



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

RECONOCIMIENTO DE SEÑAS CON MOVIMIENTO EN EL LENGUAJE
DE SEÑAS MEXICANO

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

DANIEL HERNÁNDEZ PEÑA

T U T O R

DRA. VERÓNICA ESTHER ARRIOLA RÍOS



Ciudad Universitaria, CD. MX., 2021



Universidad Nacional
Autónoma de México

Dirección General de Bibliotecas de la UNAM

Biblioteca Central



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

*A mi familia y amistades que me apoyaron durante la realización
de este proyecto ...*

Agradecimientos

A la Dra. Verónica Esther Arriola Ríos por guiarme y darme las herramientas para realizar este proyecto.

A Araceli, Moises, Alexia, Héctor, Juan Guillermo, Claudia, Beatriz Ulloa, Alejandra, Martín, Ximena, Cassandra por ayudarme a grabar los videos que fueron la base de este proyecto.

A Erika Lizbeth y Adriana Yazmín por ayudarme e inspirarme a sobrepasar los problemas que llegué a tener.

A mi familia por darme las fuerzas para seguir adelante con el proyecto y las problemáticas que trajo consigo.

Índice general

1. Introducción	1
2. Lengua de Señas Mexicana (LSM)	3
2.1. Modalidad	3
2.2. Gramática	4
2.3. Tipo de señas	4
2.4. Ubicación de las señas	4
2.5. Dirección del movimiento de la mano	6
3. Redes neuronales	12
3.1. Redes Neuronales	12
3.1.1. Perceptrón	12
3.1.2. Neurona sigmoide	13
3.2. Red neuronal con propagación hacia adelante	13
3.2.1. Sesgo	14
3.3. Entrenamiento de una red neuronal con propagación hacia adelante	17
3.3.1. Inicializando los pesos	17
3.3.2. Vector gradiente	17
3.3.3. Función de error	17
3.3.4. Descenso por el gradiente	18
3.3.5. Propagación hacia atrás	18
3.4. Descenso por el gradiente estocástico	20
3.5. Regularización en redes neuronales	21
3.5.1. Regularización de parámetro L^2	21
3.5.2. Regularización L^1	21
3.5.3. Aumento de información	21
3.5.4. Early Stopping	22
3.5.5. Dropout	22
3.5.6. Normalización en lotes	23
3.6. Red neuronal con propagación hacia adelante profunda	24
3.6.1. Arquitectura de un MLP	25
3.7. Redes Neuronales convolucionales	25

3.7.1.	Operador convolución	25
3.7.2.	Campos locales receptivos	26
3.7.3.	Pesos y sesgos compartidos	27
3.7.4.	Capas de agrupación	27
	Max pooling	28
3.8.	R-CNN	28
3.8.1.	Funcionamiento de R-CNN	30
3.9.	Fast R-CNN	31
3.9.1.	Funcionamiento de Fast R-CNN	32
3.10.	Redes Neuronales Residuales (ResNet)	32
3.11.	Faster R-CNN	33
3.11.1.	Redes de Propuesta de Región (RPN)	35
	Funcionamiento de una RPN	36
3.11.2.	Funcionamiento de Faster R-CNN	37
3.12.	Detección de Pose de una o varias personas	37
3.12.1.	Máquinas de Pose Inferenciales (PM)	37
	Jerarquía de detección	38
	Características de contexto	39
3.12.2.	Máquinas de Pose Convolucionales (CPM)	40
	Funcionamiento de una CPM	40
	Ubicación de partes claves usando la evidencia local de imagen	41
	Predicción secuencial con características de contexto espaciales aprendidas	41
3.12.3.	Estimación de pose 2D en tiempo real multipersonas usando Campos de Afinidad de Partes (PAF)	42
	Detección y Asociación Simultánea	43
	Mapas de confianza para la detección de partes	43
	PAFs por asociación de partes	44
3.12.4.	Openpose	45
3.12.5.	Funcionamiento de OpenPose	47
3.13.	Redes Neuronales Recurrentes (RNN)	47
3.13.1.	Redes LSTM	50
	Funcionamiento de LSTM	50
	Entrenamiento de LSTM	52
4.	Clasificación de series de tiempo	54
4.1.	Dynamic Time Warping (DTW)	54
4.2.	FastDTW	56
4.2.1.	Algoritmo FastDTW	57
4.3.	K vecinos cercanos	58
4.3.1.	Matriz de confusión	60
5.	Experimentación	61
5.1.	Primer Experimento: Faster R-CNN + KNN	61

5.1.1. Entrenamiento del detector	61
5.1.2. Entrenando al clasificador	67
Trayecto puro	68
5 Vecinos Cercanos	69
5.2. Pruebas	73
5.2.1. Interpretación de resultados	73
5.2.2. Conclusión	75
5.3. Segundo Experimento: OpenPose + LSTM	75
5.3.1. Detección de puntos clave por medio de OpenPose	75
Grabación y manipulación de Videos	77
5.3.2. Preparación de los datos	78
5.3.3. Red LSTM	79
5.3.4. Entrenamiento de la red	81
5.3.5. Resultados	83
5.3.6. Interpretación de los resultados	83
5.3.7. Experimento Extra	85
5.3.8. Conclusión del experimento Extra	85
6. Conclusión	91
Bibliografía	93

Capítulo 1

Introducción

Desde mediados de 1900 se han empezado a idear formas de comunicarse con tecnología por medio de gestos y/o al oprimir botones que alimentan instrucciones a una computadora. De las primeras ideas introducidas en esa época se destacan el uso de guantes, los cuales, por medio de sensores, identifican los movimientos de las manos para traducir el significado del movimiento que realizan. Uno de los primeros guantes, patentado en 1983 por Gary Grimes perteneciente a los *Bell Labs*, se puede observar la patente en 1.1. A lo largo del tiempo se siguieron presentando guantes capaces de interpretar algún tipo de comandos hacia un computador, entre estos se encuentran el DataGlove (1987), Power Glove (1989), CyberGlove(1990), entre otros. Si bien varios tenían una funcionalidad inicial básica, éstos se fueron adaptando para lograr un reconocimiento de lenguaje de señas.

No obstante, estos guantes siempre han sido ineficientes para poder traducir el lenguaje de señas al lenguaje hablado, dado que únicamente es capaz de saber los rasgos manuales de la seña, i.e el movimiento de la mano, sin poner a contexto los rasgos no manuales del señante, tales como emociones o expresiones faciales que se presentan al interpretar una seña. Otro problema que surge es que realizan una traducción literal del idioma hablado al idioma de seña, cuando cada lengua de señas posee su propia lingüística y obliga al señante a cambiar su forma de interpretar las señas. Finalmente el uso de guantes para la interpretación de señas resulta ser un problema al momento de la comunicación, esto dado que el señante es obligado a usar el guante, y los accesorios necesarios, para que funcione correctamente, lo cual incomoda al señante, dado que preferiría, en ese caso, usar un lenguaje escrito para la comunicación.[2]

Al mismo tiempo que se desarrollaba la idea de los guantes, se empezó a lograr la detección de objetos dentro de imágenes, siendo en 2001 el primer detector de rostro de una persona[3][4], el cual usaba una ventana deslizante para analizar la imagen y clasificar objetos en caso de encontrarlos, estos detectores han ido mejorando de manera considerable, siendo capaces de detectar objetos donde hay más ruido alrededor de éstos. Ante esto se empiezan a generar detectores específicos de mano, para poder reconocer la posición de la mano dado una seña realizada, sacrificando la cantidad de detalles obtenidos por el guante pero haciendo el proceso más sencillo de realizar al no necesitar accesorios extra para el envío de señales. Entre estas

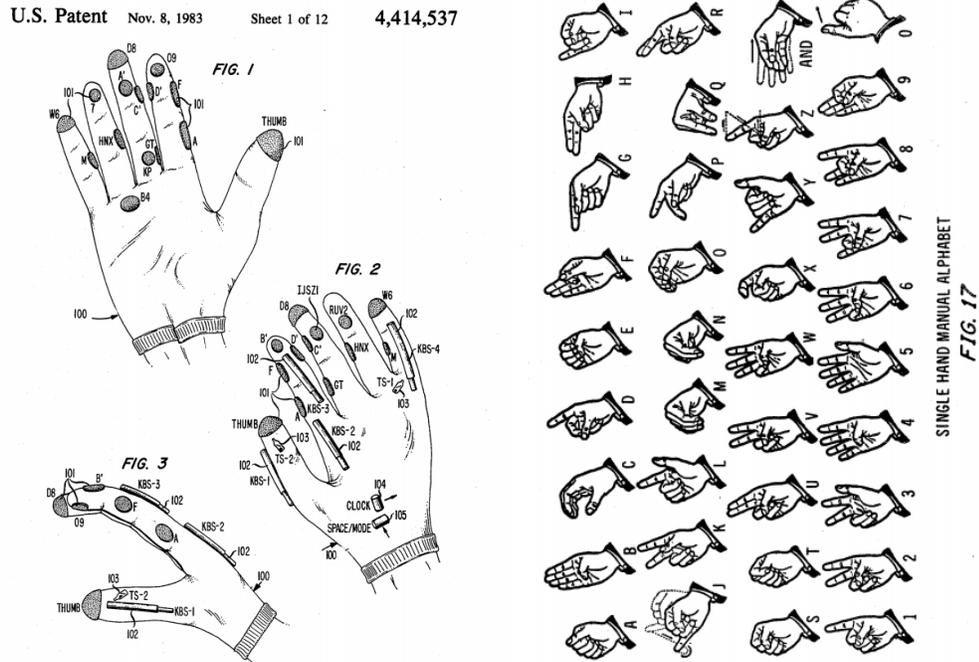


Figura 1.1: Patente introducida en 1983 por Gary Grimes junto al alfabeto capaz de identificar. Imagen tomada de la patente[1]

nuevas formas destaca la forma de usar una cámara web para observar interpretación de las señas y, tras pasarlo por medio de un *algoritmo de detección de manos*, regresa su respectiva interpretación, algunos ejemplos de esto se pueden ver en [5] [6] y [7].

En base a estos avances, presentamos un sistema, basado en un experimento de una universidad coreana[7], el cual permite la interpretación de 28 señas del lenguaje de señas mexicano (LSM). Si bien el sistema no es tan complejo, éste representa un primer paso para un sistema más extenso para clasificar todas las señas dentro del LSM, previa recopilación de un conjunto de datos de entrenamiento con dichas señas.

Para lograr este sistema, primero se hace una introducción al LSM, donde se muestran sus características principales, las cuales son de utilidad para poder diferenciar entre las distintas señas que se diferenciarán. Después se introducirán las redes neuronales, donde se mostrarán dos tipos de redes neuronales: Convolucionales y Recurrentes; las convolucionales nos servirán para sistemas de detección de los movimientos de la seña y las recurrentes nos ayudarán a encontrar patrones dentro de una secuencia. También se introducirá una medida de distancia para poder medir la diferencia de posición entre ambas manos para poder agrupar entre las distintas posiciones y vincularlas a alguna seña. Finalmente se realizan dos experimentos, uno por cada sistema de detección de manos, exponiendo los resultados y conclusiones de cada uno, para mostrar la conclusión final de la tesis junto a un plan a futuro para mejorar el sistema.

Capítulo 2

Lengua de Señas Mexicana (LSM)

De acuerdo al Diario Oficial de la Federación (DOF), una lengua de señas se define de la siguiente manera:

“Lengua de una comunidad de sordos, que consiste en una serie de signos gestuales articulados con las manos y acompañados de expresiones faciales, mirada intencional y movimiento corporal, dotados de función lingüística, forma parte del patrimonio lingüístico de dicha comunidad y es tan rica y compleja en gramática y vocabulario como cualquier lengua oral” [8].

En base a esta definición, se entiende que la LSM es una lengua que, de igual forma que las lenguas habladas, posee una gramática, modalidad, reglas, entre otras cosas, que la identifican, para el alcance de la tesis definiremos brevemente algunas de las características de la lengua de señas.

2.1. Modalidad

La modalidad se define de la siguiente manera por la Real Academia Española (RAE): “Expresión de la actitud del hablante en relación con el contenido de lo que se comunica.” [9].

Es decir, la modalidad es cómo se percibe y expresa una lengua. Para el caso de las lenguas habladas, la modalidad es auditiva-verbal, que se va manifestando en el tiempo, por otro lado, en el caso de la LSM, la modalidad es visual-gestual-manual, que se va manifestando en el tiempo y en el espacio.

La lengua de señas se caracteriza por ser comunicada a través de las manos, la cara y el cuerpo, logrando la comunicación por medio de medios visuales, auxiliándose con expresiones faciales y corporales para comunicar algo más allá de la palabra.

2.2. Gramática

La gramática se define de la siguiente manera por la RAE: “Parte de la lingüística que estudia los elementos de una lengua, así como la forma en que estos se organizan y se combinan.” [10].

Aplicada a la lengua de señas, la gramática es la forma en que se enlazan los elementos de la lengua (señas) para formar discursos, la gramática se estructura por medio del espacio y del movimiento, donde, para la LSM, estos límites se definen como el límite vertical, el límite horizontal y un límite que indica la distancia entre las manos y el cuerpo. Estos límites ayudan a definir el marco de la comunicación y, en caso de que se vean sobrepasados, se entiende como que dicha seña conlleva un énfasis o exageración.

2.3. Tipo de señas

Las señas que realiza un señante se pueden separar en cuatro categorías, las cuales son de acorde a cuántas manos se usa para la seña y si los movimientos son simultáneos, estas categorías se pueden ver en la tabla 2.1.

Tabla 2.1: Posibles tipo de seña

Tipo de Seña	Definición
Seña manual (SM)	Seña que se articula con sólo una mano
Seña bimanual (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

2.4. Ubicación de las señas

La ubicación se entiende por el espacio en donde se realizan las señas, tomando en cuenta los límites definidos en la gramática, en particular se hace énfasis a la ubicación referente a la cara, cuerpo, brazos, manos y en la ubicación con respecto al cuerpo. Las ubicaciones se

pueden ver en la tabla 2.2.

Tabla 2.2: Ubicación de las señas. Tomada de [11]

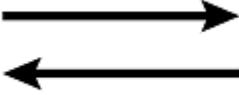
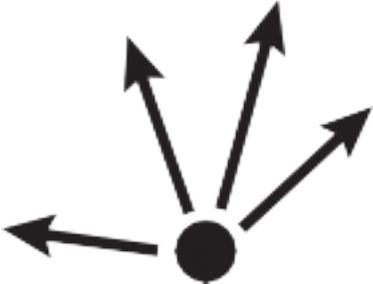
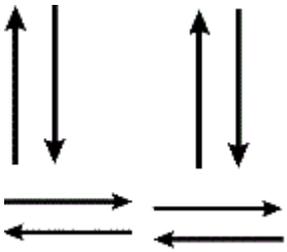
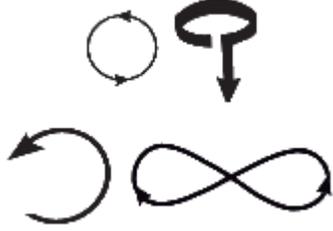
	Imagen lateral de la ubicación	Cara	Cuerpo	Brazos	Manos
1					
2					
3					
4					
5					
6					



2.5. Dirección del movimiento de la mano

Entre las direcciones básicas del movimiento de la(s) mano(s) al realizar una seña, que incluyen el movimiento hacia arriba, hacia abajo, hacia la izquierda, hacia la derecha, hacia al frente, hacia detrás y los movimientos que se pueden componer con éstos, también se encuentran los siguientes, que suelen ser movimientos más complejos. Los posibles movimientos se encuentra en [2.3](#).

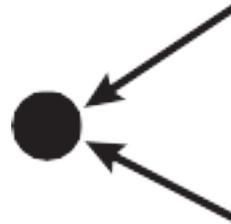
Tabla 2.3: Movimientos de las manos al realizar las señas. Tomada de [11]

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 mueve de manera circular, generando un círculo o un semicírculo.	

Espirales La mano o el brazo siguen una trayectoria de movimientos circulares.



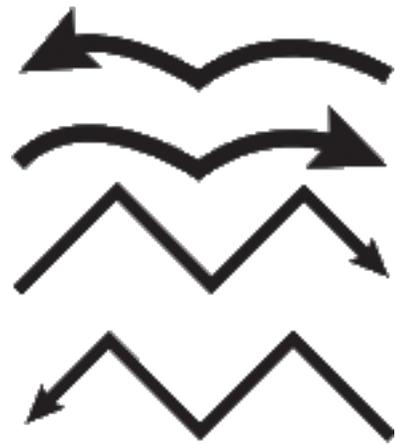
Flexión de dedos Los dedos se retraen.



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 pasa 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.



Una vez teniendo esto, se puede hacer una clasificación de palabras en lengua de señas, siendo un paso antes de poder reconocer frases, dado que para reconocer frases se tiene que poder detectar los rasgos no manuales, los cuales involucran a la expresión facial y los gestos y son más sutiles que el movimiento de manos.

Capítulo 3

Redes neuronales

En este capítulo se introducen distintas redes neuronales, desde las redes más básicas, que únicamente consisten en un perceptrón, hasta las redes LSTM, las cuales nos permiten el aprender secuencias con dependencias de largo alcance, el objetivo del capítulo consiste en lograr identificar objetos en una imagen, en nuestro caso las manos de una persona, para finalmente clasificar videos de señas.

Para las redes en general se toma de base a [12] complementando con [13], para las redes convolucionales se toma de base a [14] complementando con [15], para los métodos RCNN, Fast-RCNN y Faster-RCNN se utilizan como base [16–18], para la sección de detección de una o más personas se utilizan como base [19–22] respectivamente, y para las RNN y LSTM se utiliza de base a [23] complementando con [24, 25]

3.1. Redes Neuronales

3.1.1. Perceptrón

Una red neuronal es un tipo de sistema computacional, inspirado en las redes neuronales biológicas. Estos sistemas aprenden a realizar tareas a partir de ejemplos que son alimentados a estas redes.

La red neuronal más sencilla se compone a partir de unidades llamadas **perceptrones**, las cuales son clasificadores lineales binarios, i.e. una combinación lineal entre las entradas, donde la salida pertenece al conjunto $(0, 1)$, y se le aplica la función piso a la salida de la combinación para poder obtener la salida binaria. Un perceptrón se puede visualizar en la figura 3.1.

Para evitar que la salida del perceptrón siempre sea la misma se introduce a la unidad, llamada peso, la cual tiene como función indicar la importancia de la entrada para la salida, estos pesos se denotan por w_i y su dominio son los \mathbb{R} . El efecto que tienen los pesos en las entradas se puede ver en la ecuación 3.1.

$$\text{Salida} = \begin{cases} 0 & \text{si } \sum_j w_j x_j \leq \text{límite} , \\ 1 & \text{si } \sum_j w_j x_j > \text{límite} \end{cases} \quad (3.1)$$

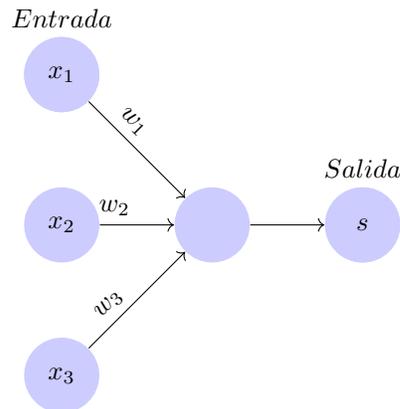


Figura 3.1: Un perceptrón toma como entrada x_1, x_2, x_3 y produce una única salida s .

3.1.2. Neurona sigmoide

Una desventaja del perceptrón viene siendo que se dificulta la modificación de los pesos dado que un pequeño ajuste puede causar que cambie la salida por completo, y dado que únicamente hay dos posibles valores, no es lo más óptimo para estructuras más complejas. Para ello se introduce una nueva operación que reemplazará a la operación piso, la función sigmoide, definida en la ecuación 3.2, con su aplicación en el perceptrón sigmoide en la ecuación 3.3, con una salida dentro del conjunto $[0, 1)$. A los perceptrones con la función sigmoide se les conoce como perceptrón sigmoide o neurona sigmoide y la diferencia entre ambas funciones se puede visualizar en la figura 3.2.

$$\sigma \equiv \frac{1}{1 + e^{-z}} \quad (3.2)$$

$$\frac{1}{1 + \exp\left(-\sum_j w_j x_j - b\right)} \quad (3.3)$$

3.2. Red neuronal con propagación hacia adelante

Teniendo neuronas más complejas, surge la necesidad de poder juntarlas con otras neuronas para lograr aproximar de mejor manera a los resultados deseados, y lograr representar funciones más complejas. Logramos esto al hacer las **redes neuronales con propagación hacia adelante**, que indica la forma en que las redes se comunicarán entre ellas, que es propagando la información hacia adelante, haciendo que la neurona que se encuentra más adelante tome como entrada la salida de las redes más al principio, la figura 3.3 muestra una red neuronal más compleja con propagación hacia adelante.

Para calcular la salida de toda la red, se utiliza la ecuación 3.3 para calcular la salida de cada neurona, empezando desde la capa de entrada, propagando los valores obtenidos hasta la capa

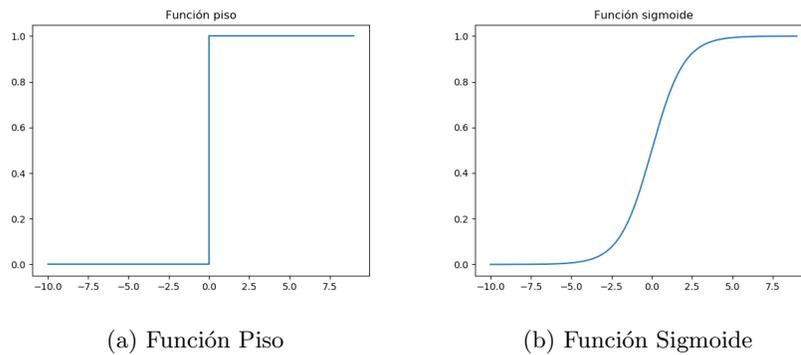


Figura 3.2: Funciones de activación

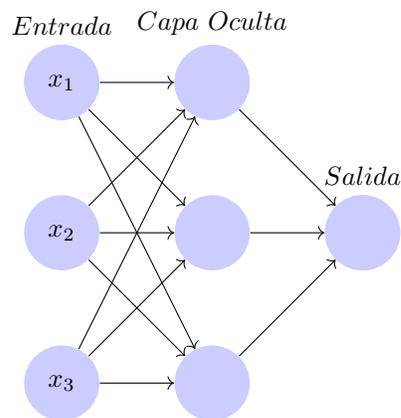


Figura 3.3: Red neuronal prealimentada con propagación hacia adelante

de salida.

3.2.1. Sesgo

Para mejorar el cálculo de las redes se utiliza un nuevo parámetro llamado sesgo, el cual nos permite cambiar el valor de la función hacia la izquierda o hacia la derecha, como se puede apreciar en la figura 3.4. El sesgo es una neurona extra dentro de una capa de la red con un valor inicial, normalmente 1, la cual se comporta como un nodo de la red, con la diferencia de que su salida no es afectada por los valores de la capa anterior. El rol del sesgo es poder ajustar de mejor manera a la red respecto a los datos del entrenamiento, que sucede al agregar este nuevo valor a los cálculos de la capa, provocando un ajuste al cálculo final. De igual forma que con los pesos, el sesgo se va ajustando conforme va aprendiendo la red. Generalmente en este tipo de redes neuronales, donde la información va hacia adelante, cada unidad es la suma

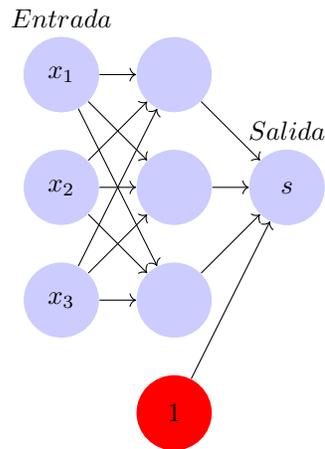
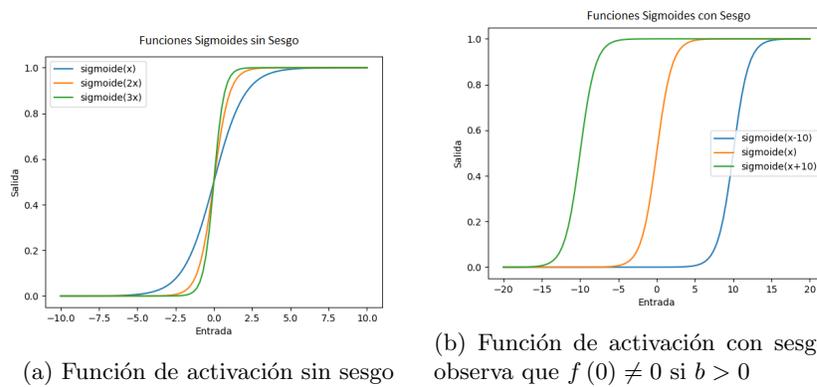


Figura 3.4: Sesgo en una red, denotado en la neurona roja, el cual ajusta el peso que conecta a la neurona x_1 con la salida s



(a) Función de activación sin sesgo

(b) Función de activación con sesgo, se observa que $f(0) \neq 0$ si $b > 0$

Figura 3.5: Uso de sesgo en la red

ponderada de la entrada, como se ve en la ecuación 3.4.

$$a_j = \sum_i w_{ji} z_i \quad (3.4)$$

Para cada elemento en el conjunto de entrenamiento se va aplicando la ecuación 3.4 y al elemento z_j , propagándose las activaciones de las neuronas hacia adelante, el procedimiento se repite hasta llegar a las neuronas en la capa de salida. Este proceso es llamado **propagación hacia adelante**, esto porque se van propagando los valores de todas las operaciones hacia adelante; la capa de salida.

Notación	Significado
w_{jk}^l	Peso en la conexión de la k -ésima neurona en la $(l - 1)$ -ésima capa hacia la j -ésima capa neurona
b_{jk}^l	Sesgo en la conexión de la k -ésima neurona en la $(l - 1)$ -ésima capa hacia la j -ésima capa neurona
a_j^l	Activación de la j -ésima neurona en la l -ésima capa.
z_j^l	La suma ponderada de las neuronas en la capa l
z_j^l	La suma ponderada de la función de activación de la neurona j en la capa l

Tabla 3.1: Notaciones a usar

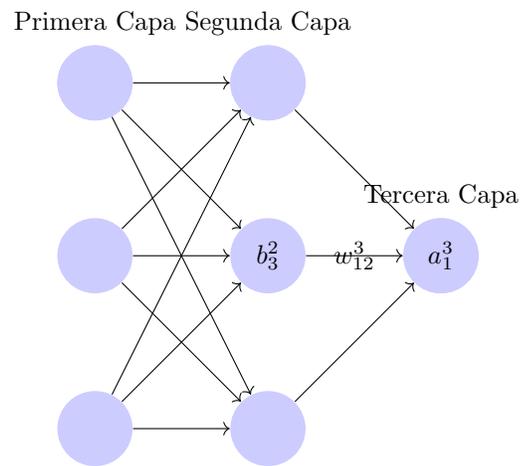


Figura 3.6: Diagrama de la red mostrando la notación a usar

3.3. Entrenamiento de una red neuronal con propagación hacia adelante

3.3.1. Inicializando los pesos

Viendo la estructura de la red definida en la figura 3.4, junto con las ecuaciones 3.1 y 3.2, podemos ver que la salida de la red es determinada tanto por los pesos como por los sesgos, pero generalmente la estructura de la red al ser definida con valores iniciales cualesquiera no aproxima las funciones que deseamos que realice, para ello se tiene que ir ajustando a la red acorde al problema, a esta acción se le conoce como **entrenamiento**, que consiste en ir ajustando a los pesos y sesgos para ir aproximando la salida deseada.

Para inicializar los pesos de la red, generalmente se toma una muestra aleatoria basada en la distribución Gaussiana con media cero y desviación estándar de uno.

3.3.2. Vector gradiente

El vector gradiente o velocidad de cambio de la función $f(x_1, \dots, x_n)$, denotado por ∇f , es un vector en donde se obtiene la dirección de mayor crecimiento de f . Para lograr esto, el vector gradiente tiene la siguiente definición:

$$\nabla f = \begin{bmatrix} \frac{\delta f}{\delta x_1} \\ \frac{\delta f}{\delta x_2} \\ \dots \\ \frac{\delta f}{\delta x_n} \end{bmatrix} \quad (3.5)$$

Teniendo que cada entrada del vector corresponde a la derivada parcial de f respecto a la entrada x_n . Este vector nos ayudará a obtener los pesos óptimos, que es donde la función de error, definida posteriormente, alcanza un mínimo.

3.3.3. Función de error

La función de error es la función que representa la diferencia entre los datos obtenidos por medio de la red y los datos reales. La función de error más usada es la función MSE, definida como:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \quad (3.6)$$

Donde w representa a los pesos, b a los sesgos que se tienen en la red, n el número de entradas de entrenamiento y a el vector obtenido en la salida de la red cuando x es la entrada.

Habiendo definido el gradiente y la función de error, tenemos que para poder entrenar a una

red (léase ajustar los pesos de la red) es necesario obtener el gradiente de la función de error, vista en la figura 3.7, e ir manipulando los pesos y sesgos de acorde el resultado. Este proceso es llamado **Descenso por el gradiente**.

$$\nabla C(w, b) = \begin{bmatrix} \frac{\partial C}{\partial w} \\ \frac{\partial C}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum_x x (y(x) - a) \\ \frac{1}{N} \sum_x (y(x) - a) \end{bmatrix} \quad (3.7)$$

3.3.4. Descenso por el gradiente

Para entender lo que se realiza durante el entrenamiento de la red, consideremos el gradiente de la función de error respecto a un peso de la red, el vector $\nabla f(w)$ nos indica los posibles escenarios:

- Si el gradiente es cero: El peso no contribuye al error en la red
- Si el gradiente es positivo: El peso debe disminuirse para poder disminuir el error en la red
- Si el gradiente es negativo: El peso debe aumentarse para poder disminuir el error en la red

En base a esto, lo que se busca es ver cómo modificar los pesos para lograr obtener el mínimo de la función de error a usar, en nuestro caso la función 3.6, esto se logra al aplicar las ecuaciones 3.8 y 3.9, en base al gradiente obtenido para actualizar los pesos y sesgos respectivamente, teniendo un nuevo valor, η , el cual es un hiperparámetro, generalmente con un valor entre 0.0 y 1.0, que indica a la red qué tan “rápido” se adaptará dados los resultados obtenidos, i.e qué tanto se ajustan los valores respecto al gradiente obtenido.

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \quad (3.8)$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \quad (3.9)$$

Al aplicar repetidamente esta actualización, los pesos y sesgos van “rodando hacia abajo”, encontrando el mínimo de la función de error, logrando los pesos logren aproximar a los resultados esperados.

3.3.5. Propagación hacia atrás

Una vez teniendo el vector gradiente de la salida de una neurona, es posible saber qué pesos, activaciones o sesgos incrementan al error de ésta, por lo que se busca una forma de modificar estos valores y así disminuir el error. Pero una red neuronal se compone de varias neuronas que tienen conexiones con otras neuronas, las cuales se encuentran en distintas capas de la red, por lo que un ajuste de una neurona en la capa i afectará a las neuronas de la capa j con $j > i$, tomando esto en cuenta se introduce el algoritmo de **propagación hacia atrás**.

El algoritmo de propagación hacia atrás, también llamado **backdrop** o **propagación del error**, se compone de dos etapas: La primer etapa consiste en calcular las activaciones de cada neurona, por medio de la ecuación 3.4, e ir propagando las activaciones hasta llegar a la capa de salida. La segunda etapa consiste en la actualización de pesos y sesgos en base al vector gradiente, el cual se va calculando al ir propagando hacia atrás los errores de la red, esto por medio de las ecuaciones 3.8 y 3.9.

Teniendo esto, retomemos el calculo del vector gradiente del error en la neurona n , E_n , respecto al peso w_{ji} , el cálculo de este vector dependerá de los resultados de las $n-1$ neuronas anteriores, por lo que dado un cambio, sin importar la magnitud de éste, en la activación de la neurona $n-1$ influye directamente en la activación de la neurona n y, por consecuente, en su error. En base a este pensamiento, tenemos que E_n depende en el peso w_{ji} sólo por medio de la suma ponderada a_j hacia la unidad j , por lo que podemos aplicar la regla de la cadena a E_n y obtenemos lo siguiente:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{a_j} \frac{\partial a_j}{\partial w_{ji}} \quad (3.10)$$

También se introducirá una nueva notación, la cual se refiere al error de la neurona j en la capa l , denotada por δ_j^l :

$$\delta_j^l \equiv \frac{\partial E_n}{\partial a_j^l} \quad (3.11)$$

Usando esta ecuación sustituimos en la ecuación 3.4 y podemos escribir:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \quad (3.12)$$

Sustituyendo las ecuaciones 3.11 y 3.12 en la ecuación 3.10 obtenemos:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \quad (3.13)$$

Podemos observar que lo que la ecuación nos dice es que el vector gradiente es obtenido al multiplicar el error de la capa j por la activación de la capa i , esto facilita de gran forma el cálculo de el gradiente, sería cuestión de calcular los errores de las capas. Para poder calcular los errores, retomamos donde se detiene el algoritmo de propagación hacia adelante, en la capa de salida, se puede definir lo siguiente:

$$\delta_j = y(x_j) - z_j \quad (3.14)$$

Donde $y(x_j)$ es la salida esperada en la neurona j , y a la salida obtenida en la neurona j . Para ir evaluando los errores δ en las capas ocultas, se utiliza la regla de la cadena de la siguiente manera:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad (3.15)$$

Donde se suma a través de las unidades k , las cuales la unidad j tiene conexiones, al sustituir la ecuación 3.11 en la ecuación 3.15 y, usando la ecuación 3.4, obtenemos la siguiente fórmula,

conocida como la fórmula de propagación hacia atrás:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad (3.16)$$

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

Una vez obtenidos los errores, se aplica recursivamente la ecuación 3.16 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 perteneciente al conjunto de entrenamiento 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 3.14.
- Una vez obtenidos los errores, éstos se propagan hacia atrás usando la ecuación 3.16, esto para cada neurona en la capa oculta de la red.
- Se utiliza la ecuación 3.13 para calcular las derivadas requeridas, agilizando el cálculo
- Se actualizan los pesos y sesgos utilizando las ecuaciones 3.9 y 3.8

Si bien propagación hacia atrás junto con descenso por el gradiente logra entrenar la red, el proceso puede llegar a ser un procedimiento lento ya que depende de la dimensión del conjunto de entrenamiento, la estructura de la red, entre otros factores.

3.4. Descenso por el gradiente estocástico

Un problema que surge al entrenar redes es el problema del gradiente explosivo y el gradiente desvaneciente, estos problemas hacen difícil saber hacia qué dirección deberían moverse los parámetros, dado que uno hace que el gradiente crezca de manera exponencial y el otro hace que el gradiente se vuelva tan pequeño que el gradiente se estanca y la red deja de aprender. Para combatir estos problemas se introduce al descenso estocástico por el gradiente (SGD), que incrementa la velocidad de entrenamiento e introduce un nuevo parámetro de ruido que no se desvanece al llegar a un mínimo. SGD funciona tomando un subconjunto de los gradientes a calcular y toma el promedio de sus gradientes, i.e

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^m \nabla C_{X_j} \quad (3.17)$$

Utilizando estas aproximaciones, podemos cambiar las reglas de actualización a lo siguiente:

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \quad (3.18)$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l} \quad (3.19)$$

Este proceso es repetido hasta utilizar todas las entradas originales de entrenamiento, esto es llamado completar una época de entrenamiento.

3.5. Regularización en redes neuronales

Para entrenar a una red se utiliza al conjunto de entrenamiento, pero para probar qué tan buena es la aproximación a la función es necesario introducir dos conjuntos más, de lo contrario la red únicamente aprende a dar de salida al conjunto de entrenamiento, estos conjuntos son el conjunto de **prueba**, conjunto para optimizar al error de la red, y el conjunto de **validación**, conjunto para evaluar el error en las predicciones. Pero estos nuevos conjuntos no son la única forma de optimizar o evaluar a una red, también se usa un procedimiento llamado **regularización**. Se entiende por regularización al proceso de ajustar al algoritmo de entrenamiento para lograr disminuir el error de generalización, esto es disminuir el error dentro del conjunto de prueba, pero cuidando de no afectar al error de entrenamiento. Esto se logra al añadir restricciones al modelo o añadiendo términos extras a la función de error. Las formas más comunes de regularización son las siguientes:

3.5.1. Regularización de parámetro L^2

La regularización L^2 , también llamada *weight decay* o decaimiento del error, consiste en acercar los pesos cerca al origen, esto por medio de agregarle el parámetro de normalización $w(\theta) = \frac{1}{2}\|w\|_2^2$ a la función de error, es decir:

$$E = E_0 + \frac{\lambda}{2n} \sum_w w^2 \quad (3.20)$$

Donde E_0 representa la función de coste original no regularizada y λ el parámetro de regularización, i.e qué tanto queremos se regularicen los pesos.

3.5.2. Regularización L^1

También está la regularización L^1 , donde el parámetro de regularización se define por $w(\theta) = \|w\|_1 = \sum_i |w|$, teniendo a la función de error como:

$$E = E_0 + \frac{\lambda}{n} \sum_w \|w\| \quad (3.21)$$

Donde E_0 representa la función de error original no regularizada y λ el parámetro de regularización.

3.5.3. Aumento de información

No todas las formas de regularización involucran una modificación a la función de error, uno de estos métodos es el aumento de información. Como su nombre lo indica, este método

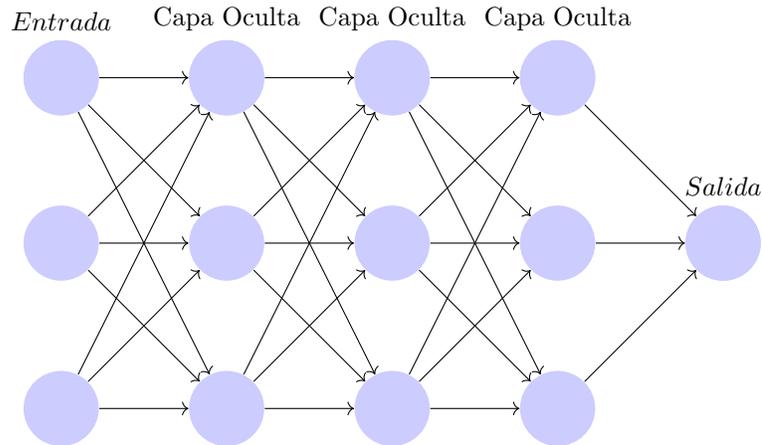


Figura 3.7: Red completa

consiste en aumentar la información que se le proporciona a la red, aunque, en la práctica, la cantidad de información que se tiene es limitada. Para nuestros usos, reconocimiento de objetos en una imagen, este problema puede ser atacado al crearse información artificial, esto dado que una imagen que contiene a un objeto no deja de contener a un objeto si la imagen es rotada o se le aplica un filtro, estas operaciones no modifican al objeto y ayudan a aumentar la generalización al momento del reconocimiento.

3.5.4. Early Stopping

Al entrenar modelos que pueden tener problemas de sobre ajuste (*overfitting*), en éstos se observa que el error de entrenamiento disminuye conforme se va entrenando, pero el error en el conjunto de validación empieza a aumentar nuevamente, esto significa que se puede tener un modelo con un mejor error de validación al regresar los parámetros a un estado anterior, cuando se tenía un error de entrenamiento bajo. Esta estrategia es conocida como **early stopping**.

3.5.5. Dropout

Dropout consiste en tomar la red completa e ir “eliminando” neuronas en las capas ocultas, eliminando también las conexiones que éstas tenían con las demás neuronas, posteriormente se entrena la nueva red y se almacenan los cambios propuestos a los parámetros, este procedimiento es repetido varias veces, donde en cada paso se eliminan distintas neuronas para entrenar una red distinta en cada iteración. Al finalizar, se analizan los resultados obtenidos en cada entrenamiento y se promedian para ser aplicados en la red original. Se puede observar cómo se forma un ensamblaje de neuronas teniendo como red inicial a la figura 3.7 para posteriormente eliminar varias neuronas, lo cual nos deja a la figura 3.8 como la red a entrenar.

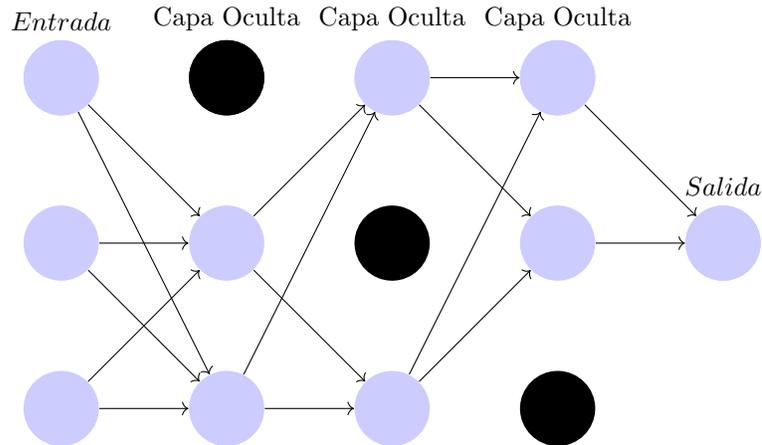


Figura 3.8: Red después de aplicarle *dropout*, los nodos en color negro son los que han sido "borrados" por lo que no se toman en cuenta sus conexiones al momento de entrenar

3.5.6. Normalización en lotes

La normalización en lotes es un método de normalización que en vez de normalizar directamente a la red o a la función de error, normaliza las entradas que recibirá la red. Dos métodos son conocidos para lograr esto: La normalización y estandarización. La normalización escala todos los valores de un conjunto al rango $[0,1]$, para ello se le aplica la ecuación 3.22 a todos los elementos dentro del conjunto

$$x_{nueva} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3.22)$$

La estandarización hace que el conjunto posea media cero y varianza unitaria, para ello se aplica la ecuación 3.23 a cada elemento del conjunto

$$x_{nueva} = \frac{x - \mu}{\sigma} \quad (3.23)$$

Donde μ indica la desviación estándar del conjunto. La normalización en lotes provee una forma de reparametrizar casi cualquier red, esto ayuda a reducir el problema de coordinar las actualizaciones de los resultados entre las capas de la red, la normalización se puede aplicar a cualquier entrada o capa oculta de la red.

Para aplicar normalización en mini lotes, se elige un subconjunto de las activaciones de la capa, \mathbf{H} , también llamado mini lote, este mini lote es normalizado usando la ecuación 3.24

$$\mathbf{H}' = \frac{\mathbf{H} - \mu}{\sigma} \quad (3.24)$$

donde μ es el vector conteniendo la media de cada unidad, y σ el vector con las desviaciones estándar de cada unidad. La red opera con \mathbf{H}' como lo haría con \mathbf{H} , aplicándose las ecuaciones

3.25 y 3.26 a la hora de entrenar

$$\mu = \frac{1}{m} \sum_i H_i \quad (3.25)$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum_i (H_i - \mu)^2} \quad (3.26)$$

donde δ es un valor positivo pequeño como 10^{-8} , puesto para evitar el gradiente indefinido en \sqrt{z} en $z = 0$. El algoritmo de normalización en lotes puede visualizarse en el algoritmo 1.

Algoritmo 1 Cómo se entrena un mini lote, los parámetros γ y β almacenan la escala y giro del valor normalizado, estos valores pueden ayudar a regresar a su valor original a x haciendo $\gamma^k = \sqrt{\text{Var}[x^k]}$ y $\beta^k = E[x^k]$, algoritmo tomado de [26]

Entrada: Valores de x en un mini-lote: $\mathcal{B} = \{x_{1\dots m}\}$;

Parámetros a aprender: γ, β

Salida: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned} \mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{ media del mini lote} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{ varianza de mini-lote} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{ normalizar} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{ escala y cambio} \end{aligned}$$

3.6. Red neuronal con propagación hacia adelante profunda

Las redes neuronales con propagación hacia adelante, también llamados perceptrones multicapa (MLP) son modelos de aprendizaje profundo, son modelos que, idénticamente a los anteriores, tienen la meta de poder aproximarse a alguna función f^* . Recordamos que la primera capa es llamada la capa de entrada y la última capa es llamada la capa de salida y las capas entre éstas dos son llamadas las capas ocultas dado que la información del entrenamiento no muestra la salida esperada en estas capas.

Las redes neuronales se pueden ver como composiciones de distintas funciones. Por ejemplo, si tenemos a las funciones $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ conectadas, éstas se pueden ver como la función compuesta $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ siendo $f^{(1)}$ la primera capa, $f^{(2)}$ la segunda y $f^{(3)}$ la salida.

3.6.1. Arquitectura de un MLP

La arquitectura muestra cuántas unidades debe tener y cómo se conectan entre ellas, donde las redes se acomodan en grupos de unidades llamadas capas, en estas se tiene que la primera capa está dada por:

$$h^{(1)} = g^{(1)} \left(W^{(1)T} x + b^{(1)} \right) \quad (3.27)$$

La segunda capa está dada por:

$$h^{(2)} = g^{(2)} \left(W^{(2)T} h^{(1)} + b^{(2)} \right) \quad (3.28)$$

Y así sucesivamente.

A primera vista, pareciera que para lograr aprender una función no lineal es necesario diseñar un modelo especializado para dicha tarea, en base a esta línea de pensamiento surge el **Teorema universal de aproximación** [27][28], que nos indica que si tenemos una red con propagación hacia adelante con al menos una capa de entrada, una capa de salida y una capa oculta con una función de activación sigmoide es capaz de aproximar a cualquier función, esto siempre y cuando la red tenga las suficientes unidades ocultas. Pero no garantiza que el entrenamiento de la red converja, llegando a tener el problema del gradiente explosivo o desvaneciente, esto también provocando que el entrenamiento sea muy costoso

3.7. Redes Neuronales convolucionales

Un problema que surge al intentar usar las redes neuronales para el reconocimiento de objetos en una o varias imágenes, viene siendo que la estructura de dichas redes no toma en cuenta la estructura de una imagen, tratando de la misma forma a dos pixeles localizados en lados opuestos de la imagen, además, la identidad de un objeto en una imagen a reconocer es invariante sobre las operaciones sobre la imagen, tales como : rotación, aplicación de filtro, etc., cosa que una red tiene problemas para detectar, para resolver estos problemas surgen las redes neuronales convolucionales.

Las redes convolucionales, también llamadas redes neuronales convolucionales, o CNN, son un tipo de red neuronal que posee una topología de rejilla, el nombre “red neuronal convolucional” indica que la red utiliza una operación llamada **convolución**. Las redes convolucionales son simplemente redes neuronales que utilizan convolución en al menos una de sus capas.

3.7.1. Operador convolución

La convolución es una operación entre dos funciones evaluadas en los reales, definida de la siguiente manera:

$$s(t) = \int x(a) w(t-a) \text{ da para el caso continuo} \quad (3.29)$$

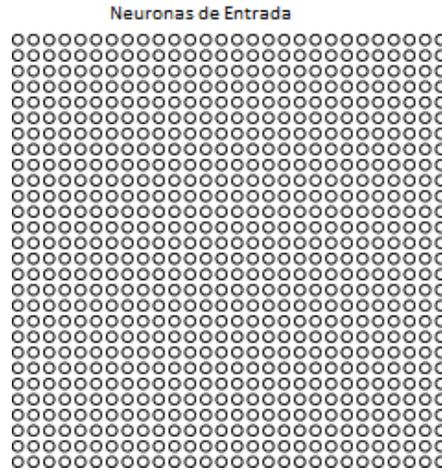


Figura 3.9: Representación de la entrada de una CNN, imagen tomada de [14]

$$s(t) = \sum_{a=-\text{inf}}^{\text{inf}} x(a)w(t-a) \text{ para el caso discreto} \quad (3.30)$$

El operador también se puede denotar con un asterisco, obteniendo:

$$s(t) = (x * w)(t) \quad (3.31)$$

Las CNN usan tres importantes ideas para atacar los problemas previamente descritos: **pesos compartidos**, **agrupación** y **campos locales receptivos**

3.7.2. Campos locales receptivos

Al tener las CNN estructura tipo rejilla, resulta más claro su comportamiento si se cambia la forma en que normalmente se visualizan las redes, siendo que en vez de ser una línea vertical de neuronas, éstas se visualizan como un cuadrado de $n \times n$ neuronas, como se muestra en la figura 3.9.

A diferencia de las redes mostradas anteriormente, las CNN no son redes completamente conectadas, en vez de eso hacen conexiones en pequeñas regiones, como se muestra en la figura 3.10. Esta región es llamado el campo local receptivo de la neurona oculta, donde cada conexión aprende un peso y sesgo a partir del campo receptivo. Posteriormente, se desliza el campo local receptivo a través de la entrada, hasta construir todas las neuronas de la capa oculta, el paso siguiente a la figura 3.10 se puede visualizar en la figura 3.11

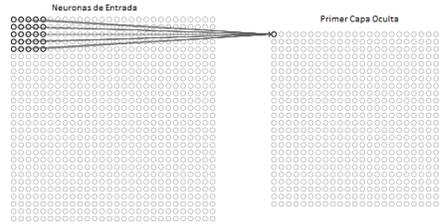


Figura 3.10: Se observa un cuadrado de neuronas en la capa de entrada se conecta a una neurona de la capa oculta, imagen tomada de [14]

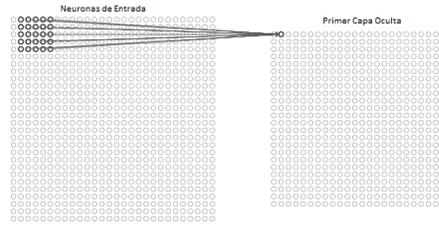


Figura 3.11: El cuadrado se mueve una unidad a la derecha, construyendo la siguiente neurona de la capa oculta. El movimiento puede variar y ser más grande, éste es conocido como *stride*, imagen tomada de [14]

3.7.3. Pesos y sesgos compartidos

Por los campos locales receptivos, tenemos que cada neurona en la capa oculta tiene un sesgo y $r \times r$ pesos que lo conectan hacia su campo local receptivo, no obstante, todas las neuronas ocultas comparten los mismos pesos y sesgos, i.e para la j, k -ésima neurona oculta, su salida es:

$$\sigma \left(b + \sum_{l=0}^z \sum_{m=0}^z w_{l,m} a_{j+l, k+m} \right) \quad (3.32)$$

Esto indica que todas las neuronas en la capa oculta detectan exactamente la misma característica, pero en diferentes lugares de la imagen de entrada, por ello, se refiere al mapeo de la capa de entrada a la capa oculta como el 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, donde cada uno tiene pesos y sesgos compartidos de acorde a la característica a detectar. En la figura 3.12 se observa que se desean detectar tres características en una imagen, por lo que se tienen tres distintos mapeos desde la capa de entrada.

3.7.4. Capas de agrupación

Además de las capas convolucionales, las redes también contienen unas capas llamadas capas de agrupación, estas capas suelen ir después de las convolucionales siendo su función el

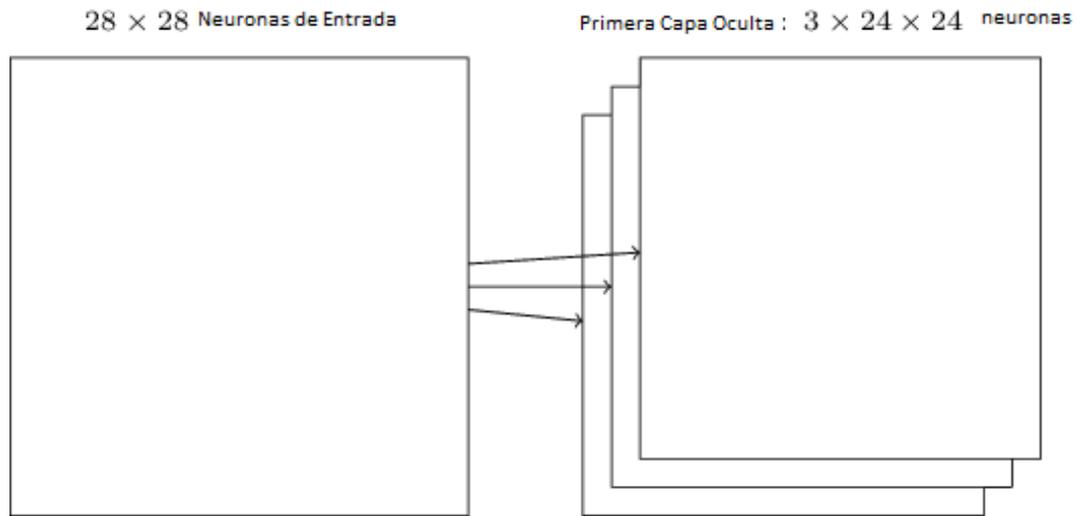


Figura 3.12: La capa de entrada posee varios mapeos hacia la primera capa oculta, donde cada mapeo es capaz de detectar una característica diferente en la imagen de entrada, esto es logrado por medio de los pesos y sesgos compartidos, imagen tomada de [14]

simplificar la información que da de salida la capa convolucional, en otras palabras, una capa de agrupación toma cada mapa de características y lo condensa. La forma más común, y que se usará en lo restante del capítulo, es conocida como *max pooling*.

Max pooling

De igual forma que en la construcción de la capa convolucional, se va recorriendo el mapa de características en un cuadrado de $n \times n$, tomando la salida de cada neurona y agarrando el valor máximo de ese cuadrado, ese valor máximo será el valor de la neurona en la capa de agrupación, este procedimiento es observado en la figura 3.13.

Con lo definido anteriormente, podemos visualizar la conexión entre las neuronas de entrada, las capas convolucionales y las capas de agrupación en la figura 3.14.

3.8. R-CNN

Si bien las redes convolucionales son capaces de identificar objetos dentro de las imágenes, no son tan eficientes, o cuando las imágenes no son tan sencillas empiezan a tener problemas en la detección, para ello surge R-CNN[16], que son Regiones de redes neuronales convolucionales. R-CNN consiste en dividir una imagen en distintas regiones, donde se le asigna una red convolucional a cada región que tiene la tarea de detectar al objeto dentro de dicha región. R-CNN se compone de tres distintos módulos:

- Un módulo que genera las propuestas de región en la imagen.

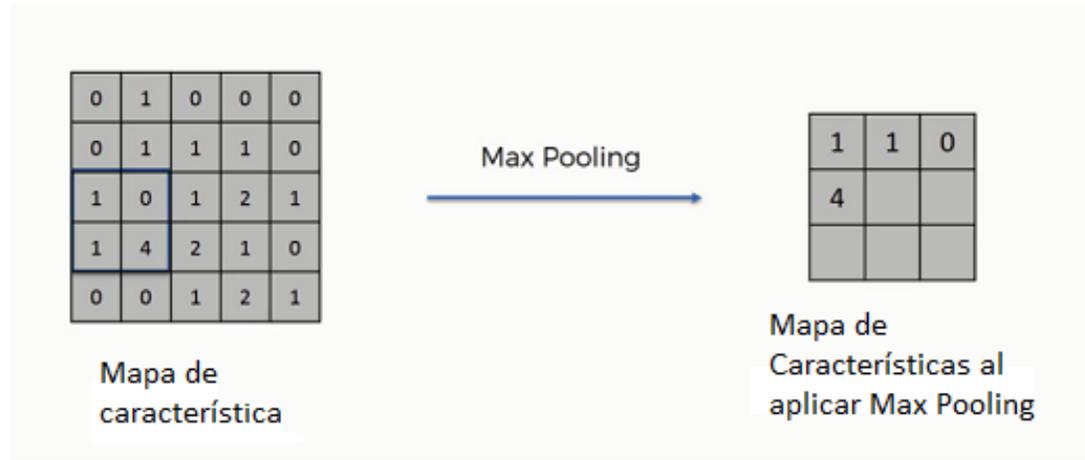


Figura 3.13: En el caso de esta capa de agrupación se toma un *stride* de 2 unidades, resultando en que la capa convolucional de 5×5 se convierta en una capa de agrupación de 3×3 , imagen tomada de [29]

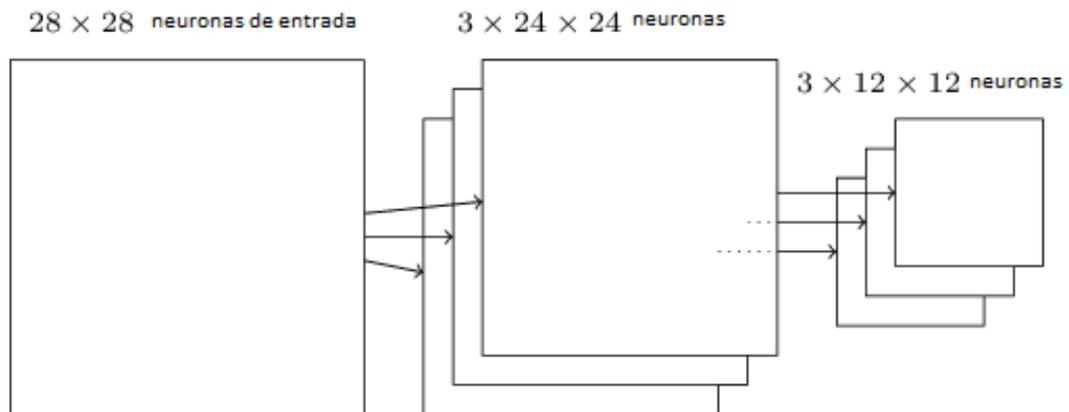


Figura 3.14: Red convolucional con lo dicho previamente, imagen tomada de [14]

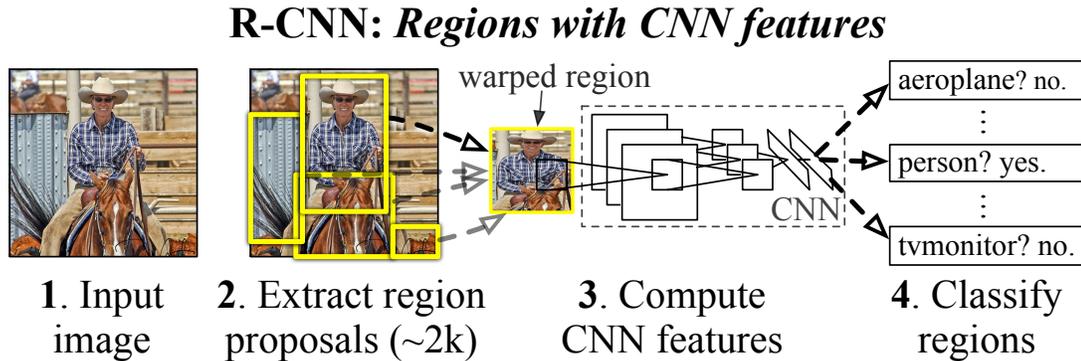


Figura 3.15: Modelo R-CNN, imagen tomada de [16]

- Una red convolucional que cubre al modelo, obteniendo las características de cada región.
- Un conjunto de máquinas de vectores de soporte que clasifican a los objetos a ser detectados.

3.8.1. Funcionamiento de R-CNN

El funcionamiento general se puede observar en la figura 3.15, donde se aprecia el funcionamiento de cada módulo. Primero se obtienen las regiones en las que se dividirá la imagen, esto por medio de la búsqueda selectiva [30][31], ésta, de forma resumida, consiste en los siguientes pasos:

- Se genera una propuesta de varias regiones candidatas inicial, serán la base del algoritmo
- Se empiezan a combinar las regiones similares en regiones más grandes
- Las regiones resultantes se utilizan para producir las regiones candidatas

Una vez teniendo las regiones en la imagen, se empiezan a extraer las características de éstas utilizando una CNN, la CNN extrae un vector de dimensión 4096, esto logrado a través de cinco capas convolucionales y dos capas completamente conectadas.

Una vez que se tienen las características empieza la detección de las clases, para lograrlo se agarra el vector previamente obtenido y se alimenta a las máquinas de vector de soporte, donde cada máquina de vector indica si dicha característica corresponde a la clase a detectar.

R-CNN propone una forma de detectar objetos dentro de una imagen, no obstante, este método posee varias desventajas que hace no sea factible utilizarlo en modelos de tiempo real, entre éstas las principales son:

- Entrenar al modelo R-CNN es un proceso largo, que requiere un proceso de varias etapas dado que se tiene que clasificar 2000 regiones por imagen.

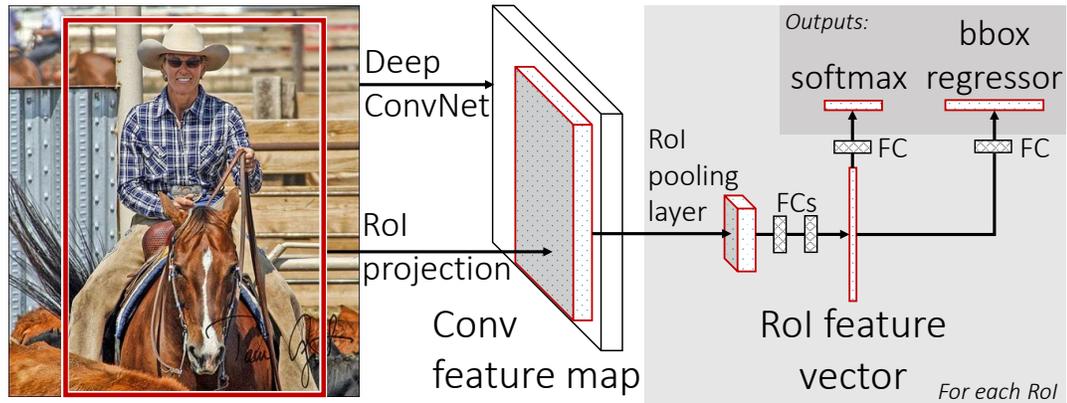


Figura 3.16: Modelo Fast R-CNN, imagen tomada de [17]

- Entrenar al modelo es muy caro en tiempo y espacio.
- La detección de objetos es demasiado lenta.
- La búsqueda selectiva es un algoritmo fijo, por lo que no es posible entrenarlo

3.9. Fast R-CNN

A partir de R-CNN y para poder disminuir las desventajas existentes, se propone el modelo Fast R-CNN[17], que toma como base al modelo R-CNN y lo modifica para mejorar la velocidad de detección.

La característica principal de Fast R-CNN consiste en generar una capa de agrupación de regiones de interés (RoI), ésta funciona al usar *max pooling* para convertir las características alimentadas en una región de tamaño $H \times W$, cada RoI se puede ver como (r, c, h, w) donde los valores especifican la esquina izquierda superior (r, c) , su altura y anchura (h, w) . A diferencia de R-CNN, Fast R-CNN funciona procesando la imagen de entrada en varias capas convolucionales, generando un mapa de características en base a éstas, posteriormente, por cada propuesta de objeto a reconocer, se extrae un vector de dicho mapa. Cada uno de estos vectores se alimenta a una secuencia de capas completamente conectadas, que se separa en dos capas de salidas, una siendo del estimado sobre las k clases de objetos, con una extra representando al fondo, y la otra codifica en 4 números reales la posición de la caja que rodea al objeto.

Se puede apreciar el funcionamiento de Fast R-CNN en la figura 3.16.

3.9.1. Funcionamiento de Fast R-CNN

De igual forma que en R-CNN, se toma a la imagen de entrada, ésta es alimentada a una CNN, la cual genera las regiones de interés, estas regiones pasan por una capa de agrupación RoI, la cual es una capa que ejecuta *max pooling* y produce un mapa de características de tamaño fijo, para ajustar su tamaño acorde a la entrada de la CNN mencionada, esto por cada región de interés.

Cada región es alimentada a una red completamente conectada con una capa softmax, donde la función es $\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$, al final, junto con una capa de regresión lineal, basada en la función de error $L1 = \sum_{i=1}^n \|y_{verdadero} - y_{predicido}\|$, para obtener las clases detectadas y las cajas que lo rodean, respectivamente. Si bien Fast R-CNN es más eficiente que R-CNN, también posee desventajas, en donde la principal es que se sigue utilizando búsqueda selectiva para poder generar las regiones, provocando que no siempre se obtengan las regiones más óptimas, y también tarda 2 segundos por imagen, lo cual hace que cuando se utilizan conjuntos de datos muy grandes no sea eficiente.

3.10. Redes Neuronales Residuales (ResNet)

Se hace un pequeño hincapié antes de seguir con los modelos de R-CNN, esto para introducir un modelo de redes que ayudará en gran manera a la detección de los objetos, el texto está basado en [32].

Conforme se han ido desarrollando más modelos de redes convolucionales, se ha empezado a hacer la pregunta de si es tan sencillo entrenar a las redes como es sencillo agregarle capas; cuando se ha intentado esto empiezan a surgir los problemas del gradiente explosivo y gradiente desvaneciente. Esto se ha empezado a resolver al usar capas normalizadas, que permiten a las redes converger usando propagación hacia atrás con SGD, pero al hacer esto surge otro problema, el problema de la **degradación**, esto provoca que la precisión de la red empiece a deteriorarse conforme se van agregando más capas. Para resolver esto se introduce un marco de trabajo residual, éste funciona al hacer que la capa no espere el resultado de la capa inmediatamente anterior, sino que se introduce una conexión atajo que se salta una o más capas, como se observa en la figura 3.17. Esto se logra haciendo que en vez de que la capa reciba un mapeo normal de $x \rightarrow y$ con una función $H(x)$, se define un mapeo residual utilizando $F(x) = H(x) - x$, que puede ser reacomodado como $H(x) = F(x) + x$, donde $F(x)$ y x representan las capas apiladas y la función identidad, respectivamente.

Para esto, se tiene que si el mapeo es óptimo es más sencillo tener una solución como $F(x) = 0$ en vez de tener una como $F(x) = x$ al usar capas convolucionales apiladas como función, por lo que $F(x)$ es conocida como la función residual.

Existen dos tipos de conexiones residuales: La primera conexión se usará cuando la entrada y salida de la conexión atajo tienen las mismas dimensiones y se define como:

$$y = F(x, \{W_i\}) + x \quad (3.33)$$

La otra conexión se usará cuando las dimensiones difieran entre la entrada y salida de la conexión atajo, por lo que se agregarán ceros para rellenar a la dimensión más pequeña dándonos

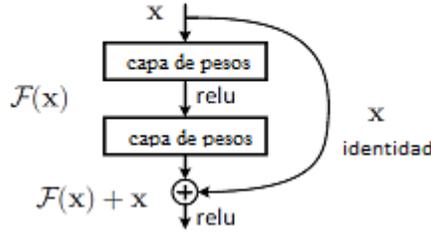


Figura 3.17: Un bloque residual que define la conexión atajo entre dos capas, imagen tomada de [32]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2 _x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3 _x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4 _x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5 _x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10 ⁹	3.6×10 ⁹	3.8×10 ⁹	7.6×10 ⁹	11.3×10 ⁹

Tabla 3.2: Arquitecturas de ImageNet. Tabla tomada de [32]

la siguiente ecuación:

$$y = F(x, \{W_i\}) + W_s x \tag{3.34}$$

Se puede observar en la figura 3.18 cómo se construye una red de 34 capas residual, donde existe una conexión atajo cada dos capas. En la figura 3.2 se observa la construcción de las capas en distintos modelos que utilizan capas residuales, y en la tabla 3.4 se observa la tabla comparativa de estos modelos. En nuestro caso se elige una red de 101 capas (Resnet-101), dado que tiene una considerable mejora en la precisión y no se observa el problema de la degradación.

3.11. Faster R-CNN

Basándose en Fast R-CNN surge un nuevo modelo, Faster R-CNN [18], éste se compone de dos módulos: en el primer módulo se utiliza una red neuronal, en nuestro caso siendo ResNet 101, para generar las regiones y en el segundo módulo se utiliza el detector Fast R-CNN para detectar objetos en las regiones que propone ResNet. Todo el modelo es una red unificada mostrada en la figura 3.19.

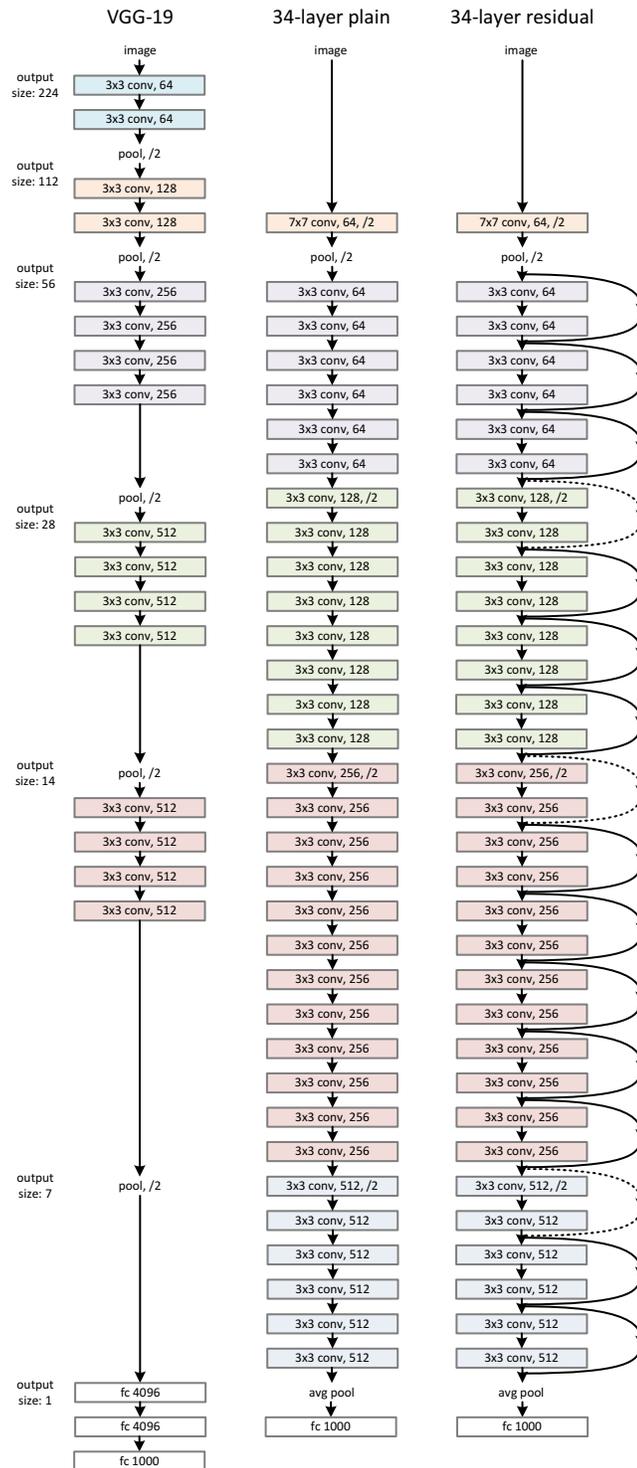


Figura 3.18: Representación de varios modelos de aprendizaje profundo, imagen tomada de [32]

method	top-1 err.	top-5 err.
VGG [33] (ILSVRC'14)	-	8.43 [†]
GoogLeNet [34] (ILSVRC'14)	-	7.89
VGG [33] (v5)	24.4	7.1
PReLU-net [35]	21.59	5.71
BN-inception [26]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	19.38	4.49

Tabla 3.3: Porcentajes de error (%) de **single-model** sobre el conjunto de validación de ImageNet. Tabla tomada de [32]

method	top-5 err. (test)
VGG [33] (ILSVRC'14)	7.32
GoogLeNet [34] (ILSVRC'14)	6.66
VGG [33] (v5)	6.8
PReLU-net [35]	4.94
BN-inception [26]	4.82
ResNet (ILSVRC'15)	3.57

Tabla 3.4: Porcentajes de error (%) de **ensamblajes**. Tabla tomada de [32]

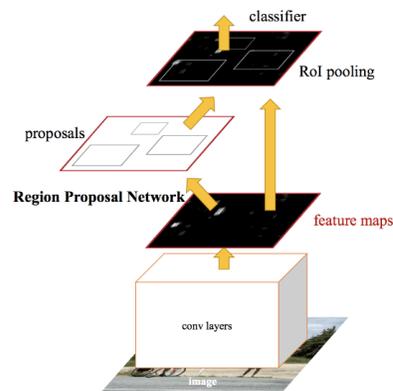


Figura 3.19: Modelo Faster R-CNN, imagen tomada de [18]

3.11.1. Redes de Propuesta de Región (RPN)

Una red de propuesta de región (RPN) es una red que toma de entrada a una imagen, sin importar su tamaño, y regresa un conjunto de propuestas rectangulares de objeto, cada una con un puntaje de pertenecer a una clase o de pertenecer al fondo. Se puede ver la estructura

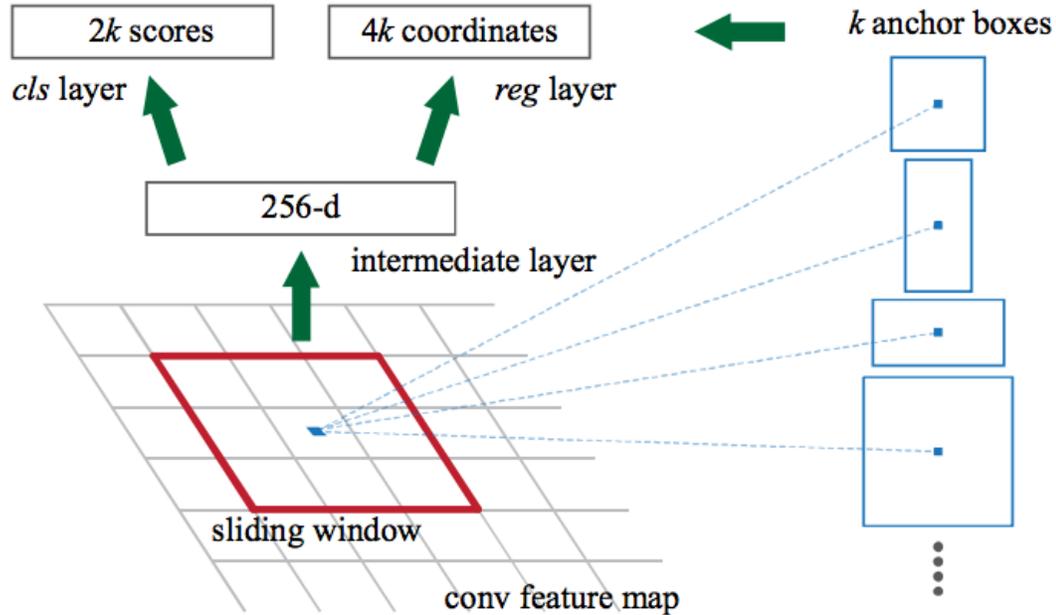


Figura 3.20: RPN, tomado de [18]

general de una RPN en la figura 3.20.

Funcionamiento de una RPN

El primer paso de una RPN consiste en tomar el mapa de características de una imagen, sin importar el tamaño, de ahí empieza a generar propuestas de regiones deslizando una ventana espacial de dimensión $n \times n$ dentro de el mapa recibido. En cada paso que se desliza la ventana se mapea a un mapa de características menor, en este caso de dimensión 256. Este nuevo mapa se alimenta a a dos capas hermanas completamente conectadas, una de estas capas es una capa de regresión que se encarga de generar las cajas *reg* y la otra capa es una capa de clasificación *cls*.

Para generar estos nuevos mapas, por cada ventana deslizante que se tiene, se empiezan a predecir múltiples propuestas de región, donde se define como el límite a k . En base a esto, la capa *reg* da de salida $4k$ propuestas, siendo correspondientes a las coordenadas de la caja, mientras que la capa *cls* da de salida $2k$ propuestas, siendo correspondientes a la probabilidad de pertenecer a alguna clase y a la probabilidad de pertenecer a la clase de fondo. Entonces, las k propuestas que tenemos serán llamadas *anclas*, siendo éstas relativas a las k cajas que rodearán a los objetos. En particular, las anclas se centran a la mitad de la ventana deslizante, teniendo asociados una escala y una relación de aspecto. Para entrenar las RPN, se le asigna una etiqueta de clase (ser o no ser objeto) a cada ancla, en base a esto se minimiza una función de error siguiendo el entrenamiento de Fast R-CNN, entrenar una RPN puede ser por medio de usar propagación hacia atrás con SGD.

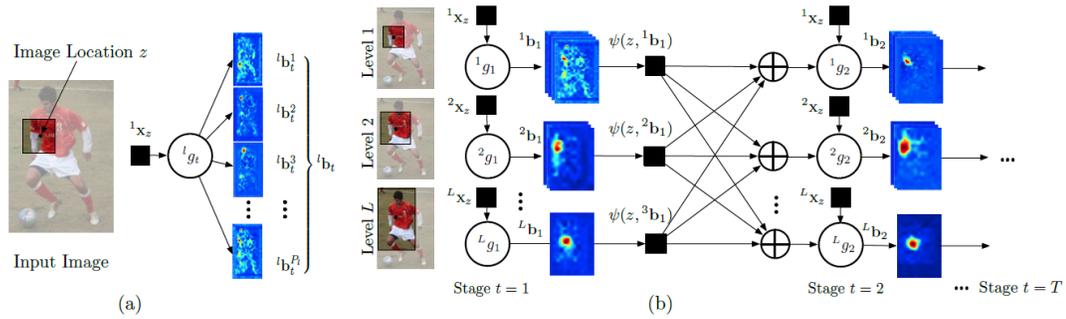


Figura 3.21: (a) Muestra la predicción de clase de un parche cualquiera en la ubicación z de la imagen. (b) Muestra las etapas de las máquinas de inferencia, mostrando el pase de mensajes entre clasificadores para ir refinando la detección conforme pasan las etapas. Tomada de [19].

3.11.2. Funcionamiento de Faster R-CNN

De igual forma que en los modelos RCNN anteriores, se toma una imagen cualquiera como entrada, ésta es alimentada a la red convolucional que se encargará de regresar su respectivo mapa de características. Este mapa de características es alimentado a las redes de propuesta de región, donde regresa los mapas de característica de cada ventana, teniendo las propuestas del objeto junto con su puntaje de ser objeto. Estas propuestas junto con sus puntajes se envían a una capa de agrupación RoI para ajustar todas las propuestas al mismo tamaño.

Finalmente, las propuestas son alimentadas a una capa completamente conectada, la cual tiene una capa softmax y una capa de regresión lineal; las tareas de estas capas con clasificar el objeto y regresar las cajas que los rodean.

3.12. Detección de Pose de una o varias personas

Últimamente se han diseñado múltiples sistemas de detección de pose de una o varias personas. En esta sección se iniciarán con las Máquinas de Pose Inferenciales, las cuales son capaces de detectar la pose de una persona en una imagen, terminando la sección con el sistema OpenPose, el cual es capaz de detectar la pose de una o más personas en imágenes a color.

3.12.1. Máquinas de Pose Inferenciales (PM)

Las máquinas de pose inferenciales surgen como iniciativa para identificar la pose de una persona dentro de una imagen, esto al modelar la pose como un problema estructural y proponiendo la ubicación de los pixeles de cada "parte" del cuerpo de una persona. Formalmente, se busca poder modelar la ubicación de las partes del cuerpo en la imagen $Y_p \in \mathbb{Z} \subset \mathbb{R}^2$, siendo \mathbb{Z} el conjunto de las posibles (u, v) ubicaciones en la imagen. Para poder clasificar y detectar cada parte del cuerpo dentro de la imagen, se utilizarán máqui-

nas de inferencia, las cuales consisten en secuencias de clasificadores, $g_t(\cdot)$, que predicen la ubicación de cada parte, esto por cada etapa de la detección. Este proceso puede verse en la figura 3.21.

Durante este proceso, en cada etapa, $t \in \{1 \dots T\}$, el clasificador predice, para cada parte, su respectiva confianza de clase en la ubicación $Y_p = z, \forall z \in \mathbb{Z}$, tomada de las características de la imagen, $x_z \in \mathbb{R}^d$, y de la información contextual del clasificador anterior, esto dentro del vecindario que rodea a Y_p .

Conforme van progresando las etapas, las confianzas calculadas van siendo cada vez más refinadas para la parte Y_p , teniendo que la confianza $Y_p = z$ se calcula, para la etapa t con la siguiente ecuación:

$$b_t(Y_p = z) = g_t^p(x_z; \oplus_{i=1}^P \psi(z, b_{t-1}^i)) \quad (3.35)$$

donde

$$b_{t-1}^p = \{b_{t-1}(Y_p = z)\}_{z \in \mathbb{Z}} \quad (3.36)$$

es el conjunto de confianzas del clasificador anterior evaluado en cada z para la p -ésima parte. Teniendo la función de características $\psi : z \times \mathbb{R}^{|\mathbb{Z}|} \rightarrow \mathbb{R}^{d_c}$, la cual calcula las características contextuales de los clasificadores anteriores, siendo \oplus el operador para la concatenación de vectores.

Jerarquía de detección

Para una detección más sencilla de las partes, se utiliza una jerarquía de partes, siendo el nivel más bajo el nivel que contiene a las partes anatómicas y el nivel más alto el que contiene al cuerpo completo, teniendo que toda parte en el nivel l se encontrará componiendo a una parte más compleja en el nivel $l + 1$. Se denotarán por P_1, \dots, P_l a la cantidad de partes dentro de cada nivel $l \in L$ dentro de la jerarquía, y por ${}^l g_t(\cdot)$ al clasificador de la t -ésima etapa en el l -ésimo nivel, produciendo un puntaje para la p -ésima parte. Para estas máquinas de inferencia se utilizará un único clasificador multiclase que produce las confianzas para todas las partes, dado un nivel particular de la jerarquía. Este clasificador se denotará por ${}^l g_t(\cdot)$.

Para obtener un primer estimado de las confianzas para la ubicación z de cada parte, en la primer etapa ($t = 1$) de la secuencia, el predictor ${}^l g_1(\cdot)$ toma de entrada a las características calculadas dentro de la ubicación z , clasificando a la pieza en una de las posibles P_l partes o en una clase de fondo, esto para las partes dentro del l -ésimo nivel de la jerarquía, en el centro de z .

En base a esto, el clasificador del l -ésimo nivel, en la primer etapa, produce los siguientes valores de confianza:

$${}^l g_1(x_z^1) \rightarrow \{{}^l b_1^p(Y_p = z)\}_{p \in 0..P_l} \quad (3.37)$$

Siendo ${}^l b_1^p(Y_p = z)$ el puntaje predicho por ${}^l g_1(\cdot)$ para la asignación de la p -ésima parte en el l -ésimo nivel dentro de la primer etapa en z .

Análogamente, se representan todas las confianzas de P en l dentro de $z = (u, v)^T$ como ${}^l b_t^p$, siendo w y h el ancho y alto de la imagen, respectivamente. Usando esto obtenemos la siguiente ecuación:

$${}^l b_t^p[u, v] = {}^l b_t^p(Y_p - (u, v)^T) \quad (3.38)$$

Se denota por ${}^l b_t \in \mathbb{R}^{w \times h \times P_t}$ a los mapas de confianza para todas las partes dentro de l . En etapas subsecuentes, la confianza de cada parte se calcula de manera similar usando la ecuación 3.36, para lograr apalancar al contexto de la jerarquía hacia los distintos niveles de la jerarquía, se utiliza la siguiente ecuación:

$${}^l g_t \left(x_z^l \oplus_{l \in 1..L} \psi \left(z, {}^l b_{t-1} \right) \right) \rightarrow \{ {}^l b_t^p (Y_p = z) \}_{p \in 0..P_t} \quad (3.39)$$

Viendo a la figura 3.21 se tiene que en la segunda etapa, ${}^l g_2$ toma como entrada a las características x_z^l y a las características de confianza calculadas por ψ , esto por cada parte de la etapa anterior.

Características de contexto

Para capturar las correlaciones espaciales entre las confianzas de cada parte respecto a sus vecinos se definen dos tipos de **contexto**, éste asociado a los mapas de confianza, los cuales serán denotados por ψ_1 y ψ_2 .

Características de contextos de partes : El mapa ψ_1 , en z , toma de entrada a los mapas de cada ubicación de cada parte en l y produce un vector de características extraído en z , esto en el mapa ${}^l b_t^p$. Estos vectores se denotarán por $c_1 \left(z, {}^l b_t^p \right)$, por lo que para obtener al mapa ψ_1 se tiene la siguiente ecuación:

$$\psi_1 \left(z, {}^l b_t^p \right) = \oplus_{p \in 0..P_t} c_1 \left(z, {}^l b_t^p \right) \quad (3.40)$$

Es decir, ψ_1 es la concatenación de puntajes en z , siendo extraídos de los mapas de todas las partes por cada nivel de la jerarquía. ψ_1 codifica la información del vecindario que rodea a z .

Características de contexto compensadas : El segundo mapa, ψ_2 , codifica interacciones de largo alcance entre distintas partes. Para lograr esto, primero se obtienen los K picos por cada mapa de confianza de cada parte, ${}^l b_t^p$, en el nivel l . Posteriormente se calcula el vector de compensación en coordenadas polares, desde z a cada pico, esto en el mapa de la p -ésima parte y el l -ésimo nivel, denotado por ${}^l o_k^p \in R^+ \times \mathbb{R}$. El conjunto de características compensadas calculadas a partir de un mapa de una parte se define por :

$$c_2 \left(z, {}^l b_{t-1}^p \right) = [{}^l o_1^p; \dots; {}^l o_k^p] \quad (3.41)$$

De igual forma que con ψ_1 , ψ_2 se construye al concatenar el contexto de características compensadas $c_2 \left(z, {}^l b_{t-1}^p \right)$ por cada parte dentro de la jerarquía, obteniendo:

$$\psi_2 \left(z, {}^l b_{t-1}^p \right) = \oplus_{p \in 0..P_t} c_2 \left(z, {}^l b_{t-1}^p \right) \quad (3.42)$$

De forma resumida, se tiene que el parche de contexto de características ψ_1 captura la información gruesa respecto a la confianza de los puntos vecinos, mientras que el parche de contexto de características compensadas ψ_2 captura información precisa relativa a la información de

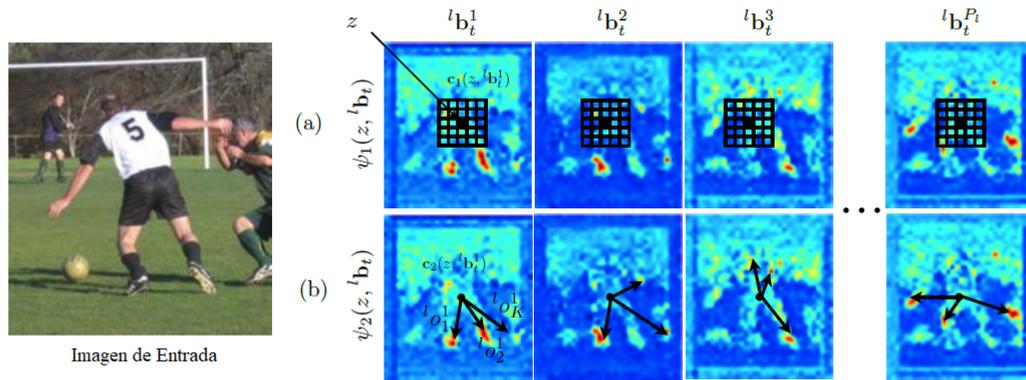


Figura 3.22: **Mapas de características de contexto** (a) Muestra el proceso de ψ_1 mientras que (b) muestra el proceso de ψ_2 . Tomada de [19].

ubicación. Finalmente, ψ se construye al concatenar a ψ_1 y a ψ_2 .

$$\psi(\cdot) = [\psi_1(\cdot); \psi_2(\cdot)] \quad (3.43)$$

La forma en que ψ_1 y ψ_2 funcionan puede verse en la figura 3.22.

3.12.2. Máquinas de Pose Convolucionales (CPM)

LAS CPMs, basadas en las PM, de igual forma se utilizan para identificar la pose de una persona en una imagen, esto al heredar los beneficios de las PM y combinarlo con las ventajas de las arquitecturas convolucionales, agregando los **campos receptivos** al proceso, los cuales se encargan de capturar las interacciones de largo alcance entre partes.

El proceso de una CPM se puede ver en la figura 3.23. Las CPM serán formadas por secuencias de redes convolucionales, las cuales producen, de manera repetitiva, mapas de confianza 2D por cada ubicación z de cada parte p .

Funcionamiento de una CPM

La primer etapa de las CPM se centra en la ubicación de los puntos clave por medio de evidencia local de imagen, para esto se utiliza a una red compuesta de S capas convolucionales, seguidas de capas 1×1 convolucionales, logrando una arquitectura completamente convolucional. Si bien la salida de la red detecta sin problemas partes muy “identificables”, como la cabeza u hombros, a la red se le complica la identificación de partes menos claras. No obstante, los mapas obtenidos en el vecindario de las partes pueden ser usados para la detección de las partes faltantes, esto al hacer que el predictor usado $g(\cdot)$ use el contexto espacial ψ_t en etapas subsecuentes ($t > 1$) en los mapas que rodean a z y mejorar sus predicciones al tomar en cuenta las configuraciones geométricas de las partes del cuerpo.

En otras palabras, a partir de la segunda etapa, el clasificador g_t acepta las características x_z^t

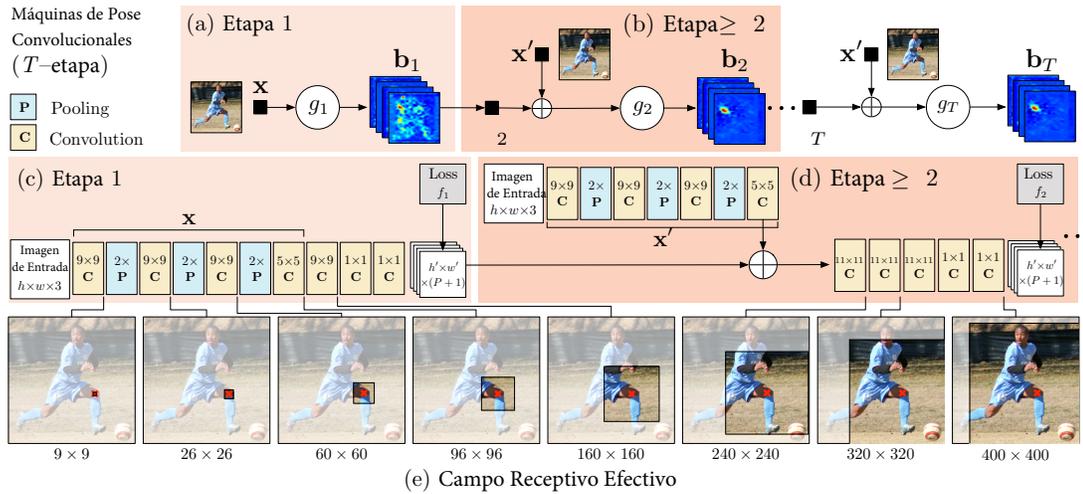


Figura 3.23: **Arquitectura y campos receptivos de las Máquinas de Pose Convolucionales.** Se muestra la arquitectura convolucional y los campos receptivos a través de una CPM con T etapas. Tomada de [20].

y las características de las creencias formadas a través de ψ , esto por cada parte de la etapa anterior. Se definirá a ψ como al campo receptivo del predictor en las creencias de la etapa anterior, esto dado que para las CPM no hay una función explícita que calcule a ψ . El funcionamiento de los campos receptivos se puede ver en la figura 3.24.

Ubicación de partes claves usando la evidencia local de imagen

La primer etapa de las CPM predice las creencias de partes únicamente usando la evidencia local de imagen. La evidencia es local dado que el campo receptivo de la primer etapa de la red es restringido a un pequeño parche alrededor de la salida de la ubicación del pixel.

Predicción secuencial con características de contexto espaciales aprendidas

Mientras que la velocidad de detección en los puntos de referencia del cuerpo con apariencia consistente, como la cabeza u hombros, puede ser favorable, las precisiones generalmente son menores para los puntos de referencia bajo la cadena cinética del esqueleto humano dada la gran variación entre la configuración y la apariencia. El entorno de los mapas de confianza alrededor de una ubicación de cada parte, si bien con mucho ruido, pueden llegar a ser muy informativos. En la segunda etapa de una máquina de pose, el clasificador g_2 acepta como entrada las características de imagen x_z^2 y como características calculadas en las creencias a través de la función de características ψ para cada parte en la etapa anterior.

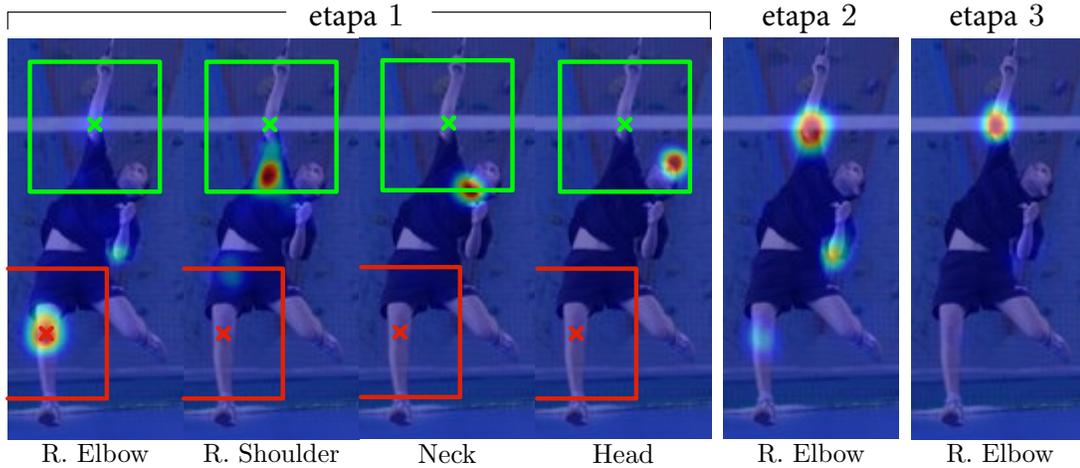


Figura 3.24: Contexto espacial a partir de los mapas de confianza. Tomada de [20].



Figura 3.25: Proceso del nuevo método. Tomada de [21].

3.12.3. Estimación de pose 2D en tiempo real multipersonas usando Campos de Afinidad de Partes (PAF)

Este nuevo sistema es parecido a las CPM, primero tomando una imagen a color de tamaño $w \times h$ y produciendo una salida en las ubicaciones 2D de los puntos clave por cada persona en la imagen. Este proceso se puede ver en la figura 3.25. El sistema inicia en cuando se recibe una imagen de entrada, la cual es enviada a una red con propagación hacia adelante, encargándose de producir, de manera simultánea, un conjunto, S , que contiene a los mapas de confianza de partes y otro conjunto L , que contiene campos vectoriales, donde se codifica el grado de asociación entre partes.

De manera más formal, sea $S = (S_1, \dots, S_J)$ el conjunto con los J mapas de confianza, un mapa por parte, teniendo que $S_j \in \mathbb{R}^{w \times h}$, $j \in \{1 \dots J\}$. Sea $L = (L_1, \dots, L_C)$ el conjunto con C campos vectoriales, un campo por extremidad, donde $L_c \in \mathbb{R}^{w \times h \times z}$, $c \in \{1 \dots C\}$ que se encarga de representar la ubicación L_c de la imagen en un vector 2D.

Una vez se tiene esto, los mapas de confianza y los campos de afinidad son procesados usando inferencia voraz para dar de salida a los 2D puntos clave de todos los puntos clave de todas las

personas en la imagen.

El funcionamiento se va desglosando a lo largo de la sección.

Detección y Asociación Simultánea

Para que la red logre predecir de manera simultánea los mapas de confianza y los campos vectoriales, ésta se divide en dos ramas. LA rama superior, mostrada en color beige, predice los mapas de confianza, la rama inferior, mostrada en color azul, predice los campos de afinidad. Para ello, cada rama es una arquitectura iterativa de predicción, esto basado en las CPM. Primero, la red analiza la imagen por medio de una red convolucional, generando al conjunto de mapas de características, los cuales serán las entradas en la primer rama de cada etapa. En esta primer etapa, la red se encarga de producir un conjunto de predicción de mapas de confianzas, $S^1 = \rho^1(F)$, y aun conjunto de PAFs, $L^1 = \phi^1(F)$, donde ρ^1 y ϕ^1 son las CNN de inferencia en la primer etapa, agregándose las características originales de la imagen F, esto para poder producir predicciones más refinadas:

$$S^t = \rho^t(F, S^{t-1}, L^{t-1}), \forall t \geq 2 \quad (3.44)$$

$$L^t = \phi^t(F, S^{t-1}, L^{t-1}), \forall t \geq 2 \quad (3.45)$$

Siendo ρ^t y ϕ^t las CNN de inferencia en la etapa t .

Las funciones de pérdida para las ramas son:

$$f_s^t = \sum_{j=1}^J \sum_p W(p) \cdot \|S_j^t(p) - S_j^*(p)\|_2^2 \quad (3.46)$$

$$f_c^t = \sum_{j=1}^J \sum_p W(p) \cdot \|L_c^t(p) - L_c^*(p)\|_2^2 \quad (3.47)$$

Siendo S_j^* el “verdadero” mapa de confianza de las partes y L_c^* los “verdaderos” campos vectoriales, W una máscara binaria con $W(p) = 0$. Teniendo que el objetivo general es

$$f = \sum_{t=1}^T (f_S^t + f_L^t) \quad (3.48)$$

Mapas de confianza para la detección de partes

Para evaluar la ecuación 3.48, se generan los mapas S^{ast} de las partes clave anotadas 2D, siendo que cada mapa es una representación de la creencia de que una parte en particular ocurre en cada ubicación del pixel. Esto nos dice que, idealmente, si únicamente hay una persona, únicamente debería existir en cada mapa un pico, esto en caso de encontrarse la parte correspondiente y, en caso de haber Q personas, debería haber un pico correspondiente por cada parte visible, j , por cada persona k .

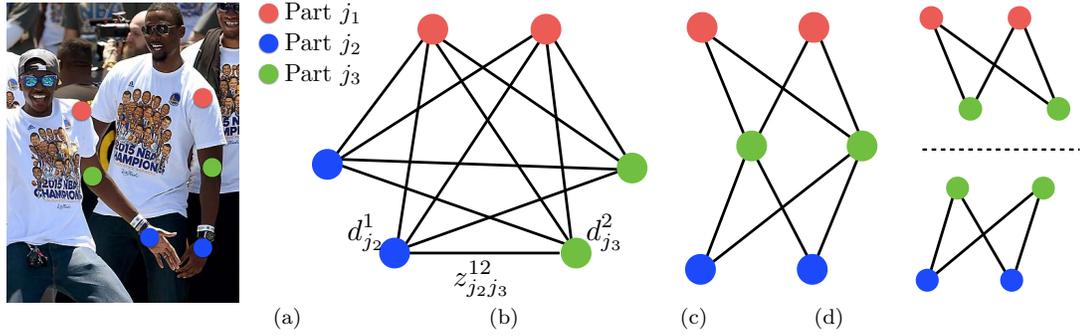


Figura 3.26: Correspondencia entre gráficas. (a) Imagen Original con detección de partes. (b) Gráfica K -partita. (c) Estructura del árbol. (d) Un conjunto de gráficas bipartitas. Tomada de [21].

Para lograr esto, primero se generan los mapas individuales, $S_{j,k}^*$, por cada k persona en la imagen. El valor de la ubicación $p \in \mathbb{R}^2$ en $S_{j,k}^*$ se define en la siguiente ecuación.

$$S_{j,k}^*(p) = \exp\left(-\frac{\|p - x_{j,k}\|_2^2}{\sigma^2}\right) \quad (3.49)$$

Donde σ controla la distribución del pico.

Entonces se tiene que el “verdadero” mapa a ser predicho es una agrupación de los mapas individuales de confianza por medio de un operador Max.

$$S_j^*(p) = \max_k S_{j,k}^*(p) \quad (3.50)$$

Se toma el máximo de los mapas de confianza para que la posición de diferencia entre picos permanezca distintiva.

PAFs por asociación de partes

Teniendo el conjunto de partes detectadas, se necesita una medida de confianza que cuantifique el grado de asociación de pares entre las partes del cuerpo. Aquí es donde entran las PAF, éstos representan la ubicación y relación de cada región respecto a las extremidades. Estos datos se representan en un campo vectorial 2D, uno por extremidad, siendo que, por cada pixel en el área de una extremidad, un vector 2D codifica la dirección a la que apunta una parte de una extremidad a la otra. Para esto cada tipo de extremidad tiene un PAF correspondiente, el cual une a dos partes del cuerpo asociadas, como se ve en la siguiente ecuación.

$$L_{c,k}^*(p) = \begin{cases} v & \text{si } p \text{ pertenece a la parte } c, k \\ 0 & \text{en otro caso} \end{cases} \quad (3.51)$$

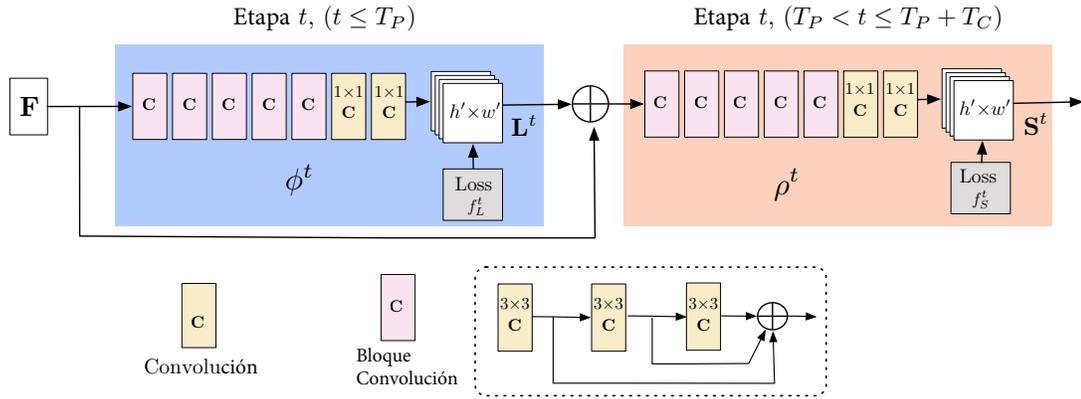


Figura 3.27: Arquitectura. Tomada de [22].

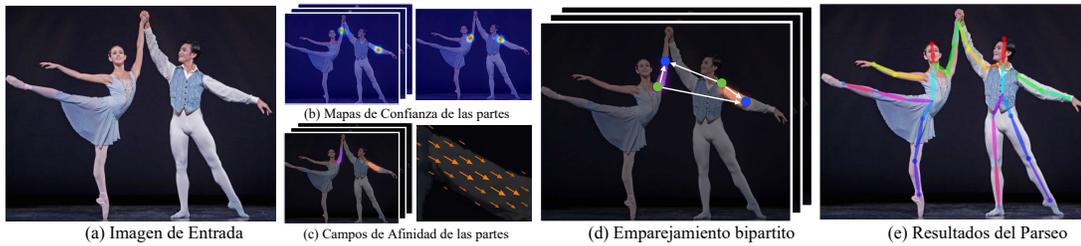


Figura 3.28: Pipeline de OpenPose. Tomada de [22].

Las “verdaderas” PAFs promedio de los Campos de Afinidad de todas las personas en la imagen se puede obtener por la siguiente ecuación:

$$L_c^*(p) = \frac{1}{n_c(p)} \sum_k L_{c,k}^*(p) \quad (3.52)$$

Siendo $n_c(p)$ el número de vectores no cero que apuntan a p entre las k personas. Es decir, el promedio de píxeles donde las extremidades de distintas personas chocan.

3.12.4. Openpose

OpenPose es el nombre del sistema capaz de detectar de manera conjunto los puntos clave del cuerpo de una o varias personas en una imagen.

Este sistema está basado en la arquitectura del sistema anterior, únicamente cambiando la arquitectura de la red, siendo que se utilice la misma arquitectura sin importar la etapa en que se encuentre el proceso. Esta nueva arquitectura se puede observar en las figuras 3.27 y 3.28. Salvo por este pequeño cambio, el proceso es el mismo que el anterior.

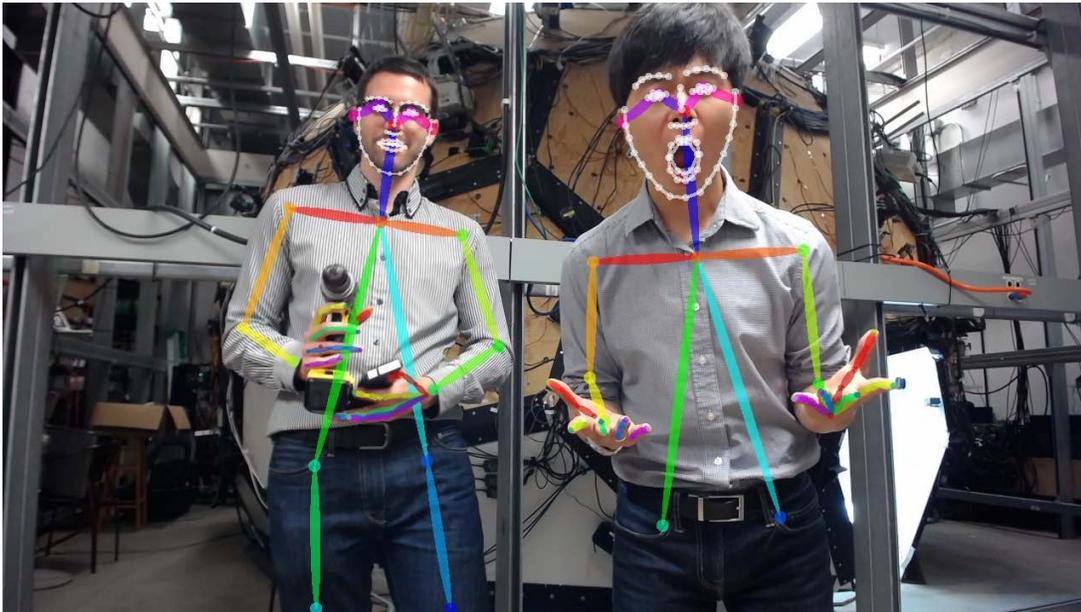


Figura 3.30: Muestra de OpenPose sobre un par de personas, mostrando la segmentación en manos y cara. Tomada de [22].

3.12.5. Funcionamiento de OpenPose

El funcionamiento de OpenPose es idéntico al funcionamiento del framework introducido anteriormente: Toma una imagen, genera sus mapas de confianza y sus APFs, agrupamiento bipartito y arma el cuerpo completo al final.

Siendo la diferencia la definición de las etapas, dado que previamente se hacía distinción entre la primer etapa y sus subsecuentes, mientras que para esta versión las etapas se definen de la misma forma, siendo en este caso la misma red la que obtiene las características iniciales de la imagen a la red que va obteniendo los mapas de confianza. Esto se ve en la figura 3.27

De igual forma se pueden apreciar los resultados al usar openpose en las figuras 3.29 y 3.30

3.13. Redes Neuronales Recurrentes (RNN)

En los capítulos anteriores, las redes introducidas únicamente sirven para los problemas donde la solución no depende de los estados anteriores de la red, es decir, la información únicamente fluye hacia adelante. Esto significa que las redes anteriores no logran aprender dependencias de largo alcance al no tener toda la información necesaria para realizar una predicción. Para ello surgen las Redes Neuronales Recurrentes, RNN, las cuales sirven para detectar patrones en secuencias, tales como la identificación de patrones en la interpretación de una palabra en el lenguaje de señas, que es nuestra meta.

La forma en que las RNN funcionan es al agregar ciclos dentro de la estructura de la red, se puede apreciar un ciclo en la figura 3.33, los cuales permiten a la red “recordar” información

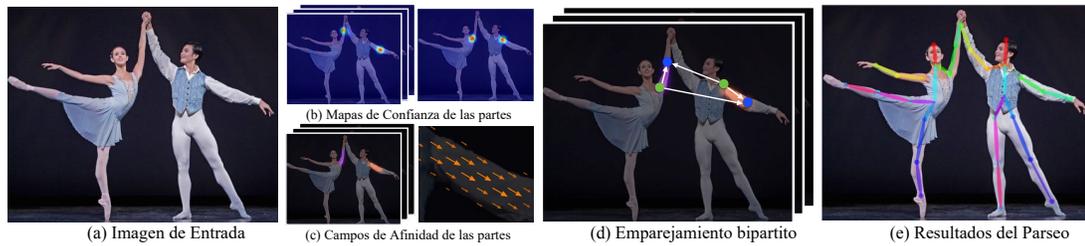


Figura 3.31: Proceso de Openpose. (a) Se toma una imagen para alimentarse a la CNN. (b) Se obtienen los mapas de confianza de partes. (c) Se obtienen los campos de afinidad. (d) El parseo ejecuta una coincidencia bipartita para asociar las partes candidatas. (e) Se arman las partes para el pose completo de cuerpo de las personas en la imagen. Tomada de [22].

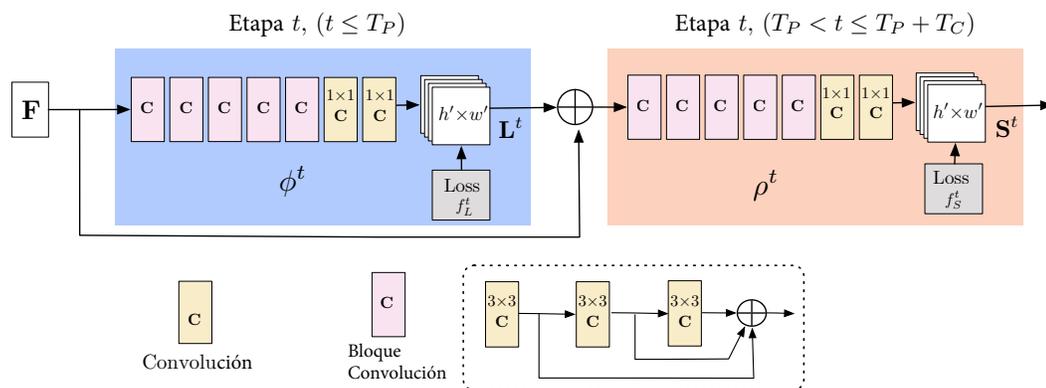


Figura 3.32: Arquitectura de una CNN multi etapa. Las primeras etapas predicen PAFs, mientras que las últimas predicen los mapas de confianza. Tomada de [22].

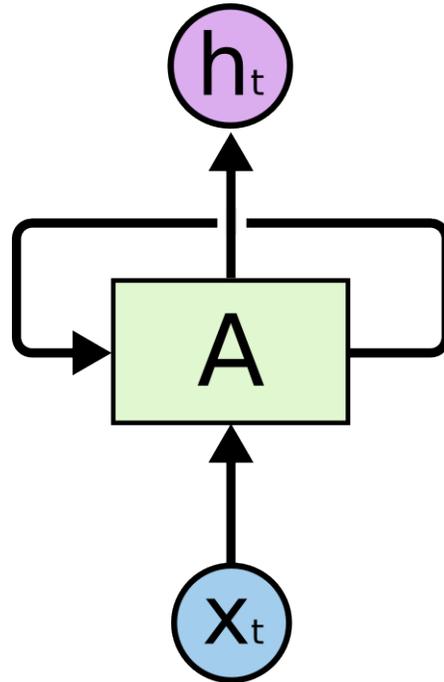


Figura 3.33: Un ciclo en una red recurrente doblada. Tomada de [23].

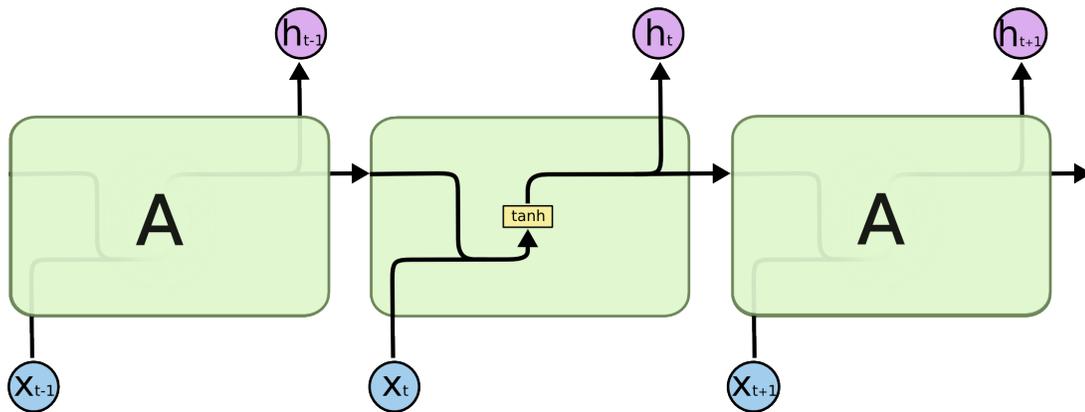


Figura 3.34: Ejemplo de un módulo de una red RNN, que se compone de una capa tanh. Tomada de [23].

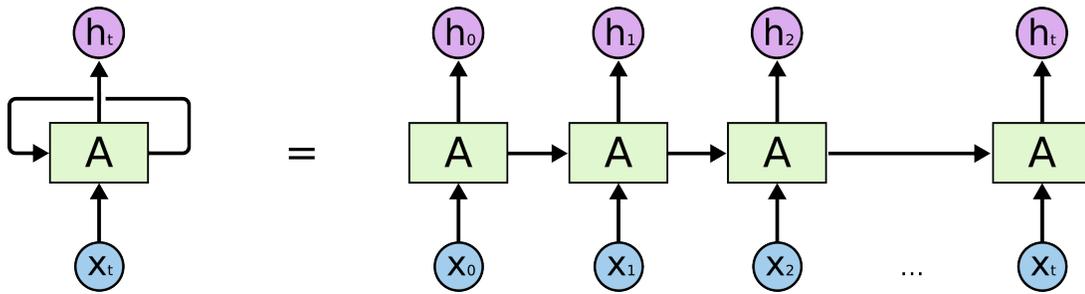


Figura 3.35: Un ciclo de una red recurrente desdoblada. Tomada de [23].

vista previamente. este ciclo se puede pensar como una pequeña red dentro de la red principal, siendo generada al “desdoblar” el ciclo en k capas, como se puede ver en la figura 3.35. Al desdoblar la red, se generan secuencias de capas, las cuales se especializan en la clasificación de las secuencias.

Si bien las RNN son capaces de aprender secuencias, su estructura no es tan compleja, vista en la figura 3.34, como para permitir asociar patrones de largo alcance en secuencias muy largas, dado que conforme más largas son las dependencias, mayor es la probabilidad de tener un problema de el gradiente, ya sea explosivo o desvaneciente, lo cual provoca que la red empiece a olvidar información previamente aprendida. Para lograr aprender secuencias sin estos problemas surgen las **Redes de Memoria de Corto-Largo alcance**, LSTM.

De igual forma que en las RNN, las LSTM poseen ciclos dentro de su estructura, pero en este caso el nodo con ciclos posee una estructura más compleja, visto en la figura 3.36, la cual permite recordar o desechar información que se considera importante o inútil, permitiendo la red aprenda secuencias más largas y complejas, logrando controlar los problemas del gradiente. Las LSTM se componen de celdas de estado, mostradas en la figura 3.40, junto con varias puertas que regulan la información, mostradas en las figuras 3.38, 3.39 y 3.41 junto con su respectiva ecuación, donde entre más cercano el resultado a 1 mayor información se conserva, teniendo que en caso de ser 1 se inserta toda la información y en caso de ser 0 se desecha toda la información. En particular, todas las puertas se conectan al estado de la celda, el cual funciona como una vía que atraviesa a la celda, transportando la información desde la entrada al final de la capa, donde, dependiendo del resultado de las puertas, se decide qué hacer con la información.

3.13.1. Redes LSTM

Funcionamiento de LSTM

Primero, la capa LSTM recibe información de las capas anteriores, donde la puerta de olvido decide qué realizar con la entrada L_{t-1} , junto con su observación x_t^* , para posteriormente agregarlo al estado de la celda C_{t-1} , esto se puede ver en la figura 3.38.

El siguiente paso es decidir la información que se almacenará en el estado de la celda, en este

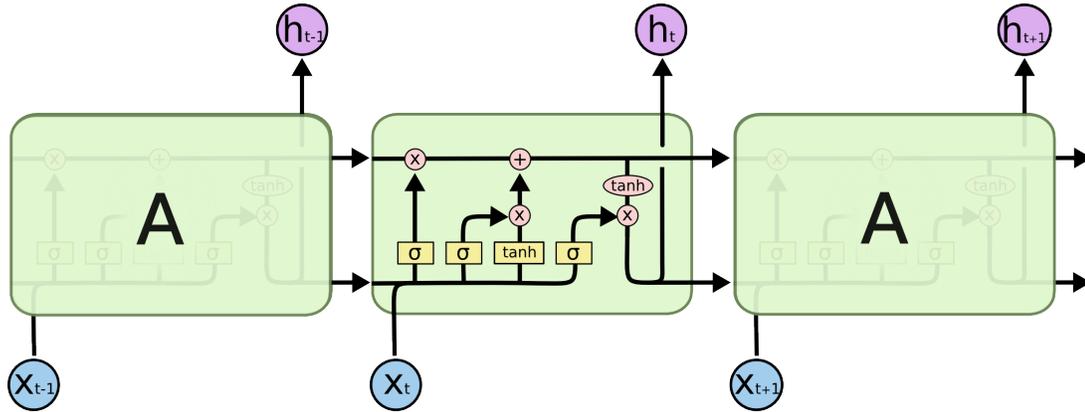
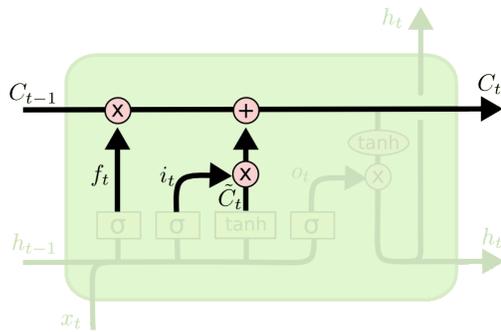
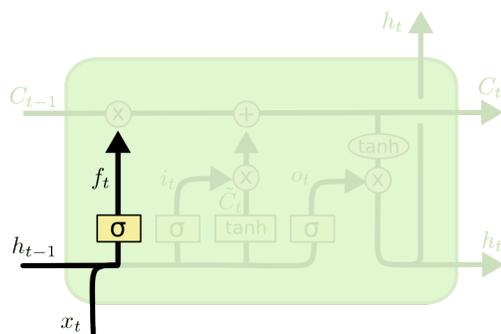


Figura 3.36: Ejemplo de un módulo de una red LSTM, que se compone de cuatro distintas capas, representadas por σ , σ , \tanh y σ . Tomada de [23].



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 3.37: **Estado de la celda:** Vía por donde pasa la memoria de la celda para actualizarse de acorde a la información recibida. Tomada de [23].



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Figura 3.38: **Puerta de Olvido:** Puerta encargada de dar paso únicamente a la información útil de la información recibida. Tomada de [23].

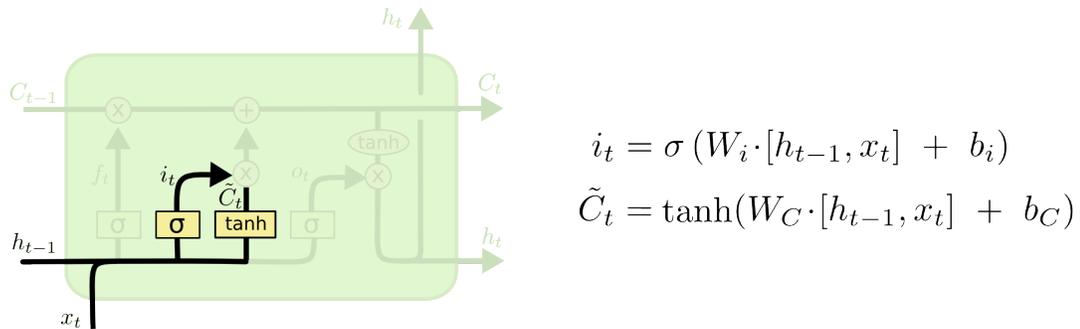


Figura 3.39: **Puerta de Entrada:** Puerta encargada de agregar información al estado de la celda. Tomada de [23].

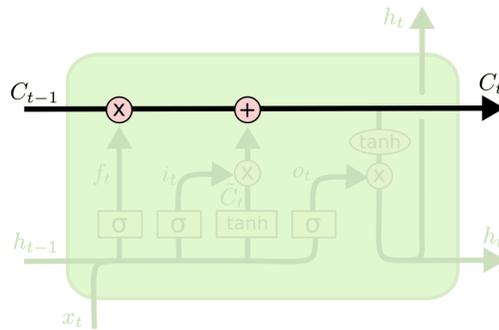


Figura 3.40: Celda de estado. Tomada de [23].

paso la puerta de entrada decide qué valores se actualizarán para, posteriormente, alimentar el resultado a una capa \tanh , la cual se encarga de crear un vector candidato, \tilde{C}_t , para agregarlo al estado de la celda, esto se puede ver en la figura 3.39.

Al tener al vector candidato, se actualiza el estado anterior C_{t-1} al nuevo estado C_t , multiplicando al estado anterior por el resultado de la capa de olvido, f_t , olvidando lo que se decidió olvidar. Añadiendo a este resultado la multiplicación del vector candidato, \tilde{C}_t , con el resultado de la capa de entrada, i_t . Esto se puede ver en la figura 3.37.

Finalmente, se decide la salida, esto basado en C_t de forma filtrada al alimentarse a una capa sigmoide. Una vez teniendo el resultado, se agrega a C_t a una capa \tanh y se multiplica por el resultado anterior. Esto se puede ver en la figura 3.41.

Entrenamiento de LSTM

Como un LSTM mantiene la estructura de las redes anteriores, se entrena de igual forma usando SGD con una función de pérdida.

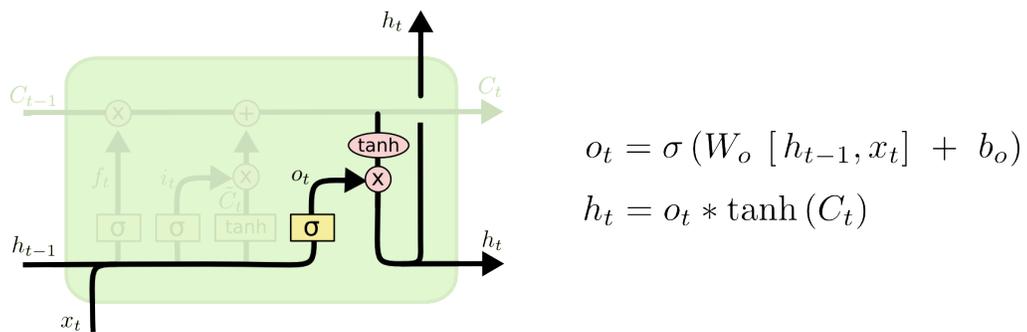


Figura 3.41: Izquierda: **Puerta de Salida:** Puerta encargada de administrar la salida que dará la celda. Tomada de [23].

Capítulo 4

Clasificación de series de tiempo

En este capítulo se buscan formas de clasificar secuencias dadas cajas que rodean a las manos, usando Faster R-CNN. Para ello se introduce Dynamic Time Warping[36] y su sucesor FastDTW[37] para tener una unidad de medida entre las secuencias. Una vez se tienen las diferencias entre secuencias se utiliza K-vecinos cercanos[38] para clasificar las diferentes señas en agrupaciones de acuerdo con las diferencias obtenidas. Una vez teniendo las cajas que rodean al objeto buscado, en nuestro caso las manos de una persona, lo que hacemos es “seguir” el movimiento de dicho objeto, almacenando las coordenadas de la esquina inferior izquierda y esquina superior derecha por cada detección. Al hacer esto se obtiene la trayectoria que tiene el objeto a través de todas las imágenes proporcionadas, esto se puede ver como una serie de tiempo de la trayectoria de la mano respecto al tiempo que tomó realizarse la seña. Teniendo las series de tiempo, quisiéramos una forma en la que se pudiera comparar las series de tiempo de dos objetos, sin importar que una sea más pequeña que la otra. Para ello utilizaremos *Fast Dynamic Time Warping* (FastDTW).

4.1. Dynamic Time Warping (DTW)

Antes de poder definir lo que es FastDTW, primero es necesario definir lo que es DTW, dado que para su funcionamiento FastDTW utiliza DTW.

“En análisis de series de tiempo, DTW es un algoritmo para medir las similitudes entre dos secuencias temporales, que pueden variar en tiempo o velocidad, por ejemplo, las similitudes en patrones de personas caminando pueden ser detectadas usando DTW, incluso cuando una persona caminara más rápido que la otra, o si hubiera aceleraciones y desaceleraciones durante el periodo de observación.”[39]

Este capítulo toma de base [36].

Como se menciona en la cita anterior, DTW mide la similitud entre dos secuencias independientes, llamadas $X := (x_1, x_2, \dots, x_N)$ que tendrá tamaño N y $Y := (y_1, y_2, \dots, y_M)$ que tendrá tamaño de tamaño M, donde N y M no son necesariamente del mismo tamaño. Para comparar

ambas secuencias, se utiliza una medida de costo local, definida por c , la cual se define como:

$$c : F \times F \rightarrow \mathbb{R}_{\geq 0} \quad (4.1)$$

Donde F corresponde a un espacio fijo de características, teniendo que $x, y \in F$.

Para esta medida de costo c se tiene que si $c(x, y)$ es pequeña indica que hay un costo bajo y x y y son similares, mientras que si la medida $c(x, y)$ es grande se tiene que hay un alto costo y x y y no son similares. Para poder determinar el resultado de la ecuación 4.1 se utiliza una matriz de costo, definida como $C(n, m) := c(x_n, y_m)$, que contiene los resultados de encontrar la medida costo local para cada par de elementos (x, y) , donde $x \in X, y \in Y$. La meta es encontrar un camino en esta matriz, llamada alineamiento, entre X y Y , teniendo éste el menor costo. Para poder encontrar un camino útil para nosotros definiremos un **camino alineado** (*warped path*). Un camino alineado es una secuencia $p = (p_1, \dots, p_L)$ con $p_l = (n_l, m_l) \in [1 : N] \times [1 : M]$ para $l \in [1 : L]$ que satisface las siguientes condiciones:

- Límites del camino : $p_1 = (1, 1)$ y $p_L = (N, M)$
- Condición de monotonicidad : $n_1 \leq n_2 \leq n_3 \leq \dots \leq n_L$ y $m_1 \leq m_2 \leq m_3 \leq \dots \leq m_L$
- Condición de un paso : $p_{l+1} - p_l \in \{(1, 0), (0, 1), (1, 1)\}$ para $l \in [1 : L - 1]$

El (N, M) -camino alineado define, como su nombre lo indica, un alineamiento entre la secuencia X y Y , asegurando que el primer y último elemento estén alineados (condición 1), el camino también respeta que si un elemento en X precede a otro esto también debe suceder para los elementos en Y (condición 2), y finalmente el camino es continuo, donde ningún elemento en X y Y puede ser omitido y no hay replicaciones en el alineamiento (condición 3).

El costo total del camino alineado, con respecto a la ecuación 4.1 como función de medida está dada por:

$$c_p(X, Y) := \sum_{l=1}^L c(X_{n_l}, Y_{m_l}) \quad (4.2)$$

Además, un camino alineado óptimo entre X y Y es un camino alineado p^* que tiene el menor costo entre todos los posibles caminos alineados, por lo que se puede definir a la distancia $DTW(X, Y)$ entre X y Y como el costo total de p^* de la siguiente manera:

$$DTW(X, Y) := C_{p^*}(X, Y) = \min\{c_p(X, Y) | p \text{ es un } (N, M)\text{-camino alineado}\} \quad (4.3)$$

Una idea para poder obtener el camino óptimo, p^* es probar todos los posibles caminos alineados entre X y Y , pero el realizar este procedimiento lleva a una complejidad exponencial con respecto a los tamaños de las secuencias. Para poder agilizar la búsqueda de p^* se introduce un algoritmo con complejidad $O(N, M)$, pero antes de definir el algoritmo se define lo siguiente:

$$X[1 : N] := (x_1, x_2, \dots, x_n) \quad (4.4)$$

$$Y[1 : M] := (y_1, y_2, \dots, y_m) \quad (4.5)$$

Estas dos ecuaciones se sustituyen en la ecuación 4.3, obteniendo la siguiente ecuación:

$$D(n, m) := DTW(X[1 : N], Y[1 : M]) \quad (4.6)$$

Aquí $D(n, m)$ es la matriz de costo acumulado, ésta satisface las siguientes condiciones:

- $D(n, 1) = \sum_{k=1}^n c(x_k, y_1)$.
- $D(1, m) = \sum_{k=1}^m c(x_1, y_m)$.
- $D(n, m) = \min\{D(n-1, m-1), D(n-1, m), D(n, m-1)\} + c(x_n, y_m)$, para $1 < n \leq N$ y $1 < m \leq M$.

En particular, la tercer condición nos facilita un cálculo recursivo de la matriz D , la inicialización de ésta puede ser hecha agregando un renglón, $D(n, 0) := \infty$ para $n \in [1 : N]$ y $D(0, m) := \infty$ para $m \in [1 : M]$, y $D(0, 0) := 0$, entonces la condición 3 se mantiene para n y m .

En base a esto, el algoritmo de DTW se puede visualizar en el algoritmo 2

Algoritmo 2 Algoritmo DTW para obtener el camino alineado óptimo, p^* , tomado de [36]

- 1: **procedure** DTW(D)
 - 2: **Entrada:** Una matriz de costo acumulado D .
 - 3: **Salida:** Un camino alineado óptimo, p^*
 - 4: **Procedimiento:** El camino óptimo p^* es calculado en reversa a partir de los índices empezando por $p_L = (N, M)$. Supongamos que $p_l = (n, m)$ ha sido calculado. En el caso de $(n, m) = (1, 1)$, uno debe tener $l = 1$ y se habrá terminado, de lo contrario:
 - 5: $p_{l-1} = \begin{cases} (1, m-1) & \text{si } n = 1, \\ (n-1, 1) & \text{si } m = 1, \\ \text{argmin}\{D(n-1, m-1), D(n-1, m), D(n, m-1)\} & \text{en otro caso} \end{cases}$
-

4.2. FastDTW

Si bien DTW logra encontrar el camino alineado más óptimo entre dos secuencias independientes, el ser de complejidad cuadrática provoca que su uso sea limitado al momento de encontrar los caminos alineados entre secuencias muy grandes, para ello se hicieron varios métodos que agilizan el encontrar p^* , estos métodos se pueden diferenciar en tres categorías:

- Restricciones: Al momento de evaluar la matriz de costo, se limita el número de celdas a ser evaluadas.
- Abstracción de la información: Se abstraen las secuencias originales en secuencias más pequeñas y se aplica DTW sobre las nuevas secuencias más pequeñas.
- Indexar: Se reduce el número de veces que se utiliza DTW durante la clasificación o agrupamiento de las series de tiempo.

El utilizar estos métodos agilizan a DTW por un factor constante grande, pero el algoritmo sigue siendo $O(N^2)$ en tiempo y espacio, y entre más se utilizan estos métodos más empieza a disminuir la precisión de p^* y no necesariamente se encuentra el camino óptimo. Para combatir

ello surge FastDTW, FastDTW utiliza un aproximamiento multinivel inspirado por el aproximamiento utilizado en la bisección de gráficas.

La bisección de una gráfica consiste en partir una gráfica en partes iguales, donde la suma de las aristas puede ser partida en la mayor cantidad de pedazos posibles. Un aproximamiento multinivel se puede utilizar para encontrar la solución en una gráfica pequeña, y ,una vez teniendo el resultado de dicha gráfica, se va aumentando el tamaño de la gráfica original, donde cada vez que se incrementa el tamaño se va 'arreglando' la solución anterior para ajustarse a la nueva gráfica. El problema de encontrar el camino óptimo entre dos secuencias puede ser resuelto utilizando el aproximamiento multinivel, donde FastDTW es capaz de encontrar un camino alineado preciso en tiempo y espacio lineal.

4.2.1. Algoritmo FastDTW

El algoritmo FastDTW utiliza tres operaciones importantes, éstas son:

- Encogimiento: Se encoje la serie de tiempo en una más pequeña que preserve la curvatura original.
- Proyección: Se encuentra un camino alineado en la serie de tiempo encogida, usándose este camino como el inicial para encontrar el camino alineado en la serie de tiempo original.
- Refinamiento: Se refina el camino alineado de la serie de tiempo encogida a través de pequeños ajustes para poder ser usado en la serie de tiempo original.

Encogimiento reduce el tamaño de la serie de tiempo al promediar los pares de puntos adyacentes entre sí, teniendo que la nueva serie de tiempo es más pequeña por un factor de dos, esto se ejecuta varias veces para producir múltiples series de tiempo de tamaño reducido, todas éstas siendo de la mitad de tamaño que la original a la que se le aplica el algoritmo.

La proyección toma un camino alineado de una serie de tiempo más pequeño y determina sobre qué casillas en la siguiente serie de tiempo (ésta siendo de un tamaño más grande pero no necesariamente la original) pasa a través, como el tamaño se va incrementando por un factor de dos, un punto en el camino alineado puede tener un mapeo de al menos 4 puntos, donde es mayor a 4 puntos si $\|X\| = \|Y\|$. Este camino proyectado se utiliza como heurística al momento de refinar la solución para encontrar al camino alineado en la siguiente serie de tiempo.

Refinamiento encuentra al camino alineado óptimo dentro del vecindario dentro del camino proyectado, donde el tamaño del vecindario se indica por medio de un parámetro llamado **radio**.

La razón por la que DTW toma $O(N^2)$ es porque al momento de llenar la matriz de costo cada casilla debe de ser llenada para asegurar que se encuentra el camino alineado óptimo, y el tamaño de la matriz crece de forma cuadrática con respecto al tamaño de las series de tiempo, pero para FastDTW la matriz de costo sólo se llena en el vecindario dentro del camino proyectado de la serie de tiempo anterior, y como la longitud del camino alineado crece linealmente con el tamaño de la entrada de las series de tiempo FastDTW es un algoritmo de $O(N)$.

¿Cómo funciona FastDTW?

Primero FastDTW utiliza encogimiento para crear todas las series de distintos tamaños que

serán evaluadas, utilizándose DTW sobre la serie de tiempo más pequeña para encontrar el camino alineado óptimo que será proyectado en la siguiente serie de tiempo, la serie que es más grande por un factor de dos, para refinar el camino proyectado. DTW únicamente se utiliza sobre las celdas en el camino proyectado, por lo que puede ser que el camino óptimo en el nuevo tamaño no se encuentre dentro del camino proyectado, para ello se utiliza el **radio** que controla el número de celdas adicionales en cada lado del camino proyectado que serán evaluados junto con el camino proyectado. El algoritmo completo puede verse en el algoritmo 3.

Algoritmo 3 Algoritmo FastDTW para obtener el camino mínimo alineado, p^* , en $O(N)$ tomado de [37]

```

1: procedure FASTDTW( $X, Y, r$ )
2:   Entrada:  $X$ , una serie de tiempo de tamaño  $\|X\|$ 
3:    $Y$ , una serie de tiempo de tamaño  $\|Y\|$ 
4:    $r$ , la distancia de búsqueda fuera del camino proyectado.
5:   Salida: Un camino alineado de distancia mínima entre  $X$  y  $Y$ 
6:   // El tamaño mínimo de la resolución encogida
7:   La distancia del camino alineado entre  $X$  y  $Y$ 
8:   Procedimiento:
9:    $minTSsize = r + 2$ .
10:  si  $\|X\| \leq minTSOR\|Y\| \leq minTSsize$  entonces
11:    // Caso Base: Para una serie de tiempo de tamaño pequeño se utiliza el algoritmo
    DTW
12:    Return  $DTW(X, Y)$ 
13:  else
14:    // Caso Recursivo: Se proyecta el camino alineado desde el tamaño reducido hacia
    el tamaño actual. Se corre DTW únicamente a través del camino proyectada, junto con las
    celdas dentro del rango del radio
15:     $shrunkX = X.reduceByHalf()$ 
16:     $shrunkY = Y.reduceByHalf()$ 
17:     $lowResPath = FastDTW(shrunkX, shrunkY, r)$ 
18:     $window = ExpandedResWindow(lowResPath, X, Y, r)$ 
19:    Return  $DTW(X, Y, window)$ 

```

FastDTW no garantiza siempre encontrar el camino óptimo entre dos series de tiempo, pero en caso de no hacerlo encuentra un camino mínimo óptimo, para siempre poder encontrar el camino alineado óptimo como en DTW se tendría que incrementar el parámetro del radio para que éste fuera del tamaño de al menos una serie de tiempo de entradas, pero esto sería una generalización de DTW y por ende FastDTW tendría una ejecución de $O(N^2)$.

4.3. K vecinos cercanos

Esta sección es tomada de [38].

El algoritmo de K vecinos cercanos es uno de los algoritmos de clasificación más fundamentales y simples cuando se tiene poca o nula información sobre el conjunto de información sobre el que será aplicado. El algoritmo K vecinos cercanos nos ayuda a resolver el problema donde, dado un conjunto X de muestras, se busca identificar a K distintas clases en base a alguna unidad de medida. Supongamos que tenemos un conjunto de entrada, $C_E = (x_1, x_2, \dots, x_n)$, donde cada elemento tiene p características, definidas como $(x_{i1}, x_{i2}, \dots, x_{ip})$, teniendo esto podemos definir la distancia Euclidiana entre el ejemplar x_i y x_j en la ecuación 4.7.

$$d(x_i, x_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \dots + (x_{ip} - x_{lp})^2} \quad (4.7)$$

Una forma de poder visualizar cómo funciona el algoritmo de K vecinos cercanos, es utilizando diagramas de Voronoi, como se muestra en la figura 4.1. Un diagrama de Voronoi se define como una partición de un plano en regiones basado en las distancias a un punto en particular del plano por celda, formalmente está definido en la ecuación 4.8.

$$R_i = \{x \in X : d(x, x_i) \leq d(x, x_m), \forall i \neq m\} \quad (4.8)$$

Donde R_i es la celda o región de Voronoi que encapsula a los elementos dentro del conjunto de X que satisfacen la condición de tener una distancia menor o igual al ejemplar x_i que al ejemplar x_m . El diagrama de Voronoi posee dos importantes características:

- Todos los posibles puntos dentro de un ejemplar de Celda de Voronoi son los vecinos más cercanos al ejemplar característico x_i de esa celda.
- Para cualquier ejemplar, el ejemplar más cercano está determinado por la región de Voronoi más cercana.

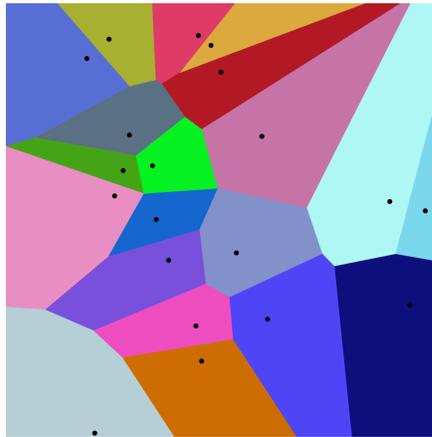


Figura 4.1: Diagrama de Voronoi, donde cada color una región de Voronoi o clase, imagen tomada de [40]

Viendo ambas características, se observa que la forma en que K vecinos cercanos logra clasificar es asignando una etiqueta de clase a cada casilla de Voronoi y, por medio de un esquema de votos, verifica qué clase es más frecuente dentro de los K vecinos más cercanos, i.e de las K casillas más cercanas.

Generalmente K suele ser impar para evitar que al contar los vecinos cercanos se tenga un empate, y en el caso donde $K = 1$ se le conoce como clasificación del vecino más cercano.

4.3.1. Matriz de confusión

Para poder saber qué tan bien clasifica un algoritmo de clasificación se utiliza una matriz de confusión, esta matriz funciona tomando en cuenta las veces en que el ejemplar dentro del conjunto de prueba se predijo correcta o incorrectamente usando como entrenamiento el conjunto de entrenamiento. De manera más formal, sea x_i un ejemplar dentro del conjunto de entrenamiento y x_j un ejemplar dentro del conjunto de prueba, donde x_i tiene la etiqueta de clase ω y se clasificó a x_j con la etiqueta de clase $\hat{\omega}$, donde $\hat{\omega} \in \Omega$ contiene todas las etiquetas de clase del conjunto de entrenamiento. Usando lo anterior, la matriz de confusión, que será de tamaño $\Omega \times \Omega$ se construye de la siguiente manera:

$$\text{Matriz} = \begin{cases} C_{(\omega, \hat{\omega})} = 0 & \text{si } \omega \neq \hat{\omega}, \\ C_{(\omega, \hat{\omega})} = 1 & \text{si } \omega = \hat{\omega} \end{cases} \quad (4.9)$$

Una vez se han clasificado todos los elementos dentro del conjunto de prueba, se calcula la precisión del clasificador usando la siguiente ecuación:

$$Acc = \frac{\sum_{\omega} c_{(\omega, \omega)}}{n_{total}} \quad (4.10)$$

Para nuestro caso se utiliza K vecinos cercanos, con la medida de distancia siendo FastDTW sobre las series de tiempo.

Capítulo 5

Experimentación

Teniendo en cuenta todo lo definido anteriormente, se harán dos experimentos distintos para la detección de las señas. En el primer experimento se usará un detector de manos Faster R-CNN para alimentar las posiciones de las manos a un clasificador KNN, el cual usará FastDTW para calcular las diferencias entre secuencias. Este experimento tendrá un porcentaje precisión muy bajo, por lo que no le será posible distinguir las posibles señas en los videos, por lo que será necesario realizar cambios a la forma en que se aborda el problema, cosa que se realizará en los siguientes experimentos. En el segundo experimento, se utiliza el detector de pose OpenPose, con el cual se obtendrán los puntos clave de las manos, brazos y cara. Estos puntos clave serán alimentados a una red LSTM, la cual clasificará las señas en base a las secuencias obtenidas.

5.1. Primer Experimento: Faster R-CNN + KNN

Este experimento está basado en [5], usando Faster R-CNN para la detección de manos, FastDTW para la unidad de medida y KNN para la agrupación de resultados en busca de patrones.

Una vez teniendo el detector Faster R-CNN y los algoritmos definidos anteriormente, es posible empezar a juntarlos para clasificar las distintas señas con movimiento dentro del LSM que fueron seleccionadas, primero es necesario entrenar el detector Faster R-CNN.

5.1.1. Entrenamiento del detector

Para entrenar al detector Faster R-CNN se utiliza un conjunto de 14,251 imágenes, donde en éstas se ubican una o más manos, estas imágenes fueron tomadas de tres distintos conjuntos de datos, el primero es el dataset *EgoHands: A Dataset for Hands in Complex Egocentric Interactions*[41], que contiene 48 videos grabados por medio de Google Glass de interacciones complejas entre dos personas, donde estos videos son vistos por medio de la perspectiva de una persona. También se utiliza *Large-scale Multiview 3D Hand Pose Dataset*[42], que contiene varias vistas de manos haciendo distintas poses, y finalmente se utilizan varios cuadros de videos grabados en el servicio social para agregar imágenes donde hay personas haciendo señas,

donde estas imágenes son aumentadas utilizando aumento de información, esto por medio del repositorio de github *Image augmentation for machine learning experiments*. [43], que aplica varios filtros y/o rotaciones a las imágenes para ir generando nuevas imágenes que preservan la identidad del objeto a ser reconocido, donde en nuestro caso son las manos.

Muestras de los distintos dataset se pueden ver en la figura 5.1.

Una vez teniendo todo lo anterior, ya es posible entrenar a nuestro modelo, para entrenarlo se



Figura 5.1: Se observan cuatro imágenes de cada dataset usado, el primer renglón pertenece a [41], el segundo a [42] y el tercero al generado a través de los videos grabados durante el servicio social

utiliza *Tensorflow*, en particular, *Tensorflow Object detection API* [44], el cual provee varios modelos de redes y formas de entrenarlos. Dentro de los modelos provistos está *faster rcnn 101 kitti*, que es el modelo previamente definido, Faster R-CNN utilizando Resnet-101 para generar las regiones, donde este modelo es pre entrenado por medio de *KITTI Vision data set* [45], se entrena usando la configuración por defecto, cambiando el número de clases a detectar por 1 (la mano). La configuración es la siguiente:

```

model {
  faster_rcnn {
    num_classes: 1
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1987
      }
    }
  }
}

```

```
feature_extractor {
  type: "faster_rcnn_resnet101"
  first_stage_features_stride: 16
}
first_stage_anchor_generator {
  grid_anchor_generator {
    height_stride: 16
    width_stride: 16
    scales: 0.25
    scales: 0.5
    scales: 1.0
    scales: 2.0
    aspect_ratios: 0.5
    aspect_ratios: 1.0
    aspect_ratios: 2.0
  }
}
first_stage_box_predictor_conv_hyperparams {
  op: CONV
  regularizer {
    l2_regularizer {
      weight: 0.0
    }
  }
  initializer {
    truncated_normal_initializer {
      stddev: 0.009999999776482582
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.699999988079071
first_stage_max_proposals: 100
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    fc_hyperparams {
      op: FC
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
    }
    initializer {
      variance_scaling_initializer {
        factor: 1.0
      }
    }
  }
}
```

```
        uniform: true
        mode: FAN_AVG
    }
}
}
use_dropout: false
dropout_keep_probability: 1.0
}
}
second_stage_post_processing {
  batch_non_max_suppression {
    score_threshold: 0.30000001192092896
    iou_threshold: 0.6000000238418579
    max_detections_per_class: 100
    max_total_detections: 100
  }
  score_converter: SOFTMAX
}
second_stage_localization_loss_weight: 2.0
second_stage_classification_loss_weight: 1.0
}
}
train_config {
  batch_size: 1
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}
optimizer {
  momentum_optimizer {
    learning_rate {
      manual_step_learning_rate {
        initial_learning_rate: 9.99999747378752e-05
        schedule {
          step: 500000
          learning_rate: 9.99999747378752e-06
        }
        schedule {
          step: 700000
          learning_rate: 9.9999974752427e-07
        }
      }
    }
  }
  momentum_optimizer_value: 0.899999761581421
}
use_moving_average: false
}
gradient_clipping_by_norm: 10.0
fine_tune_checkpoint: "faster_rcnn_resnet101_kitti_2018_01_28/model.ckpt"
from_detection_checkpoint: true
```

```

    num_steps: 800000
  }
  train_input_reader {
    label_map_path: "data/object-detection.pbtxt"
    tf_record_input_reader {
      input_path: "data/train.record"
    }
  }
}
eval_config {
  num_examples: 500
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "data/object-detection.pbtxt"
  tf_record_input_reader {
    input_path: "data/test.record"
  }
}
}

```

Donde FC hace referencia a las capas completamente conectadas (*Fully Connected*). En particular se observa que se utiliza SOFTMAX para el post proceso en la segunda etapa, se utiliza max pool, se utiliza el operador convolución, se utiliza la regularización L2 en ambas etapas. También se observa que se encuentra de la configuración, en el apartado de inicializar la varianza, *mode : Fan AVG*. Esto indica cómo se escalará la varianza que se va comunicando entre las capas, donde en nuestro caso se escala tomando un promedio entre el número de conexiones entrantes y de salida ($n = (\text{fan in} + \text{fan out})/2.0$), conocida como **xavier initializer**. Se entrena por un total de 225,000 iteraciones, obteniendo los resultados mostrados en las figuras 5.2, 5.3 y 5.4. Se observa que no hay gran cambio en el entrenamiento del clasificador, esto se debe a que el clasificador **ya se encontraba pre-entrenado** con el conjunto de datos KITTI, por lo que pudo haberse omitido el entrenamiento y no hubiera habido cambios respecto a la detección.

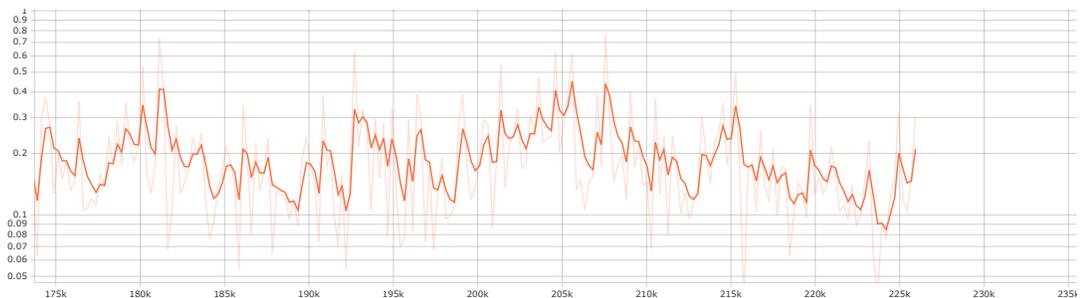


Figura 5.2: Gráfica de pérdida total durante el entrenamiento

```

Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.687
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.980
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.829
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.700
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.665
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.717
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.436
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.747
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.747
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.700
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.730
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.775
INFO:tensorflow:Writing metrics to tf summary.
INFO:tensorflow:DetectionBoxes_Precision/mAP: 0.686589
INFO:tensorflow:DetectionBoxes_Precision/mAP (large): 0.716564
INFO:tensorflow:DetectionBoxes_Precision/mAP (medium): 0.664506
INFO:tensorflow:DetectionBoxes_Precision/mAP (small): 0.700000
INFO:tensorflow:DetectionBoxes_Precision/mAP@.50IOU: 0.979968
INFO:tensorflow:DetectionBoxes_Precision/mAP@.75IOU: 0.828723
INFO:tensorflow:DetectionBoxes_Recall/AR@1: 0.435528
INFO:tensorflow:DetectionBoxes_Recall/AR@10: 0.747446
INFO:tensorflow:DetectionBoxes_Recall/AR@100: 0.747446
INFO:tensorflow:DetectionBoxes_Recall/AR@100 (large): 0.774566
INFO:tensorflow:DetectionBoxes_Recall/AR@100 (medium): 0.729963
INFO:tensorflow:DetectionBoxes_Recall/AR@100 (small): 0.700000
INFO:tensorflow:Losses/Loss/BoxClassifierLoss/classification_loss: 0.081319
INFO:tensorflow:Losses/Loss/BoxClassifierLoss/localization_loss: 0.105424
INFO:tensorflow:Losses/Loss/RPNLoss/localization_loss: 0.027313
INFO:tensorflow:Losses/Loss/RPNLoss/objectness_loss: 0.120040
INFO:tensorflow:Metrics written to tf summary.

```

Figura 5.3: Resultados al probar el clasificador en el conjunto de prueba, se observa que hay un mAP (Precisión promedio) de 0.686589 al promediar sobre todas las IoU (Intersección sobre la Unión)

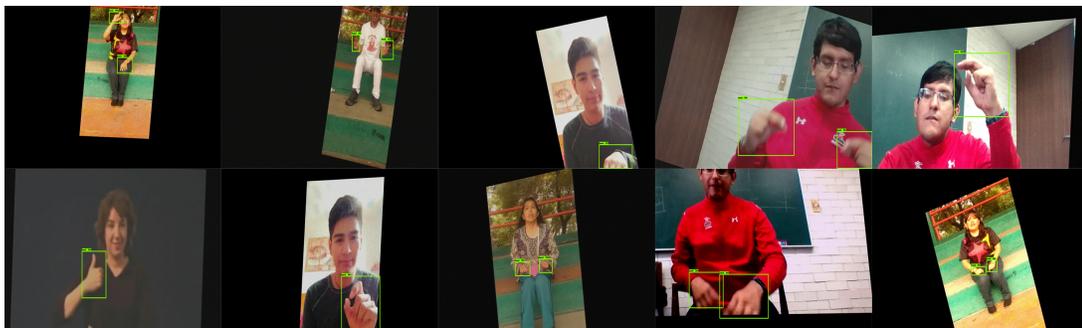


Figura 5.4: Resultados al probar el clasificador en el conjunto de prueba

5.1.2. Entrenando al clasificador

Una vez teniendo el detector de manos funcionando, es posible pasar a la siguiente tarea, la cual es la clasificación de señas. Para esta tarea se usarán los videos del programa DIELSEME[46], donde se eligen pseudo aleatoriamente 28 números, los cuales nos indicarán a las 28 señas a usar, donde se eligen a las siguientes.

- Agua: (st f) Líquido que, cuando está limpio, no tiene sabor, color, ni olor; se encuentra en ríos, lagos, mares y en otros lugares como las nubes, de donde cae en forma de lluvia; cuando está muy fría se vuelve hielo y cuando se hierve se vuelve vapor
- Alto: (aj) Que mide mucho, desde abajo hacia arriba
- Apagar: (vb) Hacer que se acabe el fuego o la corriente eléctrica de las lámparas u otros aparatos como la televisión o la radio
- Ayudar: (vb) Apoyar, cooperar para que se haga algo
- Campo: (st m) Terreno fuera de la ciudad donde se siembran y cultivan las verduras, semillas y frutos que comemos
- Crecer: (vb in) Desarrollarse un ser vivo
- Cuerpo: (st m) Conjunto de las partes que forman a las personas o a los animales
- Despedir: (vb, vb pn) Correr a alguien de un trabajo
- Dinero: (st m) Conjunto de monedas y billetes que sirven para comprar
- Escribir: (vb) Poner palabras o números en un papel u otro lugar
- Foto: (st f) Fotografía
- Fruto: (st f) Producto de las plantas que es carnoso y jugoso, como la naranja, la pera o el mango
- Hacer: (vb) Crear, producir, provocar algo
- LavarCara: Lavar cara (vb pn) Limpiar uno mismo una parte de su cuerpo
- Madera: (st f) Parte dura de los árboles, la de adentro del tronco; sirve para hacer muebles, puertas, pisos y muchas cosas más, como el papel.
- Miedo: (st m) Sentimiento que llega cuando algo te asusta, o cuando crees que hay algún peligro o algo malo puede pasar
- Morder: (vb) Clavar los dientes con fuerza en algo. Nota: La seña del verbo cambia según el sujeto que muerde u objeto que se muerde
- Nariz: (st f) Parte saliente de la cara de una persona; tiene dos huecos por donde se respira y se huele

- Padrino: (st m) Hombre que presenta o acompaña a alguien, que es su ahijado, a una ceremonia como el bautizo o una boda, y se compromete a ayudarlo
- Pelicula: (st f) Cinta que sirve para tomar fotos o hacer cine.
- Persona: (st f) Ser humano, hombre, mujer, niña o niño
- Postre: (st m) Platillo, generalmente dulce, que te sirven al final de la comida
- Primero: (aj) Que está antes
- Quedar: (vb in) Estar algo o alguien en un lugar
- Saber: (vb) Tener conocimiento de algo
- Sangre: (st f) Líquido de color rojo que circula por las arterias y las venas de las personas y algunos animales
- Senorita: (st f) Mujer joven o que no está casada
- Suerte: (st f) Lo que pasa por casualidad, sin que lo esperes
- Toro: (st m) Macho de la vaca; tiene dos cuernos grandes en la cabeza, y su carne se come

Éstas serán realizadas por diez personas varias veces, dándonos un total de 1,457 videos, donde se acelerará y alentará el video, por una velocidad de 0.25, 0.50, 1.25 y 1.50, esto para poder tener una mayor cantidad de videos, resultando en un total de 5,831 videos. Posteriormente se dividen los videos en un 80 % y 20 % para tener un conjunto de entrenamiento y un conjunto de prueba, teniendo 1,334 videos en el conjunto de prueba y 4,497 en el de entrenamiento.

Trayecto puro

Antes de poder empezar a ver las manos dentro de los videos que se tienen, es necesario definir qué es lo que se hará con éstas, para ello se introduce el concepto del trayecto puro, el cual se define de la siguiente manera:

- Primero, se observa si hay movimiento de la mano, i.e si se ha detectado que la posición de la mano ha cambiado en relación al cuadro anterior, de ser así se empieza a registrar el trayecto puro, de lo contrario se espera a que se detecte movimiento
- Después, se continúa generando el trayecto puro mientras se siga detectando movimiento dentro de los cuadros de video.
- Finalmente, cuando se termina el video se termina de generar el trayecto puro

Tomando en cuenta lo anterior, se va generando el camino que se usará para registrar la trayectoria de la siguiente manera:

$$\begin{aligned} X_c &= \frac{1}{n} \sum_{t=1}^n x_t \\ Y_c &= \frac{1}{n} \sum_{t=1}^n y_t \\ \theta_t &= \arctan \left(\frac{Y_c - y_t}{X_c - x_t} \right) \end{aligned} \quad (5.1)$$

Donde X_c y Y_c son los puntos de centro, n la longitud del trayecto puro que se ha detectado hasta este punto, esto por cada mano. Es decir, se va almacenando el ángulo de cada seña θ_t , como la trayectoria original de la seña, cambiándose el centro de la seña acorde a donde se vayan detectando las manos.

Este método está basado en el ángulo (orientación) como característica principal del movimiento, donde la orientación está basada en (X_c, Y_c) donde X_c y Y_c son el centro de gravedad del trayecto puro, siendo calculadas por la ecuación 5.1, donde al ángulo de orientación θ_t , es dividido entre 30 para cuantificar los posibles escenarios entre doce, donde el vector obtenido como trayecto puro será el alimentado hacia DTW.

Estos conceptos se pueden ver en las figuras 5.5 y 5.6.

Como puede suceder que el detector de manos falle, se implementa una medida de seguridad si de un cuadro donde se detecte a la mano en la posición (5, 12), en el siguiente cuadro no se detecta la mano debiéndose detectar, se asume que la posición de la mano en el cuadro no detectado será la misma que la posición en el cuadro anterior, para poder permitir un trayecto puro más fiable.

5 Vecinos Cercanos

Teniendo el trayecto puro definido, se itera sobre todos los videos (tanto de entrenamiento como de prueba) para generar un archivo numpy, el cual contiene el trayecto puro de la mano izquierda y la mano derecha, éste siendo almacenada como una tupla, i.e $[[1,2],[4,5],\dots,[2,5]]$ Teniendo estos archivos, se genera un pickle con los trayecto puro del conjunto de entrenamiento, para poder alimentar al clasificador de vecinos cercanos. Teniendo el pickle, se implementa 5-NN, se entrena asignando cada elemento dentro del pickle a un vecino, teniendo que al clasificar un sólo video se harán 4,497 comparaciones con FastDTW, donde se elegirán las 5 distancias menores. Para visualizar de mejor manera las comparaciones, mostraré dos cálculos distintos de FastDTW, de dos ejemplares de prueba distintos. Primero tomaremos un ejemplar perteneciente a la clase de Agua, y lo compararemos con un ejemplar de entrenamiento, de la misma clase, se pueden observar los resultados en las figuras 5.7 y 5.8, en este ejemplo no se observa mucho movimiento de la mano derecha, por lo que se visualiza otro ejemplo, en este caso ambos ejemplares de la clase Toro, en la figura 5.8

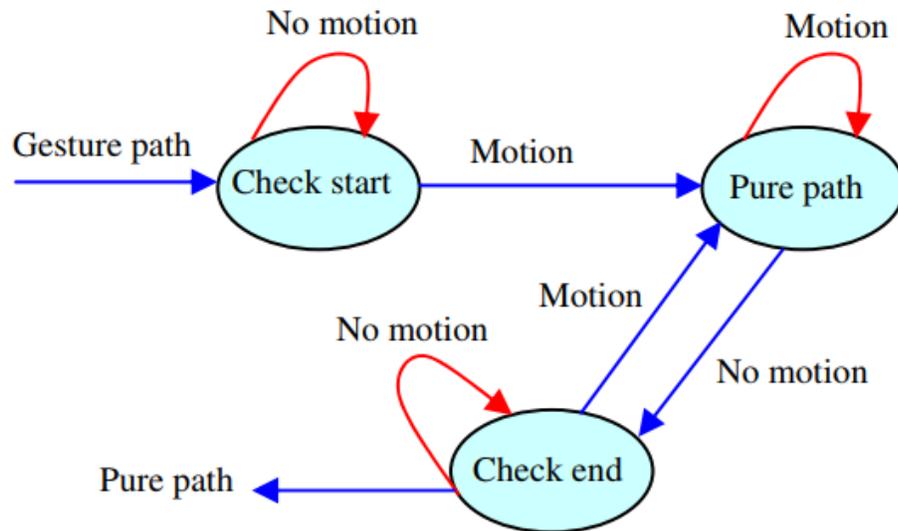


Figura 5.5: Máquina de estados donde se indican los posibles estados al hacerse el trayecto puro

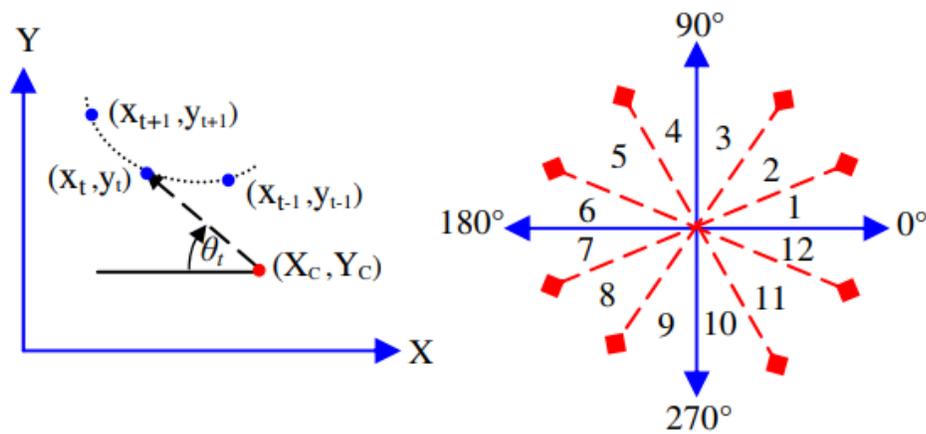
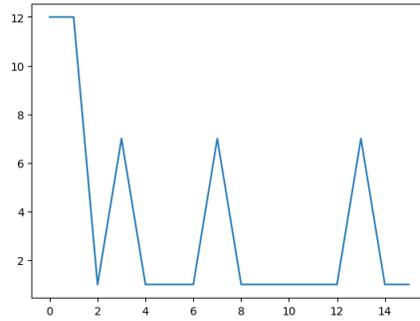
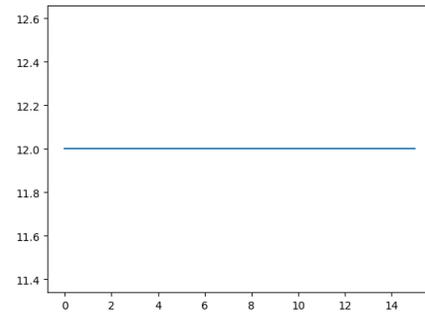


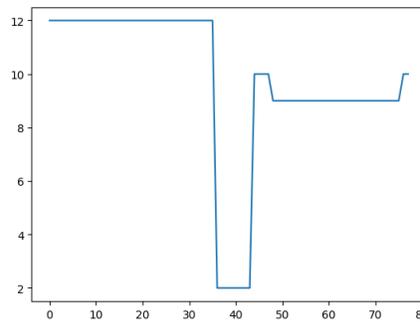
Figura 5.6: Ángulos resultantes al momento de calcular el trayecto puro



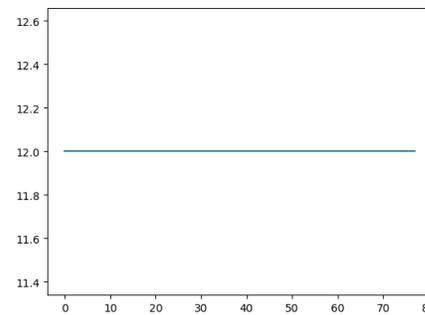
(a) Trayecto puro de la mano izquierda del ejemplar de entrenamiento perteneciente a la clase Agua



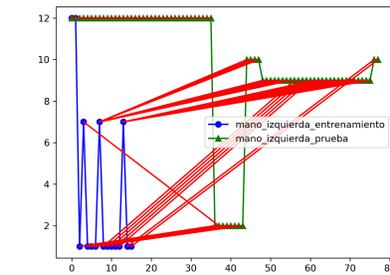
(b) Trayecto puro de la mano derecha del ejemplar de entrenamiento perteneciente a la clase Agua



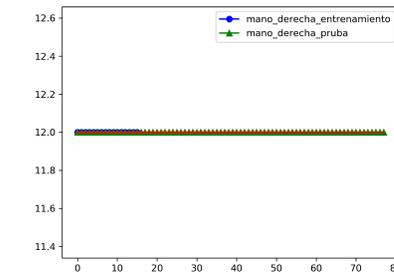
(c) Trayecto puro de la mano izquierda del ejemplar de prueba perteneciente a la clase Agua



(d) Trayecto puro de la mano derecha del ejemplar de prueba perteneciente a la clase Agua

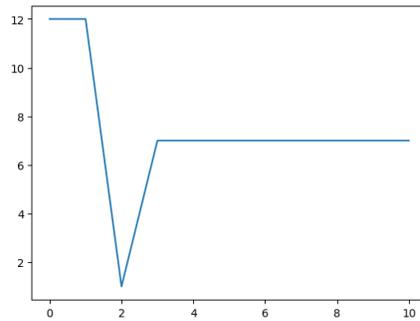


(e) FastDTW entre la mano izquierda del ejemplar de entrenamiento y mano izquierda del ejemplar de prueba

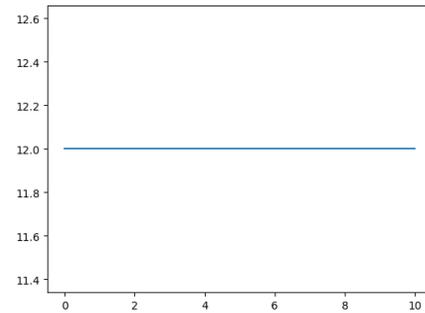


(f) FastDTW entre la mano derecha del ejemplar de entrenamiento y mano derecha del ejemplar de prueba

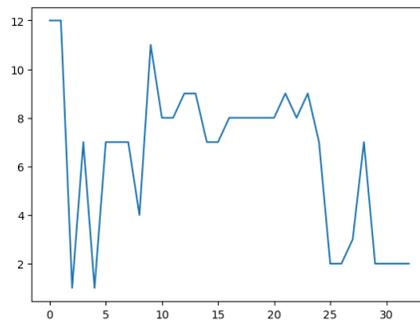
Figura 5.7: FastDTW entre la mano derecha del ejemplar de entrenamiento y mano izquierda del ejemplar de prueba.



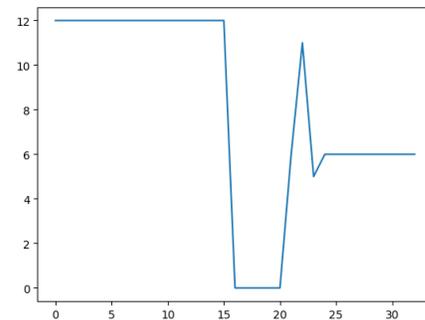
(a) Trayecto puro de la mano izquierda del ejemplar de entrenamiento perteneciente a la clase Toro



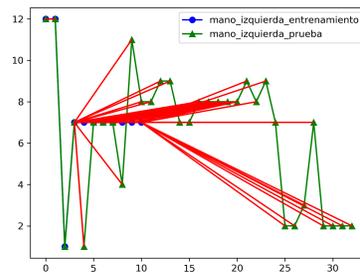
(b) Trayecto puro de la mano derecha del ejemplar de entrenamiento perteneciente a la clase Toro



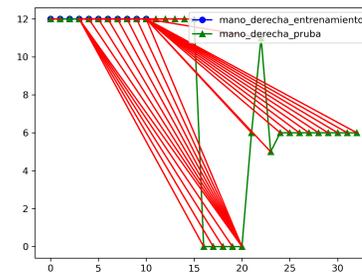
(c) Trayecto puro de la mano izquierda del ejemplar de prueba perteneciente a la clase Toro



(d) Trayecto puro de la mano derecha del ejemplar de prueba perteneciente a la clase Toro



(e) FastDTW entre la mano izquierda del ejemplar de entrenamiento y mano izquierda del ejemplar de prueba



(f) FastDTW entre la mano derecha del ejemplar de entrenamiento y mano derecha del ejemplar de prueba

Figura 5.8: FastDTW entre la mano derecha del ejemplar de entrenamiento y mano izquierda del ejemplar de prueba

5.2. Pruebas

Teniendo el clasificador 5-NN ya es posible hacer las pruebas para ver qué tan preciso es nuestro clasificador, para ello se itera sobre los archivos npy obtenidos previamente, en este caso siendo sobre los ejemplares de prueba, y se empiezan a calcular las distancias del ejemplar en el conjunto de prueba hacia todos los ejemplares dentro de los vecinos cercanos, obteniendo la etiqueta “predecida”, que es la clase a la cual se predice pertenece el video de prueba, y la etiqueta predicha, que es la clase a la cual pertenece correctamente el video de prueba. Se realiza todos esto con cada video de entrenamiento, generando un archivo que contiene todas las etiquetas verdaderas, y todas las etiquetas verdades, esto para generar la matriz de confusión y ver qué tan preciso es el detector. Los resultados se pueden ver en las figuras 5.9 y 5.1

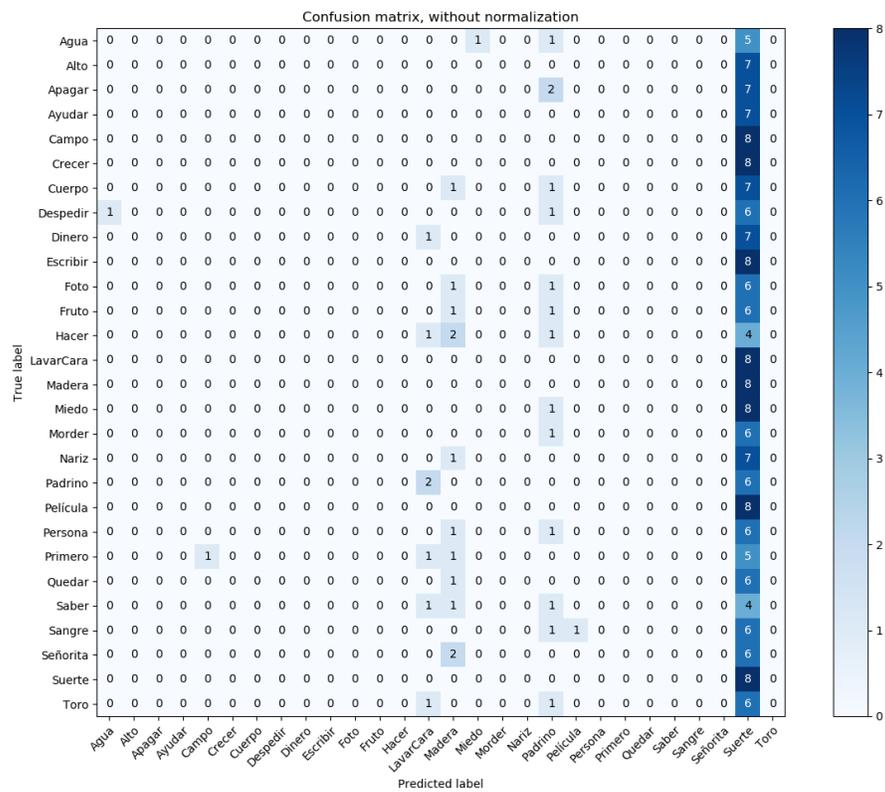


Figura 5.9: Matriz de confusión

5.2.1. Interpretación de resultados

Visualizando los resultados en la figura 5.9 se aprecian que la gran mayoría de las señas fueron clasificadas en una sola seña, **suerte**, por lo que el clasificador no es para nada preciso.

Tabla 5.1: Clasificación de las clases en el primer experimento.

Clase	Veces clasificada correctamente	Veces clasificada incorrectamente	Veces totales	Porcentaje de correctud
Agua	18	21	39	46 %
Alto	21	16	37	57 %
Apagar	67	18	85	79 %
Ayudar	20	17	37	54 %
Campo	26	15	41	63 %
Crecer	32	9	41	78 %
Cuerpo	27	17	44	61 %
Despedir	25	14	39	64 %
Dinero	25	15	40	62 %
Escribir	27	13	40	68 %
Foto	24	18	42	57 %
Fruto	23	17	40	57 %
Hacer	16	24	40	40 %
Lavar Cara	23	19	42	55 %
Madera	25	15	40	62 %
Miedo	20	26	46	43 %
Morder	24	12	36	67 %
Nariz	20	20	40	50 %
Padrino	25	16	41	61 %
Película	21	20	41	51 %
Persona	26	14	40	65 %
Primero	25	16	41	61 %
Quedar	29	8	37	78 %
Saber	19	17	36	53 %
Sangre	22	18	40	55 %
Señorita	29	12	41	71 %
Suerte	27	13	40	68 %
Toro	26	14	40	65 %

Posteriormente se elimina la clase **suerte** del conjunto de posibles señas (tanto en entrenamiento como prueba) y, nuevamente, las señas son clasificadas en una sola seña, **toro**, volviendo a tener terribles resultados. Por lo que se puede concluir que el clasificador únicamente junta a todas las clases dentro de una, siéndole imposible poder diferenciar entre clases en base al conjunto de entrada.

5.2.2. Conclusión

En base a los resultados, y a las pruebas subsecuentes, se observa que la clasificación basada en KNN y FastDTW no es capaz de diferenciar entre las distintas señas que se registraron en los videos.

¿A qué se debe esto?

Una de las razones por las que este clasificador no es preciso consiste en que únicamente revisa la trayectoria de ambas manos, comparando la trayectoria de una mano con las trayectorias de la misma mano. Por lo que en señas donde únicamente se utiliza una mano, como agua o campo, se concentra únicamente en la trayectoria de una mano para diferenciar entre señas, y al ser los movimientos posibles muy primitivos (arriba, abajo, izquierda,...) y la única forma de saber si una mano está abierta o no es por medio del tamaño de la caja que rodea a la mano, esto por medio de la diferencia entre cajas en distintos puntos del video.

Otra razón es que es muy poca información que regresa FastDTW para saber si una seña realmente es parecida a otra seña, únicamente regresa un número que, entre más grande sea, más distinta es la trayectoria de la mano respecto a la mano siendo comparada, y al ser los videos de una seña distintos entre sí, se tiene que la diferencia entre estos videos puede ser más grande que la diferencia entre señas de distintas clases.

Por lo que se recomienda un clasificador más robusto y que considere de mejor manera los movimientos de las manos para poder clasificar de forma más precisa las señas. Siendo en este caso, el uso de OpenPose para la detección de las manos y de LSTM para la clasificación de señas.

5.3. Segundo Experimento: OpenPose + LSTM

Este segundo experimento funciona al tomar un mayor número de las características del señante para poder tener un patrón más claro, todas las características siendo tomadas por medio de Openpose. Está basado en un experimento[7] del KETI (Instituto de Tecnología Electrónica de Corea), donde buscan traducir oraciones en el lengua de señas coreano, por medio de secuencias de palabras, apreciándose varias relaciones entre palabras y oraciones en la tabla 5.10.

Primero, empezaremos mostrando el uso de OpenPose, explicando las características que se obtienen, para posteriormente dirigirnos hacia la clasificación de las secuencias.

5.3.1. Detección de puntos clave por medio de OpenPose

Usaremos Openpose para la detección de 124 puntos clave de una persona, donde se clasifican de la siguiente manera.

- 12 puntos clave que corresponden a la estimación de pose del cuerpo.
- 70 puntos clave que corresponden a la estimación de pose de la cara.
- 2x21 puntos clave que corresponden a la estimación de pose de las manos.

ID	Korean Sentence	English sentence	Sign gloss
1	화상을 입었어요.	I got burned.	FIRE SCAR
2	폭탄이 터졌어요.	The bomb went off.	BOMB
3	친구가 숨을 쉬지 않아요.	My friend is not breathing.	FRIEND BREATHE CANT
4	집이 흔들려요.	The house is shaking.	HOUSE SHAKE
5	집에 불이 났어요.	The house is on fire.	HOUSE FIRE
6	가스가 새고 있어요.	Gas is leaking.	GAS BROKEN FLOW
7	112에 신고해주세요.	Please call 112.	112 REPORT PLEASE
8	도와주세요.	Help me.	HELP PLEASE
9	너무 아파요.	It hurts so much.	SICK
10	무릎 인대를 다친 것 같아요.	I hurt my knee ligament.	KNEE LIGAMENT SCAR

Figura 5.10: Diez ejemplos del anotador de señas. Se anota cada seña con cinco oraciones del lenguaje natural coreano y una única glosa de señas.

Donde juntaremos todos estos puntos clave en arreglos de la forma $[V0, V1, T]$, donde $V0$ es el vector con las posiciones X de todos los puntos clave, $V1$ es el vector con las posiciones Y de todos los puntos clave, y T es la etiqueta del video. Pero, antes de llegar a ello, hablaremos de los videos usados y su preparación antes de ser alimentados a openpose. Los puntos a usar se pueden apreciar en la figura 5.11.

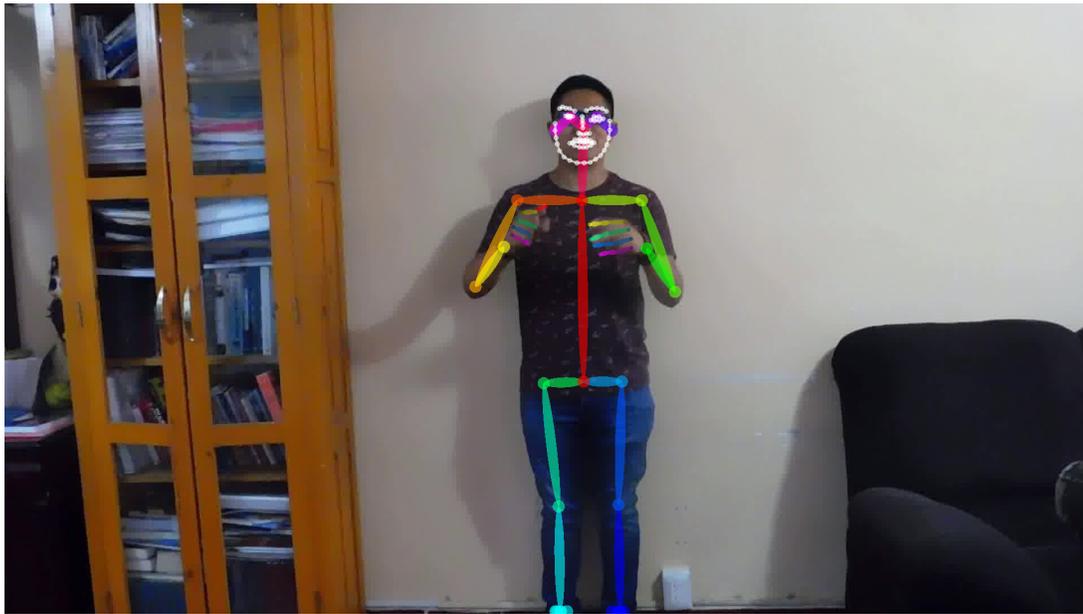


Figura 5.11: Puntos clave usados para open pose.

Grabación y manipulación de Videos

Para este experimento se grabaron a 6 personas, que serán los señantes, grabando cada una de las 28 señas 10 veces. Esto nos genera 60 videos por seña, 1680 videos en total.

Una vez teniendo estos videos, se generan 5 videos a partir de cada uno, esto de la siguiente manera: Dado un video cualquiera de una seña $S = (f_1, f_2, \dots, f_l)$ que contiene l cuadros de video de f_1 a f_l , se seleccionan n cuadros, en nuestro caso 15, de manera aleatoria, calculando la longitud de espacio entre cada cuadro de video de la siguiente manera:

$$z = \left\lfloor \frac{l}{n-1} \right\rfloor = \left\lfloor \frac{l}{14} \right\rfloor \quad (5.2)$$

Una vez teniendo la ecuación, se empiezan a extraer las secuencias de los cuadros con la siguiente secuencia $Y = (0, z, 2z, \dots, (n-1)z) \in \mathbb{N}^n$, y será llamada la secuencia base. Posteriormente se genera una secuencia de números aleatorios $R = (r_1, r_2, \dots, r_n) \in [1, z]^n$ y se utiliza para calcular la suma de la secuencia aleatoria y la secuencia base. Poniendo un ejemplo, supongamos se tiene un video de 60 cuadros de video por segundo, con duración de 2 segundos, esto nos da un total de 120 cuadros de video que corresponden al video. Sacando la secuencia base, tenemos:

$$z = \left\lfloor \frac{120}{14} \right\rfloor = 8 \quad (5.3)$$

Con todo esto definido, tenemos la siguiente ecuación para la generación de las listas de los cuadros a extraer:

$$V_{j,k} = \text{randint}(0, z) + \text{secuenciabase}[k], j \in [0, 5], k \in [0, 15] \quad (5.4)$$

Teniendo estas listas, cada una representa los cuadros de video a extraer:

- 15, 20, 24, 33, 43, 52, 62, 68, 73, 82, 91, 99, 109, 117
- 11, 16, 29, 33, 41, 49, 63, 64, 75, 81, 88, 101, 110, 119
- 13, 18, 25, 32, 44, 49, 56, 64, 78, 83, 95, 99, 111, 118
- 16, 30, 39, 44, 55, 62, 71, 72, 86, 92, 102, 110, 115
- 12, 20, 28, 34, 46, 50, 60, 70, 77, 85, 93, 99, 106, 113

Una vez con estas secuencias se extraen dichos cuadros de videos del video principal y se obtienen 5 ejemplares por cada video.

Tras aplicar esto a todos los videos, se obtienen 8400 videos en total.

Con esto concluye la manipulación de los videos.

Retomando OpenPose, es momento de alimentar los videos grabados al sistema Openpose. Al momento de ir procesando cada cuadro de video, OpenPose da de salida tres vectores con la forma (x, y, θ) , donde x hace referencia a la coordenada x de la parte, y a la coordenada y de la parte y θ a la confianza de que la parte pertenezca a la clase asignada. Estos tres vectores corresponden a las partes clave de la cara, el cuerpo y las manos. Por cada video leído, se creará un archivo que contiene tres columnas: las coordenadas x normalizadas, las coordenadas y normalizadas y las etiquetas de la clase a la que pertenece la parte del cuerpo, donde por el orden de los datos se identifica el punto clave al que pertenece. Una vez teniendo todos los videos de un señante se dividen en dos conjuntos, conjunto de entrenamiento y conjunto de prueba; se realiza este procedimiento para todos los señantes, juntando todos los conjuntos en dos archivos, el conjunto total de entrenamiento y el conjunto total de prueba. Estos conjuntos finales serán los que se introducirán a la red, ajustándose dependiendo de las características de dichos conjuntos, en particular haciendo que todas las secuencias del conjunto tengan la misma longitud. Para visualizar el funcionamiento de Openpose, se muestran dos figuras, cada uno representando a una seña, éstas están en las figuras 5.12 y 5.13.

Posteriormente se introducen los cambios a los datos y a la red que los procesará.

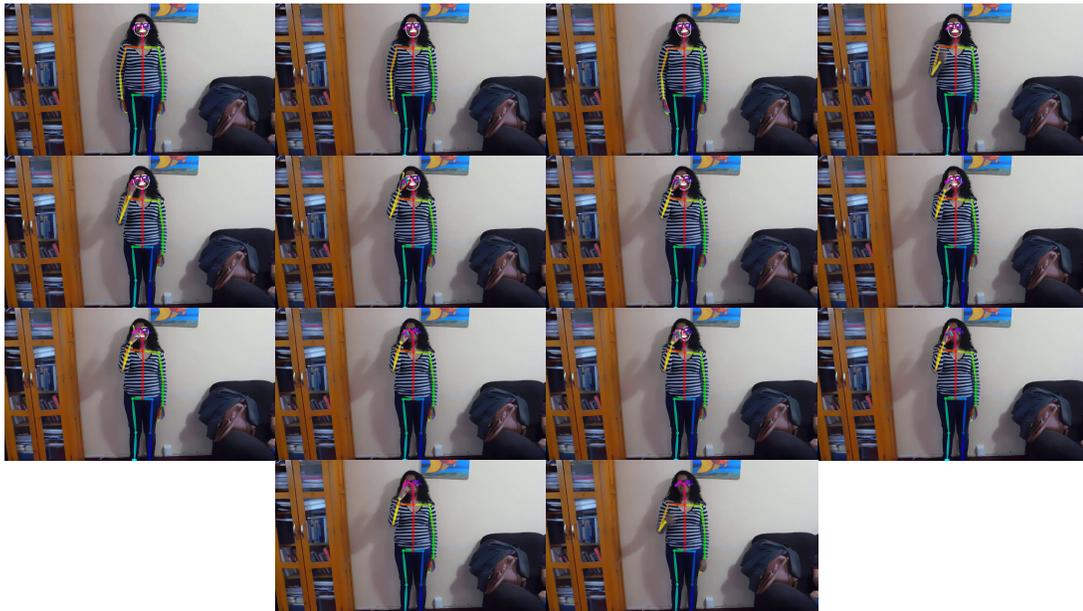


Figura 5.12: Movimientos hechos para representar a la seña Agua, mostrando lo que observa OpenPose

5.3.2. Preparación de los datos

La preparación de los datos consiste en dos principales cambios:

1. Igualar el tamaño de las secuencias del conjunto de entrenamiento para que tengan el

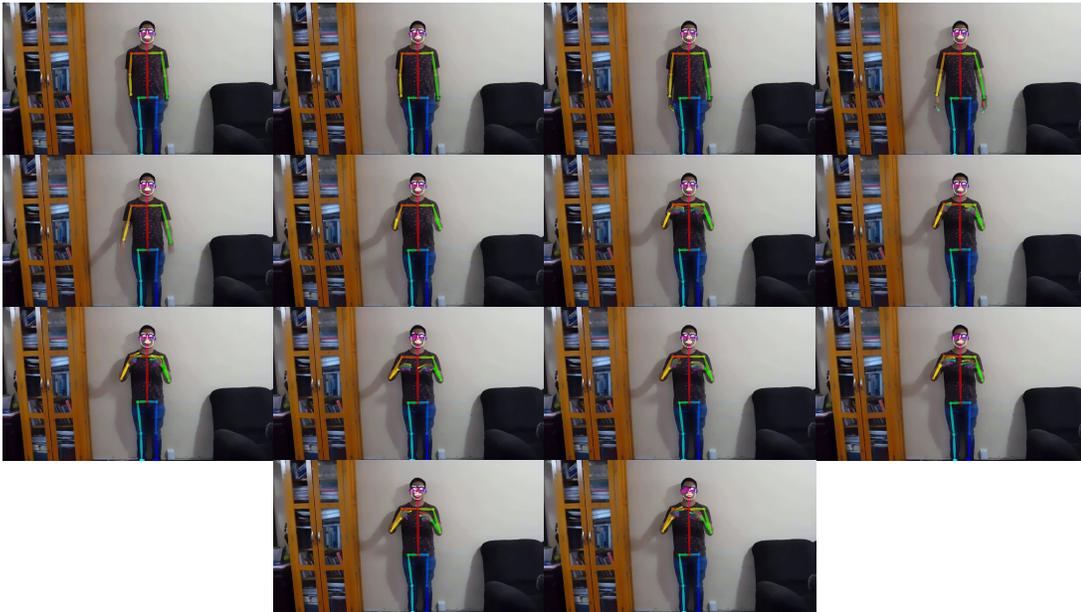


Figura 5.13: Movimientos hechos para representar a la seña Persona, mostrando lo que observa OpenPose

mismo tamaño, agregando secuencias de 0s en caso de requerirlo, repitiendo el proceso en el conjunto de prueba.

2. Se cambia el nombre de la etiqueta de la secuencia de entrenamiento por un número, de acuerdo a la tabla 5.2, posteriormente se aplica la función *to_categorical* de keras al conjunto de etiquetas para convertir cada elemento en un vector de 0s, salvo en la posición obtenida de la tabla, obteniendo los resultados en la tabla 5.3. Este proceso se repite con el conjunto de prueba.
3. Finalmente, el conjunto de pruebas se divide en dos conjuntos del mismo tamaño, este nuevo conjunto se llamará el conjunto de validación y servirá para validar los resultados del entrenamiento.

Estos conjuntos se muestran en la tabla 5.4.

5.3.3. Red LSTM

La red que se usará contendrá las redes previamente definidas en el capítulo 2, siendo creada por medio de Keras, ésta tiene las siguientes capas:

La red es una red secuencial, es decir, la red tiene topología lineal. La red inicia con dos capas LSTM con 10 unidades, donde la primera capa recibe el conjunto de entrenamiento, previamente amoldado a la capa, y de salida regresa el mismo conjunto para ser procesado en la segunda capa LSTM, esta segunda capa LSTM igual recibe el conjunto y lo vuelve a regresar.

Nombre de la seña	Número Asignado
Agua	0
Alto	1
Apagar	2
Ayudar	3
Campo	4
Creecer	5
Cuerpo	6
Despedir	7
Dinero	8
Escribir	9
Foto	10
Fruto	11
Hacer	12
LavarCara	13
Madera	14
Miedo	15
Morder	16
Nariz	17
Padrino	18
Película	19
Persona	20
Primero	21
Quedar	22
Saber	23
Sangre	24
Señorita	25
Suerte	26
Toro	27

Tabla 5.2: Etiquetas asignadas a las señas

Posteriormente viene una capa convolucional 1D (convolución temporal) la cual ayuda a aprender secuencias, teniendo un filtro (tamaño de la salida de la capa) de tamaño 15 y un kernel (ventana convolucional) de tamaño 15. Finalmente se aplica MaxPool a los datos, se aplanan y se alimentan a una capa completamente conectada con función softmax para dar la probabilidad de que la secuencia pertenezca a una seña en particular y dar la clase con mayor probabilidad como la clasificada.

La creación de la red, en python es la siguiente:

```
import keras as keras
from keras.layers import LSTM, Conv1D, MaxPool1D, Flatten, CuDNNLSTM, Flatten
from keras.layers.core import Dense, Dropout, Activation
```

Nombre de la seña	Resultado
Agua	[1, 0, 0, 0, ..., 0, 0, 0, 0]
Alto	[0, 1, 0, 0, ..., 0, 0, 0, 0]
...	...
...	...
...	...
...	...
Suerte	[0, 0, 0, 0, ..., 0, 0, 1, 0]
Toro	[0, 0, 0, 0, ..., 0, 0, 0, 1]

Tabla 5.3: Conjunto de etiquetas al aplicarle la función *to_categorical*. Cada elemento se vuelve un arreglo con las posibles etiquetas, habiendo un 1 en el lugar correspondiente a la clase

Nombre del Conjunto	Porcentaje de información total
Conjunto de entrenamiento	80 %
Conjunto de prueba	10 %
Conjunto de validación	10 %

Tabla 5.4: Tabla con los distintos conjuntos obtenidos

```

from keras.models import Sequential
from keras.optimizers import rmsprop
import numpy as np
_shape = np.array(X_train.shape[1], X_train.shape[2])
model = Sequential()
model.add(CuDNNLSTM(10, input_shape=_shape, return_sequences=True))
model.add(CuDNNLSTM(10, input_shape=_shape, return_sequences=True))
model.add(Conv1D(filters=15, kernel_size=15 ))
model.add(MaxPool1D())
model.add(Flatten())
model.add(Dense(28, activation='softmax'))

```

5.3.4. Entrenamiento de la red

La red se entrenará por 150 épocas, con un tamaño de lote de 256 y un conjunto de validación equivalente al 50% del conjunto de prueba. Al momento de entrenar se utilizará RMSprop (*Root Mean Square Propagation*) como optimizador de la red y para la función de pérdida se usa entropía cruzada categórica. Durante el entrenamiento se obtiene la siguiente información del script:

Model: sequential_2

```

-----
Layer (type)                Output Shape          Param #
=====

```

```

cu_dnnlstm_3 (CuDNNLSTM) (None, 2604, 10) 560
-----
cu_dnnlstm_4 (CuDNNLSTM) (None, 2604, 10) 880
-----
conv1d_2 (Conv1D) (None, 2590, 15) 2265
-----
max_pooling1d_2 (MaxPooling1 (None, 1295, 15) 0
-----
flatten_2 (Flatten) (None, 19425) 0
-----
dense_2 (Dense) (None, 28) 543928
=====
Total params: 547,633
Trainable params: 547,633
Non-trainable params: 0
-----
None

```

Train on 9040 samples, validate on 1130 samples

Epoch 1/150

9040/9040 [=====] - 23s 3ms/step - loss: 2.7402 - acc: 0.2423
- val_loss: 2.4243 - val_acc: 0.2982

Epoch 2/150

9040/9040 [=====] - 9s 948us/step - loss: 1.7709 - acc:
0.4531 - val_loss: 1.8738 - val_acc: 0.3956

Epoch 3/150

9040/9040 [=====] - 8s 939us/step - loss: 1.3423 - acc:
0.5713 - val_loss: 1.4591 - val_acc: 0.5248

Epoch 4/150

9040/9040 [=====] - 9s 948us/step - loss: 1.0501 - acc:
0.6538 - val_loss: 1.1134 - val_acc: 0.6283

Epoch 5/150

9040/9040 [=====] - 9s 950us/step - loss: 0.8919 - acc:
0.7048 - val_loss: 0.9023 - val_acc: 0.7044

Epoch 6/150

9040/9040 [=====] - 9s 944us/step - loss: 0.7559 - acc:
0.7540 - val_loss: 0.9471 - val_acc: 0.7080

Epoch 7/150

9040/9040 [=====] - 9s 945us/step - loss: 0.6805 - acc:
0.7778 - val_loss: 0.8583 - val_acc: 0.7115

Epoch 8/150

9040/9040 [=====] - 9s 945us/step - loss: 0.5918 - acc:
0.8127 - val_loss: 1.1035 - val_acc: 0.7000

Epoch 9/150

9040/9040 [=====] - 9s 941us/step - loss: 0.5427 - acc:
0.8272 - val_loss: 0.7467 - val_acc: 0.7681

Epoch 10/150

9040/9040 [=====] - 8s 940us/step - loss: 0.4939 - acc:
0.8376 - val_loss: 0.7796 - val_acc: 0.7611

....

```

Epoch 140/150
9040/9040 [=====] - 9s 947us/step - loss: 0.0095 - acc:
0.9992 - val_loss: 0.3382 - val_acc: 0.9637
Epoch 141/150
9040/9040 [=====] - 9s 943us/step - loss: 0.0231 - acc:
0.9960 - val_loss: 0.3532 - val_acc: 0.9593
Epoch 142/150
9040/9040 [=====] - 8s 940us/step - loss: 0.0121 - acc:
0.9981 - val_loss: 0.3420 - val_acc: 0.9628
Epoch 143/150
9040/9040 [=====] - 9s 948us/step - loss: 0.0132 - acc:
0.9979 - val_loss: 0.3398 - val_acc: 0.9628
Epoch 144/150
9040/9040 [=====] - 9s 945us/step - loss: 0.0188 - acc:
0.9972 - val_loss: 0.5569 - val_acc: 0.9301
Epoch 145/150
9040/9040 [=====] - 9s 944us/step - loss: 0.0130 - acc:
0.9983 - val_loss: 0.3670 - val_acc: 0.9566
Epoch 146/150
9040/9040 [=====] - 9s 947us/step - loss: 0.0099 - acc:
0.9991 - val_loss: 0.6917 - val_acc: 0.9195
Epoch 147/150
9040/9040 [=====] - 9s 949us/step - loss: 0.0183 - acc:
0.9973 - val_loss: 0.3408 - val_acc: 0.9637
Epoch 148/150
9040/9040 [=====] - 9s 950us/step - loss: 0.0212 - acc:
0.9962 - val_loss: 0.3585 - val_acc: 0.9611
Epoch 149/150
9040/9040 [=====] - 9s 943us/step - loss: 0.0091 - acc:
0.9994 - val_loss: 0.3431 - val_acc: 0.9619
Epoch 150/150
9040/9040 [=====] - 9s 946us/step - loss: 0.0154 - acc:
0.9980 - val_loss: 0.3291 - val_acc: 0.9628
Best validation acc of epoch: 0.9672566356912123

```

De igual forma se gráfica el proceso de entrenamiento, se puede ver en la figura [5.14](#).

5.3.5. Resultados

Teniendo al modelo entrenado es momento de ver qué tan preciso es en el conjunto de prueba. Al pasar el conjunto de prueba a la red entrenada, previamente moldados los datos, se obtienen los resultados mostrados en la tabla [5.5](#) junto a su matriz de confusión en [5.15](#).

5.3.6. Interpretación de los resultados

Al obtener los resultados se aprecia que el clasificador tiene una precisión buena, 97.34%, mostrando que es posible lograr la detección de señas por medio de redes LSTM, tomando en cuenta que la información alimentada posea suficiente información para diferenciar los diferentes movimientos de las señas, tal como la información proporcionada por Openpose. También

Tabla 5.5: Clasificación de las clases en el segundo experimento.

Seña	Número de Videos	Número de Clasificaciones Correctas	Número de Clasificaciones Incorrectas	Porcentaje de precisión
Agua	32	32	0	100 %
Alto	50	49	1	98 %
Apagar	52	50	2	96.15 %
Ayudar	45	44	1	97.77 %
Campo	32	32	0	100 %
Crecer	42	40	2	95.23 %
Cuerpo	36	35	1	97.22 %
Despedir	35	33	2	94.28 %
Dinero	47	47	0	100 %
Escribir	39	39	0	100 %
Foto	41	41	0	100 %
Fruto	37	37	0	100 %
Hacer	33	33	0	100 %
LavarCara	36	36	0	100 %
Madera	44	43	1	97.72 %
Miedo	40	40	0	100 %
Morder	37	36	1	97.29 %
Nariz	40	40	0	100 %
Padrino	42	42	0	100 %
Película	37	35	2	94.59 %
Persona	41	38	3	92.68 %
Primero	46	44	2	95.65 %
Quedar	48	45	3	93.75 %
Saber	44	43	1	97.72 %
Sangre	35	35	0	100 %
Señorita	42	42	0	100 %
Suerte	30	28	2	93.33 %
Toro	48	48	0	100 %
Total	1131	1107	24	97.34 %

se puede observar que en las señas con un porcentaje inferior al 100 % la seña posee características compartidas con varias señas, por ejemplo, la seña “Alto” consiste en el movimiento de una sola mano hacia arriba, este movimiento también se aprecia en las señas “Crecer”, “Hacer”. Y el caso opuesto, donde la seña presenta características más únicas, la seña obtiene un porcentaje 100 % de precisión, esto para las señas “LavarCara”, “Padrino”, “Toro”, “Señorita”.

En base a todo esto se tendría que hacer más robusta la red para ser más “observadora”

y lograr distinguir las diferencias más sutiles de las distintas señas, dado que en un conjunto con un número mucho mayor de señas la red, en su estado actual, tendría detecciones erróneas entre las señas, y si en un conjunto de 5 señas parecidas hubo una con 93% de detección, si hubieran 50 señas parecidas, o hasta más, la detección podría llegar a ser menor al 50%, teniendo más falsos positivos que positivos.

5.3.7. Experimento Extra

Se hizo un experimento extra, tomando los videos grabados en el primer experimento y tomando el clasificador del segundo experimento.

El proceso es análogo al del segundo experimento, únicamente cambiando los videos de entrada. Se muestran los datos del entrenamiento en las figuras 5.16, y los datos de las pruebas en 5.6 con su matriz de confusión en 5.17.

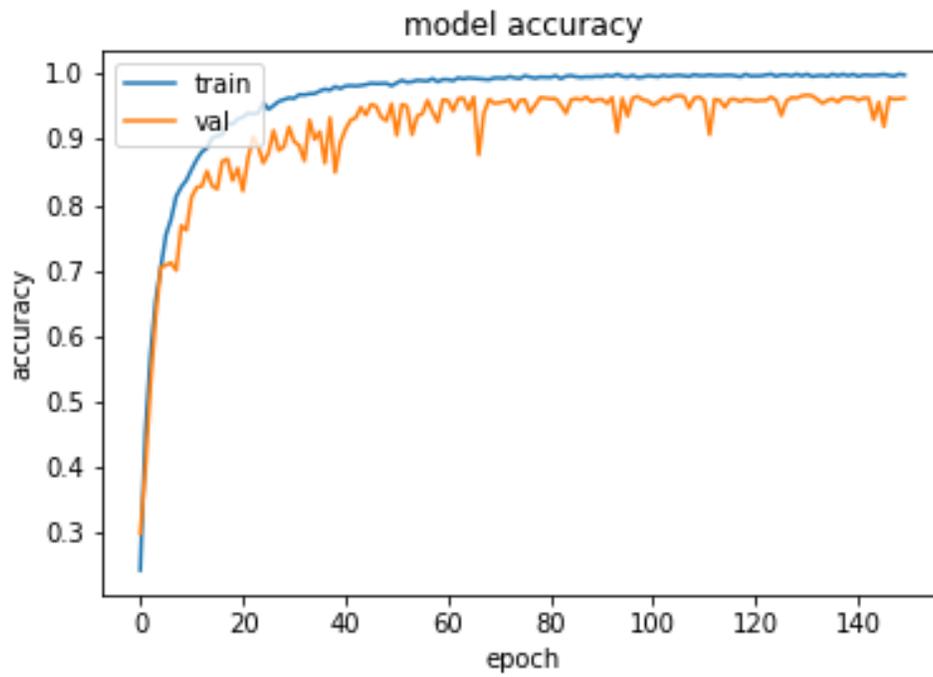
5.3.8. Conclusión del experimento Extra

Al obtener los resultados se aprecia que el clasificador vuelve a tener una precisión buena, 99.73%, mostrando que el clasificador no depende de un tipo de video en particular para lograr la detección y, de forma más importante, muestra que la clasificación anterior no es capaz de lograr diferenciar entre las características del movimiento de las señas probadas.

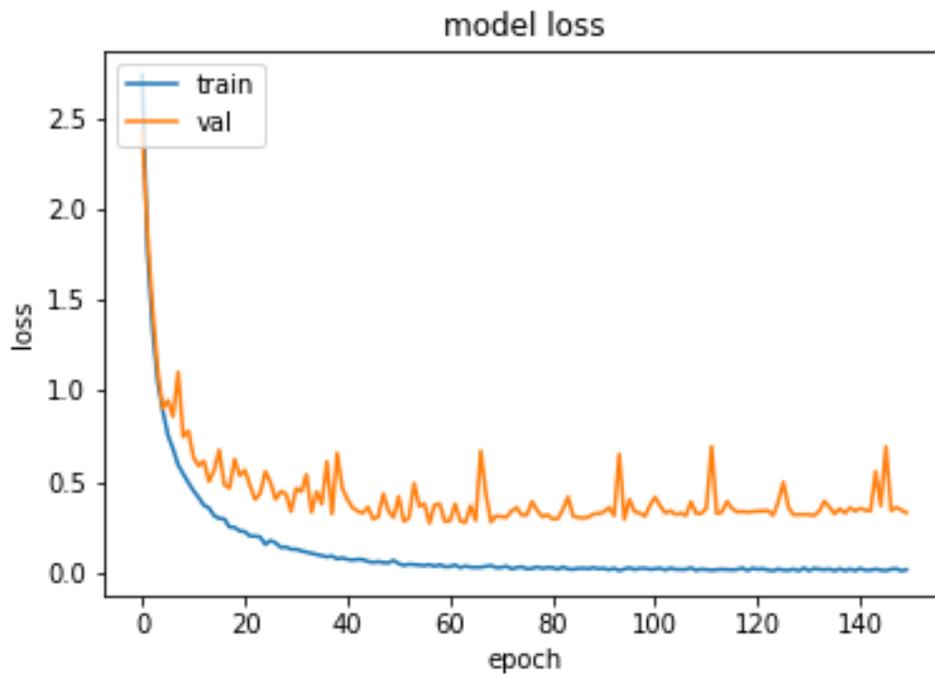
En este caso los videos son más sencillos y no se muestra todo el cuerpo completo, como en los otros videos, si no que va variando la visibilidad del cuerpo, desde únicamente de la cintura para arriba hasta también mostrar los pies. Por lo que, al haber menos información proporcionada por Openpose, el clasificador agarra las características necesarias para lograr clasificar, logrando que en únicamente 3 señas haya habido un único error de clasificación. Pero en el caso de que las señas fueran más parecidas entre sí o tuvieran un movimiento más complejo, el detector tendría más problemas para la identificación al no poseer tanta información.

Tabla 5.6: Clasificación de las clases en el tercer experimento.

Seña	Número de Videos	Número de Clasificaciones Correctas	Número de Clasificaciones Incorrectas	Porcentaje de precisión
Agua	42	42	0	100 %
Alto	41	41	0	100 %
Apagar	33	33	0	100 %
Ayudar	42	42	0	100 %
Campo	32	32	0	100 %
Crecer	45	45	0	100 %
Cuerpo	44	44	0	100 %
Despedir	44	44	0	100 %
Dinero	36	36	0	100 %
Escribir	36	36	0	100 %
Foto	43	42	1	97.67 %
Fruto	45	44	1	97.77 %
Hacer	37	37	0	100 %
LavarCara	32	32	0	100 %
Madera	40	40	0	100 %
Miedo	40	40	0	100 %
Morder	46	46	0	100 %
Nariz	42	42	0	100 %
Padrino	38	38	0	100 %
Película	38	38	0	100 %
Persona	41	41	0	100 %
Primero	47	47	0	100 %
Quedar	36	36	0	100 %
Saber	43	43	0	100 %
Sangre	45	45	0	100 %
Señorita	43	42	1	97.67 %
Suerte	37	37	0	100 %
Toro	39	39	0	100 %
Total	1120	1117	3	99.73 %



(a) Precisión del entrenamiento sobre el conjunto de entrenamiento y validación.



(b) Pérdida del entrenamiento sobre el conjunto de entrenamiento y validación.

Figura 5.14: Pérdida y precisión de la red durante su entrenamiento.

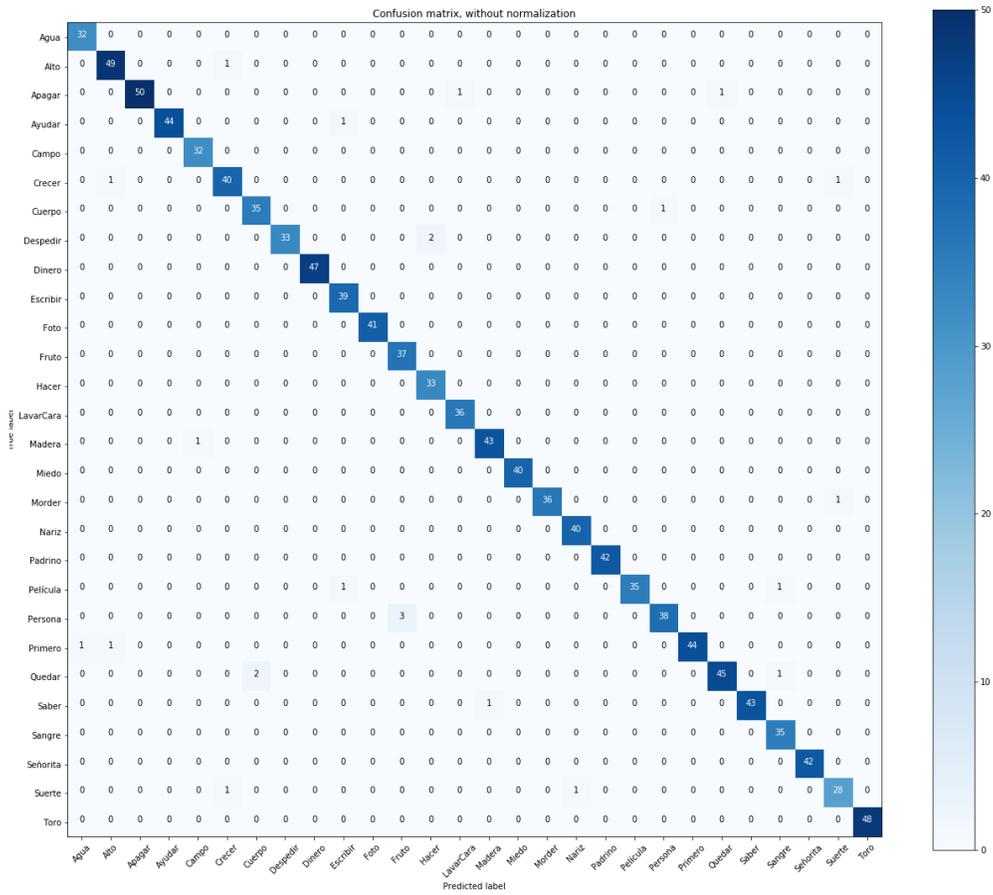
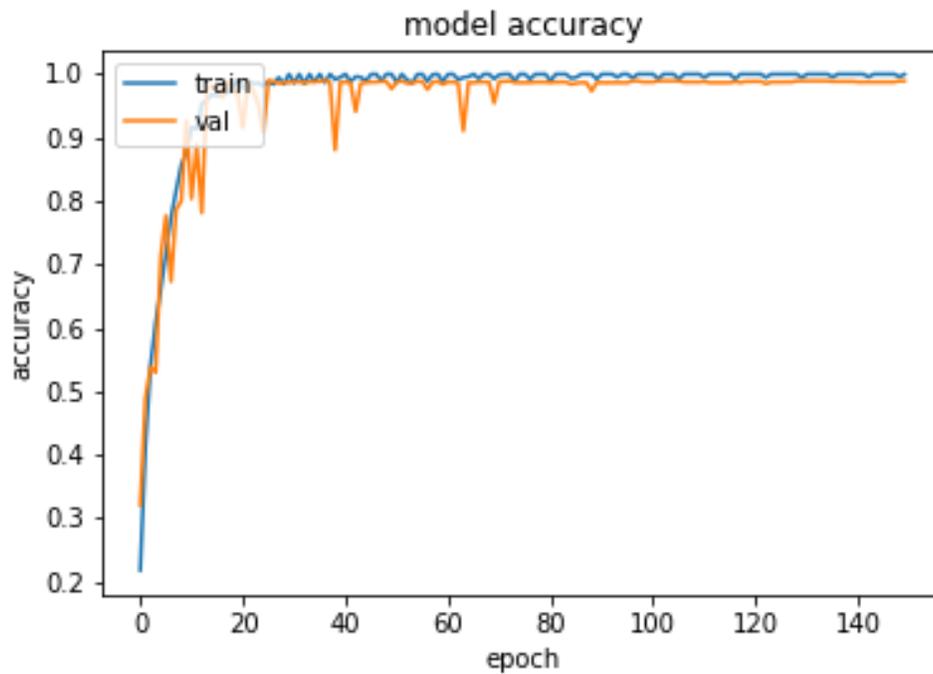
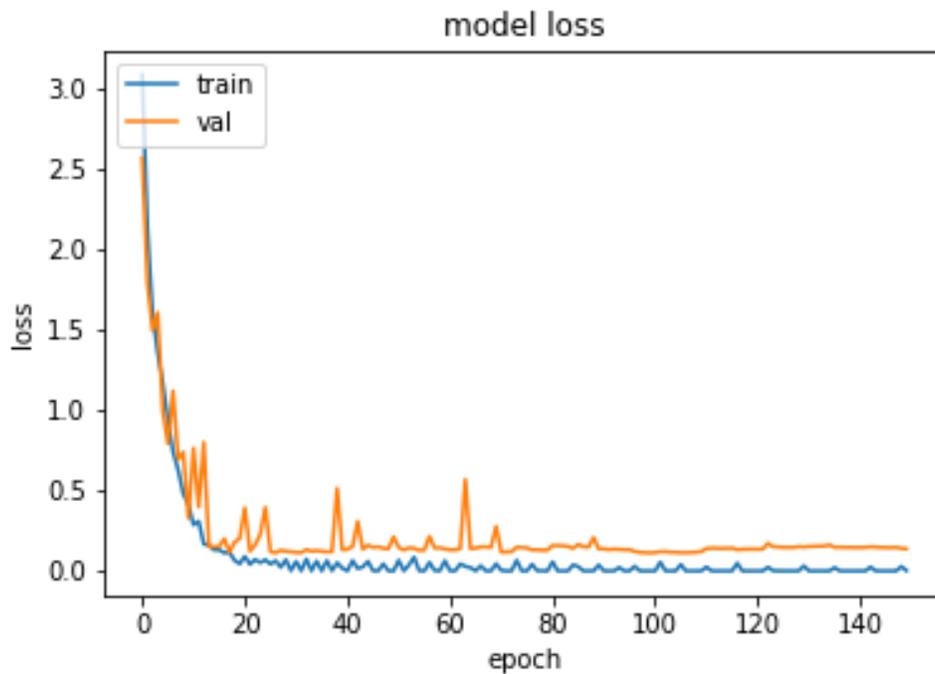


Figura 5.15: Matriz de confusión



(a) Precisión del entrenamiento sobre el conjunto de entrenamiento y validación.



(b) Pérdida del entrenamiento sobre el conjunto de entrenamiento y validación.

Figura 5.16: Pérdida y precisión de la red durante su entrenamiento.

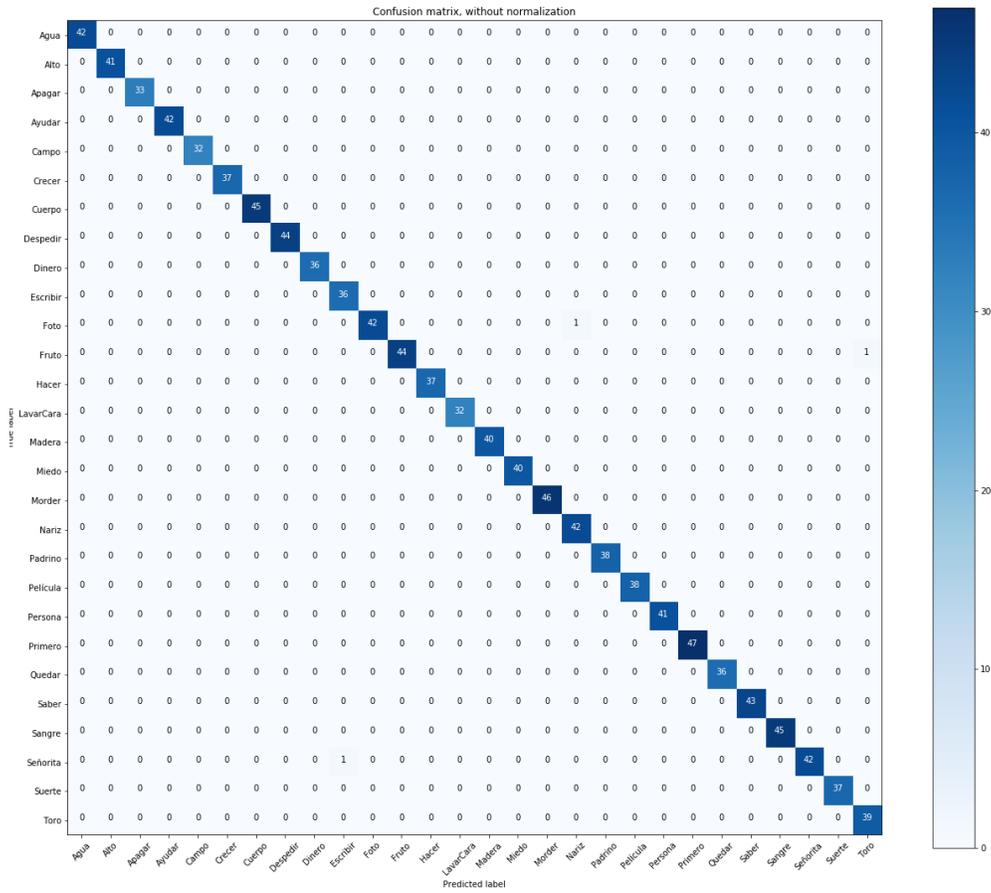


Figura 5.17: Matriz de confusión

Capítulo 6

Conclusión

En este trabajo se abordó el problema de la detección de algunas palabras dentro de la lengua de señas mexicana sin el uso de algún tipo de dispositivo extra, tales como un guante. Los experimentos muestran que es posible la detección de estas señas, pero que no es algo tan sencillo como únicamente seguir la trayectoria, sino que se deben de poder observar características más específicas de las manos y el cuerpo para su detección. Si bien para este trabajo únicamente bastaría el obtener las características de los brazos y manos para la detección, se utilizan también las características de la cara para mostrar que éstos también son útiles para la detección.

No obstante, el sólo reconocer unas pocas señas es ineficiente como una herramienta para facilitar la comunicación entre la lengua de señas y el lenguaje hablado, por lo que hay más trabajo por hacer, detallo lo que considero importante para lograr una comunicación posible entre los dos tipos de lenguas. El sistema idóneo se compone de dos módulos distintos, el módulo para detección de señas y el módulo para formar oraciones dadas las señas detectadas, ya sea para traducir la estructura de la lengua de señas a una estructura del lenguaje hablado, o para transliterar las palabras interpretadas a su representación literal, donde la primera interpretación depende tanto de rasgos manuales como no manuales, en [6.1](#) se observan ejemplos de éstas oraciones.

Para el primer módulo es necesario el tener los movimientos atómicos, de los cuales se componen todas las señas, movimiento de dedo hacia alguna dirección, movimiento de brazo hacia alguna dirección, mano abierta / cerrada, entre otros, como se puede observar en la figura [5.11](#). Una vez se tiene este movimiento atómico, junto a una dirección o posición, se empieza a registrar la construcción de todas las señas dentro del LSM a partir de los movimientos atómicos, por ejemplo, para la seña agua los movimientos atómicos serían los siguientes:

- Movimiento de mano dominante hacia arriba hasta llegar a la altura de la barbilla.
- Doblar el dedo índice de la mano dominante (3x).
- Regresar la mano dominante a la posición neutral.

Teniendo la construcción de las señas bastaría identificar los movimientos atómicos realizados

Rasgos Manuales - ¿Qué seña se articula?	Rasgos No Manuales	Traducción a lenguaje hablado
NO DESCANSAR	Movimiento de la cabeza de derecha a izquierda para expresar malestar o disgusto	¿No! ¡Déjenme en paz!
UNO CUAL PRO QUEDAR*	Cabeceo hacia atrás * indica sobre qué vocablo se acentúa el levantamiento de las cejas y el descenso de la cabeza hacia el frente	¿Cuál escogiste? ¿Yo me quedo?

Tabla 6.1: Ejemplos tomados de [47]

durante la interpretación de cada seña y alimentarlos a una estructura de aprendizaje, la cual daría de salida a la seña correspondiente.

Otra forma de lograr la detección de las señas es enseñar la seña como tal a un sistema de detección (como en este trabajo), que si bien es más “sencillo” de realizar, se necesita una cantidad considerable de información por cada seña, y teniendo en cuenta que hay una gran cantidad de señas sería un conjunto de información muy grande, a comparación del otro método que únicamente requiere un conjunto de los movimientos atómicos. Para la traducción de las señas identificadas, en el caso donde únicamente es transliterar no es necesario realizar otra cosa, pero en caso de querer traducir de la lengua señada a lenguaje hablado se requiere un sistema capaz de saber la estructura del lenguaje de señas al lenguaje hablado, pudiéndose realizar con una RNN o un Transformador que servirá de “traductor” entre las dos lenguas, tomando en cuenta los rasgos manuales y rasgos no manuales.

Una vez se tiene un sistema así en funcionamiento es posible lograr una comunicación entre una persona señante y una persona no señante. Para la comunicación entre una persona no señante y una persona señante puede recurrirse a un sistema de reconocimiento automático de voz para mostrar visualmente las palabras habladas.

Bibliografía

- [1] Gary J. Grimes. Digital data entry glove interface device, 1981.
- [2] MICHAEL ERARD. Why sign-language gloves don't help deaf people, nov 2017. URL <https://www.theatlantic.com/technology/archive/2017/11/why-sign-language-gloves-dont-help-deaf-people/545441/>.
- [3] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features, 2001.
- [4] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.
- [5] Josep Maria Carmona Leyva and Joan Climent. A performance evaluation of hmm and dtw for gesture recognition. pages 236–243, 09 2012. doi: 10.1007/978-3-642-33275-3_29.
- [6] Mahmoud Elmezain and Ayoub Al-Hamadi. Gesture recognition for alphabets from hand motion trajectory using hidden markov models. pages 1192 – 1197, 01 2008. doi: 10.1109/ISSPIT.2007.4458209.
- [7] Sang-Ki Ko, Chang Jo Kim, Hyedong Jung, and Choong Sang Cho. Neural sign language translation based on human keypoint estimation. *CoRR*, abs/1811.11436, 2018. URL <http://arxiv.org/abs/1811.11436>.
- [8] Diario Oficial de la Federación. Ley general para la inclusión de las personas con discapacidad. http://www.diputados.gob.mx/LeyesBiblio/pdf/LGIPD_120718.pdf. Publicada el 12-07-2018.
- [9] Real Academia Española. Modalidad, 01 2020. URL <https://dle.rae.es/modalidad>.
- [10] Real Academia Española. Gramática, 01 2020. URL <https://dle.rae.es/?id=JQukZIX>.
- [11] Gobierno de la Ciudad de México. *Diccionario de Lengua de Señas Mexicana LSM Ciudad de México*. Ciudad de México, México, 2017.
- [12] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

-
- [13] Jeff Heaton. *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*. CreateSpace Independent Publishing Platform, 2015. ISBN 1505714346. URL <https://www.amazon.com/Artificial-Intelligence-Humans-Learning-Networks/dp/1505714346>.
- [14] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018. URL <http://neuralnetworksanddeeplearning.com/>.
- [15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [16] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL <http://arxiv.org/abs/1311.2524>.
- [17] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL <http://arxiv.org/abs/1504.08083>.
- [18] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL <http://arxiv.org/abs/1506.01497>.
- [19] Varun Ramakrishna, Daniel Munoz, Martial Hebert, J. Andrew (Drew) Bagnell, and Yaser Ajmal Sheikh. Pose machines: Articulated pose estimation via inference machines. In *Proceedings of (ECCV) European Conference on Computer Vision*, pages 33 – 47, September 2014.
- [20] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines, 2016.
- [21] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields, 2016.
- [22] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Real-time multi-person 2d pose estimation using part affinity fields, 2018.
- [23] Christopher Olah. Understanding lstm networks. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 2015.
- [24] Pathmind. A beginner’s guide to lstms and recurrent neural networks. <https://pathmind.com/wiki/lstm>.
- [25] missinglink.ai. Deep learning long short-term memory (lstm) networks: What you should remember. <https://missinglink.ai/guides/neural-network-concepts/deep-learning-long-short-term-memory-lstm-networks-remember/>.
- [26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL <http://arxiv.org/abs/1502.03167>.

- [27] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, March 1991. ISSN 0893-6080. doi: 10.1016/0893-6080(91)90009-T. URL [http://dx.doi.org/10.1016/0893-6080\(91\)90009-T](http://dx.doi.org/10.1016/0893-6080(91)90009-T).
- [28] George Cybenko. Approximation by superpositions of a sigmoidal function. *math cont sig syst (mcss)* 2:303-314. *Mathematics of Control, Signals, and Systems*, 2:303–314, 12 1989. doi: 10.1007/BF02551274.
- [29] Convolutional neural networks (cnn): Step 2 - max pooling. URL <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-max-pooling>. Accessed: 2019-02-21.
- [30] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013. doi: 10.1007/s11263-013-0620-5. URL <http://www.huppelen.nl/publications/selectiveSearchDraft.pdf>.
- [31] Convolutional neural networks (cnn): Step 2 - max pooling. URL <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.
- [32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL <http://arxiv.org/abs/1512.03385>.
- [33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL <http://arxiv.org/abs/1409.1556>.
- [34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL <http://arxiv.org/abs/1409.4842>.
- [35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL <http://arxiv.org/abs/1502.01852>.
- [36] Meinard Müller. *Information Retrieval for Music and Motion*. Springer, 2007. ISBN 3540740473. URL <https://www.amazon.com/Information-Retrieval-Motion-Meinard-M%C3%BCller/dp/3540740473?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3540740473>.
- [37] Stan Salvador and Philip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *KDD workshop on mining temporal and sequential data*. Citeseer, 2004.
- [38] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009. doi: 10.4249/scholarpedia.1883. revision #137311.

-
- [39] Wikipedia contributors. Dynamic time warping — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Dynamic_time_warping&oldid=885921603. [Online; accessed 3-March-2019].
- [40] Wikipedia contributors. Voronoi diagram — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Voronoi_diagram&oldid=886651770. [Online; accessed 10-March-2019].
- [41] 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.
- [42] Francisco Gomez-Donoso, Sergio Orts-Escolano, and Miguel Cazorla. Large-scale multi-view 3d hand pose dataset. *CoRR*, abs/1707.03742, 2017. URL <http://arxiv.org/abs/1707.03742>.
- [43] Alexander Jung. Image augmentation for machine learning experiments. <https://github.com/aleju/imgaug>, 2017.
- [44] Tensorflow. Tensorflow object detection api. https://github.com/tensorflow/models/tree/master/research/object_detection, 2017.
- [45] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [46] Secretaría de Educación Pública. Diccionario español - lengua de señas mexicana, 2006.
- [47] Ling. Boris Fridman Mintz. Gestos y oraciones de la lengua de señas de México. 1996.