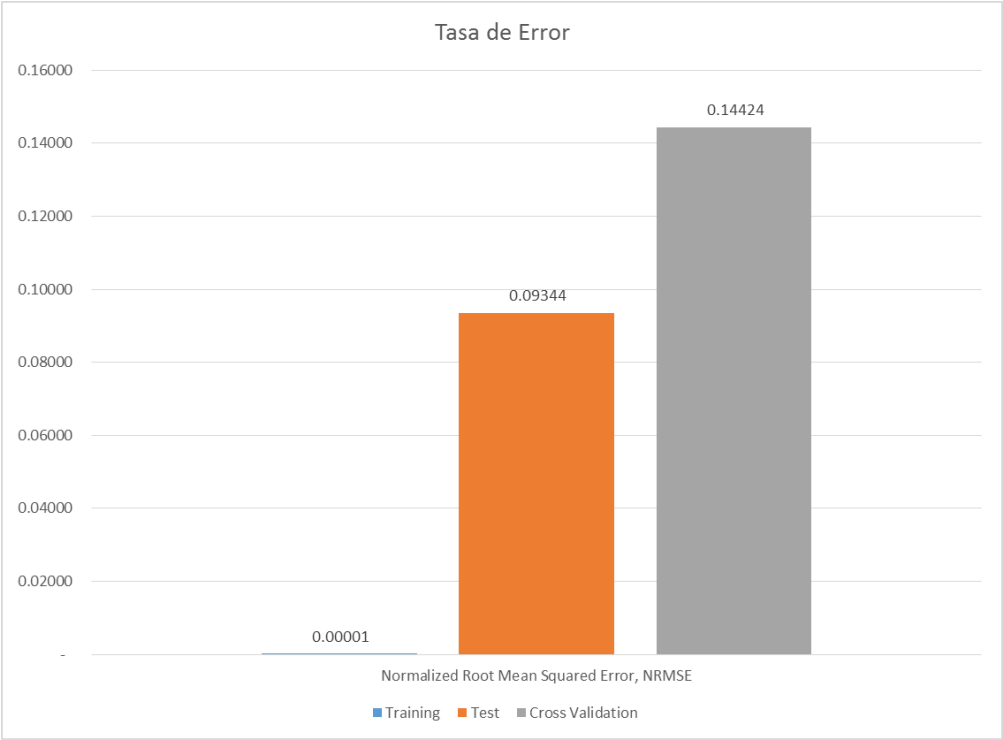


Figura 4.1. Comparación de tasa de error en entrenamiento, prueba y validación cruzada

- Exactitud o *accuracy* (ACC), en general, es la suma de los aciertos negativos y positivos entre la suma total de aciertos positivos y negativos, y falsos positivos y negativos. Lo que quiere decir que si los acierto negativos (TN) dismuyen lo mas cercano a cero o nulo, se obtendrá un resultado excelente.

$$ACC = \frac{TP+TN}{TP+FN+FP+TN} \quad \text{EC. 4}$$

En la Figura 4.2, se observa que durante las tres etapas, realizó la clasificación de las clases correctamente.

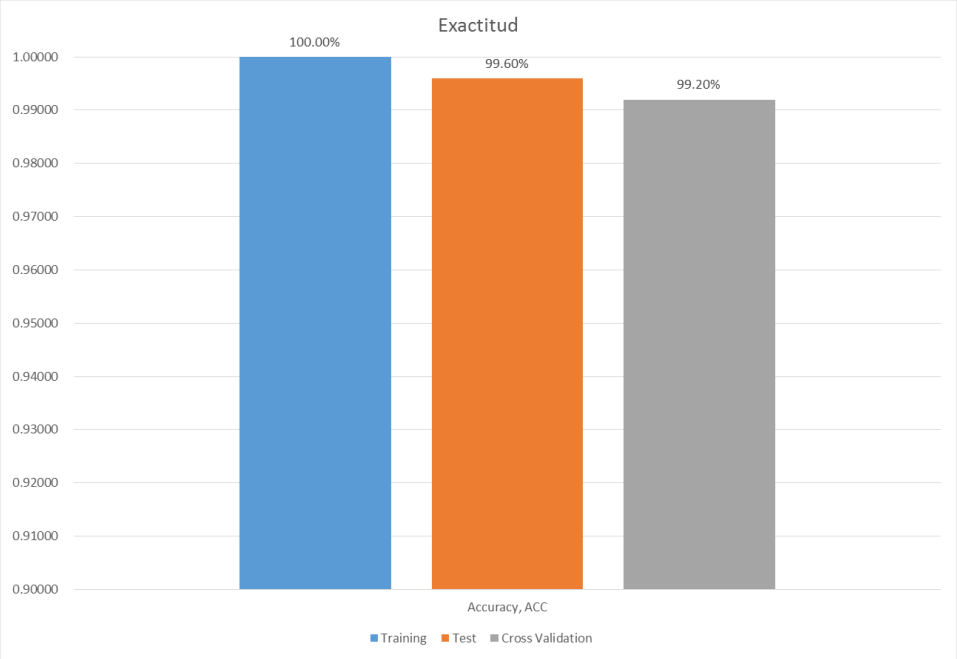


Figura 4.2. Comparación de exactitud en entrenamiento, prueba y validación cruzada

Se observa un resultado de 99.7% en general en la etapa de entrenamiento para todas las clases. Esto quiere decir que, de las 1500 imágenes, clasificó correctamente las 10 clases.

La verificación con las imágenes de prueba (750 imágenes), arrojó como resultado general un 99.6%.

Y por último, la verificación con las imágenes de validación cruzada (750 imágenes), da como resultado un 99.2%.

Estos diferentes métricos ayudan a medir diferentes aspectos del desempeño del modelo; esto facilita la determinación del comportamiento, de manera más generalizada.

4.1.2 Resultados del entrenamiento de la red AE con imágenes de la cámara *Creative* y sensor Kinect

A continuación, se muestran datos en la Tabla 4.2 sobre el resultado obtenido del entrenamiento con imágenes de la cámara Creative (repositorio FingerspellingRecognition[4]) e imágenes creadas con el sistema intérprete utilizado con Kinect.

Tabla 4.2. Métricos de desempeño de entrenamiento con imágenes de cámara *Creative* y sensor *Kinect*

Métrico	Creative	Kinect
NRMSE	0.01684	0.1442
ACC	99.95%	99.20%
F1S	100%	96%
BER	0.00042	0.0076

El desempeño obtenido es un poco mejor con las imágenes de la cámara Creative (Figura 4.3)

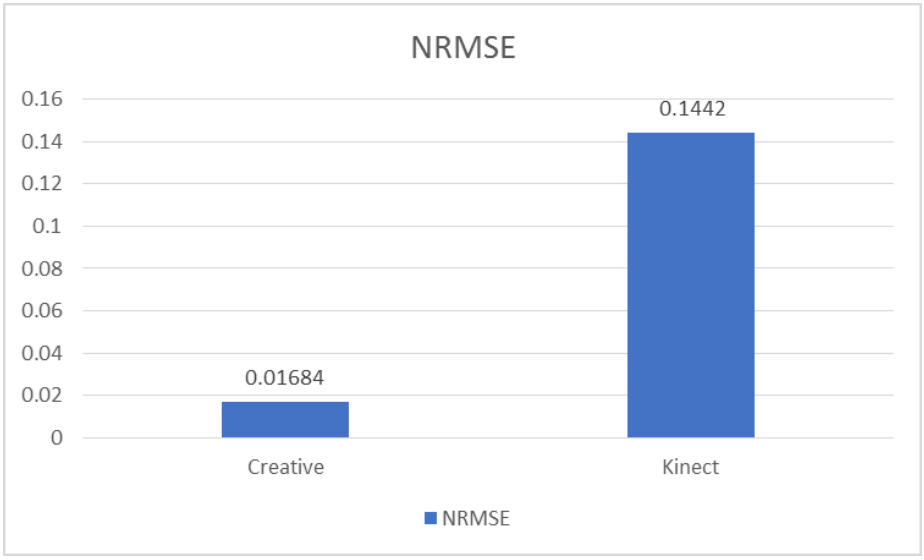


Figura 4.3 Comparación de tasa de error en validación cruzada, *Creative vs Kinect*

La exactitud obtenida con ambos sistemas ha sido satisfactoria, encontrándose poca diferencia en la exactitud del sistema. (Figura 4.4)

[4] https://github.com/byeongkeun-kang/FingerspellingRecognition

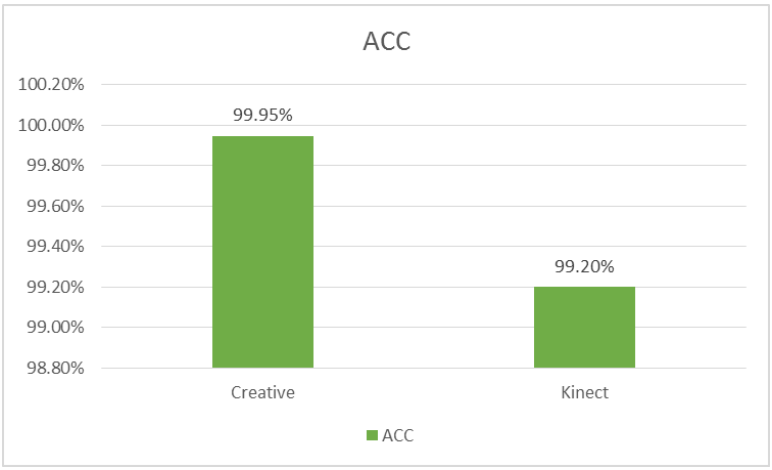


Figura 4.4. Comparación de exactitud en entrenamiento, *Creative vs Kinect*

En seguida se ilustran dos métricos, tasa de error balanceado (BER) (Figura 4.5) y precisión/ exhaustividad (F1-Score) (Figura 4.6). Se observa que el valor BER en las imágenes Creative es ligeramente mayor al de Kinect, de igual manera es un poco mejor, representando la precisión y exhaustividad.

- Tasa de error balanceado o *balanced error rate* (BER),

$$BER = \frac{1}{2}\left(\frac{FP}{TN+FP} + \frac{FN}{FN+TP}\right) \quad \text{EC. 5}$$

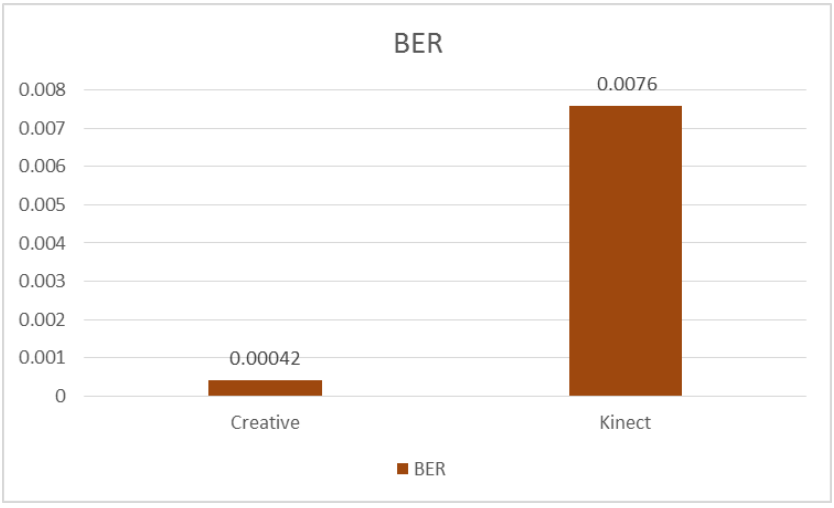


Figura 4.5. Comparación de tasa de error balanceado en entrenamiento, *Creative vs Kinect*

- Precisión y exhaustividad (F1-Score),

$$F_1 - Score = 2 \times \frac{\left(\frac{TP}{TP+FP}\right) \times \left(\frac{TP}{TP+FN}\right)}{\left(\frac{TP}{TP+FP}\right) + \left(\frac{TP}{TP+FN}\right)} \quad \text{EC. 6}$$

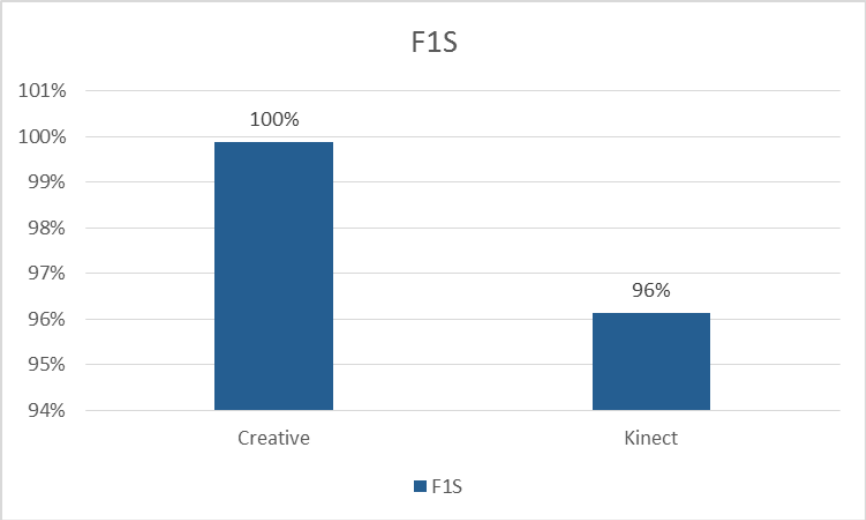


Figura 4.6. Comparación de precisión y exhaustividad en entrenamiento, *Creative vs Kinect*

En estas ecuaciones, TP (*true positive*) es un verdadero positivo; son los casos en el que predice si verdaderamente corresponde a la seña. TN (*true negative*) es un verdadero negativo; indica si verdaderamente no corresponde a la seña.

FP (*false positive*) es un falso positivo; es el caso en el que se predice que una seña pertenece a una clase pero en realidad pertenece a otra.

FN (*false negative*) es un falso negativo; es donde se predice incorrectamente que una seña no pertenece a una clase en específico.

4.1.3 Resultados de la validación del sistema

En base a los resultados obtenidos con el entrenamiento individual con las imágenes de *Kinect* y la comparación con las imágenes cámara *Creative*, se puede definir que el sistema es robusto para clasificar imágenes de profundidad.

Para validar el rendimiento del sistema, se realizó un diseño de experimento que consiste en: Capturar con el Kinect 400 imágenes. 10 de cada letra, de manera

consecutiva, empezando por la “A” y terminando por la “L” (exceptuando las letras “J” y “K”, ya que estas letras presentan movimiento), como se muestra en la Tabla 4.3.

Tabla 4.3. Tabla del diseño de experimento

Iteration	A	B	C	D	E	F	G	H	I	L	Miss recog.
1	1	1	1	1	1	1	1	1	1	1	
2	1	1	1	1	1	1	1	1	1	1	
3	0	1	0	1	1	0	1	1	1	1	cL, fG
4	0	1	1	1	1	0	1	1	0	1	aC, fB, iF
5	0	1	1	1	1	1	1	1	1	1	
6	1	1	1	1	1	1	1	0	1	1	
7	0	1	1	1	0	0	1	1	1	1	fD
8	1	1	1	1	1	0	1	1	0	1	fB, iD
9	1	1	1	1	0	1	1	1	1	0	eI, lC
10	1	1	1	0	1	0	1	1	1	1	dE, fD
11	1	1	1	1	1	0	1	1	1	1	fD
12	0	1	1	1	1	0	1	1	1	1	fD
13	1	1	0	0	0	0	1	1	0	1	dL, eI, fD, iB
14	1	1	1	0	0	1	1	1	0	1	dL, eA, iB
15	1	1	1	1	0	0	1	1	0	1	eA, fG, iD
16	1	1	1	0	1	0	1	1	1	1	dL
17	0	1	1	0	1	0	1	1	0	1	fD, iF
18	1	1	1	0	0	0	1	1	0	1	dE, eA, fD, iB
19	1	1	1	1	1	0	1	1	0	1	fD, iD
20	1	1	1	0	1	0	1	1	1	1	dE, fE
21	1	1	1	1	1	0	1	1	0	1	fD, iB
22	1	1	1	0	1	0	1	1	0	1	dE, fD, iB
23	1	1	0	0	0	1	1	1	0	1	dE, iB
24	1	1	1	0	0	1	1	0	0	1	dE, eG, hG, iB
25	1	1	1	1	0	1	1	1	0	1	eI, iB
26	1	1	1	1	0	1	1	0	0	1	eA, iB
27	1	1	1	1	0	0	1	1	0	1	eI, iB
28	1	0	1	1	0	1	1	0	0	1	eA, hG, iB
29	1	1	1	1	0	0	1	1	0	1	eB, fE, iB
30	1	1	1	0	1	1	1	1	0	1	, iB
31	1	1	1	0	0	1	1	1	0	1	dG, eI, iB
32	1	1	1	1	1	1	1	1	0	1	, iB
33	1	1	1	0	1	1	1	1	0	1	dE, iB
34	1	1	0	0	1	1	1	1	0	1	cG, dF, iB
35	1	1	1	0	1	1	0	1	0	1	dF, iB
36	1	1	1	0	1	0	1	1	0	1	dF, fI
37	1	1	1	0	0	1	1	1	1	0	dE, eA, lC
38	1	1	1	1	1	0	1	1	0	1	, iB
39	1	1	1	0	0	0	1	1	0	1	dF, eA, iB
40	1	1	1	0	0	0	1	1	0	1	dF, eA, fB, iA
Accuracy	0.85	0.975	0.9	0.525	0.575	0.45	0.975	0.9	0.325	0.95	

El “1” representa el valor positivo o acierto. El “0” representa el valor negativo o falso. En la columna “*Miss recog”* se observa el resultado de las letras que no corresponden a la letra deseada durante el proceso de validación.

Los resultados de este experimento se resumen en la Tabla 4.4:

Tabla 4.4. Tabla de resultado del experimento

Column1	A	B	C	D	E	F	G	H	I	L	TotalRecog	Good	Target	Total Miss	Unknown	Fail
A	34		1								35	34	0.85	6	5	1
B		39									39	39	0.975	1	1	0
C			36				1			1	38	36	0.9	4	2	2
D				21	8	4	1			3	37	21	0.525	19	3	16
E	9	1			23				5		38	23	0.575	17	2	15
F		3		10	2	18	2				35	18	0.45	22	5	17
G							39				39	39	0.975	1	1	0
H							2	36			38	36	0.9	4	2	2
I	1	20		3		2			13		39	13	0.325	27	1	26
L			2							38	40	38	0.95	2	0	2
Output	44	63	39	34	33	24	45	36	18	42		297			22	81
	0.294	0.6154	0.083	0.619	0.435	0.33	0.154	0	0.385	0.105						

En general, el sistema fue capaz de clasificar correctamente el 74%, incorrecto el 20% y desconocido solo el 6% del total de las capturas (Figura 4.7).

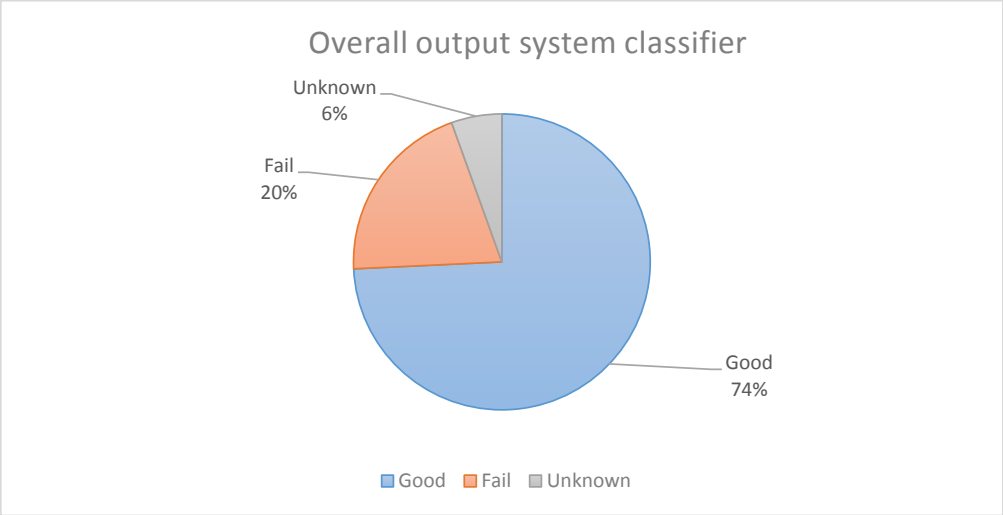


Figura 4.7. Resultados generales del sistema clasificador

El mejor resultado de exactitud se encuentra en la letra “B” y “G” con un 98% (Figura 4.8).

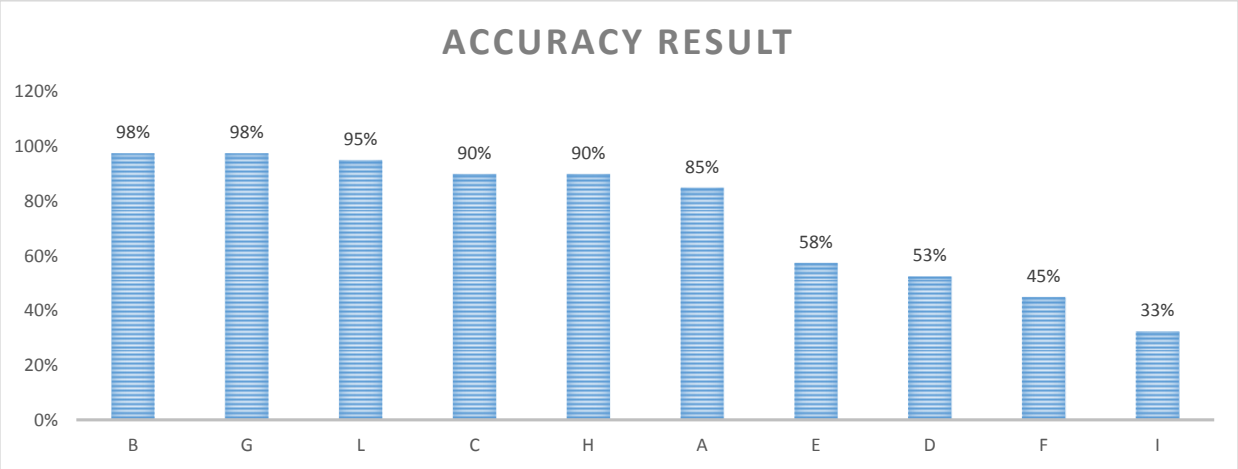


Figura 4.8. Resultado de exactitud por cada letra

En la Figura 4.9, se puede observar el orden de mayor a menor en referencia a la cantidad de ocasiones en que el sistema no fue asertivo.

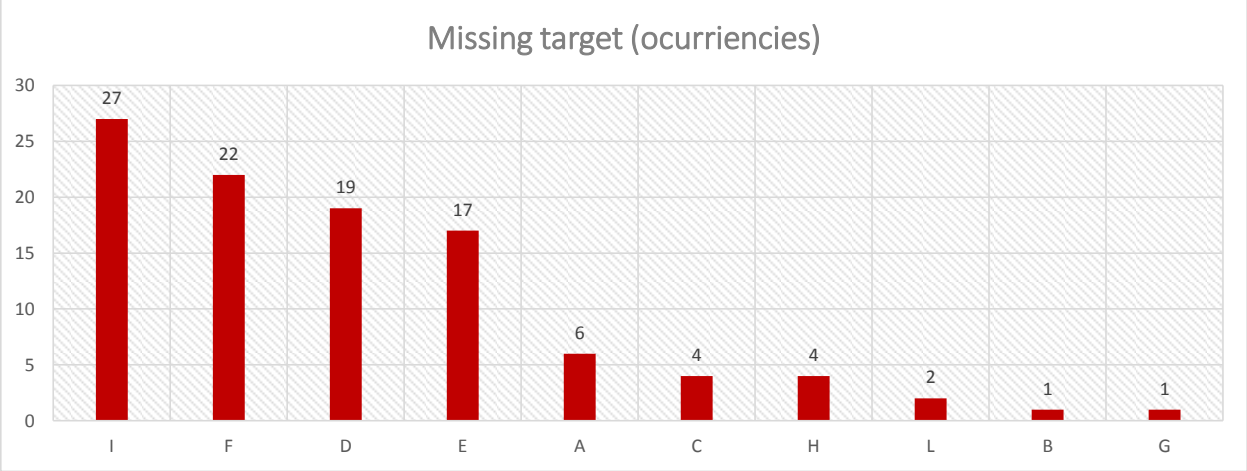


Figura 4.9. Total de Falsos Positivos y Falsos Negativos por letra

Destacando con el peor resultado la letra “I”. A continuación, se muestra un diagrama de pastel (Figura 4.10), en el que se observa que es mayor el valor de Falsos Positivos con un 51% (letra “B”) contra un Verdaderos Positivos de un 33% (letra “I”).

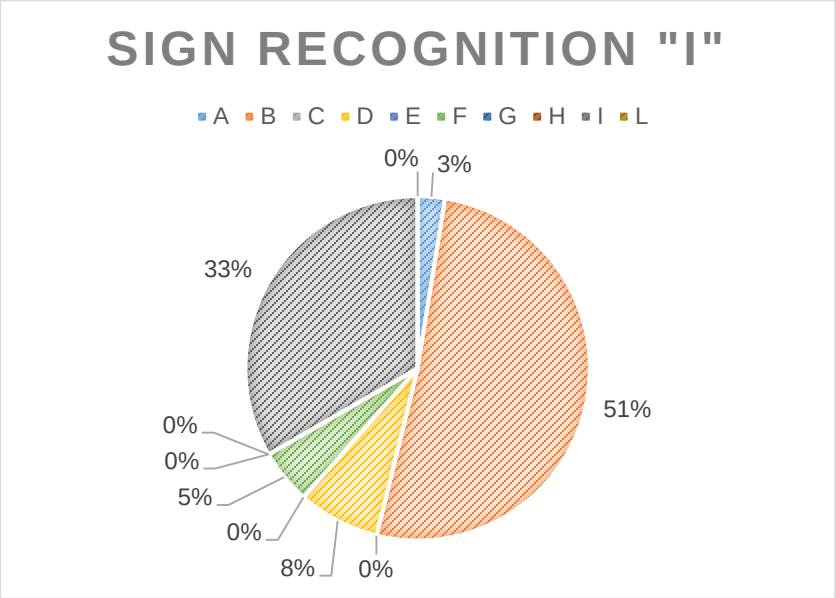


Figura 4.10. Resultados del sistema de reconocimiento de la letra “I”

En base a los análisis previos se puede definir que el mejor rendimiento del sistema en base a cada seña se encuentra en la letra “H” con un 90% de efectividad, ya que esta seña no afecta al reconocimiento de las demás señas, ocasionando que el sistema lo clasifique correctamente (Figura 4.11).

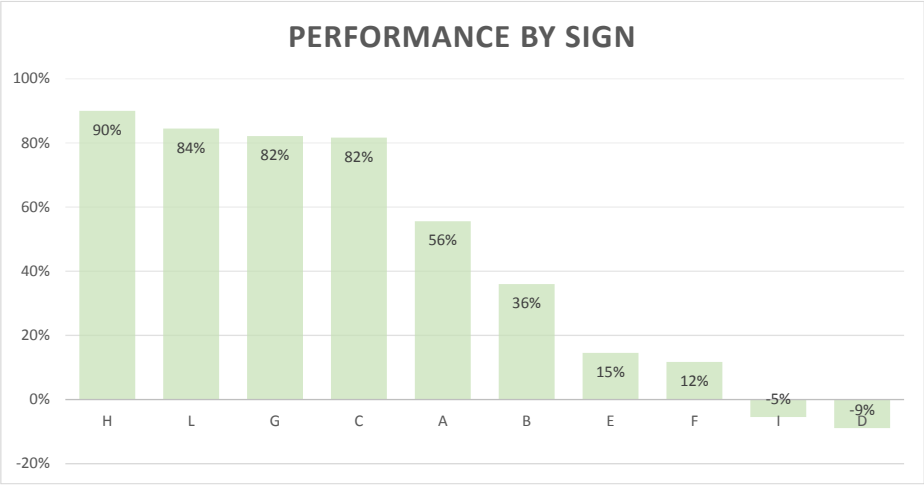


Figura 4.11. Rendimiento del sistema por seña

4.2 Conclusiones

En este proyecto se planteó una hipótesis, a la cual se le dará respuesta para cumplir de manera satisfactoria con la investigación.

“El empleo de inteligencia artificial en un traductor de lenguaje de señas basado en un sistema de visión, contribuirá al desarrollo de un sistema más robusto y menos sensible a perturbaciones en un 90%, lo cual contribuirá a desarrollar un sistema que permita la comunicación a nivel básico con personas que desconocen el lenguaje de señas.”

El desempeño del sistema intérprete obtuvo en el mejor de los casos un 98%, por lo que tiene un comportamiento óptimo al reconocer las letras B y G del alfabeto de la lengua de señas mexicana.

En comparación con el objetivo específico **“Validar el algoritmo con inteligencia artificial para el reconocimiento de imágenes”** se eligió el repositorio *FingerspellingRecognition* se implementó en el diseño de la red profunda *autoencoders* para el entrenamiento, su desempeño ha sido casi excelente, con una exactitud de 99.95% y una tasa de error del 0.01684.

Con respecto al objetivo **“Crear un conjuntos de datos de entrenamiento para la red neuronal”,** se realizó y se obtuvo una exactitud del 99.2% con una tasa de error del 0.14424.

En general se puede concluir que el comportamiento de la red neuronal AE de aprendizaje profundo, es efectiva para el reconocimiento de imágenes de profundidad capturadas por el sensor Kinect, que en este caso, funciona para identificar letras del alfabeto de la lengua de señas mexicana.

4.3 Mejoras y trabajos futuros

Como trabajo futuro se puede considerar complementar los siguientes pasos:

1) La cantidad de muestras para el entrenamiento. Incrementar este valor puede dar un mejor rendimiento en el resultado del análisis con las imágenes elaboradas con la cámara *Creative.*
2) Sujetos diferentes. Este factor puede ayudar a que la red profunda puede aprender más características que asocian a una misma clase y sea más precisa al momento de clasificar.
3) A demás de trabajar con capturas de imágenes, se puede considerar iniciar con el diseño de un intérprete de movimientos con señas, que utilice como base el sensor *Kinect* y la implementación del algoritmo de la red profunda AE para el reconocimiento de ideogramas del lenguaje de señas, que a diferencia de la dactilología, estas señas son con movimientos de la mano y gestos.

ANEXOS

Codigo C#

```
namespace Microsoft.Samples.Kinect.DepthBasics
{
    using System;
    using System.ComponentModel;
    using System.Diagnostics;
    using System.Globalization;
    using System.IO;
    using System.Windows;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using Microsoft.Kinect;

    /// <summary>
    /// Interaction logic for MainWindow
    /// </summary>
    public partial class MainWindow : Window, INotifyPropertyChanged
    {
        /// <summary>
        /// Map depth range to byte range
        /// </summary>
        private const int MapDepthToByte = 8000 / 256;

        /// <summary>
        /// Active Kinect sensor
        /// </summary>
        private KinectSensor kinectSensor = null;

        /// <summary>
        /// Reader for depth frames
        /// </summary>
        private DepthFrameReader depthFrameReader = null;

        /// <summary>
        /// Description of the data contained in the depth frame
        /// </summary>
        private FrameDescription depthFrameDescription = null;

        /// <summary>
        /// Bitmap to display
        /// </summary>
        private WriteableBitmap depthBitmap = null;

        /// <summary>
        /// Intermediate storage for frame data converted to color
        /// </summary>
        private byte[] depthPixels = null;
```

```
        /// <summary>
        /// Current status text to display
        /// </summary>
        private string statusText = null;

        /// <summary>
        /// Initializes a new instance of the MainWindow class.
        /// </summary>
        ///
        // Create the MATLAB instance
        MLApp.MLApp matlab = new MLApp.MLApp();

        // Variable used to store pictures in folder
        public static int inc = 0;
        public MainWindow()
        {
            // get the kinectSensor object
            this.kinectSensor = KinectSensor.GetDefault();

            // open the reader for the depth frames
            this.depthFrameReader = this.kinectSensor.DepthFrameSource.OpenReader();

            // wire handler for frame arrival
            this.depthFrameReader.FrameArrived += this.Reader_FrameArrived;

            // get FrameDescription from DepthFrameSource
            this.depthFrameDescription = this.kinectSensor.DepthFrameSource.FrameDescription;

            // allocate space to put the pixels being received and converted
            this.depthPixels = new byte[this.depthFrameDescription.Width *
this.depthFrameDescription.Height];

            // create the bitmap to display
            this.depthBitmap = new WriteableBitmap(this.depthFrameDescription.Width,
this.depthFrameDescription.Height, 96.0, 96.0, PixelFormats.Gray8, null);

            // set IsAvailableChanged event notifier
            this.kinectSensor.IsAvailableChanged += this.Sensor_IsAvailableChanged;

            // open the sensor
            this.kinectSensor.Open();

            // set the status text
            this.StatusText = this.kinectSensor.IsAvailable ?
Properties.Resources.RunningStatusText
                                        : Properties.Resources.NoSensorStatusText;

            // use the window object as the view model in this simple example
            this.DataContext = this;
```

```
            // initialize the components (controls) of the window
            this.InitializeComponent();
        }

        /// <summary>
        /// INotifyPropertyChangedPropertyChanged event to allow window controls to bind to
changeable data
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

        /// <summary>
        /// Gets the bitmap to display
        /// </summary>
        public ImageSource ImageSource
        {
            get
            {
                return this.depthBitmap;
            }
        }

        /// <summary>
        /// Gets or sets the current status text to display
        /// </summary>
        public string StatusText
        {
            get
            {
                return this.statusText;
            }

            set
            {
                if (this.statusText != value)
                {
                    this.statusText = value;

                    // notify any bound elements that the text has changed
                    if (this.PropertyChanged != null)
                    {
                        this.PropertyChanged(this, new PropertyChangedEventArgs("StatusText"));
                    }
                }
            }
        }

        /// <summary>
        /// Execute shutdown tasks
        /// </summary>
        /// <param name="sender">object sending the event</param>
        /// <param name="e">event arguments</param>
```

```
        private void MainWindow_Closing(object sender, CancelEventArgs e)
        {
            if (this.depthFrameReader != null)
            {
                // DepthFrameReader is IDisposable
                this.depthFrameReader.Dispose();
                this.depthFrameReader = null;
            }

            if (this.kinectSensor != null)
            {
                this.kinectSensor.Close();
                this.kinectSensor = null;
            }
        }

        /// <summary>
        /// Handles the user clicking on the screenshot button
        /// </summary>
        /// <param name="sender">object sending the event</param>
        /// <param name="e">event arguments</param>
        private void ScreenshotButton_Click(object sender, RoutedEventArgs e)
        {
            if (this.depthBitmap != null)
            {
                //// create a png bitmap encoder which knows how to save a .png file
                //BitmapEncoder encoder = new PngBitmapEncoder();

                //// create frame from the writable bitmap and add to encoder
                //encoder.Frames.Add(BitmapFrame.Create(this.depthBitmap));

                //string time = System.DateTime.UtcNow.ToString("hh'-'mm'-'ss",
CultureInfo.CurrentUICulture.DateTimeFormat);

                //string myPhotos =
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);

                //string path = Path.Combine(myPhotos, "KinectScreenshot-Depth-" + time + ".png");

                //****** NEW ******
                int Left = (int)ROI.Margin.Left;
                int Top = (int)ROI.Margin.Top;
                int Right = (int)ROI.Margin.Right;
                int Bottom = (int)ROI.Margin.Bottom;
                int Height = (int)ROI.Height;
                int Width = (int)ROI.Width;



                // create a png bitmap encoder which knows how to save a .png file
                BitmapEncoder encoder = new PngBitmapEncoder();
```

```
            CroppedBitmap cropImg = new CroppedBitmap(this.depthBitmap.Clone(), new
Int32Rect(Left - 75, Top, Width, Height));
            // create frame from the writable bitmap and add to encoder
            encoder.Frames.Add(BitmapFrame.Create(cropImg));

            if(true)
            {



                                                                        //string path =
Path.Combine(myPhotos, "Subject1-IR-" + time + ".png"); //please use for training images
            }
            else
            {

            }

            // uncomment for save pictures for training
            //string myPhotos =
"C:\\Users\\Ezequiel\\Documents\\MATLAB\\PRP\\Kinect_depthCaptures_Mar292019";
            //string path = Path.Combine(myPhotos, "S01_C10_" + inc.ToString() + ".png");

            // *****************
            // uncomment for evaluation
            string myPhotos = "C:\\Users\\Ezequiel\\Pictures\\Evaluate";

            string path = Path.Combine(myPhotos, "S01.png");                    //please use only
for validate
                                                        //string path = Path.Combine(myPhotos,
"Subject1-IR-" + time + ".png"); //please use for training images
            if (File.Exists(path))
                File.Delete(path);

            // write the new file to disk
            try
            {
                // FileStream is IDisposable
                using (FileStream fs = new FileStream(path, FileMode.Create))
                {
                    encoder.Save(fs);
                }

                /***** NEW *****/
                if(true)
                {
                    //imCapture.Uid = path;

                    //// Create the MATLAB instance
                    //MLApp.MLApp matlab = new MLApp.MLApp();
```

```
                // Change to the directory where the function is located
                matlab.Execute(@"cd C:\Users\Ezequiel\Documents\MATLAB\PRP");

                // Define the output
                object result = null;

                // Call the MATLAB function myfunc
                matlab.Feval("prpEvaluate", 1, out result, "S01");

                // Display result
                //object[] res =  result as object[];

                object[] res = (object[])result;

                var x = (double[,])res[0];

                if (x[0, 0] >= 0.95)
                   MessageBox.Show("Letter #A");// Letter #1
                else if (x[1, 0] >= 0.95)
                   MessageBox.Show("Letter #B");// Letter #2
                else if (x[2, 0] >= 0.95)
                   MessageBox.Show("Letter #C");// Letter #3
                else if (x[3, 0] >= 0.95)
                   MessageBox.Show("Letter #D");// Letter #4
                else if (x[4, 0] >= 0.95)
                   MessageBox.Show("Letter #E");// Letter #5
                else if (x[5, 0] >= 0.95)
                   MessageBox.Show("Letter #F");// Letter #2
                else if (x[6, 0] >= 0.95)
                   MessageBox.Show("Letter #G");// Letter #3
                else if (x[7, 0] >= 0.95)
                   MessageBox.Show("Letter #H");// Letter #4
                else if (x[8, 0] >= 0.95)
                   MessageBox.Show("Letter #I");// Letter #5
                else if (x[9, 0] >= 0.95)
                   MessageBox.Show("Letter #L");// Letter #4

                                        /*************************/
            }

            inc++;  //increment picture counter

            this.StatusText = string.Format(CultureInfo.CurrentCulture,
Properties.Resources.SavedScreenshotStatusTextFormat, path);
        }
        catch (IOException)
        {
            this.StatusText = string.Format(CultureInfo.CurrentCulture,
Properties.Resources.FailedScreenshotStatusTextFormat, path);
        }
```

```
        }
    }

    /// <summary>
    /// Handles the depth frame data arriving from the sensor
    /// </summary>
    /// <param name="sender">object sending the event</param>
    /// <param name="e">event arguments</param>
    private void Reader_FrameArrived(object sender, DepthFrameArrivedEventArgs e)
    {
        bool depthFrameProcessed = false;

        using (DepthFrame depthFrame = e.FrameReference.AcquireFrame())
        {
            if (depthFrame != null)
            {
                // the fastest way to process the body index data is to directly access
                // the underlying buffer
                using (Microsoft.Kinect.KinectBuffer depthBuffer = depthFrame.LockImageBuffer())
                {
                    // verify data and write the color data to the display bitmap
                    if (((this.depthFrameDescription.Width * this.depthFrameDescription.Height) ==
(depthBuffer.Size / this.depthFrameDescription.BytesPerPixel)) &&
                        (this.depthFrameDescription.Width == this.depthBitmap.PixelWidth) &&
(this.depthFrameDescription.Height == this.depthBitmap.PixelHeight))
                    {
                        // Note: In order to see the full range of depth (including the less reliable far
field depth)
                        // we are setting maxDepth to the extreme potential depth threshold
                        ushort maxDepth = ushort.MaxValue;

                        // If you wish to filter by reliable depth distance, uncomment the following line:
                        //// maxDepth = depthFrame.DepthMaxReliableDistance

                        this.ProcessDepthFrameData(depthBuffer.UnderlyingBuffer, depthBuffer.Size,
depthFrame.DepthMinReliableDistance, maxDepth);
                        depthFrameProcessed = true;
                    }
                }
            }
        }
```

```
            if (depthFrameProcessed)
            {
                this.RenderDepthPixels();
            }
        }

        /// <summary>
        /// Directly accesses the underlying image buffer of the DepthFrame to
        /// create a displayable bitmap.
        /// This function requires the /unsafe compiler option as we make use of direct
        /// access to the native memory pointed to by the depthFrameData pointer.
        /// </summary>
        /// <param name="depthFrameData">Pointer to the DepthFrame image data</param>
        /// <param name="depthFrameDataSize">Size of the DepthFrame image data</param>
        /// <param name="minDepth">The minimum reliable depth value for the frame</param>
        /// <param name="maxDepth">The maximum reliable depth value for the frame</param>
        private unsafe void ProcessDepthFrameData(IntPtr depthFrameData, uint
depthFrameDataSize, ushort minDepth, ushort maxDepth)
        {
            // depth frame data is a 16 bit value
            ushort* frameData = (ushort*)depthFrameData;

            // convert depth to a visual representation
            for (int i = 0; i < (int)(depthFrameDataSize / this.depthFrameDescription.BytesPerPixel);
++i)
            {
                // Get the depth for this pixel
                ushort depth = frameData[i];

                // To convert to a byte, we're mapping the depth value to the byte range.
                // Values outside the reliable depth range are mapped to 0 (black).
                this.depthPixels[i] = (byte)(depth >= minDepth && depth <= maxDepth ? (depth /
MapDepthToByte) : 0);
            }
        }

        /// <summary>
        /// Renders color pixels into the writeableBitmap.
        /// </summary>
        private void RenderDepthPixels()
        {
            this.depthBitmap.WritePixels(
                new Int32Rect(0, 0, this.depthBitmap.PixelWidth, this.depthBitmap.PixelHeight),
                this.depthPixels,
                this.depthBitmap.PixelWidth,
                0);
        }

        /// <summary>
```

```
        /// Handles the event which the sensor becomes unavailable (E.g. paused, closed,
unplugged).
        /// </summary>
        /// <param name="sender">object sending the event</param>
        /// <param name="e">event arguments</param>
        private void Sensor_IsAvailableChanged(object sender, IsAvailableChangedEventArgs e)
        {
            // on failure, set the status text
            this.StatusText = this.kinectSensor.IsAvailable ?
Properties.Resources.RunningStatusText
                                            : Properties.Resources.SensorNotAvailableStatusText;
        }
    }
}
```

Codigo Matlab

```
clear all; close all; clc;

%% Training a Deep Neural Network for Digit Classification
% This example shows how to use Neural Network Toolbox(TM) to train a deep
% neural network to classify images of digits.
%
% Neural networks with multiple hidden layers can be useful for solving
% classification problems with complex data, such as images. Each layer can
% learn features at a different level of abstraction. However, training
% neural networks with multiple hidden layers can be difficult in practice.
%
% One way to effectively train a neural network with multiple layers is by
% training one layer at a time. You can achieve this by training a special
% type of network known as an autoencoder for each desired hidden layer.
%
% This example shows you how to train a neural network with two hidden
% layers to classify digits in images. First you train the hidden layers
% individually in an unsupervised fashion using autoencoders. Then you
% train a final *softmax* layer, and join the layers together to form a deep
% network, which you train one final time in a supervised fashion.
%
% Copyright 2014-2015 The MathWorks, Inc.

%% Data set
% This example uses synthetic data throughout, for training and testing.
% The synthetic images have been generated by applying random affine
% transformations to digit images created using different fonts.
%
% Each digit image is 28-by-28 pixels, and there are 5,000 training
% examples. You can load the training data, and view some of the images.

% Load the training data into memory
%[xTrainImages, tTrain] = digittrain_dataset;

% ver Table 1 en el paper
% en el primer experimento, el autor usa un solo sujeto, asi que aqui
% se puede cargar datos de sujetos individuales y al final promediar el
% desempeno de todos los 5 sijetos. Empezamos con S01 (el primer sujeto)
[X_input, y_target] = prpGetData3('S01');
%%

% el autor usa 50% para entrenamiento, 25% para validacion, y 25% para
% pruebas, osea (3100/1550/1550) muestras. aqui empezamos crossvalidation
% para obtener el error en entrenamiento, prueba, y validacion crosvalidado
% como Dios manda.
[idxTrain, idxRest] = crossvalind('HoldOut', length(X_input), 0.5); % 50%
xTrainImages = X_input(idxTrain);  tTrain = y_target(:, idxTrain);
X_rest = X_input(idxRest); y_rest = y_target(:, idxRest);

[idxTest, idxVal] = crossvalind('HoldOut', length(X_rest), 0.5);
xTestImages = X_rest(idxTest); tTest = y_rest(:, idxTest); % 25%
xValImages = X_rest(idxVal); tVal = y_rest(:, idxVal); % 25%
clear X_rest y_rest idx*
```

```
% Display some of the training images
%clf    %PLEASE UNCOMMENT
% for i = 1:20
%     subplot(4,5,i);
%     imshow(xTrainImages{i});
% end
% ********* DISPLAY ALL IMAGES *********
% for i = 1:45
%     subplot(5,9,i);
%     imshow(xTrainImages{i});
% end

%%
% The labels for the images are stored in a 10-by-5000 matrix, where in
% every column a single element will be 1 to indicate the class that the
% digit belongs to, and all other elements in the column will be 0. It
% should be noted that if the tenth element is 1, then the digit image is a
% zero.

%% Training the first autoencoder
% Begin by training a sparse autoencoder on the training data without using
% the labels.
%
% An autoencoder is a neural network which attempts to replicate its input
% at its output. Thus, the size of its input will be the same as the size
% of its output. When the number of neurons in the hidden layer is less
% than the size of the input, the autoencoder learns a compressed
% representation of the input.
%
% Neural networks have weights randomly initialized before training.
% Therefore the results from training are different each time. To avoid
% this behavior, explicitly set the random number generator seed.
rng('default')

%%
% Set the size of the hidden layer for the autoencoder. For the autoencoder
% that you are going to train, it is a good idea to make this smaller than
% the input size.
hiddenSize1 = 100;

%%
% The type of autoencoder that you will train is a sparse autoencoder. This
% autoencoder uses regularizers to learn a sparse representation in the
% first layer. You can control the influence of these regularizers by
% setting various parameters:
%
% * |L2WeightRegularization| controls the impact of an L2 regularizer for
% the weights of the network (and not the biases). This should typically be
% quite small.
% * |SparsityRegularization| controls the impact of a sparsity regularizer,
% which attempts to enforce a constraint on the sparsity of the output from
% the hidden layer. Note that this is different from applying a sparsity
% regularizer to the weights.
% * |SparsityProportion| is a parameter of the sparsity regularizer. It
% controls the sparsity of the output from the hidden layer. A low value
% for |SparsityProportion| usually leads to each neuron in the hidden layer
```

```
% "specializing" by only giving a high output for a small number of
% training examples. For example, if |SparsityProportion| is set to 0.1,
% this is equivalent to saying that each neuron in the hidden layer should
% have an average output of 0.1 over the training examples. This value must
% be between 0 and 1. The ideal value varies depending on the nature of the
% problem.
%
% Now train the autoencoder, specifying the values for the regularizers
% that are described above.

fprintf('Training autoencoder1\n');
autoenc1 = trainAutoencoder(xTrainImages,hiddenSize1, ...
    'MaxEpochs',500, ...
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
    'ScaleData', false, ...
    'ShowProgressWindow',false);

save AUT1 autoenc1
%%
% You can view a diagram of the autoencoder. The autoencoder is comprised
% of an encoder followed by a decoder. The encoder maps an input to a
% hidden representation, and the decoder attempts to reverse this mapping
% to reconstruct the original input.

%view(autoenc1)

%% Visualizing the weights of the first autoencoder
% The mapping learned by the encoder part of an autoencoder can be useful
% for extracting features from data. Each neuron in the encoder has a
% vector of weights associated with it which will be tuned to respond to a
% particular visual feature. You can view a representation of these
% features.



%%
% You can see that the features learned by the autoencoder represent curls
% and stroke patterns from the digit images.
%
% The 100-dimensional output from the hidden layer of the autoencoder is a
% compressed version of the input, which summarizes its response to the
% features visualized above. Train the next autoencoder on a set of these
% vectors extracted from the training data. First, you must use the encoder
% from the trained autoencoder to generate the features.

feat1 = encode(autoenc1,xTrainImages);

%% Training the second autoencoder
% After training the first autoencoder, you train the second autoencoder in
% a similar way. The main difference is that you use the features that were
% generated from the first autoencoder as the training data in the second
% autoencoder. Also, you decrease the size of the hidden representation to
% 50, so that the encoder in the second autoencoder learns an even smaller
% representation of the input data.
```

```
fprintf('Training autoencoder2\n');
hiddenSize2 = 50;
autoenc2 = trainAutoencoder(feat1,hiddenSize2, ...
    'MaxEpochs',100, ...
    'L2WeightRegularization',0.002, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.1, ...
    'ScaleData', false, ...
    'ShowProgressWindow',false);


save AUT2 autoenc2
%%
% Once again, you can view a diagram of the autoencoder with the
% |view| function.

%view(autoenc2)

%%
% You can extract a second set of features by passing the previous set
% through the encoder from the second autoencoder.

feat2 = encode(autoenc2,feat1);

%%
% The original vectors in the training data had 784 dimensions. After
% passing them through the first encoder, this was reduced to 100
% dimensions. After using the second encoder, this was reduced again to 50
% dimensions. You can now train a final layer to classify these
% 50-dimensional vectors into different digit classes.

%% Training the final softmax layer
% Train a softmax layer to classify the 50-dimensional feature vectors.
% Unlike the autoencoders, you train the softmax layer in a supervised
% fashion using labels for the training data.
fprintf('Training softmax network\n');
softnet = trainSoftmaxLayer(feat2,tTrain,'MaxEpochs',500);

%%
% You can view a diagram of the softmax layer with the |view| function.

%view(softnet)

%% Forming a stacked neural network
% You have trained three separate components of a deep neural network in
% isolation. At this point, it might be useful to view the three neural
% networks that you have trained. They are |autoenc1|, |autoenc2|, and
% |softnet|.

%view(autoenc1)
%view(autoenc2)
%view(softnet)
```

```
%%
% As was explained, the encoders from the autoencoders have been used to
% extract features. You can stack the encoders from the autoencoders
% together with the softmax layer to form a deep network.

deepnet = stack(autoenc1,autoenc2,softnet);

save DEEP1 deepnet
%%
% You can view a diagram of the stacked network with the |view| function.
% The network is formed by the encoders from the autoencoders and the
% softmax layer.
view(deepnet)

%%
% With the full deep network formed, you can compute the results on the
% test set. To use images with the stacked network, you have to reshape the
% test images into a matrix. You can do this by stacking the columns of an
% image to form a vector, and then forming a matrix from these vectors.

% Get the number of pixels in each image
% imageWidth = 28;
% imageHeight = 28;
imageWidth = 115;
imageHeight = 151;
inputSize = imageWidth*imageHeight;

% Turn the test images into vectors and put them in a matrix
xTest = zeros(inputSize,numel(xTestImages));
for i = 1:numel(xTestImages)
    xTest(:,i) = xTestImages{i}(:);
end

%%
% You can visualize the results with a confusion matrix. The numbers in the
% bottom right-hand square of the matrix give the overall accuracy.

%y = deepnet(xTest);
%plotconfusion(tTest,y);

%% Fine tuning the deep neural network
% The results for the deep neural network can be improved by performing
% backpropagation on the whole multilayer network. This process is often
% referred to as fine tuning.
%
% You fine tune the network by retraining it on the training data in a
% supervised fashion. Before you can do this, you have to reshape the
% training images into a matrix, as was done for the test images.

% Turn the training images into vectors and put them in a matrix
xTrain = zeros(inputSize,numel(xTrainImages));
for i = 1:numel(xTrainImages)
    xTrain(:,i) = xTrainImages{i}(:);
end
fprintf('Training final deep network\n');
```

```
% Perform fine tuning
%deepnet = train(deepnet,xTrain,tTrain,'ShowProgressWindow',false);
%deepnet = train(deepnet,xTrain,tTrain,'useParallel','yes','useGPU','yes');
deepnet = train(deepnet,xTrain,tTrain);

save DEEP2 deepnet
%
% You then view the results again using a confusion matrix.

% error en entrenamiento
y = deepnet(xTrain);
fprintf('Training Error\n');
prpGetMetrics(y, tTrain);
plotconfusion(tTrain,y);
save Train_sub1 y tTrain

% error en test set
y = deepnet(xTest);
fprintf('Testing Error\n');
prpGetMetrics(y, tTest);
plotconfusion(tTest,y);
save Test_sub1 y tTest

%% Y al final hacemos la validacion
% Turn the test images into vectors and put them in a matrix
xVal = zeros(inputSize,numel(xValImages));
for i = 1:numel(xValImages)
    xVal(:,i) = xValImages{i}(:);
end

y = deepnet(xVal);
fprintf('Validation Error\n');
prpGetMetrics(y, tVal);
plotconfusion(tVal,y);
save Val_sub1 y tVal
```

Referencias

(s.f.). Obtenido de MathWorks: https://la.mathworks.com/help/matlab/learn_matlab/product-description.html

Aldrete, M. (2009). Caracterizacion de la Lengua de Señas Mexicana.

Alnowami, M., Alnwaimi, B., Tahavori, F., Copland, F., & Wells, K. (2012). A Quantitative Assessment of Using the Kinect for Xbox360 for Respiratory Surface Motion Tracking. In Proceedings of the Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, and Modeling.

Alonso, A., & Díaz, E. (2008). Universidad y discapacidad: indicadores de buenas prácticas y estándares de actuación para programas y servicios. SIGLOCERO Rev. .Esp.Discapacidad Intelectual.

Álvarez Torres, N. A., Vázquez Cuevas, I. J., & Velarda Anaya, O. (2017). METODOLOGÍA PARA ESTANDARIZAR RECONOCIMIENTO DE DACTILOLOGIA EN LENGUA DE SEÑAS MEXICANA MEDIANTE UN GUANTE MULTISENSORIAL.

Bouchard, G. (2008). Efficient Bounds for the Softmax Function and Applications to Approximate Inference in Hybrid models.

Clerc, M. (2010). *Particle Swarm Optimization.* ISTE.

Coates, A., Lee, H., & Ng, A. (2010). An Analysis of Single-Layer Networks in Unsupervised Feature Learning.

Córdova Lucero, F. A. (Diciembre 2002). *DETECCIÓN DE ROBO/ABANDONO DE OBJETOS DE INTERIORES UTILIZANDO CAMARAS DE PROFUNDIDAD.*

Diaz Celis, C. A., & Romero Molano, C. A. (2012). Navegación de robot móvil usando Kinect, OpenCV y Arduino.

El-Iaithy, R., Jidong, H., & Yeh, M. (2012). Study on the Use of Microsoft Kinect for Robotics Applications. In Proceedings of the Position Location and Navigation Symposium (PLANS).

Escamilla Bouchan, I., & Luna Soto, V. (2018). *Revista de Prototipos Tecnológicos.* ECORFAN.

García Cambronero, C., & Gómez Moreno, I. (2006). ALGORITMOS DE APRENDIZAJE: KNN & KMEANS.

Gokturk, S. B., Yalcin, H., & Bamji, C. (2005). A Time-Of-Flight Depth Sensor – System Description, Issues and Solutions.

Hernández, M.R, Rodríguez, V.M, Parra, F.J., & Vel. (2014). Las Tecnologías de la Información y la Comunicación (TICs) en la Enseñanza-Aprendizaje de la Química Orgánica a Través de Imágenes, Juegos y Video.

Humphries, T., Kushalnagar, P., Mathur, G., Napoli, D., Padden, C., Rathmann, C., & Smith, S. (2012). Language acquisition for deaf children: Reducing the harms of zero tolerance to the use of alternative approaches.

Jaramillo González, G. F. (2009). REDES NEURONALES APLICADAS AL ANÁLISIS DE IMÁGENES PARA EL DESARROLLO DE UN PROTOTIPO DE UN SISTEMA DE SEGURIDAD.

Jockey, P. (1989). Stuffed Bear as teaching aid. U.S Patent 4799889.

Kang, B., Tripathi, S., & Nguyen, T. Q. (2015). Real-time Sign Language Fingerspelling Recognition using.

Kramer, J. P., Lindener, P., & George, W. R. (1988). Communication system for deaf, deaf-blind, or non-vocal individuals using instrumented glove U.S. Patent 5047952.

Lachat, E., Macher, E., Landes, T., & Grussenmeyer, P. (2015). *Assessment and Calibration of a RGB-D Camera (Kinect v2 Sensor) Towards a Potential Use for Close-Range 3D Modeling.*

Laenger, McFarland, C. J., R., S., & Peel, H. H. (1976). Method and apparatus for communicating with people U.S. Patent 4074444.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Macmillan.

Ledesma, S. (2006). Las Redes Neuronales implementación y consideraciones prácticas. Fifth Mexican International Conference on Artificial Intelligence.

Leybon Ibarra, J., Ramírez Barba, M., & Taboada Picazo, V. (2006). SENSor Foto-Eléctrico Aplicado al Movimiento de los Dedos de las Manos.

Martínez Álvarez, R. (2017). *Análisis de las máquinas "Sparse Autoencoders" como extractores de características.*

McPherson, G. (s.f.). Obtenido de Creative, Technology, Design & Code: http://blog.nelga.com/setup-microsoft-kinect-on-mac-os-x-10-9-mavericks/

Mejía Vilet, J. (2005). Procesamiento Digital de Imágenes.

Mikolov, T. (2010). Recurrent neural network based language model.

Munoz Perez, J. (2012). Redes Neuronales artificiales.

Ng, A. (2011). Sparse autoencoder. *CS294A Lecture notes [online]*.

Núñez, M. (2008). *Procesamiento de imágenes en Medicina Nuclear.* Montevideo, Uruguay.

Oikonomidis, I., Kyriazis, N., Padeleris, P., & Argyros, A. (2011). Efficient Model-based 3D Tracking of Hand Articulations using Kinect.

Pagliari, D., & Pinto, L. (2015). Calibration of Kinect for Xbox One and Comparison between the Two Generations of Microsoft Sensors.

Pedemonte, M., & Narins, P. (1999). Las células ciliadas de la cóclea, un ejemplo de transducción bidireccional.

Perez Perez, E. (2017). *Diseño de una metodología para el procesamiento de imagenes mamograficas basada en técnicas de aprendizaje profundo.*

Regularización para lograr dispersión: Regularización L_1. (3 de May de 2018). Obtenido de Aprendizaje Automatico: https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization?hl=es-419

Rivas Perea, P. (2011). Perfomance Metrics. En *ALGORITHMS FOR TRAINING LARGE-SCALE LINEAR PROGRAMMING SUPPORT VECTOR REGRESSION AND CLASSIFICATION* (págs. 271-275).

Sapora, R. (1980). Educational apparatus and method for control of deaf individuals in a mixed teaching environment. U.S. Patent 4368459.

Schindhelm, C. (2012). Evaluating SLAM approach for Microsoft Kinect. In Proceedings of the 8th International Conference on Wireless and Mobile Communications.

Serafín de Fleischmann, M., & González Pérez, R. (2011). Diccionario de Lengua de Señas Mexicana.

Sordera y pérdida de la audición. (15 de Mazo de 2019). Obtenido de Organización Mundial de la Salud: https://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss

Sotolongo, G., & Guzman, M. V. (2009). *Aplicaciones de las redes neuronales. El caso de la Bibliometría.*

Sparks, W. B. (1981). Educational device especially for the blind and the blind-deaf. U.S. Patent 4378215.

Tashev, I. (2011). Recent Advances in Human-Machine Interfaces for Gaming and Entertainment.

Tovar L. (1999). La lengua escrita como segunda lengua para el niño sordo. Revista El Bilingüismo de los Sordos.

Verma, B., & Blumenstein, M. (2008). Pattern Recognition Technologies and Aplicaions: Recent Advance. USA: IGI Global.

Zhang , Z. (2012). Microsoft Kinect Sensor and Its Effect.