UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO
ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS
FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ARQUITECTURA DE INTERPRETACIÓN DE EXPRESIONES COMUNES DE LA LENGUA DE SEÑAS DEL PERÚ AL IDIOMA ESPAÑOL

Para optar el título profesional de:
INGENIERO INFORMÁTICO Y DE SISTEMAS

Presentado por:
Br. Naysha Naydu Diaz Ccasa
Br. Yuri Vladimir Huallpa Vargas
Asesor:
Dr. Lauro Enciso Rodas

CUSCO - PERÚ
2019

Esta investigación esta dedicada a las futuras generaciones y nuestros compañeros con el objetivo de ayudarlos a comprender mejor este campo e incentivarlos

Agradecimientos

Deseo agradecer a mis padres Juan Ramón y Luisa que sin ellos nada de lo que he logrado seria posible, a mi hermana que siempre esta a mi lado apoyandome en todo.

- Naysha Naydu Diaz Ccasa

Gracias a mis padres Filomena y Damian por brindarme siempre su apoyo incondicional, a mis hermanos que me motivan a seguir adelante y ser una mejor persona.

- Yuri Vladimir Huallpa Vargas

Un agradecimiento especial a nuestro asesor Dr. Lauro Enciso Rodas que tuvo gentileza de apoyarnos e incentivarnos. Finalmente un agradecimiento general a todos nuestros amigos y compañeros que se tomaron un tiempo para ayudarnos en la creación de la base de datos para nuestro proyecto.

Resumen

La lengua de señas se percibe a través de la vista y requiere el uso de la cabeza, cuello, torso y brazos para transmitir información bajo un espacio temporal. Como cualquier otra lengua el LSP está conformado por una sintaxis, gramática y léxico diferentes del idioma oficial. El 2003 se propuso la iniciativa de educación inclusiva para personas sordas, pero no tuvo un efecto, posteriormente el ministerio de educación MINEDU, cambio el panorama y la ley 29535 dio su reconocimiento a la lengua de señas para la investigación, difusión y enseñanza para personas sordas por interpretes acreditados. Sin embargo actualmente el LSP se encuentra dentro de las lenguas minoritarias del Perú según la Dirección General de Educación Básica Especial las personas con discapacidad auditiva se ven en la necesidad de aprender esta lengua para interactuar en la sociedad a diferencia del resto de personas que no sufren de esta discapacidad y no tienen la necesidad de aprender esta lengua, por lo que se crea una barrera en la comunicación, pese a las legislaciones del estado es muy común ver la indiferencia a esta comunidad, ya sea voluntaria o involuntariamente. Mediante técnicas de Deep Learning[1] se facilita la interpretación del LSP y con una mejora en la tasa de precisión[2] frente a modelos similares, se construye un traductor unidireccional que permita captar las señas de una persona con un dispositivo e interpretarlas en nuestro idioma. Por otro lado, se genera un *dataset* de vídeos de 10 señas almacenados en 100 *frames* aproximadamente cada uno. El modelo de solución alimenta a la arquitectura con datos generados por un sensor Kinect, el sensor es capaz de generar un vídeo compuesto por tres tipos de datos: *frames* RGB, *Depth*[3] y *Skeleton*[4], los datos son agrupados según el modelo para extraer las características de cada *frame* y posteriormente alimentan la parte recurrente encargada de la traducción. Finalmente, nuestro modelo propuesto obtuvo una tasa de exactitud de 99.23 %, una tasa muy aceptable que contribuirá a futuros trabajos dentro de este campo.

Palabras Clave

Lengua de señas, deep learning, redes recurrentes, sensor Kinect.

[1]Tipo de *machine learning*, una técnica que permite que los sistemas informáticos mejoren con la experiencia y los datos (Goodfellow et al., 2016).

[2]Indicador que determina que proporción de identificaciones positivas fue correcta.

[3]Datos tomados por el sensor infrarrojo del Kinect, representan distancias

[4]Datos que contienen un esquema del esqueleto con los puntos de movimiento que se detectan.

Abstract

Sign language is perceived through sight and requires the use of the head, neck, torso and arms to transmit information under a temporal space. Like any other language, the LSP is made up of a syntax, grammar and lexicon different from the official language. In 2003 the initiative of inclusive education for deaf people was proposed, but did not have an effect, later the Ministry of Education MINEDU changed the landscape and Law 29535 gave its recognition to sign language for research, dissemination and teaching for deaf people by accredited interpreters. However, according to the General Directorate of Special Basic Education, at present the LSP is within the minority languages of Peru. According to the General Directorate of Special Basic Education, people with hearing disabilities see the need to learn this language in order to interact in society, unlike the rest of people who do not suffer from this disability and do not have the need to learn this language. Deep Learning[1] techniques facilitate the interpretation of the LSP and with an improvement in the accuracy rate[2] compared to similar models, a unidirectional translator is built to capture the signs of a person with a device and interpret them into our language. On the other hand, a textdataset is generated from videos of 10 signs stored in approximately 100 textframes each. The solution model feeds the architecture with data generated by a Kinect sensor, the sensor is capable of generating a video composed of three types of data: *frames* RGB, *Depth*[3] and *Skeleton*[4], the data is grouped according to the model to extract the characteristics of each *frame* and then feeds the recurrent part in charge of the translation. Finally, our proposed model obtained an accuracy rate of 99.23 %, a very acceptable rate that will contribute to future work in this field.

Keywords

Sign language, deep learning, recurrent networks, sensor Kinect.

[1]Type of textmachine learning, a technique that allows computer systems to improve with experience and data (Goodfellow et al., 2016).

[2]Indicator that determines what proportion of positive identifications was correct.

[3]Data taken by the infrared sensor of the Kinect, represent distances

[4]Data containing an outline of the skeleton with the movement points detected.

Índice general

Índice de figuras

Índice de Tablas

Nomenclatura

BLSTM Bidirectional Long short-term memory

BRNN Bidirectional recurrent neural networks

CE Cross entropy

CMOS Complementary metal-oxide-semiconductor o semiconductor complementario de óxido metálico

CNN Convolutional neural network o ConvNet

DIGEVE Dirección general de educación básica Especial

DL Deep learning, aprendizaje profundo

DNN Deep neural network

FC Fully connected, capas completamente conectadas

fps Frames per second, fotogramas por segundo

GD Gradient descent

GLM Generalized linear model

GRU Gated recurrent units

IA Inteligencia artificial

IR infrared, infrarojo

KAF Kernel activation functions

LR Logistic regression, regresion logistica

LRelU Leaky ReLU

LSP Lengua de señas del Peru

LSTM Long short-term memory

ML Machine learning

MLP Multi-layer perceptron

MLPCONV Multi-layer perceptron despues de una capa convolucional

NIN Network in network

NLP Natural language processing

NN Neural network, red neuronal

NUI Natural user interfaces

ReLU Rectified Linear Unit

RGB Red, green, blue, en español rojo, verde y azul

RMSProp Root mean square propagation, propagación de la media cuadrática de la raíz

RNN Recurrent neural network

SGD Stochastic gradient descent

SVM Support vector machine, maquina de vector soporte

TL Transfer learning

TOF Time of flight

Parte I

Generalidades

Capítulo 1

Aspectos generales

1.1. Problema de investigación

1.1.1. Descripción del problema

En la actualidad existen muchas modelos DL (*Deep Learning*) que se ven cada vez más en la vida cotidiana, desde el uso de nuestros celulares y como accedemos a ellos utilizando parámetros biométricos como detección de rostro o huella digital, tomarnos una foto y que un programa cualquiera detecte automáticamente el rostro para aplicarle modificaciones estéticas o una traducción de un idioma a otro en segundos, muchas de las interpretaciones y traducciones están basadas en el uso de NLP (*Natural Language Processing*), término colectivo que se refiere al procesamiento computacional automático del lenguaje humano, es un gran desafío, ya que el lenguaje humano es ambiguo y siempre cambiante por lo que requiere un enfoque algorítmico más estadístico, los enfoques dominantes actuales del NLP se basan mayormente en el *Machine Learning* (Goldberg and Hirst, 2017), aunque recientemente también con modelos DL, la dificultad existente aquí es la interpretación de gestos humanos, un vídeo compuesto por *frames*[1], es equivalente a una traducción de un conjunto de imágenes del LSP a texto en el idioma español.

En el Perú 532 mil personas sufren de discapacidad auditiva y 262 mil con limitaciones permanentes para poder hablar (INEI, 2015), aunque representan un grupo pequeño de la sociedad es su derecho no ser excluidos por lo que es necesario destruir esa barrera de comunicación que existe. Una lengua de señas es creada por la propia comunidad sorda con sus reglas gramaticales, su estructura individual específica y diferentes principios con un léxico distinto, en el LSP utiliza la estructura de sujeto-objeto-verbo, en cambio el español usa el sujeto-verbo-objeto (Navarro, 2015). Para desarrollar un intérprete de la lengua de señas no basta con una red neuronal simple de clasificación pasando como entrada un gesto (palabra) y su respectiva etiqueta(traducción de la palabra) ya que la comunicación humana suele ser muy compleja, la lengua de señas que es un medio de comunicación para personas con discapacidad auditiva no es la excepción, por lo que se necesita conocer el significado de los gestos en conjunto para expresar una idea, es equivale a traducir un dialogo de un idioma a otro, una buena traducción no se realiza traduciendo palabra por palabra del dialogo, igualmente la interpretación de la

[1]Cuadros de vídeo son las mínimas imágenes completas que se tienen de una secuencia de vídeo

Figura 1.1: Ejemplo de sistemas inteligentes.
Fuente: Sotos (2017)

lengua de señas requiere la captura de movimientos y configuración de manos[2] para una buena interpretación. Al tratarse de una lengua de señas no todos los gestos son estáticos muchos dependen del movimiento que se realice y las partes del cuerpo involucradas, así dos palabras pueden comenzar con un movimiento similar y terminar de diferente manera lo que eleva un poco el nivel de complejidad que necesitara el intérprete para relacionar los *frames* de una seña móvil y diferenciarlos de otra similar, también dentro de toda lengua existen sinónimos en este caso del LSP dos señas completamente diferentes pueden significar lo mismo. Para poder interpretar las distintas señas del LSP se debe considerar el tipo de entrada o datos, que puede capturar el intérprete, tanto la vista humana como una cámara tienen limitaciones, como la posición o superposición de las manos que no permite la detección por el ángulo de visión, a diferencia de la visión humana una cámara no logrará captar al cien por ciento los movimientos, se verá limitada por diversos factores como ruidos de *salt-and-pepper*[3], desenfoque de imagen, exceso de luz u oscuridad, etc. Aumentando la dificultad para distinguir la posición de los dedos ocasionando una mala interpretación.
Finalmente, para poder construir una arquitectura que alcance una tasa de acierto aceptable en base al estado del arte, es necesario la identificación de técnicas de DNN (*Deep Neural Network*) y NLP, para la confección de la arquitectura, los modelos como CNN (*Convolutional Neural Network*) y RNN (*Recurrent Neural Network*) son básicos en la construcción de intérpretes, pero para obtener una

[2]Es la forma en que se coloca la(s) mano(s) al realizar una seña, o las posiciona para expresar alguna palabra (DIGEBE, 2015)

[3]Este tipo de ruido suele producirse cuando la señal de la imagen es afectada por intensas y repentinas perturbaciones o impulsos

tasa de exactitud aceptable es necesario evaluar técnicas de optimización, que se describirán más adelante en los próximos capítulos y realizar evaluaciones que muestren la eficiencia de la arquitectura propuesta.

1.1.2. Formulación del problema

¿Se puede construir una arquitectura para solucionar el problema de la comunicación entre el LSP y el idioma español?

1.2. Antecedentes

Durante la última década los investigadores han prestado mucha importancia al NLP, al reconocimiento de acciones y traducción de un idioma a otro. Se ha contribuido con una gran variedad de proyectos que intentan resolver y superar el estado del arte, en décadas pasadas el poder de cómputo y la necesidad de grandes cantidades de datos fueron las limitantes para realizar estas tareas.

1. En (Ullah et al., 2018) se realiza el reconocimiento de acciones sobre tres *dataset UCF101*[4], *HMDB51*[5] y *YouTube actions*[6], cada *dataset* se divide en 3 conjuntos, *training*, *testing* y *valitation* de 60, 20 y 20 % respectivamente, consiguiendo superar el estado de arte en los 3 *dataset* con 91.21 %, 87.64 % y 92.84 % respectivamente.

    - Se hace uso de la arquitectura convolucional *AlexNet* para la extracción de características de un conjunto de *frames* tomados cada 1 segundo y omitidos cada 6 *frames*.
    - Las salidas de cada CNN son tomados por una red *Bidirectional-LSTM* (*Long Short Term Memory*) con dos capas para analizar la secuencia de patrones ocultos en el espacio y tiempo, el estado final de cada intervalo de tiempo es analizado para el reconocimiento de cada acción.

2. (Raffel and Ellis, 2015) propone un modelo de atención que solo depende de los estados ocultos de la salida de una red recurrente, cada estado oculto es calculado mediante una función de activación LReLU(*Leaky Rectified Linear Unit*), debido a que los autores indican que esta función de activación acelera la convergencia de la RNN y el modelo de atención usa funciones de activación *TanH* y *LRelU*.

    - El modelo propuesto consigue una tasa de acierto de 99.9 % con 100 épocas en un conjunto de 1000 secuencias, además mantiene la misma precisión con secuencias muy largas (10000) y cortas.
    - El modelo es incapaz de distinguir el orden de una secuencia, el cual es una principal desventaja para modelos secuenciales que dependen del estado anterior.

[4]Center for research in computer vision: `http://crcv.ucf.edu/data/UCF101.php`

[5]Human motion database: `http://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/`

[6]Center for reseach in computer vision: `http://crcv.ucf.edu/data/UCF_YouTube_Action.php`

3. En (Masood et al., 2018) se propone dos enfoques para la inferencia de gestos de la lengua de señas, para el cual se utiliza el dataset LSA64 (Ronchetti et al., 2016), el primer enfoque llamado *prediction approach* realiza primeramente el procesamiento de las entradas eliminando el background y manteniendo la región de las manos, seguidamente es ingresado a la arquitectura *inception v3* y luego cada video es representada por una secuencia de predicciones hecha por la CNN para cada frame individual, finalmente se pasa a una red recurrente *LSTM* para poder aprender la información temporal, en el segundo modelo llamado *pool layer approach* pasa las salidas de la CNN a un capa de pooling obteniendo así un vector de características de 2048 y finalmente es pasado a una red recurrente.

    - El primer enfoque consigue una exactitud de 80.87 %.
    - El segundo enfoque tiene una exactitud de 95.21 %.

4. En (Luong et al., 2015) propone dos modelos de atención *Global Attention y Local Attention*, el cual difieren en la dependencia de todo los datos de entrada o solo de una pequeña parte para una determinada salida.

    a) **Global Attention:** La idea de un *Global Attention* es considerar todos los estados ocultos del *encoder* para tener el contexto, en cada paso de tiempo primero se infiere el alineamiento de la variable "*a*" basado en la actual salida "*ht*" y todos los estados ocultos "*hs*" del *encoder* luego es calculado el vector de contexto mediante un promedio de acuerdo a "*a*" y todos los estados ocultos del *encoder*, finalmente ser procesado por un *softmax* para predecir la respectiva salida.

    b) **Local attention:** En este modelo primero se predice un "*pt*" que es la posición de alineamiento de la actual salida "*ht*" y posteriormente es generado el vector contexto con todos los estados ocultos "*hs*" del *encoder* que se encuentran dentro de una ventana generado por el "*pt*".

    El entrenamiento se hizo sobre *WMT* [7] para traducir sentencias de inglés a alemán y viceversa, ambos modelos consiguieron un nuevo estado del arte en *WMT14* y *WMT15*, superando así a los modelos que no incorporan *Attencion Model*.

5. En (Mao et al., 2017) proponen una NN(Neural Network) que consiste de 3 módulos como la extracción de características, *encoder-decoder* y la fusión. Los datos extraídos son imágenes RGB (Red, Green and Blue) y coordenadas de esqueleto de un sensor kinect 2.0, para la extracción de características se usa el CNN *VGG16* y cuya salida son alimentadas al *encoder* y seguidamente al *decoder*, las coordenadas son directamente alimentadas a otro *encoder-decoder*, finalmente se desarrolló una ecuación matemática para poder fusionar estos dos tipos de datos.

    - Se usó una base de datos construido por los mismos autores que está constituido de 90 señas chinas en alta definición 1920X1080 y 100 ejemplos de cada una.

[7] English to German traslation dataset

- Las coordenadas del esqueleto están conformadas por 25 coordenadas XYZ.
- El modelo propuesto tiene una tasa de acierto de 93.5 % en las imágenes RGB y en la fusión consigue una taza de acierto de 94.7 %

1.3. Justificación

El desarrollo de esta arquitectura se realizará con el fin de proveer una herramienta para construir un medio de comunicación entre la lengua del LSP y el idioma español, esta arquitectura será capaz de interpretar una secuencia de *frames*, mediante un proceso de DL se realizará una traducción al idioma español de la frase en LSP y brindando una arquitectura funcional para un sistema de comunicación bidireccional futuro que pueda usarse en el sector público o privado. Otros motivos relevantes son disminuir la barrera de la comunicación, que existe entre las personas con discapacidad auditiva y el resto de la comunidad peruana, utilizando las técnicas de DL (*Deep Learning*) que mejor se adapten para este proyecto en particular, provocando que la comunidad de sordomudos tenga mayor participación en la sociedad y la población común logre aprender bases del LSP. Los motivos secundarios son el aporte científico de la creación de un *dataset* que ayude a otros proyectos relacionados con el LSP para que logren hacer avances y el camino de la recolección de datos se les simplifique.

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar una arquitectura de interpretación de expresiones comunes de la lengua de señas del Perú al idioma español para construir un medio de comunicación entre ambos.

1.4.2. Objetivo específico

1. Investigar y seleccionar técnicas de procesamiento de imágenes y extracción de características usando modelos DL.
2. Crear *datasets* de vídeos con datos RGB, *depth* y coordenadas de esqueleto para entrenar modelos de aprendizaje.
3. Evaluar la arquitectura propuesta en dos *datasets* para probar la tasa de exactitud.

1.5. Alcances y limitaciones

1.5.1. Alcances

- Se interpretaron 10 expresiones comunes del LSP para la configuración de manos, que constan de 20 palabras en el idioma español, dentro de la lengua

de señas se hace uso del símbolo de interrogación "?" que se ve contemplado en el presente trabajo.

- Las 10 expresiones comunes del LSP son seleccionadas bajo nuestro propio criterio. Se determinó bajo la sección "El vocabulario" del libro "Lengua de señas del Perú", la longitud de la traducción de la seña, que debe de tener un máximo de 5 palabras incluido el signo "?", la dificultad de la configuración de manos, desde una frase con señas simples hasta una frase de señas compuestas.

- La traducción final solo se da manera unidireccional mostrando un texto en español de cualquier persona realizando cualquier seña que se encuentre en el vocabulario propuesto.

- Para los gestos se consideró la parte superior del cuerpo humano, desde la cintura.

1.5.2. Limitaciones

- Los movimientos están restringidos por el arco de visión de la cámara y la distancia a ella que se especifican más adelante.

- Se tiene como limitante el poder de cómputo necesarios para entrenar una gran cantidad de datos.

- La obtención de datos también significa una limitante, no es posible la captura de inmensas cantidades de datos en los tres tipos de datos ya mencionados puesto que es necesario el uso del sensor Kinect para obtenerlos y no se ha encontrado una base de datos con las características requeridas hasta el momento de elaborado este trabajo.

1.6. Metodología

Por naturaleza del proyecto de investigación, se utilizará el enfoque cuantitativo que es de tipo secuencial y probatorio que parte de una idea y concluye con la elaboración de reporte de resultados (Sampier, 2010), teniendo un alcance descriptivo para la recopilación de información y medición. La metodología para el desarrollo del proyecto es:

1. **Revisión de literatura de la lengua de señas:** En este primer paso, al ser nuevo el tema, es necesario documentarse adecuadamente de los términos y conceptos, para lo cual se revisó mucho del libro oficial propuesto por el ministerio de educación del Perú y la DIGEVE, "Lengua de señas del Perú", y se realizó consultas con los interpretes de la OMAPED (Oficina Municipal de Atención a la Persona con Discapacidad) para poder entender este campo de la mejor manera.

2. **Selección de literatura:** Se realiza una investigación preliminar de los temas base para este problema planteado. Primeramente, la revisión de textos

para el estado de arte, DL, DNN, CNN, RNN, intérpretes de texto, preprocesamiento de imágenes, extracción de características y modelos secuenciales. Además de cursos online (Coursera) para un mejor entendimiento del estado de arte. En base a nuestro problema y con las limitaciones presentadas se seleccionan modelos convoluciones y recurrentes.

3. **Recolección de *frames*:** En esta fase se realiza la recolección secuencia de *frames* RGB, *Depth* y *Skeleton* de señas LSP para entrenar la arquitectura propuesta y evaluarla. También se recolectan dos *datasets* extras, VideoLSP10_Depth y VideoLSP10_Skeleton uno para el entrenamiento del modelo *Depth-ResNet50* y otro para el *Skeleton-ResNet50*. Para la recolección de *frames* se debe tomar en cuenta los puntos de iluminación y el escenario no debe ser un escenario muy cargado de personas. Aquí interviene el tipo de herramienta que se utiliza para la recolección de datos, el Kinect al ser el único sensor para este proyecto limita mucho la captura de datos por día, por lo que esta etapa se alarga demasiado.

4. **Implementación del modelo convolucional:** Utilizando nuestros datos recolectados y en base al estado de arte se realiza una primea implementación solo para extraer las características de la entrada de datos, se aplican las técnicas de preprocesamiento de imágenes para los datos, el entrenamiento es segmentado y los resultados guardados en un archivo hdf5. Se utiliza el método de *transfer learning* para el entrenamiento de datos RGB, para los datos depth y skeleton se realiza un entrenamiento desde cero con VideoLSP10_Depth y VideoLSP10_Skeleton.

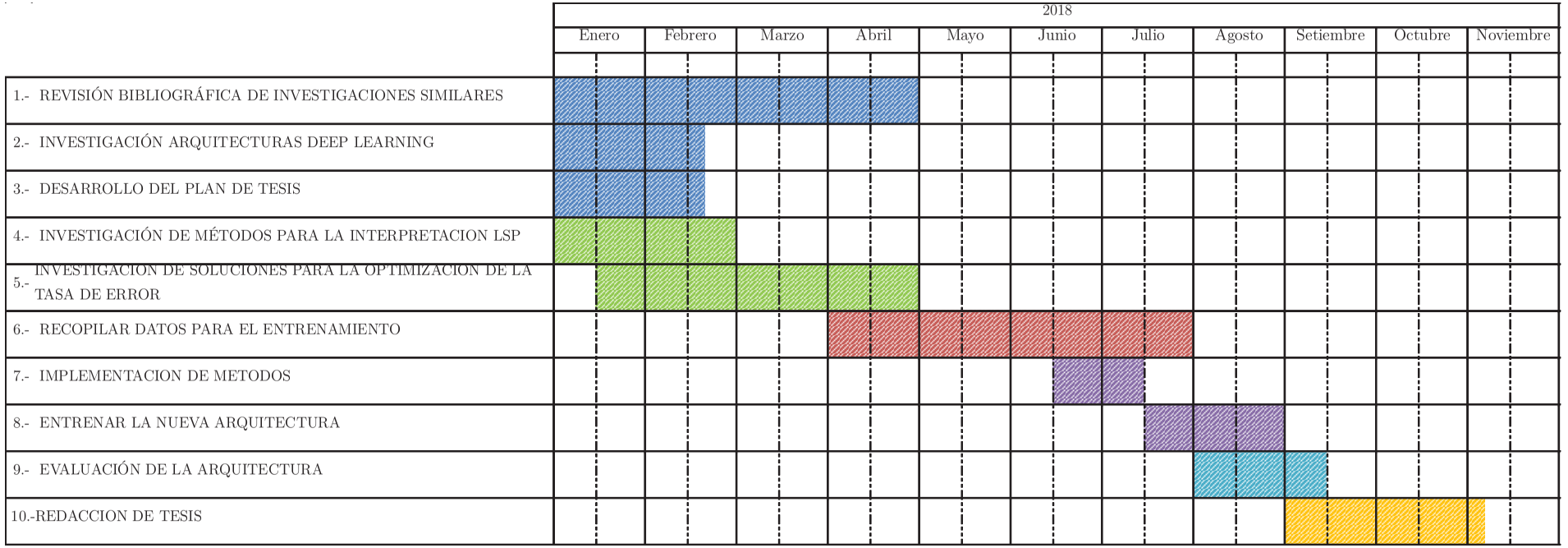
5. **Implementación de la arquitectura base:** En base a la selección de literatura, se confecciona la parte secuencial del modelo, se acopla al modelo confeccionado anteriormente formando el modelo base, para el modelo secuencial en base al problema planteado se hace énfasis en modelos bidireccionales, *encoder* y *attention decoder*, con los resultados obtenidos en el punto anterior y se entrena el modelo secuencial.

6. **Experimentación con modelos:** Partiendo de la estructura básica en el paso anterior, se realizan modificaciones de parámetros e hiperparametros para la red neuronal, también se aplican técnicas de regularización y optimización para poder encontrar un modelo que satisfaga nuestros objetivos. En este punto se utiliza el método de prueba y error para modificar la arquitectura y se adapte de mejor manera a los datos recolectados y el objetivo final.

7. **Evaluación de la arquitectura:** Contando con un modelo que cumpla con las especificaciones del problema se realiza una evaluación usando un modelo similar, la arquitectura LSA64 planteada por Masood et al. (2018), nuestra arquitectura final es adaptada para poder trabajar con el *dataset* de LSA64.

Herramientas utilizadas: Para el desarrollo del software se utilizo librerías como Tensorflow, Keras, OpenCV, Numpy, Matplotlib, Pandas y HDF5. Tambien se utilizaron lenguajes como C# y python 2.7

1.7. Cronograma de actividades



1.8. Costo y presupuesto

Los gastos dentro de este proyecto se describen a continuación.

COMPONENTES	COSTO	FINANCIAMIENTO
Procesador ITL i5 Core 2.8 Ghz	$ 190.01	Propio
Disco duro interno Seagate Barracuda 1 TB	$ 41.92	Propio
Placa madre MSI Z370-A PRO - ATX	$ 118.00	Propio
Tarjeta de video Asus Dual GTX 1060-06G	$ 408.54	Propio
Fuente de poder EVGA 500W W1/80+	$ 43.31	Propio
Impuesto de envio 1	$ 41.75	Propio
Impuesto de envio 2	$ 102.57	Propio
Sensor Kinect v.1	$ 60.15	Propio
TOTAL	$ 1006.25	

Tabla 1.1: Costos y presupuestos

Parte II

Marco teórico

Capítulo 2

Marco conceptual

En primer lugar se debe hacer una diferenciación con los términos que generalmente se manejan dentro del campo de las ciencias de la computación. Según el campo de estudio existen diferentes definiciones, por ejemplo una arquitectura es la visión que tiene un programador con respecto a un computador, esta definido por un lenguaje, registros y memoria, así existen muchos tipos de arquitecturas según (Harris and Harris, 2010). Sin embargo la IEEE define una arquitectura como la organización fundamental de un sistema representada por sus componentes, la relación entre ellos y con el entorno y los principios que guían su diseño y evolución (de Pablos Heredero, 2004).
En *Deep Learning* se hace referencia a una arquitectura como redes neuronales que contienen muchas capas (Jones, 2017) y se define como una estructura de las redes neuronales (Briega, 2017).

2.1. Descripción del dispositivo Kinect

Kinect fue concebido originalmente con el nombre de "Project Natal" y es un dispositivo desarrollado para la consola de juego Xbox 360, Kinect proporciona una NUI (Natural User Interfaces) para poder interactuar con el dispositivo mediante el movimiento y la voz. Este sensor no está limitado solamente al campo de los vídeo juegos, actualmente es usado por muchos desarrolladores para sus propias aplicaciones basadas en la detección de movimiento.El Kinect es relativamente nuevo la documentación existente es algo limitada, Windows desarrollo su propia librería para C# el Kinect SDK.

2.1.1. Componentes del Kinect

El Kinect está compuesto por un sensor de profundidad, cámara a color y un conjunto de micrófonos; en la parte posterior del sensor esta ensamblado un motor para modificar el grado de inclinación horizontal del sensor. El sensor esta compuesto por una cámara de color, emisor de IR, sensor IR (infrarojo) de profundidad, motor de inclinación, un conjunto de micrófonos y un LED de estado.

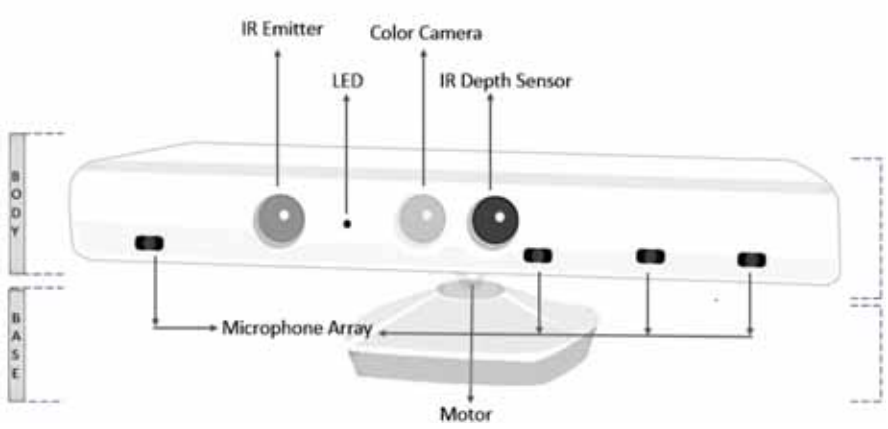


Figura 2.1: Componentes exteriores del Kinect
Fuente: Jana (2012)

Cámara de color

Esta cámara es responsable de capturar los datos de color del vídeo detectando en rojo, azul y verde, la secuencia de datos devuelta un conjunto de *frames*. El Kinect v.1 soporta 30 fps (frames per second) con la cámara de color con una resolución de 640 x 480 píxeles a 1280 x 960 píxeles con 12 fps. La visibilidad que abarca es de 43 grados verticales y 57 grados en horizontal.

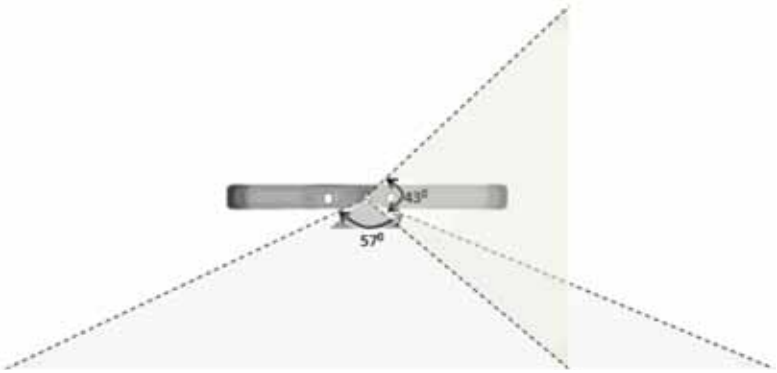


Figura 2.2: Grado de visibilidad
Fuente: Jana (2012)

Emisor IR y sensor IR de profundidad

Para el sensor de profundidad existen dos componentes que trabajan en conjunto, el sensor IR de profundidad y el emisor IR. El emisor IR emite constantemente luces infrarrojas en patrones de puntos pseudoaleatorios sobre el frente de la cámara, estos puntos son solo visibles para el sensor IR de profundidad, que los lee y calcula la distancia entre sensor y el objeto donde se proyecta el punto IR. La secuencia de datos emitida por el sensor de profundidad es de 640x480 píxeles, 320x240 píxeles y 80x60 píxeles.
El sensor Kinect tiene la capacidad de capturar datos crudos en 3D utilizando el emisor IR y el sensor IR de profundidad que es un sensor monocromo CMOS

(Complementary metal - oxide - semiconductor)[1]. Cuando se necesite capturar datos de profundidad el chip *PrimeSense* envía una señal, figura 2.4 al emisor IR para encender el infrarrojo (1), y envía otra señal al sensor IR de profundidad para que capture (2) mientras que el emisor emite la señal (3) el sensor IR de profundidad va leyendo los datos y calculando la distancia (4) y los pasa al chip *PrimeSense* (5) y finalmente crea una imagen de profundidad por cuadro y la pasa al flujo de salida (6).

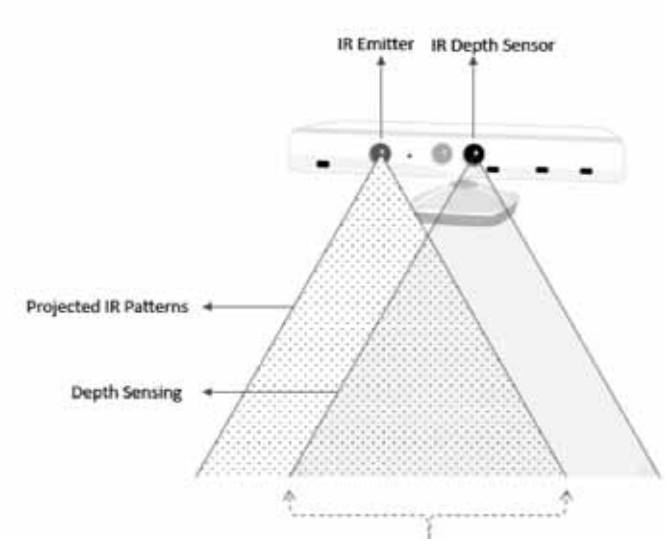


Figura 2.3: Emisor IR y sensor IR de profundidad
Fuente: Jana (2012)

Motor de inclinación

Para obtener una visualización correcta, el Kinect posee un motor en la base para mover la cámara verticalmente hasta unos 27 grados y -27 grados, esta configuración de movimiento no se debe hacer manualmente.

Conjunto de micrófonos

El Kinect consta de cuatro micrófonos, no solo para capturar el sonido sino ubicar la dirección del sonido. Las ventajas son que la captura y reconocimiento de voz se hace con supresión de ruido.

LED

Se coloca un LED entre la cámara y el proyector IR. Se usa para indicar el estado del dispositivo Kinect. El color verde del LED indica que los controladores del dispositivo Kinect se han cargado correctamente.

[1]Se encuentra en las cámaras digitales más actuales, es utilizado para convertir imágenes en datos de carácter digital. Es posible integrar más funciones como por ejemplo control de luminosidad, corrector de contraste o un conversor analógico-digital.

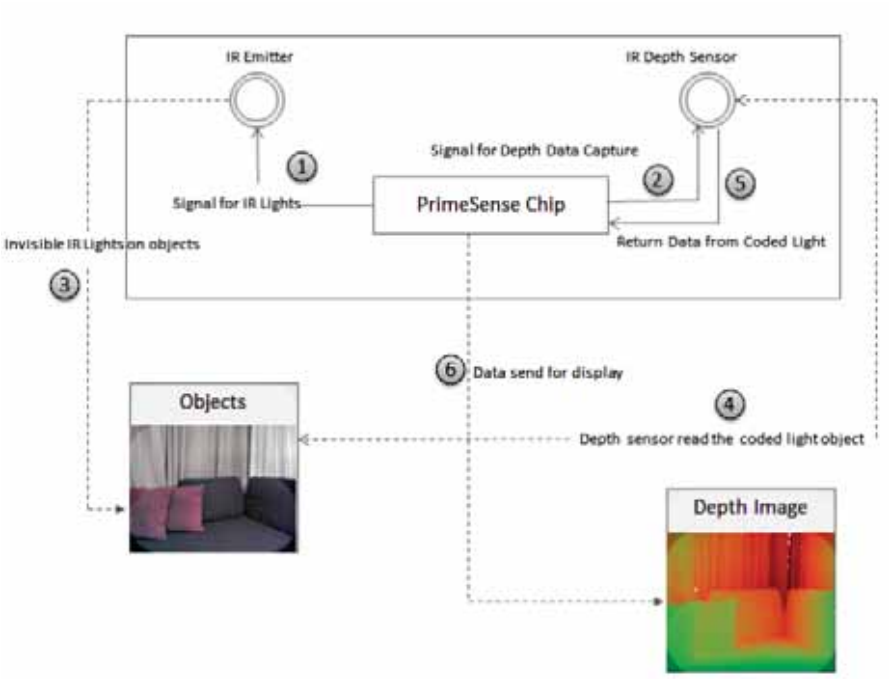


Figura 2.4: Secuencia de obtención de imagen de profundidad
Fuente: Jana (2012)

2.1.2. Kinect para Windows versus Kinect para Xbox

Ambas versiones son similares, el Kinect para Xbox tiene como principal función la experiencia de videojuegos mientras que el Kinect para Windows tiene como objetivo principal el desarrollo aunque se puede desarrollar en ambos, la diferencia radica en la distancia que el Kinect comienza a detectar, el Kinect para Windows contiene *Near Mode* que permite detectar objetos a partir de los 40cm a diferencia de la versión por defecto dentro del Kinect de Xbox desarrollada para jugadores que detecta partir de los 80cm a 4 metros.

2.1.3. Usos de Kinect

El Kinect con sus características tiene una amplia gama de aplicaciones a desarrollar como:

- Captura de vídeo en tiempo real usando el sensor de color
- Rastrear un cuerpos humanos y responder a sus movimientos y gestos como una interfaz de usuario natural
- Medir las distancias de los objetos
- Análisis de datos 3D y realización de un modelo 3D y medición
- Generación de un mapa de profundidad de los objetos rastreados
- Reconocimiento de una voz humana y desarrollo de aplicaciones*hands-free* que pueden controlarse mediante voz

Para tener un mayor entendimiento el Kinect se puede aplicar a los siguientes campos:

- Salud: Desarrollo de aplicaciones para la atención médica, como la medición de ejercicios, monitoreo de pacientes, sus movimientos corporales, etc.
- Robótica: Desarrollo de sistema de navegación para robots ya sea mediante el seguimiento de gestos humanos, comandos de voz o movimientos del cuerpo humano
- Educación: Creación aplicaciones para que ayude a aprender materias ya sea por gestos y comandos de voz.
- Sistema de seguridad: Kinect puede usarse para desarrollar sistemas de seguridad donde pueda rastrear el movimiento o la cara del cuerpo humano y enviar las notificaciones
- Realidad virtual: Con la ayuda de la tecnología Kinect 3D y el seguimiento de gestos humanos
- Entrenador: Kinect puede usarse potencialmente como entrenador midiendo los movimientos de las articulaciones del cuerpo humano, proporcionando retroalimentación en vivo a los usuarios
- Militar: Kinect puede usarse para construir *drones* inteligentes para espiar las líneas enemigas

2.2. Lenguaje de señas

2.2.1. Terminologías

Lingüística en la lengua de señas

El lenguaje es definido en (Daza, 2005) como una manera racional de exteriorizar las distintas formas de en las que se interpreta la realidad en base una serie de intenciones, motivaciones y necesidades que se presentan. El lenguaje según Chomsky es innato. No puede ni aprenderse, ni olvidarse; porque es una capacidad que nos permite adquirir nuestra propia lengua, por otro lado la lengua es definido en (García Benavides, 2004) como un concepto que no alude a una facultad, ya que se adquiere, se enseña y se aprende, se considera como un sistema de elementos fonéticos y morfológicos que se rige por unas reglas, que presenta unos niveles y que se puede diferenciar en el orden estructural o significativo de acuerdo con el conglomerado de hablantes y los territorios que ocupe geográfica y políticamente.
El termino Lenguaje de señas y lengua de señas se ha ido usando indistintamente, esto se debe a que las primera publicaciones en las que se basaron los estudiosos en esta área eran publicaciones en ingles, (Lyons, 1984) explica que en el idioma ingles existe solo una palabra para definir lenguaje, en un sentido general como una capacidad inherente al hombre y lengua, en un sentido mas particular, *language*, a diferencia de otras lenguas europeas diferenciadoras como el italiano y el español que emplea los términos lenguaje y lengua. Por tanto es correcta el uso de

la terminología lengua de señas, pero es normal encontrar trabajos con el termino lenguaje de señas que hacen referencia a la lengua y no lenguaje.
La sordera posee diferentes tipos según diversos enfoques como la localización de la lesión, el grado de pérdida auditiva,las causas y la edad del comienzo de la sordera. Sin embargo a pesar de distintos mitos (García Benavides, 2004) señala que el término *sordomudo* es incorrecto porque señala que la persona sorda no puede hablar, debido a su sordera, esto no es cierto, puesto que si la persona hace una terapia para logrará hablar. Dentro de la comunidad de personas sordas se hace uso una determinada lengua de señas que corresponde a un determinado país o región, no todas a las lenguas de señas son iguales aunque son interpretables si se conoce un determinada lengua de señas.

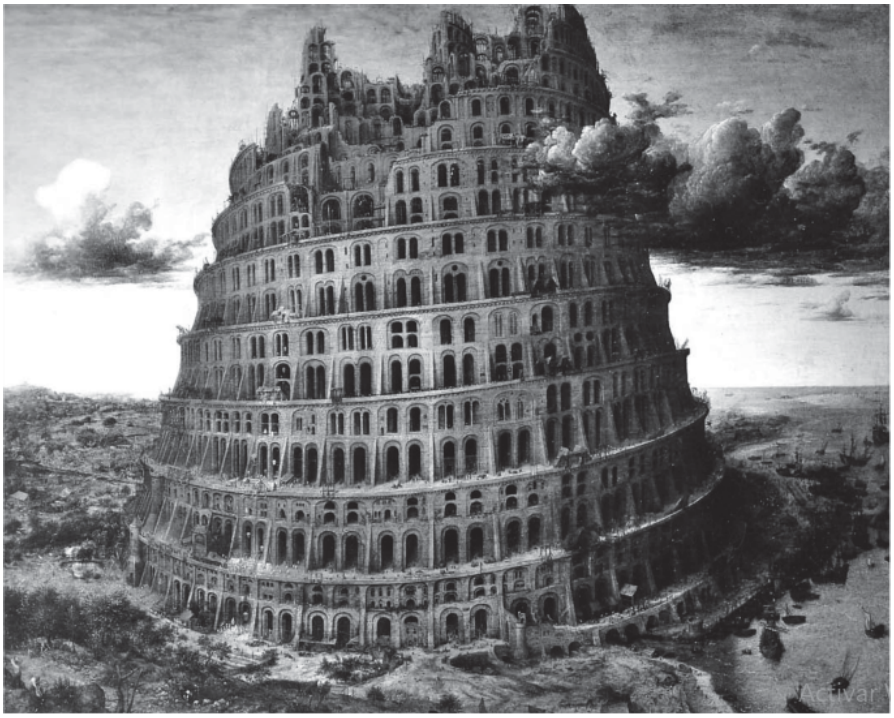


Figura 2.5: Imagen de la torre de Babel
Fuente: Daza (2005)

2.2.2. LSP

La lengua de señas peruanas con siglas LSP fue creada y es utilizada por la comunidad de sordos del Perú, al igual que otras lenguas minoritarias usadas en el Perú, actualmente está reconocida por el estado mediante la Ley 29535, el Ministerio de Educación, en el año 1987 publicó un Manual de Lengua de Señas que se ha ido modificando y corregido.

Figura 2.6: Lugares de articulación del LSP
Fuente: DIGEBE (2015)

Fonología

La Lengua de Señas posee una "fonología" abstracta : la Querología, que analiza, al igual que en los fonemas de las lenguas orales, los parámetros formacionales o queremas de las señas: configuración de las manos, orientación movimiento de la mano, punto de contacto, plano y componente no manual. Dentro de LSP los queremas se articulan en posiciones establecidas del cuerpo como en la figura 2.6

Base del LSP

El Numero gramatical, dentro del LSP solo basta con agregar un numero cardinal, un adjetivo de cantidad o un articulo antes de la seña de la palabra correspondiente. Se puede observar en la figura 2.7 parte superior el numero gramatical

"tres" y la seña correspondiente a "libro", la seña central en la figura que se muestra comienza con un adverbio de cantidad "mucho", y la seña correspondiente a "automovil" finalmente la ultima seña de la figura comienza con un articulo y la señal de "niños".

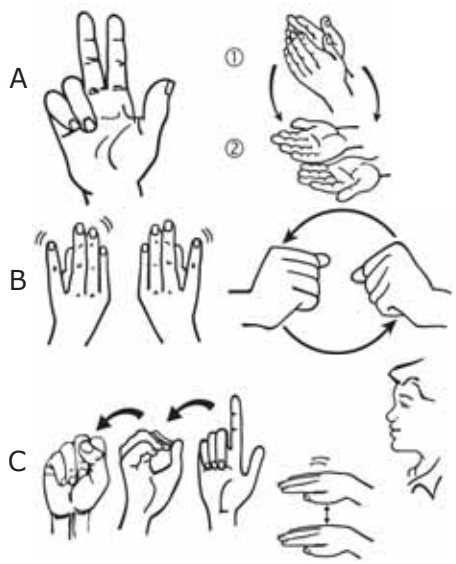


Figura 2.7: Numero Gramatical.
Superior (A) "tres libros", Centro (B) "muchos autos" e Inferior (C) "los niños".Fuente: DIGEBE (2015)

Genero gramatical, se puede expresar el genero de dos maneras, si la palabra es un sustantivo heterogéneos existe una seña para ella, en cambio en los sustantivos comunes véase figura2.8 si se desea establecer un genero se utiliza la seña de masculino señalando el bigote o en caso contrario se establece la seña de femenino señalando el labio inferior.

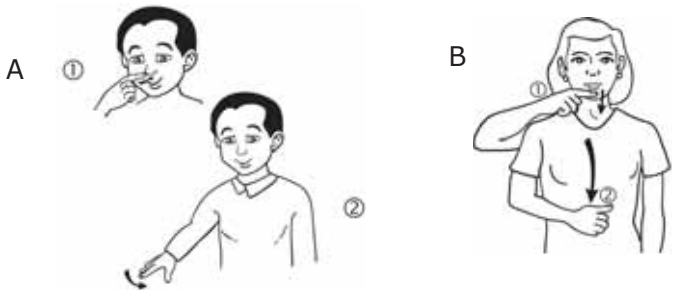


Figura 2.8: Señas para sustantivos heterogéneos.
Ejemplo de la seña para genero masculino (izquierda) "niño varón". (Derecha) Señal para "hija mujer". Fuente: DIGEBE (2015)

Tiempos Verbales, para poder expresar el tiempo verbal se utilizan señas dirigiendo la mano hacia atrás por el hombro indicando pasado y para indicar futuro lanzando la mano ligeramente hacia adelante como en la figura 2.9. El tiempo presente esta implícito por lo que no es necesaria su seña, el cuerpo del emisor esta en

linea de separación entres las otras dos referencias temporales, según (DIGEBE, 2015) no representan el tiempo sino visiones del tiempo proyectadas hacia atrás o adelante.

Figura 2.9: Tiempo Verbal en el LSP
Fuente: DIGEBE (2015)

Formas interrogativas, los componentes faciales son un elemento importante de la lengua de señas que siempre deben acompañar a las formas interrogativas, pues sin ellos el mensaje que se quiere transmitir quedaría incompleto, perdiéndose parte de la información (DIGEBE, 2015). También es común ver que dentro de la comunidad de sordos se puede expresar las preguntas con un signo interrogativo "?" dibujado con el dedo índice de la mano derecha. Algunas señas de preguntas interrogativas se muestran en la figura 2.10.

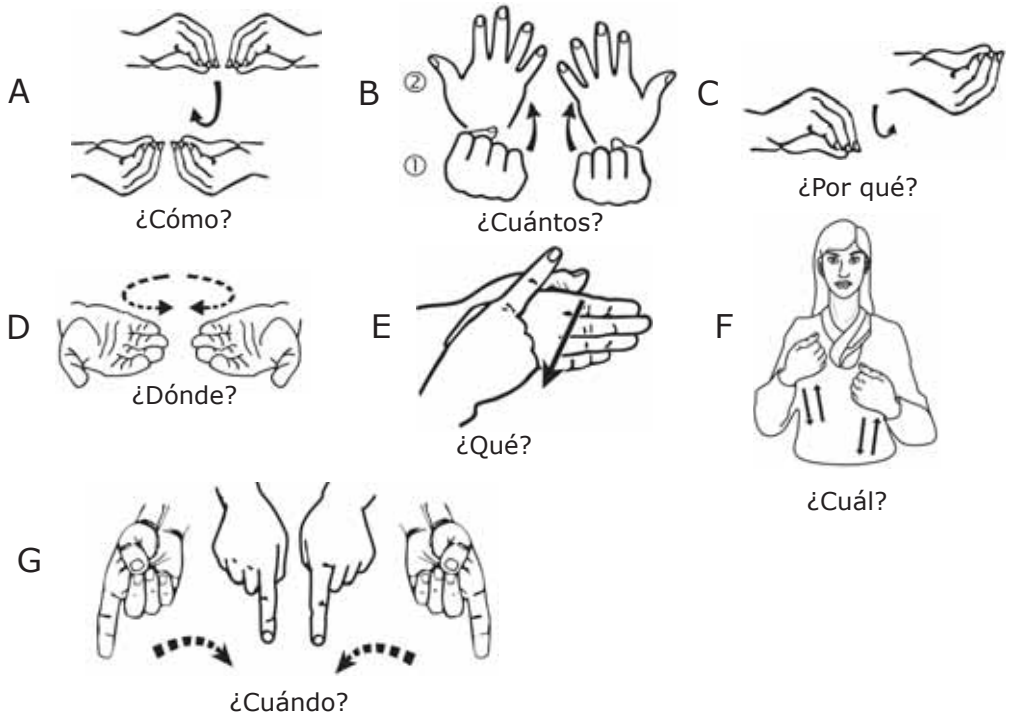


Figura 2.10: Señas de preguntas
Fuente: DIGEBE (2015)

Pronombres, para referirse a las personas en lengua de señas, se utiliza el dedo índice de la mano derecha, a esta referencia se le llama deixis, la deixis se usa para referirse a los pronombres personales, los demostrativos, los adverbios referenciales de lugar. El Uso de la deixis para referirse a los pronombres sean personales, posesivos y demostrativos, se usa el dedo índice (deíctico) como se ejemplifica en la figura2.11.

Figura 2.11: Señas Pronombres
Fuente: DIGEBE (2015)

2.3. Inteligencia artificial

Una característica que nos ha diferenciado del resto de animales y nos ha elevado entre ellos es la inteligencia, en los últimos años se ha visto reflejada en los avances tecnológicos perfilados a crear sistemas inteligentes, aportes teóricos y aplicativos que emulan esta característica humana tan especial que es el objetivo del campo de la IA, según (Isasi and Galvan, 2004) la IA (Inteligencia Artificial) se divide en dos grandes áreas la inteligencia simbólica (*top – down*) que consiste desarrollar sistemas con características que se puedan definir como inteligentes, su enfoque es el estudio de mecanismos del razonamiento humano de alto nivel, los sistemas expertos siguen este esquema, se introducen reglas que contienen conocimiento de un experto en la materia y haciendo uso de mecanismo de inferencia equivalentes al razonamiento humano se sacan conclusiones. El otro campo es la inteligencia subsimbólica (*down - up*) que parte de sistemas genéricos que van adaptándose y construyéndose para resolver un problema, su enfoque es el estudio los mecanismos físicos que nos capacitan como seres inteligentes y nos diferencian de sistemas autómatas preprogramados, como el sistema nervioso, el cerebro, su estructura funcionamiento y características, en este diseño de abajo hacia arriba se tratan de emular estas características físicas y se van generando cómputos más complejos mediante mecanismos prefijados de aprendizaje aquí se hallan las Redes Neuronales.

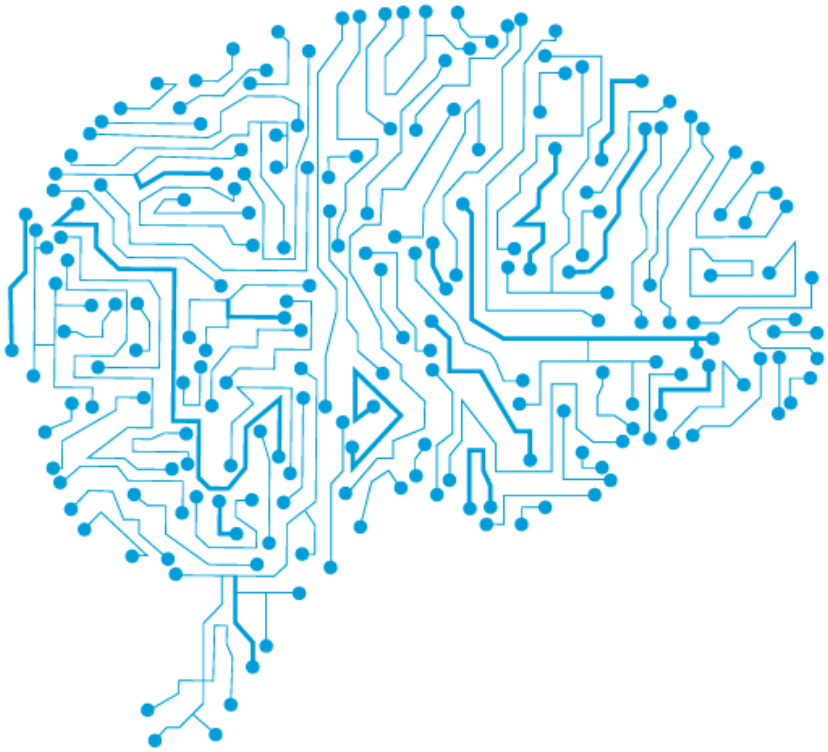


Figura 2.12: Representación de la IA
Fuente: Benitez (2017)

2.3.1. Visión computacional

La visión computacional es la ventana al mundo de muchos organismos. Su función principal es reconocer y localizar objetos en el ambiente mediante el procesamiento de las imágenes. En los últimos años la visión por computadora es una de las áreas de más avance gracias al aprendizaje profundo, el reconocimiento facial ya tiene mejor funcionamiento y es común verlo en nuestro entorno. El entusiasmo por aprendizaje profundo de la visión computacional se debe a que se está permitiendo la visualización de nuevas aplicaciones y que la comunidad científica se ha vuelto más creativa e inventiva al diseñar arquitecturas y algoritmos de redes neuronales. Los problemas de visión computacional generalmente son la clasificación de imágenes, el reconocimiento de imágenes donde toman como entrada una imagen, detección de objetos, para un conductor automático que detecte autos y peatones.

2.4. Redes neuronales

La comunicación neuronal está formada por tres partes, los receptores que captan información del exterior e interior del organismo, el sistema nervioso que elabora la información para transmitirla y los órganos efectores (músculos o glándulas) que interpretan las señales transmitidas como acciones motoras u hormonales.

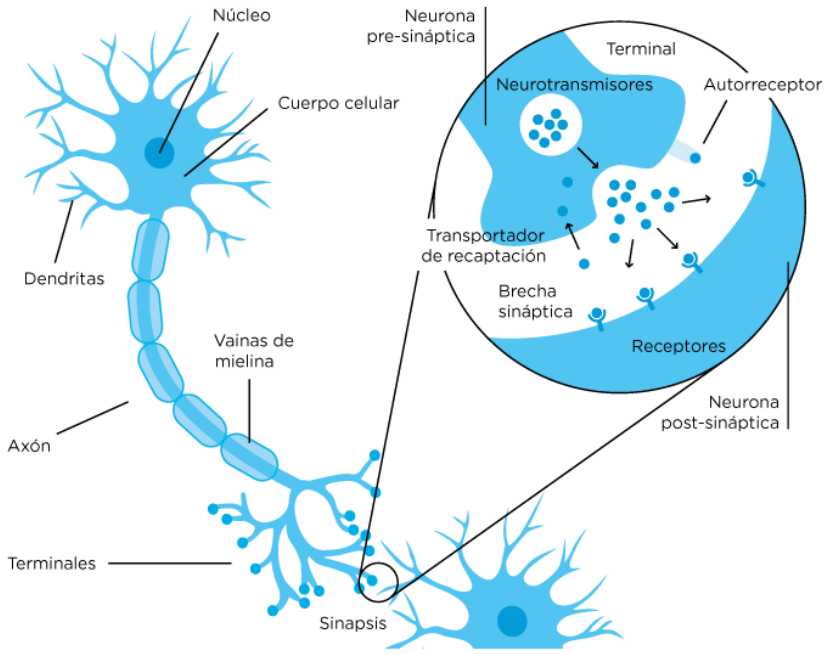


Figura 2.13: Explicación de la sinapsis
Fuente: Pedro Bekinschtein (2017)

El eslabón dentro de este sistema es la neurona responsable de las señales químicas o impulsos electro-químicos que envían o almacenan información atreves de una

red formada por sí mismas, las neuronas que reciben estos impulsos almacenan el impulso codificándolo para transmitirlo por una ramificación de salida denominado axón.
El axón se comunica con ramificaciones de entrada denominadas dendritas, no hay una conexión directa o fusión, se trata de un espacio liquido donde existen elementos ionizados (iones de sodio y potasio), estos elementos tienen propiedades de conductividad que hace posible la transmisión del impulso eléctrico, actuando como potenciadores o inhibidores de señal, a este proceso se le conoce como sinapsis, figura 2.13. La transmisión de impulsos no es lineal, no se proyectan todas las señales enviadas a una neurona, se transmite una señal de activación que al llegar a un umbral inhibe la transmisión.

2.4.1. Modelo computacional

Neurona artificial

La neurona artificial posee un estado interno, nivel de activación, se denomina S a sus posibles estados, utilizando esta función, el nivel de activación puede cambiar a partir de las señales que recibe o entradas denotadas como x, para lo cual debe calcularse primero el total de la célula Ei, la suma de las entradas ponderas, la multiplicación de las señales por sus pesos asociados w equivalente a la fuerza de conexión sináptica.

$$E = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n \tag{2.1}$$

$$E_i = X^T W \tag{2.2}$$

Donde $X = x_1, x_2, \cdots, x_n$, $W = w_1, w_2, \cdots, w_n$ y T denota una capa, las señales E son procesadas por la función de activación denominadas F.

Estructura básica

La forma en la que las neuronas se conectan entre si se denomina conectividad o arquitectura de red, la figura 2.14. La estructura consta de niveles formados por neuronas a los que denominaremos capas, la capa inicial recibe patrones representados como vectores, las capas de en medio o capas ocultas y las capas de salida, cada interconexión actúa como ruta de comunicación pasando valores numéricos, estos son evaluados por los pesos y se van ajustando durante la fase de aprendizaje. El funcionamiento de la red es simple la capa de entrada introduce el vector de entrada, genera una salida que es propagada por la red hasta la neurona de destino, una vez que todo el vector se propago por la red se producirá un vector de salida $\vec{S}$.

$$\vec{S} = F(F(\vec{X}.W_1).W_2) \tag{2.3}$$

Aprendizaje

Las redes neuronales están basadas en ejemplos, ligados al aprendizaje. Deben de ser significativos (cantidad) y representativos (uniformidad) de lo contrario la NN será ineficiente o excesivamente especializada. El aprendizaje consiste en determinar el valor óptimo de los pesos, el proceso consiste en ir introduciendo los

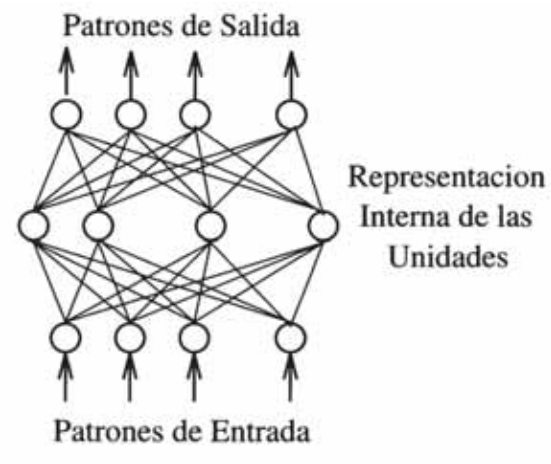


Figura 2.14: Esquema de red
Tres capas totalmente interconectadas. Fuente: Isasi and Galvan (2004)

ejemplos y modificando pesos según el esquema de aprendizaje, al finalizar se comprueba el cumplimiento del criterio de convergencia, de no realizarse se vuelve a modificar los pesos y se vuelve a introducir los ejemplos. El periodo de aprendizaje puede ser determinado por factores como el número de ciclos determinados a priori para la introducción de los conjuntos de entrada, cuando el error descienda por debajo de un valor prefijado o la modificación de pesos se vuelve irrelevante, de manera que la modificación de pesos se realice con menos intensidad que al principio. El esquema de aprendizaje determina el tipo de problema que puede llegar a resolver la red, se distinguen tres tipos:

- **Aprendizaje supervisado:** Los datos de entrada a la red tienen dos atributos, el vector de entrada y un parámetro de información adicional denominado etiqueta. Al introducir los datos de salida la red, se hace una comparación de los valores obtenidos y las etiquetas, valores deseados, la diferencia entre ambos determinara la magnitud en la que se modificaran los pesos posteriormente en el ajuste. En este tipo de aprendizaje existe un profesor externo que determina si la red tiene un comportamiento adecuado y de actuar correctamente en la modificación de pesos.

- **Aprendizaje no supervisado:** Solo tiene los datos de entrada y etiquetas, no existe un profesor externo que la supervise. En este caso la red neuronal modificara los pesos usando información interna. En este caso la NN determinara características: redundancias, rasgos significativos o regularidades, y depende únicamente de los valores de entrada.

- **Aprendizaje por refuerzo:** Es similar al aprendizaje supervisado no se dispone de información concreta del error cometida por la red para cada ejemplo, únicamente se determina si la salida es adecuada. El proceso consiste en modificar los pesos hasta que las salidas sean lo más parecidas a las etiquetas, aun así, esto no garantiza que la red sea buena, esto puede causar *overfitting*.

Para determinar mejor la eficiencia de la red se dividen los datos en entrada en dos conjuntos, entrenamiento y validación. El conjunto de entrenamiento determinará

el valor de los pesos, pero no el grado de error, el conjunto de validación será completamente distinto al de entrenamiento y medirá el error, si es pequeño se garantiza la generalidad de la red, este último conjunto también puede utilizarse para guiar el aprendizaje en conjunción con el conjunto de entrenamiento.

2.5. Deep learning

La base para las futuras tendencias tecnológicas está detrás del DL, esta existe desde hace algunos años sin embargo recientemente esta tomando impulso en proyectos como clasificadores de *spam*, predictores de clics en anuncios, calculador de posición de otros autos para el manejo de coches automáticos y entre otros. Pero ¿por qué DL funciona tan bien? Basándonos en (Andrew, 2017) en los últimos años las digitalización de la sociedad impulso incremento de información gracias a las cámara de bajo costo incrustadas en los dispositivos móviles, el tiempo que se pasa en los ordenadores, páginas web y el bajo costo para conectarse a internet han hecho posible que en los últimos 20 años se logre obtener una gran cantidad de datos que los algoritmos de aprendizaje tradicional no pueden explorar en su totalidad, si se entrena una red pequeña el rendimiento será mejor a comparación de métodos tradicionales y si la red aumenta el rendimiento también lo hará. Andrew Ng investigador de la inteligencia artificial e informático teórico sostiene que si se desea alcanzar un rendimiento muy alto es necesario tener en cuenta dos puntos: entrenar una red lo suficientemente grande para aprovechar la gran cantidad de datos que se posee y tener una gran cantidad de datos, sin embargo tener una red inmensa y una gran cantidad de datos aseguran el éxito, eventualmente los datos se agotarán y la red tardara en entrenarse un tiempo excesivo. El impulso del DL depende también de las innovaciones de algoritmos que tratan de mejorar la velocidad de cálculo de las NN. El proceso de entrenamiento de una NN consta de tres partes: idea, codificación y experimentación como se ve en el gráfico 2.26, se plantea una idea de una arquitectura, se implementa la arquitectura en un código y se procede a experimentar cuan bien se ejecuta, observando los resultados se generan más ideas y se vuelve a codificar entrando en este bucle, si el tiempo de entrenamiento es muy largo el tiempo de recorrido de este ciclo lo será también, por lo que las innovaciones de algoritmos para el cálculo de las redes neuronales ayudara a la implementación de la arquitectura detectando más rápidamente los resultados y mejorándolos.

2.5.1. Regresión logística

La regresión logística o por sus siglas en ingles LR (Logistic Regression) es un algoritmo de aprendizaje utilizado cuando las etiquetas Y son binarias (0 o 1) en problemas de aprendizaje supervisado, su objetivo es minimizar el error entre las predicciones y el conjunto de entrenamiento. Dada una imagen representada por un vector de características X el algoritmo evaluara la probabilidad de que la imagen sea una seña o no.

$$\text{Dado } x,\ \hat{y} = P(y = 1|x) \ \text{ donde } 0 \leq \hat{y} \leq 1 \tag{2.4}$$

La salida $\hat{y}$ es igual a la ecuación (2.2), pero no es un buen algoritmo para una clasificación binaria, es una funcionalidad lineal, pero como se busca una restricción

entre [0,1] se utiliza la función *sigmoide* eliminando la linealidad, se observa que los valores de $\hat{y}$ en la ecuación (2.5).

$$\hat{y} = \sigma(W^T x + b) \tag{2.5}$$

- Si z es un número positivo grande, entonces $\sigma(z) = 1$
- Si z es un numero negativo grande o pequeño, entonces $\sigma(z) = 0$
- Si z es cero, entonces $\sigma(z) = 0{,}5$

Para entrenar los parámetros w y b es necesario definir la función de costo. Dado como premisa

$$\begin{gathered} \hat{y}^{(i)} = \sigma(W^T x^{(i)} + b) \\ \text{Donde } \sigma(z) = \frac{1}{1+e^{-z^{(i)}}} \\ \text{Dado } \{(x^{(i)}, y^{(i)}), \ldots, (x^{(m)}, y^{(m)})\} \\ \text{Se desea } \hat{y}^{(i)} \approx y^{(i)} \end{gathered} \tag{2.6}$$

2.5.2. Función loss

Es función de perdida $L(\hat{y}^{(i)}, y^{(i)})$ mide la discrepancia entre la predicción $\hat{y}^{(i)}$ y la salida deseada $\hat{y}^{(i)}$, en otras palabras calcula el error para un solo ejemplo de entrenamiento, se podría expresar entonces como el promedio de la diferencia al cuadrado $\frac{1}{2}(\hat{y}^{(i)} - y)^{(i)})^2$ sin embargo cuando se llega la fase de aprendizaje el problema de optimización se vuelve no convexo, quiere decir que se llegarían a obtener muchos puntos óptimos y no necesariamente el verdadero punto óptimo, para dar solución a este caso se define la función de perdida logística como la ecuación (2.7)

$$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)})) + (1 - y^{(i)})) \log(1 - \hat{y}^{(i)})) \tag{2.7}$$

Categorical cross-entropy

Para entender mejor (Singh, 2018) describe los conceptos de *surprisal*, "Grado de sorpresa al ver el resultado", Vijendra Singh en su publicación *Understanding Entropy*, *Cross-Entropy and Cross-Entropy Loss*, se define con la ecuación (2.8), *entropy* es la media ponderada del *surprisal* y *cross entropy*, dado p_i como la probabilidad real del resultado y q_i la estimación de la probabilidad, cada evento ocurrirá por la probabilidad p_i mientras que el *surprisal* esta dado por q_i como en la ecuación (2.9), así a medida que la distribución de la probabilidad estimada se aleja de la distribución de la probabilidad deseada, el CE (Cross Entropy)aumenta o viceversa, por tanto minimizara CE acerca el resultado a la distribución deseada, la figura 2.15 ayuda visualizar el CE entre dos distribuciones. La función Roja representa una distribución de probabilidad deseada, la función naranja representa la distribución de probabilidad estimada y la barra púrpura muestra la entropía cruzada entre estas dos distribuciones(área bajo la curva azul).

$$S = \log(1/n) \tag{2.8}$$

$$c = \sum_{0}^{n} p_i \log(1/q_i) \tag{2.9}$$

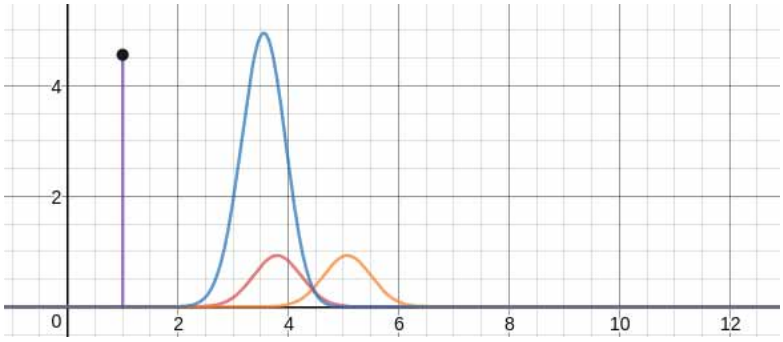


Figura 2.15: Gráfico de Cross Entropy
Fuente Singh (2018)

Una función de perdida también conocida como *negative log likelihood*, (Singh, 2018) es una de las funciones mas utilizadas en DNN y se basa en el modelo de *cross entropy*.
Esta función (Goldberg and Hirst, 2017) define su uso para una interpretación probabilística de las puntuaciones, sea $y = y_{[1]}...y_{[n]}$ el vector que representa la distribución multinomial verdadera sobre las etiquetas $1, 2, 3, ..., n$ y sea $\hat{y} = \hat{y}_{[1]}...\hat{y}_{[n]}$ el clasificador lineal de salida que fue transformado por la función *softmax*, el *categorical cross entropy loss* mide la diferencia entre la distribución real de las etiquetas y y la distribución prevista de la etiqueta $\hat{y}$ definiéndose en la ecuación (2.10)

$$L_{cross-entropy}(\hat{y}, y) = -\sum_{i} y_{[i]} \log(\hat{y}_{[i]}) \tag{2.10}$$

Función costo

Es el promedio de la función de perdida de todo el conjunto de entrenamiento, al entrenar el modelo de LR vamos a encontrar parámetros w y b que minimicen los costos totales para la función J.

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \tag{2.11}$$

2.5.3. Gradiente descendente

Se utiliza la gradiente descendente para aprender los parámetros w y b en su conjunto de entrenamiento, se desea encontrar $J(w, b)$.
Se ilustra en la figura 2.16 convexa, donde w es un número real único y b es un único número real, la función de coste $J(w, b)$ es alguna superficie por encima de los ejes w y b, para encontrar el parámetro óptimo se inicializa con valores aleatorios u otros métodos, se implementa un bucle como en (2.12) hasta que el algoritmo converja, en cada paso descenderá por la gráfica convexa hasta alcanzar el punto óptimo.

$$\text{Repetir } \left\{ w := w - \alpha \frac{\partial J(w)}{\partial w} \right\} \tag{2.12}$$

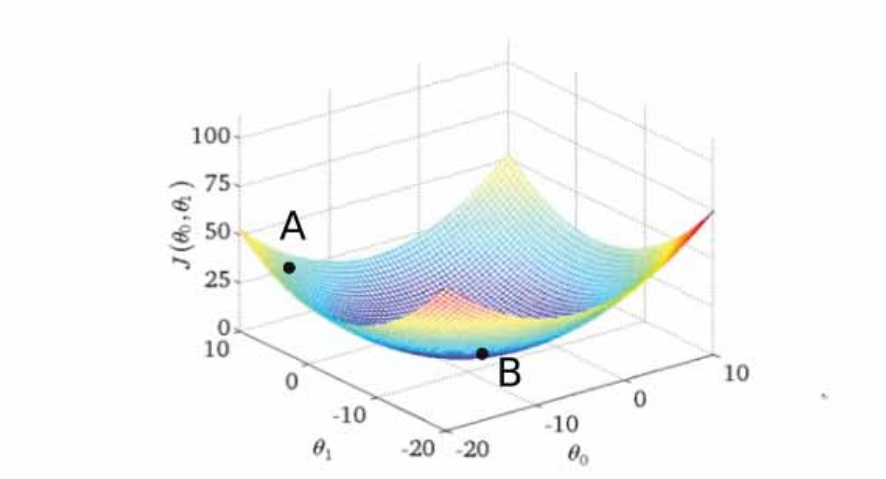


Figura 2.16: Gradient Descent
Fuente: Kathuria (2018)

$$w_{t+1} = w_t - \alpha \frac{1}{n} (\sum_{i=1}^{m} \nabla_w L(w)) \tag{2.13}$$

Stochastic gradient descent

El SGD (Stochastic Gradient Descent) es una simplificación drástica, (Bottou, 2012) en lugar de calcular la gradiente exacta de $J(w, b)$ cada iteración estima este gradiente sobre la base de un ejemplo escogido al azar (x^i, y^i), esperando que la ecuación (2.15) se comporte como su expectativa, ecuación (2.13). Cada actualización de parámetros en SGD se calcula generalmente con un *minibatch* en lugar del *batch* completo, haciendo que se reduzca la varianza en la actualización del parámetro y la convergencia sea más estable, por otro lado permite que el cálculo aproveche operaciones matriciales altamente optimizadas que deben usarse en un cálculo bien vectorizado del costo y gradiente. Esta actualización (Ruder, 2016) se realiza con una alta varianza que hace que la función objetivo fluctúe mucho como en la figura 2.17 lo que le permite saltar a nuevos y potencialmente mejores mínimos locales.

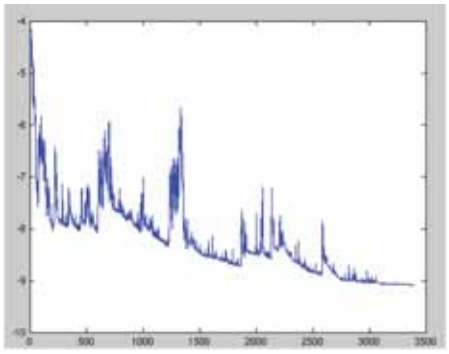


Figura 2.17: SGD fluctuación
Fuente: Ruder (2016)

Es importante destacar que el costo computacional, ecuación (2.11),en (Zachary C. Lipton, 2017) la iteración en la GD (Gradient Descent) es directamente proporcional con el tamaño del conjunto de datos de entrenamiento m, cuanto mas grande sea m el costo computacional para la GD sera mas alto. SGD da una solución, en lugar de calcular el gradiente $\nabla J(w_t; x^i, y^i)$, usa muestras aleatoriamente uniformes i y computa $\nabla J_i(w_t; x^i, y^i)$, dónde β es la cardinalidad del *minibatch* y el *learning rate* α.

$$\nabla_w J_\beta(w_t; x^i, y^i) = \frac{1}{\beta} \sum_{i \in \beta} \nabla J_i(w_t; x^i, y^i) \tag{2.14}$$

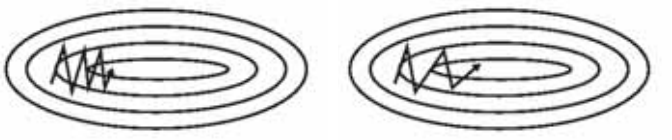
$$w_{t+1} = w_t - \alpha_t \nabla_w J_\beta(w_t; x^i, y^i) \tag{2.15}$$

El SGD (Andrew Ng, 2013) tiene un *learning rate* α que en general es mucho más pequeño que en GD porque hay mucha más variación en la actualización, es recomendable usar un *learning rate* constante lo suficientemente pequeña que proporcione una convergencia estable en la época inicial.

Momentum

La SGD tiene problemas cuando la superficie se curva es mucho más pronunciada en una dimensión comúnmente en torno a los óptimos locales, oscilando a través de las *slopes of the ravine* mientras hace progresos vacilantes a lo largo del fondo hacia el óptimo local. *Momentum* (Ruder, 2016) es un método que ayuda a acelerar el SGD en una dirección relevante y amortigua las oscilaciones. Esto se hace añadiendo γ con el vector de actualización del paso de tiempo anterior al vector de actualización actual ecuación(2.16). Esencialmente, cuando utilizamos el *momentum* aumenta para las dimensiones cuyos gradientes apuntan en las mismas direcciones y reduce las actualizaciones para las dimensiones cuyos gradientes cambian de dirección. Asi se la convergencia es más rápida y con una menor oscilación.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2.16}$$



(a) SGD sin Momentum (b) SGD con Momentum

Figura 2.18: Comparación del Momentum
Fuente: (Ruder, 2016)

Gráfico computacional

Los cálculos de las redes neuronales están organizados en términos de *forward pass* o *forward propagation* donde se calculan las salidas de la NN, seguidamente un *back pass* o *back propagation* que se utiliza para obtener las gradientes o derivadas.

Para calcular la función de *loss* (2.7) es necesario calcular las ecuaciones (2.5) y (2.2), en otras palabras la ecuación *loss* depende de ellas, la LR utilizando las gradientes descendentes es utilizada para modificar los parámetros w y b y reducir la función de perdida L. La figura 2.19 es una representación de LR, primero se realiza el *forward propagation* (flechas verdes), ingresan los parámetros x, vector de características; w y b , inicializados, para calcular z, se le aplica la función *sigmoidal* para calcular al salida final $\hat{y}$, finalmente se calcula la función *Loss* L.

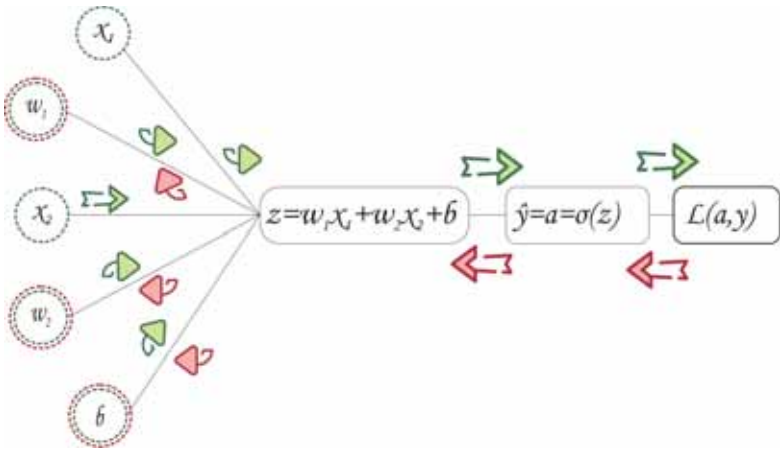


Figura 2.19: Gráfico computacional.
Fuente: Propia

El segundo paso es *back propagation* (flechas rojas) comenzando de la salida, se calcula la primera derivada, la función de salida con respeto a la variable en este caso a (2.17).

$$\frac{\partial L(a,y)}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a} \tag{2.17}$$

Se deriva el siguiente, la derivada de la función *loss*, la función de salida, con respecto a z (2.18)

$$\frac{\partial a}{\partial z} = a(1-a) \tag{2.18}$$

$$\frac{\partial L(a,y)}{\partial z} = \frac{\partial L(a,y)}{\partial a}.\frac{\partial a}{\partial z} = a - y \tag{2.19}$$

Finalmente en el *back propagation* derivamos la salida L con respecto a w y b, se realizan entonces las actualizaciones de w y b usando la tasa de aprendizaje α. Estas modificaciones de w y b afectaran a L para optimizarlo.

$$\begin{aligned} w_1 &= w_1 - \alpha\partial w_1 \\ w_2 &= w_2 - \alpha\partial w_2 \\ b &= b - \alpha\partial b \end{aligned} \tag{2.20}$$

2.5.4. Vectorización

Dentro del DL se encuentran grandes cantidades de datos para entrenar, como ya se menciona es necesario que el código se ejecute rápidamente y estos inmensos

datos se procesen lo más eficaz y veloz para encontrar una respuesta, la vectorización entonces se deshace de código explicito que hace más lento el código.

$$z = W^T x + b \tag{2.21}$$

Para la LR son necesarios los parámetros w y x en la ecuación x un conjunto muy grande de características y w el conjunto de pesos (2.22), si procedemos a operar como en la ecuación (2.21) el tiempo de ejecución será mayor a comparación de usar otras funciones, Python posee la librería cientifica *numpy* y dentro de ella el comando *dot*, que multiplicaría w y x sin necesidad de un *for* explícito.

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^1 & x^2 & \cdots & x^n \\ \vdots & \vdots & & \vdots \end{bmatrix}, W = \begin{bmatrix} \vdots & \vdots & & \vdots \\ w^1 & w^2 & \cdots & w^n \\ \vdots & \vdots & & \vdots \end{bmatrix} \tag{2.22}$$

Dentro del entrenamiento contamos con varios ejemplos de entrenamiento, varias variables z y activaciones (2.23) de tamaño (n_x, m) para poder calcular Z se construye una matriz $(1xm)$ y calculamos con el comando *dot* (2.24) en lugar de hacer un *for* o bucle; b al ser un único parámetro es necesario expandirlo realizando un *broadcasting*[2] para poder calcular Z y a al mismo tiempo en menos líneas de código.

$$\begin{aligned} &z^{(1)} = w^T x^{(1)} + b; z^{(2)} = w^T x^{(2)} + b; z^{(3)} = w^T x^{(3)} + b \\ &a^{(1)} = \alpha(z^{(1)}); a^{(2)} = \alpha(z^{(2)}); a^{(3)} = \alpha(z^{(3)}) \end{aligned} \tag{2.23}$$

$$\begin{aligned} &z = [z^{(1)}, z^{(2)}, \ldots, z^{(n)}] = w^T X + [b, b, b, \ldots, b] \\ &= [w^T x^{(1)} + b, w^T x^{(2)} + b, \ldots, w^T x^{(m)} + b] \end{aligned} \tag{2.24}$$

2.5.5. Shallow neural network

La notación que se utiliza generalmente es como se muestra en la figura 2.20, el superíndice con corchete $[i]$ se refiere a la cantidad asociada de la pila de nodos ocultos, capa 1, capa 2, capa n y el superíndice (i) hace referencia a los ejemplos de entrenamiento.
En una red de DL el conjunto de entrenamiento contiene valores de entrada x así como salidas y. para entender mejor el siguiente grafo representa el primer nodo de la figura anterior $a_1^{[1]}$, que tiene tres característica de entrada, para poder obtener el valor de $a_1^{[1]}$ calculamos la salida z.

$$\begin{aligned} z_1^{[1]} &= W_1^{[1]} x + b_1^{[1]} \\ a_1^{[1]} &= \alpha(z_1^{[1]}) \end{aligned} \tag{2.25}$$

Con este resultado visto en la ecuacion (2.25)obtenemos $a_1^{[1]}$ que en el paso posterior servirá de entrada para la capa 2, estos pasos se repiten para cada nodo del gráfico y avanza capa por capa, por supuesto que sería innecesario un bucle extenso, para esto se utilizaran técnicas de vectorización al momento de implementar.
Dentro de una red *shallow* se implementa la vectorización de la siguiente manera, con el uso de los pesos w, sea w una matriz de los pesos para cada nodo, $w_1^{[1]} = [\ldots]$

[2]El termino *broadcasting* describe como la librería *numpy* trata arreglos con diferentes tamaños al momento de realizar operaciones matemáticas

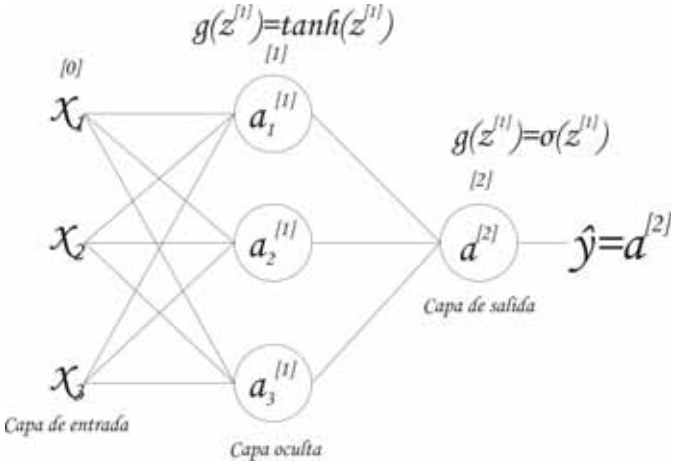


Figura 2.20: Estructura Superficial o Shallow.
Fuente: Andrew (2017)

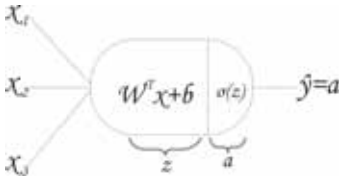


Figura 2.21: Salida de la Red.
Fuente: Andrew (2017)

usamos la transpuesta para poner realizar la multiplicación con los vectores x, si se tiene varios $w_i^{[l]T}$ siendo l el numero de capa, e i el numero de nodo, se utiliza $W^{[l]}$ para representarlos, y de igual manera X para representar a todos los $x^{(i)}$, el $i - esimo$ ejemplo de entrenamiento.

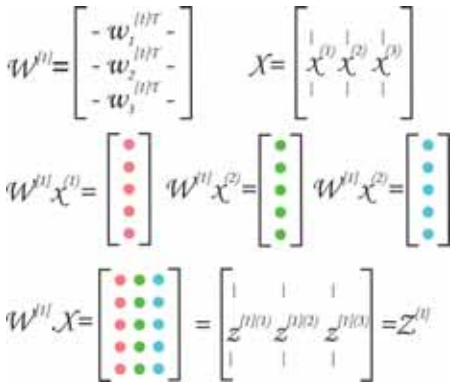


Figura 2.22: Vectorización.
Fuente: Andrew (2017)

Funciones de activación

Como ya se vio la función de activación es como una función de aplastamiento o limitación para los valores finitos de la salida. Entre las más conocidas esta la función *sigmoide* pero existen mejores funciones dependiendo del problema a resolver.

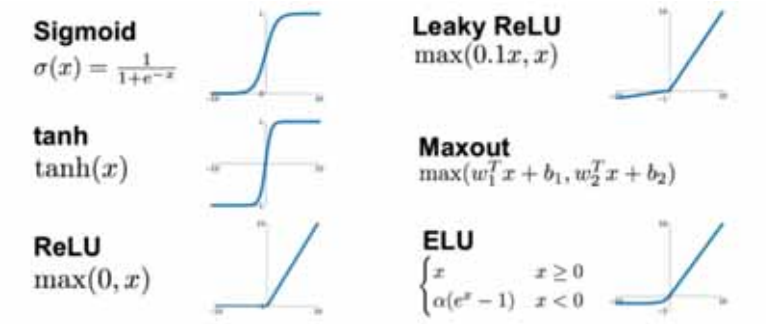


Figura 2.23: Funciones de activación
Fuente: Jadon (2018)

La función *sigmoide* es una función especial dentro del DL por su simplificación durante el *backpropagation*. En (Jadon, 2018) la función *sigmoidal* se enfrenta a las gradientes saturadas, la función oscila entre 0 y 1 y pueden permanecer contantes, las gradientes tendrán valores muy inferiores por lo que no habrá descenso en la gradiente descendente. Andrew Ng sugiere utilizarla en capas de salida en problemas de clasificación binaria. Las funciones *sigmoide* y tangente hiperbólica están relacionadas (Goodfellow et al., 2016) aunque para la utilización es mejor el uso de la tangente hiperbólica debido a que se asemeja más a la función de identidad, en el sentido de que $\tanh 0 = 0$ el entrenamiento de una DNN se asemeja al entrenamiento de un modelo lineal siempre y cuando las activaciones de la red puedan mantenerse pequeñas. Esto facilita el entrenamiento de la red tanh, aun así se enfrenta al problema de los Gradientes Saturados, (Jadon, 2018) es un buen ejemplo en caso de entradas mayores a cero, las gradientes serán todos positivos o negativos, pero conducir a un problema de *vanishing* o *exploding*. La función ReLU (*Rectified Linear Unit Activation*) es la más utilizada debido a (Jadon, 2018) su simplicidad durante el *backpropagation*, su bajo costo computacional, no tiene saturación y converge más rápido que otras funciones, el problema que se identifica aquí es el punto muerto del ReLU La función *leaky ReLu* es la versión mejorada de ReLU tiene todas las propiedades de ReLU y no tiene el problema de ReLU muerto. Se pueden formar variaciones del *leaky* ReLU La función *Maxout*, se introdujo el 2013, nunca se satura ni muere, pero es caro y duplica los parámetros. La función *KafNETS* algunas funciones como ReLU funcionan intercalando proyecciones y funciones de activación, en KAF (Kernel Activation Function) se hace el cambio con funciones no paramétricas. Generalmente las NN optan por diversas funciones de activación, (Jadon, 2018) en su mayoría ReLU por su simplicidad y facilidad de cálculo tanto en el *forward* y *backward*, en ciertos casos otras funciones de activación dan mejores resultados, como *sigmoidal* se utiliza en la capa final, cuando queremos que nuestras salidas sean aplastadas entre $[0, 1]$, o tanh se utilizen en RNNs y LSTMs.

Función softmax

La función *softmax* se define (Patterson and Gibson, 2017) como una generalización de la regresión logística que puede aplicarse a datos continuos en lugar una clasificación binaria, puede tener múltiples límites de decisión y se encuentra a menudo en la capa de salida de un clasificador. La activación *softmax* devuelve la distribución de probabilidad sobre clases de salida mutuamente excluyentes, como el reconocimiento de dígitos manuscritos de datos MNIST, donde cada etiqueta(0 - 9) es mutuamente excluyente.
Una predicción segura (Buduma and Locascio, 2017) tendra una sola entrada en el vector cercana a 1, mientras que las entradas restantes estaban cerca de 0. Una predicción débil tendría múltiples etiquetas posibles que son más o menos igualmente probables.
El propósito de la función de activación de *softmax* (Warren S. Sarle, 2002) es reforzar las restricciones en las salidas, que se encuentren entre cero y uno, y cuya suma de probabilidades es 1. Sea la entrada a cada unidad de salida q_i para $i = 1, ..., c$, donde c es el número de categorías. Entonces la salida de *softmax* p_i se representa en la ecuación(2.26), la expresión exp representa la función exponencial e^q.

$$p_i = \frac{exp(q_i)}{\sum_{j=1}^{c} exp(q_j)} \tag{2.26}$$

2.5.6. Deep neural network

Una red superficial o *shallow Neural Network* solo posee una capa, sin embargo, las redes neuronales están formadas de n capas.

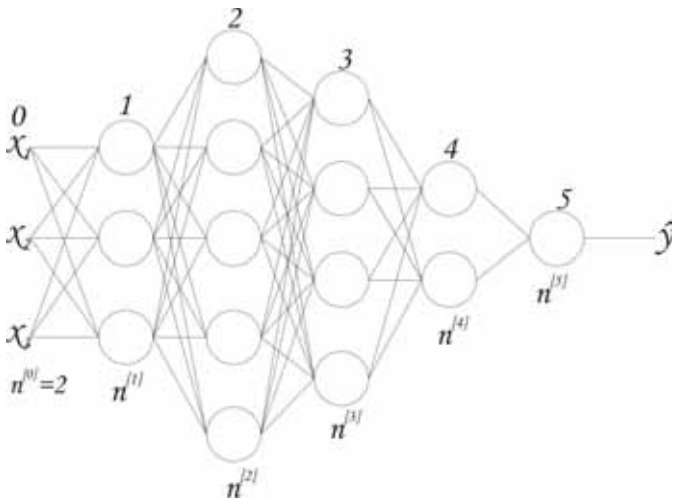


Figura 2.24: Arquitectura Deep Learning.
Fuente: Andrew (2017)

La figura 2.24 denotamos como L el número de capas, $n^{[l]}$ el numero de unidades en la capa L y $a^{[l]}$ la activación en la capa l, consideramos la capa de entrada como la capa 0, el gráfico es una DNN de 5 capas, muestra 4 capas ocultas. En DNN la forma de *forward* cambia con respecto a las redes *shallow*, ya no se utilizara la ecuación (2.25), la red se ha vuelto más profunda y se necesitan pasar todos los

estados de activación de una capa a otra en este caso se usa la vectorización con la ecuación modificada (2.27).

$$\begin{aligned} Z^{[1]} &= W^{[1]}A^{[0]} + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \end{aligned} \tag{2.27}$$

Esta ecuación dentro de un bucle para cada capa representa el *forward* de DNN. Dimensionalmente Z representa las salidas, tiene la forma de $(n^{[l]}, m)$, W de la forma $(n^{[l]}, n^{[l-1]})$ y b $(n^{[l]}, 1)$

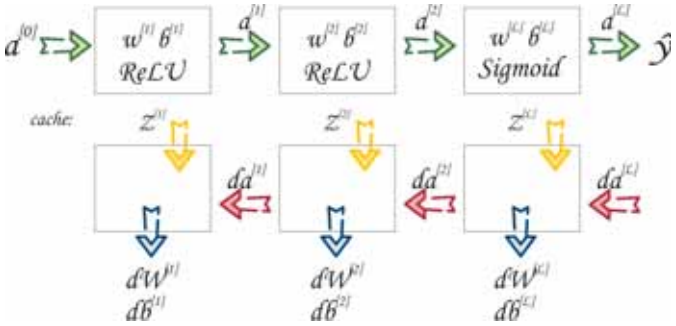


Figura 2.25: Funcionamiento Deep Learning.
Fuente: Andrew (2017)

2.6. Mejorando las redes neuronales profundas

2.6.1. Configuracion deep learning

Hacer una buena elección de los conjuntos *training*, *development* y *test* pueden hacer un mejor funcionamiento de NN, al entrenar una red se toman muchas decisiones como el numero de capas. unidades ocultas, el *learnning rate* ,la función de activación y otros hyperparametros, que deben ser ajustados iterativamente para un buen funcionamiento.
Acertar a la primera con el valor de los hiperparemetros es imposible, el desarrollo de este tipo de arquitecturas depende de la practica e experimentación. La figura 2.26 ejemplifica el ciclo que sigue el desarrollo de una arquitectura DL, comenzamos en la generación de una idea, como querer construir un NN con determinadas capas y parámetros, consecuentemente codificamos las ideas y finalmente probamos en la fase de experimentación analizando los resultados podemos variar la elección de los parámetros y volver a iniciar el ciclo tantas veces como sea necesario.

2.6.2. Conjuntos de datos

Para tener una NN funcional es fundamental tener datos con los cuales trabajar y dividir estos datos en conjuntos que ayuden en las distintas fases de la construcción de la red, en primera tenemos el *training set* o conjunto de entrenamiento que se utiliza para entrenar el algoritmo y obtener los pesos adecuados, el siguiente es el *development set* o conjunto de validación cruzada que se encarga de comparar

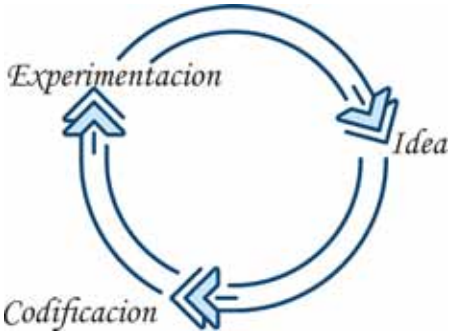


Figura 2.26: Ciclo de Desarrollo de un DNN.
Fuente: Andrew (2017)

los distintos modelos que trabajan mejor con el *dev set* y finalmente el *test set* o conjunto de prueba que después de obtener el modelo adecuado obtendrá un estimado imparcial de que tan bien funciona el modelo elegido. como una buena practica es recomendable usar una proporción de 98 %, 1 % y 1 % si se tuviera una inmensa cantidad de datos sin embargo para datos mas tradicionales es mejor la proporción 60 %, 20 % y 20 %. La elección de datos también juega un papel importante es recomendable usar datos con la misma distribución, el *training set* ajustara el modelo a ciertas características y si los datos del *test* o *dev set* son de una distribución diferente jamas se llegara un resultado aceptable

2.6.3. Métodos de regularización

Problemas frecuentes

Uno de los conceptos importantes en este tema es la compensación entre *bias* y *variance*. En la figura 2.27 se muestra una distribución de datos si se traza una linea (a) representando al LR, no es un buen ajuste para los datos, a esto se le denomina *high bias* o *underfitting*, si se traza una curva como en (c) es un ajuste mas perfecto pero no es bueno al especificar demasiado ya que el modelo solo se centrara en dichas caracteristicas, por ultimo un ajuste tipo (b) seria mas optimo, *low variance* y *bias*. Un *high varince* es producido por una significante diferencia entre los porcentajes de exactitud del *training set* y del *dev set*, un *high bias* es producido por un porcentaje significativamente alto del *traning set* y *dev set*, implica que ninguno de los conjuntos se ajusta adecuadamente.

Es común que los modelos caigan en un *overfitting* o un *underfiting*, lo importante es conocer los métodos mediante los cuales se evite o solucione dichos inconvenientes.
Para solucionar los problemas del *high bias* es posible convertir la red en otra red mas grande aumentando las unidades o capas ocultas, encontrar otra arquitectura que se ajuste mejor al problema o realizar un entrenamiento mas largo de los algoritmos de optimización. Para solucionar los problemas de *variance* es recomendable incrementar los datos, aunque no siempre este al alcance del investigador en cuyo caso se recomiendan realizar regularizaciones.

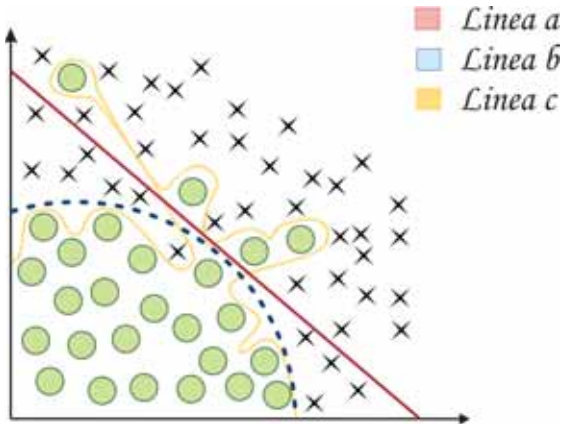


Figura 2.27: Ajuste de Datos.
Fuente: Propia

Regularización

Teniendo la función de costo (2.7) agregamos la regularización L2 también llamada *weight decay*, que penaliza los matriz de pesos por ser dimensionalmente grande, se muestra en la ecuación (2.28) que si lambda λ toma valores reales $w \approx 0$ logrando cambiar el peso de las unidades ocultas y simplificando la NN asemejándolo a un LR apilado con la misma profundidad, esto causa que el *overfiting* se parezca mas al *underfiting* pero con el valor medio de lambda λ podemos llegar al caso intermedio.

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=0}^{n}(L(\hat{y}^{(i)}, y^{(i)})) + \frac{\lambda}{2m}\sum_{l=1}^{L}((((||W||)_F)^2) \tag{2.28}$$

2.6.4. Regularización dropout

Una técnica poderosa es el *dropout* que consiste en ir por la NN cambiando la probabilidad de eliminar un nodo así para cada capa la probabilidad es aleatoria de mantener el nodo o eliminarlo y sus respectivas salidas reduciendo la NN. (Gal and Ghahramani, 2016) el *dropout* es una técnica popular de regularización donde las unidades de red se enmascaran aleatoriamente durante el entrenamiento en la figura 2.28 debe generar salidas significantes, puesto que el *dropout* elimina las entradas, el nodo no puede confiar en cualquier característica o cualquier entrada que puede irse aleatoriamente, al igual que la regularización L2 el efecto de implementar *dropout* es que disminuye el peso y algunas regularizaciones exteriores que previenen el *overfittig*.

Variational dropout

Los RNNs son modelos potentes que a falta de una regularización dificultan el manejo de datos pequeños y para evitar el *overfitting* a menudo utilizan *early stopping*. Las investigaciones recientes bayesianas y de aprendizaje profundo como las de (Gal and Ghahramani, 2016) muestran que el *dropout* puede ser interpretada como una aproximación variacional (mezcla de dos gaussianos con pequeñas variaciones y con la media de un gaussiano fijada en cero) a la parte posterior

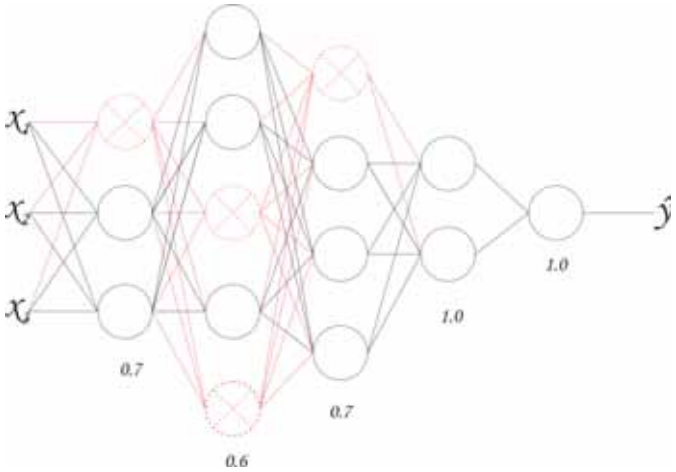


Figura 2.28: Técnica de DropOut.
Fuente: Gal and Ghahramani (2016)

de una red neural bayesiana (NN). En modelos RNN probabilísticos, se realizan inferencias variacionales aproximadas en estos modelos bayesianos probabilísticos llamadas RNNs Variacionales que conduce una optimización manejable idéntica a la realización de una nueva variante de *dropout* en RNNs.

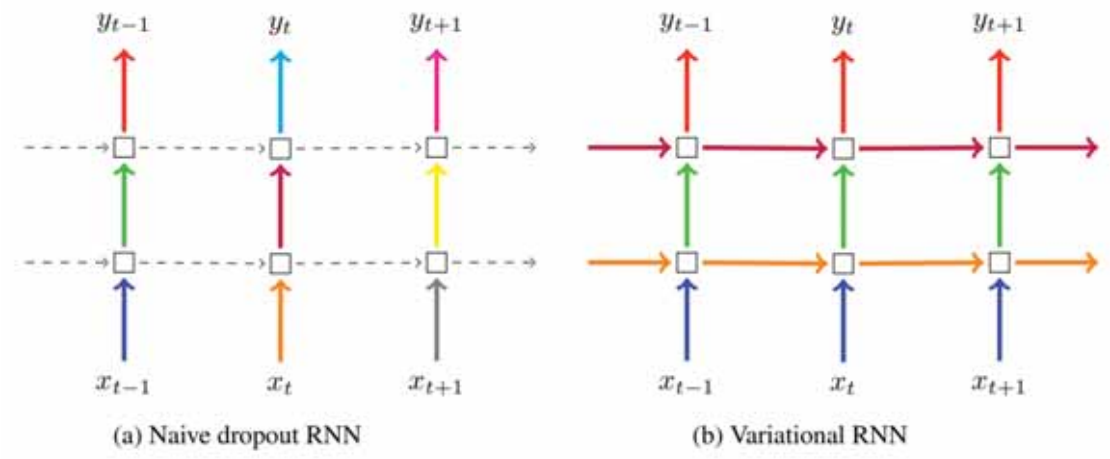


Figura 2.29: Dropout en Recurrent Neural Networks
Respresentacion de la técnica dropout siguiendo la interpretación bayesiana (derecha) en comparación con la técnica estándar en el campo (izquierda).
Fuente: Gal and Ghahramani (2016)

En figura 2.29 las conexiones de color representan entradas *dropout*, cada color corresponde a diferentes máscaras *dropout*. Las técnicas actuales (izquierda) utilizan diferentes máscaras en diferentes etapas de tiempo, sin que se produzca un *dropout* en las capas recurrentes, ya que el uso de diferentes máscaras con estas conexiones conduce a un rendimiento deteriorado según Gal y Ghahramani, mientras que el *Variational* RNN (derecha) utiliza la misma máscara de *dropout* para las entradas de la RNN, también se usa una mascara de *dropout* en los estados recurrentes de la RNN.

2.6.5. Inicialización de pesos

Para iniciar la construcción de una red es importante es inicializar correctamente la matriz de peso.

Inicialización en ceros

Se convierte en un modelo equivalente al modelo lineal. Cuando se ajusta todo el peso a 0, la derivada con respecto a la función *loss* es la misma para cada w en $W^{[l]}$, por lo tanto, todos los pesos tienen los mismos valores en la iteración subsiguiente, las unidades ocultas se vuelven simétricas y continúas durante todas las n iteraciones que ejecute y hace que su red no sea mejor que un modelo lineal.

Inicialización aleatoria

Este proceso sirve para ruptura de simetría, proporciona una precisión mayor haciendo que cada neurona tenga un calculo diferente, en este método los pesos se inicializan muy cerca de cero, pero al azar

2.6.6. Métodos de inicialización

La inicialización de los pesos de las redes neuronales es un campo de estudio completo, (Brownlee, 2018) ya que la inicialización cuidadosa de la red puede acelerar el proceso de aprendizaje

- Ceros : inicializador que genera tensores inicializados a 0.
- Unos : Inicializador que genera tensores inicializados a 1.
- Constante : Inicializador que genera tensores inicializados a un valor constante.
- RandomNormal : Inicializador que genera tensores con una distribución normal.
- RandomUniform : Inicializador que genera tensores con una distribución uniforme.
- TruncatedNormal : Inicializador que genera una distribución normal truncada.
- VarianceScaling : inicializador capaz de adaptar su escala a la forma de los pesos.
- Ortogonal : Inicializador que genera una matriz ortogonal aleatoria.
- Identidad : Inicializador que genera la matriz de identidad.
- lecun_uniform : inicializador uniforme LeCun.
- glorot_normal : Inicializador normal de Glorot, también llamado inicializador normal de Xavier.

- glorot_uniform : Inicializador uniforme de Glorot, también llamado inicializador uniforme Xavier.
- he_normal : inicializador normal.
- lecun_normal : inicializador normal de LeCun.
- he_uniform : Él inicializador de escala de varianza uniforme.

Recomendaciones

Para redes profundas,(Doshi, 2018) se puede usar una heurística para inicializar los pesos dependiendo de la función de activación no lineal. En lugar de dibujar desde la distribución normal estándar, se dibuja W desde la distribución normal con la varianza k/n, donde k depende de la función de activación aunque no resuelve completamente ayuda a mitigar los problemas de *exploding* y *vanishing gradients*.
Las activaciones mas comunes utilizadas son ReLu que se multiplica los valores aleatorios de W por la ecuación (2.29) y Glorot llamada inicialización Xavier es similar al Relu solo que el numerador es uno, ecuación (2.30)

$$\sqrt{\frac{2}{size^{[l-1]}}} \tag{2.29}$$

$$\sqrt{\frac{1}{size^{[l-1]}}} \tag{2.30}$$

2.6.7. Early stopping

Para obtener un buen rendimiento se deben ajustra los hiperparametros, para definir el hiperparametro *epoch*, pasadas completas del conjunto de datos, dependiendo de la cantidad de épocas podríamos tener un *derfit* (muy pocas épocas) o en caso contrario un *overfit*.
Early Stopping intenta eliminar la necesidad de ajustar manualmente este valor. También puede considerarse un tipo de método de regularización (como la caída de peso L1/L2 y el *dropout*) en el sentido de que puede evitar un *overfitting* en la red.

La idea básica consiste (Deeplearning4j, 2014) en dividir los datos en el *train set* y *test set*. Al final de cada época evaluar el rendimiento de la red en el *test set*, si la red supera el mejor modelo anterior se guarda una copia de la red en la época actual y se toma el modelo que tenga el mejor rendimiento en el *test set*. Lo que necesitamos es un criterio que nos diga cuándo dejar de entrenar, y que requiera el menor entrenamiento para un determinado error de generalización o que dan como resultado el menor error de generalización para un determinado tiempo de formación (Prechelt, 2012)

2.6.8. Batch normalizacion

En primera se define el cambio de covariable como el cambio de la distribución del conjunto de entrada, un cambio significativo de covariables provoca que el entrenamiento del modelo sea muy lento, una solución es uniformizar el conjunto de

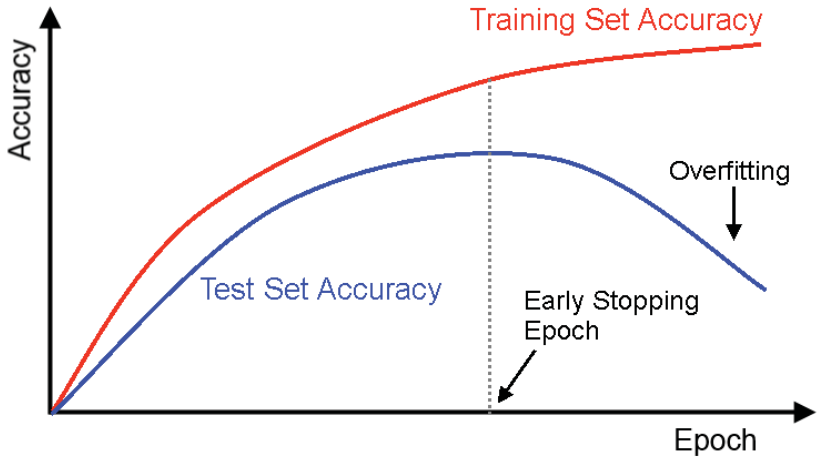


Figura 2.30: Early Stopping
Fuente: Deeplearning4j (2014)

entrada y cada unidad oculta, sin embargo cada unidad oculta cambia cada vez que hay una actualización de parámetros en la capa anterior,cambio de covariación interna, provoca que el *training* sea lento, que se necesecite un *learning rate* muy pequeño y una buena inicialización de parámetros. (Ioffe and Szegedy, 2015) propone una solución, el modelo se basa en hacer de la normalización en una parte de la arquitectura del modelo y realizar la normalización para cada *minibatch* de entrenamiento permitiendo *learning rates* mas altos y ser menos cuidadosos con la inicialización. Por otro lado actúa como un regularizador, en algunos casos eliminando la necesidad de *dropout* el resultado de este modelo muestra que se mejoró el resultado superior de ImageNet (2014) por un margen significativo utilizando solo el 7 % de los pasos de entrenamiento. El *Batch normalization* (Aggarwal, 2018) es un método para abordar los problemas de *vanish* y *explodding gradients*, que hacen que los gradientes de activación en capas sucesivas se reduzcan o aumenten en magnitud.
Para entender el funcionamiento vease figura 2.31, el BN se realiza individualmente en cada unidad. Tradicionalmente, la entrada a una capa $a^{[l-1]}$ atraviesa una transformada afín pasar por una función de activación $g^{[l]}$ para obtener la activación final $a^{[l]}$ de la unidad como en la ecuación (2.27), Pero con el *batch normalization* se utiliza con una transformación BN y se convierte en la ecuación (2.31).

$$a^{[l]} = g^{[l]}(BN(W^{[l]}a^{[l-1]})) \tag{2.31}$$

2.6.9. Adam optimization

Adam Optimization viene de la estimación del momento adaptativo, es diferente a SGD, (Kingma and Ba, 2014) es la combinación del *Adaptive Gradient Algorithm* (AdaGrad),que mantiene un *learning rate* por parámetro que mejora el rendimiento en problemas con gradientes dispersos (por ejemplo, problemas de lenguaje natural y de visión de computadora) y *RMSProp* (Propagación de la me-

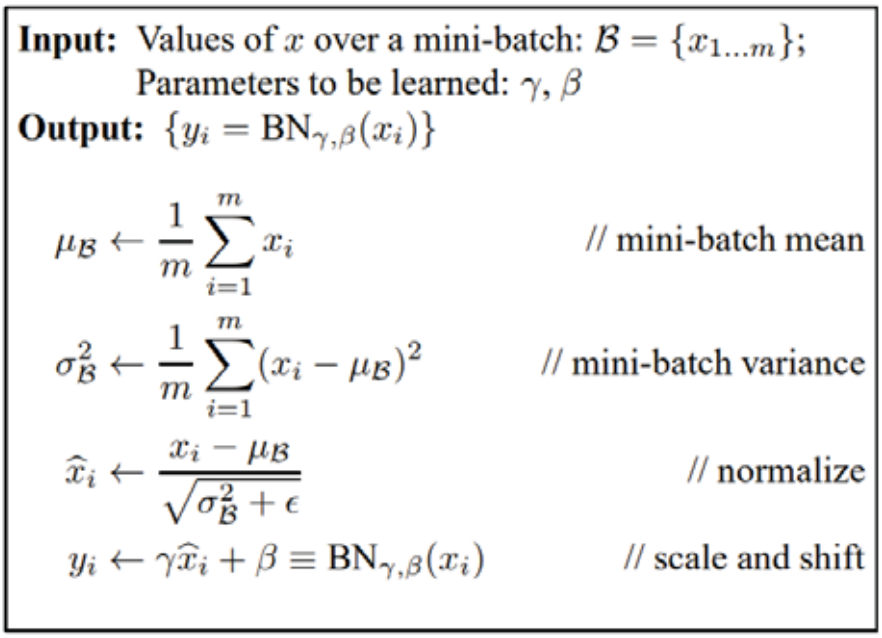


Figura 2.31: Batch Normalizing Transform
aplicado a la activación x sobre un mini-batch. Fuente: Ioffe and Szegedy (2015)

dia cuadrática de la raíz) [3] que también mantiene el learning rate por parámetro que se adaptan según el promedio de las magnitudes recientes de los gradientes para el peso, funciona bien en problemas en línea y no estacionarios (por ejemplo, ruidosos).En lugar de adaptar el learning rate de parámetros basadas en el primer momento promedio (la media) como en RMSProp, *Adam* también utiliza el promedio de los segundos momentos de los gradientes (la varianza no centrada). Es ventajoso puesto que los requisitos de memoria son relativamente bajos (aunque más altos que el GD y el GD con *momentum*), por lo general, funciona bien incluso con ajustes pequeños en los hiperparámetros. Los resultados empíricos demuestran que *Adam* funciona bien en la práctica y se compara favorablemente con otros métodos de optimización estocástica figura 2.32

Adam Optimization se calcula primero sacando un promedio ponderado exponencialmente de las gradientes anteriores y lo almacena en las variables V_{dW} y V_{db} (antes de la corrección del bias) y $V_{dW}^{corregida}$ y $V_{db}^{corregida}$ (con corrección de bias). Luego calcula un promedio ponderado exponencialmente de los cuadrados de los gradientes anteriores, y lo almacena en las variables S_dW y S_db (antes de la corrección de sesgo) y $S_{dW}^{corregida}$ y $S_{db}^{corregida}$ (con corrección de sesgo). Finalmente, los parámetros se actualizan en una dirección basada en la combinación de información del primer y segundo paso.

2.7. Transfer learning

En esta era del *Big Data* donde se cuenta con grandes cantidades de datos, aun para los nuevos modelos no es suficiente, (Ruder, 2017) los modelos aun carecen de la capacidad de generalización que son diferentes a los escenarios en los

[3] Método en el que el learning rate se adapta a cada uno de los parámetros, acelera el GD.

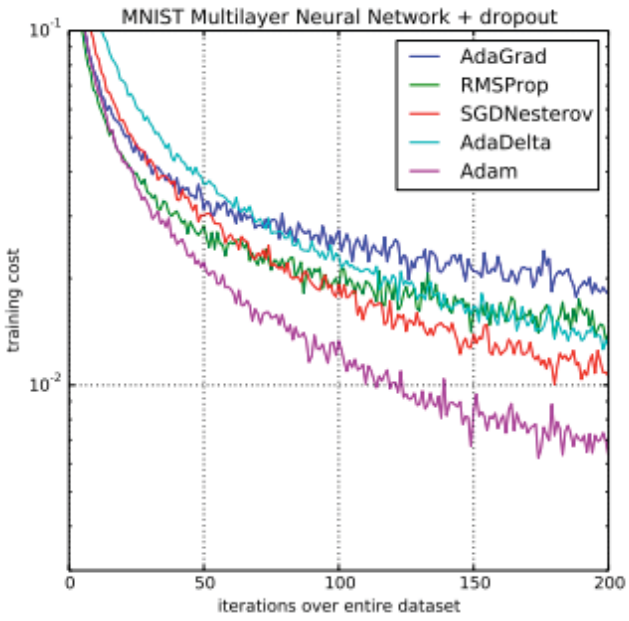


Figura 2.32: Comparacion de Adam Optimization
Comparación de Adam con otros algoritmos de optimización que entrenan un perceptrón multicapa. Fuente: Kingma and Ba (2014)

que fueron entrenados, cuando se prueben en el mundo real se encontraran mal adaptados, TL (Transfer Learning) funciona como una ayuda para lidiar con los distintos escenarios aprovechando los datos etiquetados existentes para alguna tarea o dominio, así se intenta almacenar este conocimiento y aplicarlo a nuestro problema de interés.(Donges, 2018) no es una técnica de aprendizaje automático sino una metodología de diseño de ML (Machine Learning), (Donges, 2018) TL es la reutilización de un modelo preentrenado en un nuevo problema, permite entrenar DNN con poca información.
Pese a que TL existe ya hace varios años se utiliza poco, Andrew Ng en su tutorial NIPS 2016 se refiere al TL como el próximo impulsor del éxito comercial de ML después del aprendizaje supervisado, en (Ruder, 2017) explica que esta aseveración se debe a que en los últimos años se han desarrollado modelos mas precisos casi sobrehumanos pero estos modelos están hambrientos de datos y su funcionamiento depende de enormes cantidades de datos etiquetados para lograr su rendimiento y estos datos son costos o disponibles después de años de recolección como los de ImageNet, pero muchos de estos nuevos modelos se enfrentan a condiciones que no han visto antes, es entonces que TL interviene.

2.7.1. Características

Al trabajar con una red pre-entrenada se limita en algunos aspectos en la arquitectura, por ejemplo no puede eliminar capas de CNN pero si se pueden hacer algunos cambios, debido al *parmeter sharing* (propiedad de los CNN) pueden ejecutarse con entradas de diferentes tamaños o convertir capas FC (Fully Connected) en capas convolucionales (Karpathy, 2018). Las ventajas de utilizar esta metodología es el ahorro de tiempo de entrenamiento, ya que puede llevar varias semanas,

se obtiene mejor funcionamiento de NN y no es necesario mucha información.

2.7.2. Escenarios

- Extracción de características, se toma un CNN y elimina la ultima capa FC, (Karpathy, 2018) denomina al resultado de esta capa como códigos CNN y se entrena un clasificador lineal para el nuevo conjunto de datos.
- Reutilización o Ajuste, se realiza un ajuste de pesos en ves de reemplazar o volver a entrenar mientras continua el *backpropagation*, se ajustan las capas que se desean.
- Modelo pre-entrenados, usualmente los modelos demoran semanas en ser entrenados por lo que existen modelo que ya fueron entrenados y que pueden ser ajustados al nuevo modelo.

Algunos modelos pre-entrenados que se pueden utilizar son Inception V3, Microsoft ofrece modelos a través de paquetes como MicrosoftMLR, otros modelos muy eficientes son ResNet y AlexNet.

2.7.3. Uso

Depende del tamaño del conjunto de datos y su similitud con el conjunto de datos original, (Karpathy, 2018) sugiere las siguientes reglas:

1. Si el conjunto es pequeño no es bueno ajustar el CNN por que provocaría un overfitting y si el conjunto es igual al origen las características de mayor nivel o posteriores de CNN serán relevantes en el conjunto de datos.
2. Si el conjunto de datos es grande y similar al conjunto original se puede hacer un ajuste completo.
3. Si los datos son pequeños y diferentes al conjunto original se recomienda hacer clasificador lineal y no entrenar el nuevo modelo desde la parte superior de la red que es la parte mas especifica.
4. Finalmente si el conjunto es grande y muy diferente se puede con toda confianza entrenar desde cero la CNN aunque también se puede entrenar teniendo una base ya entrenada.

2.8. Redes convolucionales

2.8.1. Convolutional neural network

¿Por qué convolución?

La visión por computadora es un área que cogió impulso por el aprendizaje profundo, esto es debido a que con las redes convoluciones CNN la extracción de características para el análisis es mas eficiente.
La ventaja según (Andrew, 2017) de usar CNN en lugar de FC son el uso compartido de parámetros y la escasez de conexiones. El uso compartido de parámetro

hace referencia aun detector de características como un detector de bordes, que es útil en una parte de la imagen, entonces puede ser útil en otras partes de la imagen y las escasas conexiones es útil para cada capa cuyo valor de salida dependa de un numero pequeño de parámetros. Todo lo mencionado hace que la CNN tenga menos parámetros para entrenar y sea menos propensa al *overffiting*

Detección de bordes

Llamada también por su traducción en ingles en muchos libros el *edge detection* de los bloques fundamentales de CNN, las capas de una NN podrían detectar vértices, haciendo que las capas mas profundas partes mas complejas de la imagen como rostros, manos u objetos.
Si tenemos una imagen en escala de grises, una sola capa, y se construye otra matriz de fXf denominada filtro, que aplicando la operación de convolución, se obtiene una imagen de nXn.
La detección de bordes se realiza aplicando un tipo de filtro a una matriz de imagen con píxeles oscuros y claros, al aplicar el filtro y detectar el cambio se obtiene el borde, esta es una forma de capturar las características de una imagen con CNN.

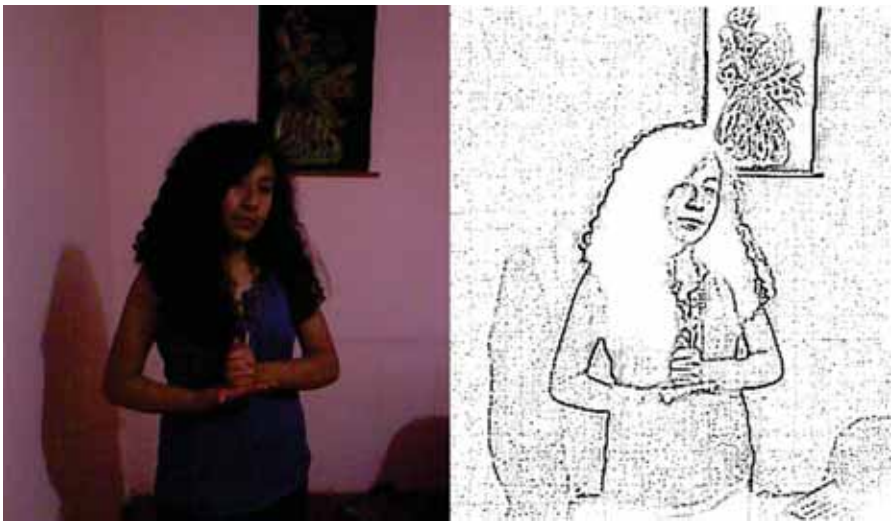


Figura 2.33: Detección de bordes.
Fuente: Propia

Padding

Las modificaciones que se hacen en una CNN hacen que el filtro se puede mover una cantidad determinada de veces y si se aplica mas filtros se ira reduciendo hasta un tamaño de 1x1. Las desventajas de aplicar los filtros son que la imagen resultado se ira reduciendo cada vez mas y que los píxeles en el borde no se utilizan tanto como los píxeles centrales. Para solucionar esto se hace uso del *padding*, que consiste en rellenar el borde de la matriz de la imagen con ceros así al aplicar una convolución no se encoja o pierda información.

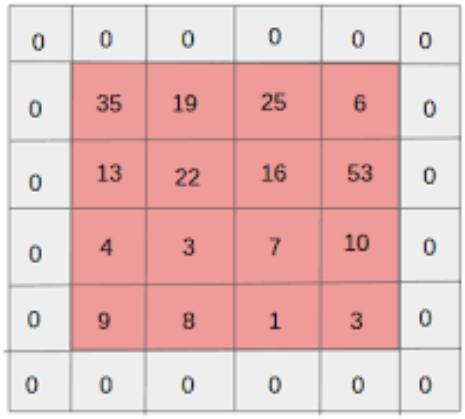


Figura 2.34: Ejemplo de padding
Fuente: Saxena (2016)

Convolución stride

Otra parte importante de CNN son los strides que son las posiciones que recorre tanto en vertical como en horizontal, este stride se calcula utilizando la ecuación (2.32).

$$\left\lfloor \frac{n + 2p - f}{s} \right\rfloor \tag{2.32}$$

Donde n es la dimensión de la imagen, p el numero *padding*, f es la dimensión del filtro y s el numero *stride*.

Convolución sobre volumen

Para hacer una convolución no solo en imágenes en escala de grises sino a color denominadas también imágenes RGB, también por regla se debe aplicar un filtro de tres capas, cada filtro corresponde a cada capa de la imagen RGB para identificarlos la terminología seria $heightXweigthX\#canales$.
Como se ve en la figura 2.35 se puede representar como un cubo tridimensional, la operación es simple, se tiene la imagen en RGB representada por el cubo de la izquierda en sus 3 canales RGB (6x6x3), posteriormente se aplica el filtro (3x3x3) sobre el cubo de 6x6x3, como se ve en la posición izquierda superior punteada, se recorre asi hasta cubrir la imagen en base al *stride*, cada filtro se multiplicara con su correspondiente canal, el resultado de esta convolución se sumara y sera el primer numero de la matriz resultante.
Dentro de las CNN no solo se necesita detección de bordes sino de distintas características, por lo que es posible aplicar a la misma imagen múltiples filtros, cada filtro tendrá su salida correspondiente y estas se apilaran formando un resultado tridimensional, como se ve en la figura 2.35 el numero de canales del resultado representara la cantidad de filtros aplicados a la imagen original.

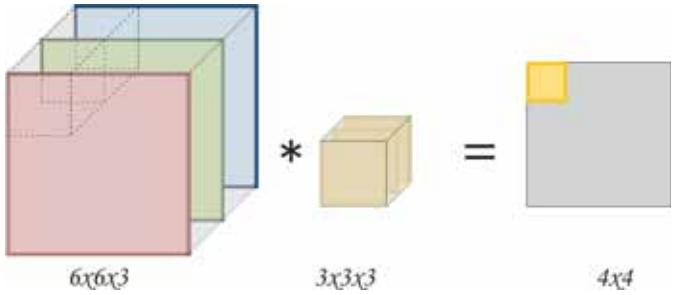


Figura 2.35: Ejemplo de convolución con filtros
Fuente: Propia

Capas pooling

Las CNN a menudo usan *pooling layers* que reduce el tamaño de la representación para acelerar el calculo, supongamos una matriz de 4x4 y aplicamos un *maxpooling* la salida seria de 2x2, es dividida en regiones y cada salida es el máximo correspondiente de cada región 2x2.
Consiste en detectar características en distintas regiones de la imagen si detecta un numero grande entonces significa probablemente que encontro una característica significativa entonces se detecta la característica en cada uno de los cuadrantes de la misma manera. Otra forma del *pooling* es el *average pooling* que funciona igualmente solo que se saca el promedio, generalmente el *maxpooling* es el mas usado excepto cuando se trabaja con DNN y se desea colapsar el resultado.

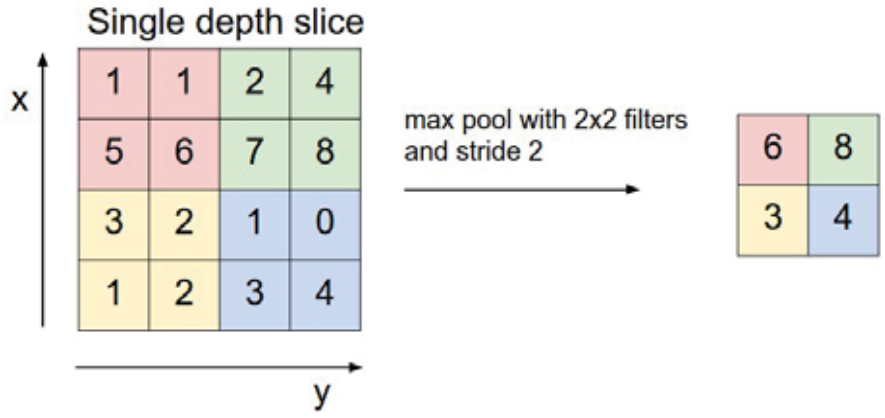


Figura 2.36: Ejemplo de stridde
Fuente: Prabhu (2018)

Capa de red convolucional

Para ver el funcionamiento de la CNN en base a las formulas de NN ya explicadas en la figura 2.35 aumentamos el *bias*[4] al primer filtro, después se aplica la función de activacion, esto para cada filtro que existiera. La primera imagen de 6x6x3 sera la entrada $a^{[0]}$, los filtros son equivalentes a los pesos W, se adiciona el

[4]Es un numero real perteneciente a la ecuación (2.27), no se debe confundir con el *bias* en el capitulo de regularización

bias, se aplica la función no lineal y se obtiene la activación $a^{[1]}$ para la siguiente capa, esta seria la equivalencia a la ecuación *forward*. 2.27.
Como ya se menciono en la CNN previene el *overffiting* asiendo que si en una capa hay 10 filtros de 3x3 tendríamos 27 parámetros, se incluye la cantidad de capas (3), mas el *bias* 28 para cada filtro lo que nos da 280 parámetros, así no importa que tan grande sea la imagen dependerá de los filtros con cuantos parámetros deseamos trabajar.

Red convolucional simple

La figura 2.37 esta inspirada en una NN clásica llamada LeNet-5 creada por Yann Lean y (Andrew, 2017), se considera *conv*1 y *pool*1 como una solo capa por convención ya que las capas debes poseer pesos y *pool*1 no las posee.
Al final de la capa 2 se agrega el *pool*2 convirtiéndolo en un vector de 400X1, luego se conectan densamente 400 unidades con 120 unidades en una conexión denominada FC, es similar a las NN simples vistas anteriormente.

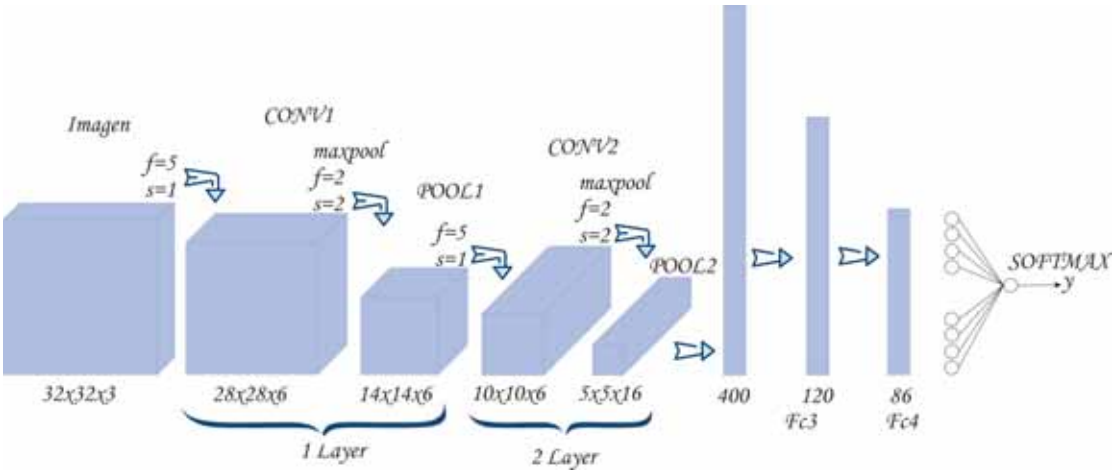


Figura 2.37: Ejemplificación de un CNN.
Fuente: Andrew (2017)

2.8.2. ResNets

En publicación de ResNets (He et al., 2016) se demuestra que la profundidad de la red es de crucial importancia. Cuando las redes más profundas comienzan a converger se expone un problema de degradación, con la profundidad de la red aumentando la precisión, se satura y luego se degrada rápidamente, esta degradación no es causada por *overfitting*. Agregar más capas a un modelo adecuadamente profundo conduce a un mayor error de entrenamiento.

La degradación de la precisión de entrenamiento indica que no todos los sistemas son igual de fáciles para optimizar. Considerando una arquitectura *shallower* y su contra parte un modelo deeper que agrega más capas sobre esta.(He et al., 2016) plantea una solución para la construcción de un modelo más profundo: las capas agregadas son un mapeo de identidad y otras capas se copian del modelo *shallower* aprendido, véase figura 2.39 esta solución indica que el modelo más profundo no

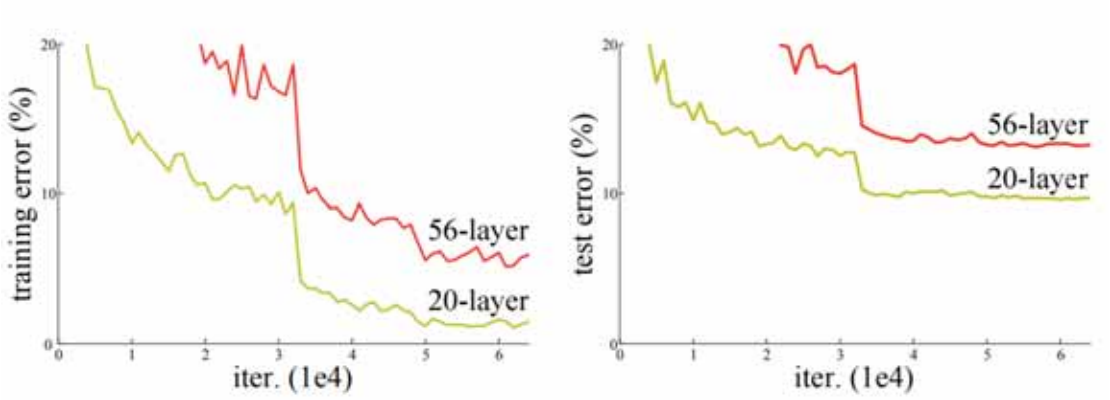
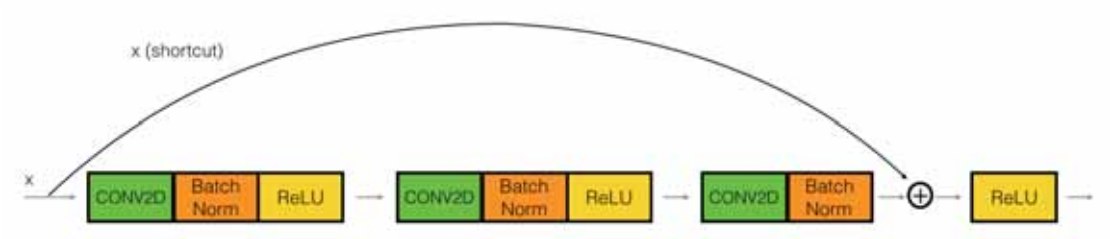
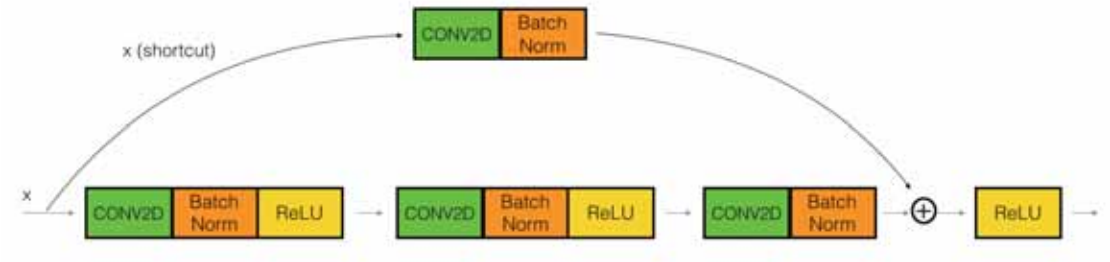


Figura 2.38: Error de entrenamiento ResNet.
Error de entrenamiento(izquierda) y error de prueba(derecha) en CIFAR-10 con redes planas de 20 y 56 capas, la red profunda tiene un mayor error de entrenamiento. Fuente:He et al. (2016)

debería producir un error de entrenamiento mayor que su contra parte *shallower*. En lugar de esperar que cada cantidad pequeña de capas apiladas se ajusten directamente a un mapeo subyacente,se deja explícitamente que estas capas se ajusten a un mapeo residual.



(a) ResNet con mapeo de identidad



(b) ResNet con mapeo de convolucional

Figura 2.39: Componenetes del ResNet
Fuente: Ferriere (2017)

Se denota el mapeo subyacente deseado como $H(x)$, dejamos que las capas apiladas no lineales se ajusten a otro mapeo de $F(x) = H(x) - x$, el mapeo original es remodelado con $F(x) + x$, plantean la hipótesis que es más fácil optimizar el mapeo residual que optimizar el mapeo original no referenciado.

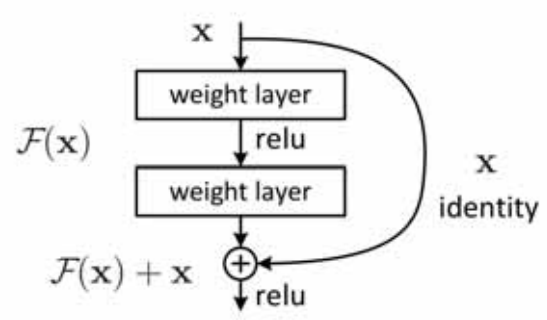


Figura 2.40: Estructura Basica ResNet.
Fuente: He et al. (2016)

La formulación de $F(x)+x$ puede realizarse mediante NN *feedforward* con conexión de acceso directo en este caso solo realizan un mapeo de identidad y sus salidas se agregan a las salidas de capas apiladas, las conexiones de acceso directo de identidad no agregan parámetros adicionales, ni complejidad computacional, toda la red puede ser entrenada de extremo a extremo por SGD con *backpropagation*. Las ventajas de ResNet se centran en dos puntos:

- Las redes residuales extremadamente profundas son fáciles de optimizar, pero las redes simples homologas muestran mayor error de entrenamiento al aumentar la profundidad.
- Las redes residuales pueden disfrutar fácilmente ganancias de precisión a partir de una gran profundidad.

Aprendizaje residual

Se considera a $H(x)$ como el mapeo subyacente o profundo, para ser ajustado por unas cuantas capas apiladas, x son las entradas a la primera de estas capas, si se parte de la hipótesis de que múltiples capas no lineales pueden aproximarse de forma asintótica a funciones complicadas, entonces es equivalente a pensar que pueden aproximarse de forma asintótica a las funciones residuales, entonces en lugar de esperar que se aproximen a $H(x)$, las capas se aproximan a una función residual $F(x) = H(x) - x$, convirtiéndose la función original en $F(x) + x$

Mapeo de identidad por accesos directos

Se considera un bloque de construcción definido como:

$$Y = F(X, W_i) + X \tag{2.33}$$

En la ecuación(2.33) X e Y son los vectores de entrada y salida de las capas consideradas. La ecuación(2.33) representa el mapeo residual que se debe aprender. La operación $F + x$ se realiza mediante una conexión de atajo y adición de elementos adoptando la segunda no linealidad después de la adición: $\sigma(y)$ Las conexiones de acceso directo en la ecuación(2.33) no introducen ni parámetros adicionales ni complejidad de cálculo que es mejor que las redes simples que tienen el mismo

número de parámetros, profundidad, ancho y costo computacional. Las dimensiones de X y F deben ser iguales en la ecuación(2.33), en caso contrario se realiza una proyección lineal W por las conexiones de acceso directo o se usa una matriz cuadrada W_s en la ecuación ecuación(2.33).

$$y = F(x, W_i) + W_s x \tag{2.34}$$

La forma de la función residual F es flexible, son posibles más capas. Pero si F tiene solo una capa, la ecuación(2.33)) es similar a una capa lineal:$y = W_1 x + x$

Red simple

Las líneas de base simple en la figura 2.41 se inspiran en redes VGG. Las capas convolucionales tienen en su mayoría filtros $3x3$ y siguen dos reglas de diseño simples: para el mismo tamaño de mapa de características de salida, las capas tienen la misma cantidad de filtros y si el tamaño del mapa de características se reduce a la mitad, el número de filtros se duplica para preservar la complejidad del tiempo por capa.
Se realiza una reducción de muestreo directamente mediante capas convolucionales que tienen una *stride* de 2. La red finaliza con una capa global de *average pooling* y una capa FC de 1000 unidades con *softmax*. El número total de capas ponderadas es 34 en la figura 2.41 (centro).

Red residual

En base a la red simple anterior, insertamos conexiones de acceso directo que convierten la red en su versión residual homóloga. Los accesos directos de identidad ecuación 2.33 se pueden usar directamente cuando la entrada y la salida tienen las mismas dimensiones, caso contrario se usan bloques de convolución (atajos de línea continua en la figura 2.41).

2.8.3. Inception

En (Lin et al., 2013) se presenta el modelo original del Inception, el filtro de convolución de una CNN se denomina GLM (Generalized Linear Model), proporciona modelos profundos con un nivel de abstracción bajo, para mejorar esto se reemplaza el GLM con un aproximador de función no lineal más potente, mejorando la abstracción, en NIN (Network in Network) el GLM se reemplaza con una estructura de micro red, que es un aproximador de función universal y una NN que se puede entrenar por *backpropagation*.
Las CNN tradicionales consisten en capas convoluciónales alternas y de *pooling* cuya salida se denomina mapa de características. El Mlpconv (Multi Layer Perceptron despues de la capa Convolucional) mapea la porción de entrada al vector de características de salida con un perceptron multicapa MLP (Multi Layer Perceptron).

Los mapas de características se obtienen deslizando el MLP sobre la entrada de manera similar a un CNN, alimentando posteriormente a la siguiente capa. La estructura de NIN se basa en el apilamiento de múltiples capas mlpconv, se llama

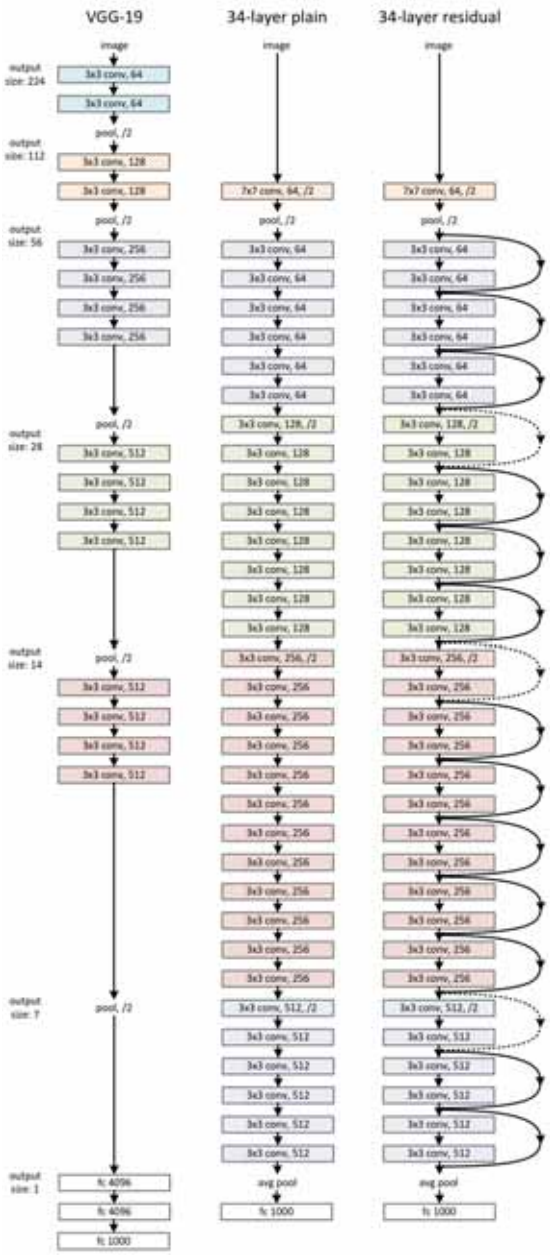


Figura 2.41: Estructura Basica ResNet.
Fuente:He et al. (2016)

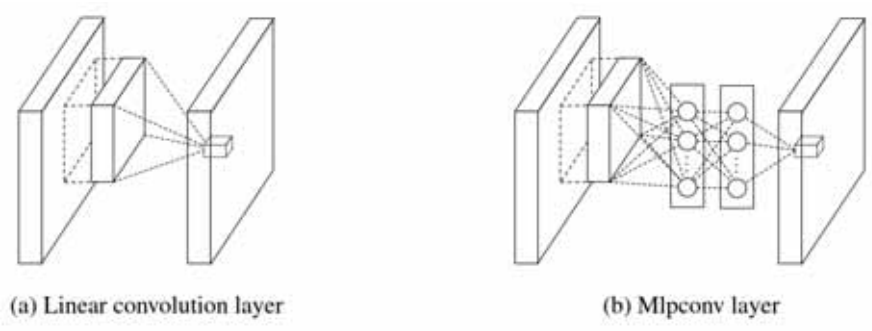


Figura 2.42: Comparación de la capa de convolución lineal y la capa de mlpconv. La capa de convolución lineal incluye un filtro lineal mientras que la capa de mlpconv incluye una micro red (perceptron multicapa). Fuente:Lin et al. (2013)

red en red ya que se tiene redes micro MLP que componen elementos de red dentro de las capas mlpconv.
Se genera directamente el promedio espacial de los mapas de características de la última capa mlpconv de categoría a través de un *global average pooling layer*, y luego el vector resultante alimenta a la capa *softmax*. En la CNN tradicional, es difícil interpretar cómo se pasa la información de nivel de categoría de la capa de costo objetivo a la capa de convolución anterior debido a las capas totalmente conectadas actúan como una caja negra en el medio, por el contrario, *global average pooling* es más significativa e interpretable, ya que impone la correspondencia entre los mapas de características y las categorías, lo que es posible gracias a un modelo local más sólido que utiliza la micro red. Además, las capas completamente conectadas son propensas a *overfitting* y dependen en gran medida de la regularización *dropout*, mientras que el *global average pooling* es un regularizador estructural, lo que previene el *overfitting* de la estructura general.

Global average pooling

NIN propone esta estrategia para reemplazar las capas tradicionales FC de CNN. La idea es generar un mapa de características para cada categoría correspondiente de la tarea de clasificación en la última capa de mlpconv. En lugar de agregar capas completamente conectadas en la parte superior de los mapas de características, tomamos el promedio de cada mapa de características, y el vector resultante se alimenta directamente a la capa de *softmax*. Una ventaja de *global average pooling* sobre las capas totalmente conectadas es que es más nativa de la estructura de convolución, al imponer correspondencias entre los mapas de características y las categorías. Por lo tanto, los mapas de características se pueden interpretar fácilmente como categorías de mapas de confianza, no hay ningún parámetro para optimizar *global average pooling*, por ende se evita el *overfitting* en esta capa, resume la información espacial y es más sólida para las traducciones espaciales de la entrada.

Estructura network in network

La estructura general de NIN es una pila de capas de mlpconv, además de *global average pooling* y la capa de costo objetivo. Se pueden agregar capas de

submuestreo entre las capas de mlpconv como en las CNN y maxout. La Figura 2.43 muestra un NIN con tres capas de mlpconv. Dentro de cada capa de mlpconv, hay un perceptron de tres capas. El número de capas tanto en NIN como en las micro redes es flexible y puede ajustarse para tareas específicas.

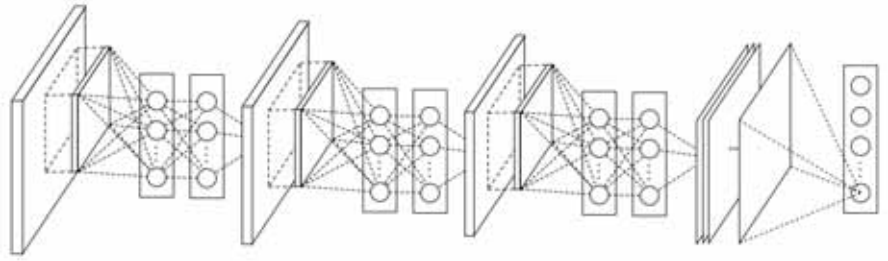


Figura 2.43: La estructura general de Network In Network
En este documento, los NIN incluyen el apilamiento de tres capas de mlpconv y una capa de agrupación global promedio. Fuente:Lin et al. (2013)

2.9. Modelos secuenciales

En los últimos años los modelos como RNN mejoraron el *speach recognition* y el NLP. Los modelos secuenciales son de diferente tipo, ejemplificando el reconocimiento de voz se realiza alimentando el modelo y este asigne una transcripción de texto, ambos son modelos secuenciales al ser la entrada un archivo de audio que se reproduce en el tiempo y la transcripción una secuencia de palabras; en un modelo de *activity recognition* se da como entrada una secuencia de videos en *frames* para reconocer en texto la actividad.
Al trabajar con NLP la secuencia de palabras por ejemplo: "Hola como estas? son representadas en un vocabulario o diccionario, este vocabulario es creado con el análisis al *training set* observando las palabras mas comunes o una investigación mas acorde a la investigación que se realiza.

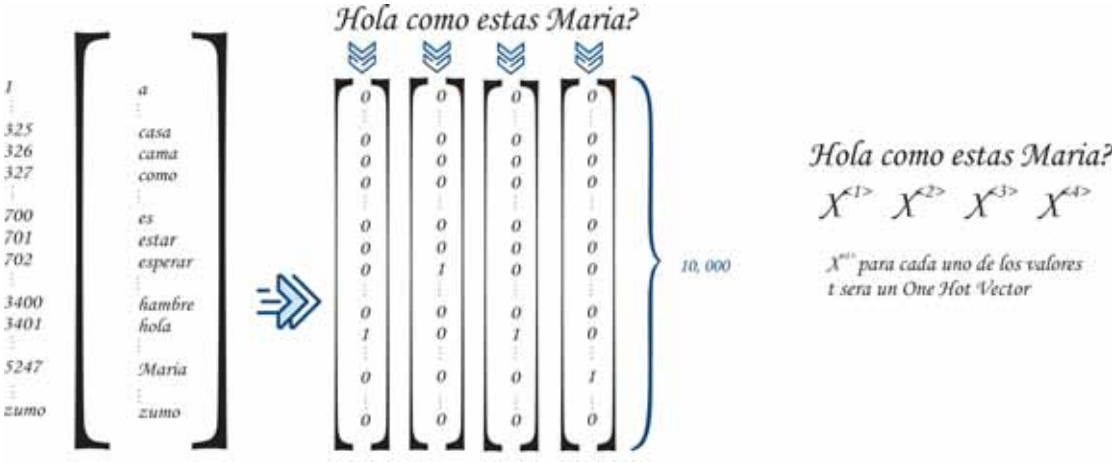


Figura 2.44: Representación de One Hot Vector.
Fuente: Andrew (2017)

2.9.1. Redes neuronales recurrentes

Si se hace uso de NN clásicas para resolver problemas secuenciales surgen dos principales problemas según Andrew (2017) las longitudes de entrada serian de distinto tamaño y no se comparten características entre las diferentes posiciones a través del tiempo. Las RNN cubre estas características, como se ve la figura 2.45 la entrada $x^{<1>}$ alimenta la primera capa como cualquier NN para predecir la salida $y^{<1>}$, cuando ingresa la segunda entrada $x^{<2>}$ para obtener $y^{<2>}$ tambien ingresa informacion del paso de tiempo anterior, la activacion $a^{<1>}$ del paso de tiempo 1 al paso de tiempo 2 y así susesivamente con el paso temporal hasta el final $T_x = T_y$.

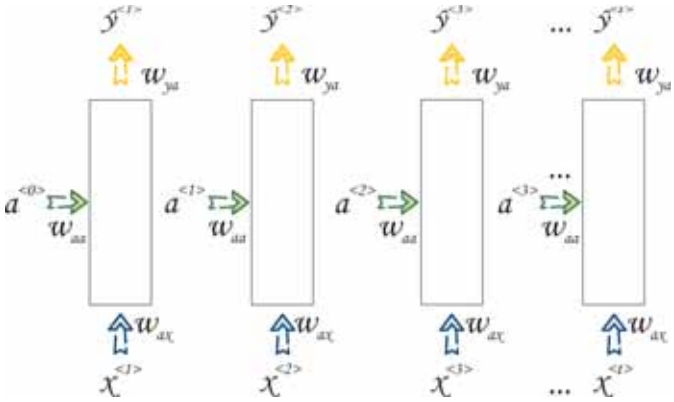


Figura 2.45: Representación del modelo secuencial.
Fuente: Andrew (2017)

Forward y backward propagation

Las ecuaciones (2.35) para *forward* tienen funciones de activacion diferentes para $a^{<t>}$ que es tanh o ReLU, mientras que para $y^{<t>}$ depende del problema.

$$\begin{aligned} a^{<t>} &= g_1(W_a[a^{<t-1>}, x^{<t>}] + b_a) \\ \hat{y}^{<t>} &= g_2(W_{ya}a^{<t>} + b_y) \end{aligned} \tag{2.35}$$

Usualmente al implementar el *backprop* lo hace el *framework*, esto se hace automaticamnente pero para el entendimiento se muestra en la figura 2.46, primero para calcular el *backprop* calculamos el costo para un paso temporal, el costo de la primera entrada es igual a la ecuación (2.11), como ya se vio en el *backprop* solo requiere de cálculos y análisis de mensajes en dirección opuesta, figura 2.46 esto le permite calcular las cantidades apropiadas y realizar la actualización de los parámetros haciendo uso del GD. Terminologicamente el *backprop* en RNN se denomina *backprop through the time*.

2.9.2. Problema vanishing gradient

Uno de los problemas mas comunes en las RNN es que se encuentran con un problema de *vanishing gradient*, entradas grandes que alimentan la red pueden tener componentes dependientes en los extremos, es decir que las primeras entradas de una secuencia temporal $X^{<t>}$ afectan a algunas entradas del extremo final de

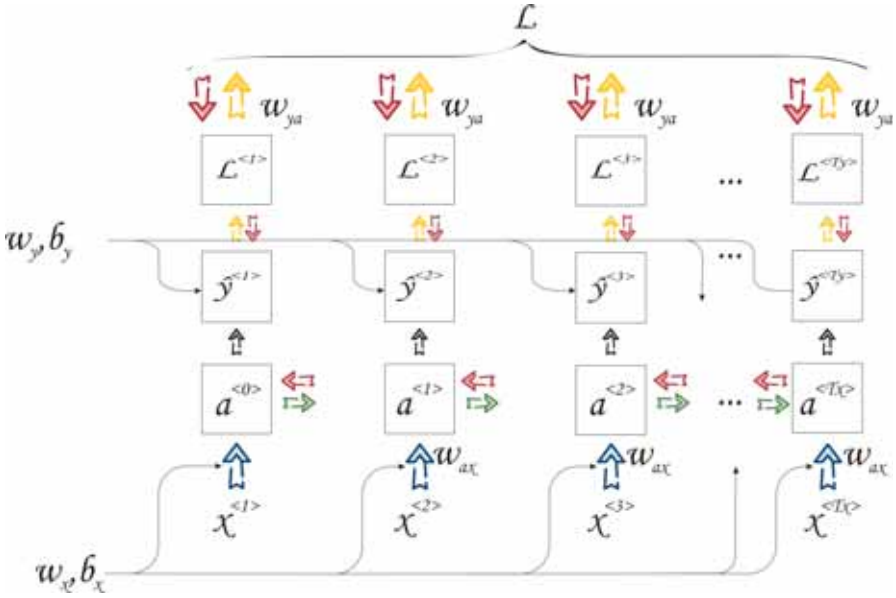


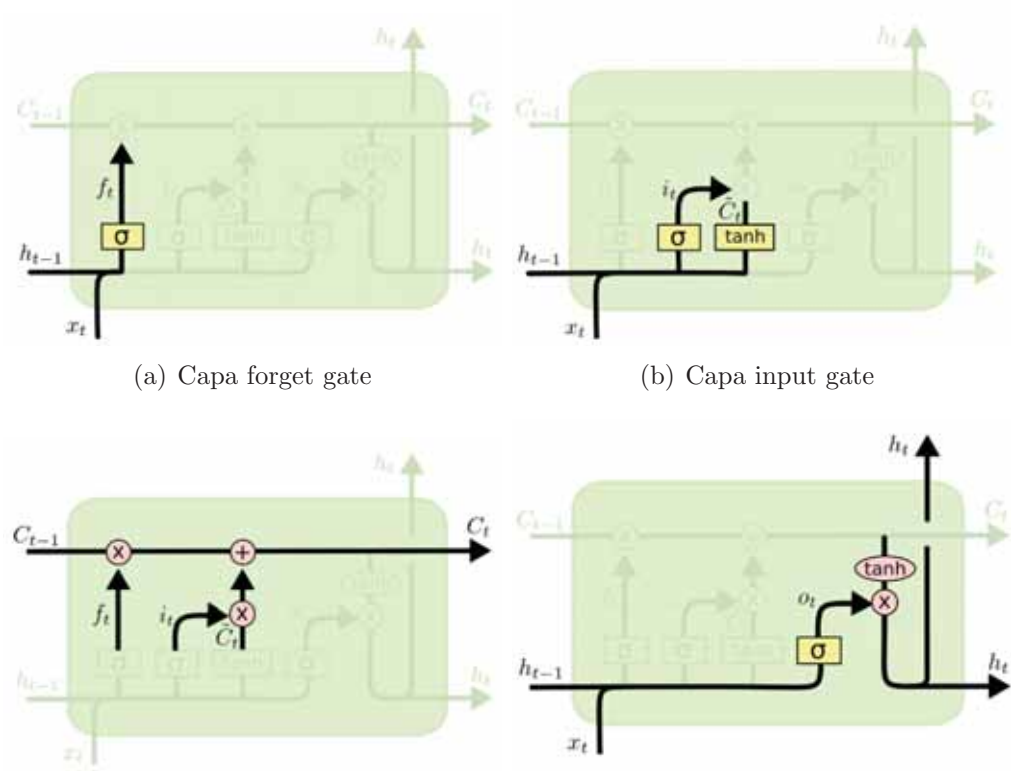
Figura 2.46: Representación de Backpropagation.
Fuente: Andrew (2017)

la entrada, podría presentar dependencias a muy largo plazo o *long-term dependencies*. Esto sucede porque en una red con demasiadas capas puede ser difícil el *backward proppagation*, debido al *vanishing gradients*, el error asociado al resultado $\hat{y}^{<Ty>}$ afecta al error de las salidas anteriores $\hat{y}^{<1>}, \hat{y}^{<2>}, \cdots, \hat{y}^{<T_y-1>}$. Esto en la practica significa que la RNN no se da cuenta que necesita capturar ciertas entradas porque de ellas dependerá las salidas posteriores que se hallan al extremo.

LSTM

Una forma de solucionar el *vanishing gradient* es el LSTM o su version reducida GRU (Gated Recurrent Units).

$$\begin{aligned} \tilde{c}^{<t>} &= \tanh(W_c[h^{<t-1>}, x^t] + b_c) \\ \Gamma_u &= \sigma(W_u[h^{<t-1>}, x^t] + b_u) \\ \Gamma_f &= \sigma(W_f[h^{<t-1>}, x^t] + b_f) \\ \Gamma_o &= \sigma(W_o[h^{<t-1>}, x^t] + b_o) \\ c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * \tilde{c}^{<t-1>} \\ h^{<t>} &= \Gamma_o * \tanh\left(c^{<t>}\right) \end{aligned} \tag{2.36}$$



(a) Capa forget gate

(b) Capa input gate

(c) Cell Memory

(d) Recortado y reducido

Figura 2.47: Funcionamiento LSTM
Fuente: Olah (2015)

En (Olah, 2015) explica el funcionamiento, el primer paso es decir que información va ser descartada del *cell state* para esto se usa la capa *sigmoide* denominada *forget gate*, figura 2.47(a), toma los valores $h_{<t-1>}$ y x, dando valores entre 1, mantendrá el valor de *cell state* y 0 descartara el valor de *cell state*. El siguiente paso decide que información se almacenara en el *cell state*, en primera la capa *sigmoide*, *input gate* en la figura 2.47(b), decide qué valores se actualizan y la capa tanh creara un vector con nuevos candidatos, $\tilde{C}_t$, que podría agregarse al estado. En el siguiente paso se combina para crear una actualización de estado, el tercer paso actualizara C_{t-1} a C_t, multiplicamos el *cell state* anterior por f_t olvidando los valores anteriores, se añade $i_t * \tilde{C}_t$ a los nuevos valores *cell state*, en función de cuánto decidimos actualizar cada valor de estado. Finalmente para generar la salida se utilizara el *cell state* en la capa *tanh* y se establece los valores entre 1 y -1, se multiplica por la salida del *output gate* y así tendremos los nuevos $h_{<t>}$.

Residual LSTM

Para crear un RNN profunda es necesario apilar capas, sin embargo se puede caer en el problema de *exploding and vanishing gradient*, basados en las conexiones residuales ya mostradas, conexión de salto de identidad entre capas adyacentes, (Su et al., 2017) incorpora una conexión residual de una capa de LSTM, figura 2.48 El cuadro naranja representa una celda LSTM, donde H_i representa la entrada, el estado oculto y la funcion LSTM, asociados con la i-esima capa LSTM, $i = (1, 2, 3, ..., L)$ y W_i es el peso de H_i. El elemento de entrada de la i-esima capa

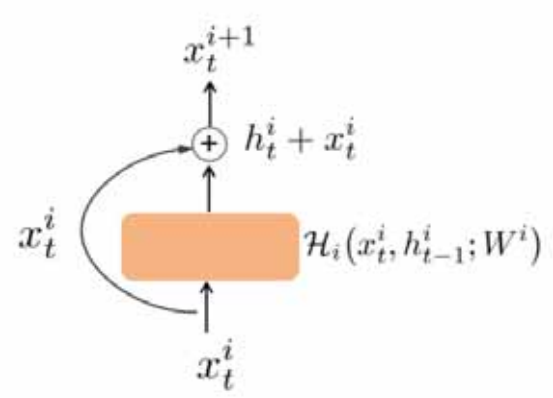


Figura 2.48: LSTM con Residual Network
Fuente: Su et al. (2017)

LSTM x_t^i es sumanda al estado oculto de la i-esima capa LSTM h_t^i, esta suma denotada por x_t^{i+1} alimentara a la siguiente capa LSTM. Un bloque LSTM residual es implementado con las ecuaciones (2.37)

$$\begin{aligned} h_t^i &= H_i(x_t^i, h_{t-1}^i; W^i) \\ x_t^{i+1} &= h_t^i + x_t^i \\ h_t^{i+1} &= H_{i+1}(x_t^{i+1}, h_{t-1}^{i+1}; W^{i+1}) \end{aligned} \tag{2.37}$$

Una vez calculado el estado oculto del *toplayer*, se puede obtener la salida z_t en la ecuación 2.38

$$z_t = \sigma(W_{hz}^L h_t^L + W_{xz}^L x_t^L + b^L) \tag{2.38}$$

2.9.3. RNN bidireccional

Es una herramienta poderosa que nos permitira tomar informacion de la secuencia anterior y posterior.

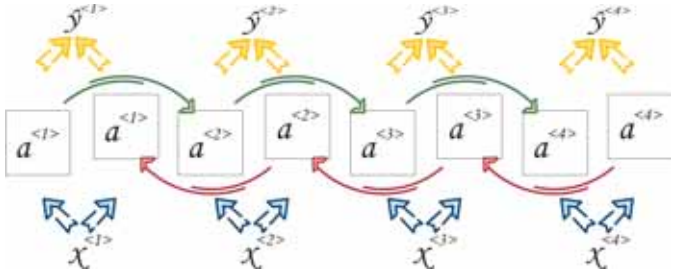


Figura 2.49: RNN Bidirectional.
Fuente: Andrew (2017)

En una RNN normal como se muestra en la figura 2.45 una salida $\hat{y}^{<t>}$ depende de los pasos de tiempo anteriores, a ese tipo de red se le denomina unidireccional, por lo contrario la figura 2.49 posee cuatro entradas $X^{<t>}$, componentes *forward*, de izquierda a derecha y *backward*, derecha a izquierda, $a^{<t>}$; cada uno de estos componentes se alimenta de $X^{<t>}$ para generar su $Y^{<t>}$. Una BRNN

(Bidirectional Recurrent Neural Network) posee componentes *forward* que calcularan primero $a^{<1>}, a^{<2>}, ..., a^{<ty>}$, mientras que la secuencia *backward* sera $a^{<ty>}, a^{<ty-1>}, ..., a^{<1>}$, este ultimo no representa el *backward propagation* , es parte del *forward propagation* que esta BRNN se divide en dos componentes. Finalmente habiendo calculado todas las activaciones pueden hacer predicciones $\hat{y}$. Según la ecuación (2.40) la salida $\hat{y}$ dependerá de la función de activación aplicada al peso W_y siendo alimentada con las activaciones de *forward* y *backward* $\overrightarrow{a}^{<t>}, \overleftarrow{a}^{<t>}$ para un tiempo $< t >$

$$a' = [\overrightarrow{a}^{<t>}, \overleftarrow{a}^{<t>}] \tag{2.39}$$

$$\hat{y}^{<t>} = g(W_y \cdot a' + b_g) \tag{2.40}$$

Donde a' puede ser una función de concatenación, adición , multiplicación y de esta manera la BRNN permite que la predicción en el momento t tome información del pasado, presente y futuro. (Andrew, 2017) generalmente en problemas de NLP una BRNN con LSTM es mas común, sin embargo la desventaja de BRNN es que es necesario tener toda la secuencia de datos para poder realizar la predicción, un ejemplo que nos pone es que en un red de reconocimiento de audio sera necesario que la persona termine de hablar para obtener la pronunciación completa antes de poder procesarla y realizar una perdición de voz.

2.9.4. Deep RNN

Hasta el momento solo se ha visto RNN de una sola capa, $a^{[l]<t>}$ donde l representa la capa y t el espacio temporal, la figura 2.50 muestra una red mas profunda de 3 capas, para este caso por ejemplo para $a^{[2]<3>}$ se realiza una activación con $a^{[1]<3>}$ y la salida $a^{[2]<2>}$. Aunque la RNN mostrada ahora no parecer muy profunda trabajar con mas de 3 capas ya es muy costoso debido a la dimensión temporal que se maneja en las redes secuenciales.

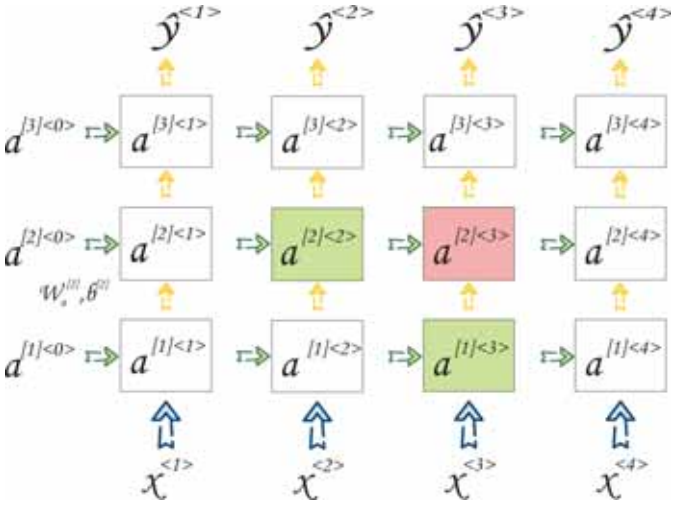


Figura 2.50: Deep RNN, 3 capas.
Fuente: Andrew (2017)

2.9.5. Attention model

Es un solución al modelo *encoder* - *decoder* visto en el paper (Cho et al., 2014), el modelo consta de dos submodelos, un codificador que es responsable de recorrer los pasos de tiempo de entrada y de codificar la secuencia completa en un vector de longitud fija llamado vector de contexto y un decodificador responsable de recorrer los pasos de tiempo de salida mientras lee el vector de contexto figura 2.51.

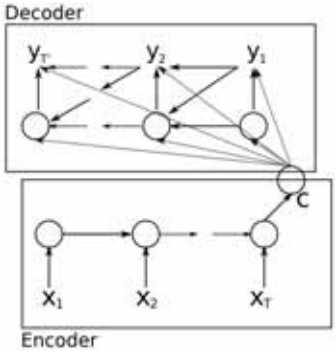


Figura 2.51: Modelo Encoder-Decoder de red neuronal recurrente.
Fuente: Cho et al. (2014)

El modelo de attention presentado en (Bahdanau et al., 2014) se introduce una extensión al modelo *encoder* − *decoder* que aprende a alinear y traducir conjuntamente. Cada vez que el modelo propuesto genera una palabra en una traducción, busca un conjunto de posiciones en una oración fuente donde se concentra la información más relevante. El modelo luego predice una palabra de destino en función de los vectores de contexto asociados con estas posiciones de origen y todas las palabras de destino generadas anteriormente.
En lugar de decodificar la secuencia de entrada en un único vector de contexto fijo, el modelo de atención desarrolla un vector de contexto que se filtra específicamente para cada paso de tiempo de salida, figura 2.52.

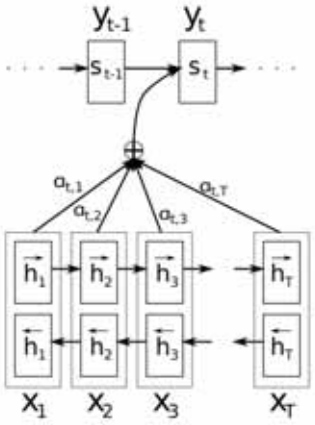


Figura 2.52: Ejemplo de Attention Model
Fuente: Bahdanau et al. (2014)

Parte III

Desarrollo del proyecto

Capítulo 3

Dataset y arquitectura propuesta

3.1. Tipos de datos

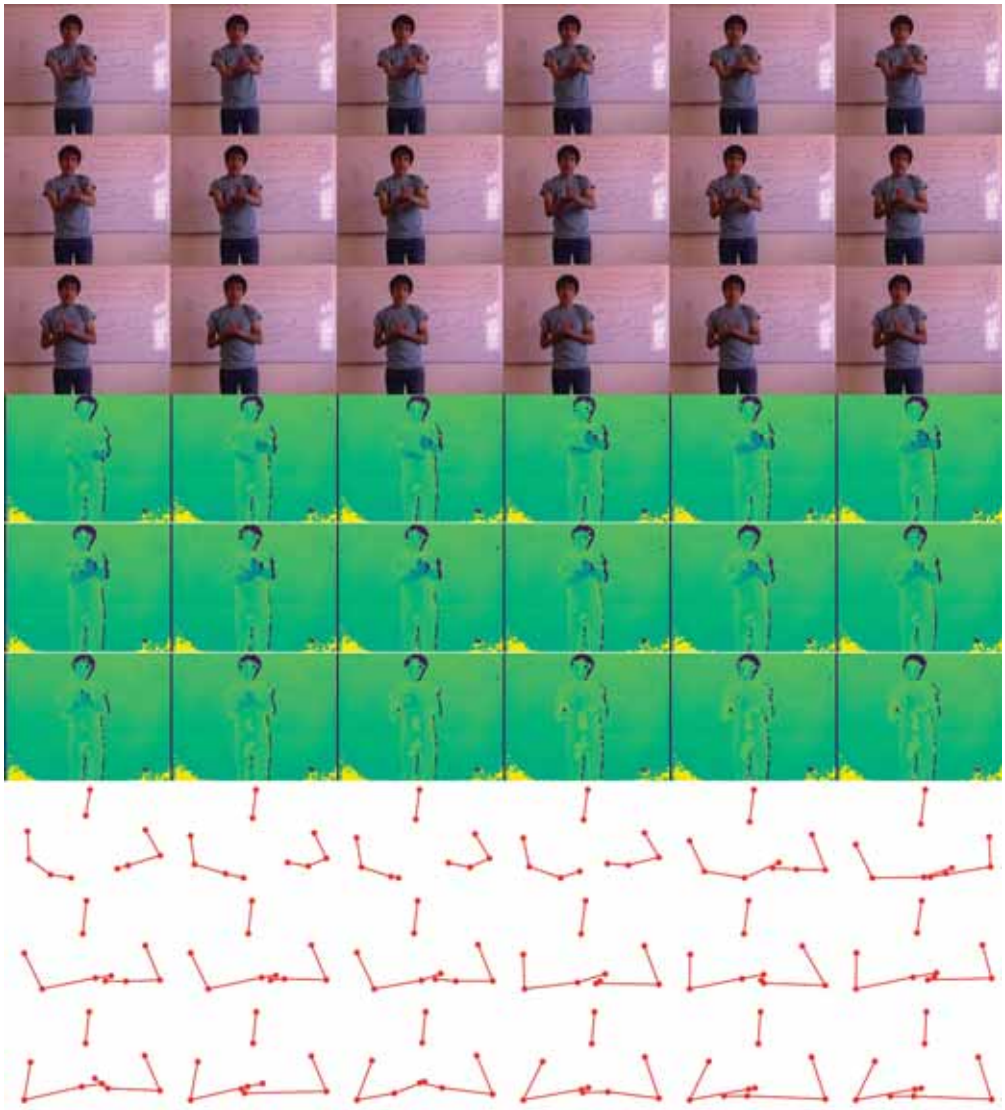


Figura 3.1: Tipos de datos del *dataset* VideoLSP10.
RGB(superior), *depth* (centro) y *skeleton* (inferior). Fuente: Propia

3.1.1. Datos Red-Green-Blue

Se refiere a la composición de colores primarios (red, green, blue), también conocidos mejor por su abreviatura RGB. Para este trabajo, los datos RGB son cada uno de los fotogramas (*frames*) en una secuencia de vídeo, donde se distingue las características de la mano,su ubicación y posición del cuerpo, véase figura 3.1.

3.1.2. Datos *depth*

Flujo de información que contiene píxeles de distancia en lugar de colores entre el punto objetivo y el sensor kinect. En la figura 3.1 se puede observar los datos *depth* como imágenes, donde morado indica los píxeles mas cercanos al sensor y amarillo los mas lejanos.

3.1.3. Datos *skeleton*

Estos datos son obtenidos mediante el rastreo de esqueleto del sensor Kinect, son un conjunto numérico de 20 puntos de articulación que forman el esqueleto representados en coordenadas (X,Y,Z), sin embargo los datos capturados para este trabajo usa el modo *seated skeleton*, para capturar únicamente los puntos por encima de la cintura. La figura 3.1 muestra una representación de los datos *skeleton* para un mejor entendimiento, no obstante es necesario redundar que el flujo de datos *skeleton* no son imágenes sino coordenadas tridimensionales.

3.2. Dataset VideoLSP10

(a) Ayudame

(b) Por favor

(c) Disculpame

(d) Donde vives tu?

(e) No entiendo

(f) Que hace tu?

(g) Hasta mañana

(h) Gracias

(i) Cual es tu nombre?

(j) Hola como estas tu?

Figura 3.2: Vocabulario del VideoLSP10.
Fuente: Propia

Se realizaron varias elaboraciones de *datasets* antes de llegar a proponer *Dataset* VideoLSP10, que es una base de datos obtenida mediante un sensor *Kinect* v1 para el *traning* y *validation* de este trabajo, los datos son de tipo RGB, *depth* y *skeleton*[1], véase figura 3.1, donde cada *frame* rgb y depth tiene una resolución de (480X640) y para los datos *skeleton* se considero 10 puntos de unión del cuerpo humano. Debido a la estructura del trabajo, VideoLSP10 esta conformado por tres partes, VideoLSP10_Depth, VideoLSP10_Join y VideoLSP10_Total.

3.2.1. VideoLSP10_Depth

VideoLSP10_Depth contiene datos tipo *depth*, con un total de 2045 imágenes de distancia, el cual se divide en 14 clases. Es importante aclarar que no son imágenes en si, sino una matriz de distancias capturadas como se indica en la subsección 2.1.1, en la tabla 3.1, "Ayudame" consta de 173 matrices de distancias. El *dataset* se usara para el entrenamiento de la arquitectura DepthResnet50 que se hablara mas adelante.

Etiqueta	**# Imagenes**	**Etiqueta**	**# Imagenes**
Ayúdame	173	Como	210
Cual	160	Disculpa	60
Donde	185	No entiendo	152
Estas	195	Gracias	96
Hola	152	Mañana	99
Nombre	135	favor	230
que	99	tu	99

Tabla 3.1: Dataset VideoLSP10_Depth

3.2.2. VideoLSP10_Join

VideoLSP10_Join contiene datos tipo *skeleton*, consta de 1701 movimientos el cual se divide en 21 clases, cada movimiento esta conformado por 10 coordenadas (x, y, z), donde cada coordenada son ubicaciones de unión de la parte superior del cuerpo humano como ya se menciono, véase tabla 3.3. VideoLSP10_Join consta de movimientos de la mano izquierda el cual a cada etiqueta se le antepone con la vocal "i", la mano derecha con la letra "d" y ambas manos con la letra "a", estos movimientos son dirigidos a ciertos puntos (arriba, delante, izquierda, derecha, cabeza, boca y centro o pecho), por ejemplo en la tabla 3.2 existen 81 secuencias para el mismo gesto "iarriba" que muestra un movimiento de la mano izquierda hacia arriba. Los datos *skeleton* son arreglos con longitud 60, el cual representan los puntos de unión del cuerpo humano. El *dataset* se usara para el entrenamiento de la arquitectura SkeletonResnet50 que se hablara mas adelante.

[1]Puntos de unión de las articulaciones del cuerpo humano representados por coordenadas (x,y,z) capturadas por el Kinect

Gesto	# secuencias	Gesto	# secuencias
iarriba	81	darriba	81
aarriba	81	iadelante	81
dadelante	81	aadelante	81
ii	81	di	81
ai	81	id	81
dd	81	ad	81
icabeza	81	dcabeza	81
acabeza	81	iboca	81
dboca	81	aboca	81
icentro	81	dcentro	81
acentro	81		

Tabla 3.2: Dataset VideoLSP10_Join

3.2.3. VideoLSP10_Total

VideoLSP10_Total contiene datos RGB, *depth* y *skeleton*; consta de 600 vídeos de LSP divididas en 10 frases comúnmente utilizadas en la vida cotidiana, seleccionadas bajo nuestro propio criterio. Cada frase consta de 60 vídeos, capturados en ambientes con luz natural, artificial y con diferentes intensidades, véase tabla 3.3. Al capturar un gesto se obtendrán un conjunto de datos RGB, *depth* y *skeleton* con la misma cantidad de *frames*, dependiendo de la complejidad del gesto la cantidad de *frames* es variable, véase figura 3.3. El *dataset* se usara para el entrenamiento de la arquitectura LSP que se hablara mas adelante.

Gestos	# vídeos
Ayúdame	60
Por favor	60
Discúlpame	60
Cual es tu nombre ?	60
Donde vives tu ?	60
No entiendo	60
Que haces tu ?	60
Hola, como estas tu ?	60
Gracias	60
Hasta mañana	60

Tabla 3.3: Dataset VideoLSP10_Total

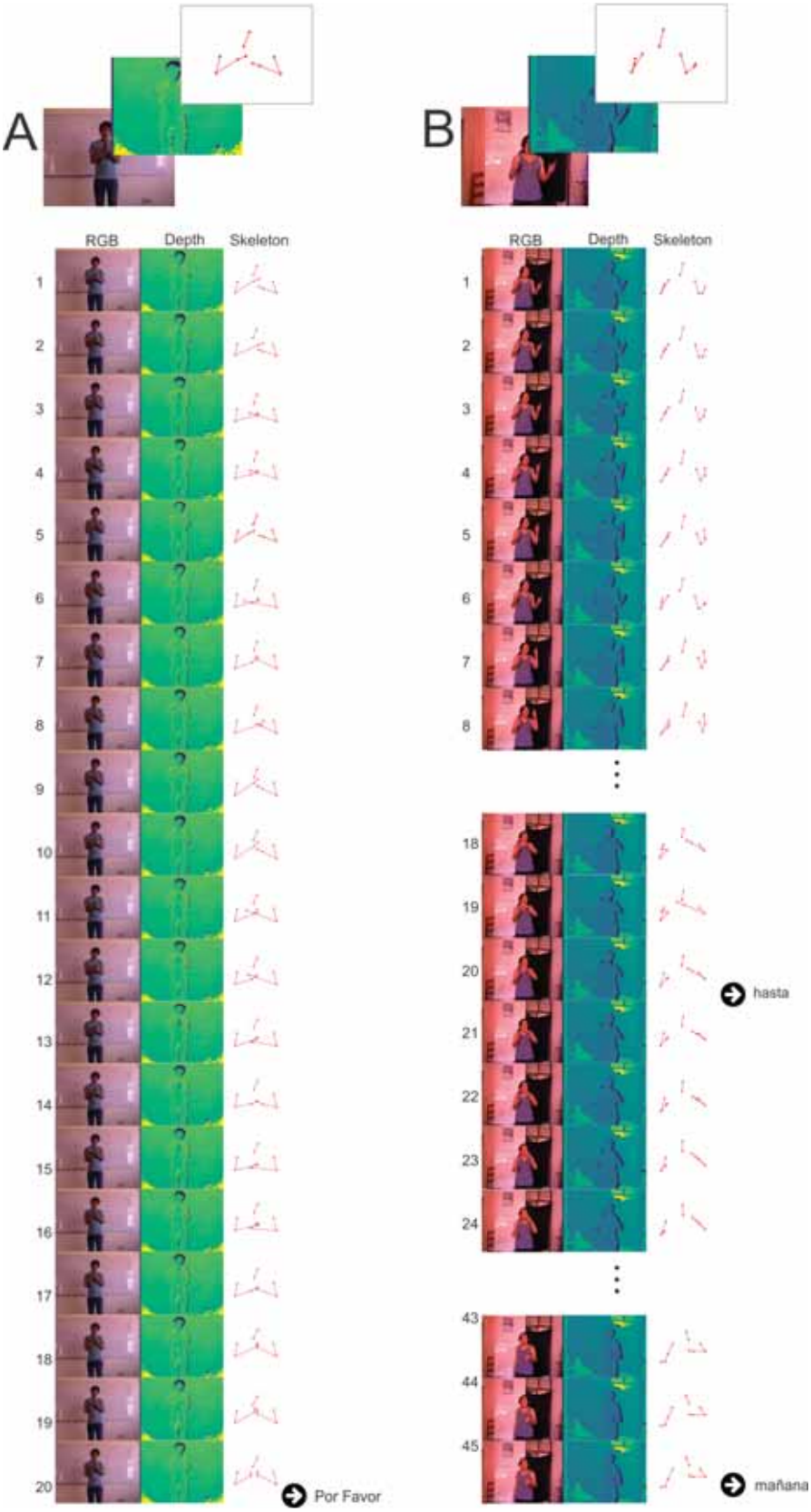


Figura 3.3: Captura de datos para el dataset VideoLSP10_Total.
Fuente: Propia

3.2.4. Distribución del conjunto de datos

Se realiza una validación cruzada *Holdout*[2], para las redes convolucionales Depth-ResNet50, Skeleton-ResNet50 y para la arquitectura LSP final. Para la red Depth-ResNet50 la división de datos se realizo de la siguiente manera: el 96 % para el *training* y 4 % para la *validation* como se aprecia en la figura 3.2.4(a). Para la red Skeleton-ResNet50 la división de datos se realizo de la siguiente manera: el 90 % para el *training* y 10 % para la *validation* como se aprecia en la figura 3.2.4(b). Finalmenete la division de datos para la arquitectura final LSP se dividió de la siguiente manera: el 84 % para el *training* y 16 % para la *validation* como se aprecia en la figura 3.2.4(c).

(a) Distribución de datos DepthResNet50

(b) Distribución de datos SkeletonResNet50

(c) Distribución de datos Arquitectura LSP

Figura 3.4: Distribución de los *datasets*.
Fuente: Propia

3.3. Arquitectura propuesta

Durante estos últimos años la inteligencia artificial y el *machine learning* tuvieron una considerable mejora en varios campos y conjuntamente se presentaron muchas técnicas para resolver un determinado problema.
La arquitectura propuesta consta de 4 fases como se ve en la figura 3.5. En la figura se identifica la región de *preprocessing*, *feature extraction CNN*, *encoder BLSTM* (bidirectional long short term memory) y finalmente el *attention decoder* que esta formado por un *attention mechanism, decoder LSTM, maxout y softmax*.
La arquitectura propuesta se desarrollo mediante un proceso experimental de prueba y error como se aprecia en la sección 4.1. Cada sección de la arquitectura propuesta es descrita a continuación.

[2]Division de los datos en dos partes: *training* y *validation*

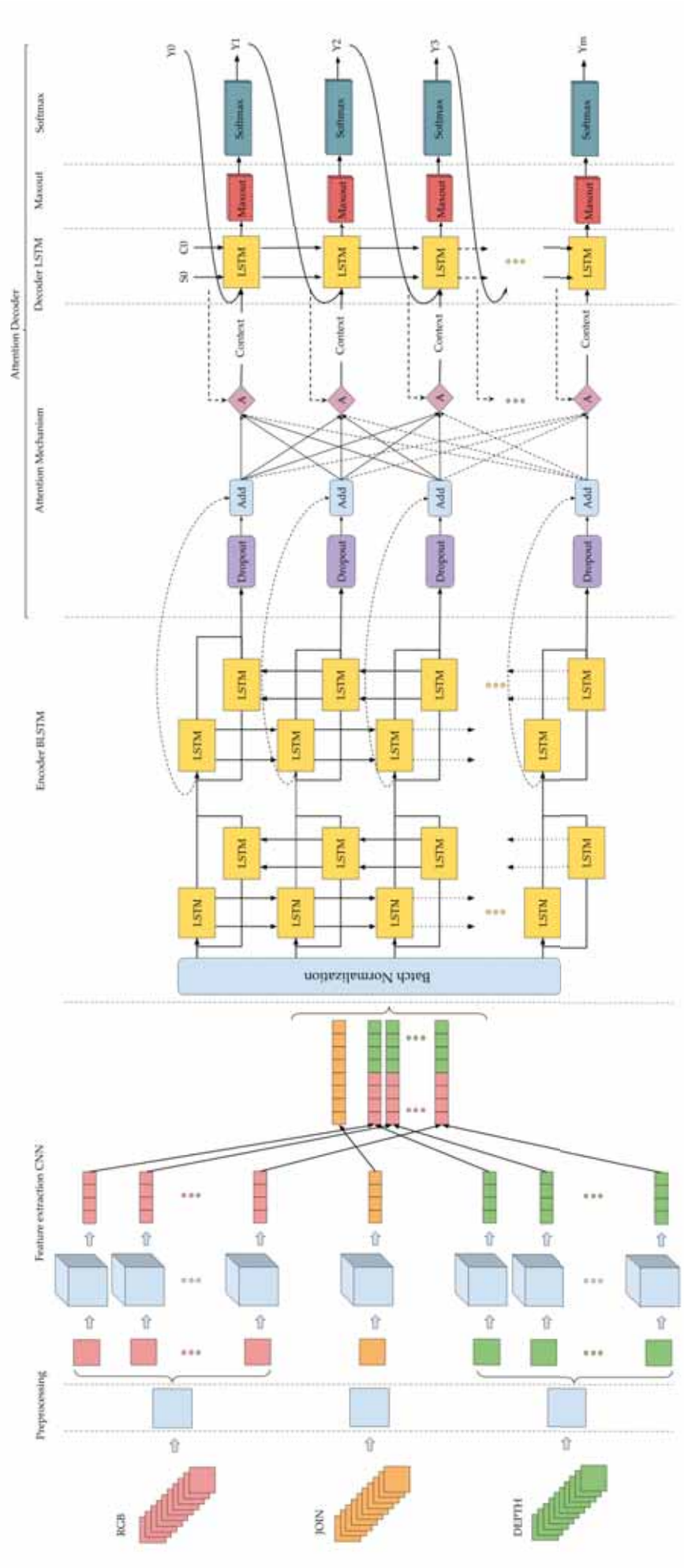


Figura 3.5: Arquitectura del proyecto.
Fuente: Propia

3.4. Preprocessing

En esta etapa se realiza el preprocesamiento para las imágenes RGB, *depth* y coordenadas de esqueleto, posteriormente se usaran como entrada del extractor de características.

3.4.1. RGB

- Sea $M = (a_{ijk})_{480\times640\times3}, \forall\, a_{ijk} \in \mathbb{R}^3$ donde M una matriz que representa un frame de video RGB, k es el numero de canales R, G y B, se realizo un escalamiento de cada pixel (a_{ij}) y $k = 1, 2, 3$ a un rango de valores entre [0,1] mediante la Ecuación 3.1.

$$M'_k = \frac{M_k - Min(M_k)}{Max(M_k) - Min(M_k)} \tag{3.1}$$

  Donde $M_k = (a_{ij})_{480\times640}$ es una matriz que representa un canal de una imagen, M'_k es de la misma dimensión que M_k con valores entre [0,1] ,Min es el valor mínimo de la matriz y Max es el valor máximo.

- Se sustrae 0.5 a cada valor a_{ij} de M'_k para $k = 1, 2, 3$ y finalmente se multiplica cada a_{ij} por 2, para tener una escala de valores entre [-1,1].

- Se redujo el tamaño de cada frame de $480 \times 640 \times 3$ a $244 \times 244 \times 3$ mediante interpolación de área de OpenCV.

3.4.2. Depth

El preprocesamiento de los datos *depth* se realizo siguiendo los siguientes pasos:

- Sea $D = (a_{ij})_{480\times640}, \forall\, a_{ij} \in \mathbb{R}^2$ una matriz que representa un frame de profundidad, se realizo una transformación de valores de cada posición (a_{ij}), donde para cada posición (a_{ij}) el sensor devuelve un κ, $\forall\kappa \in \mathbb{N}$, κ es transformado en un numero binario de 16 bits de tal manera los 3 bits menos significativos representa el actor y mediante un desplazamiento de 3 bits a la derecha se determina la distancia en milímetros figura 3.6.

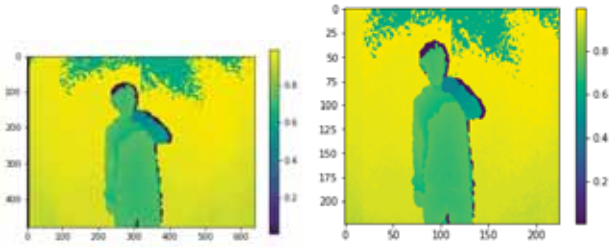
Figura 3.6: Procesamiento de datos Depth.
Fuente: Propia

- Se realizo un escalamiento de las distancias en milímetros de cada posición (a_{ij}) a una escala de [0,1] mediante (3.1).

- Luego se uso el mapeo de colores *viridis*[3] para mapear la información de profundidad de cada posición (a_{ij}) a un rango de colores, donde morado indica cerca y amarillo lejos, seguidamente después de aplicar el mapeo de colores los datos son convertidos en *RGB-A*[4], En nuestro caso usaremos únicamente los canales *RGB*.

- Se realiza un corte a cada frame manteniendo el 87.5 % de la región interna de la imagen el cual elimina el lado derecho e izquierdo de la imagen de $480X640X3$ obteniendo un tamaño de $480 \times 560 \times 3$.

- Finalmente se redujo el tamaño de la imagen a $244 \times 244 \times 3$ mediante interpolación de área, el cual sera aceptado por el extractor de características.



(a) Escalado y sin recorte (b) Recortado y reducido

Figura 3.7: Depth Frame.
Fuente: Propia

3.4.3. Coordenadas de esqueleto

Para el esqueleto se considero 10 puntos de unión (x, y, z) de un total de 20 uniones, las cuales son *hand right, wrist right, elbow right, shoulder right, shoulder center, head, shoulder left, elbow left, wrist left, hand left.* Se considero únicamente estos 10 puntos, puesto que únicamente se consideran gestos que únicamente involucran movimientos con la parte superior del cuerpo humano desde la cintura, el procesamiento se realizo de la siguiente forma.

$$M\left(P\left(f\right)\right) = \begin{pmatrix} (x_1(1), y_1(1), z_1(1)) & \cdots & (x_1(F), y_1(F), z_1(F)) \\ \vdots & \ddots & \vdots \\ (x_n(1), y_n(1), z_n(1)) & \cdots & (x_n(F), y_n(F), z_n(F)) \end{pmatrix}$$

- Se hizo el mapeo de coordenadas (x, y, z) en componentes *RGB* (Laraba et al., 2017) donde $M\left(P\left(f\right)\right)$ es una secuencia de frames de esqueleto, f es el índice de los frames, $P\left(f\right) = p_1\left(f\right), \cdots, p_n\left(f\right)$ son coordenadas de los puntos de union del esqueleto, (x, y, z), $\forall\left(x, y, z\right) \in \mathbb{R}^3$, donde n es el numero de coordenadas en cada frame, F es el numero de frames.

[3]viridis: Este mapa de colores está diseñado de tal manera que sea perfectamente uniforme perceptualmente. `https://www.youtube.com/watch?v=xAoljeRJ3lU`

[4]RGB-A : Indica los canales red, green, blues e intensidad de color

Primeramente $M\left(P\left(f\right)\right)$ se divide en sus componentes independientes representados por las siguientes matrices:

$$X = \begin{pmatrix} x_1\left(1\right) & \cdots & x_1\left(F\right) \\ \vdots & \ddots & \vdots \\ x_n\left(1\right) & \cdots & x_n\left(F\right) \end{pmatrix}$$

$$Y = \begin{pmatrix} y_1\left(1\right) & \cdots & y_1\left(F\right) \\ \vdots & \ddots & \vdots \\ y_n\left(1\right) & \cdots & y_n\left(F\right) \end{pmatrix}$$

$$Z = \begin{pmatrix} z_1\left(1\right) & \cdots & z_1\left(F\right) \\ \vdots & \ddots & \vdots \\ z_n\left(1\right) & \cdots & z_n\left(F\right) \end{pmatrix}$$

Luego se aplica un escalamiento a un rango de valores entre $[0, 1]$ para X, Y y Z mediante la ecuación (3.1), las matrices se apilan mediante la siguiente formula $S_{10\times F\times 3} = [X; Y; Z]$, donde F es el numero de frames que contiene cada expresión en lengua de señas. Se realiza este proceso para obtener una imagen con 3 canales similar a la composición de colores RGB.

- La imagen $S_{10\times F\times 3}$ se procede a redimensionar puesto que el numero de frames es variable y la imagen es muy pequeña, achatado en el primer eje. Sea $\alpha = \text{máx}(F)$ donde α es el numero de frames mas grande; se define $\kappa > 3\times\alpha \wedge \kappa\, mod\, 10 = 0$ donde $\kappa \in \mathbb{N}$; sea $I_r = \frac{\kappa}{10}$, $I_c = \frac{\kappa}{F}$ donde $I_r, I_c \in \mathbb{N}$; entonces cada fila de RGB se repite I_r veces y cada columna se repite I_c veces, finalmente se tiene una matriz $M = (a_{ijk})_{(I_r\times 10)\times(I_c\times F)\times(3)}$, figura 3.8.

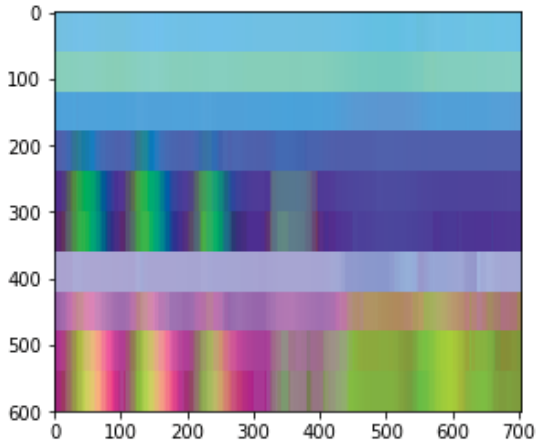


Figura 3.8: Secuencia de esqueleto representado en su respectivo RGB.
Fuente: Propia

- Se realizo el anterior paso para conseguir una imagen de dimension mayor que 600 ya que en (Laraba et al., 2017) indican que redimensionar una imagen pequeña a una mas grande causa mayor perdida que el proceso opuesto, finalmente se redimesiona mediante interpolación de área a un tamaño de $244 \times 244 \times 3$.

3.5. Feature extraction CNN

Para la etapa de extracción de características se hace uso de NN(neural network), según el estado de arte se ha determinado que las NN son mucho mas poderosas que otras técnicas (*Histograma Orientation Gradients, Hidden Markov Model,Support Vector Machine, Decision Trees* y entre otros), se hace uso de la red de convolución RestNet50 (He et al., 2016) que fue entrenada en la base de datos *ImageNet Dataset* que consta de 1.2 millones de imágenes con 1000 clases de objetos y que alcanza una precisión en la capa top-5 de 0.929, para el desarrollo del extractor de características se ha seguido los siguientes pasos.
Primeramente se realizo el entrenamiento de la CNN para los datos *depth* y *skeleton* debido a que estos datos contienen diferente distribución de información respecto a los datos con los cuales fue entrenada la red, este proceso se explica en la sección 4.3.1.

Al finalizar el entrenamiento se procede a realizar la extracción de características para cada secuencia de vídeo RGB, *skeleton* y *depth*, ver figura 3.9, para el cual es necesario eliminar las capas añadidas en el entrenamiento, en los 3 modelos se toma la ultima capa de averange pooling layer de $7 \times 7 \times 2048$ y se le aplica *max pooling* de 7×7 el cual retorna un vector de 1×2048 considerado como vector de características. Ver figura 4.1(b), al realizar este proceso se consigue que la *CNN* al procesar un determinado frame de $244 \times 244 \times 3$ de la etapa de preprocesamiento retorne un vector de características de (1×2048) el cual contiene las características mas relevantes de una imagen, este proceso se realiza para cada frame de un vídeo en los 3 tipos de datos.

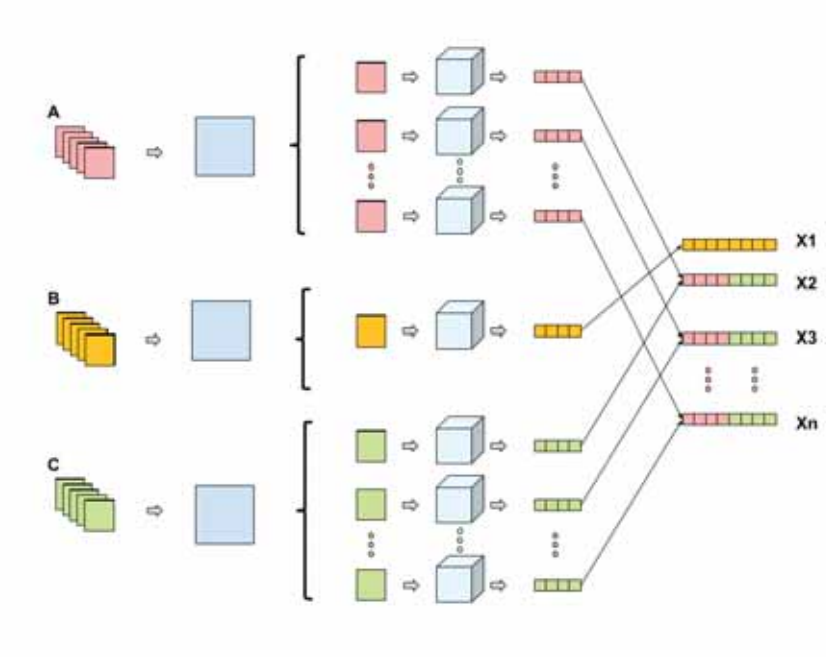


Figura 3.9: Extracción de características.
Fuente: Propia

Sea $P_i = \nu(rgb_i)$, $Q_i = \rho(depth_i)$ y $R_1 = \delta(skeleton_1)$ donde ν, ρ y δ son ResNet50, depth-ResNet50 y skeleton-ResNet50 respectively, P_i y Q_i es el i-esimo vector de características de un *frame* de entrada *rgb* y *depth* respectivamente ($i =$

$1, \cdots, n$), n es el numero de frames en un vídeo; R_1 es el vector de característica de una secuencia de *frames* de *skeleton*. Finalmente los vectores de características se combinan para generar un tensor[5] T que sera la entrada del *encoder*, ver salidas de la figura 3.9.

$$T = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_{n+1} \end{pmatrix}_{n+1,4096} = Concat \begin{pmatrix} R_1 & R_1 \\ P_1 & Q_1 \\ \vdots & \vdots \\ P_n & Q_n \end{pmatrix}_{n+1,4096}$$

3.6. Encoder BLSTM

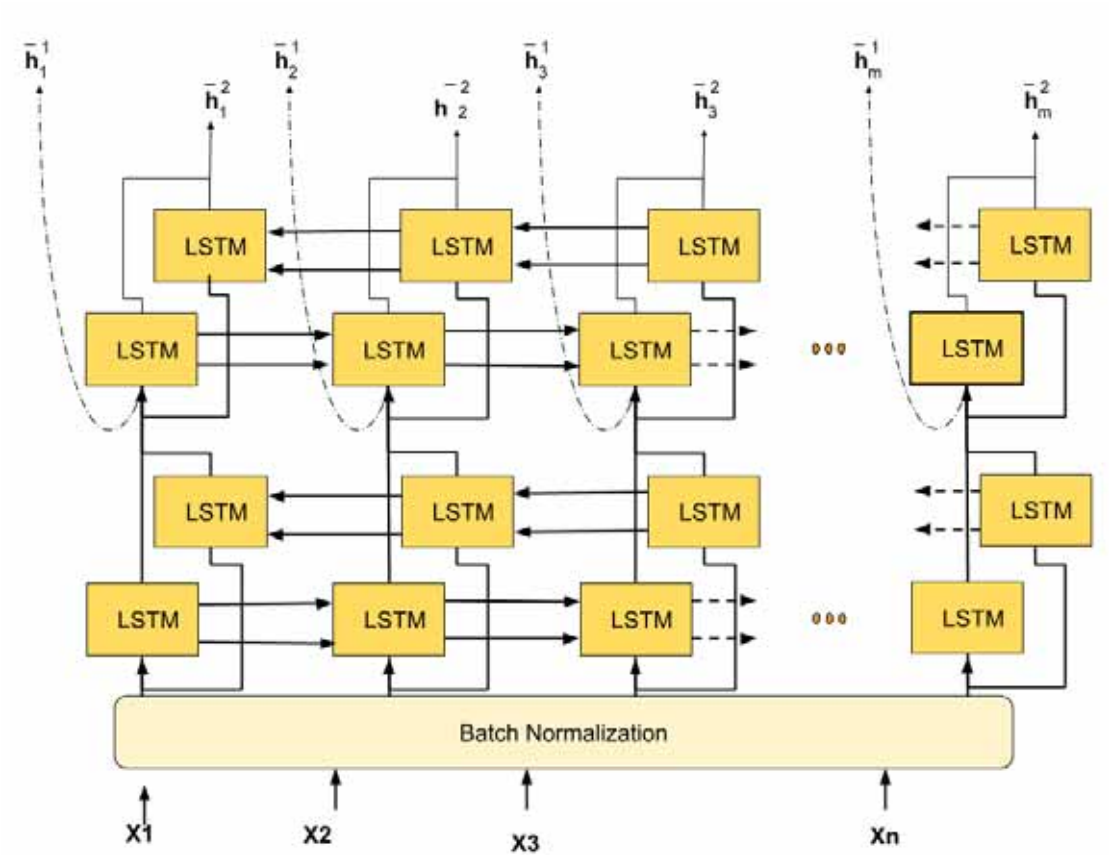


Figura 3.10: Encoder.
Fuente: Propia

El tensor T se normalizo mediante *Batch Normalitation* y seguidamente es codificado mediante un *encoder BLSTM* figura 3.10, se desarrollo un *encoder* con el propósito codificar la información de entrada y reducir su dimensión, el *encoder* se desarrollo apilando un nivel(capa) de *BLSTM* con un nivel de *recurrent BLSTM* con el fin de mejorar el rendimiento, mantener la información a mas niveles de

[5]Tensor: Un tensor es cierta clase de entidad algebraica de varios componentes, que generaliza los conceptos de escalar, vector y matriz de una manera que sea independiente de cualquier sistema de coordenadas elegido.

profundidad y evitar el problema de *vanish gradient* en la *RNN*(recurrent neural network), se usa *BLSTM* ya que cada secuencia de gestos no solo depende de los movimientos anteriores sino también de los movimientos que están por venir, cada *BLSTM* esta formado por el apilamiento de dos niveles de *LSTM* con un espacio dimensional de salida de 500 unidades para el *hidden state* y *cell state*, donde la primera pila de *LSTM* recorre el vídeo de $X_1, X_2, \cdots, X_n$ para producir un *hidden state* $\overrightarrow{h}^{<t>}$ y la segunda pila recorre de $X_n, X_{n-1}, \cdots, X_1$ para producir un *hidden state* $\overleftarrow{h}^{<t>}$ donde cada recorrido es llamado *forward* y *backward* respectivamente, el *hidden state* final de la *BLSTM* esta dada por la concatenación de los *hidden state* de ambos recorridos mediante la ecuación 2.39. Con el propósito de evitar el *overfitting* debido a la profundidad del *encoder* se aplico *dropout regularization* para cada $X_1, X_2, \cdots, X_n$ con una tasa de abandono de 0.5 y *recurrent dropout regularization* para los estados recurrentes con una tasa de abandono de 0.5 en cada unidad de *LSTM* que conforman una unidad de *BLSTM*, ver figura 2.29(b), al aplicar este tipo de regularización se garantiza que la *LSTM* tenga mejor desempeño y no conduzca a *overfitting* que al aplicar únicamente *dropout* a las partes *feedforward* de la *LSTM* como indica en (Gal and Ghahramani, 2016).

3.7. Attention Decoder

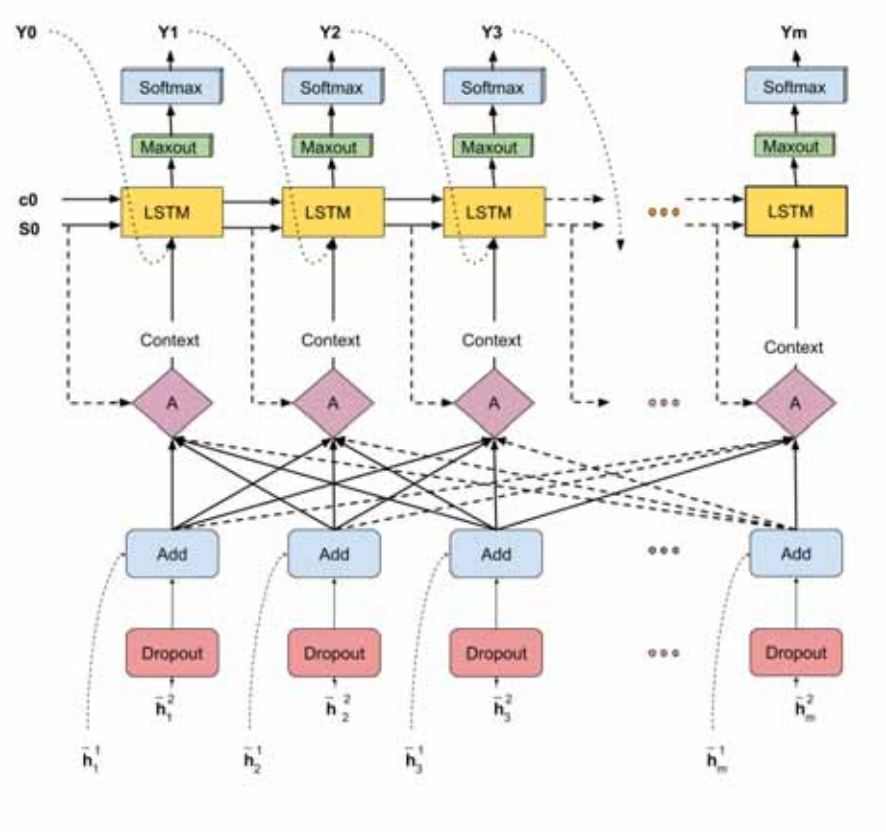


Figura 3.11: Decorder LSTM.
Fuente: Propia

A las salidas del *encoder BLSTM* se le aplica *dropout* con una tasa de abandono de 0.5, seguidamente se realiza la operación residual mediante la ecuación (2.37), la salida residual es alimentada a un *attention model* cada tiempo T_y, donde T_y viene a ser la longitud de la secuencia de salida. El modelo de atención esta dada por:

$$c_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j \tag{3.2}$$

Donde c_i es el context, T_x es la longitud de la secuencia de entrada, α_{ij} es el peso calculado en cada paso de tiempo i con h_j, h_j es la salida residual del *encoder BLSTM*.

$$\alpha_{ij} = \frac{exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} exp\left(e_{ik}\right)} \tag{3.3}$$

$$e_{ij} = \sigma\left(s_{i-1}, h_j\right)$$

En la ecuacion 3.3 σ es la función de activación *ReLu* de 64 neuronas que forman el *attention model*, cada contexto c_i es ingresado al *decoder LSTM* para producir el *hidden state* s_i, el *decoder LSTM* se desarrollo con el propósito de aprender información temporal e información espacial de la secuencia de vídeos, para el desarrollo del *decoder LSTM* se uso una capa de *LSTM* con un espacio dimensional de salida de 900 unidades, se uso *LSTM* con el objetivo de evitar el problema de *vanish gradient*. El encoder esta dado por las siguientes ecuaciones:

$$\begin{gathered} i_t = \Theta\left(c_i W_i + s_{i-1} U_i + \hat{y}_{i-1} V_i + b_i\right) \\ f_t = \Theta\left(c_i W_f + s_{i-1} U_f + \hat{y}_{i-1} V_f + b_f\right) \\ o_t = \Theta\left(c_i W_o + s_{i-1} U_o + \hat{y}_{i-1} V_o + b_o\right) \\ \hat{c}_t = tanh\left(c_i W_c + s_{i-1} U_c + \hat{y}_{i-1} V_c + b_c\right) \\ c_t = f_t \odot s_{i-1} + i_t \odot \hat{c}_t \\ s_i = o_t \odot c_t \end{gathered} \tag{3.4}$$

Donde θ es la función de activación *sigmoide* además los pesos de las matrices W, V (inicializados mediante *Xavier uniform initializer*), U (inicializado mediante una matriz ortogonal aleatoria) y b (inicializado en ceros), son parámetros entrenables, seguidamente la salida del *decoder LSTM* s_i es alimentada a una capa *Maxout* con 90 unidades ocultas y cada una contiene 5 unidades lineales, la capa *Maxout* se define mediante la ecuación:

$$\begin{gathered} hm_i\left(x\right) = \underset{j \in 1,k}{max}\, z_{ij} \\ z_{ij} = x^T W_{\ldots ij} + b_{ij} \end{gathered} \tag{3.5}$$

Donde $W \in \mathbb{R}^{dxmxk}$ y $b \in \mathbb{R}^{mxk}$ son parámetros entrenables, k es el numero de neuronas lineales que conforma una unidad oculta de *maxout* hm_i.

En la figura 3.12 se puede ver la representación de una *maxout network* con $k = 3$.

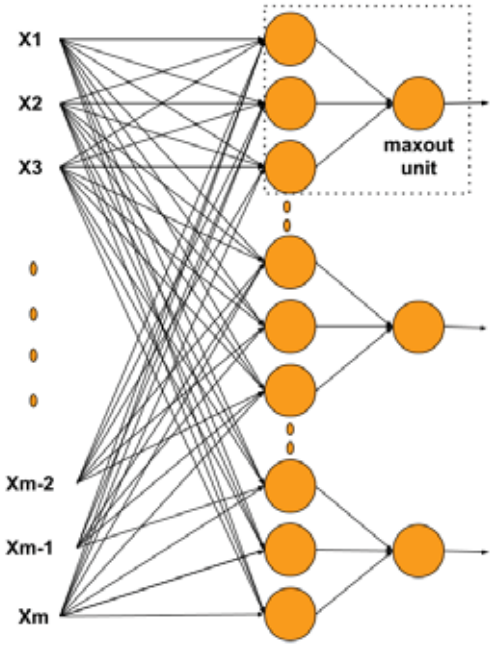


Figura 3.12: Maxout network.
Fuente: Propia

Finalmente la probabilidad de predecir una sentencia $\hat{y}_i$ esta denotada por la siguiente ecuación:

$$p(\hat{y}_i|\hat{y}_1, \cdots, \hat{y}_{i-1}, x_i) = \alpha(\rho(\hat{y}_{i-1}, s_i, c_i)) \tag{3.6}$$

Donde α es la capa *maxout* y ρ esta definido por la ecuación 3.4.

Parte IV

Proceso Experimental

Capítulo 4

Experimentacion de modelos y entrenamiento

4.1. Experimentación en la selección de parámetros y capas para la construcción de la arquitectura

El modelo propuesto se llego a desarrollar y proponer mediante el ciclo de desarrollo de una arquitectura de DNN figura 2.26, donde el desempeño de una arquitectura de *deep learning* depende tanto del numero de capas como el numero de parámetros. Para la etapa de selección de parámetros y capas se uso la base de datos VideoLSP10 de la sección 3.2, seguidamente se eligió la arquitectura que tiene mayor tasa de exactitud en los datos RGB y la combinación de los datos rgb, depth y skeleton, dicho proceso es desarrollado a continuación.

4.1.1. Experimentación en los datos RGB

(a) Se experimento con un modelo compuesto por las siguientes capas: **normalización - encoder(BLSTM) - encoder(BLSTM) - attention mechanism - decoder(LSTM) - softmax**, donde el espacio dimensional de la salida del encoder es de 200 y se aplico dropout de 0.3 a las entradas del encoder y recurrent dropout de 0.3 a los estados recurrentes, el attention mechanism esta conformado por 120 neuronas con una función de activación ReLU, el espacio dimensional de la salida del decoder es de 400. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 34 frases de un total de 100.

(b) **Encoder(BLSTM) - encoder(BLSTM) - attention mechanism - recurrent decoder(LSTM) - softmax**, donde el espacio dimensional de la salida del encoder es de 200 y se aplico dropout de 0.3 a las entradas del encoder y recurrent dropout de 0.3 a los estados recurrentes, el attention mechanism esta conformado por 120 neuronas con una función de activación ReLU, el espacio dimensional de la salida del recurrent decoder es de 400. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 67 frases de un total de 100.

(c) **Encoder(BLSTM) - encoder(BLSTM) - dropout - attention mechanism - recurrent decoder(LSTM) - maxout - dropout - softmax**, donde el espacio dimensional de la salida del encoder es de 200 y se aplico dropout de 0.3 a las entradas del encoder y recurrent dropout de 0.3 a los estados recurrentes, el dropout es de 0.3, el attention mechanism esta conformado por 120 neuronas con una función de activación tanh, el espacio dimensional de la salida del recurrent decoder es de 400, la capa maxout tiene 50 unidades ocultas de *maxout* cada una conformada por 4 unidades lineales, finalmente un dropout de 0.3. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 49 frases de un total de 100.

(d) **Encoder(BLSTM) - encoder(BLSTM) - dropout - attention mechanism - recurrent decoder(LSTM) - maxout - dropout - softmax**, donde el espacio dimensional de la salida del encoder es de 500 y se aplico dropout de 0.5 a las entradas del encoder y recurrent dropout de 0.5 a los estados recurrentes, el dropout es de 0.5, el attention mechanism esta conformado por 120 neuronas con una función de activación ReLU, el espacio dimensional de la salida del recurrent decoder es de 900, la capa maxout tiene 90 unidades ocultas de *maxout* cada una conformada por 5 unidades lineales, finalmente un dropout de 0.3. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 73 frases de un total de 100.

(e) **Normalización - encoder(BLSTM) - recurrent encoder(BLSTM) - dropout - attention mechanism - recurrent decoder(LSTM) - maxout - dropout - softmax**, donde el espacio dimensional de la salida del encoder(BLSTM) y recurrent encoder(BLSTM) es de 500 y se aplico dropout de 0.5 a las entradas del encoder y recurrent dropout de 0.5 a los estados recurrentes, el dropout es de 0.5, el attention mechanism esta conformado por 64 neuronas con una función de activación ReLU, el espacio dimensional de la salida del decoder es de 900, la capa maxout tiene 90 unidades ocultas de *maxout* cada una conformada por 5 unidades lineales, finalmente un dropout de 0.3. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 83 frases de un total de 100.

4.1.2. Experimentación con la combinación de datos

(a) **Normalización - encoder(BLSTM) - encoder(BLSTM) - attention mechanism - decoder(LSTM) - softmax**, donde el espacio dimensional de la salida del encoder es de 200 y se aplico dropout de 0.3 a las entradas del encoder y recurrent dropout de 0.3 a los estados recurrentes, el attention mechanism esta conformado por 120 neuronas con una función de activación ReLU, el espacio dimensional de la salida del decoder es de 400. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 99 frases de un total de 100.

(b) **Normalización - encoder(BLSTM) - attention mechanism - decoder(LSTM) - softmax**, donde el espacio dimensional de la salida del en-

coder es de 200 y se aplico dropout de 0.3 a las entradas del encoder y recurrent dropout de 0.3 a los estados recurrentes, el attention mechanism esta conformado por 120 neuronas con una función de activación ReLU, el espacio dimensional de la salida del decoder es de 400. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 97 frases de un total de 100.

(c) **Normalización - encoder(BLSTM) - encoder(BLSTM) - attention mechanism - decoder(LSTM) - softmax**, donde el espacio dimensional de la salida del encoder es de 500 y se aplico dropout de 0.3 a las entradas del encoder y recurrent dropout de 0.3 a los estados recurrentes, el attention mechanism esta conformado por 120 neuronas con una función de activación ReLU, el espacio dimensional de la salida del decoder es de 900. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 98 frases de un total de 100.

(d) **Normalización - encoder(BLSTM) - encoder(BLSTM) - attention mechanism - decoder(LSTM) - softmax**, donde el espacio dimensional de la salida del encoder es de 200 y se aplico dropout de 0.3 a las entradas del encoder y recurrent dropout de 0.3 a los estados recurrentes, el attention mechanism esta conformado por 64 neuronas con una función de activación ReLU, el espacio dimensional de la salida del decoder es de 400. La arquitectura se entreno usando Adam con una tasa de aprendizaje de 0.001 y el modelo predijo 96 frases de un total de 100.

(e) **Encoder(BLSTM) - encoder(BLSTM) - dropout - attention mechanism - recurrent decoder(LSTM) - maxout - dropout - softmax**, donde el espacio dimensional de la salida del encoder es de 500 y se aplico dropout de 0.5 a las entradas del encoder y recurrent dropout de 0.5 a los estados recurrentes, el dropout es de 0.5, el attention mechanism esta conformado por 120 neuronas con una función de activación ReLU, el espacio dimensional de la salida del recurrent decoder es de 900, la capa maxout tiene 90 unidades ocultas de *maxout* cada una conformada por 5 unidades lineales, finalmente un dropout de 0.3. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 95 frases de un total de 100.

(f) **Normalización - encoder(BLSTM) - residual encoder(BLSTM) - dropout - attention mechanism - decoder(LSTM) - maxout - dropout - softmax**, donde el espacio dimensional de la salida del encoder y residual encoder es de 500 y se aplico dropout de 0.5 a las entradas del encoder y recurrent dropout de 0.5 a los estados recurrentes, el dropout es de 0.5, el attention mechanism esta conformado por 64 neuronas con una función de activación ReLU, el espacio dimensional de la salida del decoder es de 900, la capa maxout tiene 90 unidades ocultas de *maxout* cada una conformada por 5 unidades lineales, finalmente un dropout de 0.3. La arquitectura se entreno usando RMSprop con una tasa de aprendizaje de 0.045 y el modelo predijo 99 frases de un total de 100.

Modelos	N	BLSTM 1			BLSTM 2			Rrecurrent BLSTM 1			D	Attention mechanism		Decoder LSTM		Recurrent decoder LSTM		Maxout			Training		# Frases inferidas de 100
		U	D	RD	U	D	RD	U	D	RD		U	FA	U	D	U	D	UL	UM	D	OP	Learning rate	
RGB	Experimentación, selección de parámetros y capas utilizando únicamente como entrada imágenes *rgb*																						
a	si	200	0.3	0.3	200	0.3	0.3	-	-	-	-	120	ReLu	400	0	-	-	-	-	-	RMSprop	0.045	34
b	-	200	0.3	0.3	200	0.3	0.3	-	-	-	-	120	ReLu	-	-	400	0	-	-	-	RMSprop	0.045	67
c	-	200	0.3	0.3	200	0.3	0.3	-	-	-	0.3	120	Tanh	-	-	400	0	4	50	0.3	RMSprop	0.045	49
d	-	500	0.5	0.5	500	0.5	0.5	-	-	-	0.5	120	ReLu	-	-	900	0	5	90	0.3	RMSprop	0.045	73
e	**si**	**500**	**0.5**	**0.5**	**-**	**-**	**-**	**500**	**0.5**	**0.5**	**0.5**	**64**	**ReLu**	**900**	**0**	**-**	**-**	**5**	**90**	**0.3**	**RMSprop**	**0.045**	**83**
COMBINACIÓN DE DATOS	Experimentación, selección de parámetros y capas usando como entrada imágenes *rgb*, datos *depth* y datos *skeleton*																						
a	si	200	0.3	0.3	200	0.3	0.3	-	-	-	-	120	ReLu	400	0	-	-	-	-	-	RMSprop	0.045	99
b	si	200	0.3	0.3	-	-	-	-	-	-	-	120	ReLu	400	0	-	-	-	-	-	RMSprop	0.045	97
c	si	500	0.3	0.3	500	0.3	0.3	-	-	-	-	120	ReLu	900	0	-	-	-	-	-	RMSprop	0.045	98
d	si	200	0.3	0.3	200	0.3	0.3	-	-	-	-	64	ReLu	400	0	-	-	-	-	-	Adam	0.001	96
e	si	500	0.5	0.5	500	0.5	0.5	-	-	-	0.5	120	ReLu	-	-	900	0	5	90	0.3	RMSprop	0.045	95
f	**si**	**500**	**0.5**	**0.5**	**-**	**-**	**-**	**500**	**0.5**	**0.5**	**0.5**	**64**	**ReLu**	**900**	**0**	**-**	**-**	**5**	**90**	**0.3**	**RMSprop**	**0.045**	**99**

Tabla 4.1: Evaluación de arquitecturas y elección de parámetros

La tabla 4.1 muestra el resumen de parámetros y capas usadas en la experimentación antes de llegar a proponer una arquitectura bastante robusta que pueda tener una buena exactitud tanto únicamente en datos RGB, *depth* y como también en la combinación de estos, en la tabla se puede apreciar que el primer modelo en la columna "COMBINACIÓN DE DATOS" de la tabla 4.1 llego a predecir correctamente 99 frases al igual que el sexto modelo, hasta este punto se debería proponer el primer modelo, puesto que tiene la misma exactitud, menor numero de capas y menor numero de parámetros, por otro lado este mismo modelo al ser evaluado únicamente con datos RGB como se aprecia en la columna "RGB" de la tabla 4.1 el cual esta ubicada en la primera posición, se observa que su exactitud recae dramáticamente puesto que únicamente infirió 34 frases, por otro lado, el sexto modelo de la sección "COMBINACIÓN DE DATOS" tiene su respectiva configuración en la quinta posición de la columna "RGB" y se observa que ha inferido correctamente 83 frases, por ende se propone dicho modelo ya que ofrece la mejor tasa de exactitud en ambos tipos de datos.

4.2. Términos de clasificación de datos

4.2.1. Clasificación

Los componentes básicos de las métricas que se usan para evaluar los modelos en este trabajo son cuatro. Un verdadero positivo (VP) es un resultado en el que el modelo predice correctamente la clase positiva. De manera similar, un verdadero negativo(VN) es un resultado en el que el modelo predice correctamente la clase negativa. Un falso positivo (FP) es un resultado en el que el modelo predice incorrectamente la clase positiva. Y un falso negativo (FN) es un resultado en el que el modelo predice incorrectamente la clase negativa (GoogleDevelopers, 2018). La precisión es la capacidad de un modelo de clasificación para devolver sólo las instancias relevantes. ¿Qué proporción de identificaciones positivas fue correcta?

$$Precision = (VP)/(VP + FP) \tag{4.1}$$

Recall es la capacidad de un modelo de clasificación para identificar todas las instancias relevantes.¿Qué proporción de positivos reales se identificó correctamente?.

$$Recall = (VP)/(VP + FN) \tag{4.2}$$

F1 *score* es la única métrica que combina memoria y precisión utilizando la media armónica[1] (Koehrsen, 2018).

$$F_1 = 2 * (Precision * Recall)/(Precision + Recall) \tag{4.3}$$

4.3. Entrenamiento de la arquitectura

El entrenamiento de nuestra arquitectura propuesta en la figura 3.5 se realizo mediante un proceso iterativo, el entrenamiento se divide en dos partes donde primeramente se realizo el entrenamiento de la *CNN* y finalmente ya habiendo entrenado se procedió a entrenar la *RNN*.

[1]Se utiliza por ejemplo en algunos casos donde es necesario promediar variaciones con respecto al tiempo (González and Felpeto, 2006).

4.3.1. Entrenamiento de la *CNN*

Para el entrenamiento de la *CNN* se considero entrenar dos arquitecturas *ResNet50* para los datos *depth* y coordenadas de esqueleto, a cada arquitectura entrenada se considero nombrar como *Depth-ResNet50* y *Skeleton-ResNet50*, en este documento de investigación la arquitectura primigenia *ResNet50* es considerado como *RGB-ResNet50*. Para el *training* del *Depth-ResNet50* se hace uso del *dataset* de la tabla 3.1 y para el *training* del *Skeleton-ResNet50* se hace uso del *dataset* de la tabla 3.2. Los datos se dividen de la siguiente manera: para el *Depth-Resnet50* 96 % es para el *training*, el 4 % para la *validation* y para el *Skeleton-ResNet50* el 90 % es para el *training* y 10 % para la *validation*. El *training* se realizo aplicando *transfer learning* para ambos casos, para dicho propósito se ha eliminado la ultima capa FC que consta de 1000 neuronas.

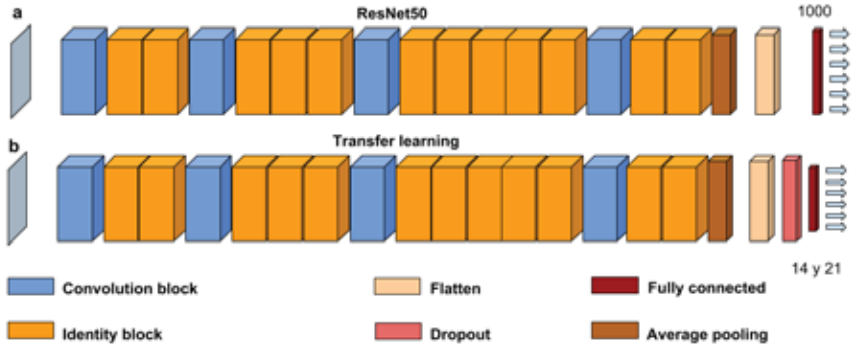


Figura 4.1: Transfer learning ResNet50.
Fuente: Propia

Posteriormente se añadió la regularización *dropout* con una tasa de abandono de 0.2 y la capa FC con una función de activación *softmax* que contiene 14 y 21 neuronas para el *Depth-ResNet50* y *Skeleton-ResNet50* respectivamente como se aprecia en la figura 4.1, donde cada peso de la capa FC agregada es inicializada mediante *Xavier uniform initializer*. Para el proceso de *training* se uso el algoritmo de *backpropagation* y se acelero usando la optimización *SGD* con una *learning rate* $\alpha = 0{,}001$ y *momentum* $\beta = 0{,}9$, finalmente se empezó a reentrenar toda la red. Es necesario aclarar que para los datos *rgb* no se ha realizo ningún tipo de entrenamiento.

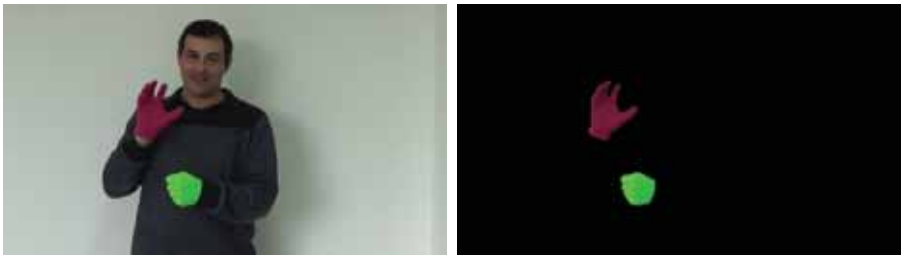
4.3.2. Entrenamiento de la *RNN*

El entrenamiento de la *RNN* se hizo utilizando el *Dataset VideoLSP10_Total* de la tabla 3.3 y se distribuye como se indica en la subsección 3.2.4. Se realizo la extracción de características de los datos *rgb, depth, skeleton* del *dataset LSP* mediante las redes de convolución *RGB-ResNet50*, *Depth-ResNet50* y *Skeleton-ResNet50* respectivamente. las características extraídas se guardo en un archivo con extensión *h5* llamado *features.h5*, se hizo este proceso puesto que no se cuenta con un poder de computo suficiente para poder realizar un entrenamiento *End to End*.

Seguidamente las características extraídas en el archivo *features.h5* es ingresado al *Encoder BLSTM* de la sección 3.6 y finalmente al *Attention decoder* de la sección 3.11. El entrenamiento se realizo aplicando el algoritmo de *backpropagation* conjuntamente con el algoritmo *RMSProp* cuyo hiperparametros se configuran de la siguiente manera *learning rate* $\alpha = 0{,}045$, $\rho = 0{,}94$, $\epsilon = 1$ y *learning decay* $= 0{,}0$, el entrenamiento se realizo en *minibach* de 4 vídeos, además se aplico una técnica de monitorización de datos llamado *Early stopping* el cual nos indica cuando el modelo esta comenzando a realiza *overfitting* en nuestros datos de entrenamiento de acuerdo a una variable monitorizada,la variable monitorizada es *val-loss* y con un hiperparametro *patience* $= 5$, el cual dentendra el entrenamiento cuando la variable *val-loss* no disminuya durante 5 *epoch*[2] y la función de perdida utilizada es *categorical crossentropy*, finalmente se realizo el entrenamiento de la *RNN*.

4.4. Modelos experimentales

Para la comparación de nuestra arquitectura se ha optado dos modelos experimentales que varían únicamente en la etapa de preprocesamiento, el modelo experimental se desarrollo en base a nuestro modelo propuesto, únicamente se han eliminado el *Depth ResNet50* y *Skeleton ResNet50* como se aprecia en la figura 4.3,se realizo dicha configuración con el propósito de comparar los modelos del paper (Masood et al., 2018) el cual usa el *dataset* LSA64 (Ronchetti et al., 2016) que cuenta únicamente con imágenes RGB.
Se realizo el preprocesamiento correspondiente de las entradas para cada modelo, para el modelo experimental 1 se realiza únicamente dimensionamiento de *frames* a $244 \times 244 \times 244$, para el modelo experimental 2 se realiza eliminación de *background*, se mantiene únicamente los guantes de colores de cada *frame* y se dimensiona a $244 \times 244 \times 244$, los resultados despues de procesar las entradas se aprecia en la figura 4.2. LSA64 consta de 64 gestos de los cuales se ha considerado únicamente 32 gestos. El entrenamiento se realizo aplicando el algoritmo de *backpropagation* conjuntamente con el algoritmo *Adam optimization* cuyo hiperparametros se configuran de la siguiente manera *learning rate* $\alpha = 0{,}001$, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ y *learning decay* $= 0{,}1$.



(a) Preprocesamiento del modelo experimental 1.

(b) Preprocesamiento del modelo experimental 2.

Figura 4.2: Procesamiento de entradas para el modelo experimental.
Fuente: Propia

[2]epoch:una iteración través de todo el conjunto de entrenamiento

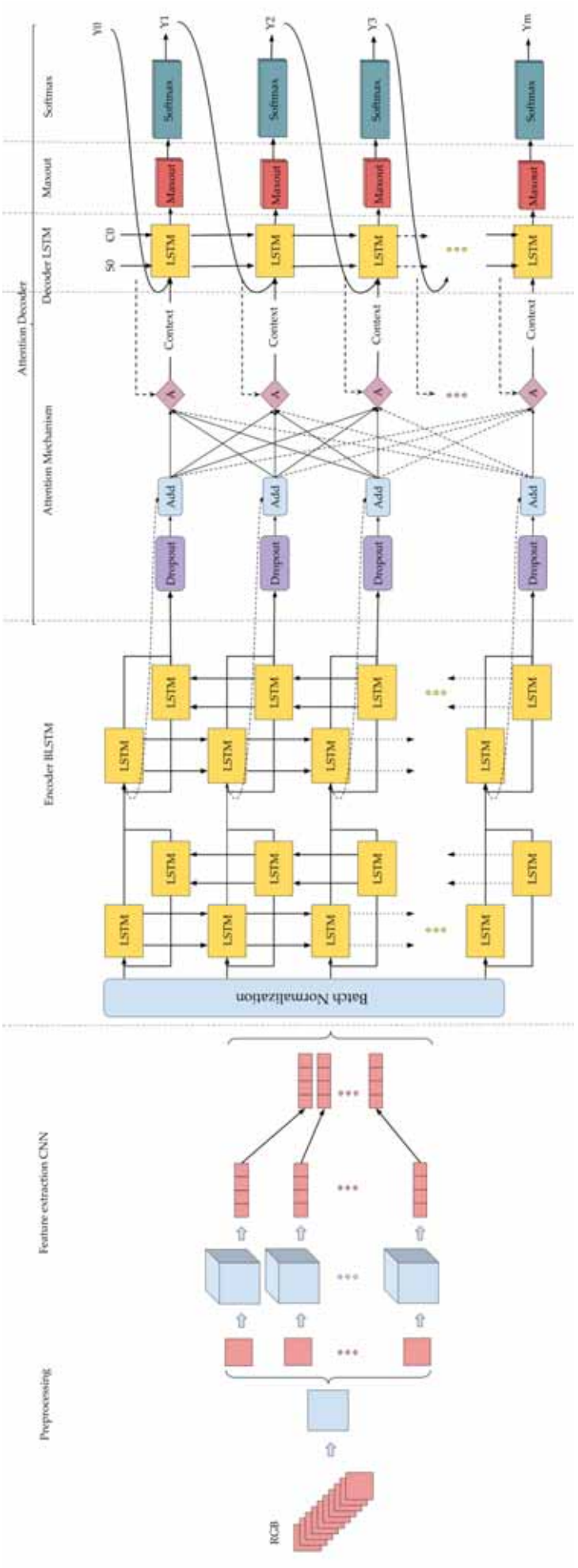


Figura 4.3: Arquitectura del modelo experimental.
Fuente: Propia

Parte V

Resultados

Capítulo 5

Resultados e Interpretaciones

A continuación, se muestra los resultados obtenidos de nuestro modelo propuesto en la base de datos VideoLSP10, asimismo el modelo propuesto se evaluara también en la base de datos LSA64 (Ronchetti et al., 2016), con el objetivo de realizar una comparación con modelos similares desarrollados en esta base de datos.

5.1. Resultados de entrenamiento de la CNN

El resultado del entrenamiento de la CNN *Depth-ResNet50* con los datos de la tabla 3.1 se muestra a continuación, donde se obtuvo una tasa de exactitud de 97.67 % en el conjunto de validación.

	Precision	Recall	F1 score	#Data
ayudame	1.0	1.0	1.0	8.0
como	1.0	1.0	1.0	11.0
cual	1.0	1.0	1.0	3.0
disculpame	1.0	1.0	1.0	8.0
donde	1.0	0.667	0.8	3.0
entiendo	1.0	1.0	1.0	12.0
estas	1.0	0.778	0.88	8.0
gracias	1.0	1.0	1.0	3.0
hola	1.0	1.0	1.0	4.0
manana	1.0	1.0	1.0	2.0
nombre	1.0	1.0	1.0	3.0
por_favor	1.0	1.0	1.0	11.0
que	1.0	1.0	1.0	5.0
tu	1.0	0.833	0.91	5.0
Avg/total	1.0	0.948	0.971	86.0

Tabla 5.1: Resultados del dataset VideoLSP10_Depth.

En la tabla 5.1 se puede apreciar los niveles de precisión, recall, F1 score y la cantidad de datos utilizada para dicha evaluación. En la columna de Precisión, se puede apreciar que "*ayudame*" tiene una precisión de 1.0, es decir, que cuando el modelo predice que es "*ayudame*" acierta el 100 % de las veces, al igual en la siguiente columna, "*ayudame*" tiene un recall de 1.0, es decir que identifica correctamente el 100 % de todas las palabras que realmente fue etiquetada como "*ayudame*", la columna "*F1score*" logra una efectividad de 1.0, es decir el modelo realiza perfectamente la inferencia para "*ayudame*" y puede generalizar su información para la predicción de datos con la misma distribución puesto que es mayor que 0.5; análogamente la fila de la palabra "*estas*" tiene una precisión de 1.0, es decir, que cuando el modelo predice que es "*estas*" acierta el 100 % de las veces,en la siguiente columna, "*estas*" tiene un recall de 0.778, es decir que identifica correctamente el 77.8 % de todas las palabras "*estas*" y la columna "*F1score*" logra 0.88, logrando una efectividad del 88.0 %.Finalmente el modelo tiene una precisión global del 100 %, un recall de 94.8 % y un F1 score de 97.1 %.

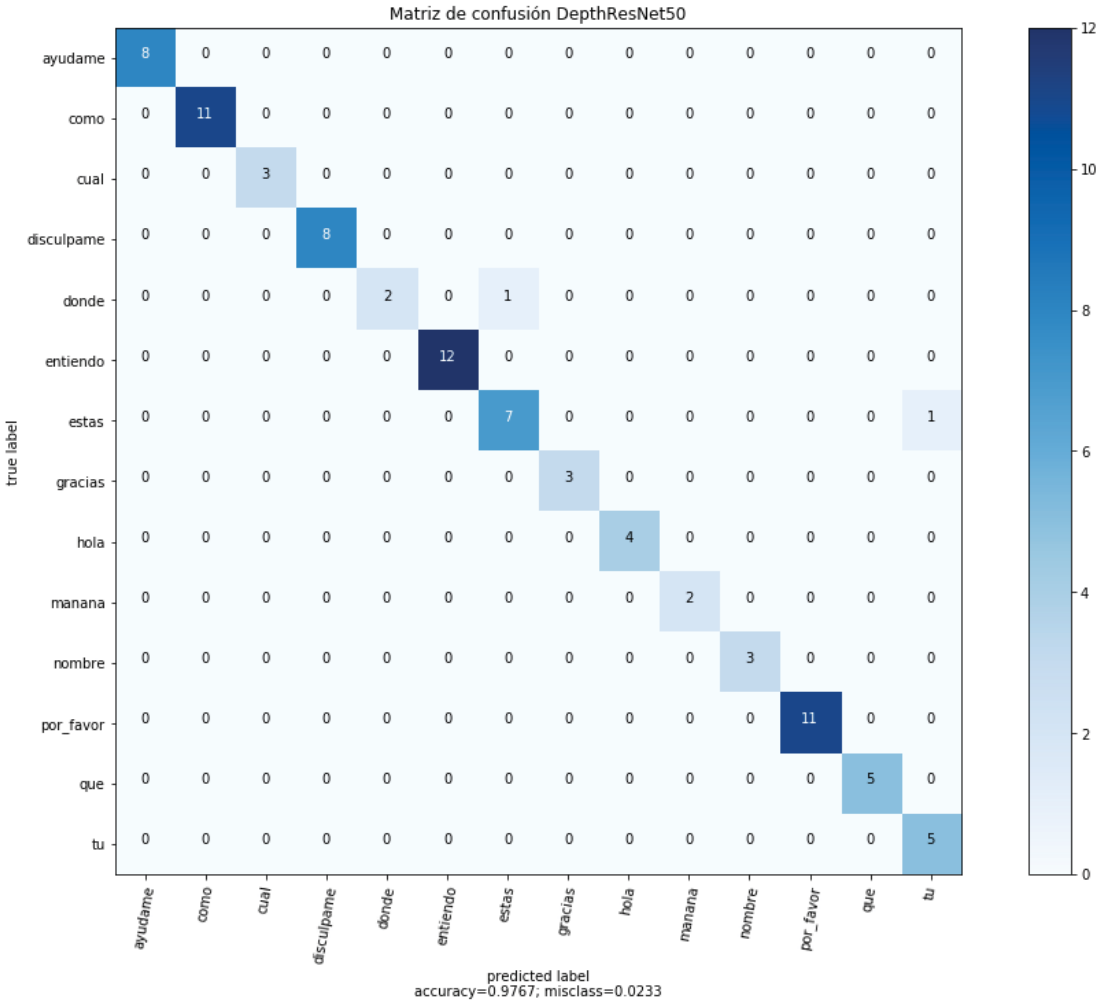
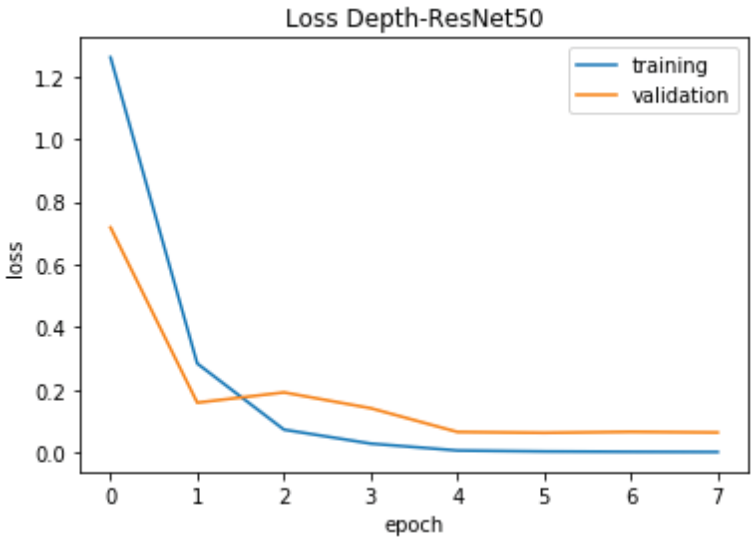
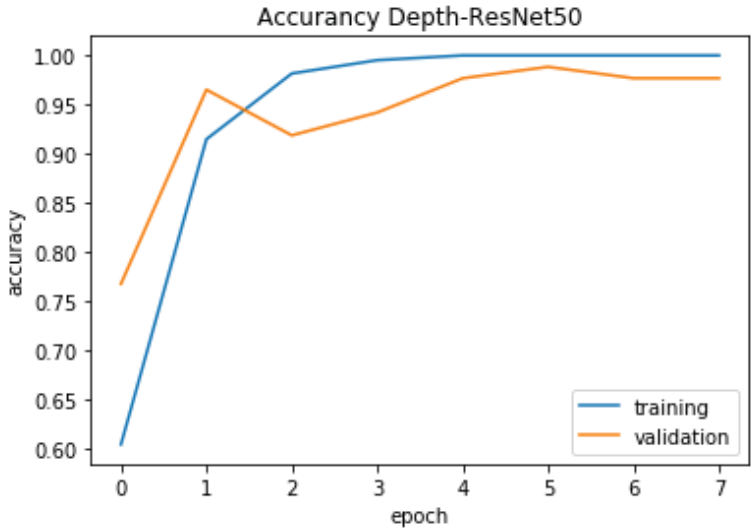


Figura 5.1: Matriz de confusión de la arquitectura Depth-ResNet50.
Fuente: Propia

Matriz de confusión 5.1 en base al vocabulario de la tabla 3.1, se muestra que logro una exactitud de 0.9767.



(a) Loss



(b) Accuracy

Figura 5.2: Train Depth-ResNet50.
Fuente: Propia

En la figura 5.2 se puede apreciar el numero de iteraciones sobre todo nuestro conjunto de entrenamiento(epoch) para poder conseguir la tasa de exactitud ya mencionada sin recurrir al *overfitting*.

El resultado del entrenamiento de la CNN *Skeleton-ResNet50* con los datos de la tabla 3.2 se muestra a continuación, donde se obtuvo una tasa de exactitud de 95.24 % en el conjunto de validación.

	Precision	Recall	F1 score	#Data
iarriba	1.0	0.375	0.55	8.0
darriba	1.0	0.75	0.86	8.0
aarriba	1.0	0.667	0.8	8.0
iadelante	1.0	1.0	1.0	8.0
dadelante	1.0	1.0	1.0	8.0
aadelante	1.0	1.0	1.0	8.0
ii	1.0	1.0	1.0	8.0
di	0.889	1.0	0.94	8.0
ai	0.875	1.0	0.93	8.0
id	1.0	1.0	1.0	8.0
dd	1.0	0.889	0.94	8.0
ad	1.0	1.0	1.0	8.0
icabeza	1.0	0.889	0.94	8.0
dcabeza	1.0	0.889	0.94	8.0
acabeza	1.0	1.0	1.0	8.0
iboca	1.0	1.0	1.0	8.0
dboca	1.0	1.0	1.0	8.0
aboca	1.0	1.0	1.0	8.0
icentro	1.0	1.0	1.0	8.0
dcentro	1.0	1.0	1.0	8.0
acentro	1.0	1.0	1.0	8.0
Avg/total	0.989	0.927	0.948	168.0

Tabla 5.2: Resultados del dataset VideoLSP10_Join.

Finalmente el modelo tiene una precisión global[1] del 98.9 %, un recall [2] de 92.7 % y un F1 score [3] de 94.8 %.

[1] ¿Qué proporción de identificaciones positivas fue correcta?
[2] ¿Qué proporción de positivos reales se identificó correctamente?
[3] Métrica que combina memoria y precisión utilizando la media armónica

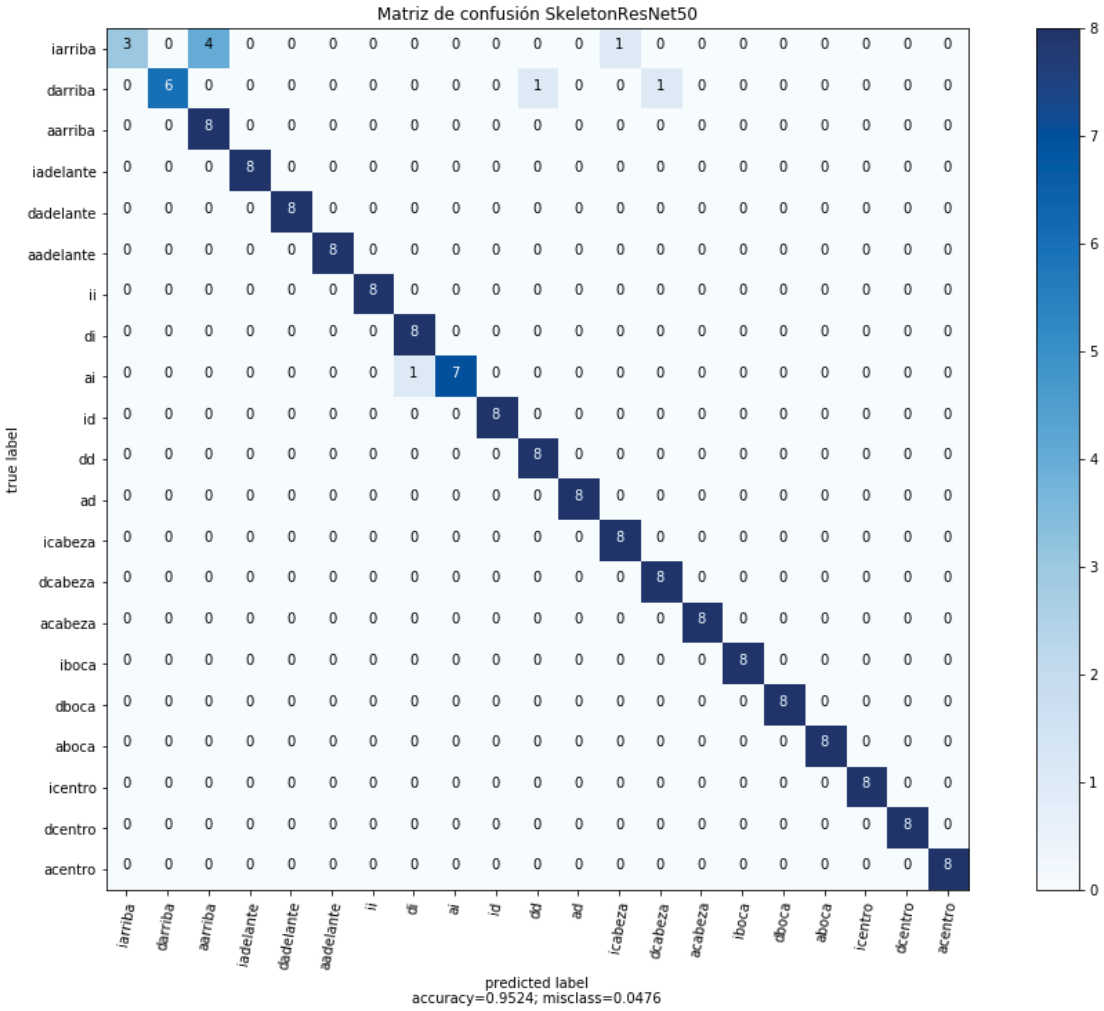
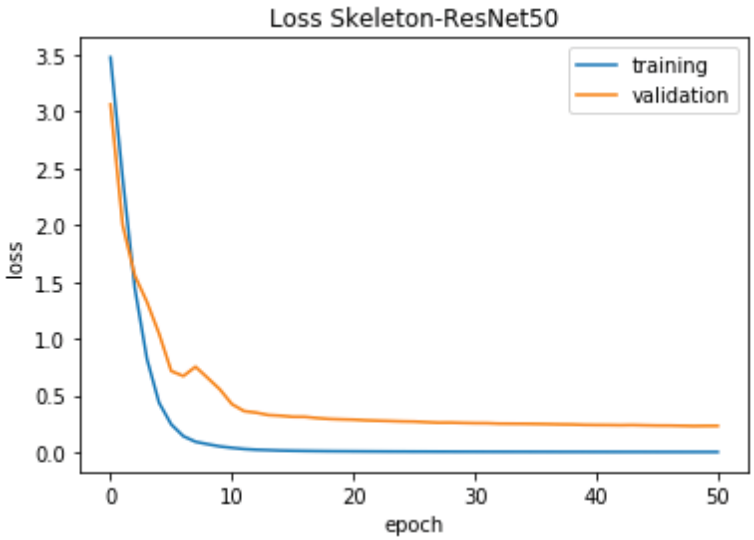
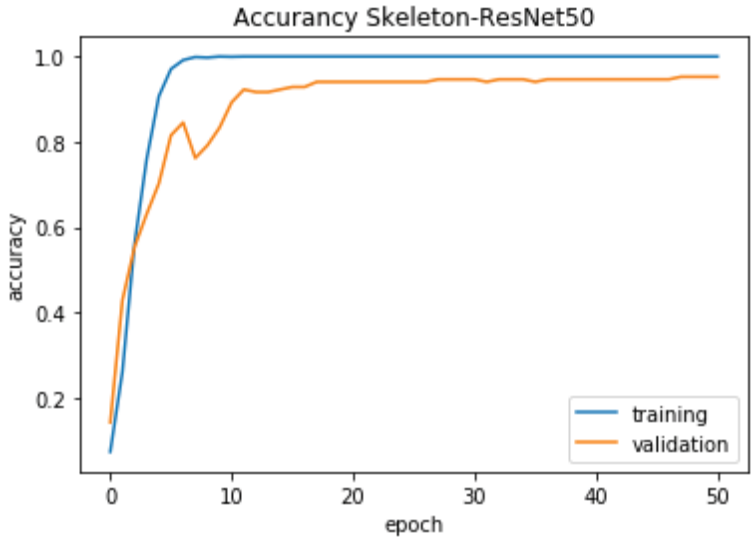


Figura 5.3: Matriz de confusión de la arquitectura Skeleton-ResNet50.
Fuente: Propia

Matriz de confusión 5.3 en base al vocabulario de la tabla 3.2, se muestra que logro una exactitud de 0.9524.



(a) Loss



(b) Accuracy

Figura 5.4: Train Skeleton-ResNet50.
Fuente: Propia

En la figura 5.4 se puede apreciar el numero de iteraciones sobre todo nuestro conjunto de entrenamiento(epoch) para poder conseguir la tasa de exactitud ya mencionada sin recurrir al *overfitting*.

5.2. Resultados de entrenamiento del modelo propuesto

El modelo propuesto se evaluó independientemente en los datos *rgb*, *depth* y la combinación de estos datos *(rgb, depth, esqueleto)*, en cada etapa se consigue una tasa de precisión aceptable.

5.2.1. Evaluación del modelo en los datos RGB

El resultado del entrenamiento del modelo propuesto con los datos RGB de la tabla 3.3 se muestra a continuación, donde se obtuvo una tasa de exactitud de 88.85 % en el conjunto de validación.

	Precision	Recall	F1 score	#Data
ayudame	0.714	0.5	0.59	10.0
como	1.0	1.0	1.0	10.0
cual	1.0	0.9	0.95	10.0
disculpame	0.75	0.75	0.75	10.0
donde	1.0	0.7	0.82	10.0
entiendo	0.833	1.0	0.91	10.0
es	1.0	1.0	1.0	10.0
estas	1.0	1.0	1.0	10.0
porfavor	0.875	0.778	0.82	10.0
gracias	0.9	0.75	0.82	10.0
haces	1.0	1.0	1.0	10.0
hasta	0.889	0.727	0.8	10.0
hola	1.0	1.0	1.0	10.0
manana	0.727	0.889	0.8	10.0
no	1.0	0.769	0.87	10.0
nombre	1.0	0.9	0.95	10.0
que	1.0	0.909	0.95	10.0
tu	1.0	0.925	0.96	40.0
vives	0.7	1.0	0.82	10.0
?	1.0	0.878	0.94	40.0
<unk>	0.0	0.0	0.0	0.0
Avg/total	0.888	0.84	0.858	260.0

Tabla 5.3: Resultados obtenidos en el *dataset* VideoLSP10_Total - RGB

En la tabla 5.3 se puede apreciar los niveles de precisión, recall, F1 score y la cantidad de datos utilizada para dicha evaluación. En la columna de precisión, se puede apreciar que "*ayudame*" tiene una precisión del 0.714, es decir, que cuando el modelo predice que es ayúdame acierta el 71.4 % de las veces, por otro lado en la siguiente columna, "*ayudame*" tiene un recall de 0.50, es decir que identifica correctamente el 50 % de todas las palabras que realmente fue etiquetada como "*ayudame*", la columna "*F*1 *score*" tiene una precisión de 0.59, es decir nuestro modelo realiza un buen trabajo para "*ayudame*" y puede generalizar su información para la predicción de datos con la misma distribución puesto que es mayor que 0.5, finalmente el modelo tiene una precisión global de 88.8 %.

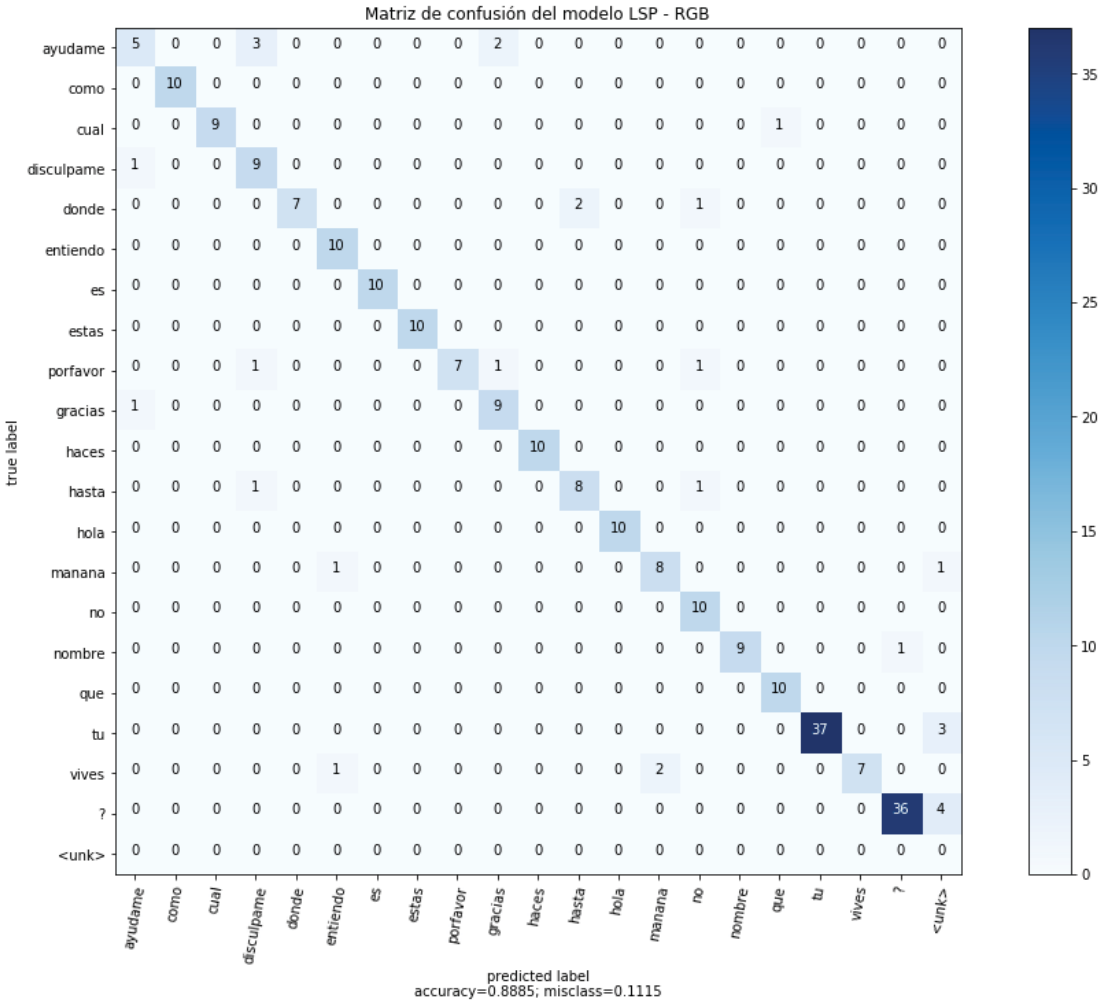
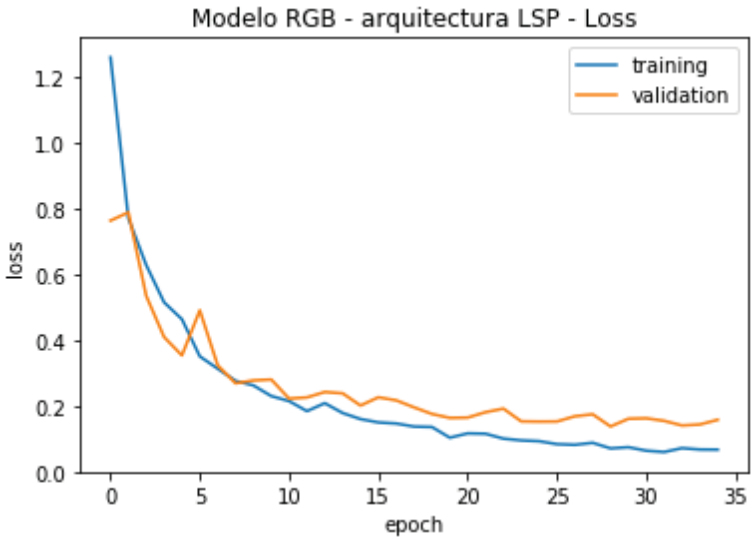
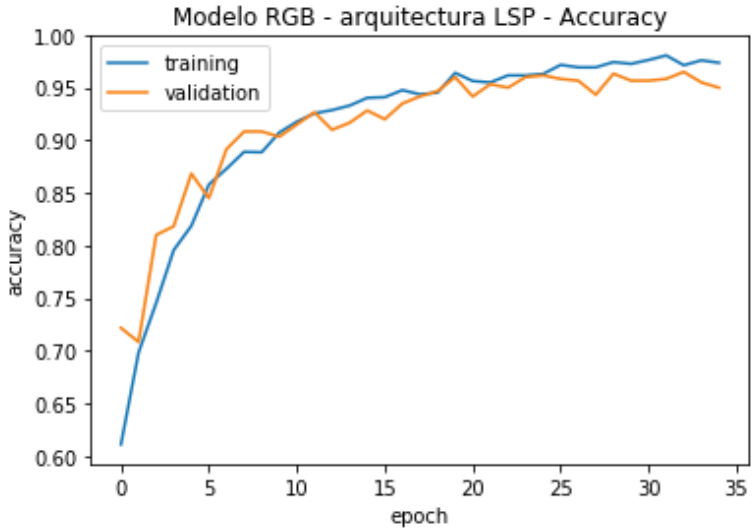


Figura 5.5: Matriz de confusión de la arquitectura LSP - RGB.
Fuente: Propia

Matriz de confusión 5.5 en base al vocabulario de la tabla 3.3, se muestra que logro una exactitud de 0.8885.



(a) Loss



(b) Accuracy

Figura 5.6: Train arquitectura LSP - RGB.
Fuente: Propia

En la figura 5.6 se puede apreciar el numero de iteraciones sobre todo nuestro conjunto de entrenamiento(epoch) para poder conseguir la tasa de precisión ya mencionada sin recurrir al *overfitting*.

5.2.2. Evaluación del modelo en los datos Depth

El resultado del entrenamiento del modelo propuesto con los datos *depth* de la tabla 3.3 se muestra a continuación, donde se obtuvo una tasa de exactitud de 85.82 % en el conjunto de validación.

	Precision	Recall	F1 score	#Data
ayudame	0.833	1.0	0.91	10.0
como	0.909	1.0	0.95	10.0
cual	1.0	0.6	0.75	10.0
disculpame	0.818	0.9	0.86	10.0
donde	0.889	0.571	0.7	10.0
entiendo	0.889	0.8	0.84	10.0
es	1.0	0.6	0.75	10.0
estas	0.909	1.0	0.95	10.0
porfavor	0.818	1.0	0.9	10.0
gracias	0.7	0.875	0.78	10.0
haces	0.889	0.889	0.89	10.0
hasta	0.909	0.909	0.91	10.0
hola	0.909	1.0	0.95	10.0
manana	0.909	1.0	0.95	10.0
no	0.8	0.889	0.84	10.0
nombre	1.0	0.7	0.82	10.0
que	0.8	1.0	0.89	10.0
tu	0.949	0.949	0.95	40.0
vives	0.8	0.571	0.67	10.0
?	0.946	0.833	0.89	40.0
<unk>	0.0	0.0	0.0	1.0
Avg/total	0.842	0.814	0.817	261.0

Tabla 5.4: Resultados obtenidos en el *dataset* VideoLSP10_Total - Depth

Finalmente el modelo tiene una precisión global[1] del 84.2 %, un recall[2] de 81.4 % y un F1 score[3] de 81.7 %

[1] ¿Qué proporción de identificaciones positivas fue correcta?
[2] ¿Qué proporción de positivos reales se identificó correctamente?
[3] Métrica que combina memoria y precisión utilizando la media armónica

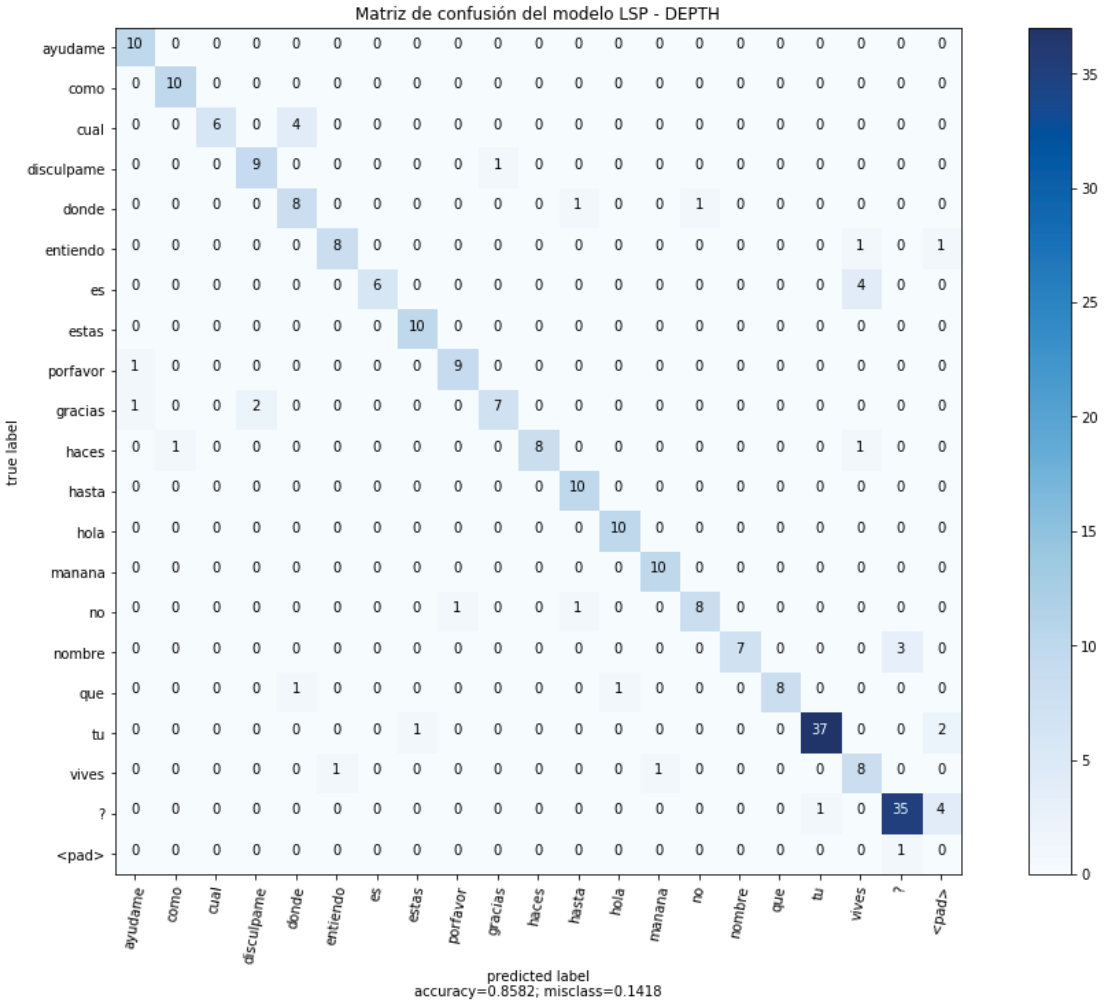
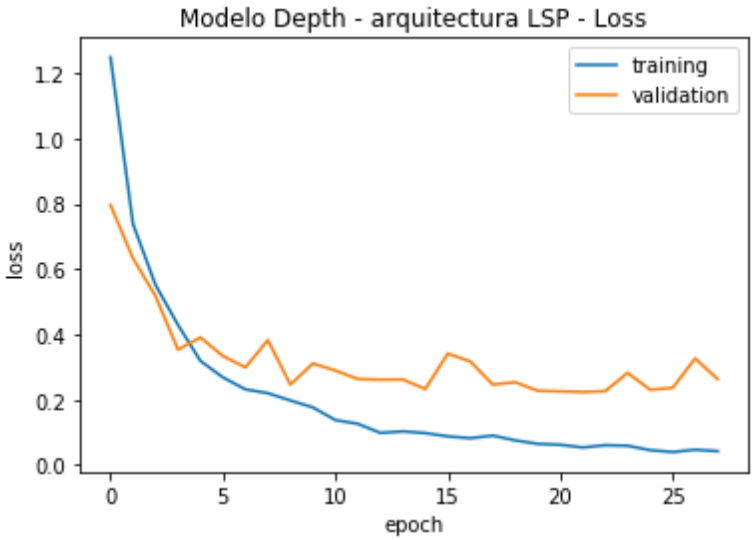
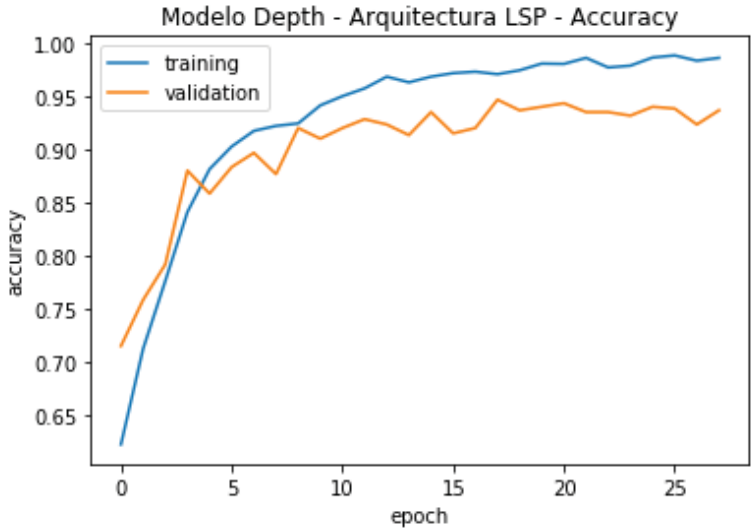


Figura 5.7: Matriz de confusión de la arquitectura LSP - DEPTH.
Fuente: Propia

Matriz de confusión 5.7 en base al vocabulario de la tabla 3.3, se muestra que logro una exactitud de 0.8582.



(a) Loss



(b) Accuracy

Figura 5.8: Train arquitectura LSP - DEPTH.
Fuente: Propia

En la figura 5.8 se puede apreciar el numero de iteraciones sobre todo nuestro conjunto de entrenamiento(epoch) para poder conseguir la tasa de precisión ya mencionada sin recurrir al *overfitting*.

5.2.3. Evaluación del modelo combinando los 3 tipos de datos

Nuestro modelo propuesto consigue una tasa de exactitud de 99.23 % puesto que al combinar la información de los datos *rgb, depth* y *skeleton*, la red tiene muchas características con la cual pueda discriminar un gesto, por ejemplo con los datos *rgb* puede identificar características visibles como el numero de dedos, diferenciar el color de la mano del color de la ropa, con los datos *depth* puede identificar la secuencia que sigue la mano, posición de las manos y brazos con respecto al cuerpo en cada ínstate, de alguna manera las distancias entre cada dedos y finalmente los datos *skeleton* se usaron para reforzar la información de la posición de las partes del cuerpo.

	Precision	Recall	F1 score	#Data
Ayudame	1.0	1.0	1.0	10.0
Como	1.0	1.0	1.0	10.0
Cual	1.0	1.0	1.0	10.0
Disculpame	1.0	1.0	1.0	10.0
Donde	1.0	1.0	1.0	10.0
Entiendo	1.0	1.0	1.0	10.0
Es	1.0	1.0	1.0	10.0
Estas	1.0	1.0	1.0	10.0
Por favor	1.0	1.0	1.0	10.0
Gracias	0.909	1.0	0.95	10.0
Haces	1.0	1.0	1.0	10.0
Hasta	0.9	1.0	0.95	10.0
Hola	1.0	1.0	1.0	10.0
Mañana	1.0	0.9	0.95	10.0
No	1.0	1.0	1.0	10.0
Nombre	1.0	1.0	1.0	10.0
Que	1.0	1.0	1.0	10.0
Tu	1.0	1.0	1.0	40.0
Vives	1.0	1.0	1.0	10.0
?	1.0	1.0	1.0	40.0
<pad>	0.0	0.0	0.0	0.0
Avg/total	0.99	0.995	0.992	260.0

Tabla 5.5: Resultados obtenidos en *dataset* VideoLSP10_Total

El numero de inferencias correctas es de 99 de 100, el modelo puede inferir frases de longitud 6, véase la tabla 5.6, la columna "*prediccion*" muestra el numero de frases inferidas, la columna "*error*" muestra el numero de frases inferidas erróneamente y la columna "$frase inferida$" muestra la frase errónea.

frases	# predicciones	# errores	frase inferida
Ayúdame	10	0	-
Por favor	10	0	-
Discúlpame	10	0	-
Cual es tu nombre?	10	0	-
Donde vives tu?	10	0	-
No entiendo	10	0	-
Que haces tu?	10	0	-
Hola, como estas tu ?	10	0	-
Gracias	10	0	hasta mañana
Hasta mañana	9	1	-

Tabla 5.6: Resultado de frases inferidas por el modelo LSP propuesto

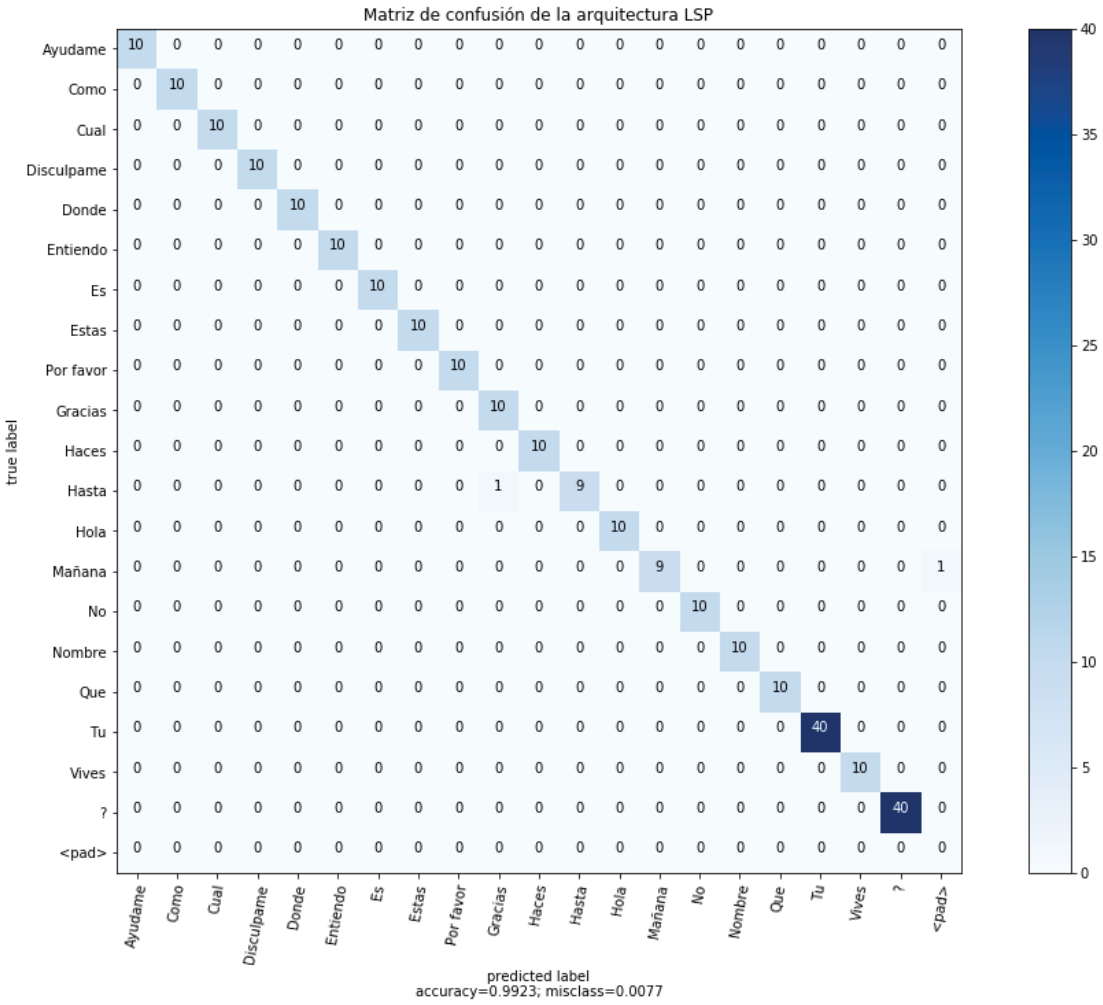
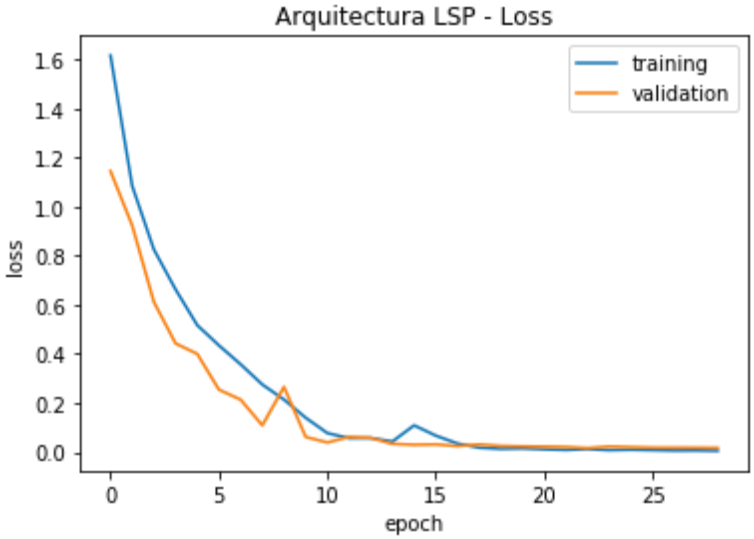
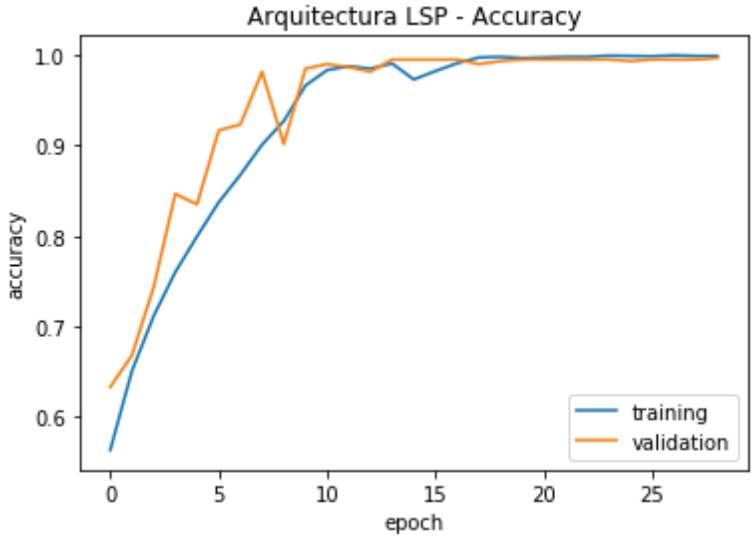


Figura 5.9: Matriz de confusión de la arquitectura LSP propuesta
Fuente: Propia

Matriz de confusión 5.9 en base al vocabulario de la tabla 3.3, se muestra que logro una exactitud de 0.9923.



(a) Loss

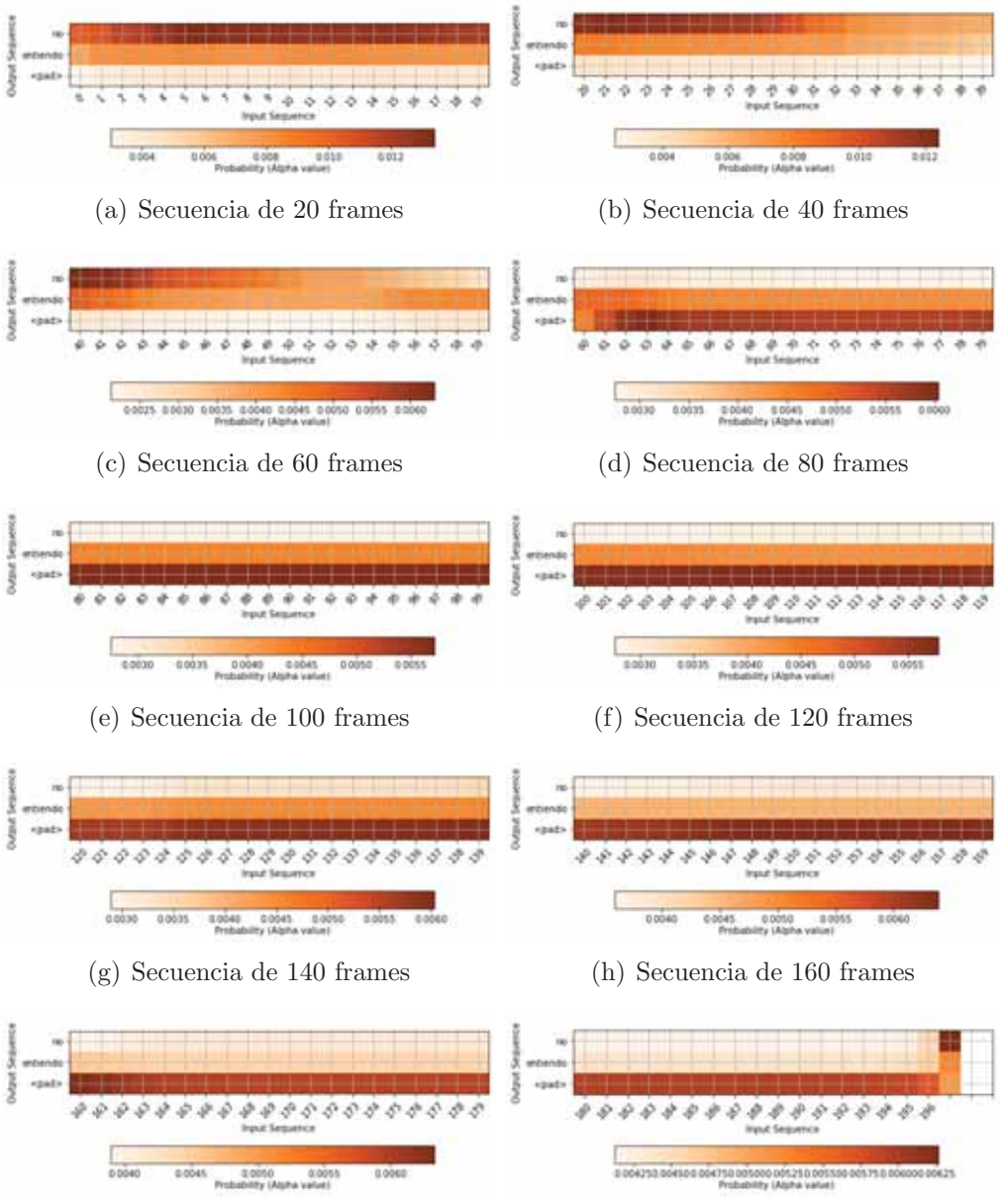


(b) Accuracy

Figura 5.10: Train arquitectura LSP propuesta.
Fuente: Propia

En la figura 5.10 se puede apreciar el numero de iteraciones sobre todo nuestro conjunto de entrenamiento(epoch) para poder conseguir la tasa de precisión ya mencionada sin recurrir al *overfitting*.

Para interpretar una determinada salida el *attention model* toma los datos de una entrada y selecciona aquellas entradas que posiblemente aporten la mayor cantidad información para una determinada salida y les asigna una ponderación mayor que el resto de las entradas, por ejemplo para la frase “no entiendo” con 197 frames, vease figura 5.11, el *attention model* elige los frames del 2 hasta la 29 considerando estas como mas relevantes para predecir la palabra “no”, este mismo proceso se realiza para toda las palabras y finalmente hasta llegar al “pad”, el “pad” hace referencia el final de una sentencia.



(a) Secuencia de 20 frames

(b) Secuencia de 40 frames

(c) Secuencia de 60 frames

(d) Secuencia de 80 frames

(e) Secuencia de 100 frames

(f) Secuencia de 120 frames

(g) Secuencia de 140 frames

(h) Secuencia de 160 frames

(i) Secuencia de 180 frames

(j) Secuencia de 200 frames

Figura 5.11: Interpretacion de frames
Tabla de analisis para la secuencia de frames de la frase No entiendo. Fuente: Propia

5.3. Evaluación de los modelos experimental

El desempeño de la arquitectura que proponemos se experimento en la base de datos LSA64 y se comparo con los modelos propuestos en el paper (Masood et al., 2018), donde nuestro modelo propuesto demuestra superioridad en la predicción.

5.3.1. Modelo experimental 1

El modelo experimental 1 consigue una tasa de exactitud de 90.62 % como se aprecia en la tabla 5.7

En la tabla 5.7 se puede apreciar los niveles de precisión, recall, F1 score y la cantidad de datos utilizada para dicha evaluación, como también se muestra la precisión global que el modelo 1 ha alcanzado.

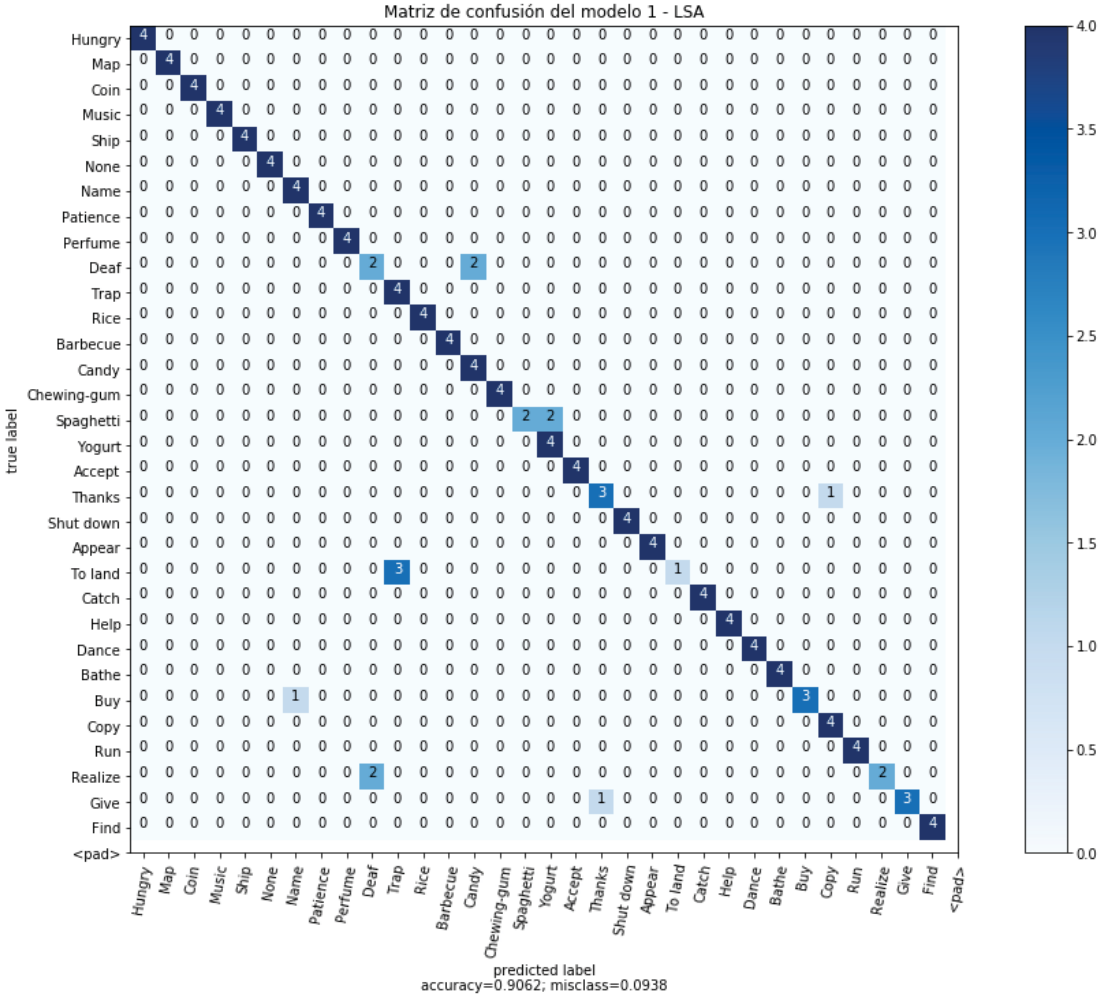
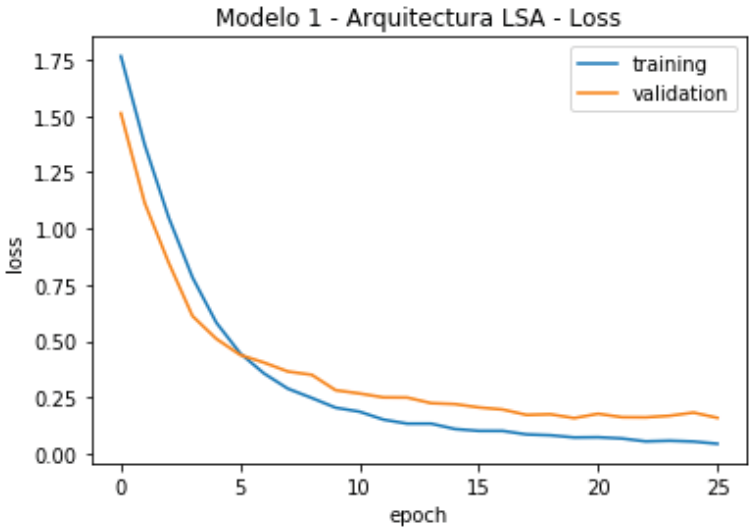


Figura 5.12: Matriz de confusión del modelo 1.
Fuente: Propia

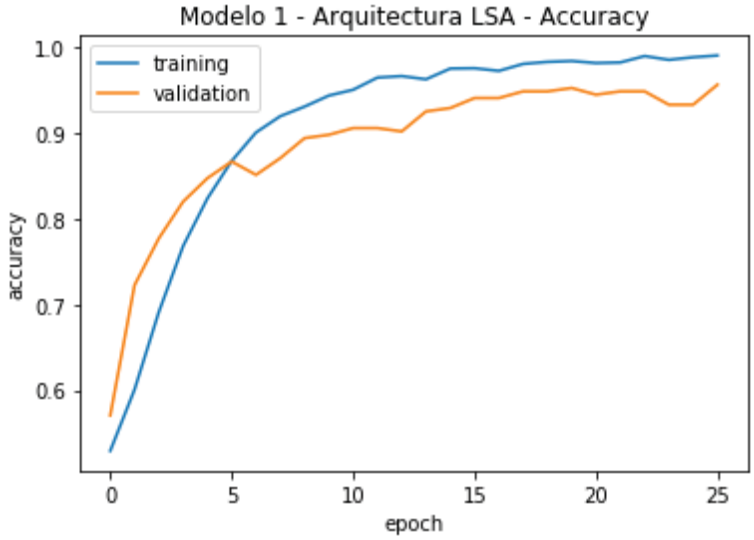
Matriz de confusión en base al vocabulario de la base de datos LSA64, donde se infirió correctamente 116 palabras de un total de 128.

	Precision	Recall	F1 score	#Data
Hungry	1.0	1.0	1.0	4.0
Map	1.0	1.0	1.0	4.0
Coin	1.0	1.0	1.0	4.0
Music	1.0	1.0	1.0	4.0
Ship	1.0	1.0	1.0	4.0
None	1.0	1.0	1.0	4.0
Name	0.8	1.0	0.89	4.0
Patience	1.0	1.0	1.0	4.0
Perfume	1.0	1.0	1.0	4.0
Deaf	0.5	0.5	0.5	4.0
Trap	0.571	1.0	0.73	4.0
Rice	1.0	1.0	1.0	4.0
Barbecue	1.0	1.0	1.0	4.0
Candy	1.0	0.667	0.8	4.0
Chewing-gum	1.0	1.0	1.0	4.0
Spaghetti	1.0	0.5	0.67	4.0
Yogurt	1.0	0.667	0.8	4.0
Accept	1.0	1.0	1.0	4.0
Thanks	0.75	0.75	0.75	4.0
Shut down	1.0	1.0	1.0	4.0
Appear	1.0	1.0	1.0	4.0
To land	0.25	1.0	0.4	4.0
Catch	1.0	1.0	1.0	4.0
Help	1.0	1.0	1.0	4.0
Dance	1.0	1.0	1.0	4.0
Bathe	1.0	1.0	1.0	4.0
Buy	0.75	1.0	0.86	4.0
Copy	1.0	0.8	0.89	4.0
Run	1.0	1.0	1.0	4.0
Realize	0.5	1.0	0.67	4.0
Give	0.75	1.0	0.86	4.0
Find	1.0	1.0	1.0	4.0
Avg/total	0.902	0.934	0.901	128.0

Tabla 5.7: Resultados obtenidos en la base de datos LSA - modelo 1.
Fuente: Propia



(a) Loss



(b) Accuracy

Figura 5.13: Train modelo 1 en la base de datos LSA64.
Fuente: Propia

En la figura 5.13 se puede apreciar el numero de iteraciones sobre todo el conjunto de entrenamiento(epoch) para poder conseguir la tasa de precisión ya mencionada sin recurrir al *overfitting*.

5.3.2. Modelo experimental 2

El modelo experimental 2 consigue una tasa de exactitud de 98.44 % como se aprecia en la tabla 5.8

En la tabla 5.8 se puede apreciar los niveles de precisión, recall, F1 score y la cantidad de datos utilizada para dicha evaluación, como también se muestra la precisión global que el modelo 1 ha alcanzado.

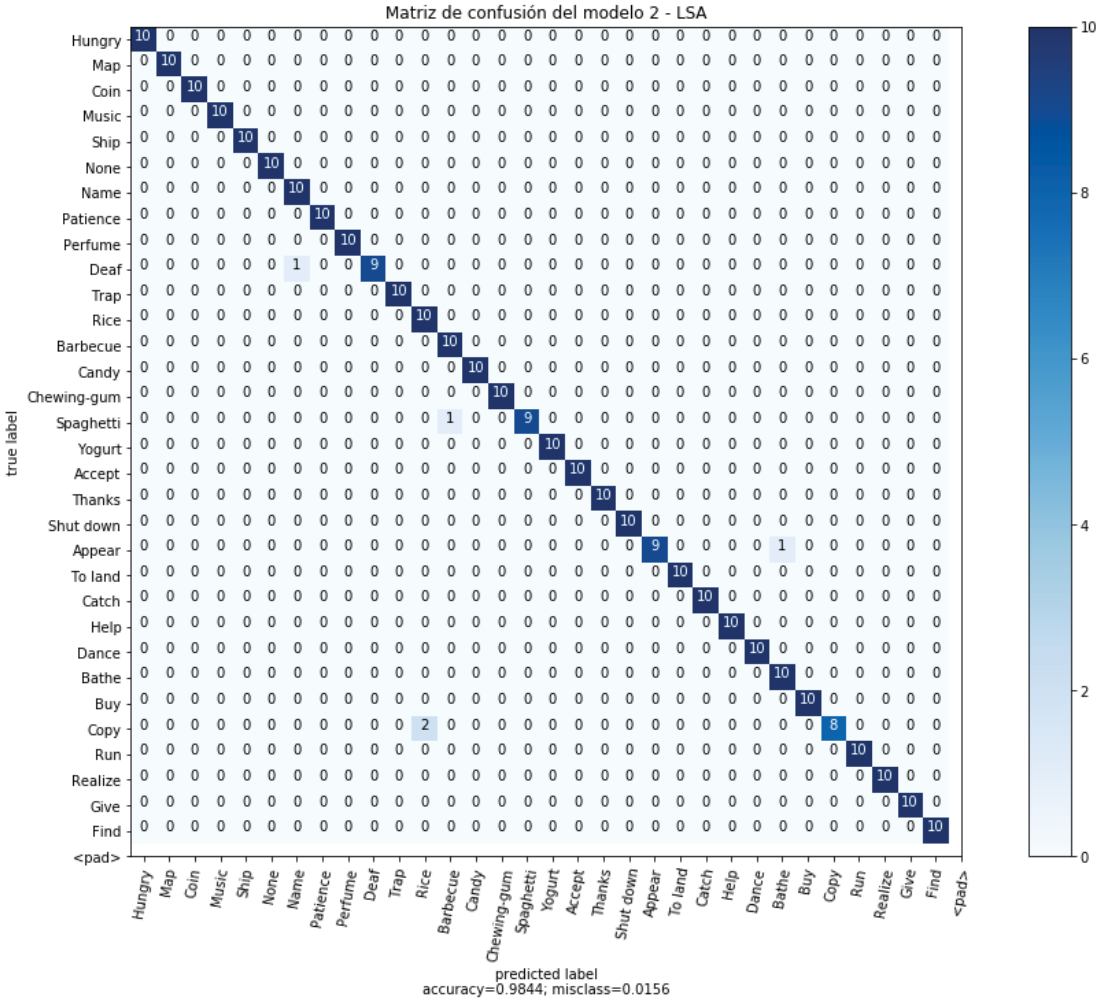


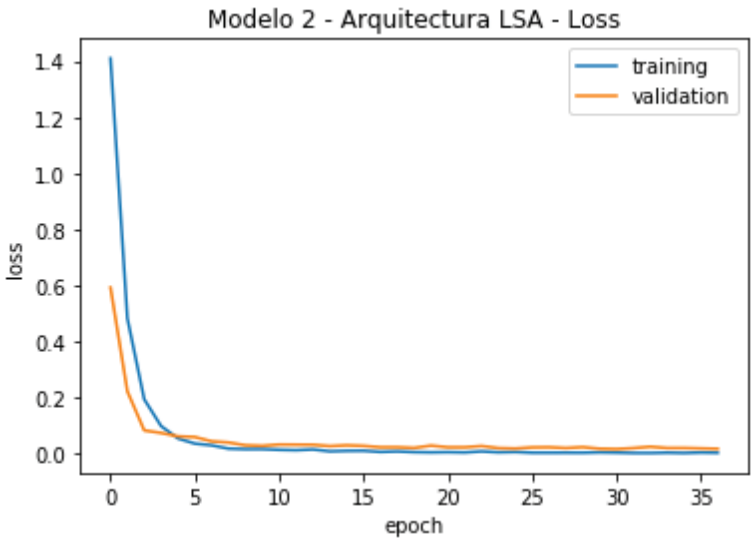
Figura 5.14: Matriz de confusión del modelo 2.
Fuente: Propia

Matriz de confusión en base al vocabulario de la base de datos LSA64, donde se infirió correctamente 315 palabras de un total de 320.

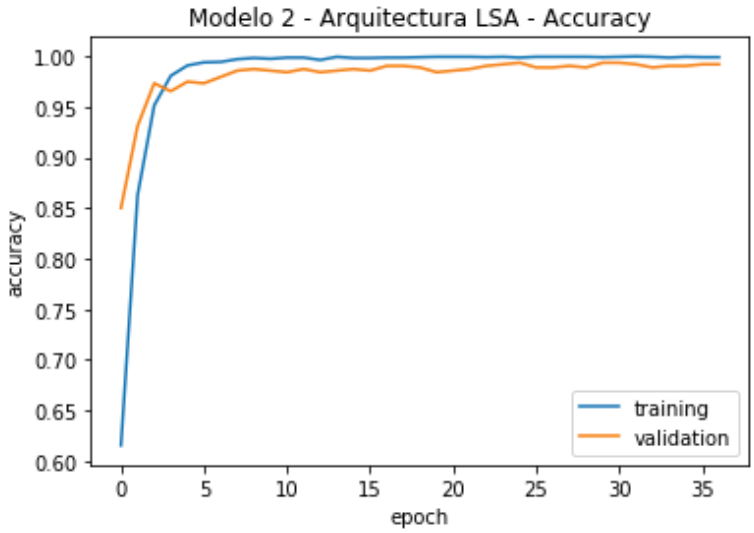
En la figura 5.15 se puede apreciar el numero de iteraciones sobre todo el conjunto de entrenamiento(epoch) para poder conseguir la tasa de precisión ya mencionada sin recurrir al *overfitting*.

	Precision	Recall	F1 score	#Data
Hungry	1.0	1.0	1.0	10.0
Map	1.0	1.0	1.0	10.0
Coin	1.0	1.0	1.0	10.0
Music	1.0	1.0	1.0	10.0
Ship	1.0	1.0	1.0	10.0
None	1.0	1.0	1.0	10.0
Name	0.909	1.0	0.95	10.0
Patience	1.0	1.0	1.0	10.0
Perfume	1.0	1.0	1.0	10.0
Deaf	0.9	1.0	0.95	10.0
Trap	1.0	1.0	1.0	10.0
Rice	0.833	1.0	0.91	10.0
Barbecue	0.909	1.0	0.95	10.0
Candy	1.0	1.0	1.0	10.0
Chewing-gum	1.0	1.0	1.0	10.0
Spaghetti	0.9	1.0	0.95	10.0
Yogurt	1.0	1.0	1.0	10.0
Accept	1.0	1.0	1.0	10.0
Thanks	1.0	1.0	1.0	10.0
Shut down	1.0	1.0	1.0	10.0
Appear	1.0	0.9	0.95	10.0
To land	1.0	1.0	1.0	10.0
Catch	1.0	1.0	1.0	10.0
Help	1.0	1.0	1.0	10.0
Dance	1.0	1.0	1.0	10.0
Bathe	1.0	0.909	0.95	10.0
Buy	1.0	1.0	1.0	10.0
Copy	0.8	1.0	0.89	10.0
Run	1.0	1.0	1.0	10.0
Realize	1.0	1.0	1.0	10.0
Give	1.0	1.0	1.0	10.0
Find	1.0	1.0	1.0	10.0
Avg/total	0.977	0.994	0.984	320.0

Tabla 5.8: Resultados obtenidos en la base de datos LSA - modelo 2



(a) Loss



(b) Accuracy

Figura 5.15: Train modelo 2 en la base de datos LSA64.
Fuente: Propia

5.3.3. Comparación

El modelo 1 se comparo usando la misma base de datos y el mismo tipo de preprocesamiento que realiza el primer enfoque *Prediction Approach* del paper (Masood et al., 2018), el modelo 2 de igual manera se comparo con el segundo enfoque *Pool Layer Approach* del paper (Masood et al., 2018) realizando el mismo tipo de preprocesamiento y el mismo tipo de datos. Se observa que nuestros modelos superan a los resultados que muestra el paper (Masood et al., 2018) y cuyos valores son plasmados en la tabla 5.9.

Modelo	**% accuracy**
Prediction Approach	80.87
Modelo 1	90.62
Pool Layer Approach	95.21
Modelo 2	98.44

Tabla 5.9: Resultados de los modelos

5.4. Detalles técnicos de software

El software utilizado para desarrollar el proyecto es:

- **Sistema operativo:** Windows 10 Pro.
- **Lenguaje de programación:** Se utilizo el lenguaje C# para desarrollar la aplicación de captura de datos y python 2.7 para la etapa de implementación de la arquitectura, como el preprocesamiento y manejo de datos.
- **Librerías:**
  - **Tensorflow:** Se utilizo como backend de la API keras.
  - **Keras:**Se utilizo para realizar en la etapa de *transfer learning*, entrenamiento, creación de las arquitecturas, métodos de optimización, funciones de activación, entre otros vistos en el desarrollo del proyecto.
  - **OpenCV:** Se utilizo para procesos de lectura de imágenes y dimensionamiento.
  - **Numpy:** Librería matemática para realizar operaciones matriciales.
  - **Matplotlib:** herramienta de visualización de datos.
  - **Pandas:** Herramienta de análisis y manipulación de datos, como tablas de información.
  - **HDF5:** Conjunto de formatos de archivos diseñados para almacenar y organizar grandes cantidades de datos, se uso para el almacenamiento del *dataset*.

Conclusiones

1. El modelo propuesto para la interpretación de LSP propone una solución tecnológica a la barrera de la comunicación creada entre las personas de lengua hablada y lengua de señas.

2. El modelo planteado en este trabajo aun restringido por las distintas limitaciones al momento de la implementación logra una precisión muy aceptable al momento de la evaluación con una tasa de 99.23 %, se podría mejorar la inferencia únicamente en los datos rgb incrementando el conjunto de entrenamiento y realizando un mejor preprocesamiento específicamente para las manos.

3. Los modelos y técnicas seleccionados como las redes Bidireccionales LSTM y *Attention Decoder* fueron muy útiles en la parte de RNN, sin embargo se obtuvo una gran mejora al usar técnicas como *maxout*, *dropout* y residual LSTM. Otros metodos utiles fueron *Early Stopping* que nos ayudo a que nuestra red neuronal no realice *overfitting*.

4. El uso de distintos tipos de información como los datos *detph*, en base a la distancia y datos *skeleton* contribuyen de mejor manera al aporte de información en esta arquitectura, aun en esta era de la información es difícil conseguir datos para una tarea especifica como la nuestra, donde se recolecto 600 vídeos con un promedio de 100 *frames*.

5. La arquitectura de redes convolucion al aplicar *transfer learning* consiguieron una tasa de exactitud de 97.6 % para los datos *depth*, 95.24 % para los datos *skeleton* y para la arquitectura final alcanzo un 99.23 % de exactitud, la evaluación se realizo en nuestra base de datos LSP. En la comparación se muestra una mejora notable frente a los modelo propuestos en el paper (Masood et al., 2018), se obtuvo un 90.62 % para el modelo auxiliar 1 y un 98.44 % para el modelo auxiliar 2, pese a las limitaciones que se tuvieron en este trabajo.

6. Se verifica también que nuestro dataset LSP es muchos mas desafiante que el *dataset* LSA64 puesto que la arquitectura RGB en nuestro *dataset* alcanzo una tasa de 88.8 % y la misma arquitectura con la misma configuración consiguió una tasa de 98.44 % en el *dataset* LSA64.

7. Según la experimentación en el desarrollo de la arquitectura se concluye que las redes residuales recurrentes tienen mejor desempeño en el *encoder* en lugar del *decoder*, las capas *maxout* pueden incrementar la precisión de una arquitectura.

8. Finalmente se concluye que un modelo consigue una precisión mas alta al incrementar y combinar diferentes tipos de información para una tarea especifica.

Trabajos Futuros

Esta investigación se llego a desarrollar un interprete para el LSP, a futuro se desea ampliar el alcance con los siguientes puntos:

- Desarrollar un *dataset* mas completo para el vocabulario de LSP
- Implementar un sistema bidireccional para el interprete LSP, que el interprete puede traducir de un lengua hablada a una lengua de señas y viceversa.
- Extender la funcionabilidad del interprete generando en lugar de una salida textual una transcripción de texto a voz.
- Implementar el interprete para que funcione con menos restricciones de software, de manera que el Kinect sea remplazado por el Kinet v2 o un sensor TOF mas pequeño, accesible y portable.

Bibliografía

Aggarwal, C. (2018). *Neural Networks and Deep Learning: A Textbook*. Springer.

Andrew, N. (2017). Deep learning specialization. deeplearning.ai. from https://www.deeplearning.ai/.

Andrew Ng, Jiquan Ngiam, C. Y. F. (2013). Optimization: Stochastic gradient descent. Deep Learning Computer Science Department, Stanford University. from http://ufldl.stanford.edu/tutorial/.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *ArXiv e-prints*.

Benitez, A. (2017). Inteligencia artificial. emaze. https://app.emaze.com/@AOIWQOCOQ#1.

Bottou, L. (2012). *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg.

Briega, R. E. L. (2017). Introducción al deep learning. IAAR Capacitacion. https://iaarhub.github.io/capacitacion/2017/06/13/introduccion-al-deep-learning/.

Brownlee, J. (2018). Why initialize a neural network with random weights? Machine Learning Mastery. from https://machinelearningmastery.com/.

Buduma, N. and Locascio, N. (2017). *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. O'Reilly Media.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *ArXiv e-prints*.

Daza, S. L. P. (2005). Lenguaje, lengua, habla, idioma y dialecto. In *Lenguas del Mundo. Por la ruta de Babel*, number n.º 71 in La Tadeo. Fundacion Universidad de Bogota Jorge Tadeo Lozano.

de Pablos Heredero, C. (2004). *Informática y comunicaciones en la empresa*. Biblioteca de economía y finanzas. ESIC Editorial.

Deeplearning4j (2014). Early stopping. Eclipse Deeplearning4j Development Team. from Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0.

DIGEBE (2015). *Lengua de Señas Peruana*. Ministerio de Educacion, 2da edition.

Donges, N. (2018). Transfer learning. Medium, Towards Data Science. from https://towardsdatascience.com/transfer-learning-946518f95666.

Doshi, N. (2018). Deep learning best practices (1) — weight initialization. Medium. from https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94.

Dzmitry, B. and KyungHyun Cho, Y. B. (2015). Neural machine traslation by jointly learning to align and translate. *ICLR*.

Ferriere, P. (2017). Resnet architecture question. GitHub Inc. from https://github.com/matterport/Mask_RCNN/issues/117.

Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc.

García Benavides, I. (2004). La lingüística en el lenguaje de señas. *Cultura Sorda*. from http://www.cultura-sorda.org/la-linguistica-en-el-lenguaje-de-senas/.

Goldberg, Y. and Hirst, G. (2017). *Neural Network Methods in Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

González, C. and Felpeto, A. (2006). *Tratamiento de datos*. Díaz de Santos.

Goodfellow, I., Bengio, Y., and Courvil, A. (2016). *Deep Learning*. MIT Press. from http://www.deeplearningbook.org.

GoogleDevelopers (2018). Curso intensivo de aprendizaje automatico. Aprendizaje Profundo, Google Developers. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative.

Grimaldo, D. (2015). Un analisis de las particularidades de la lengua de señas peruana. Pontificia Universidad Catolica del Peru, PuntoEdu. https://puntoedu.pucp.edu.pe/noticias/un-analisis-de-las-particularidades-de-la-lengua-de-senas-peruana/.

Harris, D. and Harris, S. (2010). *Digital Design and Computer Architecture*. Computer organization bundle, VHDL Bundle Series. Elsevier Science.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

INEI (2015). *Peru, Características de la Población con Discapacidad*. Instituto Nacional de Estadística e Informática, Jesús María, Lima.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*.

Isasi, P. V. and Galvan, I. M. L. (2004). *Redes de Neuronas Artificiales. Un enfoque Práctico*. Pearson Educación, Madrid, España.

Jadon, S. (2018). Introduction to different activation functions for deep learning. Medium. from https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092.

Jana, A. (2012). *Kinect for Windows SDK Programming Guide*. Packt Publishing, Birmingham.

Jones, M. T. (2017). Deep learning architectures. IBM developer. https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/.

Karpathy, A. (2018). Transfer learning. CS231n: Convolutional Neural Networks for Visual Recognition. from http://cs231n.github.io/transfer-learning/.

Kathuria, A. (2018). Intro to optimization in deep learning: Gradient descent. paperspace, Series: Optimization. from https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *ArXiv e-prints*.

Koehrsen, W. (2018). Beyond accuracy: Precision and recall. Medium, Towards Data Science. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c.

Laraba, S., Brahimi, M., Tilmanne, J., and Dutoit, T. (2017). 3d skeleton-based action recognition by representing motion capture sequences as 2d-rgb images. *Computer Animation and Virtual Worlds*, 28.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400.

Luong, M., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

Lyons, J. (1984). *Introduccion al lenguaje y a la Linguistica*. Teide, Barcelona, n.º 1 edition.

Mao, C., Huang, S., Li, X., and Ye, Z. (2017). Chinese sign language recognition with sequence to sequence learning. In *Computer Vision*, pages 180–191, Singapore. Springer Singapore.

Masood, S., Srivastava, A., Thuwal, H. C., and Ahmad, M. (2018). Real-time sign language gesture (word) recognition from video sequences using cnn and rnn. In Bhateja, V., Coello Coello, C. A., Satapathy, S. C., and Pattnaik, P. K., editors, *Intelligent Engineering Informatics*, pages 623–632, Singapore. Springer Singapore.

Navarro, S. (2015). Cinco nociones sobre el lenguaje de señas peruano. Pontificia Universidad Catolica del Peru, PuntoEdu. https://puntoedu.pucp.edu.pe/noticias/cinco-nociones-sobre-el-lenguaje-de-senas-peruano/.

Olah, C. (2015). Understanding lstm networks. colah's blog. from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Patterson, J. and Gibson, A. (2017). *Deep Learning: A Practitioner's Approach*. O'Reilly Media.

Pedro Bekinschtein, Daniel Calvo, L. C. (2017). *Un libro sobre drogras*. El Gato y la Caja.

Perú, C. (2010). Decreto supremo que aprueba el reglamento de la ley nº 29535, ley que otorga reconocimiento oficial a la lengua de señas peruana. *El Peruano*. from http://busquedas.elperuano.pe/normaslegales/decreto-supremo-que-aprueba-el-reglamento-de-la-ley-n-29535-decreto-supremo-n-006-2017-mimp-1554509-5/.

Prabhu (2018). Understanding of convolutional neural network (cnn) - deep learning. Medium.

Prechelt, L. (2012). *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg.

Raffel, C. and Ellis, D. P. W. (2015). Feed-forward networks with attention can solve some long-term memory problems. *CoRR*, abs/1512.08756.

Ronchetti, F., Quiroga, F., Estrebou, C., Lanzarini, L., and Rosete, A. (2016). Lsa64: A dataset of argentinian sign language. *XX II Congreso Argentino de Ciencias de la Computación (CACIC)*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Ruder, S. (2017). Transfer learning - machine learning's next frontier. Sebastian Ruder. from http://ruder.io/transfer-learning/.

Sampier, R. (2010). *Metodología de la investigación*. Editorial Félix Varela.

Saxena, A. (2016). Convolutional neural networks (cnns): An illustrated explanation. XRDS.

Singh, V. (2018). Understanding entropy, cross-entropy and cross-entropy loss. Medium. from https://medium.com/@vijendra1125/understanding-entropy-cross-entropy-and-softmax-3b79d9b23c8a.

Sotos, E. (2017). Cómo afectará la inteligencia artificial a nuestras vidas. Código Nuevo. https://www.codigonuevo.com/sociedad/afectara-inteligencia-artificial-vidas.

Su, P., Ding, X., Zhang, Y., Miao, F., and Zhao, N. (2017). Learning to predict blood pressure with deep bidirectional LSTM network. *CoRR*, abs/1705.04524.

Sucar, L. E. and Gómez, G. (2015). *Visión Computacional*. Instituto Nacional de Astrofísica, Óptica y Electrónica.

Ullah, A., Ahmad, J., Muhammad, K., Sajjad, M., and Baik, S. W. (2018). Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE Access*, 6:1155–1166.

Valls, J. (2014). *Introducción al lenguaje*. Editorial UOC, S.L.

Warren S. Sarle, Cary, N. (2002). Usenet newsgroup part 2 (of 7): What is a softmax activation function? Comp.ai.neural-nets FAQ. from http://www.faqs.org/faqs/ai-faq/neural-nets/part2/.

Zachary C. Lipton, Mu Li, A. S. (2017). Gradient descent and stochastic gradient descent from scratch. Gluon: Deep Learning - The Straight Dope. from https://gluon.mxnet.io/chapter06_optimization/gd-sgd-scratch.html.

Zhang, W., Liu, G., and Tian, G. (2017). Hha-based cnn image features for indoor loop closure detection. In *2017 Chinese Automation Congress (CAC)*, pages 4634–4639.