UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

RECONOCIMIENTO AUTOMÁTICO DE LENGUA DE SEÑAS MEXICANA USANDO UNA RED NEURONAL SWIN TRANSFORMER

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
DANIEL HERNÁNDEZ PEÑA

TUTOR O TUTORES PRINCIPALES
DR. GIBRAN FUENTES PINEDA
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Ciudad de México, abril, 2023

Resumen

En el presente trabajo se modificó, entrenó y evaluó una red neuronal *Swin Transformer* con el objetivo del reconocimiento automático de la Lengua de Señas Mexicana. Para poder lograr esto se recopiló un conjunto de datos de señas dentro de la Lengua de Señas Mexicana y se etiquetó con su respectivo significado en el idioma Español.

Para poder comparar nuestra red neuronal con otros experimentos que se han realizado sobre otras Lenguas de Señas se utilizaron como base las métricas de *Top 5 accuracy*, la cual indica el porcentaje de señas correctamente identificadas dentro de las 5 señas con mayor similitud y *Top 1 accuracy*, la cual es el porcentaje de señas correctamente identificadas dentro de la seña con mayor similitud.

Los resultados experimentales muestran que es posible el reconocimiento automático de señas aisladas dentro de la LSM

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Se estima que el 5 % de la población mundial (430 millones de personas) padecen de algún tipo de discapacidad auditiva, llegando a un 10 % de la población (más de 700 millones de personas) para el año 2050. En el caso de México, de acuerdo a un censo realizado en 2010 por el INEGI[1], el 4.84 % de la población (aprox. 5 millones) indicó tener algún tipo de discapacidad. Este porcentaje fue incrementando, y en el censo realizado en el 2020 por el INEGI[2], el 16.5 % (aprox. 5 millones de la población indicó tener algún tipo de discapacidad, donde el 24.4 % (aprox. 5 millones) indicó tener dificultades para escuchar y el 10.7 % (aprox. 2 millones) indicó tener problemas para hablar o comunicarse. Este aumento se debe, principalmente, al tener que la esperanza de vida de la población mexicana ha ido en aumento y teniendo que al envejecer es más probable que se desarrolle algún tipo de discapacidad auditiva y/o del habla, teniendo que recurrir a otros métodos de comunicación, entre ellos el aprender a comunicarse por medio de la LSM.

Estos datos permiten distinguir que la comunicación entre personas con algún tipo de discapacidad auditiva y/o del habla no se puede realizar con tanta facilidad al no haber los suficientes recursos ni infraestructura para garantizar una comunicación fluida.

1.1. Planteamiento del problema

Para afrontar el problema de comunicación entre personas con discapacidad auditiva y/o del habla, se han intentado varios métodos, la mayoría de las cuales se enfoca en el uso de modelos de traducción del habla a texto. Sin embargo, los traductores actualmente disponibles en español no son compatibles con la lengua de señas, esto al existir diferencias gramaticales y de sintaxis entre la LSM y el idioma español. Al utilizar ese tipo de métodos se tiene que la traducción se hace en idioma español y no en Lengua de Señas Mexicana (LSM).

1.2. Objetivos

1.2.1. Objetivos generales

Desarrollar un modelo basado en redes neuronales profundas capaz de reconocer la seña dentro de la LSM realizada en un video.

1.2.2. Objetivos específicos

Los objetivos específicos se listan a continuación:

- Recolectar y etiquetar un conjunto de videos con señas dentro de la LSM.
- Investigar el estado del arte sobre reconocimiento de lengua de señas.
- Desarrollar un modelo basado en la red neuronal Swin Transformer[34].
- Evaluar distintas configuraciones de la red Video Swin Transformer[35].

1.3. Retos para el trabajo

Al realizar el trabajo se afrontaron varios problemas listados a continuación

- En la actualidad existen diversos conjuntos de videos etiquetados de señas, cuyos tamaños van de decenas de miles a cientos de miles. Sin embargo, estos conjuntos son de la lengua de señas americana, británica o española y casi no existen recursos de la lengua de señas mexicana. Para poder entrenar los modelos es necesario contar con conjuntos de videos en los que cada seña sea realizada en diversas condiciones por diferentes señantes. Por tal motivo se generó uno propio. Esto provocó un gran uso de tiempo para identificar la fuente de los posibles videos y su etiquetado conforme a la seña que representan.
- De igual forma que en el lenguaje hablado, se tiene que para la lengua de señas existen diversas formas de expresar una palabra u oración por lo que es importante tomar en cuenta las variaciones.
- Los recursos de cómputo que se requieren normalmente tanto en cuestión de memoria como en procesamiento
- Para algunas señas se debe de considerar tanto el movimiento de las manos como la manera en que se realiza la seña. Dentro de esto se considera la posición de las manos respecto al cuerpo, el movimiento del rostro para complementar a la seña.

1.4. Metodología

La metodología para el reconocimiento de lengua de señas dentro de la LSM a partir de videos consistió de las siguientes etapas:

Para este trabajo se hicieron dos tareas muy importantes: La elaboración del conjunto de datos y la elaboración del modelo a ser entrenado.
1)Para la primera parte se buscaron videos dirigidos a personas con discapacidad auditiva, los cuales tuvieran algún señante que interpretaran lo hablado a seña. Se buscó de esta forma porque se obtienen datos que son cercanos a los reales al no ser un entorno controlado y al contener un vocabulario amplio al hablar sobre temáticas escolares.
2)Para la segunda parte se observó al estado del arte respecto a lo buscado, para finalmente utilizar los modelos Video Swin *Transformer*, los cuales son arquitecturas *Transformer* diseñadas para el procesamiento de videos, tomando en cuenta el uso de un decodificador al final para clasificar de manera correcta las señas procesadas.

1.5. Organización de la tesis

El trabajo se encuentra estructurado de la siguiente manera:

- **Capítulo 2**: Se expone el estado del arte respecto al reconocimiento de diferentes Lenguas de Señas, junto a las principales ventajas y desventajas de los métodos.
- **Capítulo 3**: Se presenta el marco teórico con el cual se fundamente el presente trabajo: desde la neurona hasta las arquitecturas *Transformer*.
- **Capítulo 4**:Se describe a detalle el procedimiento para la recolección y etiquetado del conjunto de videos de señas en la LSM, junto con los elementos que lo componen y un análisis exploratorio del mismo.
- **Capítulo 5**: Se detalla la arquitectura Video Swin Transformer con la cual se llevará a cabo el reconocimiento de señas en la LSM.
- **Capítulo 6**:Se muestran los resultados obtenidos con el modelo entrenado y evaluado en el conjunto de videos recopilado.
- **Capítulo 7**: Se exponen las conclusiones y direcciones para trabajo futuro.

Capítulo 2

Estado del arte

En este capítulo se abordan los métodos propuestos en la literatura para el reconocimiento de una lengua de señas a un lenguaje hablado.

2.1. El problema de el reconocimiento

Los principales métodos de traducción de lengua de señas a lenguaje hablado surgen a inicios de los años 1980s. No obstante, el desarrollo de esta área de traducción no ha sido tan acelerada como el de otras áreas del lenguaje, como el reconocimiento entre dos idiomas hablados. Esto debido a que, en comparación con el lenguaje hablado, sólo un porcentaje pequeño de la población usa lengua de señas y por lo mismo no haber datos como los hay para la traducción escrita o hablada.

Al ser la comunicación por medio del uso de mano no es posible el uso de métodos de traducción usados en lenguaje hablado o escrito, ya que dentro de las Lenguas de Señas se tienen rasgos manuales (la forma en que se seña) y rasgos no manuales (la manera en que se seña). Estos últimos resultan un gran desafío al no poder identificar los rasgos no manuales de una persona mientras seña.

En los últimos años se han recopilado conjuntos de datos para poder entrenar diversos sistemas para el reconocimiento automático de la LSM, una selección de éstos se puede observar en la Tabla 2.1, la mayoría se consideran artificiales, ya que son grabados en un ambiente controlado en vez de tomarlos del uso cotidiano, por lo que al intentar ser usados como sistemas en un ambiente más realista no se obtienen los resultados esperados. También se observa que un conjunto pequeño de personas son las que realizan las señas, siendo el más significativo BSL Corpus, con 249 señantes para 5,000 señas. Esto causa que no se logren captar diferencias entre las formas de señar una palabra, donde la seña puede empezar y/o terminar en un lugar distinto dependiendo de cada señante, de la misma forma que dos hablantes no entonan de la misma manera las palabras a pesar de haber aprendido en un entorno similar.

Estos conjuntos de datos han sido utilizados para entrenar y evaluar modelos de reconocimiento de señas y traducción de lengua de señas a lenguaje hablado. La gran mayoría se basa en glosar las señas requeridas (en caso de la identificación de oraciones); por ejemplo, si las señas corresponden a las señas “HOMBRE TIEMPO 2 AÑOS

Tabla 2.1: Comparaciones de diferentes conjuntos con sus respectivos modelos. Los conjuntos corresponden a las Lenguas de señas Americana (ASL), Británica (BSL), China (CSL), Finlandesa (FinSL), Alemana (DGS). En la tabla se muestra el número de señas dentro del conjunto de datos, si el conjunto incluye la co-articulación (si se realiza más de una seña en el video), el número de anotaciones por seña (las características de cada interpretación) y la manera en que se obtuvieron los datos (laboratorio, televisión, internet o por medio de lexicones). La tabla se obtiene de BSL-1K[5]

Nombre del conjunto	Lengua	co-articulación	#señas	#anotaciones (promedio por seña)	#señantes	fuente de los videos
ASLLVD [6]	ASL	No	2742	9K (3)	6	lab
Devisign [17]	CSL	No	2000	24K (12)	8	lab
MSASL [27]	ASL	No	1000	25K (25)	222	lexicones, web
WLASL [32]	ASL	No	2000	21K (11)	119	lexicones, web
S-pot [54]	FinSL	Sí	1211	4K (3)	5	lab
Purdue RVL-SLLL [56]	ASL	Sí	104	2K (19)	14	lab
Video-based CSL [26]	CSL	Sí	178	25K (140)	50	lab
SIGNUM [55]	DGS	Sí	455	3K (7)	25	lab
RWTH-Phoenix [31, 11]	DGS	Sí	1081	65K (60)	9	TV
BSL Corpus [48]	BSL	Sí	5K	50K (10)	249	lab
BSL-1k [5]	BSL	Sí	1064	273K (257)	40	TV

MORIR" se considera su interpretación como "El hombre lleva dos años muerto".

Respecto a los métodos usados para traducción de lengua de señas a lenguaje hablado, en las siguientes secciones se mencionarán desde los más básicos, como el uso de sensores dentro de un guante para reconocer señas, hasta los métodos más recientes como los basados en Redes Convolucionales Gráficas o arquitecturas Transformer para el reconocimiento de palabras u oraciones. Estos métodos han sido aplicados en ambientes controlados (grabados en un lugar sin ruido) y no controlados (tomados de algún programa de televisión, noticias, clima, etc.).

2.2. Guante como método de traducción

De los primeros métodos explorados para el reconocimiento de señas se tiene el uso de guantes en una o ambas manos de la persona que realiza las señas [37]. Se puede considerar algo intuitivo, al seguir la línea de pensamiento de si se usan las manos para hablar, entonces al saber el movimiento de las manos se puede saber qué se está señando. De esta clase de métodos se tiene una de las primeras patentes, realizada a finales de 1983, la que se puede ver en la Figura 2.1. La finalidad de este guante era la de reconocer de la Lengua de Señas Americana (ASL) a partir de las señales transmitidas del guante a una computadora que, en base a los movimientos recibidos, indica la seña que fue realizada [10, 4].

Posteriormente se realiza el reconocimiento de las señales recibidas del guante, al activarse sensores dependiendo de los movimientos de la mano y/o dedos. Los primeros trabajos que emplearon esta clase de métodos únicamente tomaban en cuenta el plano 2D (movimiento hacia arriba/abajo e izquierda/derecha) mientras que trabajos más recientes se han enfocado en el movimiento 3D a partir de un giróscopo que calcula la profundidad de la seña desde la persona señante. Para obtener la seña en base a las

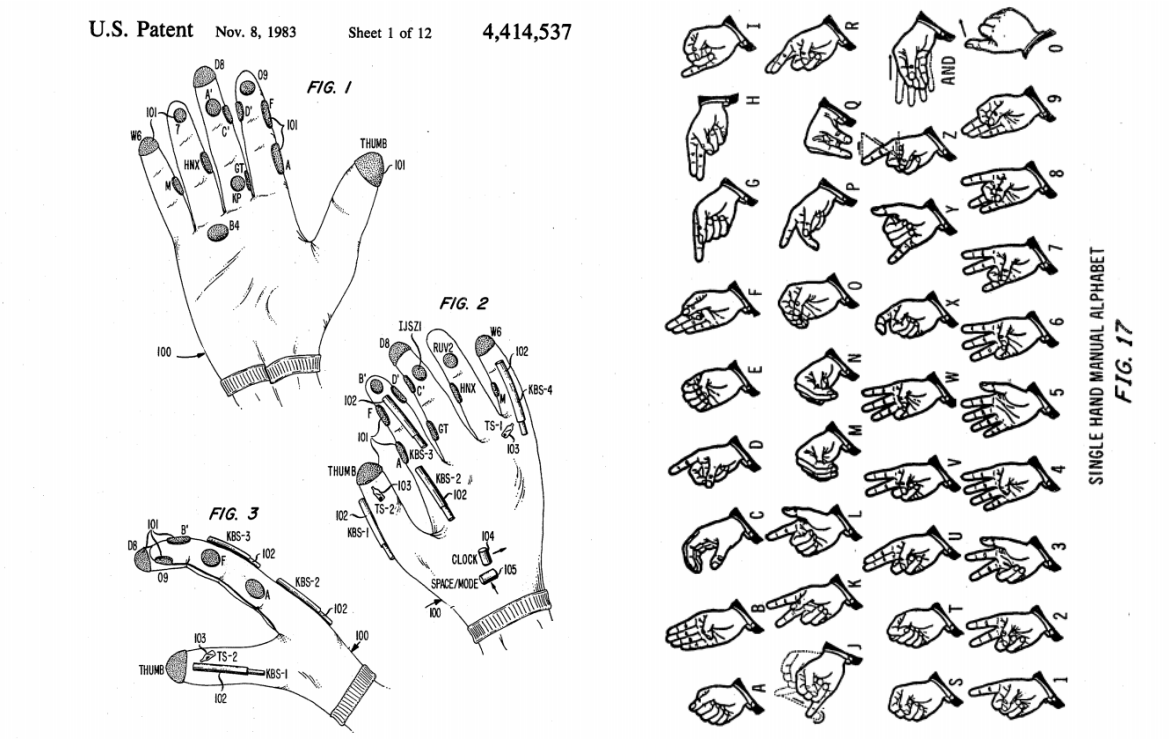


Figura 2.1: Patente introducida en 1983 por Gary Grimes junto al alfabeto capaz de identificar. Imagen tomada de [24].

señales recibidas se utilizan algoritmos de aprendizaje como árboles de decisión, redes neuronales, métodos de agrupación o modelos ocultos de Márkov.

A pesar de poder seguir los movimientos de la mano con exactitud, esta clase de métodos es rechazada por las comunidades que utilizan lengua de señas, dado que obligan al señante a utilizar un guante para poder comunicarse, además de no tomar en cuenta las normas de la lengua al realizar el reconocimiento.

2.3. Trayectoria de las manos

Esta clase de métodos se enfoca en el uso de dispositivos no invasivos y portátiles para obtener los movimientos de las manos, donde unos se enfocan en la mayor granularidad posible (la posición de los dedos de cada mano) y algunos métodos únicamente en seguir el movimiento de la mano. En esta clase de métodos se utiliza visión computacional para detectar el contraste de la mano contra el fondo y poder seguir su movimiento[50]. El efecto del contraste para poder ubicar la imagen se ve en la Figura 2.3, donde posteriormente se utilizan redes convolucionales para identificar las manos y seguir su trayectoria[58], lo que se puede ver en la Figura 2.4. En estos métodos la entrada es una imagen o video de una persona realizando una seña, generalmente con un fondo blanco para el contraste, a partir del cual se obtiene la secuencia de movimientos que alimenta a un algoritmo de clasificación como K-vecinos cercanos para identificar la seña realizada.[45]

Figura 2.2: José Hernández Rebollar muestra el guante que diseñó, llamado AcceleGlove, el cual promete "reconocer" lengua de señas a lenguaje hablado. (Stephen J. Boitano / AP) [23]

(a) Imagen original.

(b) Imagen después de aplicar filtros de contraste.

Figura 2.3: Contraste entre colores para obtener la silueta de la mano y eliminar el fondo, imagen tomada de [9].

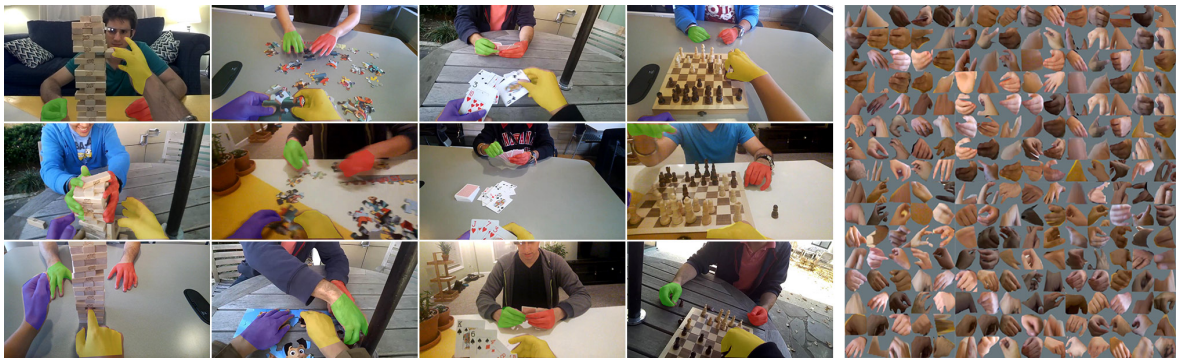


Figura 2.4: Método de identificación de manos por medio de redes neuronales convolucionales, imagen tomada de [8].

2.3.1. Siguiendo el centro aproximado de la mano

Para enfocarse en un único punto de la mano y facilitar el cálculo de la trayectoria y la clasificación, algunos métodos han empleado una caja que rodea a la mano para ubicar su centro en coordenadas (x, y) al calcular la trayectoria que va tomando la mano al realizar la seña. Esto se puede observar en la Figura 2.5. Estos métodos, si bien son sencillos, no logran identificar con tanta granularidad el movimiento de las manos por lo que no son tan precisos al momento de diferenciar distintas señas si éstas comparten movimientos en común, dado que es necesario saber la posición de la mano y dedos para poder diferenciar.

2.3.2. Utilizando el contraste de la mano para obtener diversas características de la mano

Para extraer más características de las manos y diferenciar entre señas que comparten movimientos y difieren en la posición de los dedos se utiliza el contraste de la mano para identificar qué dedos se han movido durante la seña. Un método de extracción de características de este tipo se muestra en la Figura 2.6.

Este método retoma al procedimiento que utilizaba guantes, pero tomando las características a utilizar por medio de videos o múltiples imágenes.

2.4. Pose de la persona

Conforme progresaba el campo de Inteligencia Artificial, surge una clase de métodos que es más robusta a las demás, la cual no sólo busca observar lo que hacen las manos, sino también lo que hace el cuerpo completo, ayudando así a captar toda la información relativa a la seña. Estas características son rasgos no manuales que dan "vida" a las señas, análogo a como dos oraciones pueden tener las mismas palabras, pero diferentes sentimientos (alegría, tristeza, enojo, etc.) [5, 52].

Para obtener la pose de una persona, se utilizan múltiples imágenes o videos de personas realizando señas para alimentar a distintos sistemas, como *OpenPose* [15],

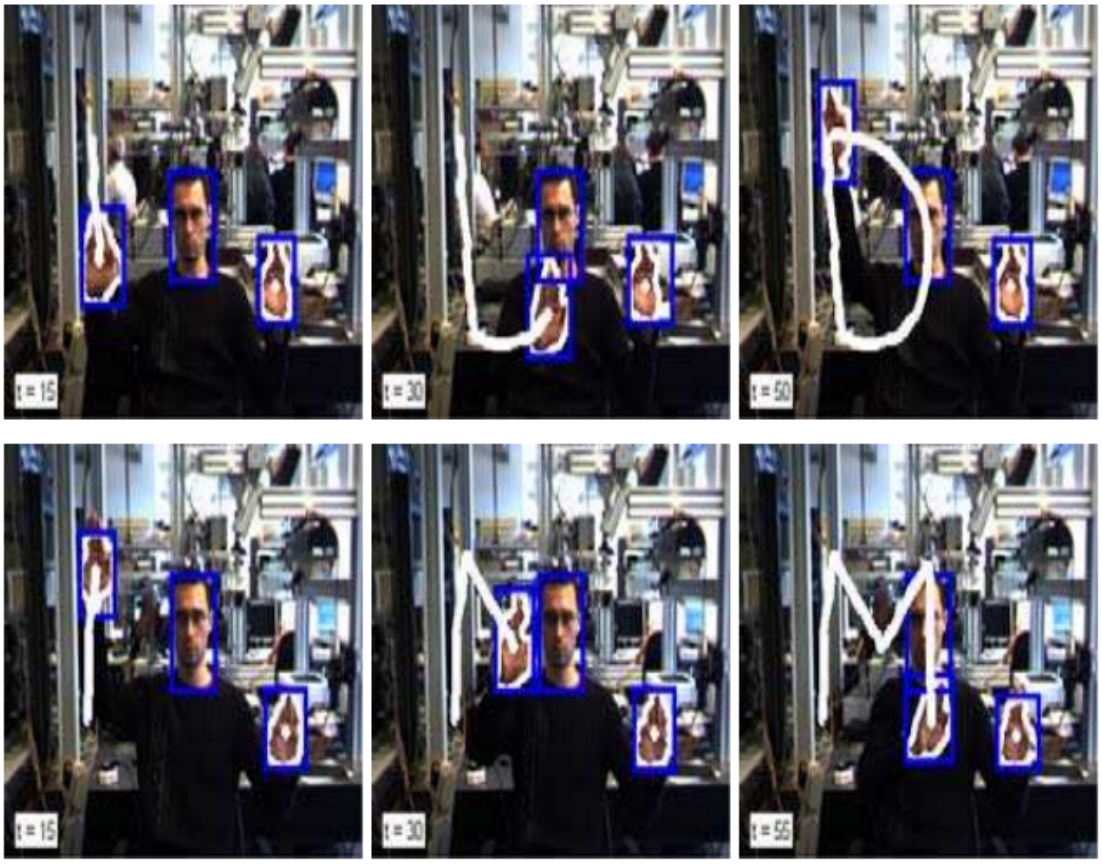
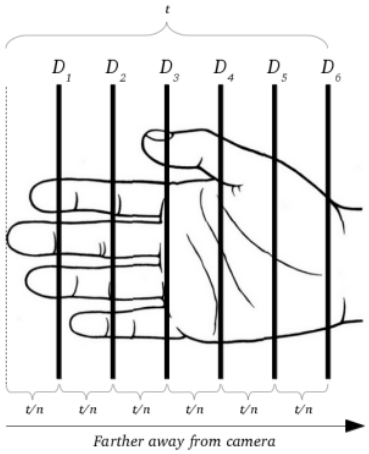


Figura 2.5: Trayectoria de la mano a partir del centro de la caja que la rodea, imagen tomada de [22].

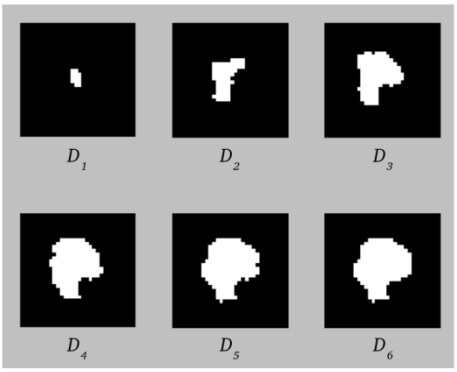
DCPose [36], *HRNet* [51], etc. La forma en que Openpose muestra la pose calculada de dos personas se ve en la Figura 2.7.

Al obtener más características que otros métodos se pueden analizar distintos aspectos de la seña, por ejemplo: los movimientos que hacen los brazos y manos, movimientos del rostro como fruncimiento de seña, sonrisa, etc., vocalización de la seña que se está realizando [13, 29]. Al tener tanta granularidad se necesita que los métodos de aprendizaje sean más complejos, porque se puede usar toda la pose observada y utilizar completamente todas las características obtenidas de la pose, pero al ser la mayoría de la pose despreciable se enfoca principalmente en la pose de las partes utilizadas para la comunicación (manos, brazos, cara). Para poder aprovechar estas características se utilizan tecnologías más recientes como las arquitecturas Transformer [53, 59, 14, 39, 20]. Un ejemplo de un sistema de traducción de lengua de señas basado en una red neuronal Transformer se puede observar en la Figura 2.8.

El proceso de este método es alimentar los videos o imágenes a un sistema de detección de pose, separar las características conforme se desea para alimentarlas a una red neuronal Transformer.



(a) Capas usadas para descomponer la mano.



(b) Imágenes que muestran la profundidad binaria , se observa cómo va cambiando la imagen conforme más dedos tienen movimiento.

Figura 2.6: Método que utiliza la profundidad de las imágenes para identificar los movimientos más finos de la mano, imagen tomada de [46].

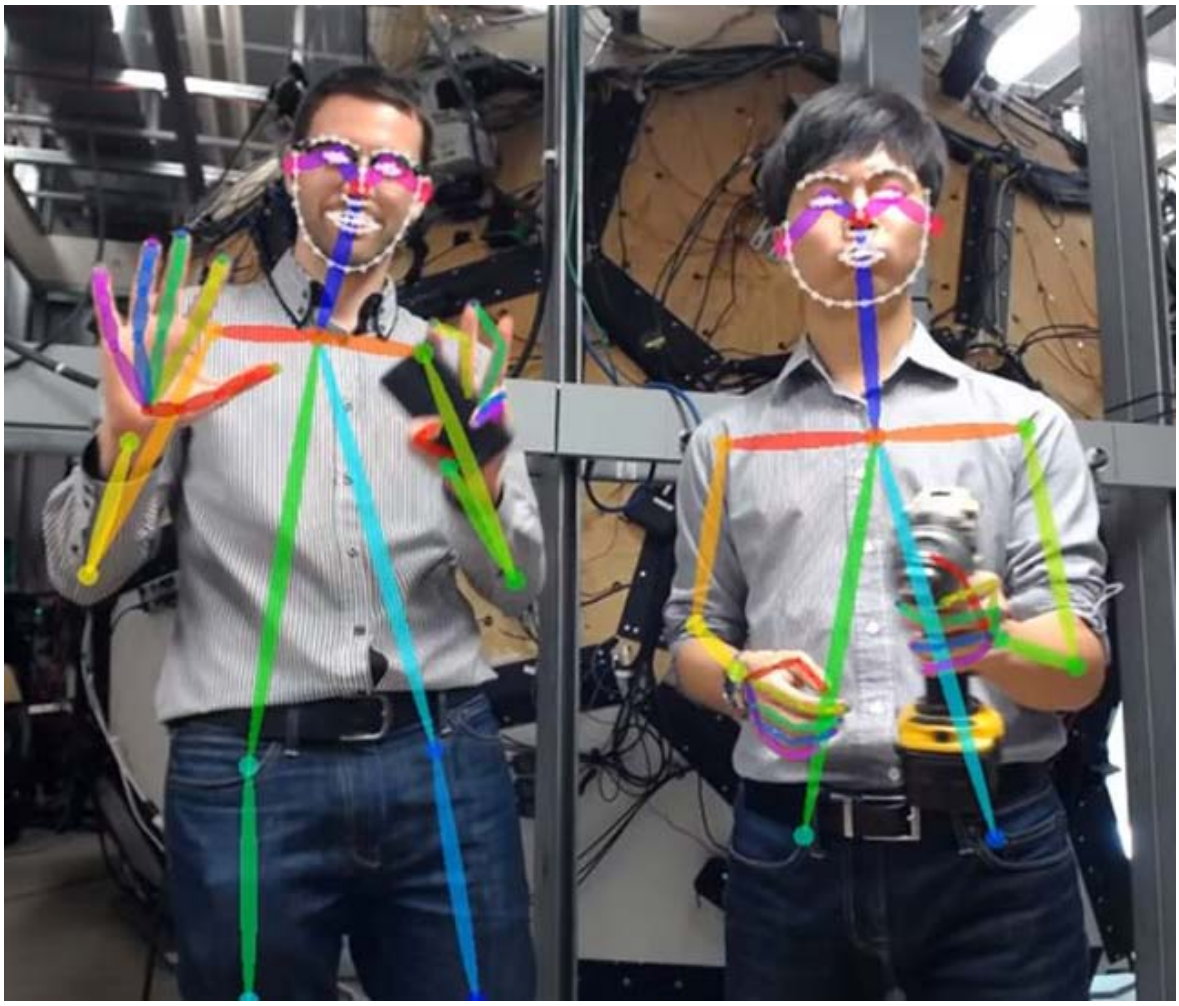


Figura 2.7: Pose de varias personas obtenida utilizando OpenPose, se observa cómo dibuja un esqueleto y la silueta del rostro, imagen tomada de [15].

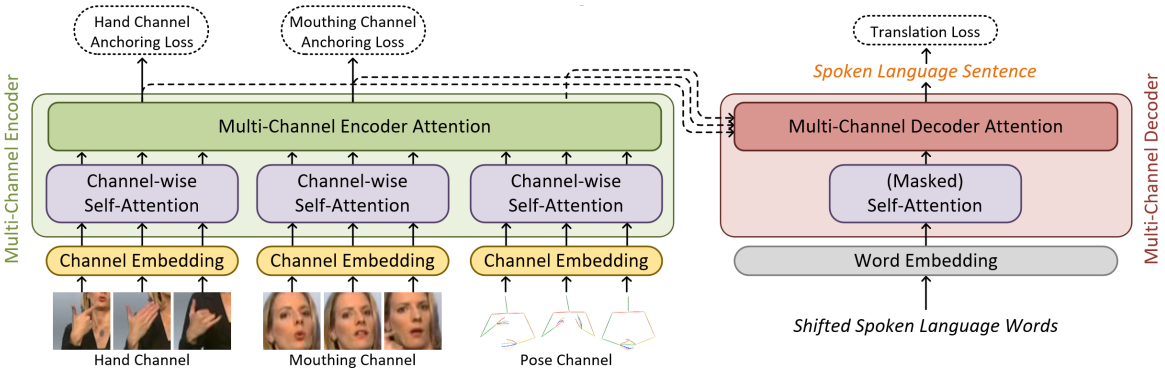


Figura 2.8: Método que utiliza una red neuronal Transformer y pose de una persona para el reconocimiento de señas. En particular se observa que se tienen tres distintos canales para lograr la tarea: Canal de Mano, Canal de Boca (en caso de vocalizar la palabra) y Canal de Pose, imagen tomada de [12].

Capítulo 3

Marco teórico

3.1. Neurona artificial

Dentro del área de las redes neuronales, el primer concepto a introducir es la **neurona artificial** o **perceptrón**, la cual está compuesta de un vector de entradas $\mathbf{x} \in [0,1]^d$ y un vector de pesos $\mathbf{w} \in \mathbb{R}^d$, donde a cada entrada le corresponde un único peso

La neurona se define de la siguiente manera:

$$Neurona\left(\mathbf{x},\mathbf{w}\right) = \varphi\left(\sum_{i=0}^{k} w_i x_i\right); w_i \in \mathcal{W}, x_i \in \mathcal{X}. \tag{3.1}$$

La neurona artificial se puede ver en la Figura 3.1

Dentro del perceptrón se tiene una función de activación φ que transforma de manera no lineal la suma pesada y representa la salida de la neurona. Esta transformación se realiza hacia un rango predefinido. Dentro de las diversas funciones de activación, la función escalón unitario fue una de las primeras funciones usadas, dada su simplicidad y facilidad de cálculo, La función escalón unitario se define como:

$$\varphi\left(x\right) = \begin{cases} 0 & \text{si } x \leq 0\ , \\ 1 & \text{si } x > 0. \end{cases} \tag{3.2}$$

Como se puede observar, la función escalón unitario mapea los valores de la suma pesada a 0 o 1.

Conforme se van utilizando las neuronas artificiales se empieza a querer representar funciones más complejas, siendo la función piso muy sencilla dado a que su rango de salida es binario y un cambio muy bajo de pesos y/o valores puede cambiar completamente la salida de la función, que es algo que se busca evitar al momento de ajustar finamente los pesos. Para resolver este problema, se empezaron a utilizar otras funciones de activación, entre ellas la función sigmoide, la cual está dada por:

$$\varphi\left(x\right) = \frac{1}{1 + exp\left(-\sum_j \boldsymbol{w}_j \boldsymbol{x}_j\right)}. \tag{3.3}$$

No obstante, una sola neurona es incapaz de representar funciones más complejas como lo mostraron Minsky y Papert [38]. Para resolver esto, se construye una estructura de

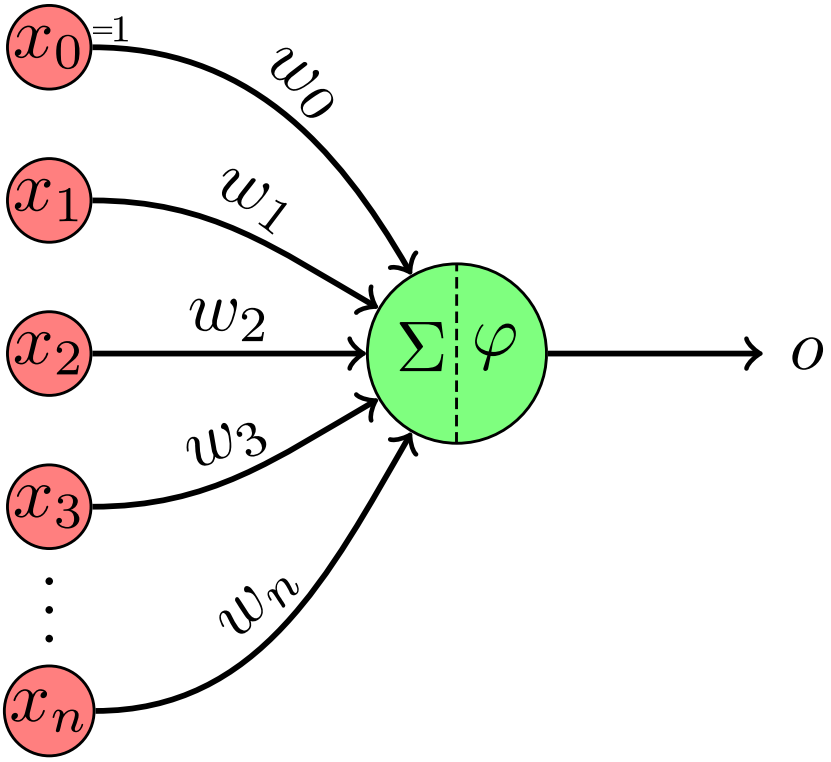


Figura 3.1: Representación granular de una neurona, imagen tomada de [18].

neuronas, donde un conjunto de neuronas se vuelve la entrada de otro conjunto de neuronas hasta obtener una salida, esta estructura se llama **red neuronal**, con cada neurona teniendo sus correspondientes entradas y pesos.

3.2. Redes neuronales

En general, una red neuronal se divide por tres tipos de **capas**. La primer capa de la red, llamada **capa de entrada**, es donde se encuentran las neuronas que reciben al vector de entradas. Después se encuentran capas intermedias, llamadas **capas ocultas**, cuyas neuronas reciben la salida de la capa de entrada y envían su salida hacia la siguiente capa, la cual puede ser otra capa oculta o capa de salida. Finalmente, se encuentra la última capa, llamada **capa de salida**, la cual recibe la salida de la última capa oculta para calcular la salida de la red en base a los valores obtenidos por las demás capas. Además de los pesos, cada neurona dentro de una capa de la red tiene un parámetro llamado **sesgo**, que traslada la suma pesada de las entradas por los pesos. Un ejemplo de una red neuronal sencilla se puede observar en la Figura 3.2

3.3. Entrenamiento de la red

Conforme aumenta la complejidad de las funciones a aproximar, empieza a resultar más complicado el ajustar sus valores, esto dado a que cada perceptrón tiene sus pesos y

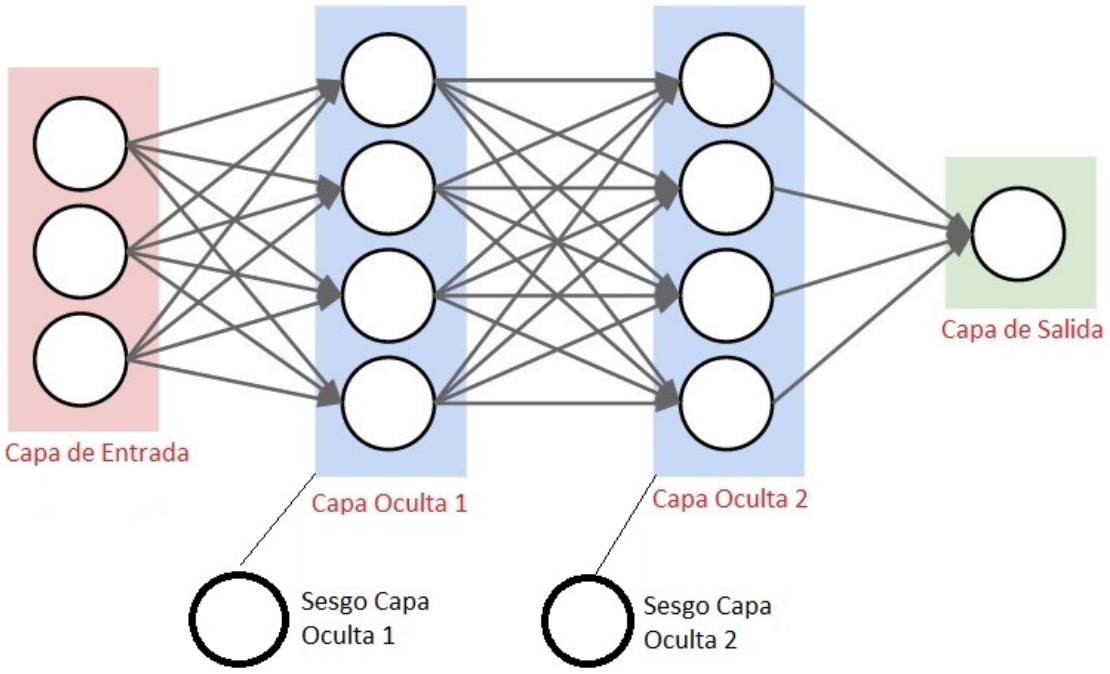


Figura 3.2: Diagrama de una red neuronal con dos capas ocultas, imagen tomada de [40].

sesgos asociados. Para poder ajustar estos valores se necesita poder comparar la salida obtenida de la salida deseada para obtener los ajustes a realizar sobre estos valores y así obtener la salida deseada. Para poder definir este proceso, se definirán varios conceptos listados a continuación.

3.3.1. Inicialización

Lo primero que se realiza es asignar valores a los pesos y sesgos dentro de la red, esto se realiza por medio de una Distribución Gaussiana de media cero con desviación estándar de 0,01. En base a estos valores se calculará la salida para posteriormente ir ajustando los valores iniciales.

3.3.2. Función de error

La función de error se encarga de calcular la diferencia entre la respuesta obtenida de la red ($\hat{y}$) y la salida esperada de la red (y) dada una entrada (x). La función de error Error Cuadrático Medio (*Mean Squared Error*) se define de la siguiente forma:

$$C\left(\boldsymbol{w}, \boldsymbol{b}\right) \equiv \frac{1}{n} \sum_{i=1}^{n} \left(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\right)^2, \tag{3.4}$$

con $w \in \mathcal{W}$, $b \in \mathcal{B}$, $n = \|\mathcal{X}\|$.

Esta función permite calcular el desempeño de la red respecto a los datos de entrenamiento y verificar si la red está obteniendo las salidas esperadas. Al ser una función de error se espera que haya un error mínimo, lo cual muestra que hay similitud entre la respuesta obtenida y la respuesta esperada. No obstante, el tener un error bajo sobre el conjunto de entrenamiento no necesariamente indica que la red no requiere ajustes, dado que puede llegar a haber problemas de **sobreajuste**, que ocurre cuando la red "aprende" el conjunto de entrenamiento, pero al ver elementos nuevos es incapaz de reproducir el desempeño que tuvo al ser entrenada.

3.4. Descenso por el gradiente

Para disminuir el error se tienen que ajustar los parámetros, esto por medio del algoritmo **Descenso por el gradiente**, el cual utiliza el negativo del gradiente de la función de error para obtener la dirección de mayor disminución respecto a $\boldsymbol{w}$ y $\boldsymbol{b}$ y así lograr modificar los valores hacia un menor error. La manera en que se realizan los cálculos se puede ver en las siguientes ecuaciones.

$$\boldsymbol{w}_k' = \boldsymbol{w}_k - \eta \frac{\partial C}{\partial \boldsymbol{w}_k}, \tag{3.5}$$

$$\boldsymbol{b}_l' = \boldsymbol{b}_l - \eta \frac{\partial C}{\partial \boldsymbol{b}_l}, \tag{3.6}$$

con $\boldsymbol{w}_k \in \mathcal{W}$ y $\boldsymbol{w}_l \in \mathcal{B}$.

Para evitar cambios bruscos del error, se utiliza el hiperparámetro η, el cual representa la tasa de aprendizaje y controla qué tanto se modifican los parámetros dados los gradientes.

3.5. Propagación hacia atrás

Al entrenar una red neuronal se deben actualizar los parámetros de la capa de salida y de las N capas ocultas. Debido a que la salida de una capa oculta posteriormente se procesa por una o más capas, para calcular el gradiente de la función de error respecto a que un peso o sesgo específico de dicha capa pueda involucrar una larga cadena de derivadas parciales. Para hacer el cálculo de forma más eficiente, se introduce al algoritmo de **propagación hacia atrás**[28]. Antes de definir el algoritmo a usar, se definirán notaciones que facilitan el entendimiento del algoritmo.

Como primer cambio, se utiliza el superíndice l para indicar que el elemento pertenece a la capa l, lo cual nos define la siguiente ecuación:

$$a_j^l = \varphi\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l,\right), \tag{3.7}$$

donde $\boldsymbol{w}_{jk}^l \in \boldsymbol{W}^l$ representa los pesos de las conexiones hacia la neurona j desde la neurona k y $b \in \boldsymbol{b}_{\boldsymbol{j}}^l$ representa los sesgos de la capa l. Esta definición nos ayuda a

simplificar el cálculo de la activación de una neurona dada la capa anterior. Las salidas de toda la capa se pueden expresar en notación matricial como:

$$\boldsymbol{a}^l = \varphi\left(\boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right), \tag{3.8}$$

Con esto podemos definir la suma ponderada de la capa l de la siguiente manera: donde

$$\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l. \tag{3.9}$$

Finalmente, sustituyendo las ecuaciones, definimos la activación de la capa l de la siguiente manera:

$$\boldsymbol{a}^l = \varphi\left(\boldsymbol{z}^l\right). \tag{3.10}$$

El algoritmo de propagación hacia atrás se compone de dos etapas. La primera etapa consiste en propagar las entradas capa por capa hasta obtener las salidas correspondientes. La segunda etapa consiste en ir actualizando los pesos de la red en base al negativo del vector gradiente de la función de error, el cual se va calculando al ir propagando hacia atrás los errores de la red, por medio de las ecuaciones 3.5 y 3.6.

Teniendo esto, deseamos calcular el gradiente de la neurona n, denotado por $\frac{\partial C_n}{\partial \boldsymbol{w}_{ji}}$, el cual depende de los resultados de las $n-1$ neuronas anteriores. Siguiendo esta línea de pensamiento, para obtener el resultado de C_n se aplica la regla de la cadena para tomar en cuenta los resultados de las $n-1$ anteriores, dándonos la siguiente ecuación:

$$\frac{\partial C_n}{\partial \boldsymbol{w}_{ji}} = \frac{\partial C_{n-1}}{\boldsymbol{a}_j} \frac{\partial \boldsymbol{a}_j}{\partial \boldsymbol{w}_{ji}}. \tag{3.11}$$

Para seguir con el algoritmo, se introduce una nueva notación que se refiere al error de la neurona j en la capa l, denotada por $\delta \boldsymbol{a}_j^l$, la cual se puede ver en la ecuación:

$$\delta \boldsymbol{a}_j^l \equiv \frac{\partial C_n}{\partial \boldsymbol{a}_j^l}. \tag{3.12}$$

Posteriormente sustituimos en la ecuación 3.9 y nos da lo siguiente:

$$\frac{\partial \boldsymbol{a}_j}{\partial \boldsymbol{w}_{ji}} = \boldsymbol{z}_i. \tag{3.13}$$

Sustituyendo las ecuaciones 3.12 y 3.13 en la ecuación 3.11 obtenemos:

$$\frac{\partial C_n}{\partial \boldsymbol{w}_{ji}} = \delta \boldsymbol{a}_j \boldsymbol{z}_i. \tag{3.14}$$

Lo que la ecuación nos dice es que el gradiente se obtiene al multiplicar el error de la capa j por la activación de la capa i, facilitando en gran medida los cálculos de los errores de cada capa.

Posteriormente se calculan los errores de las neuronas, lo que se define de la siguiente manera:

$$\delta \boldsymbol{a}_j = \boldsymbol{y}_j - \hat{\boldsymbol{y}}_j, \tag{3.15}$$

con $\boldsymbol{y}_j$ siendo la salida esperada y $\hat{\boldsymbol{y}}_j$ la salida obtenida, ambos en la neurona j.

Para obtener los errores δ de las capas ocultas, se utiliza la regla de la cadena de la siguiente manera:

$$\delta \boldsymbol{a}_j \equiv \frac{\partial C_n}{\partial \boldsymbol{a}_j} = \sum_k \frac{\partial C_n}{\partial \boldsymbol{a}_k} \frac{\partial \boldsymbol{a}_k}{\partial \boldsymbol{a}_j}, \tag{3.16}$$

donde se suma a través de las k neuronas donde cada una tiene j conexiones.

Al sustituir la ecuación 3.12 en la ecuación 3.16 y usando la ecuación 3.9, obtenemos la siguiente fórmula conocida como la fórmula de propagación hacia atrás:

$$\delta \boldsymbol{a}_j = \frac{\partial C}{\partial \boldsymbol{a}_j^l} \varphi' \left(z_j^l \right). \tag{3.17}$$

Aquí se observa que la ecuación es el gradiente de la función de error escrita en términos de los errores de la red, los cuales son propagados hacia atrás.

Una vez obtenidos los errores, se aplica recursivamente la ecuación 3.17 para evaluar los errores en las capas faltantes, esto sin importar el tipo de topología que tenga la red.

El algoritmo de propagación hacia atrás puede resumirse de la siguiente manera:

- Se alimenta un vector a la red, propagándose las activaciones hacia adelante hasta la capa de salida.
- Se evalúan los errores para la capa de salida usando la ecuación 3.15.
- Una vez obtenidos los errores, se propagan hacia atrás usando la ecuación 3.17, esto para cada neurona en la capa oculta.
- Se utiliza la ecuación 3.14 para calcular las derivadas parciales requeridas, lo que agiliza el cálculo.
- Se actualizan los pesos y sesgos utilizando las ecuaciones 3.5 y 3.6.

Si bien la propagación hacia atrás junto con descenso por el gradiente logra entrenar la red, el proceso puede llegar a ser lento ya que conforme incrementa el tamaño de la red, junto al tamaño del conjunto a alimentar, el tiempo para realizar una actualización a los pesos y sesgos puede ser considerable.

3.6. Descenso por el gradiente estocástico

Los pesos y sesgos de las neuronas llegan a tener dificultades respecto al tiempo empleado. Por lo que dependiendo del número de actualizaciones que se desea tener, se puede llegar a tardar mucho tiempo. Para poder combatir esto, se introduce el **Descenso por el gradiente estocástico** (SGD).

En el descenso por gradiente estocástico se aproxima el gradiente usando un subconjunto (llamado minilote) del conjunto original que se alimenta y realiza el algoritmo

de descenso por gradiente sobre el minilote en vez de tomar el subconjunto completo. La forma en que se calcula el SGD se puede observar en la siguiente ecuación:

$$\nabla C\left(\mathcal{X}\right) \approx \frac{1}{m}\sum_{j=1}^{m}\nabla C_{\boldsymbol{x}_j}, \tag{3.18}$$

con $\boldsymbol{x} \in \mathcal{X}$ y $\bigcup_{i=1}^{m} \boldsymbol{x}_i = \mathcal{X}$.

Por medio de estas aproximaciones, podemos cambiar las reglas de actualización previamente vistas por las siguientes:

$$\boldsymbol{w}_k \to w_k' = \boldsymbol{w}_k - \frac{\eta}{m}\sum_j \frac{\partial C_{\boldsymbol{x}_j}}{\partial \boldsymbol{w}_k}, \tag{3.19}$$

$$\boldsymbol{b}_l \to b_l' = \boldsymbol{b}_l - \frac{\eta}{m}\sum_j \frac{\partial C_{\boldsymbol{x}_j}}{\partial \boldsymbol{b}_l}. \tag{3.20}$$

Este proceso se repite hasta recorrer todos los subconjuntos producidos, donde se dirá que se completó una época de entrenamiento.

3.7. Redes neuronales convolucionales (CNN)

Existen diversos tipos de redes neuronales, en esta sección describiremos las **redes convolucionales**[43], las cuales son útiles para nuestra tarea. Las CNN se enfocan en el procesamiento de datos con estructura de rejilla, como las imágenes, siendo su función la extracción de características para resolver tareas.

Como su nombre lo indica, estas redes utilizan la operación de convolución, definida de la siguiente forma en el ámbito de procesamiento digital de imágenes:

$$O\left(i,j\right) = \sum_{k=1}^{m}\sum_{l=1}^{n} I\left(i+k-1, j+l-1\right)\boldsymbol{K}\left(k,l\right), \tag{3.21}$$

donde $\boldsymbol{I}$ representa una imagen vista como matriz y $\boldsymbol{K}$ un *kernel* a aplicar sobre la imagen.

Como la definición anterior de convolución aplica sobre imágenes, se deben de ajustar las redes neuronales para poder procesar las imágenes, tomando en cuenta que la información espacial es algo importante que recordar. Por ello las redes convolucionales tienen conexiones con una estructura tipo rejilla, como se observa en la Figura 3.3. Además, las redes convolucionales tienen tres componentes importantes para procesar eficientemente este tipo de datos: **campos receptivos locales** , **capas de agrupación**, **pesos compartidos**.

3.7.1. Campos receptivos locales

A diferencia de las redes anteriores, las CNN no son completamente conectadas, es decir, no todas las neuronas de la capa l se conectan con todas las neuronas de la capa $l+1$. Esto debido a que se hacen conexiones de tamaño $n \times n$, llamadas ventanas

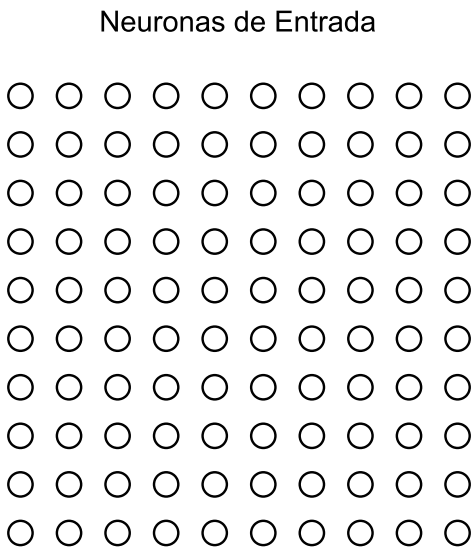


Figura 3.3: Representación de la entrada de una CNN, imagen tomada de [41].

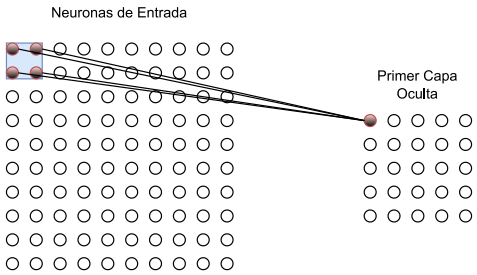


Figura 3.4: Se observa como un cuadrado de neuronas en la capa de entrada se conecta a una neurona de la capa oculta, imagen tomada de [41].

o regiones, que se muestran en la Figura 3.4. Esta región se conoce como el **campo receptivo local** de la neurona oculta y se encargará de procesar una característica de la entrada dentro de la ventana.

Una vez que se han procesado los elementos dentro del campo, se empieza a deslizar la ventana hasta cubrir la imagen en su totalidad, donde el movimiento de la ventana (el número de neuronas a desplazar) se define por un parámetro llamado paso, siendo que la ventana se va desplazando paso neuronas hasta haber cubierto toda la entrada. Se puede observar un ejemplo del paso viendo las Figuras 3.4 y 3.5, que corresponde a un paso de tamaño 1.

3.7.2. Pesos y sesgos compartidos

Como cada campo identifica una única característica a lo largo de toda la imagen, únicamente necesita aprender a encontrar dicha característica. Esto se logra al tener un sesgo y $r \times r$ pesos que lo conectan hacia las neuronas de entrada, teniendo que, para la j, k-ésima neurona oculta, su salida se calculará de la siguiente manera:

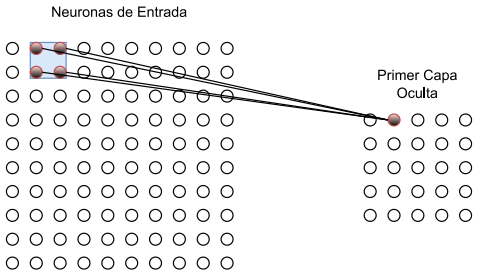


Figura 3.5: El cuadrado se mueve una unidad a la derecha, construyendo la siguiente neurona de la capa oculta. El movimiento puede variar y ser más grande, éste es conocido como paso, imagen tomada de [41]

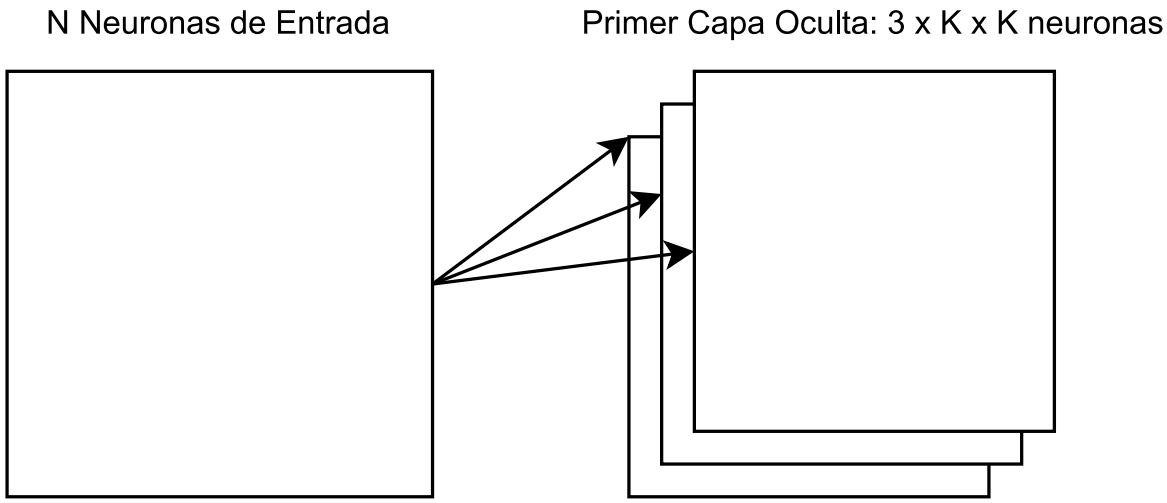


Figura 3.6: La capa de entrada posee varios mapeos hacia la primera capa oculta, donde cada uno es capaz de detectar una característica diferente en la imagen de entrada, esto se logra por medio de los pesos y sesgos compartidos, imagen tomada de [41].

$$\varphi\left(b+\sum_{l=0}^{z}\sum_{m=0}^{z}\boldsymbol{w}_{lm}\boldsymbol{a}_{j+l,k+m}\right). \tag{3.22}$$

Por ello se nombra al mapeo de la capa de entrada a la capa oculta como un *mapa de características convolucionales*, llamándose a los pesos que definen al mapa de características pesos compartidos y a los sesgos que lo definen como sesgos compartidos.

Cuando se desea detectar más de una característica, se generan más mapas de características (i.e más capas ocultas), donde cada uno tiene su matriz de pesos y sesgos de acuerdo a la característica a detectar. En la Figura 3.6 se observa que se desean detectar tres características en una imagen, por lo que se tienen tres distintos mapeos desde la capa de entrada.

Figura 3.7: En el caso de esta capa de agrupación se toma un paso de 2 unidades, resultando en que el mapa de características de 5×5 se convierta en un mapa de 3×3, imagen tomada de [3].

3.7.3. Capas de agrupación

Además de las capas convolucionales, las redes convolucionales también contienen unas capas llamadas capas de agrupación, éstas suelen ir después de las convolucionales siendo su función el simplificar o agrupar la información que da de salida la capa convolucional. En otras palabras, una capa de agrupación toma cada mapa de características y lo condensa en un mapa de características más pequeño. La forma más común es conocida como *max pooling*.

3.7.4. *Max pooling*

Para generar una capa de agrupación se recorre al mapa de características en un bloque de $k \times k$, tomando la salida de cada bloque y agarrando el valor máximo dentro del bloque. Se recorre todo el mapa de características para terminar con una imagen de tamaño $\frac{n}{k} \times \frac{n}{k}$, siendo cada elemento el máximo encontrado en los bloques del mapa original. Este procedimiento se puede ver en la Figura 3.7.

Con los elementos mencionados, una red neuronal convolucional se puede ver en la Figura 3.8.

3.8. Atención

Al ir avanzando el desarrollo de métodos y arquitecturas de las redes neuronales, se idearon distintas formas de visualizar los problemas a resolver, ante esto surgió la **atención**. El mecanismo de atención en redes se basa en la atención desde la perspectiva biológica, en particular se identifica que la atención se basa en dos conceptos:

- Señal volitiva: Es el actuar de una persona para enfocarse en algo que comprende.

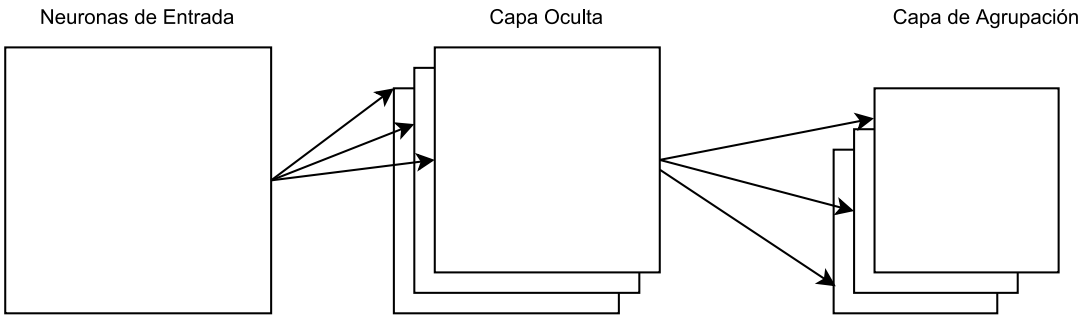


Figura 3.8: Red convolucional completa, al inicio se encuentran las neuronas de entrada, las cuales se conectan a una capa oculta para extraer las características y finalmente se alimenta cada capa oculta a una capa de agrupación para disminuir el tamaño de parámetros. Imagen basada en [41].

- Señal no volitiva: Es la prominencia y notoriedad de los objetos en el entorno.

Bajo esos conceptos, se puede imaginar una arquitectura neuronal que considere ambas señales para lograr clasificar o identificar objetos. Para las señales no volitivas, al únicamente depender del entorno, se puede utilizar una capa completamente conectada al sólo enfocarse en las entradas que se le alimentan. Buscando una manera de introducir las señales volitivas a la red, llamadas **consultas** para la arquitectura. Dada una consulta, los mecanismos de atención sesgan la selección sobre las entradas sensoriales, llamadas **valores** para la arquitectura, a través de la agrupación de la atención. Con esto se observa que a cada valor le corresponde una señal no volitiva, llamada **llave**, lo que se puede pensar como la forma en que el objeto llama su atención dentro del entorno. La forma de diseñar esta arquitectura se conforma, entonces, de una agrupación de atención tal que dada una consulta, ésta pueda interactuar con llave para guiar al sesgo de selección sobre los valores. Una forma visual de cómo funciona el mecanismo de atención [7] se puede ver en la Figura 3.9

Visto de una forma más formal, sea $q \in \mathbb{R}$ una consulta, $k \in \mathbb{R}^d$ una llave de dimensión d y $v \in \mathbb{R}$ un valor, tales que:

$$a\left(q, k, v\right) = \sum_i S\left(q, k_i\right) v_i, \tag{3.23}$$

donde a indica el mecanismo de atención dada una consulta, una llave y un valor, y $S\left(q, k_i\right)$ es la función de similitud entre la consulta y la llave. Se tiene que esta fórmula define al mecanismo de atención más simple y el cual será la base para mecanismos de atención más complejos.

3.8.1. Atención de producto punto escalado

Una forma de calcular la atención es por medio del producto escalar. Pero el uso de esta operación requiere que la consulta y llave tengan el mismo tamaño vectorial d, con esto se asegura que la varianza del producto escalar siempre sea 1, sin importar el tamaño de los vectores, se utiliza la siguiente ecuación:

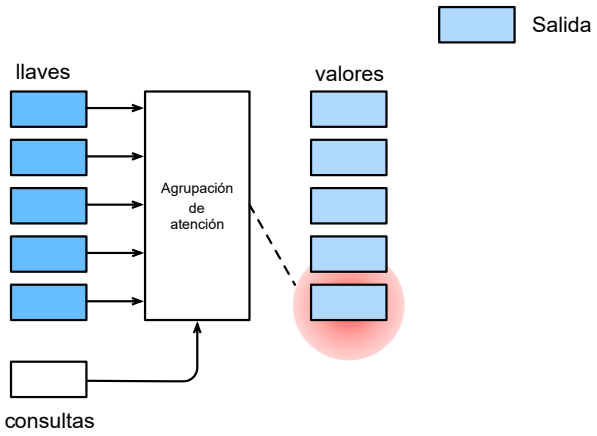


Figura 3.9: Funcionamiento del mecanismo de atención sobre los valores por medio de la agrupación de atención, la cual incorpora consultas y llaves, para finalmente generar una salida. Imagen basada en [60].

$$a\left(\boldsymbol{q}, \boldsymbol{k}\right) = \frac{\boldsymbol{q}^T \boldsymbol{k}}{\sqrt{d}}, \tag{3.24}$$

Teniendo esto, la atención de producto punto escalado se calcula por medio de la siguiente ecuación:

$$\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)\boldsymbol{V} \in \mathbb{R}^{n\times v} \tag{3.25}$$

La atención mostrada calcula únicamente una agrupación de atención dados las matrices $\boldsymbol{Q}, \boldsymbol{K}$ y $\boldsymbol{V}$, pero también se requiere el poder capturar dependencias de corto y largo alcance dada una secuencia, por lo que se busca usar diferentes representaciones de submatrices de $\boldsymbol{Q}, \boldsymbol{K}$ y $\boldsymbol{V}$. Esto es logrado por medio de la **atención multi-cabeza**.

3.8.2. Atención multi-cabeza

La atención multi-cabeza transforma las matrices $\boldsymbol{Q}, \boldsymbol{K}$ y $\boldsymbol{V}$ con h proyecciones lineales independientemente aprendidas. Se conoce a esta atención como multi-cabeza al ser cada agrupación de atención h una cabeza y enfocarse en un distinto subgrupo de las matrices.

De manera más formal, sea $\boldsymbol{q} \in \mathbb{R}^{d_q}$ una consulta, $\boldsymbol{k} \in \mathbb{R}^{d_k}$ una llave y $\boldsymbol{v} \in \mathbb{R}^{d_v}$ una llave, cada cabeza $h_i\,(i = 1, ..., h)$ se calcula de la siguiente manera:

$$h_i = f\left(\boldsymbol{W}_i^{(q)}\boldsymbol{q}, \boldsymbol{W}_i^{(k)}\boldsymbol{k}, \boldsymbol{W}_i^{(v)}\boldsymbol{v}\right), \tag{3.26}$$

donde $\boldsymbol{W}_i^{(q)} \in \mathbb{R}^{p_q\times d_q}, \boldsymbol{W}_i^{(k)} \in \mathbb{R}^{p_k\times d_k}, \boldsymbol{W}_i^{(v)} \in \mathbb{R}^{p_v\times d_v}$ son parámetros aprendidos, y f es la atención de producto punto escalado.

La salida de la atención multi-cabeza es una transformación lineal con parámetros aprendidos $\boldsymbol{W}_o \in \mathbb{R}^{p_o\times hp_v}$, visto de la siguiente manera:

$$\boldsymbol{W}_o \begin{bmatrix} h_i \\ . \\ . \\ . \\ h_h \end{bmatrix} \in \mathbb{R}^{p_o} \tag{3.27}$$

3.8.3. Atención propia

Al tener que $\boldsymbol{q}, \boldsymbol{k}$ y $\boldsymbol{v}$ provienen del mismo lugar, siendo que cada consulta atiende todas las parejas de llave-valor y genera una salida de atención, este proceso se conoce como atención propia. De manera más formal, sean $x_1, ..., x_n$ donde $x_i \in \mathbb{R}^d \, (1 \leq i \leq n)$ los *tokens* de entrada, la atención propia da de salida una secuencia del mismo tamaño $y_1, ..., y_n$ donde:

$$y_i = f\left(x_i, (x_1, x_1), ..., (x_n, x_n)\right) \in \mathbb{R}^d \tag{3.28}$$

Siendo f la atención multi-cabeza.

3.8.4. Codificación posicional

Teniendo que para las secuencias recibidas el orden de cada palabra es importante para procesarlas, se debe de recordar la posición en que se encuentran las palabra, de otra forma se tendría una bolsa de palabras que daría siempre la misma salida. Para esto se utilizan dos ecuaciones que se suman a las incrustaciones de entrada dentro de las secuencias de entrada y salida, por lo que estas codificaciones tendrán la misma dimensión que las incrustaciones, esto para facilitar la suma entre ambos vectores. El cálculo se hace de la siguiente forma:

$$P\left(i, 2_j\right) = \sin\left(\frac{i}{10000^{2_j/d}}\right) \tag{3.29}$$

$$P\left(i, 2_{j+1}\right) = \cos\left(\frac{i}{10000^{2_j/d}}\right), \tag{3.30}$$

con $P\left(i, j\right)$ representando al mapeo de la posición i de la secuencia de entrada al índice (i, j) de la incrustación posicional, p_i representando la posición de la i-ésima palabra dentro de la secuencia y d el tamaño de la incrustación a realizar

3.9. Red Transformer

En un principio la red Transformer se utilizaba para el procesamiento de textos, por lo que su arquitectura inicial se basaba en tener un bloque codificador y decodificador para procesar las entradas y salidas, respectivamente. Pero teniendo el gran potencial de esta red se buscó una manera de adaptarla para procesar imágenes, dándonos la arquitectura en la Figura 3.10, la cual se conoce como ***Vi**sion **T**ransformer* (ViT)[21].

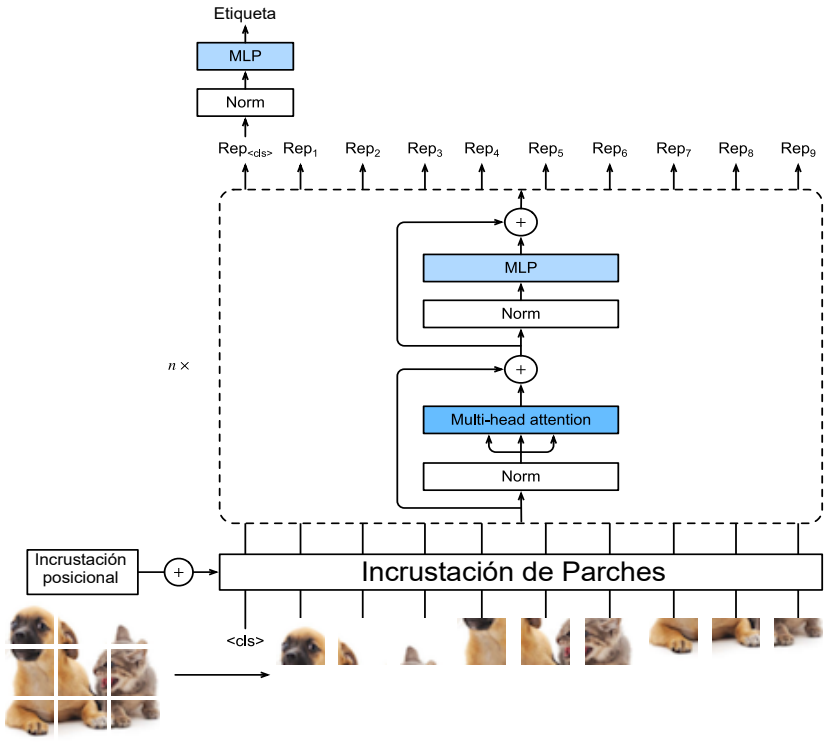


Figura 3.10: Arquitectura de un Transformer de visión o *Vision Transformer* (ViT), donde el bloque Transformer se encuentra dentro de las líneas punteadas. Imagen tomada de [60].

Explicando de forma más profunda la arquitectura ViT, ésta recibe de entrada una imagen de tamaño (h, w, c), siendo h el alto, w el ancho y c el número de canales de la imagen. Esta imagen se parte en una secuencia de $m = hw/p^2$ parches, con (p, p) siendo el alto y ancho del parche, respectivamente, donde cada parche se aplana para tener un vector de longitud cp^2, esto para tener un procesamiento más sencillo. Estos m parches se proyectan, junto a un *token* "<cls>" (clase), a una secuencia de $m + 1$ vectores, junto a la incrustación posicional. Este *token* aplicará auto atención sobre todos los m parches para finalmente transformar la salida del bloque Transformer a una clase.

Capítulo 4

Conjunto de datos

Antes de poder realizar el diseño y experimentación del modelo, se necesita obtener un conjunto de datos. Ante esto se dio a la tarea de buscar diversos recursos electrónicos en donde se tiene a señantes señando oraciones completas en LSM en vez de únicamente señar una sola palabra. Esto ayuda a obtener datos más cercanos a los de la vida real al tener ruido implícito dado que una seña puede interpretarse de distintas formas dependiendo el contexto y la persona que la realiza.

4.1. Obtención del conjunto de datos

Derivado de la pandemia SARS-COV2 [42] surgieron diversos recursos digitales dirigidos al público en general. Algunos de éstos iban particularmente dirigidos hacia personas con discapacidad auditiva, los cuales abarcan varios temas de interés, tales como el educativo, informativo, problemas sociales, etc. Dentro de estos recursos se eligió el producido por la Secretaría de Educación Pública (SEP), dentro del programa "Aprende en casa 1 y 2"[49]. Este programa se enfocó en la generación de contenido educativo e informativo con material dirigido tanto a estudiantes como tutores, madres y padres de familia, buscando subsanar el rezago educativo provocado por el cierre de escuelas durante la pandemia.

Específicamente, se descargaron 216 videos del programa, desde su canal de Youtube [49], los cuales se segmentan de la siguiente forma:

- 66 videos del tópico "Aprendiendo a amar a México", los cuales se enfocan en describir la historia de cada estado junto a sus principales atracciones. Cada uno tiene una duración de 60 minutos.
- 20 videos dirigidos a 1° de primaria.
- 25 videos dirigidos a 2° de primaria.
- 25 videos dirigidos a 3° de primaria.
- 20 videos dirigidos a 4° de primaria.
- 25 videos dirigidos a 5° de primaria.

- 20 videos dirigidos a 6° de primaria.
- 1 video dirigido a educación especial.
- 4 videos dirigidos a Educación Inicial.
- 10 videos dirigidos a educación preescolar, en los cuales se da una introducción a los tópicos vistos en 1° de primaria.

Para construir el conjunto de datos, lo primero que se realizó fue obtener una transcripción de todos los videos por medio de Torchaudio[57]. Una vez se tiene la transcripción, se generó una lista de frecuencia de palabras de todos los videos. Se seleccionaron las 85 señas con mayor frecuencia de esta lista, con las cuales se etiquetó el conjunto de datos para entrenar al modelo. Estas señas se muestran en la Tabla 4.1 y corresponden a palabras y frases del lenguaje español.

Agua	Amigo	Año	Árbol	Arriba
Arte	Ave	Ayudar	Azul	Belleza
Bienvenido	Casa	Cero	Cinco	¿Cómo estás?
Conocer	Cuatro	Diez	Domingo	Dos
Edificio	Escuela	Familia	Feliz	Gracias
Guerra	Historia	Hola	Hoy	Iglesia
Jabón	Jueves	Jugar	Lavarse las manos	Leer
Libro	Lunes	Madera	Maestro	Mañana
Mar	Martes	Mejor	México	Miedo
Miércoles	Mil	Mismo	Mundo	Museo
Música	Niño	Noche	Nombre	Norte
Nosotros	Número	Ocho	Otros	País
Piedra	Piñata	Pueblo	Río	Rojo
Sábado	Saber	Semana	Siempre	Siete
Sol	Sur	Tiempo	Todo	Trabajo
Treinta	Tres	Uno	Ustedes	Veinte
Ver	Veracruz	Verde	Vida	Viernes

Tabla 4.1: Conjunto de señas a utilizar.

Con las palabras elegidas se buscó, por medio de la transcripción, la ubicación de cada una y al ser encontrado el intervalo en donde se realiza la seña, se extrajo el *clip*. Se realizó el procedimiento en al menos 10 interpretaciones de la seña, procurando tener distintos señantes dentro del conjunto de la seña. En la Tabla 4.2 se muestra el número de *clips* en los que aparece cada señante en el conjunto de datos recopilado.

Clave del Señante	**Número de clips en los que aparece**
1	68
2	40
3	17

5	26
6	59
7	49
8	67
9	40
10	19
11	71
12	59
13	25
14	22
15	50
16	161
17	26
18	2
19	24
20	2
21	54

Tabla 4.2: # de videos por señante dentro del conjunto de datos extraído

4.2. Análisis de los *clips* extraídos

Dentro de esta sección se analizará un pequeño conjunto de los *clips* extraídos para comparar señas similares entre sí. Se observa que, a pesar de interpretar una misma seña, puede haber diversas formas de realizar el movimiento con las manos. Estas diferencias se dan por el lugar donde se empieza a señar, el contexto de la oración y las emociones descritas. Un subconjunto de las señas extraídas se puede ver en la Figura 4.1, en la cual se observan 4 señas realizadas por 4 señantes distintos.

Dentro de algunos *clips* extraídos, hay interpretaciones de señas que son similares, pero representan distintas cosas. Para poder distinguir el significado de la interpretación se toma en cuenta la posición de las manos respecto al cuerpo, además de la flexión de los dedos y codos durante la interpretación. También se tienen casos donde la misma seña tiene distintas interpretaciones, distinguiéndose particularmente por el lugar de inicio y final de la interpretación. Un ejemplo de esto se puede observar en la Figura 4.2, donde el inicio de la seña depende de la palabra anterior. Por ejemplo, la interpretación del 7 difiere cuando indica un conteo de personas a cuando es un conteo de animales.

También se tienen señas con movimientos similares entre sí con distintos significados, o movimientos diferentes a pesar de ser la misma representación. Esto se puede ver en la Figura 4.3, donde se observan 3 señas: la primera corresponde a **sol**, la segunda a **árbol** y la tercera y cuarta a **tres**.

Con el conjunto de datos obtenido previamente se tienen datos variados, pero no tan numerosos, por lo que se busca aumentar el número de *clips* sin afectar la integridad

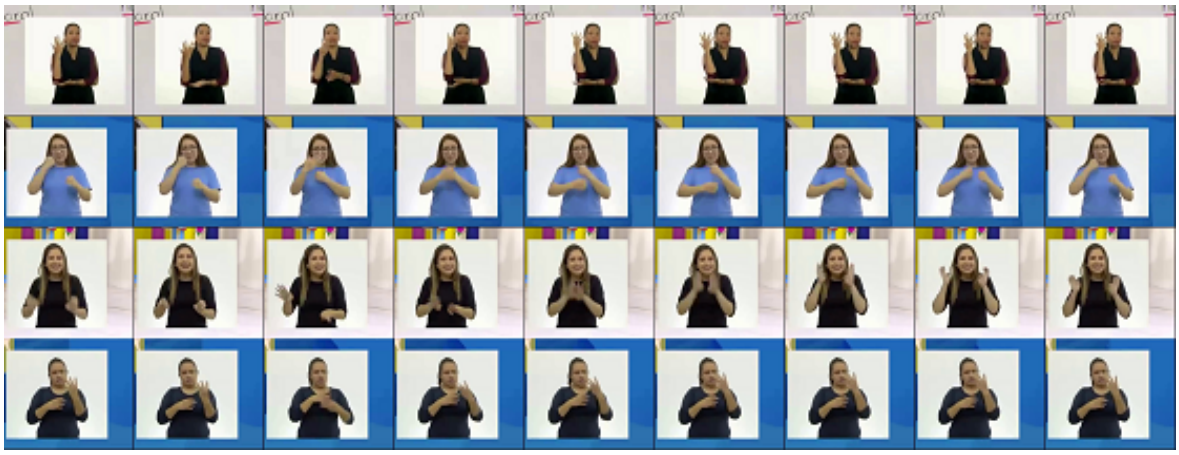


Figura 4.1: Se muestran cuatro señas. De arriba hacia abajo, extendiéndose hacia la derecha la interpretación: 1) Árbol, 2)Año, 3) ¿Cómo estás? y 4) Tres.

Figura 4.2: Se muestra la señalización de la palabra **siete**, se observa que la seña no siempre empieza ni termina en el mismo lugar. Esto se da al depender de la seña inmediata anterior y la mano dominante del señante.

del conjunto de datos. Para generar un conjunto de datos más grande, se extraen de cada *clip* los cuadros de video que le componen para ir construyendo nuevos *clips*, los cuales tendrán un menor número de cuadros de video a los del original, preservando el orden original de los cuadros de videos para poder seguir representando a la seña original. Para asegurar que los nuevos *clips* son únicos se utiliza la siguiente ecuación, tomada de [30]:

$$z = \left\lfloor \frac{l}{n-1} \right\rfloor = \left\lfloor \frac{l}{4} \right\rfloor, \tag{4.1}$$

donde, dado un video cualquiera de una seña $S = (f_1, f_2, \ldots, f_l)$ la cual se compone de l cuadros de video, denotados por f_1 a f_l, se seleccionan n cuadros, en nuestro caso 5, de manera aleatoria. En base a la fórmula se generan 5 *subclips* por cada *clip*. Esto nos da un total de 8900 *subclips* que componen al conjunto de datos a utilizar. En la Figura

Figura 4.3: Se muestra un comparativo entre tres señas, en el cual se observa la similitud entre dos diferentes interpretaciones en la primera y segunda fila, siendo señas diferentes (sol y árbol). Y se observan diferencias notorias entre dos interpretaciones de la misma seña en las últimas dos filas (tres)

4.4 se observa cómo a partir de un *clip* se generan 5 *subclips* de una misma duración.

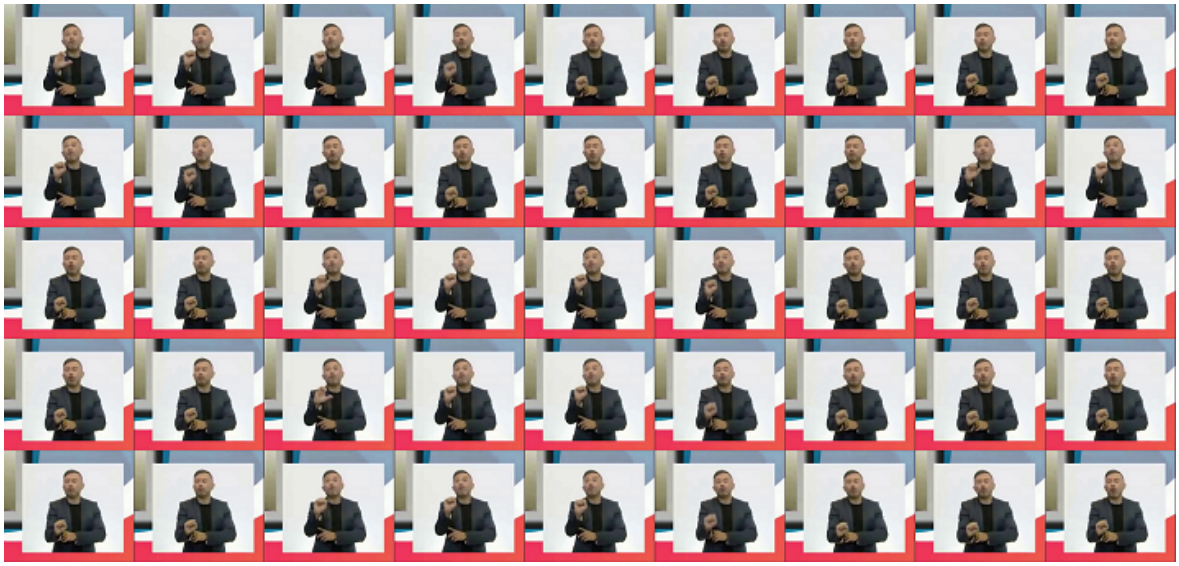


Figura 4.4: Muestras de 5 mini-videos de 9 cuadros. A pesar de provenir del mismo video principal, los mini-videos muestran diferencias entre sí

Con este conjunto final se generaron 6 conjuntos de datos diferentes entre sí que difieren en el tamaño del cuadro de video y en si la imagen se encuentra en escala de grises o a color. Estos conjuntos se nombran de la siguiente manera:

- C300Color: Todos los videos son de tamaño 300×300 y se encuentran a color
- C500Color: Todos los videos son de tamaño 500×500 y se encuentran a color
- C1000Color: Todos los videos son de tamaño 1000×1000 y se encuentran a color

- C300Gris: Todos los videos son de tamaño 300×300 y se encuentran en escala de grises
- C500Gris: Todos los videos son de tamaño 500×500 y se encuentran en escala de grises
- C1000Gris: Todos los videos son de tamaño 1000×1000 y se encuentran en escala de grises

Teniendo los conjuntos de datos a usar, se configura un modelo para poder traducir las señas previamente mencionadas. Pero antes de diseñar el modelo, se introducirán las arquitecturas Swin Transformer (***S****hifted* ***Win****dows Transformers*) en las que se basará.

Capítulo 5

Transformer para reconocimiento de señas

La arquitectura Swin Transformer (ST) [34] surge ante la necesidad de adaptar la arquitectura Transformer a tareas de visión computacional. La particularidad de esta arquitectura consiste en la modificación de la arquitectura Transformer de visión (ViT) [21] por una que construye mapas de características jerárquicos, lo que permite que se tenga una complejidad computacional lineal respecto al tamaño de la imagen al momento de calcular la atención, a diferencia de ViT que toma tiempo cuadrático respecto al alto y ancho de la imagen. La diferencia del proceso para recorrer las imágenes entre ST y ViT se puede ver en la Figura 5.1.

Estos mapas son comparables a los procesados por las CNN, por lo que se pueden utilizar diversas arquitecturas convoluciones sobre éstos (por ej. FPN [33] o U-Net [47]) para poder resolver diversos problemas de visión computacional tales como identificación de objetos. Dentro de estos mapas se utiliza una atención local, definida más adelante, lo que permite calcular de una forma más rápida la atención sin perder información de la imagen. Para describir cómo logra realizar esto, primero se describirá la arquitectura de forma general, para posteriormente profundizar entre sus distintos componentes. La arquitectura general de ST, junto al Bloque Swin Transformer (STB), se puede ver en la Figura 5.2.

5.1. Arquitectura general

Dentro de la arquitectura ST, como primer paso se alimenta la imagen a una capa de Partición en Parches, la cual divide la imagen en parches, los cuales son tratados como *tokens*, evitando que un píxel pertenezca a más de un parche. Para buscar características dentro de los parches, se utiliza el bloque ST, el cual utiliza atención con deslizamiento de ventanas, ambos conceptos se definen más adelante.

Más específicamente, la arquitectura ST recibe una imagen RGB de tamaño $H \times W \times 3$ y la alimenta a la capa de Partición en Parches, donde el componente de cada parche es la concatenación de los valores RGB. El tamaño del parche es de 4×4, teniendo que la dimensión de las características del parche es de $4 \times 4 \times 3 = 48$.

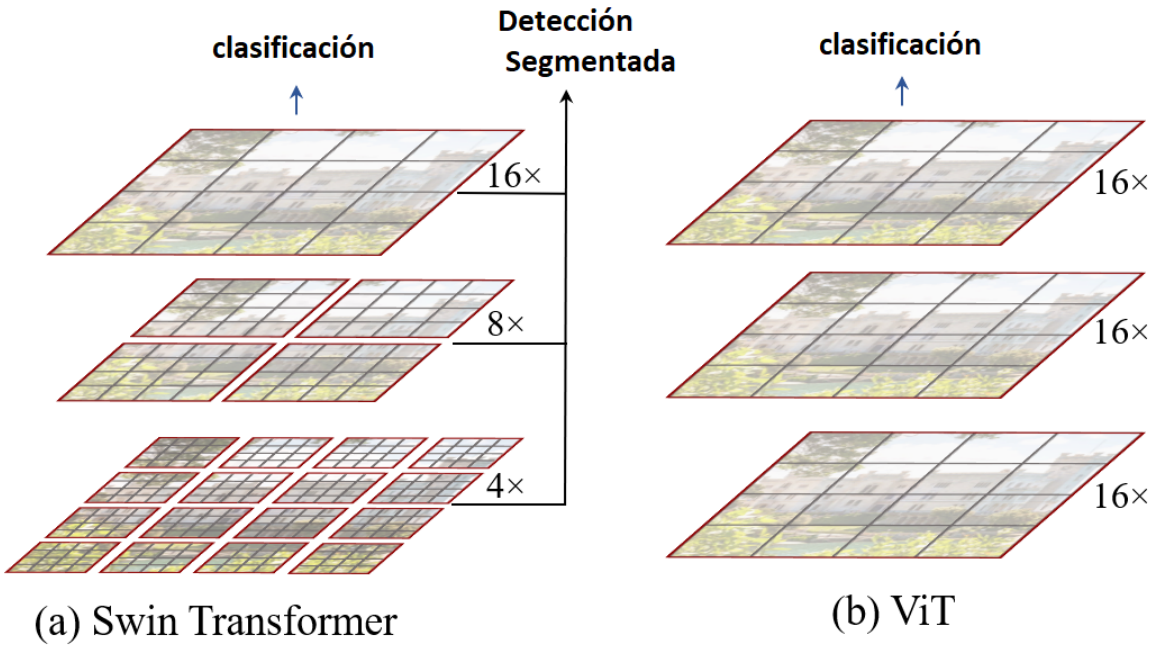


Figura 5.1: (a) La forma en que Swin Transformer va generando los mapas de características jerárquicos dada una imagen. Esto permite igualar el rendimiento de otros Transformer de visión, pero con un tiempo considerablemente menor. (b) Otras arquitecturas Transformer[21] producen mapas de características con de resolución baja, además de aplicar atención global sobre sus distintas capas, provocando que se tenga una complejidad computacional cuadrática respecto al tamaño de la imagen. Imagen tomada de [34]

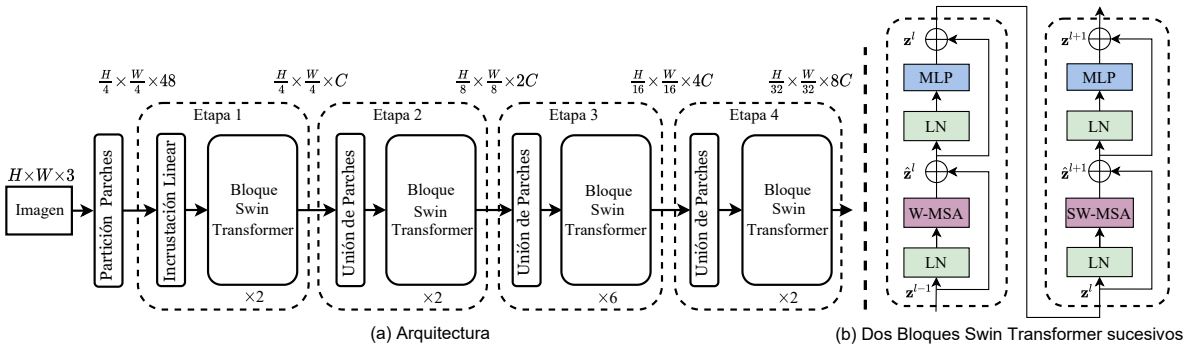


Figura 5.2: (a) La arquitectura de Swin Transformer (Swin-T).(b) Dos bloques Transformers Swin, donde W—MSA y SW—MSA son módulos de atención propia multicabeza con configuración normal y configuración de ventana deslizada, respectivamente. Imagen tomada de [34]

Posteriormente, cada parche se alimenta a una capa lineal de incrustación, la cual proyecta las características a una dimensión arbitraria, denotada por C. Para finalizar la primera etapa, se alimentan a un bloque ST, el cual mantiene el mismo número de parches $\left(\frac{H}{4} \times \frac{W}{4}\right)$ respecto a los recibidos, con lo que finaliza la primera etapa.

Al ser una arquitectura jerárquica, se tiene que las etapas $2, 3$ y 4 son similares a la primera etapa, con la diferencia de que en vez de tener una capa lineal de incrustación, se usará una capa de Unión de Parches, la que une a los parches para disminuir su número conforme la arquitectura se vuelve más profunda. Esto nos da que, para la segunda etapa, la capa concatena las características de cada grupo de 2×2 parches vecinos, aplicando una capa lineal en las características concatenadas de dimensión $4C$. Esto reduce el número de parches por un múltiplo de $2 \times 2 = 4\,(2x)$, siendo la dimensión de salida $2C$, la cual se alimenta al bloque ST, finalizando la segunda etapa. Para la tercer y cuarta etapa se repite el mismo proceso, siendo las resoluciones de salida $\left(\frac{H}{16} \times \frac{W}{16}\right)$ y $\left(\frac{H}{32} \times \frac{W}{32}\right)$ y las dimensiones de salida $4C$ y $8C$, respectivamente.

5.2. Bloques ST

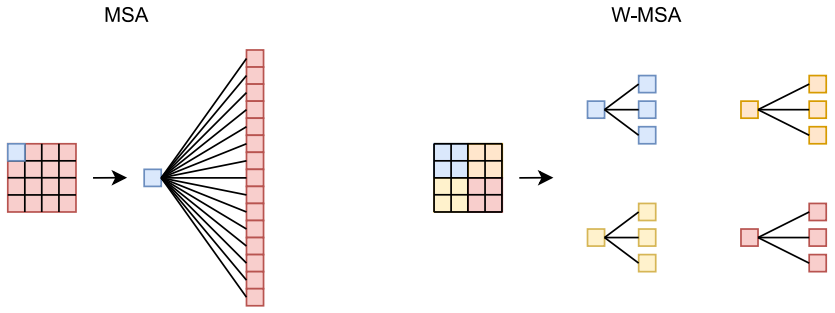


Figura 5.3: Del lado izquierdo se observa cómo un bloque Transformer en la arquitectura ViT realiza atención global (MSA) sobre todos los píxeles que lo componen. Del lado derecho se observa cómo un bloque ST realiza atención local con deslizamiento de ventana (W—MSA) al dividir los píxeles en parches, donde cada uno corresponde a una sola ventana.

La composición de estos bloques se puede observar en el inciso b) de la Figura 5.2. Estos nuevos bloques se construyen tomando el bloque Transformer utilizado en ViT, pero modificando la forma en que se aplica la atención. La diferencia entre un bloque Transformer de ViT y un bloque ST se puede apreciar en la Figura 5.3 . De manera más formal, supongamos que una ventana contiene un parche de tamaño $M \times M$, la complejidad de una atención global (MSA) y una atención local (W—MSA) dada una imagen de tamaño $h \times w$ se calcula de la siguiente manera:

$$\omega\,(MSA) = 4hwC^2 + 2\,(hw)^2\,C \tag{5.1}$$

$$\omega\,(W^{-}MSA) = 4hwC^2 + 2MhwC, \tag{5.2}$$

Como se puede observar, para la atención global se tiene complejidad cuadrática respecto al tamaño $h \times w$ y para la atención local la complejidad es lineal respecto a M. Esto es lo más relevante de este bloque, dado que conforme incrementa el tamaño de la imagen recibida el cálculo de complejidades cuadráticas se empiezan a volver insostenible, mientras que para la complejidad lineal el tiempo de ejecución no aumenta tan considerablemente.

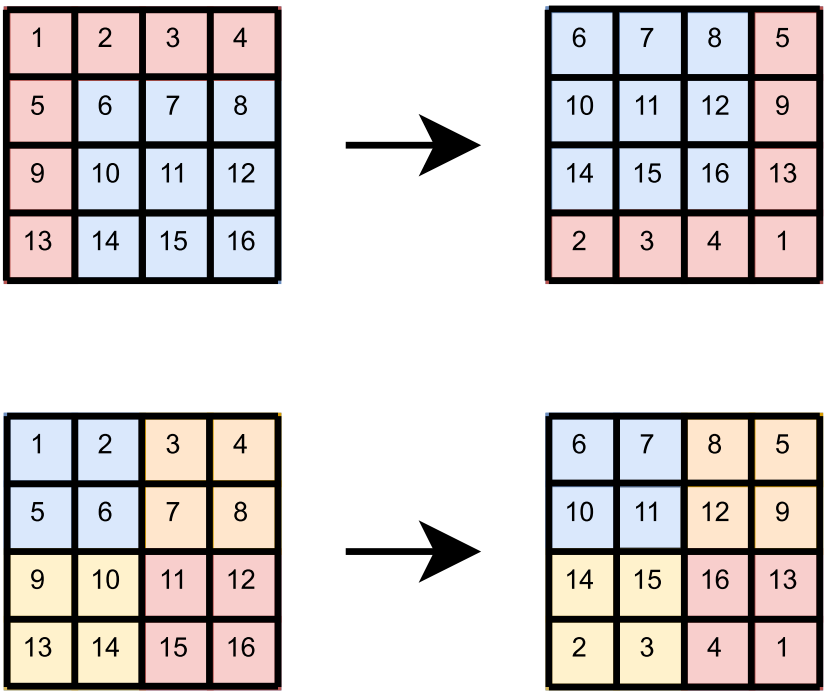


Figura 5.4: Del lado superior se observa cómo se mueven los parche al aplicarse la operación SW—MSA. Del lado inferior se observa cómo cambian los parche de ventana y, por ende, los cálculos para la atención local.

Adentrándonos a la forma en que la atención local se calcula, se observa que el bloque ST únicamente aplica atención sobre los parches dentro de su ventana, lo que limita su poder de representación. Por lo tanto, es necesario calcular la atención sobre el parche dentro de otras ventanas. Para lograr esto se introduce un deslizamiento de ventanas, que se puede observar en la Figura 5.4, el cual modifica la posición de los parches provocando que las ventanas cambien las características que contienen, lo que permite que cada ventana calcule la atención local tomando en cuenta nuevas características. Retomando la Figura 5.2 en el inciso b), el primer módulo utiliza las ventanas generadas por la capa de partición de parches, la cual empieza desde el píxel de la esquina superior izquierda para ir generando mapa de características de 8×8 en 2×2 ventanas de tamaño 4×4 (M = 4). Posteriormente, el siguiente módulo "desliza las ventanas" modificando su contenido, estos deslizamientos se realizan con una magnitud de $\left(\frac{M}{2}, \frac{M}{2}\right)$ píxeles. Con esta aproximación, la salida del bloque ST se calcula de la siguiente manera:

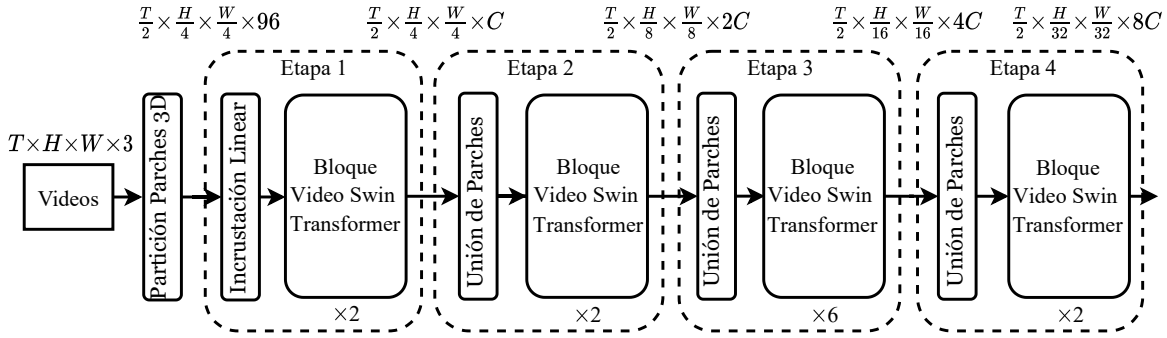


Figura 5.5: Arquitectura general de Video Swin Transformer, en su versión Tiny. Imagen tomada de [35]

$$\begin{aligned}
\hat{\boldsymbol{z}}^{l} &= W^{-}MSA\left(LN\left(\boldsymbol{z}^{l-1}\right)\right) + z^{l-1} \\
\boldsymbol{z}^{l} &= MLP\left(LN\left(\hat{\boldsymbol{z}}^{l}\right)\right) + \hat{\boldsymbol{z}}^{l} \\
\hat{\boldsymbol{z}}^{l+1} &= SW^{-}MSA\left(LN\left(\boldsymbol{z}^{l}\right)\right) + \boldsymbol{z}^{l} \\
\boldsymbol{z}^{l+1} &= MLP\left(LN\left(\hat{\boldsymbol{z}}^{l+1}\right)\right) + \hat{\boldsymbol{z}}^{l+1},
\end{aligned} \tag{5.3}$$

donde $\hat{\boldsymbol{z}}^{l}$ y $\boldsymbol{z}^{l}$ denotan las características de salida del módulo (S)W—MSA y del módulo MLP para el bloque l, respectivamente; W—MSA y SW—MSA denotan la atención local basada en ventanas y la atención local calculada posteriormente al desplazar las ventanas, respectivamente.

Finalmente, para el cálculo de la autoatención, se utiliza el sesgo de posición relativa $\boldsymbol{B} \in \mathbb{R}^{M^2\times M^2}$ por cada cabeza, calculando la siguiente similitud:

$$a\left(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}\right) = \text{SoftMax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{T}}{\sqrt{d}} + \boldsymbol{B}\right)\boldsymbol{V}, \tag{5.4}$$

donde $\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V} \in \mathbb{R}^{M^2\times d}$ son la consulta, llave y valor, respectivamente; d es la dimensión de la consulta y llave; y M^2 es el número de parches en una ventana. Dado que la posición relativa a lo largo de cada eje cae dentro del rango $[-M+1, M-1]$, se parametriza en una matriz de sesgos $\boldsymbol{B} \in \mathbb{R}^{(2M-1)\times(2M-1)}$ y los valores de $\boldsymbol{B}$ se toman de $\mathcal{B}$.

5.3. Video Swin Transformer (VST)

La arquitectura Video Swin Transformer (VST) [35] es una extensión de ST para poder procesar videos en vez de únicamente imágenes. La arquitectura general de VST se puede observar en la Figura 5.5. A primera vista es muy similar a la arquitectura ST, pero agrega un parámetro de temporalidad para capturar las dependencias secuenciales de los videos (secuencias de imágenes).

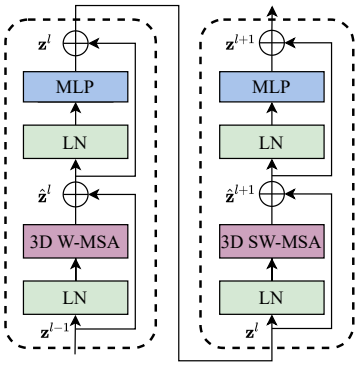


Figura 5.6: Una ilustración mostrando a dos bloques Video Swin Transformer sucesivos.

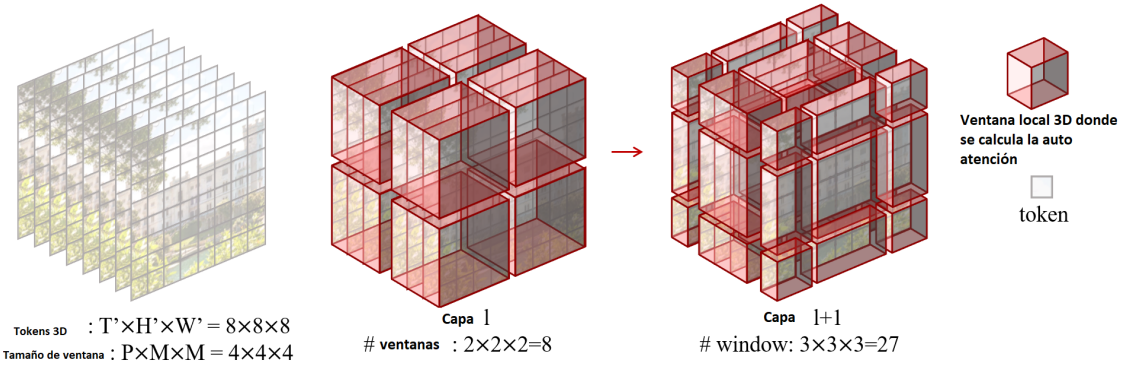


Figura 5.7: Una muestra del proceso de generación de las ventanas 3D deslizadas, se inicia con una dimensión de $8 \times 8 \times 8$ para los tokens y $4 \times 4 \times 4$ para las ventanas, para la capa l se tiene un total de 8 ventanas $(2 \times 2 \times 2)$ y para la capa $l+1$ se tienen 27 ventanas $(3 \times 3 \times 3)$

Como se puede observar, en VST el video de entrada tiene una dimensión de $(T \times H \times W \times 3)$, donde T denota el número de cuadros de video, cada uno con $H \times W \times 3$ píxeles. Para el caso de VST, los parches se obtienen por medio de una capa de partición de parche 3D, donde cada parche tiene un tamaño de $2 \times 4 \times 4 \times 3$. Esto significa que la capa de partición de parche genera parches 3D de $\frac{T}{2} \times \frac{H}{4} \times \frac{W}{4}$, cada uno representado con vectores de características de 96 dimensiones. Al igual que en la ST, se realiza una reducción de resolución durante 4 etapas y una reducción espacial de $2\times$, pero sin afectar a la dimensión temporal para seguir la estructura jerárquica de la arquitectura ST. Finalmente, la capa de unión de parches concatena las características de cada grupo de 2×2 parches vecinos espaciales y aplica una capa lineal para proyectar las características concatenadas a la mitad de su dimensión. En esta arquitectura se define un Bloque Video Swin Transformer (VSTB), el cual reemplaza al bloque ST por uno de autoatención de ventana deslizante 3D, dejando los demás componentes intactos.

Al igual que los Bloques ST, los Bloques VST carecen de conexiones entre las distintas ventanas, por lo que se extiende el mismo mecanismo de ST entre ventanas 3D para modificar los parches que cada ventana cubre. Esto se logra al hacer que la

primer capa del módulo de autoatención use la configuración sin deslizar las ventanas, obteniendo $\frac{T'}{P} \times \frac{H'}{M} \times \frac{W'}{M}$ parches 3D. Para la segunda capa se desliza la ventana a lo largo de los ejes temporales, de altura y de anchura por $\frac{P}{2}, \frac{M}{2}, \frac{M}{2}$ parches para obtener la nueva configuración.

$$\begin{aligned} \hat{\boldsymbol{z}}^l &= 3DW^-MSA\left(LN\left(\boldsymbol{z}^{l-1}\right)\right) + \boldsymbol{z}^{l-1} \\ \boldsymbol{z}^l &= FFN\left(LN\left(\hat{\boldsymbol{z}}^l\right)\right) + \hat{\boldsymbol{z}}^l \\ \hat{\boldsymbol{z}}^{l+1} &= 3DSW^-MSA\left(LN\left(\boldsymbol{z}^l\right)\right) + \boldsymbol{z}^l \\ \hat{\boldsymbol{z}}^{l+1} &= 3DSW^-MSA\left(LN\left(\boldsymbol{z}^{l+1}\right)\right) + \boldsymbol{z}^{l+1}, \end{aligned} \tag{5.5}$$

donde $\hat{\boldsymbol{z}}^l$ y $\boldsymbol{z}^l$ denotan las características de salida del módulo 3D(S)W—MSA y del módulo FFN para el bloque l, respectivamente; 3DW—MSA y 3DSW—MSA denotan la atención propia multi-cabeza basada en ventanas 3D y la configuración de partición de ventanas deslizantes, respectivamente.

Respecto a la atención, se utiliza la descrita en la arquitectura ST (ecuación 5.5), recordando que la posición relativa de cada eje cae dentro del rango $[-P+1, P-1]$ para el eje temporal y $[-M+1, M-1]$ para los demás ejes. En esta atención se parametriza una matriz de sesgos $\hat{\mathcal{B}} \in \mathbb{R}^{(2P-1)\times(2M-1)\times(2M-1)}$ y los valores de $\mathcal{B}$ se toman de $\hat{\mathcal{B}}$. Esto nos da finalmente la arquitectura de VST, que es muy similar a la ST, pero considerando una nueva dimensión para la temporalidad del video. Dentro de esta arquitectura, existen las siguientes variaciones:

- Swin-Tiny (Swin-T): $C = 96$, donde el número de capas por etapas es $= \{2, 2, 6, 2\}$
- Swin-Small (Swin-S): $C = 96$, donde el número de capas por etapas es $= \{2, 2, 18, 2\}$
- Swin-Base (Swin-B): $C = 128$, donde el número de capas por etapas es $= \{2, 2, 18, 2\}$
- Swin-Large (Swin-L): $C = 192$, donde el número de capas por etapas es $= \{2, 2, 18, 2\}$

Dentro de este trabajo se entrenarán y probarán las 4 arquitecturas previamente mencionadas, analizando casos donde los modelos fueron previamente entrenados en trabajos de visión.

Capítulo 6

Experimentación y resultados

Teniendo la arquitectura y el conjunto de datos, se realiza la experimentación. Como se mencionó, se tienen 6 distintos conjuntos de datos con los que se entrenarán y evaluarán 4 modelos desde cero (Swin-T, Swin-B, Swin-S, LSTM) y dos modelos pre-entrenados: un modelo Swin-T pre-entrenado con Kinetics600 y un modelo Swin-B pre-entrenado con Kinetics400. Estos experimentos se realizaron con la finalidad de comparar el rendimiento de los diferentes modelos al tener y no tener transferencia de conocimiento.

6.1. Preparando el entrenamiento

Dentro del repositorio de VST [35], se encuentran los archivos para poder entrenar y probar los diversos modelos. Las diferentes configuraciones de VST se muestran en la Tabla 6.1, todas ellas entrenadas durante 50 épocas.

6.2. Preparando los datos

Se dividió a los conjuntos mostrados en el Capítulo 4 en subconjuntos de entrenamiento (80 % del conjunto), validación (10 % del conjunto) y prueba (10 % del conjunto). Al generar las particiones se utilizó un muestreo estratificado para garantizar una representación equitativa entre las distintas clases.

Se obtuvieron los índices de cada video asignado al subconjunto de entrenamiento, validación y prueba. Esto se realizó para que, dentro de las versiones generadas, los índices de los videos utilizados para cada subconjunto fueran los mismos sin importar

Modelo	Learning Rate	Weight Decay	# Parámetros
Swin-T	1e-3	0.02	28 Millones
Swin-S	1e-3	0.02	50 Millones
Swin-B	3e-4	0.05	88 Millones

Tabla 6.1: Configuración de los modelos tomadas de [35]

la versión a la que pertenecieran. Posterior a esto nos queda la distribución de cada seña dentro de cada conjunto, como se observa en la Tabla B.1.

6.3. Métricas de entrenamiento

Para evaluar el rendimiento de los modelos se emplearon métricas las cuales se definen a continuación: Sea m el número de ejemplos, m_j el ejemplo en la posición j, c el número de clases, $f(i,j)$ la probabilidad de que i pertenezca a la clase j, siendo f una función indicatriz y $C(i,j)$ una función con valor igual a 1 si el elemento j pertenece a la clase i y 0 en caso contrario. Para el uso de estas métricas se utilizará la biblioteca Scikit-learn[44].

6.3.1. *Accuracy*

Es la forma más directa de evaluar a un clasificador, se define como el grado de predicciones correctas de un modelo, por la siguiente fórmula:

$$Accuracy = \frac{\sum_{i=1}^{m} \sum_{j=1}^{c} f(i,j)\, C(i,j)}{m}, \tag{6.1}$$

donde se conocerá a *Top 1 Accuracy* cuando se toma la predicción con mayor probabilidad del modelo y a *Top 5 Accuracy* cuando se obtienen las 5 predicciones con mayores probabilidades del modelo.

6.3.2. *Recall*

Es la proporción de elementos de la clase j que fueron predichos como clase j, es decir:

$$Recall(j) = \sum_{i=1}^{m} \frac{f(i,j)\, C(i,j)}{m_j}, \tag{6.2}$$

donde j es el índice de una clase.

6.3.3. *Precision*

Es la proporción de predicciones asignadas a la clase j que realmente sí pertenecen a dicha clase, es decir:

$$Precision(j) = \frac{\sum_{i=1}^{m} f(i,j)\, C(i,j)}{\sum_{i=1}^{m_j} C(i,j)}, \tag{6.3}$$

donde j es el índice de una clase.

6.3.4. F *score*

Es la media armónica de las medidas *Precision* y *Recall*, calculada de la siguiente manera:

$$F_1 = \frac{2 \times recall(j) \times precision(j)}{recall(j) + precision(j)}, \tag{6.4}$$

6.3.5. Promedio ponderado

El promedio ponderado es el promedio aritmético, pero asignando pesos a las clases de acuerdo al número de elementos en cada una. Se calcula de la siguiente manera:

$$\text{Promedio ponderado} = \frac{\sum_{j=1}^{c} \left(\frac{\sum_{i=1}^{m} f(i,j) C(i,j)}{m_j} \times \frac{m_j}{m} \right)}{c}, \tag{6.5}$$

6.3.6. Promedio macro

El promedio macro se define como el promedio aritmético de las precisiones parciales de cada clase, calculado con la siguiente fórmula:

$$\text{Promedio macro} = \frac{\sum_{j=1}^{c} \frac{\sum_{i=1}^{m} f(i,j) C(i,j)}{m_j}}{c}, \tag{6.6}$$

6.4. Resultados

Con los hiperparámetros definidos al inicio se prosigue a entrenar todos los modelos con sus respectivos conjuntos de datos, para posteriormente evaluar cada modelo con el correspondiente subconjunto de prueba y analizar las diferencias de respuestas entre todos los modelos.

La *accuracy Top-1* y *accuracy Top-5* obtenidas en el subconjunto de prueba por cada modelo, se listan en la Tabla 6.2 mientras que los valores de las otras métricas se listan en la Tabla 6.3.

La distribución de videos por cada seña dentro de cada subconjunto se observa en la Tabla B.1.

Lo primero que notamos respecto a los resultados es que las métricas de la gran mayoría de los modelos indican que tienen buen rendimiento, salvo los modelos Swin-S entrenados con los conjuntos de datos C500Gris y C1000Gris que tuvieron valores considerablemente más bajos en todas las métricas. De igual forma, el modelo pre-entrenado con K400 tuvo el mejor rendimiento entre todos los modelos, al llegar a un $AccuracyTop5$ de 100 % y $AccuracyTop1$ de 99.88 %.

Además, se observa que los promedios ponderados de las diferentes métricas indican que los modelos son capaces de diferenciar correctamente entre distintas clases a pesar de haber clases con mayor número de elementos.

#	**Conjunto de Datos**	**Modelo**	% ***Accuracy Top-1***	% ***Accuracy Top-5***
1	C300Color	Swin-T	95.61 %	99.32 %
2	C500Color	Swin-T	96.17 %	99.77 %
3	C1000Color	Swin-T	95.16 %	99.66 %
4	C300Gris	Swin-T	93.37 %	99.66 %
5	C500Gris	Swin-T	94.71 %	99.55 %
6	C1000Gris	Swin-T	92.69 %	99.66 %
7	C300Color	Swin-S	94.04 %	99.55 %
8	C500Color	Swin-S	92.47 %	99.10 %
9	C1000Color	Swin-S	91.23 %	99.43 %
10	C300Gris	Swin-S	90.33 %	99.43 %
11	C500Gris	Swin-S	63.25 %	92.92 %
12	C1000Gris	Swin-S	50.89 %	86.62 %
13	C300Color	Swin-B	95.73 %	99.21 %
14	C500Color	Swin-B	96.51 %	99.43 %
15	C1000Color	Swin-B	95.28 %	99.55 %
16	C300Gris	Swin-B	95.05 %	99.32 %
17	C500Gris	Swin-B	95.16 %	99.55 %
18	C1000Gris	Swin-B	95.05 %	99.55 %
19	C300Color	Kinetics400	99.88 %	100 %
20	C300Color	Kinetics600	94.49 %	99.55 %
21	C300Color	LSTM	91.23 %	91.23 %

Tabla 6.2: *Accuracy Top-1* y *Top-5* de los experimentos realizados

#	***Precision***		***Recall***		***F1-score***	
	promedio macro	promedio ponderado	promedio macro	promedio ponderado	promedio macro	promedio ponderado
1	0.95748	0.95922	0.95412	0.95618	0.95230	0.95435
2	0.96212	0.96365	0.96000	0.96180	0.95777	0.95958
3	0.95597	0.95663	0.95020	0.95169	0.94873	0.94999
4	0.93856	0.94114	0.93230	0.93371	0.93145	0.93355
5	0.95551	0.95700	0.94471	0.94719	0.94445	0.94668
6	0.93061	0.93203	0.92431	0.92697	0.92407	0.92621
7	0.93966	0.94220	0.93765	0.94045	0.93639	0.93915
8	0.92923	0.93078	0.92240	0.92472	0.91974	0.92189
9	0.91905	0.92182	0.91002	0.91236	0.90759	0.91032
10	0.90635	0.90880	0.90266	0.90337	0.89986	0.90144
11	0.66579	0.66510	0.63042	0.63258	0.61769	0.61944
12	0.52771	0.52865	0.50366	0.50899	0.48708	0.49094
13	0.96250	0.96331	0.95569	0.95730	0.95422	0.95564
14	0.96752	0.96811	0.96353	0.96517	0.96273	0.96395
15	0.95480	0.95666	0.95059	0.95281	0.94960	0.95177
16	0.94853	0.95043	0.94824	0.95056	0.94655	0.94874
17	0.95372	0.95527	0.94941	0.95169	0.94901	0.95102
18	0.95283	0.95412	0.94902	0.95056	0.94669	0.94824
19	0.99893	0.99898	0.99882	0.99888	0.99882	0.99887
20	0.93942	0.94197	0.94235	0.94494	0.93932	0.94195
21	0.91029	0.92528	0.91840	0.91236	0.90877	0.91346

Tabla 6.3: Rendimiento de los modelos respecto a *Precision*, *Recall* y *F1-score*

6.4.1. Comparación de resultados

En un trabajo previo[25] se utilizó un modelo LSTM entrenado con videos procesados por OpenPose[16] para extraer la pose de la persona señante y, en base a ella, identificar la seña que se estaba realizando. Los resultados de dicho trabajo y la comparación de los modelos presentados en el presente trabajo se muestran en la Tabla 6.4.

Analizando la tabla, se observa cómo cae el rendimiento del modelo LSTM al incrementar el número de clases, esto derivado de que empieza a haber señas con movimientos muy similares y la pose obtenida por OpenPose no logra ser lo suficientemente

Tipo de modelo	**Número de clases**	***Accuracy* Top 1**
LSTM	28	97.34 %
LSTM	85	91.34 %
Swin-B (K400)	85	99.88 %

Tabla 6.4: Comparación del modelo usado en trabajos anteriores con el trabajo actual.

granular como para que el modelo pueda diferenciar los movimientos más pequeños que se hacen. No obstante, algo muy importante que notar es el tiempo de entrenamiento entre una arquitectura LSTM y una VST, dado que el primero se procesó en OpenPose para alimentar las salidas a la arquitectura, realizando todo este proceso en un tiempo de 3 horas. En comparación, la arquitectura VST se alimenta directamente los videos sin tener que usar otra herramienta, pero toma más de 3 días en ser entrenada. A pesar del buen rendimiento del modelo, aún falta el probar sobre un conjunto de datos aún mayor y validar que los modelos sean capaces de generalizar a otros señantes y escenarios.

Capítulo 7

Conclusiones y trabajo a futuro

En este trabajo se abordó el problema de reconocimiento de señas de la LSM utilizando arquitecturas Transformer. Para lograr este objetivo se tuvo que generar un conjunto de datos de 85 clases en base a videos que tenían a personas señantes.

Dentro de los experimentos realizados se observa que los modelos Swin Transformer poseen un rendimiento mayor a los modelos LSTM. Sin embargo, es importante considerar el compromiso entre el rendimiento y el costo computacional, ya que en muchos casos no se cuenta con los recursos de cómputo necesarios para entrenar los modelos y en muchas aplicaciones se tienen recursos muy limitados para realizar la clasificación (por ej. si se busca ejecutar en dispositivos móviles).

Al finalizar este trabajo se observó que el mayor problema para realizar más trabajos sobre el reconocimiento de la LSM reside en la falta de datos que contengan personas señando con sus respectivas etiquetas. Por lo que al haber dichos datos se empezará a acelerar el desarrollo de diferentes soluciones ante este tipo de problemáticas. Igual faltará probar la arquitectura VST sobre problemas más ambiciosos, como se ha hecho en otros trabajos[5] sobre otras lenguas de señas, en particular el reconocimiento de frases y su traducción al español.

Con lo dicho previamente, como trabajo a futuro lo principal será la generación de un conjunto de datos mucho más extenso y más complejo para poder realizar trabajos sobre éstos que involucren diversas técnicas y heurísticas para el reconocimiento tanto de señas de la LSM como frases y oraciones. Como meta final se espera poder tener un modelo que sirva como traductor entre una persona señante y una persona no señante.

Apéndice A

Características de la LSM

La LSM es una lengua de señas que, de igual forma que las lenguas habladas, posee una gramática, modalidad y reglas que la diferencian del lenguaje hablado. Para el alcance de este trabajo definiremos brevemente algunas de las principales características de la LSM.

A.1. Tipo de señas

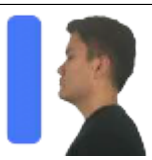
Dentro de la forma en que se distinguen las señas, se pueden obtener cuatro categorías, las cuales agrupan de acuerdo al número de manos usadas durante la seña, y si los movimientos se realizan al mismo tiempo o en diferente tiempo. Estas categorías se pueden ver en la Tabla A.1.

Tipo de Seña	**Definición**
Seña manual (SM)	Seña que se articula con sólo una mano
Seña bi-manual (SB)	Seña que se articula con dos manos, de manera dispar
Seña simétrica (SS)	Seña que se articula con dos manos, de manera simétrica, ya sea por medio de movimientos de espejo, idénticos e inversos o inversamente proporcionales
Seña compuesta (SC)	Seña que se articula a través de al menos dos señas simples

Tabla A.1: Agrupaciones a las que puede pertenecer una seña.

A.2. Ubicación de las señas

Es el espacio en donde se realizan las señas, en particular, la ubicación de las señas se realiza respecto a la cara, el cuerpo, los brazos y las manos, mientras que la ubicación de profundidad se realiza respecto al cuerpo. Éstas se pueden ver en la Tabla A.2. Si una seña se sale del espacio del cuerpo, indica el énfasis en dicha seña, causado principalmente por algún tipo de emoción.

	Imagen lateral de la ubicación frontal	Cara	Cuerpo	Brazos	Manos
1					
2					
3					
4					
5					
6					
7					

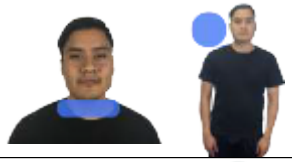
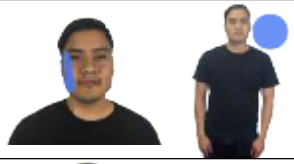
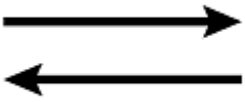
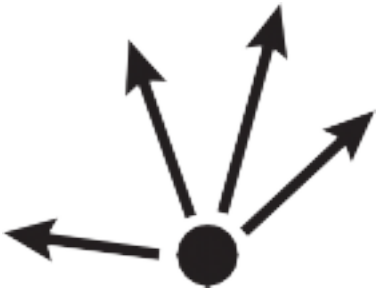
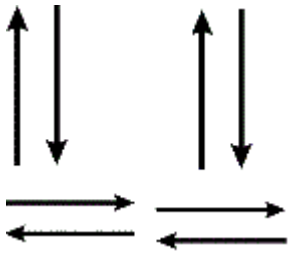
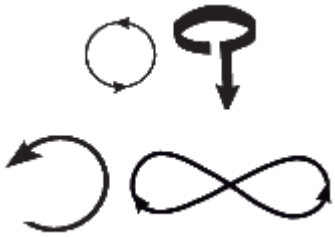
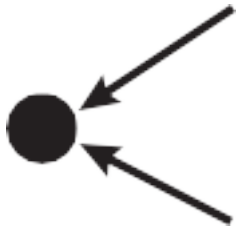
8	
9	
10	
11	
12	
13	

Tabla A.2: Ubicación en el cuerpo en donde se puede realizar una seña. Imagen tomada de [19]

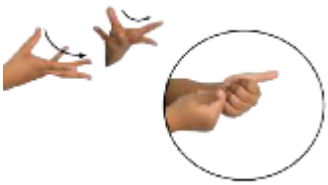
A.3. Dirección del movimiento de la mano

Las direcciones que puede realizar una mano durante una seña son: movimiento hacia arriba, hacia abajo, hacia la izquierda, hacia la derecha, hacia el frente y hacia atrás. También se incluyen los movimientos compuestos a partir de éstas, llamados movimientos complejos. Los movimientos con su respectiva ilustración se pueden ver en la Tabla A.3.

Movimiento	Descripción del movimiento	Imagen
Movimiento lineal o recto	Este movimiento consiste en hacer un movimiento recto en cualquier dirección.	

Movimiento en forma de arco	Este movimiento consiste en realizar un arco al mover las manos, brazos o la muñeca.	
Extensión de dedos	Se extienden los dedos.	
Vaivén	Se mueven de manera alternada los brazos o las manos.	
Circular	Las manos, los brazos o la muñeca se mueven de manera circular, generando un círculo o un semicírculo.	
Espirales	La mano o el brazo sigue una trayectoria de movimientos circulares.	
Flexión de dedos	Los dedos se retraen.	
Ondulante	El movimiento de las manos o brazos es ondular.	

Salto	La mano o dedos simulan saltos, ya sea uno o varios.	
Movimiento vibratorio local	La mano tiembla.	
Cabeceo de muñeca	La mano se mueve únicamente flexionando la muñeca.	
Aplanado	Se tocan y separan los dedos índice medio o índice y pulgar.	
Apulgarado	El dedo índice o medio se extiende al estar tocando al pulgar.	
Cambios progresivos en los dedos	Se mueven a los dedos de manera alternada.	

Deslizamiento	Los dedos se deslizan sobre el pulgar.	
Zigzag	El dedo índice realiza una trayectoria de la Z.	
Siete	La mano realiza una trayectoria del número 7.	
Rotación de muñeca	La muñeca o antebrazo giran y cambia la orientación de la mano.	
Choque	Las manos chocan.	
Doblar	El pulgar se mantiene inmóvil mientras los otros dedos se cierran contra la palma.	
Cruzado	Los antebrazos se cruzan hacia el centro y las manos llegan a un punto cercano.	

Simétrico	Las manos parten de un mismo punto y luego se abren hacia abajo, hacia arriba o hacia los lados.	
Prensar	Los dedos índice y pulgar sujetan la otra mano u otra parte del cuerpo.	

Tabla A.3: Movimientos de las manos al realizar las señas. Imagen tomada de [19]

Apéndice B

Segmentación del conjunto de datos

La segmentación utilizada para el entrenamiento, la validación y prueba de los modelos es la siguiente:

Seña	**#Entrenamiento**	**#Prueba**	**#Validación**	**#Total**
Museo	80	10	10	100
Arriba	80	10	10	100
Mundo	80	10	10	100
Mar	80	10	10	100
Ustedes	80	10	10	100
Bienvenido	80	10	10	100
Ayuda	80	10	10	100
Ver	80	10	10	100
Sur	80	10	10	100
Jueves	80	10	10	100
Dos	80	10	10	100
Mañana	80	10	10	100
Madera	80	10	10	100
Hoy	120	15	15	150
Rojo	80	10	10	100
Saber	80	10	10	100
Nosotros	80	10	10	100
Río	80	10	10	100
Siete	80	10	10	100
Gracias	80	10	10	100
Casa	80	10	10	100
Piñata	80	10	10	100
Sábado	80	10	10	100
Maestro	80	10	10	100
Música	80	10	10	100
Nombre	80	10	10	100
Jabón	80	10	10	100

Sol	80	10	10	100
Ocho	80	10	10	100
Pueblo	80	10	10	100
Leer	80	10	10	100
Mejor	124	13	13	130
Número	80	10	10	100
Niño	80	10	10	100
Diez	80	10	10	100
Mil	120	15	15	150
Hola	80	10	10	100
Todo	80	10	10	100
Noche	80	10	10	100
Semana	80	10	10	100
Piedra	80	10	10	100
Año	80	10	10	100
Tres	80	10	10	100
México	120	15	15	150
CInco	80	10	10	100
País	80	10	10	100
Arte	80	10	10	100
Agua	80	10	10	110
Cero	80	10	10	100
Feliz	80	10	10	100
Guerra	80	10	10	100
Norte	80	10	10	100
Mano	120	15	15	150
Ave	80	10	10	100
Veinte	80	10	10	100
Uno	80	10	10	100
Amigo	80	10	10	100
Martes	80	10	10	100
Cuatro	80	10	10	100
Historia	128	16	16	160
Tiempo	88	11	11	110
Familia	80	10	10	100
Vida	80	10	10	100
Árbol	80	10	10	100
Libro	80	10	10	100
Belleza	80	10	10	100
Trabajo	72	9	9	90
Miedo	80	10	10	100
Escuela	80	10	10	100

Iglesia	80	10	10	100
Verde	80	10	10	100
Miércoles	80	10	10	100
Otros	80	10	10	100
Azul	80	10	10	100
Siempre	80	10	10	100
Domingo	80	10	10	100
Como	80	10	10	100
Jugar	80	10	10	100
Edificio	80	10	10	100
Mismo	160	20	20	200
Lunes	80	10	10	100
Viernes	80	10	10	100
Veracruz	80	10	10	100
Conocer	80	10	10	100
Treinta	80	10	10	100
TOTAL	**7120**	**890**	**890**	**8900**

Tabla B.1: Elementos que componen a los diferentes subconjuntos utilizados en el trabajo

Bibliografía

[1] Censo de población y vivienda 2010. https://www.inegi.org.mx/programas/ccpv/2010/. Accessed: 2022-09-12.

[2] Censo de población y vivienda 2020. https://www.inegi.org.mx/programas/ccpv/2020/. Accessed: 2022-09-12.

[3] Convolutional neural networks (cnn): Step 2 - max pooling. Accessed: 2019-02-21.

[4] M. A. Ahmed, B. B. Zaidan, A. A. Zaidan, M. M. Salih, and M. M. B. Lakulu. A Review on Systems-Based Sensory Gloves for Sign Language Recognition State of the Art between 2007 and 2017. *Sensors (Basel)*, 18(7), Jul 2018.

[5] Samuel Albanie, Gül Varol, Liliane Momeni, Triantafyllos Afouras, Joon Son Chung, Neil Fox, and Andrew Zisserman. BSL-1K: scaling up co-articulated sign language recognition using mouthing cues. *CoRR*, abs/2007.12131, 2020.

[6] V. Athitsos, C. Neidle, S. Sclaroff, J. Nash, A. Stefan, Quan Yuan, and A. Thangali. The american sign language lexicon video dataset. In *CVPRW*, 2008.

[7] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. January 2015. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

[8] Sven Bambach, Stefan Lee, David J. Crandall, and Chen Yu. Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[9] Pooja Bhoir, Dr. Anil V. Nandyhyhh, D. Bormane, and Prof. Rajashri R. Itkarkar. Sign language recognition using hidden markov model. 2015.

[10] Sai Charan Bodda, Palki Gupta, Gaurav Joshi, and Ayush Chaturvedi. A new architecture for hand-worn sign language to speech translator. *CoRR*, abs/2009.03988, 2020.

[11] Necati Cihan Camgoz, Simon Hadfield, Oscar Koller, Hermann Ney, and Richard Bowden. Neural sign language translation. In *CVPR*, 2018.

[12] Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, and Richard Bowden. Multi-channel transformers for multi-articulatory sign language translation. In *ECCV 2020 Workshops*. Springer, Cham, August 2020.

[13] Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, and Richard Bowden. Sign language transformers: Joint end-to-end sign language recognition and translation, 2020.

[14] Necati Cihan Camgöz, Oscar Koller, Simon Hadfield, and Richard Bowden. Sign language transformers: Joint end-to-end sign language recognition and translation. *CoRR*, abs/2003.13830, 2020.

[15] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*, 2018.

[16] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields, 2018.

[17] Xiujuan Chai, Hanjie Wang, and Xilin Chen. The devisign large vocabulary of chinese sign language database and baseline evaluations. *Technical report VIPL-TR-14-SLR-001. Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS*, 2014.

[18] Wikimedia Commons. File:perceptron-unit.svg — wikimedia commons, the free media repository, 2020. [Online; accessed 7-September-2022].

[19] Gobierno de la Ciudad de México. *Diccionario de Lengua de Señas Mexicana LSM Ciudad de México*. Gobierno de la Ciudad de México, Ciudad de México, México, 2017.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[22] Mahmoud Elmezain, Ayoub Al-Hamadi, Gerald Krell, Sherif El-Etriby, and Bernd Michaelis. Gesture recognition for alphabets from hand motion trajectory using hidden markov models. In *2007 IEEE International Symposium on Signal Processing and Information Technology*, pages 1192–1197, 2007.

[23] MICHAEL ERARD. Why sign-language gloves don't help deaf people, nov 2017.

[24] Gary J. Grimes. Digital data entry glove interface device, 1981.

[25] Daniel Hernández Peña. Reconocimiento de señas con movimiento en el lenguaje de señas mexicano, 2021.

[26] Jie Huang, Wengang Zhou, Qilin Zhang, Houqiang Li, and Weiping Li. Video-based sign language recognition without temporal segmentation. In *AAAI*, 2018.

[27] Hamid Reza Vaezi Joze and Oscar Koller. MS-ASL: A large-scale data set and benchmark for understanding american sign language. In *BMVC*, 2019.

[28] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[29] Sang-Ki Ko, Chang Jo Kim, Hyedong Jung, and Choongsang Cho. Neural sign language translation based on human keypoint estimation, 2019.

[30] Sang-Ki Ko, Chang Jo Kim, Hyedong Jung, and Choongsang Cho. Neural sign language translation based on human keypoint estimation. *Applied Sciences*, 9:2683, 07 2019.

[31] Oscar Koller, Jens Forster, and Hermann Ney. Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers. *Computer Vision and Image Understanding*, 141:108–125, 2015.

[32] Dongxu Li, Cristian Rodriguez Opazo, Xin Yu, and Hongdong Li. Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison. In *WACV*, 2019.

[33] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[34] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[35] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer, 2021.

[36] Zhenguang Liu, Haoming Chen, Runyang Feng, Shuang Wu, Shouling Ji, Bailin Yang, and Xun Wang. Deep dual consecutive network for human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 525–534, June 2021.

[37] Syed Atif Mehdi and Yasir Niaz Khan. Sign language recognition using sensor gloves. pages 2204 – 2206 vol.5, 12 2002.

[38] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[39] Taro Miyazaki, Yusuke Morita, and Masanori Sano. Machine translation from spoken language to sign language using pre-trained language model as encoder. In *Proceedings of the LREC2020 9th Workshop on the Representation and Processing*

of Sign Languages: Sign Language Resources in the Service of the Language Community, Technological Challenges and Application Perspectives, pages 139–144, Marseille, France, May 2020. European Language Resources Association (ELRA).

[40] Martin Musiol. Speeding up deep learning computational aspects of machine learning, 01 2016.

[41] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018.

[42] OPS. La oms caracteriza a covid-19 como una pandemia, 2020.

[43] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[45] Lionel Pigou, Mieke Van Herreweghe, and Joni Dambre. Gesture and sign language recognition with temporal residual networks. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 3086–3093, 2017.

[46] Lucas Rioux-Maldague and Philippe Giguère. Sign language fingerspelling classification from depth and color images using a deep belief network. In *2014 Canadian Conference on Computer and Robot Vision*, pages 92–97, 2014.

[47] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[48] Adam Schembri, Jordan Fenlon, Ramas Rentelis, Sally Reynolds, and Kearsy Cormier. Building the British sign language corpus. *Language Documentation & Conservation*, 7:136–154, 2013.

[49] SEP. Aprende en casa, 2020.

[50] Ala Sidig, Hamzah Luqman, and Sabri Mahmoud. Arabic sign language recognition using optical flow-based features and hmm. 05 2018.

[51] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation, 2019.

[52] Hamid Vaezi Joze and Oscar Koller. Ms-asl: A large-scale data set and benchmark for understanding american sign language. In *The British Machine Vision Conference (BMVC)*, September 2019.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[54] Ville Viitaniemi, Tommi Jantunen, Leena Savolainen, Matti Karppa, and Jorma Laaksonen. S-pot – a benchmark in spotting signs within continuous signing. In *LREC*, 2014.

[55] U. von Agris, M. Knorr, and K. Kraiss. The significance of facial features for automatic sign language recognition. In *2008 8th IEEE International Conference on Automatic Face Gesture Recognition*, 2008.

[56] R. B. Wilbur and A. C. Kak. Purdue RVL-SLLL American sign language database. *School of Electrical and Computer Engineering Technical Report, TR-06-12, Purdue University, W. Lafayette, IN 47906.*, 2006.

[57] Yao-Yuan Yang, Moto Hira, Zhaoheng Ni, Anjali Chourdia, Artyom Astafurov, Caroline Chen, Ching-Feng Yeh, Christian Puhrsch, David Pollack, Dmitriy Genzel, Donny Greenberg, Edward Z. Yang, Jason Lian, Jay Mahadeokar, Jeff Hwang, Ji Chen, Peter Goldsborough, Prabhat Roy, Sean Narenthiran, Shinji Watanabe, Soumith Chintala, Vincent Quenneville-Bélair, and Yangyang Shi. Torchaudio: Building blocks for audio and speech processing. *arXiv preprint arXiv:2110.15018*, 2021.

[58] Yuancheng Ye, Yingli Tian, Matt Huenerfauth, and Jingya Liu. Recognizing american sign language gestures from within continuous videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2145–214509, 2018.

[59] Kayo Yin. Sign language translation with transformers. *CoRR*, abs/2004.00588, 2020.

[60] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *CoRR*, abs/2106.11342, 2021.