UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Posgrado en Ciencia e Ingeniería de la Computación

COMBINACIÓN DE MODELOS OCULTOS DE MARKOV CON CNNs PARA LA CATEGORIZACIÓN SEMÁNTICA DE OBJETOS.

T E S I S

que para optar por el grado de

Maestro en Ciencia e Ingeniería de la Computación

PRESENTA:
Oscar Durón Luna

Tutor:
Dr. Jesus Savage Carmona
Facultad de Ingeniería, UNAM

Ciudad Universitaria, CD. MX., febrero 2023

Agradecimientos

Este trabajo fue financiado por CONACYT, a través de la beca asignada por el Posgrado en Ciencia e Ingeniería de la Computación de la UNAM. También agradezco a DGAPA-UNAM por el financiamiento, a través del proyecto PAPIIT AG101721 “Modelos computacionales para el comportamiento adaptable de robots de servicio y de humanos”.

Agradezco a toda mi familia, por la paciencia y apoyo que me dieron durante este proceso. En especial a mis padres Cecilia y Raymundo, que siempre estuvieron para darme apoyo emocional, consejos y paciencia. Gracias por todo el sacrificio y esfuerzo durante tantos años que me permitieron llegar hasta este punto.

A a la colaboración entre la UNAM y la Universidad de Tamagawa de Japón, y en especifico al Dr. Hiroyuki Okada, al Dr. Luis Ángel Contreras Toledo y al Dr. Jesus Savage Carmona por darme la oportunidad de tener la increíble experiencia de realizar una estancia en el extranjero en la que pudiera seguir desarrollando parte de mi trabajo de investigación. De igual manera agradezco a PAEP de la UNAM por haberme dado un apoyo económico para realizar esta estancia de investigación, sin la cual no hubiera sido mayormente posible.

Al Dr. Jesus Savage por darme la oportunidad de realizar esta tesis y de permitirme entrar al mundo de la robótica.

A mis amigos y compañeros de la licenciatura, la maestría y del laboratorio de bio-robótica de la UNAM, que han sido fundamentales en muchos momentos específicos durante la carrera y la maestría.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra universidad o publicación. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Oscar Durón Luna. Ciudad de México, 12 de enero de 2023.

Resumen

Las Redes Neuronales Profundas tienen una amplia implementación en la actualidad para poder realizar tareas de reconocimiento, segmentación y clasificación de una gran cantidad de objetivos. La aplicación de estas técnicas también es utilizada en el área de la robótica de servicio, tanto para la investigación así como para la realización de tareas específicas en ambientes competitivos. Una de estas tareas implica la interacción que tendrá el robot de servicio con el humano para realizar lo que le indiquen o detecte, por ejemplo, al asistir a la persona que esté realizando una acción particular y que el robot pueda reconocerlo y realice una tarea correspondiente. Entre las diversas maneras en que un robot puede procesar una acción humana e inferirla es mediante las técnicas antes mencionadas, que son parte de la *Inteligencia Artificial*. Por lo anterior, en este trabajo se propone un sistema reconocedor de acciones mediante el uso combinado de Redes Neuronales Convolucionales con otra técnica llamada *modelos ocultos de Markov*. Este reconocedor de acciones tendrá como objetivo poder integrarse a un sistema semántico de objetos el cuál solo utiliza Redes Neuronales como técnicas para reconocimiento de acciones y objetos, de esta manera se tendrán dos sistemas con los cuales se pretenderá comparar su desempeño y observar sus ventajas y desventajas de esta propuesta de trabajo con el anterior sistema utilizando una metodología similar. Se presenta la manera en como este sistema procesará la información que reciba para poder hacer una inferencia de acciones mediante la utilización de modelos ocultos de Markov, así como la metodología utilizada para poder entrenar estos modelos y los requerimientos previos necesarios para que éstos puedan leer la información entrante.

Se hace un análisis detallado del conjunto de datos utilizado para el entrenamiento de estos

modelos, ya que se observa que tuvo una amplia influencia en los resultados presentados con la metodología propuesta. Este conjunto de datos está conformado por una serie de videos en las cuales se presentan distintas personas, de manera individual y en un ambiente controlado, realizando diferentes acciones de manera *natural* y fue creado y utilizado para el sistema antes mencionado que utiliza solamente Redes Neuronales. Por lo anterior se presenta un análisis para saber si resulta conveniente utilizar esta misma información con una técnica de reconocimiento distinta (modelos ocultos de Markov).

Abstract

Deep Neural Networks techniques have several implementations at the present time for recognition, segmentation and classifications of different tasks. The use of these techniques is also used in domestic robots research field, as well as specific tasks in competitive enviroments. One special task is the human-robot interaction which the robot will do something depending of what it detects or the humans command to it to do, for example to assist the human that is doing a particular action so the robot can recognize it and realize the corresponding task. Among the different ways for the robot to process and make an inference of the action recognized, is with the use of this techniques and algorithms which are part of the *Artificial Intelligence* area. Having said that, in this work an action recognition system is proposed using the combination of convolutional neural networks (CNN) and hidden Markov models (HMM) algorithms. This action recognition system can be integrated to an existing object semantic system, so the performance of this new system cam be compared to the original one, as well as watch their advantages and disadvantages. This work will show how the income data will be processed, so the proposed system can do an inference of an action using HMM algorithms. The metodology of this works for the training of the models is also shown, as well as the prerequisites so this Markov models can process the income data.

A detailed analysis is made of the dataset used in this work for the training and evaluation of the models, because it was observed that it has a big influence in the performance of the models

and the results shown using the metodology used. This dataset has a set of videos in which, in a controlled enviroment, different people are doing specific actions in a *natural* manner. Because of this, a different analysis is shown where if it is convenient the use of this dataset and the metodology using HMM recognition techniques.

Índice general

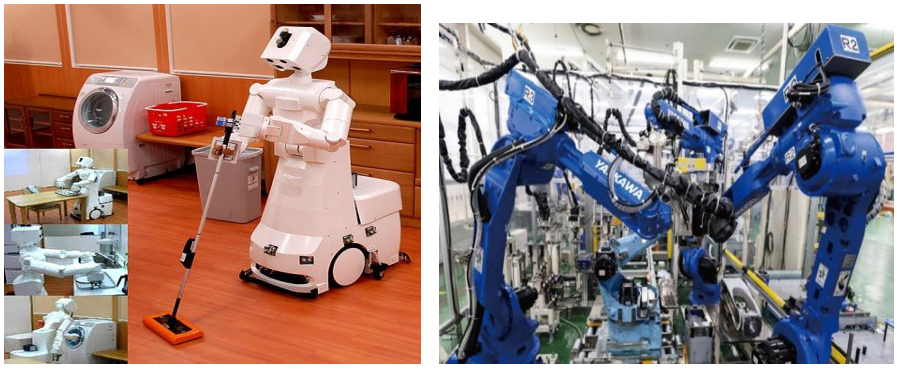
Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Con el avance acelerado de la tecnología en los últimos años, particularmente en el área de la robótica ha ayudado a explorar múltiples disciplinas que anteriormente no era posible (figura [1-1]). La incorporación de nuevos materiales, tecnologías de navegación, de sensores y recientemente con la integración de la Inteligencia Artificial (IA), la robótica logra abarcar diversas áreas de investigación o desarrollo y producción. Particularmente en la robótica de servicio, disciplina que tiene el objetivo de realizar tareas domésticas de manera parcial o totalmente autónoma para el beneficio de un humano, se beneficia haciendo uso de técnicas de visión computacional [29] y de IA para la detección y reconocimiento de objetos, lo que permite realizar más eficientemente estas tareas.



(a) robots de servicio, imagen de [18] (b) robots en la industria, imagen de [2]

Figura 1-1: Ejemplos de la robótica en distintas áreas

La visión computacional, parte de la investigación del área de visión artificial, tiene como objetivo extraer características de una imagen y procesarlas e interpretarlas en una computadora, análogamente a como un humano, mediante la vista y el cerebro identifican objetos de interés y sus posiciones en el medio ambiente, que llegan de la información de los ojos y así realizar una acción posterior.

1.1. Definición del Problema

Para un robot de servicio es importante que tenga un sistema suficientemente completo que le permita realizar tareas adecuadamente. Entre las labores a realizar está el reconocer o anticipar acciones humanas y poder actuar con base en la información procesada. Para poder detectar y reconocer una acción en particular el sistema debe ser lo suficientemente robusto para procesar múltiples secuencias de imágenes provenientes de las cámaras que integran al robot. Estos sistemas deben pasar previamente por un proceso de entrenamiento para que sepan manejar los datos correctamente, usando un amplio conjunto de datos relacionado a la tarea que deben ejecutar.

Figura 1-2: Ejemplo de una secuencia de movimientos humano. Imagen de [27]

Una manera en que se puede efectuar el reconocimiento es mediante una secuencia de movimientos de la persona en imágenes, las cuales se detectaría el movimiento de las extremidades de la persona y trataría de encontrar un patrón que describa cada acción. Estas secuencias de movimientos de extremidades o del "esqueleto" de la persona, obtenidos de un conjunto de

datos, permitirán entrenar un sistema que prediga la acción más probable de acuerdo a una secuencia de datos.

Paralelamente es necesario que se reconozcan los objetos que interactúan con la persona que está realizando una acción, por lo que también se necesita un sistema reconocedor de objetos. Una vez que el primer sistema prediga la acción más probable y el segundo sistema detecte y reconozca el objeto, se necesita un tercer sistema que unifique y relacione los anteriores resultados y haga inferencias del objeto y sus propiedades.

1.2. Justificación

Un reconocedor de acciones permite al robot de servicio adquirir más información de su entorno y poder actuar o responder de mejor manera cuando interactúe con un humano. Para la persona que desarrolla estos sistemas es conveniente tener distintas herramientas que hagan una misma tarea, ya que se pueden hacer comparaciones de rendimiento, estadísticas de ejecución, precisión, exactitud, entre otras métricas para poder evaluar la herramienta que mejor convenga de acuerdo a la tarea a realizar.

Este trabajo de investigación ofrecerá una alternativa de herramienta en el sistema reconocedor de acciones, de acuerdo con el proyecto [40] que se continúa trabajando y se basó éste. También ofrecerá datos estadísticos para comparar entre ambas técnicas empleadas.

1.3. Objetivo

El objetivo general del presente trabajo es:

Crear un sistema que infiera usos de objetos de acuerdo con información de acciones humanas provenientes de cámaras de un robot de servicio, implementando un Modelo Oculto de Markov para el reconocimiento de acciones.

Para poder cumplir con lo anterior se plantean los siguientes objetivos particulares:

- Aumentar información al dataset con respecto a nuevas acciones u objetos.
- Diseñar un modulo detector de acciones implementando un modelo oculto de Markov
- Incorporar el módulo al sistema categorizador semántico de acciones

1.4. Hipótesis

Se planteó la siguiente hipótesis al iniciar este proyecto de investigación:
"Al utilizar conjuntamente modelos ocultos de Markov con Redes Neuronales Profundas en un sistema reconocedor de acciones, éste tendrá un desempeño igual o mejor que al utilizar un sistema por separado."
Para poder comprobar esta hipótesis, se desarrollarán e implementarán sistemas computacionales usando las técnicas antes mencionadas, haciendo pruebas experimentales y comparando resultados.

1.5. Descripción de la tesis

En este primer capítulo 1, Introducción, da inicio al planteamiento y justificación de realizar este trabajo de investigación, así como los objetivos e hipótesis planteados.
En el capítulo 2, Marco Teórico, se da una descripción de la teoría en la que se sustenta este trabajo, así como el estado del arte relacionado, abordando temas como sistemas de reconocimiento de objetos y secuencia de movimientos mediante Redes Neuronales y Modelos Ocultos de Markov.
En el capítulo 3, Modelos Ocultos de Markov, se aborda más detalladamente la teoría y conceptos detrás de los Modelos de Markov, esenciales para entender parte de este proyecto.
En el capítulo 4, Redes Neuronales , de manera similar se describe con más profundidad la teoría de las Redes Neuronales Profundas, fundamental para la base que se sustenta este proyecto.
En el capítulo 5, Metodología, se presenta la forma en la cual se trabajó este proyecto de tesis, lo necesario en relación con el *software* y *hardware* usados e implementados en el presente trabajo.
En el capítulo 6, Implementación, se describe detalladamente el procedimiento para desarrollar

el sistema descrito para este trabajo.

En el capítulo 7, Pruebas y Resultados, se presentan y reportan los resultados obtenidos y las pruebas hechas durante la implementación del sistema.

Por último, en el capítulo 8, Conclusiones y Trabajo Futuro, se exponen las conclusiones a las que se llegó después de implementar este sistema así como el planteamiento de posibles trabajos futuros.

Capítulo 2

Marco Teórico

En este capítulo se presentan los fundamentos teóricos que sustentan el planteamiento de este proyecto de investigación. Da inicio introduciendo el reconocimiento de acciones humanas, el estado del arte de los Modelos Ocultos de Markov así como aquellos que involucran el reconocimiento de estas acciones. Por último, se aborda el estado del arte referente a Redes Neuronales Profundas en general y más detallado con el reconocimiento de acciones humanas.

2.1. Reconocimiento de Actividades Humanas

Una acción es realizada por una persona casi todo el tiempo durante su vida, tienen el fin de alcanzar un objetivo particular. Involucra el uso de *gestos*, los cuales son el movimiento básico de alguna extremidad del cuerpo, por ejemplo, mover la cabeza o flexionar el brazo, y generalmente las acciones hacen uso de varios gestos y tiene una secuencia cronológica. La acción es realizada por una persona pero puede involucrar a un objeto o llegar a haber una interacción con otra persona, como las acciones de abrazar (a otra persona) o martillar (con un objeto *martillo*).

De acuerdo con *Aggarwal y Ryoo* [1], algunas aplicaciones en las que se puede implementar el reconocimiento de acciones son: sistemas de vigilancia, sistemas que involucren la interacción entre un humano y dispositivos electrónicos, robots domésticos, entre otros, siendo ésta última

de interés para este trabajo de investigación y el cuál se hizo mención en el Capítulo 1.

Específicamente en la última área antes mencionada, el reconocimiento de acciones puede ser implementado en diversas actividades, por ejemplo, para ayudar en la cocina cocinando u siendo de apoyo para la persona en cuestión, para vigilar la casa, para organizar objetos como puede ser la ropa, entre otros. Estos sistemas de reconocimiento de acciones analizan las secuencias de movimiento que detectan mediante cámaras y sensores para obtener la actividad u acción que se está realizando. El objetivo de estos sistemas [1] es el de clasificar correctamente los videos o secuencias de imágenes que reciben, de acuerdo a la actividad que se está realizando, en casos en los que estas acciones se estén ejecutando continuamente, el reconocimiento debe realizarse durante segmentos del video recibido, delimitados por un inicio de la actividad y un fin de la misma.

Para poder reconocer acciones u actividades hay dos tipos de enfoques de acuerdo con [1]: un enfoque *jerárquico* y un enfoque *de una sola capa* (figura 2-1), siendo el primero una representación de actividades humanas descritas en términos de otras más simples, llamados en general como *subeventos*. Estos se clasifican en la metodología de reconocimiento utilizada: enfoques estadísticos, sintácticos y basados en descripciones. Es de importancia mencionar que un ejemplo de los enfoques estadísticos son los modelos de Markov de una capa, siendo inspiración de una parte del enfoque que se usó en este trabajo y se hablará más adelante.

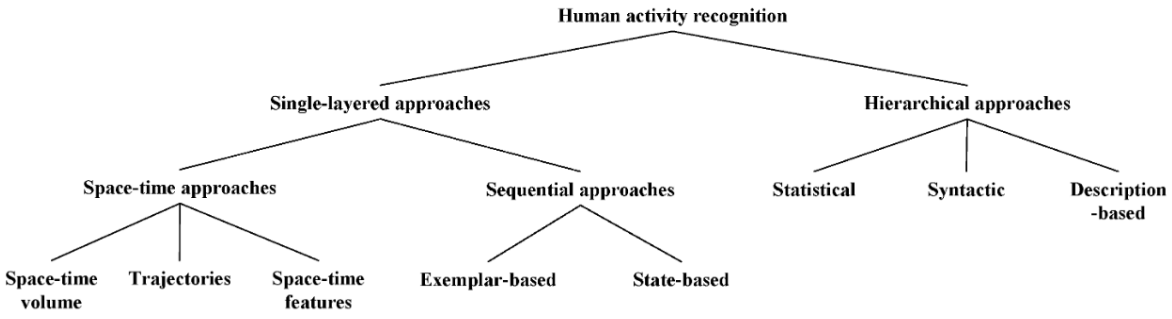


Figura 2-1: Enfoque jerárquico del reconocimiento de acciones humanas [1]

Los enfoques *de una capa* son los que representan y reconocen directamente de una secuencia de imágenes, reconociendo acciones y gestos con características secuenciales. De igual manera estos enfoques se clasifican en dos categorías dependiendo el modelado de las acciones humanas

(figura 2-1): enfoque espacio-temporal y enfoques secuenciales.

2.2. Técnicas basadas en Modelos de Markov

Como se mencionó anteriormente, el enfoque principal en este trabajo para el reconocimiento de acciones es implementando Modelos de Markov, más específicamente los Modelos Ocultos de Markov (HMM, por sus siglas en ingles).

Una cadena de Markov es un proceso estocástico discreto en el tiempo, representado normalmente mediante un conjunto de estados ligados (figura 2-2), el cual satisface la propiedad de que la probabilidad de que transición de un evento (estado) dependerá únicamente del estado actual, llamada *Propiedad de Markov*. Éstos son utilizados principalmente para procesos que varían en el tiempo.

Un Modelo Oculto de Markov es un Modelo de Markov en las que las observaciones son funciones probabilísticas de algún estado. Es un proceso doblemente estocástico [31] donde uno de éstos es un proceso estocástico *oculto*, es decir, no es observable, pero es posible hacerlo a través de otro conjunto de procesos estocásticos que producen una secuencia de símbolos observados o *emisiones*.

Lo anterior y en general la teoría matemática de los HMM se profundizará en el Capítulo 3.

Los Modelos de Markov y los HMM específicamente han sido utilizados en una amplia variedad de aplicaciones, siendo muy popular en áreas relacionadas con reconocimiento del habla ([32],[21]), modelado del lenguaje natural [25], pero también utilizado en otras como en la localización de un robot de servicio [12], para análisis de ciertas secuencias biológicas [20], o entre otras situaciones en las que se procesen datos secuenciales. Debido a la definición de estos HMM, y como posteriormente en el Capítulo 3 se hará énfasis, los HMM utilizan información secuencial discreta, por lo que en general en el estado del arte de estos modelos ([41], [20], [42], [24], [22] y los anteriores mencionados), se requiere un procesamiento previo de la información para discretizarla (o cuantizarla) y poder obtener *símbolos* que será una manera de representar la información, para poder implementar posteriormente algún modelo establecido de Markov.

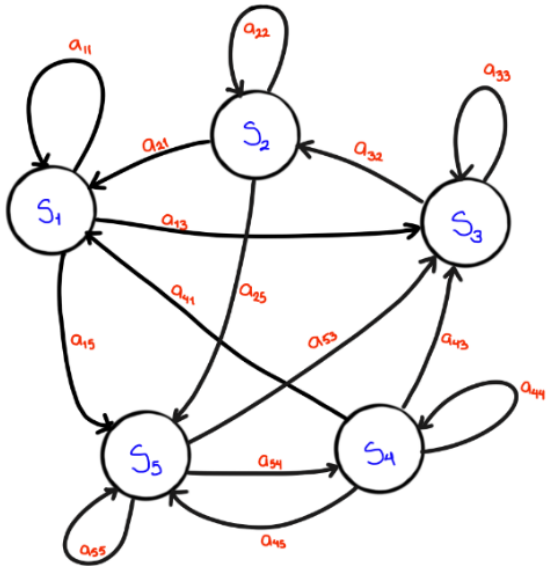


Figura 2-2: Ejemplo de una cadena de Markov donde las transiciones a_{ij} entre estados S_i se definen mediante probabilidades [37].

Por lo tanto, en esta sección se hará una breve mención del pre-procesamiento utilizado y un poco más extenso del modelo implementado.

Con la diversa utilidad de los HMM antes mencionados, naturalmente está el uso para reconocer acciones de una persona, al poder modelar estas acciones como datos secuenciales, es decir el movimiento de la persona al hacer una acción se realiza con el desplazamiento de las extremidades del cuerpo en un tiempo determinado, obteniendo la información de estos movimientos con alguna técnica determinada, como se presenta en [24] al obtener información del esqueleto de la persona mediante extracción de coordenadas usando imágenes con información de profundidad, posteriormente se procesa esa información con algoritmos de cuantización vectorial, para poder entrenar el HMM y reconocer comportamientos humanos; específicamente *K-Medias* es el algoritmo utilizado como cuantizador vectorial en este trabajo, en el cual establecen tres *clusters* por cada acción, en [22] se utiliza una cantidad de cinco clusters con el mismo algoritmo de cuantización antes mencionado, contrastando al trabajo de Yamato, Ohya y Ishii [41] el cual utilizan 72 símbolos obtenidos mediante *cuantización vectorial* para representar 6 distintas *poses* de su experimento.

Variaciones de los HMM como el modelo *Max-Margin* de Markov [42], el HMM con pesos promediados [30], el HMM acoplado (*Coupled HMM*) [3] entre otros modelos para el reconocimiento de acciones, son algunos ejemplos de la versatilidad de éstos para adaptarse al problema por resolver, por ejemplo en el último mencionado ([3]), adecuado para el reconocimiento de actividades que provengan de distintas fuentes, se implementa formando una colección de HMM en la que cada uno de estos maneja un flujo de datos.

2.3. Técnicas basadas en Redes Neuronales Profundas

Parte esencial de este trabajo requiere usar técnicas de aprendizaje automático, Entre las múltiples técnicas para reconocer acciones mediante una secuencia de imágenes, son con el uso de las Redes Neuronales (NN, por sus siglas en ingles), NN Convolucionales (CNN) o NN Recurrentes (RNN).

En el Capítulo 4 se abordará más detalladamente lo relacionado a la teoría y conceptos de las redes neuronales profundas.

2.3.1. Redes Neuronales Convolucionales

Estas redes han tenido una gran popularidad en la rama de la Inteligencia Artificial (IA) para tareas de clasificación, detección, segmentación y clasificación de imágenes. Lo anterior ha llevado a aplicar las Redes Neuronales Convolucionales (CNN, por sus siglas en ingles) en una amplia variedad de áreas de investigación que involucren el procesamiento de imágenes o videos, desde el área médica hasta el área de análisis geográfico, y particularmente en el área de robótica, donde se hace énfasis para este trabajo en la robótica doméstica. Todo lo anterior continuó con éxito debido a la eficacia de implementar este tipo de red neuronal, y ha ayudado al aprendizaje profundo a tener la fama o reconocimiento que tiene hoy en día.

Las CNN se caracterizan de las redes neuronales profundas por implementar la operación matemática de la convolución, como su nombre lo enfatiza, en las operaciones que se realizan entre las capas de entrada y las capas ocultas. La razón de utilizar esta operación es poder

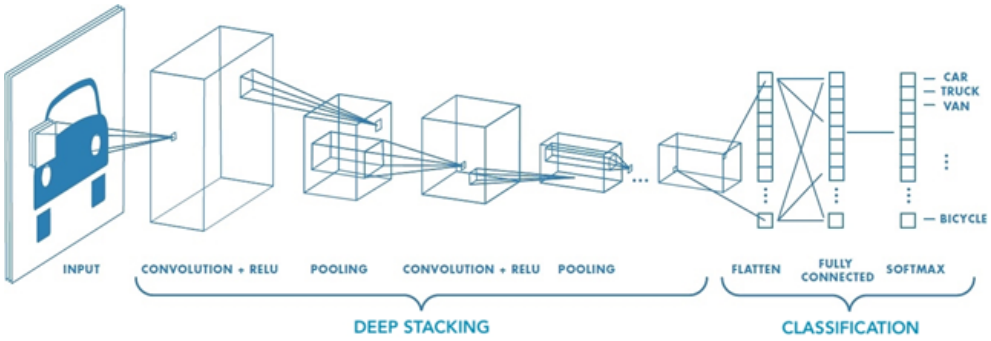


Figura 2-3: Ejemplo de una CNN [38]

obtener distintas características de una imagen, por ejemplo, obtener bordes y perfilados, usando diversos filtros o *máscaras* y poder obtener parámetros de *pesos* y *sesgos* que aprenderá la red al ser entrenada y ayudará a ejecutar las tareas asignadas posteriormente de manera más eficiente.

Para el caso particular del reconocimiento de acciones, el uso de CNN para el reconocimiento de acciones puede aplicarse de distintas formas, por ejemplo usando un banco de características de videos [17] el cual reconoce patrones lineales para reconocer acciones humanas.

Como se mencionó al inicio de este capítulo, el enfoque de este trabajo es el reconocimiento de acciones basado en el movimiento del *esqueleto* de un humano, es decir mediante una secuencia de imágenes del esqueleto de una persona y así reconocer e inferir la acción que se está ejecutando.

Como se mencionó anteriormente, en el Capítulo 4 se plantea detalladamente los conceptos de las redes neuronales y también de las CNN.

OpenPose

Debido al enfoque de este trabajo, es necesario mencionar con suficiente detalle una herramienta utilizada llamada “OpenPose”.

OpenPose [5] es una red neuronal para estimar una pose humana mediante la detección de una serie de articulaciones específicas de una persona y así construir una representación del esqueleto humano. Desarrollada con el objetivo de permitirle a las maquinas tener un mayor entendimiento de las personas en imágenes y videos en el campo del aprendizaje profundo e

Figura 2-4: Ejemplo de aplicación de OpenPose [5]

Inteligencia Artificial.

Para poder describir el funcionamiento de esta red, es posible dividirlo en tres secciones. La primera sección corresponde a la obtención y procesamiento de los datos de entrada, en la que la red toma imágenes RGB como datos de la red neuronal convolucional (CNN), con una configuración "multi-etapa de dos ramas" como se puede ver en la imagen 2-5.

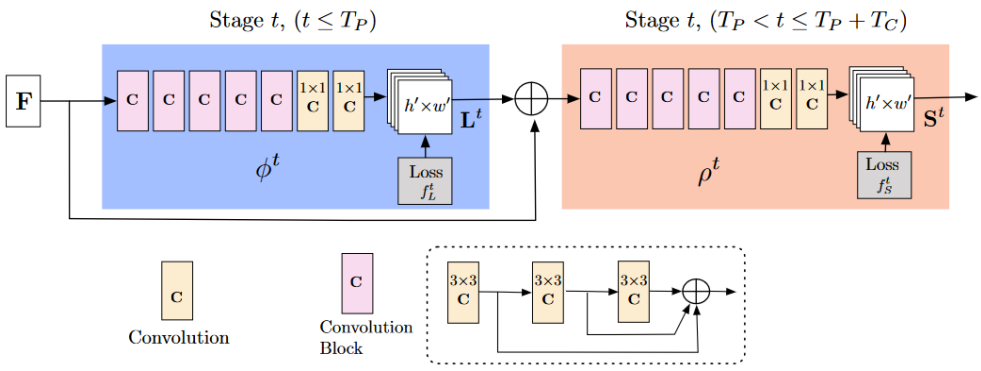


Figura 2-5: Arquitectura de la CNN de OpenPose [5]

El ser "multi-etapa" significa que solamente que la red se apila en cada etapa, lo que le ayuda a que en las últimas etapas capturen datos más refinados, y que sea de dos ramas significa que produce dos salidas distintas.

La segunda sección corresponde a la obtención de los mapas de confianza, en las que se ubican las distintas partes del cuerpo en una imagen y posteriormente obtener los mapas de afinidad, que permiten encontrar la manera en que se conectan las partes antes detectadas. En la figura

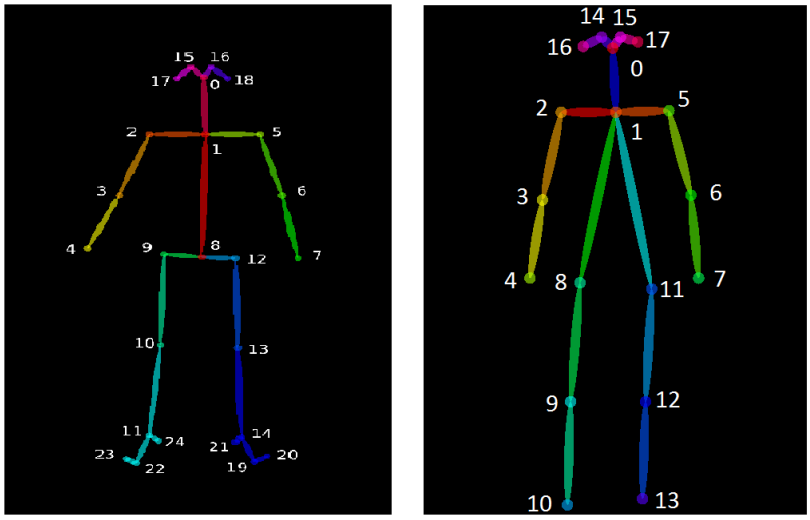
2-5, dado que es multi-etapa, en las primeras 4 etapas ([5], pág. 3) se produce el conjunto de campos de afinidad L^t y concatenado con las características de la imagen original, se producen en las siguientes 2 etapas los mapas de confianza S^t.

La ultima sección corresponde a la conexión de los puntos clave bidimensionales (ejemplo en figura 2-4 b)) generados mediante algoritmos de fuerza bruta que procesan los mapas de confianza y campos de afinidad, para poder tratarlos como si se conectan grafos. El resultado final del sistema en la figura 2-5 es un conjunto de mapas de confianza S que se puede describir matemáticamente de la forma:

$$S = (S_1, S_2, ..., S_i). \tag{2-1}$$

$$S_i = \mathbb{R}^{wxh}. \tag{2-2}$$

Donde i es el número de partes del cuerpo detectados, que puede variar de acuerdo con el conjunto de datos que se entrenó *OpenPose*. Para el conjunto de datos de COCO son $i = 20$ y para el conjunto de datos BODY_25 resulta $i = 25$, como se muestra en la figura 2-6



a) Partes del cuerpo para BODY_25

b) Partes del cuerpo para COCO

Figura 2-6: Distintos modelos del cuerpo de acuerdo a los conjuntos de datos para entrenamiento [4]

De esta manera se establece el contexto general de esta herramienta y su funcionamiento. Los detalles específicos de la implementación de esta herramienta para este trabajo se detallan

en el capítulo 5 y 6.

Mask R-CNN

De manera similar con OpenPose, es de importancia detallar otra herramienta usada en este trabajo. Parte del objetivo principal de este proyecto es relacionar el uso de objetos con la acción humana, por lo tanto es necesario tener un sistema que detecte y reconozca objetos. Dando continuidad con el proyecto en el que se está basando éste [40], se hace uso de una red neuronal profunda para detectar objetos, específicamente se usa la red *Mask R-CNN* [14]. Esta red es una CNN basado en regiones, siendo una extensión de la red *Faster R-CNN* en la que agrega una rama para predecir máscaras de objetos paralelamente con la ya implementada rama que reconoce cuadros delimitadores.

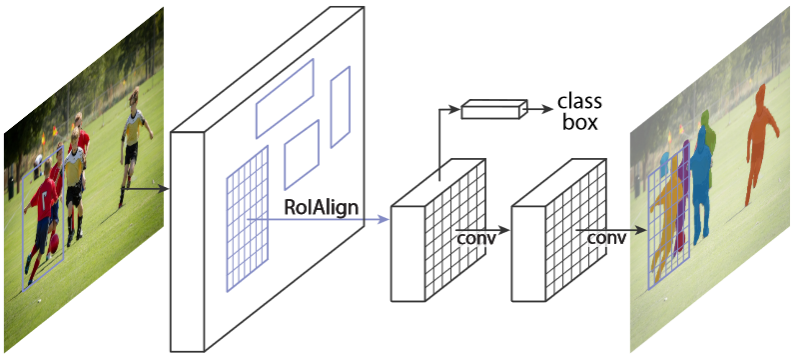


Figura 2-7: Red Neuronal Mask R-CNN [14]

Tiene el objetivo de dar resultados de segmentación de instancias en áreas de visión computacional y aprendizaje automático, es decir que puede separar y obtener regiones de píxeles de objetos distintos en una imagen o video, por lo que como se mencionó anteriormente, el resultado son cuadros delimitadores de objetos, conjuntos de cuadros de píxeles o *máscaras* de una misma clase de objeto, así como una etiqueta del objeto reconocido.

El funcionamiento general de esta red es en dos etapas, similar a la red *Faster R-CNN*. Una primera etapa o módulo igual compuesta por una red liviana RPN (Region Propousal Network, figura 2-8) que ayuda a la red proponiendo regiones en las que se debe poner atención ya que

puede existir los objetos de interés en una imagen. Paralelamente, una segunda etapa o módulo en la que predice los cuadros delimitadores y las clases a la que pertenece.

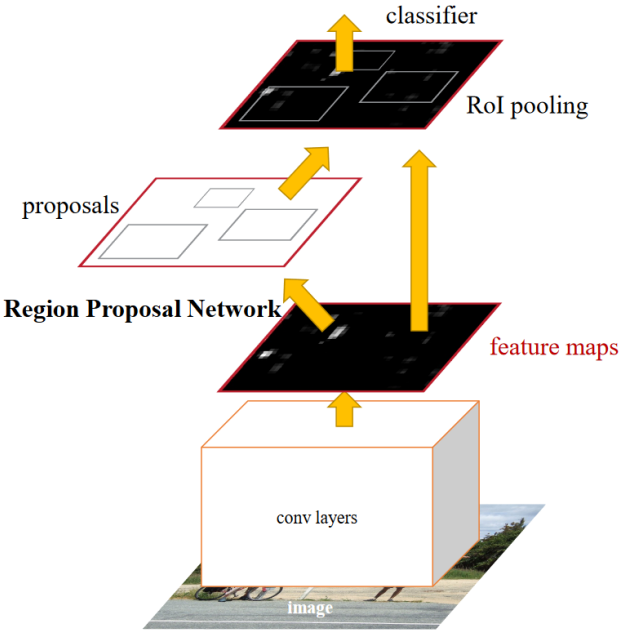


Figura 2-8: Primera etapa: módulo RPN [14]

2.3.2. Redes Neuronales Recurrentes

Las redes neuronales recurrentes (RNN, por sus siglas en inglés) son otro tipo de Red Neuronal Profunda, distinta a la convolucional mencionada anteriormente, en la que hacen uso de información utilizada en iteraciones anteriores junto con los datos de entrada para entrenar a la red con más información y hacer un uso eficiente de datos secuenciales. Son implementadas para analizar secuencias de datos temporales en un tiempo dado en áreas como por ejemplo el procesamiento del lenguaje natural, procesamiento de textos, y también particularmente se puede implementar para analizar secuencia de imágenes o video para reconocer acciones. Una extensión de estas redes RNN son las redes LSTM (*Long-Short Term Memory*), introducidas en 1997 [15] con el objetivo de solucionar algunas deficiencias de las redes RNN, específicamente con la memoria a corto plazo, cuando las secuencias de datos son suficientemente largas, estas redes no lograban *recordar* información de iteraciones anteriores lejanas, o dicho de otro modo,

las redes no lograban mantener información después de un largo periodo de tiempo. Lo anterior logró que estas redes LSTM tuvieran suficiente éxito, y se extendiera más su desarrollo a actividades mucho más complejas. El funcionamiento general y conceptos de las redes RNN se detallan en el capítulo 4.

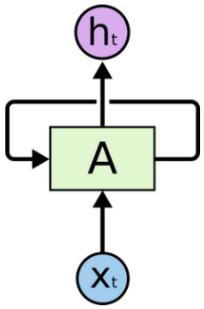


Figura 2-9: Ejemplo de una RNN simple, con datos de entrada X_t, salida h_t y donde se destaca la parte cíclica de la red RNN A [28]

Existen diversos trabajos para reconocer o predecir acciones usando este tipo de redes, por ejemplo en [8] se presenta una serie de reportes con los que se trabajó el reconocimiento, detección y anticipación de acciones, entre otros. Para los trabajos anteriores se manejó un conjunto de datos [7] en los que los videos tienen una perspectiva de *primera persona*, distinto a trabajos antes mencionados en los que el conjunto de datos de video o imágenes se enfoca generalmente en *tercera persona* (figura 2-10), por lo que no hace uso de la pose o esqueleto de una persona para reconocer o predecir una acción.

Figura 2-10: Reconocimiento de acciones, usados en *EPIC-KITCHENS CHALLENGE 2021* [7]

En general, de [8] se presenta el uso de Redes Neuronales Profundas, entre los que reportan mejores resultados, implementaron los *transformers* para modelar la naturaleza temporal de los videos a través de la *atención* para relacionar eventos distantes. Otros trabajos reportan el uso de *Rolling-Unrolling LSTM* para la anticipación de acciones. Para el reconocimiento de acciones, el reporte con mejor resultado implementa redes **SlowFast** (siendo ésta una red CNN), entrenada para predecir acciones y verbos y sustantivos que se implementan también en el conjunto de datos.

Capítulo 3

Modelos Ocultos de Markov

En este capítulo se profundizan los conceptos teóricos y matemáticos de los modelos de Markov, así como los Modelos Ocultos de Markov (*HMM*, por sus siglas en ingles), los cuáles fueron brevemente introducidos en el capítulo anterior. Tiene como objetivo establecer y poner en contexto el conocimiento básico necesario y fundamental que conlleva este trabajo de investigación.

3.1. Cadenas de Markov

Los conceptos de las cadenas de Markov fueron introducidos a inicios del siglo XX por el matemático ruso Andréi Andréyevich Márkov, con el objetivo de modelar frecuencias de aparición de vocales en textos literarios y poemas [35], teniendo un gran éxito al ser lo suficientemente simples para poder ser analizadas matemáticamente, pero de igual manera siendo lo suficientemente complejas para poder describir características no triviales de algunos sistemas. Esta teoría básica se desarrollará más adelante, en la década de 1960 e inicios de 1970 en áreas relacionadas al reconocimiento del habla, en el cual se introduciría el concepto de *modelo oculto de Markov*.

Como se mencionó en la sección 2.2 del Capítulo 2, las cadenas de Markov o también llamados *modelos* de Markov son procesos estocásticos en tiempo discreto los cuales cumplen

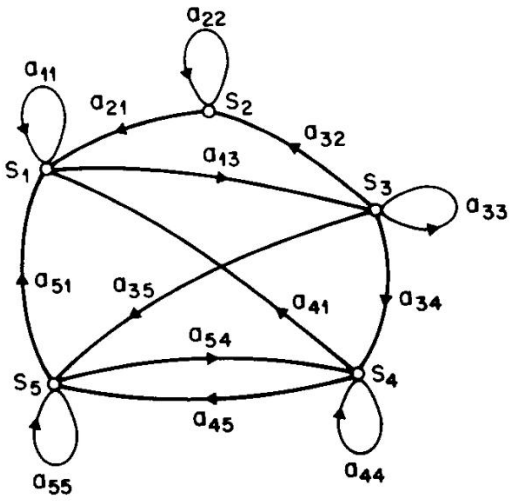


Figura 3-1: Cadena de Markov [33]

con la llamada *propiedad de Markov*, la cual establece que la probabilidad de que un estado transite a otro estado en un tiempo determinado depende únicamente del estado actual y no de los eventos anteriores. Se puede describir esta propiedad de la siguiente forma:

$$p(q_t = S_i | q_0 = S_0, q_1 = S_1 ..., q_{t-1} = S_j) = p(q_t = S_i | q_{t-1} = S_j) \tag{3-1}$$

Lo anterior con un número finito de estados S_n que se cumple para cadenas de Markov discretas de primer orden, por lo que para casos en el que un estado depende de otro además del inmediatamente anterior, se llamaran procesos de Markov de orden superior. También se puede establecer el conjunto probabilidad de transición a_{ij} de la siguiente forma:

$$a_{ij} = p(q_t = S_i | q_{t-1} = S_j), \quad 1 \leq i, j \leq N \tag{3-2}$$

Con las siguientes propiedades:

$$a_{ij} \geq 0$$

$$\sum_{j=1}^{N} a_{ij} = 1$$

De este conjunto a_{ij} se obtiene la matriz de probabilidades de transición para una unidad de tiempo variando i (filas) y j (columnas) sobre el espacio de estados $S = \{0, 1, ...\}$, $i, j \in S$:

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0N} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1N} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N0} & a_{N1} & a_{N2} & \cdots & b_{NN} \end{pmatrix} \tag{3-3}$$

También se definirá la matriz $b_j(k)$ como la distribución de probabilidad del símbolo k, $k \in \{0, 1, ..., M\}$ emitido en el estado j en el espacio de estados S, de manera:

$$B = \begin{pmatrix} b_{00} & b_{01} & b_{02} & \cdots & b_{0M} \\ b_{10} & b_{11} & b_{12} & \cdots & b_{1M} \\ b_{20} & b_{21} & b_{22} & \cdots & b_{2M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{N0} & b_{N1} & b_{N2} & \cdots & b_{NM} \end{pmatrix} \tag{3-4}$$

Así como un vector de probabilidad inicial π de manera:

$$\pi = \begin{pmatrix} \pi_0 & \pi_1 & \pi_2 & \cdots & \pi_N \end{pmatrix} \tag{3-5}$$

3.2. Modelos Ocultos de Markov

Un modelo oculto de Markov (HMM, por sus siglas en inglés:*Hidden Markov Model*) es un proceso estocástico doble de un número finito de estados, como inicialmente se mencionó también en la sección 2.2, donde las señales a modelar se asume que es un proceso de Markov con estados ocultos [33]. Los valores de salida de estos modelos pueden ser discretos, dada una distribución categórica, o continuos en el caso de ser generados por una distribución Gaussiana. Estos HMM permiten manejar datos secuenciales y proveen invariabilidad en el tiempo al momento de problemas de reconocimiento. Estos modelos incorporan la característica de tener

la habilidad de *aprender* con los datos que procesen y optimizando automáticamente el modelo con esta información. Lo anterior se detallará en secciones posteriores.

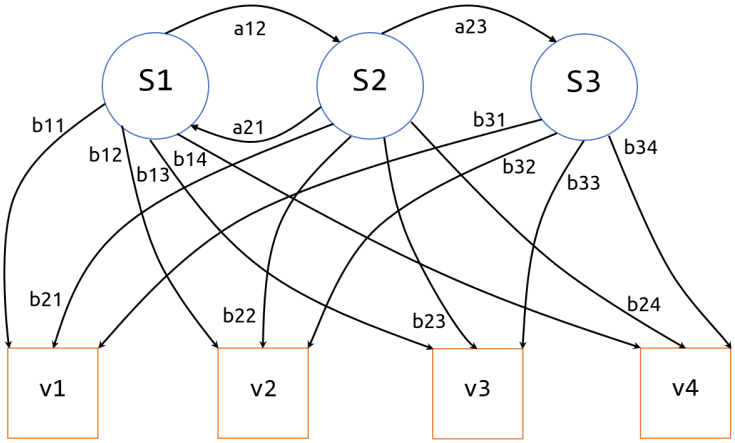


Figura 3-2: Ejemplo de un modelo de Markov Oculto [37]

Como su nombre lo dice, las observaciones V_k es aquella información de estos modelos que sí se puede conocer u observar, para poder hacer predicciones, por ejemplo los resultados de lanzar una moneda [33] o el estado del clima lluvioso, despejado, nublado, etc. [37], por lo que la naturaleza de estas observaciones dependerán del problema a tratar. De la figura 3-2, estas observaciones $\{V1, V2, V3\}$ tendrán su probabilidad b_{ij} dada el estado S_i el cual se define en la matriz de la ecuación 3-4.

3.2.1. Elementos de un HMM

Se puede describir al HMM como una tupla (λ) de la forma $\lambda = (N, M, A, B, \pi)$, donde se puede encontrar también una versión compacta [31] con únicamente los últimos tres (A, B, π). Los elementos de esta tupla se describen de la forma siguiente:

Con lo anterior y asignando valores apropiados, el modelo se puede utilizar como generador para dar una secuencia de observaciones $O = O_1 O_2 \dots O_T$; cada uno de estos O_i será uno de los símbolos de V.

Tabla 3-1: Elementos de un HMM.

Valor	Descripción
N	El número de estados *ocultos* en el modelo. Estos estados se denotan de la forma $S = \{S_1, S_2, ..., S_N\}$ y un estado en el tiempo t de la forma q_t
M	El número de símbolos o también llamado alfabeto de observaciones distintas por estado. Los símbolos se denotan individualmente de la forma $V = \{v_1, v_2, ..., v_M\}$
A	La matriz de probabilidades de transición descrita en 3-2 y 3-3
B	La matriz de probabilidades de emisiones, donde $B = \{b_j(k)\}, \quad b_j(k) = P[V_k \ en \ t | q_t = S_j]$. $1 \leq j \leq N, 1 \leq k \leq M$
π	La distribución de probabilidades inicial $\pi_i = P[q_1 = S_i], \quad 1 \leq i \leq N$

3.2.2. Los tres problemas a resolver en un HMM

Para que estos modelos sean útiles con aplicaciones reales, es necesario resolver tres problemas básicos:

1. ¿Cómo se puede calcular eficientemente la probabilidad de observación de una secuencia dado el modelo, $P(O|\lambda)$, dado una secuencia observada $O = O_1 O_2 \cdots O_T$ y el modelo $\lambda = (A, B, \pi)$?. Lo anterior es llamado el problema de evaluación e igualmente se puede abordar como un problema de puntuación sobre que tan bien el modelo coincide con la secuencia de observación dada. Para poder resolver este problema se implementa el *algoritmo de adelanto* el cual se detallará más adelante.
2. ¿Cómo se escoge una secuencia de estados correspondiente $Q = q_1 q_2 \cdots q_T$ que mejor pueda *explicar* la secuencia de observación $O = O_1 O_2 \cdots O_T$, dado lo anterior y el modelo λ?. Para poder abordar este problema eficientemente, se implementa el *algoritmo de Viterbi*, el cual se detalla también más adelante.
3. ¿Cómo se pueden ajustar los parámetros del modelo $\lambda = (A, B, \pi)$ para maximizar $P(O|\lambda)$?, es decir, encontrar la manera de maximizar la probabilidad de generar un conjunto de observaciones dado el modelo. De igual manera que los problemas anteriores, para poder resolverlo se implementa un algoritmo llamado *método de Baum-Welch.*

Algoritmo de Adelanto

Este algoritmo permite resolver el problema de evaluación, por lo que para una variable de *adelanto* $\alpha_t(i)$ de acuerdo con la siguiente definición de la probabilidad de una secuencia de observación parcial hasta el tiempo t y el estado S_i en el tiempo t:

$$\alpha_t(i) = P(O_1 O_2 \cdots O_t, q_t = S_i|\lambda) \tag{3-6}$$

Es posible resolver la ecuación de manera inductiva de la siguiente forma:

1. Inicialización:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{3-7}$$

2. Inducción:

$$a_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1 \qquad 1 \leq j \leq N \tag{3-8}$$

3. Término:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{3-9}$$

Lo anterior se puede describir en una primera etapa donde establece las probabilidades hacia adelante como la probabilidad conjunta de los estados S_i y de la observación inicial O_1. Posteriormente, de manera inductiva calcula la probabilidad parcial en el estado S_j para el tiempo $t+1$ en conjunto con las observaciones parciales. Por último, en una etapa de cálculo de la probabilidad hacia delante final, sumando las variables terminales $\alpha_T(i)$. El algoritmo antes descrito que puede encontrarse también en la literatura como *procedimiento hacia adelante*, es parte de un algoritmo general llamado *procedimiento hacia adelante-hacia atrás* [31] (del inglés *Forward-Backward Procedure*). La segunda parte del algoritmo (procedimiento *hacia atrás*) utiliza la variable $\beta_t(i)$ definida como la probabilidad de la secuencia de observación parcial desde $t+1$ hasta el final dado el estado S_i en el tiempo t y el modelo λ , es decir:

$$\beta_t(i) = P(O_{t+1}Ot + 2 \cdots O_T | q_t = S_i, \lambda) \tag{3-10}$$

De manera análoga se resuelve de manera inductiva de la siguiente forma:

1. Inicialización:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{3-11}$$

2. Inducción:

$$\begin{gathered} \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \\ t = T-1, T-2, \cdots, 1, \quad 1 \leq i \leq N \end{gathered} \tag{3-12}$$

Algoritmo de Viterbi

Para poder resolver el segundo problema se busca la secuencia "óptima" de estados asociado con la secuencia de observaciones dada. Una manera eficiente de calcular lo anterior es con el algoritmo de Viterbi. Éste utiliza una variable δ el cual será la probabilidad más alta de un camino simple, $Q = \{q_1 q_2 \cdots q_T\}$, para la secuencia observada dada $O = \{O_1 O_2 \cdots O_T\}$ y el modelo λ, definido de la forma:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P[q_1 q_2 \cdots q_t = i, O_1 O_2 \cdots O_t | \lambda] \tag{3-13}$$

Donde $\delta_t(i)$ será el valor más alto de una trayectoria o camino individual, para un tiempo t, contando las primeras t observaciones y termina en el estado S_i. Inductivamente se tiene para un tiempo $t+1$:

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij} \right] \cdot b_j(O_{t+1}) \tag{3-14}$$

El procedimiento completo para poder obtener la mejor secuencia de estados, controlado el argumento que maximiza la ecuación 3-14 por cada t y j, se hace mediante el arreglo $\psi_t(i)$ de la siguiente forma:

1. Inicialización:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \tag{3-15}$$

$$\psi_1(i) = 0 \tag{3-16}$$

2. Recursión:

$$\begin{aligned} \delta_t(j) = \max_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(O_j), \quad 2 \leq t \leq T \\ 1 \leq j \leq N \end{aligned} \tag{3-17}$$

$$\begin{aligned} \psi_t(j) = \operatorname*{argmax}_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right], \quad 2 \leq t \leq T \\ 1 \leq j \leq N \end{aligned} \tag{3-18}$$

3. Término:

$$\begin{aligned} P^* = \max_{1 \leq i \leq N} \left[\delta_t(i)\right] \\ q_T^* = \operatorname*{argmax}_{1 \leq i \leq N} \left[\delta_t(i)\right] \end{aligned} \tag{3-19}$$

4. Secuencia de estados (camino) mediante un rastreo hacia atrás:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, ..., 1 \tag{3-20}$$

Algoritmo de Entrenamiento

Por último, se encuentra este algoritmo para resolver el tercer problema antes mencionado, siendo éste el más complicado al no ser un problema trivial. Al no tener una manera analítica de poder resolver la maximización de la probabilidad de la secuencia de observaciones [33], se utilizan varios métodos los cuales no es posible decir aún cuál de ellos es el más óptimo. Entre los métodos para resolver este problema está el método de Baum-Welch, otro método equivalente llamado el método EM (del inglés *expectation-modification*), y métodos que emplean

técnicas de gradientes. Para fines de este trabajo se detallará el primero de estos métodos antes mencionados el cual es un algoritmo iterativo para poder escoger los parámetros del modelo. Adicional al anterior se utilizó un método para entrenar los modelos muy parecido pero que se puede nombrar como el *entrenamiento de Viterbi*.

El algoritmo de Baum-Welch utiliza la variable $\xi_t(i,j)$, como la probabilidad de estar en el estado S_i en el tiempo t y en el estado S_j en el tiempo $t+1$, dado el modelo λ y la secuencia de observaciones:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{3-21}$$

Utilizando las definiciones de las variables de adelanto (ec. 3-6) y retroceso (ec. 3-10), es posible redefinir $\xi_t(i,j)$ de la siguiente forma:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \tag{3-22}$$

$$= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1} \beta_{t+1}(j)} \tag{3-23}$$

Posteriormente se define una variable $\gamma_t(i)$ como la probabilidad de estar en el estado S_i en el tiempo t dada la secuencia de observaciones y el modelo, de la forma $\gamma_t(i) = P(q_t = S_i | O, \lambda)$, con el cual se puede expresar análogamente con las variables de adelanto y retroceso de la forma:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum\limits_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{3-24}$$

Lo anterior se puede relacionar con $\xi_t(i,j)$ sumando sobre j de la forma:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j). \tag{3-25}$$

donde

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{número esperado de transiciones de } S_i \tag{3-26}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{número esperado de transiciones de } S_i \text{ a } S_j \tag{3-27}$$

Es posible dar un método de reestimación de parámetros de un HMM al utilizar las anteriores fórmulas y con el concepto del conteo de ocurrencias de eventos. Para los conjuntos π, A, B las fórmulas para estas reestimaciones estarán dadas de la forma:

$$\bar{\pi}_i = \text{frecuencia esperada en el estado } S_i \text{ en el tiempo } (t=1) = \gamma_1(i) \tag{3-28}$$

$$\bar{a}_{ij} = \frac{\text{número esperado de transiciones del estado} S_i \text{ al estado } S_j}{\text{número esperado de transiciones del estado } S_i} \tag{3-29}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{3-30}$$

$$\bar{b}_j(k) = \frac{\text{número esperado de veces del estado } j \text{ observando el símbolo } v_k}{\text{número esperado de veces en el estado } j} \tag{3-31}$$

$$= \frac{\sum_{t=1}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(i)} \quad {}^{t.q.\ O_t = v_k} \tag{3-32}$$

3.2.3. Topologías

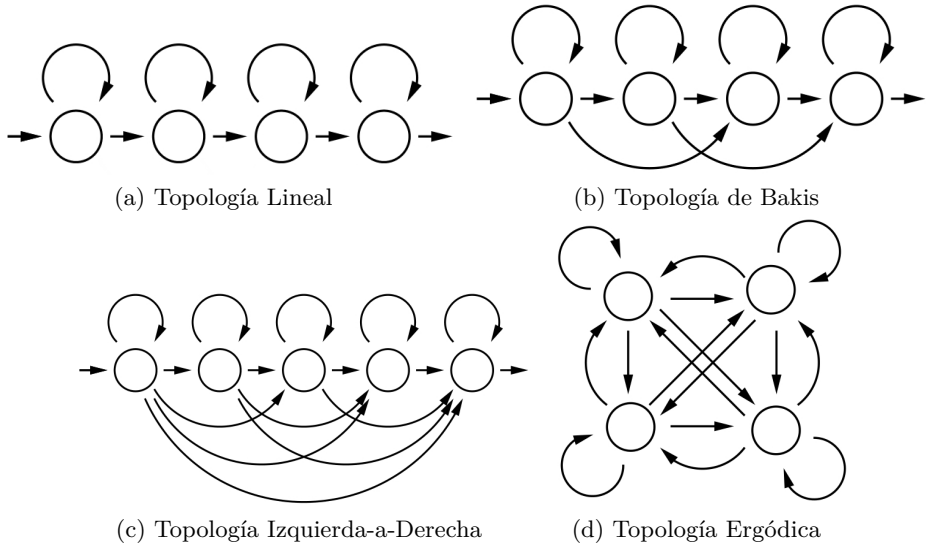
Dependiendo del problema a resolver, el tipo de la señal o información con la que se procesará en el HMM, se establecerá una topología al modelo que mejor se desempeñe. Como se ha mencionado anteriormente, esta información puede llegar como información con estructura cronológica o secuencial, por lo que no es la más óptima para estas aplicaciones que el mode-

lo tenga transiciones de estados arbitrarias [10], como lo sería un modelo (topología) *ergódico* (figura 3-3 (d)) en un HMM.

La topología más simple para un HMM es la llamada HMM *lineal* (figura 3-3 (a)), donde las transiciones solo ocurren entre el mismo estado y un único estado siguiente mediante cierta probabilidad positiva.

Otro ejemplo es el modelo de *Bakis* (figura 3-3 (b)), una topología más flexible en un HMM que permite añadir transiciones de un estado a otro estado siguiente dada una topología lineal. Esta topología es utilizada ampliamente en el área del reconocimiento de escritura y del habla automático [10].

Una variación más ámplia del anterior modelo es la topología *izquierda a derecha* (figura 3-3 (c)), en la que se añaden más transiciones entre estados en un modelo lineal para que éstos *salten* a estados más lejanos; únicamente no se permiten las transiciones a estados anteriores.



(a) Topología Lineal (b) Topología de Bakis

(c) Topología Izquierda-a-Derecha (d) Topología Ergódica

Figura 3-3: Topologías de HMM. Traducido de [10]

Con la flexibilidad que se definen las topologías en un HMM, incrementan los parámetros para el entrenamiento, por lo tanto mayores cálculos computacionales, y mayores caminos existirán a través del modelo también.

Capítulo 4

Redes Neuronales Profundas

En este capítulo se aborda brevemente conceptos y teoría matemática sobre las Redes Neuronales, con el objetivo de introducir esta área del aprendizaje profundo de la Inteligencia Artificial, así como para poner lo mejor posible en contexto todo lo mencionado en el Capítulo 2 y para el desarrollo de este proyecto respecto a este tema. Inicia dando una introducción al primer desarrollo del concepto la Neurona Artificial y su sustento matemático, posteriormente una breve introducción al modelo del *perceptrón* y su posterior implementación con el aprendizaje profundo. Por último se da una descripción y funcionamiento en general de las Redes Neuronales Profundas, así como las Redes Convolucionales y Recurrentes.

4.1. Antecedentes

4.1.1. Neurona biológica y Artificial

Los primeros estudios de las redes neuronales artificiales (ANN, por sus siglas en inglés) surgen en la década de los 50's, teniendo una inspiración biológica de las neuronas del cerebro, donde se observó los sistemas de aprendizaje y su compleja interconexión entre neuronas [26]. De manera muy general se ha descrito la composición de una neurona biológica como en la figura 4-1 , donde está formada por *dendritas* las cuales son las receptoras de señales que provengan de otras neuronas, también se encuentra la *soma*, la cual es la encargada de procesar esa señal que

recibe para luego pasar por el *axión* el cual simplemente transmite la información procesada por la soma a otras neuronas conectadas.

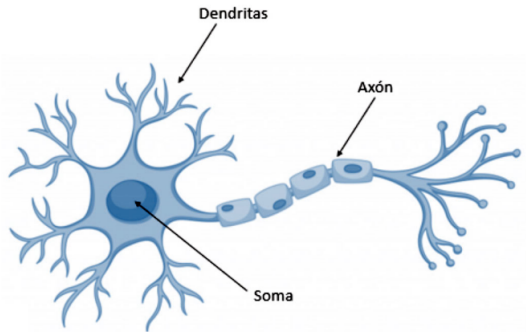


Figura 4-1: Representación de una neurona biológica [37]

Como se menciona anteriormente éste es un funcionamiento general de la neurona, aunque se debe remarcar que muchos otros procesos están pasando también en un nivel más alto [6] y que no se están modelando. También es de importancia mencionar que las neuronas reciben múltiples señales que pudieran ser el resultado del procesamiento en otras neuronas. Se estima que se encuentran $10^1 1$ número de neuronas densamente conectadas, cada una conectada en promedio con otras 10^4 neuronas y la velocidad en la que las señales se transmiten se encuentra en el orden de 10^{-3} segundos [26].

4.1.2. Modelo del Perceptrón

El primer modelo de neurona artificial que trató de simular la manera en que se maneja la información fue propuesto por McCulloch y Pitts en el año 1943, el cual introducen el modelo del *perceptrón*. Este modelo (figura 4-2), el cual se puede dividir en dos partes, la primera donde se recibe múltiples datos (*dendrita*) se procesará (*soma*) mediante una función matemática el cual multiplica cada dato de entrada por un *peso* ω_{ij} y los resultados se suman junto con un *sesgo* θ_j , posteriormente el resultado se procesa por una *función de activación*, la cual se encargará de activar la salida del perceptrón o desactivarlo. Matemáticamente se puede ver a

proceso (también llamado *estimación hacia adelante*) como:

$$u = \sum_{i=1}^{n} x_i \omega_i - \omega. \tag{4-1}$$

$$o_j = f(u). \tag{4-2}$$

Donde ω_i es el conjunto de pesos para una neurona, x_i son las entradas a la neurona, u es la suma de los productos de las entradas con los pesos y el sesgo y $f(u)$ es una función de activación, por ejemplo una función escalón unitario. Actualmente existen distintas funciones de activación, las más usadas o populares son la función *sigmoide*, la función *Rectified Lineal Unit* (*ReLU*, su forma abreviada) y la función *softmax*. Con estas funciones y de la ec. 4-2, $o_j = 1$ si $f(u)$ es mayor o igual al umbral, de otra forma refiriéndose a que se *activa* la salida del perceptrón, y $o_j = 0$ si $f(u)$ resulta menor que el umbral, se *desactiva* la salida.

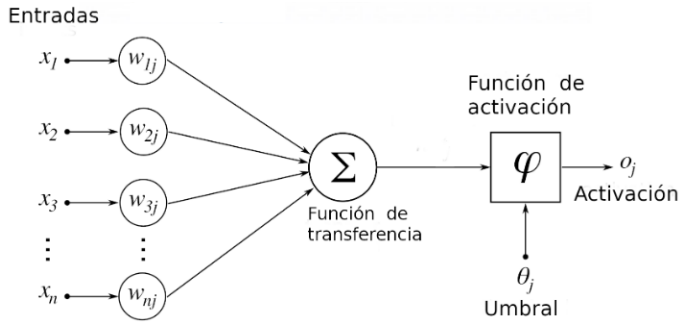


Figura 4-2: Modelo del *perceptrón*

Dado que el perceptrón se desarrolló con la orientación de resolver problemas de clasificación básicos, por ejemplo de funciones booleanas **OR** y **AND**, entre otros, se vio limitado en su momento ya que uno solo era incapaz de resolver la función **XOR**. Con el avance de la investigación de estos modelos simples, surgen en la década de los 60's modelos del perceptrón de múltiples capas, con el objetivo de resolver las limitaciones que el modelo simple no podía resolver. Este modelo multi-capa (figura 4-3)consiste de varios perceptrones conectados entre sí, dando el concepto de una red de neuronas artificiales conectadas, donde el primer conjunto o la primera capa con los datos de entrada, siendo procesados por un conjuntos de perceptrones,

en capas intermedias u *ocultas*, para hacer un proceso como el mencionado anteriormente, para que un último conjunto o capa de perceptrones finalmente obtenga los resultados del problema de clasificación.

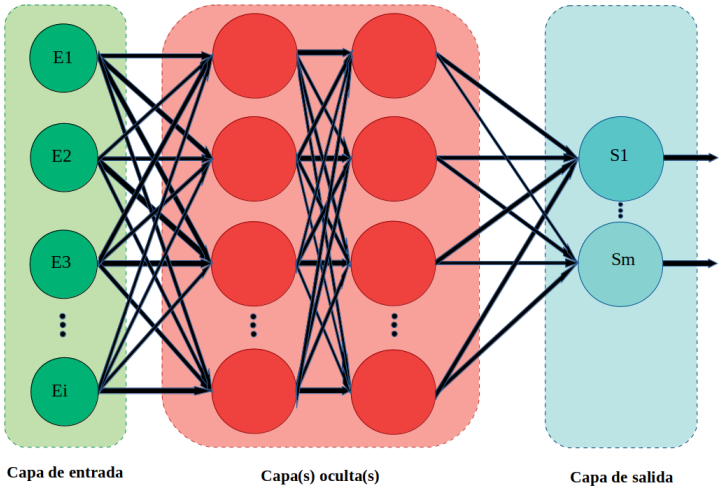


Figura 4-3: Modelo del *perceptrón multicapa*. Ejemplo de un modelo completamente conectado

Para que estos modelos puedan clasificar correctamente, es necesario establecer valores en pesos ω y sesgos θ de tal manera que la salida sea la esperada. Dado que los modelos multicapa incrementan sustancialmente la complejidad de calcular esos valores mientras más grande sea el modelo, es necesario tener un algoritmo que pueda ajustar o *aprender* esos valores para aproximar al valor de salida correcto, dando el concepto de *aprendizaje* de estos modelos.

Algoritmo de Retro-propagación de Errores

Fue presentado formalmente en 1986 en el articulo ***Learning Internal Representations by Back-Propagating Errors*** [36], un algoritmo de aprendizaje supervisado con el objetivo de obtener los valores de pesos ω_{ij} adecuados para cada una de las capas de neuronas artificiales dada las entrada x_i y salidas o_j deseadas.

De manera simple se puede describir el algoritmo de la siguiente forma:

1. De la red dada se inicializan los pesos ω y sesgos θ con valores pequeños aleatoriamente.
2. Se obtiene una salida o_j con el proceso de la estimación hacia adelante (o *feedforward* en inglés).

3. Se calcula el error de la salida anterior y la salida real o esperada mediante una función de error $E(X, \omega)$.
4. Ajustar los pesos ω y sesgos θ mediante el cálculo del gradiente de la función de error respecto a los valores actuales de pesos y sesgos.

Del inciso 3., existen diversas funciones de error $E(X, \omega)$ que pueden ser usado de acuerdo al tipo de problema a resolver, siendo una de éstas es la fórmula del error cuadrático medio, que matemáticamente se describe de la siguiente manera:

$$E(X, \omega) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{4-3}$$

Una vez obtenido el error, se obtiene el gradiente con respecto a los pesos y sesgos:

$$\frac{\partial E(X, \omega)}{\partial \omega_{ij}} = \frac{\partial}{\partial \omega_{ij}} (\frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2) \tag{4-4}$$

$$\frac{\partial E(X, \omega)}{\partial \omega_{ij}} = \frac{\partial E_d}{\partial \omega_{ij}} \tag{4-5}$$

Donde E_d es el valor de la función de error en la capa d de la red. Lo anterior se calcula primero en la última capa, para después propagar el gradiente a capas anteriores intermedias hasta finalmente llegar a la capa de entrada. Haciendo una primera actualización en la última capa de la red, mediante el cálculo de derivadas parciales respecto a los pesos ω y sesgo θ, ajustándolos (actualizándolos). Con un cálculo similar al anterior y con la ayuda de la regla de la cadena para funciones de múltiples variables, se calculan las derivadas parciales respecto a las variables anteriores en las capas intermedias hasta llegar a la primera capa.

Este proceso se iterará hasta que pasen un número establecido de *épocas* o hasta que el error sea menor a un umbral establecido.

4.2. Redes Neuronales Profundas

Como se vio en la sección anterior, el modelo multi-capa del perceptrón puede crecer hasta tener un número considerable de capas ocultas y perceptrones en cada capa. Las redes neuronales profundas se distinguen por tener múltiples capas ocultas y un número considerable de neuronas artificiales en cada capa. Son algoritmos de aprendizaje supervisado, es decir, se conoce el dato de entrada y la salida que se espera, utilizadas en el área de Inteligencia Artificial, específicamente en el Aprendizaje Profundo, para resolver problemas de reconocimiento de patrones y clasificación. Dependiendo del problema a resolver, estas redes pueden variar en tamaño y profundidad, así como la operación matemática que realiza en las capas ocultas, surgiendo de éstas una variante llamada Red Neuronal Convolucional, el cuál como su nombre lo establece, implementa la función de convolución.

4.2.1. Redes Neuronales Convolucionales

Este tipo de red neuronal, como se mencionó anteriormente y también en el capítulo 2, tiene la particularidad de realizar la función matemática de la convolución en las capas intermedias para obtener datos o características adicionales para el entrenamiento de la red. En cada capa convolucional puede haber un filtro distinto en el que obtendrá distintas características del dato de entrada. Son redes muy utilizadas en el procesamiento de imágenes, para clasificación, detección o segmentación, implementadas en múltiples áreas de desarrollo e investigación, como en la robótica, medicina, industria, geografía, entre otros.

La eficacia al implementar estas redes ayudó a popularizar el desarrollo e investigación en el área del Aprendizaje Profundo. Una red especifica que logró lo anterior es la llamada AlexNet [19], desarrollada por Alex Krizhevsky, con la colaboración de Ilya Sutskever y Geoffrey E. Hinton, con el objetivo de tener una red capaz de manejar miles de imágenes etiquetadas para el entrenamiento de una red clasificadora eficiente.

Estas redes en general, como en la figura 2-3 tomarán diversas características de la imagen a través de esta operación hasta obtener un vector de características con el cual una red densa

se encargará en un último paso de hacer la clasificación final.

Convolución

Su nombre proviene del latín *convolvere*, de convolucionar que significa "rodar juntos". Es una operación matemática que transforma dos funciones f y g a una tercera función h de forma:

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(z)g(x-z)\delta z \tag{4-6}$$

Se puede describir la ecuación anterior como el resultado de medir cuanto se superponen dos funciones, en el caso de (4-6), $g(x)$ se puede visualizar en la operación como la función *invertida* y *trasladada* en intervalos de tiempo alrededor de $f(x)$ (figura 4-4). Esta ecuación también se puede efectuar en señales discretas, análogamente a 4-6:

$$h(n) = f(n) * g(n) = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{4-7}$$

Y de manera similar se puede realizar esta operación con señales en dos dimensiones, por ejemplo, en imágenes:

$$h(x,y) = f(x,y) * g(x,y) = \sum_{\alpha=-\infty}^{\infty} \sum_{\beta=-\infty}^{\infty} f[\alpha,\beta]g[x-\alpha, y-\beta] \tag{4-8}$$

$h(x,y)$,$f(x,y)$ y $g(x,y)$ son señales discretas La señal f será el equivalente a una imagen a la cual se quiere analizar, y la señal g será llamado *filtro* porque obtendrá las características de la imagen.

Las imágenes pueden ser tratadas como señales en dos dimensiones representadas mediante matrices, en las que cada elemento corresponde al valor del píxel de la imagen. Si ésta es una imagen RGB (Red Green Blue), será un conjunto de tres matrices en las que cada una corresponde a un canal del color de la imagen. El filtro g puede ser una matriz de un tamaño distinto y menor a la imagen f, en el que, dependiendo de los valores de éste, obtendrá características específicas de la imagen a procesar, por ejemplo bordes.

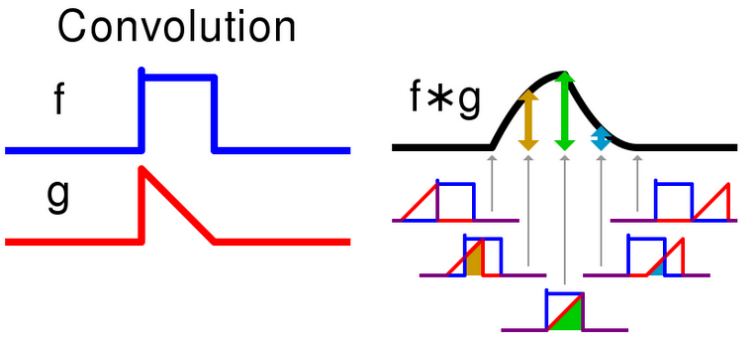


Figura 4-4: Ejemplo de convolución para dos funciones $f(x)$ y $g(x)$. Se visualiza la traslación de g invertida a través de f, midiendo el área bajo la curva. Imagen de *wikipedia (Convolution)*

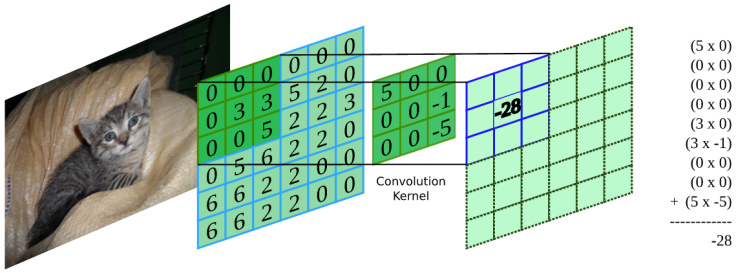


Figura 4-5: Calculo de la convolución en una imagen [34]

4.2.2. Redes Neuronales Recurrentes

Este tipo de red neuronal tiene como característica manejar datos temporales y series secuenciales. De amplio uso en áreas de reconocimiento de texto y lenguaje natural, así como generador de palabras dado un texto anterior, y también en el área de procesamiento y reconocimiento de acciones en video o secuencia de imágenes. El funcionamiento general, como se menciona en el capítulo 2 figura 2-9, consiste en general de una red densa que itera un determinado número de veces, con entradas x_t la cual puede ser la secuencia de un texto, características de una imagen de un video, entre otros, las cuales se procesarán junto con datos procesados de iteraciones anteriores (parte cíclica), y en cada iteración obtendrá resultados h_t que serán procesados de acuerdo al problema a resolver. Esta parte iterativa de la figura 2-9 se puede extender a un conjunto de redes recurrentes conectadas en una estructura tipo cadena (figura 4-6), visualizándose la manera en que la red de una iteración t recibe datos de una iteración

$t-1$.

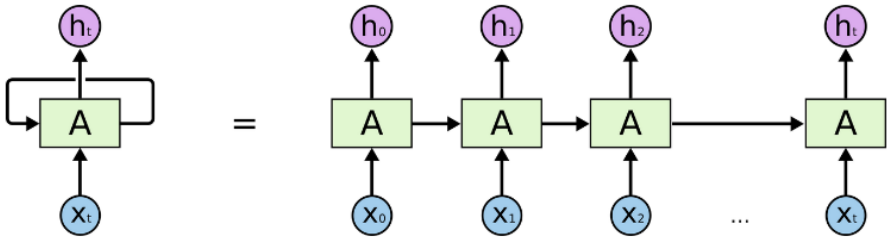
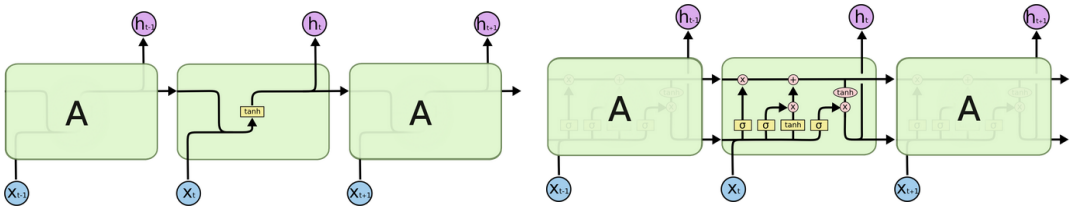


Figura 4-6: Visualización de la parte recurrente de esta red, donde en un tiempo t toma la información del tiempo anterior $t-1$ [28]

Uno de los problemas que surgen con este tipo de red neuronal es la dependencia de datos a largo plazo, donde la información en un tiempo t_i se pierda después de varias iteraciones hasta un tiempo t_j $(j >> i)$ y no pueda haber una conexión adecuada entre datos.

Redes LSTM

Con el problema mencionado anteriormente de las RNN, surgen las redes *Long-Short Term Memory* (*LSTM*), de manera abreviada) o redes de memoria a corto y largo plazo, introducidas por Hochreiter y Schmidhuber en 1997 [16] como una variante de red RNN capaz de manejar dependencias a corto y largo plazo y las cuales han sido desarrolladas y refinadas en la actualidad para tener una popularidad considerable en las áreas antes mencionadas. La arquitectura de estas redes se puede visualizar como en la figura 4-7 *b)*, haciendo una comparación de una red RNN simple con esta variante LSTM, donde la composición de la red neuronal A se ve modificada al añadir más capas o módulos con un propósito especifico.



(a) Estructura interna de una red RNN simple (b) Estructura de una red LSTM

Figura 4-7: Comparación de arquitecturas [28]

Estas redes poseen una estructura en cadena similar a las redes RNN antes mencionadas,

y sus componentes internos pueden variar mínimamente dependiendo del diseño de la red, ya que en la actualidad se han diseñado variantes de redes LSTM para un propósito en específico. La estructura interna consta en general de cuatro capas de redes neuronales (recuadros amarillos, figura 4-7 *b)*) diferenciando solo una de la red RNN simple (figura 4-7 *a)*), en la que se implementa una función de activación diferente, por ejemplo la función σ (sigma) que utiliza una función sigmoide como activación; una función *tanh* o tangente hiperbólica para otra capa interna.

El funcionamiento interno se puede ver en distintas etapas de la red, una primera es utilizando una capa neuronal σ llamada "compuerta del olvido" (figura 4-8), el cual tiene el propósito de procesar la información que se va a desechar o se va a guardar con la información de la iteración anterior h_{t-1} y los datos de entrada actual x_t.

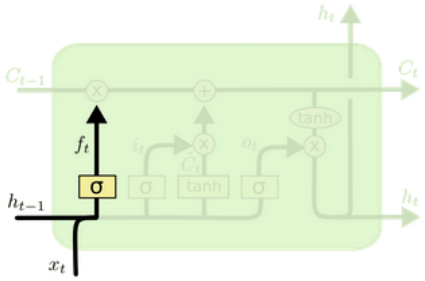
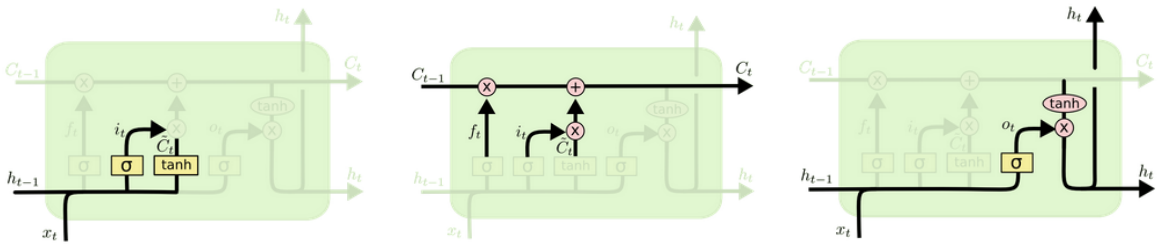


Figura 4-8: Etapa de la "Compuerta del olvido" [28]

Una siguiente etapa (figura 4-9 a)) la cual tiene el propósito de decidir qué información nueva se guardará, llamada "compuerta de la capa de entrada", con una capa σ que decidirá qué información se actualizará, y con otra capa *tanh* que crea valores candidatos que pudieran llegar a la siguiente iteración.

Estos resultados son multiplicados y sumados a la información que resulta de la etapa del olvido multiplicada por la información C_{t-1} de la iteración anterior (figura 4-9 b)). Por último una etapa que decide que información saldrá mediante una capa neuronal σ que decide que datos de la neurona sera producida a la salida (figura 4-9 c)). Este resultado será multiplicado con la nueva información C_t a través de una función *tanh* que pone la información en el rango de $(-1, 1)$. El resultado será la salida h_t que también será procesada en la siguiente iteración.



(a) Compuerta de la capa de entrada (b) Información C_t nueva (c)

Figura 4-9: Etapas siguientes de la estructura interna de la red LSTM [28]

Es de importancia detallar esta variante de red RNN debido a que como se menciona en el capítulo 1, el presente trabajo tiene como parte del objetivo comparar un sistema reconocedor semántico de acciones que implementa redes RNN [40] para el reconocimiento de acciones con un sistema que implemente Modelos Ocultos de Markov, por lo que es esencial introducir también las redes LSTM en este capitulo para poder dar un panorama general del otro sistema a comparar [40].

Capítulo 5

Metodología

En este capítulo se describe la manera en que se desarrolló este proyecto, las bases con las que ya se cuenta y se utilizó para dar continuidad al mismo. Se describe en un inicio el proyecto base con el cual se fundamenta este trabajo, así como los elementos que lo componen. Posteriormente se describe el módulo del trabajo presente que complementará lo anterior y la manera en que se añade y comparará con el trabajo base.

5.1. Sistema de categorización semántica de objetos

Como se ha mencionado desde el capítulo 1, parte del objetivo de este trabajo de tesis es dar continuidad a un sistema [40] que relaciona acciones realizadas por un humano así como la interacción con objetos al realizarlo, intentando darle un significado semántico y por tanto, poder efectuar una acción posterior o dar una posible respuesta consecuente (figura 5-1). El sistema antes mencionado es implementado mediante redes neuronales profundas (*DNN*, por sus siglas en inglés: *Deep Neural Network*) en general; concretamente para el módulo reconocedor de acciones se hace uso de redes recurrentes y convolucionales. Como también se mencionó anteriormente, se pretende comparar este módulo con uno nuevo que implemente modelos ocultos de Markov (*HMM*, por sus siglas en inglés:*Hidden Markov Models*) para reconocer acciones. Este nuevo módulo tendrá como objetivo reconocer la acción que se esté efectuando en un

momento dado de un video procesado por una red neuronal que detecta los esqueletos de la persona (OpenPose).

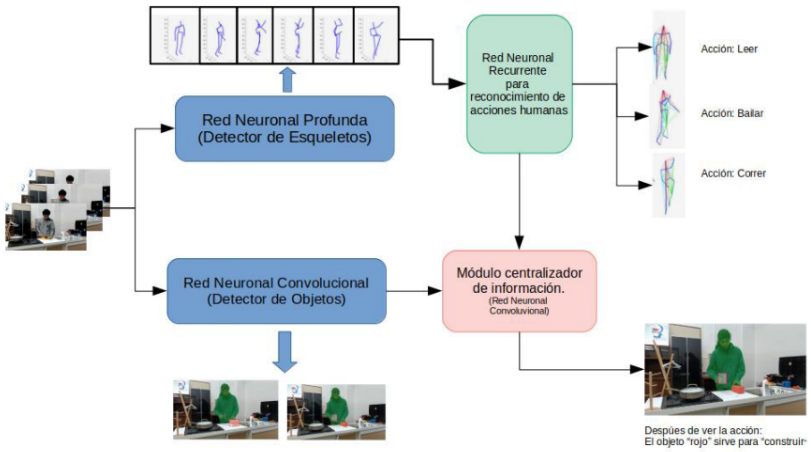


Figura 5-1: Sistema que implementa solo Redes Profundas, del cual se da continuidad [40]

5.1.1. Dataset

Para poder hacer una comparación válida entre el uso de redes neuronales y los HMM, se usará el mismo conjunto de datos creado y utilizado en [40]. Este conjunto de datos fue creado en un ambiente controlado, con el objetivo de poder realizar actividades de reconocimiento de acciones y que se pueda obtener información mediante software especializado en detectar esqueletos de personas y poder utilizar esta información posteriormente, por ejemplo, para entrenar una red neuronal o un sistema similar.

La estructura de este dataset es un conjunto de videos grabados en diversas duraciones, mediante 4 distintos ángulos y cámaras en las que un grupo de personas fueron grabadas realizando determinadas acciones de la manera más "natural" posible, sin un orden específico o duración de éstas específicas o iguales. Las 4 cámaras específicas de la figura 5-2 están detalladas y configuradas en [40]. Si bien es posible encontrarse realizando una persona una alguna acción distinta, en este dataset se podrán encontrar 12 acciones que fueron utilizadas en el sistema categorizador semántico [40]:

- Beber
- Cocinar
- Comer
- Construir
- Cortar
- Golpear
- Jugar
- Lavar
- Leer
- Limpiar
- Mezclar
- Verter

En el dataset también aparecen un conjunto de objetos específicos que como se detalla en [40], son parte de una serie de objetos que se pueden encontrar en el conjunto de datos *YCB* y que pueden ser relacionados también como objetos que se usan al realizar cierta acción humana.

(a) Con una cámara del robot *HSRB*

(b) Con una cámara *Kinect*

(c) Con una *Webcam*

(d) Con una cámara *Xtion*

Figura 5-2: Ejemplos de una acción realizada por una persona en distintos ángulos, así también algunos de los objetos que se detectarán.

5.2. Sistema reconocedor de acciones con HMM y CNN

El módulo propuesto estará integrado en el modelo general de [40], sustituyendo la red neuronal recurrente/convolucional que reconoce acciones, creando un segundo modelo general como se muestra en la figura 5-3. Éste estará compuesto de manera similar que en la figura 5-1

por los mismos otros módulos o bloques para detectar esqueletos, objetos y para centralizar los dos anteriores.

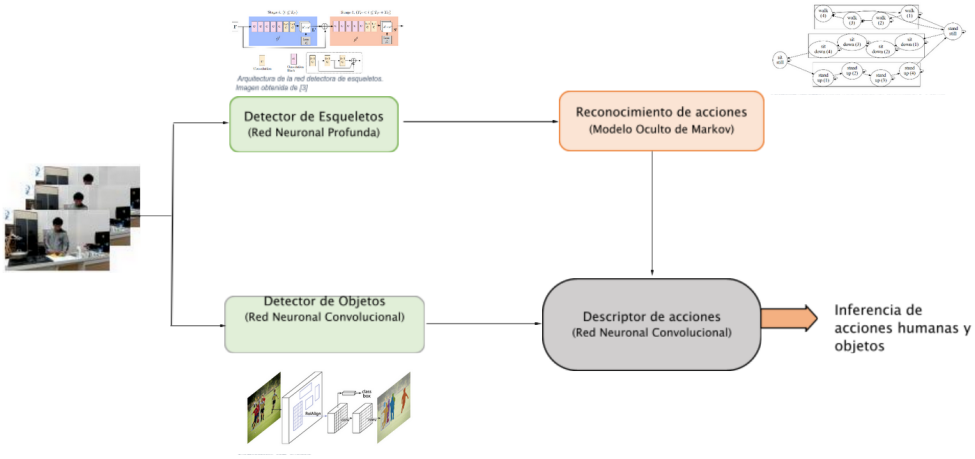


Figura 5-3: Sistema propuesto para este proyecto.

De manera equivalente se pretende utilizar la mayoría de los bloques que conforman el sistema anterior y actual, con un esperado cambio para adaptar los datos que ingresan al módulo con HMM y la información que arroja y será recolectado en el módulo centralizador posteriormente. Se describen a continuación cada uno de estos bloques, donde en general tiene un funcionamiento similar con [40], y se introduce con mayor detalle el nuevo módulo que utiliza HMM.

5.2.1. Detector de esqueletos

Para poder detectar los esqueletos de una persona, se utilizará un sistema que implementa redes neuronales, detallada en la sección 2.3.1. Este sistema (o software) permite procesar video, o en su defecto una secuencia de imágenes, y obtener coordenadas espaciales en la imagen donde se ubiquen extremidades específicas del cuerpo humano en una persona detectada.

Cada video se puede ver como una secuencia de N imágenes, la cual estará determinada por la longitud del video así como los cuadros por segundo (FPS, por sus siglas en ingles) en la que estuvo configurada la cámara que lo grabó. En un video "común", un video puede estar grabado a $30 - FPS$, lo cual quiere decir que por cada segundo de grabación, existe una secuencia de

30 cuadros o imágenes, si un video tuvo una duración de 10 segundos, existirán en total una secuencia de 300 imágenes. De [40] podemos ver que los videos se grabaron a $30FPS$, aunque a distintas duraciones cada uno de los videos claramente.

Para este proyecto se trabajó por separado la parte de la obtención de esqueletos, con el fin de obtener un "conjunto de datos" equivalente con únicamente la información que se obtiene al procesar un video con *OpenPose* y así poder *entrenar* el HMM correspondiente para detectar esqueletos. Este nuevo dataset de igual manera se etiquetó de una manera específica para poder saber a qué archivo de video corresponde del dataset original. Cada video se procesó por completo y se "extrajo" todos los vectores de esqueletos posibles guardándolos en un solo archivo por separado, por lo que el nuevo dataset contiene el mismo número de archivos que de videos del dataset anterior. Lo anterior se decidió para no tener que procesar múltiples veces un video con el software, ya que con videos de suficiente duración el procesamiento puede ser relativamente "lento".

Una vez obtenido el nuevo dataset, se crea un dataset secundario separando las acciones que aparecen en el video. Dado que el video se puede ver como una secuencia de N imágenes, utilizando programación por computadora, es posible obtener "manualmente" el inicio de una acción en un cuadro i y el fin de esa acción en un cuadro j del video ($1 \leq i < j \leq N$). Por tanto, con el apoyo del etiquetado del primer conjunto de datos de videos, se obtienen todos los $i's$ y $j's$ de cada acción en cada video, y como el nuevo dataset corresponde únicamente al conjunto de vectores de un esqueleto en una imagen, se separan desde el i-ésimo conjunto hasta el j-ésimo conjunto de vectores, y se guardan en un nuevo archivo etiquetado propiamente para distinguir la acción correspondiente. Con estos nuevos datasets se procesará y entrenará el HMM que se describirá en una sección posterior.

Dado que esta implementación solo será para el entrenamiento, también es necesario mencionar que esta metodología será distinta para cuando se aborde un problema en "tiempo real", ya que en una misma implementación se obtendrá con OpenPose la información de los esqueletos de cada cuadro de imagen y se procesará de manera "inmediata" con el **HMM entrenado** para reconocer la acción que se esté efectuando.

La manera en que se etiquetaron los nuevos dataset se detalla en el capítulo 7.

5.2.2. Detector de Objetos

El sistema que reconocerá objetos será en un principio, el mismo utilizado en [40], por lo cual no se pretende describir detalladamente en esta sección. El funcionamiento general de este sistema es mediante una red convolucional entrenada que procesará cuadros de imágenes de un video y segmentará el objeto que reconoció mediante un rectángulo de un color sobrepuesto en la imagen, el cual mediante un texto también sobrepuesto indicará el nombre que corresponde con el objeto detectado. El proceso anterior también se mencionó en la sección 2.3 y con más profundidad desde el trabajo realizado en [40].

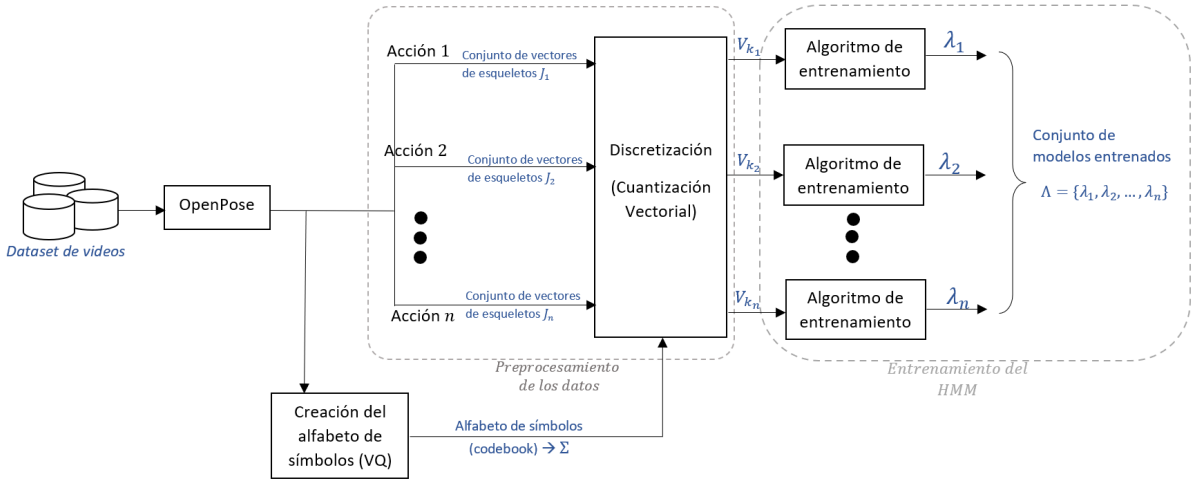
Esta red convolucional fue previamente entrenada con un conjunto de datos de imágenes inspirado en el trabajo realizado por las universidades de Yale, Carnegie Mellon University (CMU) y Berkeley, llamando a su conjunto de datos *YCB* por las iniciales de cada una de estas escuelas. Debido a que no todos los objetos aparecen en este conjunto de videos de este trabajo, se realizó desde [40] un subconjunto de objetos que sí aparecerían en el conjunto de datos creado por el autor en [40], por lo que para este trabajo, tratando de asimilar el mismo formato, se crearía nuevamente un subconjunto para el proceso de entrenamiento de la red convolucional detectora y segmentadora de objetos.

5.2.3. Módulo reconocedor de acciones con HMM

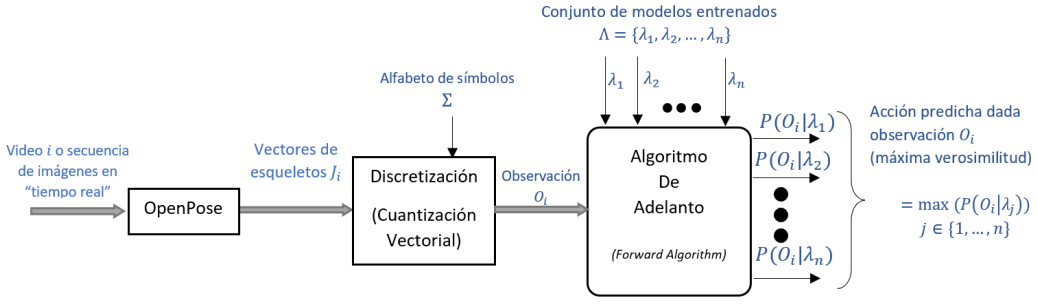
Para poder reconocer acciones, se utilizarán los datos obtenidos y tratados del detector de esqueletos mencionados anteriormente con el objetivo de adaptarlos al tipo de información que un modelo oculto de Markov maneja. En la figura 5-4 (a) para el entrenamiento y b) para pruebas e inferencia) se representa de manera general el proceso que se llevará a cabo en este módulo.

Se puede observar en la parte izquierda de la imagen a) mencionada la primera sección con el uso del detector de esqueletos mediante el sistema *OpenPose*, donde se resalta la creación de un nuevo conjunto de datos con los datos resultantes del programa.

Como se ha mencionado en múltiples ocasiones, los HMM no pueden "manejar" cierto tipo de datos, como en este caso podrían ser los vectores que resultan de utilizar el software antes mencionado, por lo que es necesario una etapa de pre-procesamiento previa al del HMM, el cual se discretizará la información del conjunto de datos mediante algoritmos de cuantización vectorial, el cual se describirá a continuación, y nos permitirá obtener un alfabeto de símbolos (discreto) los cuales permitirá *discretizar* la información de los esqueletos y éstos puedan ser procesados en un modelo propuesto de un HMM.



(a) Diagrama de flujo para el entrenamiento



(b) Diagrama de flujo para la inferencia

Figura 5-4: Diagrama de flujo para el HMM propuesto. Se separa el flujo para el entrenamiento a) y para la inferencia b)

Cuantización Vectorial

La acción de cuantizar se puede referir en general a establecer un margen de algún valor para una señal de entrada a un solo valor o nivel de salida. Para tareas de digitalización, la cuantización permite medir niveles de voltaje a cada muestra de entrada y establecer un nivel para la salida; en matemáticas se utiliza para convertir conjuntos de variables continuas a conjuntos restringidos discretos.

La Cuantificación o *cuantización* vectorial es una técnica de cuantización de procesamiento de señales, siendo una generalización de la cuantización escalar en la que se utiliza todo un vector y no un solo escalar para cuantizarlo. Diseñados en un inicio con el objetivo de la compresión de los datos, es decir, reducir el rango de bits para lograr minimizar la capacidad de los canales de comunicación o minimizar los requerimientos de las memorias de almacenamiento digital, manteniendo la fidelidad necesaria de los datos [13].

Esta técnica funciona dividiendo o agrupando en distintos grupos de vectores que contengan una cercanía entre ellos, mediante el uso de centroides. En cada grupo estará representado por uno de ellos (centroide) y contendrá aquellos vectores que tengan una mayor cercanía a él.

Existen múltiples algoritmos que implementan una cuantización vectorial, por ejemplo uno muy usado para diversas tareas es el algoritmo *K-means* o *K-medias* en español, propuesto primeramente por S. Lloyd en 1957 y publicado en 1982 [23], donde hasta la fecha se encuentran variaciones del algoritmo como por ejemplo el algoritmo *Mini Batch K-means*. Otra técnica de cuantización, muy parecido al anterior y el utilizado para este proyecto, es el algoritmo *LBG* (Linde-Buzo-Gray, por las iniciales de los autores), visto como el algoritmo generalizado de Lloyd, es un método iterativo que calcula primeramente un primer *codebook*, con un centroide equivalente al promedio de todos los vectores que se cuantizarán. Posteriormente se divide este creando dos centroides con una perturbación mínima entre ellos, es decir una modificación a los valores del primer centroide. Iterativamente se asocian los vectores a los centroides más cercanos, se recalculan los datos de los centroides con el promedio de los vectores pertenecientes a cada uno de ellos, si el cambio de estos centroides no es menor a un valor cercano a cero, se repite el paso anterior de asociar vectores a los centroides. Si el cambio fue mínimo, se proceden a

dividir los dos centroides de manera similar y se repiten los pasos anteriores de asociar vectores a centroides, recalcular la posición de cada centroide con el promedio de sus vectores asociados hasta que estas variaciones sean mínimas. Lo anterior se puede ver en la siguiente tabla de algoritmo:

Algorithm 1 Algoritmo LBG [37]

Require: Set of vectors $X = \{\overrightarrow{X}_1, \overrightarrow{X}_2, ..., \overrightarrow{X}_M\}, \quad \overrightarrow{X}_i = [x_{i,1}, x_{i,2}, ..., x_{i,N}]$

1: $\overrightarrow{C}_i = \frac{1}{M} \sum_{j=1}^{M} \overrightarrow{X}_j$
2: **while** $|D_g^t - D_g^{t-1}| > \epsilon$ **do**
3: $\quad \overrightarrow{C}_{i+1} = \overrightarrow{C}_i(1+\epsilon)$
4: $\quad \overrightarrow{C}_{i+2} = \overrightarrow{C}_i(1-\epsilon)$
5: $\quad$ **for** $1 \leq j \leq M$ **do**
6: $\quad\quad d_1 = distance(\overrightarrow{x}_j, \overrightarrow{C}_{i+1})$
7: $\quad\quad d_2 = distance(\overrightarrow{x}_j, \overrightarrow{C}_{i+2})$
8: $\quad\quad$ ▷ Cluster vectors $\overrightarrow{X}_i$ to a region G according to nearest centroid $\overrightarrow{C}_k$
9: $\quad\quad \overrightarrow{X}_i \rightarrow G_{i+1}$ **if** $d_1 < d_2$, **else** $\overrightarrow{X}_i \rightarrow G_{i+2}$
10: $\quad\quad D_g^t = D_g^t + min(d_1, d_2)$ ▷ global distance D_g^t
11: $\quad$ **end for**
12: $\quad \overrightarrow{C}_{i+1} = \frac{1}{\text{num. of vectors of } \overrightarrow{G}_{i+1}} \sum_{j=1}^{\text{No. of vector of } \overrightarrow{G}_{i+1}} \overrightarrow{X}_j$
13: $\quad \overrightarrow{C}_{i+2} = \frac{1}{\text{num. of vectors of } \overrightarrow{G}_{i+2}} \sum_{j=1}^{\text{No. of vector of } \overrightarrow{G}_{i+2}} \overrightarrow{X}_j$
14: **end while**

Este algoritmo se implementa hasta obtener un número N de centroides, siendo $N = \{1, 2, 4, 8, ..., 2^i\}$. Posteriormente este conjunto de centroides permitirá discretizar los datos de los vectores, de manera que este conjunto de N centroides se etiquetará con un valor discreto del 0 al $N-1$. Con este conjunto de etiquetas se crea el alfabeto de símbolos que permitirá discretizar los vectores que se procesen al asignarles un valor del alfabeto comparando cada vector con todos los centroides mediante una distancia euclidiana y aquel con la menor de ellas será la más *parecida* y por tanto se asignará la etiqueta del centroide al vector. Lo anterior se hace con todos los vectores del conjunto de datos por lo que todos se etiquetarán con algún símbolo del alfabeto.

Entrenamiento del modelo

Con los datos discretizados se procederá a entrenar un modelo λ por cada clase del conjunto de datos, es decir, por cada tipo de acción presente en los videos. Como se describió en el Capítulo 3 y de acuerdo con [33], se puede entrenar un modelo mediante el algoritmo de Baum-Welch para obtener las matrices A y B y el vector π.

Dado que se obtendrán los modelos por cada acción, se separan los vectores correspondientes a cada uno, como en la figura 5-4 a), obteniendo conjuntos V_{k_i} correspondientes a los símbolos de cada vector, y conjuntos Y_i que corresponderán a los estados del modelo de Markov. Este último conjunto se define manualmente al establecer la cantidad de estados $M = \{1, 2, 3, ...\}$ que tendrá el modelo. Con estos dos conjuntos trabajará el algoritmo de Baum-Welch para obtener la matriz A_i de acuerdo a las ecuaciones 3-29 y 3-30, la matriz B_i de acuerdo con las ecuaciones 3-31 y 3-32 y por último el vector π_i de acuerdo con la ec. 3-28 y así crear un modelo $\lambda_i = (A_i, B_i, \pi_i), \quad 1 \leq i \leq N$, como se define en el Capítulo 3.

Es de mencionar que este entrenamiento se hace con distintas variaciones en algunos *hiperparámetros* (no. de estados y de centroides, estructura del modelo y de los datos de entrenamiento) para poder optimizarlo y tener un mejor modelo. Lo anterior será detallado posteriormente en el Capítulo 6 y 7.

Reconocedor de acciones con el modelo

Con la ayuda de los λ_N modelos se empleará el algoritmo de Adelanto (*Forward Algorithm*) descrito en el Capítulo 3, para poder obtener la probabilidad de una secuencia O_i dada el modelo, como en la figura 5-4 b). Esta secuencia O_i vista en la figura anterior (secuencia en color azul), será igualmente un video que fue procesado por el software *OpenPose* y discretizado posteriormente con ayuda del alfabeto de símbolos. Con ayuda del algoritmo de Adelanto, se obtendrá una probabilidad, o verosimilitud, por cada modelo $P(O_i|\lambda_j), \quad 1 \leq j \leq N$ y será el máximo de estas probabilidades, o dicho de una manera más correcta, la máxima verosimilitud del modelo que mejor describirá la secuencia y por tanto implicará que la secuencia (el video) será reconocida y clasificada a una acción.

5.2.4. Descriptor de acciones

De manera similar que en el trabajo de [40], se pretende utilizar un módulo concentrador que reciba información de dos procesos anteriores e independientes y poder hacer la comparación entre ambos trabajos. Este módulo tendrá como objetivo recibir información del modelo reconocedor de objetos y del módulo reconocedor de acciones con HMM, procesarlos juntos y con su propio entrenamiento, realizar una inferencia entre el objeto y la acción recibida, como por ejemplo que en efecto sí es acorde (o no) la acción realizada con el objeto detectado, entre otras.

De acuerdo con el trabajo presentado en [40], este descriptor de acciones se implementó de igual manera con una red neuronal profunda, recibiendo la información de salida de la red neuronal detectora de objetos, es decir un vector de tamaño especifico o "encoding" con las probabilidades más altas calculadas de los objetos. Por otro lado recibe en [40] un vector de tamaño especifico de la información de la red recurrente descriptora de acciones, por lo que para este trabajo se adaptará el resultado del módulo HMM reconocedor de acciones para mandar la información de salida calculada con los modelos ocultos de Markov de manera equivalente a la información que arroja una red neuronal profunda.

La red se entrenará por separado con información resultante guardada de ambos módulos anteriormente mencionados para poder guardar los datos de éste (sus pesos) y posteriormente utilizarlos en una implementación en la que sea necesario utilizar el sistema completo en una tarea real.

Los detalles de esta red se describirán en el capítulo 6.

Capítulo 6

Implementación

6.1. Dataset

6.1.1. Etiquetado de acciones

Como se mencionó en el capítulo anterior, el conjunto de datos para este proyecto consta de videos grabados en 4 perspectivas y duraciones distintas, el cual consiste en 12 acciones establecidas, [40] realizadas por distintas personas y en un ambiente controlado.

Este *dataset*, creado en [40] fue proporcionado para este proyecto sin el etiquetado original, por lo cual, con un poco de apoyo y asesoramiento del mismo autor, se volvió a etiquetar este conjunto de datos. La manera en la que se etiquetan las acciones en los videos se realizó mediante una tabla de datos (archivo en formato.*csv*). Dado que este conjunto está separado en carpetas, primero por el tipo de cámara utilizada y en cada una con el usuario que fue grabado, cada uno con una cantidad no necesariamente igual de archivos de video, se anotó en una tabla de datos (por cada tipo de cámara) en cada columna el nombre del usuario, nombre del video, la acción percibida, el número del cuadro que empieza la acción (dado que se puede ver un video como una secuencia de cuadros o imágenes), el término de la misma, el tamaño de la acción y por último el tamaño total del video. Una parte de este etiquetado se muestra en la figura 6-1. Para poder obtener exactamente el cuadro de inicio y fin de una acción se creó un script en lenguaje *python* y utilizando librerías de manejo de videos e imágenes como *OpenCV*, se obtiene cuadro

por cuadro de un video y manualmente se anotó donde se consideró que empezó y terminó la acción.

1	User	Video	Action_in_video	Frame_init	Frame_end	Frames	Total_video_frame
2	user_1	action1	Construir	2	38	36	46
3	user_1	action2	Construir	3	58	55	120
4	user_1	action2	Jugar	48	66	18	
5	user_1	action3	Cocinar	4	19	15	78
6	user_1	action3	Mezclar	20	27	7	
7	user_1	action3	Verter	28	36	8	
8	user_1	action3	Comer	41	48	7	
9	user_1	action3	Lavar	49	77	28	
10	user_2	action1	Cocinar	6	33	27	40
11	user_2	action2	N	0	0	0	28
12	user_2	action3	Lavar	2	38	36	41
13	user_2	action4	N	0	0	0	20

Figura 6-1: Parte del etiquetado de acciones.

6.1.2. Etiquetado de objetos

Con lo mencionado a lo largo de este capítulo y el anterior, se trató de etiquetar de manera equivalente como en [40] para poder utilizarlo en un módulo reconocedor de objetos. Debido que no es el desarrollo principal de este trabajo, no se pretende describir a fondo el desarrollo de este apartado. De igual manera a como se redactará en 7, el etiquetado de objetos pasó a segundo término y por tanto no fue utilizado de manera importante.

6.1.3. Información adicional

De [40] se pueden obtener datos estadísticos de este conjunto de datos, por ejemplo la cantidad de imágenes en las que aparecen objetos y en las que está realizándose una acción son $410,896$ y $1,432,688$ respectivamente, aunque es de mencionar que debido a que este conjunto de datos no fue trabajado en este proyecto con el etiquetado original, estos números variaron considerablemente a pesar del asesoramiento del autor original para el nuevo etiquetado.

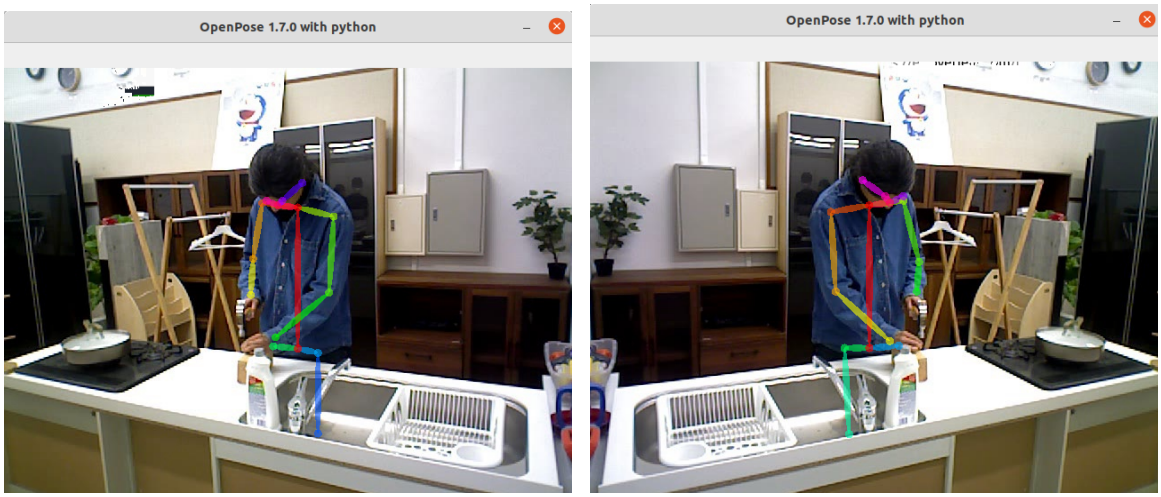
6.2. OpenPose para secuencia de movimientos

Utilizando los archivos de etiquetado para las acciones en los videos se adaptó un script en lenguaje *python* para utilizar OpenPose el cual puede procesar un video o imagen y obtener los vectores del esqueleto de una persona en ella. Como un video es una secuencia de imágenes, el programa procesa realmente cuadro a cuadro de un video y obtiene los vectores de cada uno. Este programa OpenPose obtiene en un inicio los vectores en manera de una cadena de texto, con un formato especifico donde se pueden separar las distintas personas detectadas, si es el caso, y por cada persona las n coordenadas (x, y) de cada extremidad o *joint*. Con la cadena obtenida anteriormente, el script separa el texto con cada lista de extremidades de la/s persona/s, y con ayuda de librerías para manejo de vectores y matrices, entre otros (*numpy*) se guarda esta lista a manera de matriz de tamaño $n \times 2 \times L$ en formato *.npy*, donde n es el número de *joints* que OpenPose detecta, 2 debido a las coordenadas x y y de cada extremidad y L por la cantidad de cuadros en cada video. Por lo tanto se crearán archivos *.npy* por cada video del conjunto de datos y el nombre de cada uno con un formato especifico (*<camara>_<usuario>_<nombre del video>.npy*).

6.2.1. Aumentado de la información

Adicional a lo anterior, se decidió hacer pruebas implementando y agregando un aumentado de datos al crear un espejo horizontal a cada cuadro del video (figura 6-2) por lo que al nombre de cada archivo se agregó una etiqueta en caso que fue el el aumentado (*<augmented>*), por lo que el nombre del archivo si es con el espejo horizontal quedaría de la forma: (*<camara>_<usuario>_<nombre del video>_<augmented>.npy*). Con lo anterior se creó el nuevo dataset de archivos que contienen los vectores de cada cuadro de cada video, como se mencionó en la sección 5.2.1.

Con lo anterior, sin contar el aumentado de datos, se muestra en la figura 6-3 la cantidad de secuencias por acción, es decir cantidad de archivos guardados *.npy* por cada acción detectada en una parte del video, sin contar el tamaño de cada uno de éstos que implica la duración de



(a) Obtención del esqueleto en una imagen (b) Obtención del esqueleto con un espejo horizontal

Figura 6-2: Ejemplo de crear un espejo en un horizontal al cual también se obtiene el esqueleto de la persona detectada.

cada acción detectada.

6.3. Clasificador de acciones

6.3.1. Pre-procesamiento

Antes de poder procesar los vectores se implementó un pequeño procesado previo para *limpiar* un poco los archivos *.npy* de cada acción. Lo que se hizo fue verificar y borrar vectores que contengan valores cero únicamente, ya que *OpenPose* pudo no detectar una persona en un cuadro del video. Lo anterior se notó ya que al procesar el video por el programa, en mínimas ocasiones no se obtenían los puntos de las extremidades en ciertas imágenes, por lo que *OpenPose* al no detectar y reconocer un esqueleto de la persona, da como resultado una matriz de ceros de tamaño $n \times 2$.

Para poder eliminar estos valores cero, con el objetivo de quitar "ruido" al algoritmo de cuantización y los demás utilizados posteriormente, en cada script que se programó, detectará (o no) en el archivo *.npy* de tamaño variable $[n \times 2 \times M]$, M : tamaño de la secuencia una matriz $n \times 2$ donde todos los valores sean ceros y lo eliminará, resultando en dado caso una

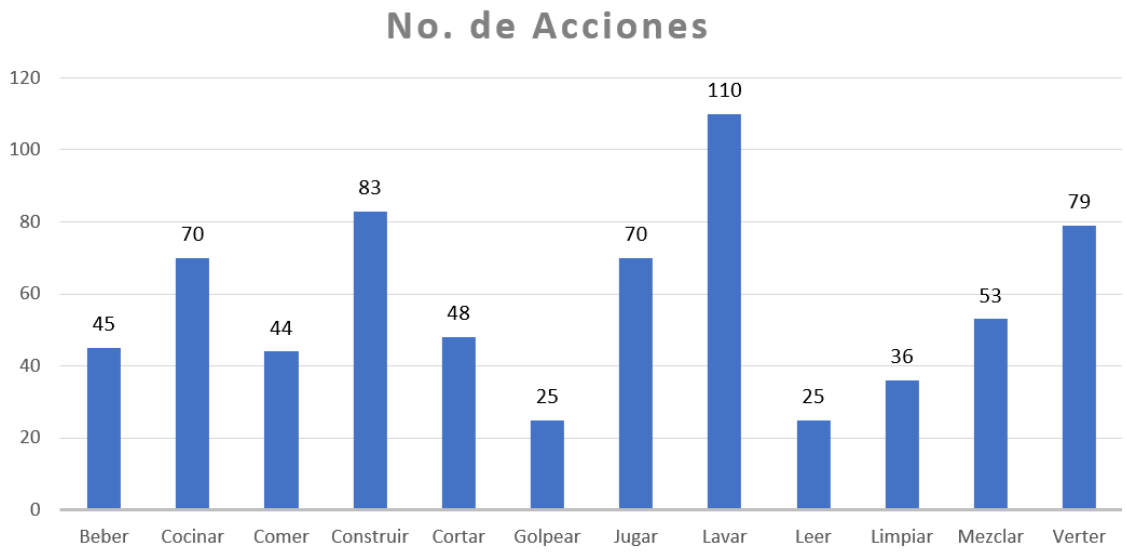


Figura 6-3: Gráfico que muestra la proporción de acciones en el nuevo dataset procesado previamente por OpenPose

nueva matriz de tamaño $[n \times 2 \times (M - i)]$, $\quad i$: cantidad de matrices cero detectadas.

Para poder optimizar el modelo oculto de Markov (HMM), se hicieron pruebas con acciones de un rango limitado de longitud, ya que como se puede apreciar en la figura 6-4, no todas las acciones tienen la misma longitud y existen algunas que son considerablemente más grandes, como se puede apreciar en la figura, la mayor parte de las acciones tiene un tamaño menor a 200 pero existen algunas que alcanzan un tamaño de más de 1000. En el capítulo 7 se detalla las variaciones que este aproximamiento obtuvo.

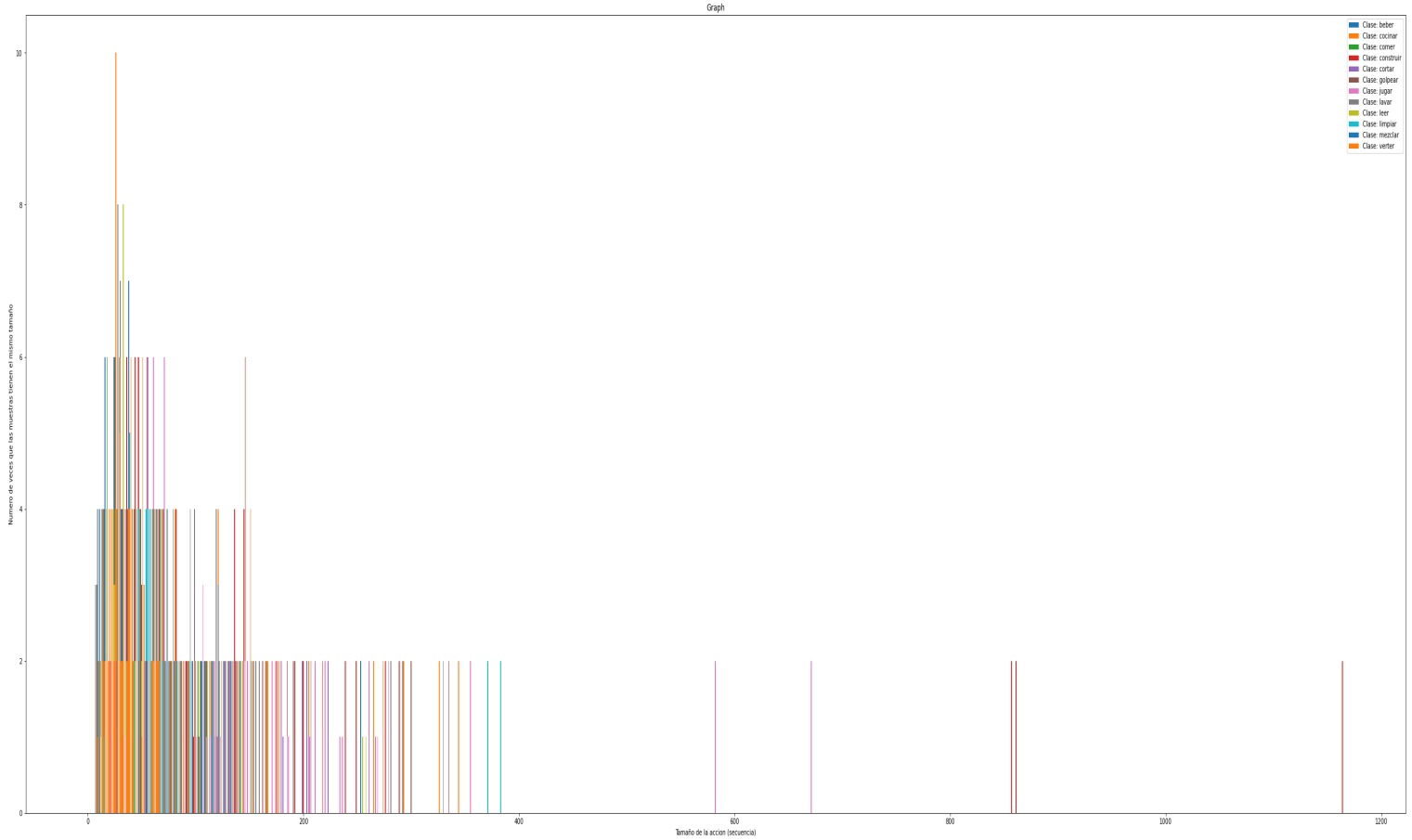


Figura 6-4: Gráfico que muestra la disparidad del tamaño/duración de acciones en general (horizontal) con respecto al número de acciones (vertical)

6.3.2. Cuantizador Vectorial

Para la parte de obtener un alfabeto de símbolos con el *codebook* resultante de emplear un algoritmo de cuantización vectorial se implementaron los algoritmos *LBG*, *K-medias* que igualmente se emplea en el estado del arte y con el objetivo de comparar posteriormente el desempeño del modelo utilizando diversos algoritmos de cuantización.

Para poder hacer una comparación válida, se crearon por cada algoritmo antes mencionado *codebooks* de distinto número de centroides. Como el primer algoritmo obtiene un número de centroides correspondientes al número 2 elevado a un exponente (2^i), para *K-medias* también se programó para obtener los mismos números de centroides, ya que éste puede ser un poco más flexible al poder obtener otra cantidad de centroides. Para cada uno de estos algoritmos, se crearon *codebooks* de tamaños N_i, $N_i = \{2, 4, 8, 16, ..., 512\}$ con el objetivo de verificar sus desempeños utilizando un HMM; las comparaciones entre todos estos de detallan en el Capítulo 7.

6.3.3. Modelo Oculto de Markov

Una vez obtenido un *codebook* de tamaño N, se discretizarán los vectores de las acciones comparando cada uno con los centroides del *codebook* y se asignará un valor discreto $[0-(N-1)]$ que corresponda al centroide más parecido.

De manera simultánea se crea un vector de estados del mismo tamaño por cada uno de los vectores de la acción. El número de estados es establecido manualmente y para poder optimizar el modelo, se hicieron varias pruebas variando la cantidad de estados en el modelo, detallado en el Capítulo 7.

Como se vio en la figura 5-4 a), el proceso de discretizar implica que todo el conjunto de datos se separó en una primera instancia para poder crear distintos modelos λ_i, por lo que para cada subconjunto de datos por tipo de acción, se discretizarán las acciones y se obtienen el vector de símbolos V_{k_i} y el vector de estados Y_i que se procesan en un script que implementa el algoritmo de Baum-Welch para poder obtener el modelo general $\lambda_i = (A_i, B_i, \pi_i)$. Este script programado en lenguaje *python*, crea un objeto el cual tendrá como atributos matrices “vacías”

correspondientes a A_i,B_i, y π_i, y mediante un método propio para ejecutar el algoritmo anterior, actualiza las matrices anteriores y genera un modelo final con los datos nuevos. Lo anterior es una pequeña variación que está inspirada en el trabajo de Rabiner [33], pero también se hizo un script para poder utilizar el entrenamiento mencionado en ese trabajo, en el que se diferencia de lo anterior mencionado al iniciar creando las matrices A, B y el vector π de manera casi aleatoria, ya que se necesita hacer cumplir que la suma de las filas en la matriz A sea 1 ([33],[31]), y dado que la matriz A describe la topología del modelo, se requiere que tengan valores en algunas filas y columnas para que tenga una estructura especifica el modelo, entre otras propiedades. Una vez creado lo anterior se implementa el algoritmo de Adelanto para obtener una matriz alpha α y el algoritmo de retroceso para obtener la matriz beta β, con los que serán necesarios para obtener la reestimación de los parámetros del HMM [33], calculando una nueva matriz ξ de la siguiente manera:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1}\beta_{t+1}(j)}{P(O|\lambda)} \tag{6-1}$$

Es decir, la probabilidad de estar en el estado S_i en el tiempo t y en el estado S_j en el tiempo $t+1$ dado el modelo y la observación. Este proceso iterativo permitirá recalcular las matrices A y B de la siguiente manera, si $\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$, entonces:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{6-2}$$

y para la matriz B

$$\bar{b}_j(k) = \frac{\sum_{t=1,s.t.O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{6-3}$$

Obteniendo nuevas matrices $\bar{A}$ y $\bar{B}$ que iterativamente se reestimarán con nuevas obser-

vaciones. Las matrices resultantes por cada modelo serán las utilizadas para posteriormente evaluar su desempeño. Para poder tener una calificación del desempeño de estos modelos, se guardarán los resultados de implementar el algoritmo de Adelanto y un modelo λ_i con todos los datos de entrenamiento por un lado y con los datos de prueba por otro. Por lo tanto, por cada modelo λ_i se tendrá un registro de cuál fue la acción inferida mediante el algoritmo utilizando un vector, por lo tanto se podrá construir con el conjunto de vectores una matriz de registros, o también llamada matriz de confusión donde se visualiza las inferencias que hicieron los modelos con todos los datos. Estos resultados se detallan en el Capítulo 7.

Para la parte de entrenamiento hizo una separación de los datos $80 - 20$, donde el $80\,\%$ se utilizó para entrenar el modelo y el restante para validarlo, intentando realizar las técnicas similar a las utilizadas para validar modelos en aprendizaje profundo y Redes Neuronales.

Estos objetos serán guardados una vez se tenga el mejor modelo optimizado para poder utilizarlo en una situación real, representado en la figura 5-4 a), mediante el algoritmo de Adelanto. Para tener el modelo mejor optimizado se implementó de igual manera el algoritmo de Adelanto con los datos de entrenamiento y prueba, donde los resultados se muestran igualmente en el Capítulo 7.

Capítulo 7

Pruebas y Resultados

En este capítulo se visualiza y describe todo lo relacionado a resultados para el proyecto propuesto, en particular en el módulo clasificador de acciones utilizando modelos ocultos de Markov. Como se llegará a mencionar en las siguientes secciones, esta propuesta de trabajo, utilizando parte de un desarrollo previo [40], da resultados distintos a los que se tenían previsto en un principio y los cuales se establecieron en parte de los objetivos de este proyecto. Debido a lo anterior, se dará una explicación con evidencia y resultados suficientes.

7.1. Clasificador de Acciones

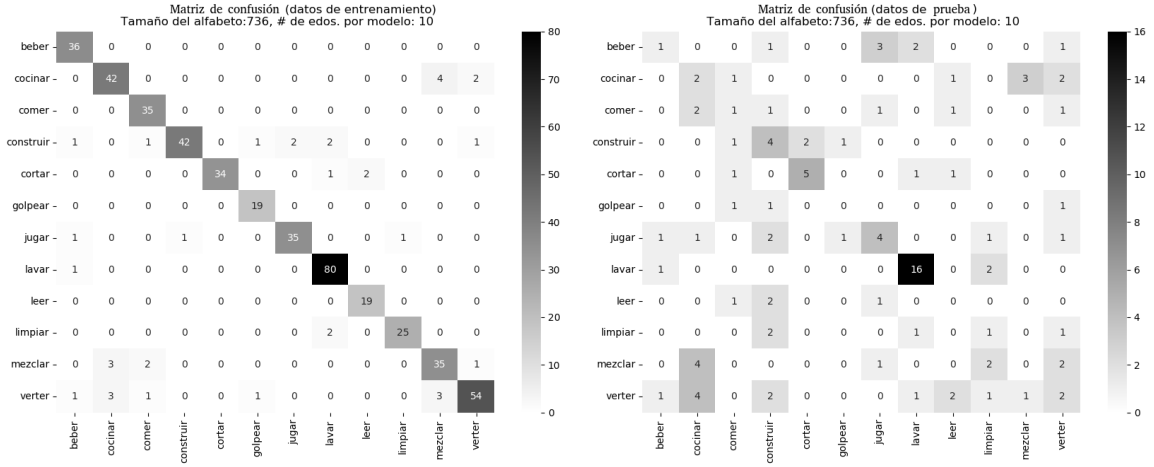
En la sección anterior se dio una descripción de la manera en que este módulo funcionará, al recibir información de un sistema que procesará una secuencia de imágenes (video) y detectará y obtendrá el "esqueleto" de una persona en una imagen, guardándolos mediante vectores en distintos archivos para su posterior uso. Por lo tanto, para poder entrenar un clasificador de acciones utilizando modelos ocultos de Markov (HMM), estos vectores de esqueletos requieren otro procesamiento para poder discretizarlos mediante técnicas de cuantización vectorial (VQ). Cada vector de esqueletos representará a 1 símbolo del alfabeto de símbolos establecido (centroides o *codebook* obtenido de utilizar VQ) por lo que por cada secuencia, se obtendrá una serie de símbolos. Con estas secuencias, a manera de vectores de símbolos, por cada clase

i se entrenará un modelo oculto de Markov λ_i mediante algoritmos de entrenamiento de este modelo, como lo son *Baum-Welch* y *Viterbi*.

7.1.1. Entrenamiento de los modelos con una primera metodología

Para poder tener el mejor modelo que pueda describir la acción a realizar es necesario hacer pruebas variando algunos hiperparámetros del modelo, como lo es el tamaño del alfabeto de símbolos y el número de estados por cada modelo, así se escogerá el modelo con el mejor desempeño de acuerdo con estas variaciones. Intentando emular la manera de calificar el desempeño de sistemas que requieren un entrenamiento previo, como lo son las redes neuronales, se separan aleatoriamente un conjunto de datos de vectores para utilizarlos al entrenar y otro conjunto para validar y hacer pruebas con los modelos ya entrenados. En el estado del arte se suele establecer un porcentaje de $70 - 80\,\%$ de datos para entrenamiento y el restante para validación o pruebas por lo que se planteó el mismo procedimiento.

Se muestra en la figura 7-2 los resultados de entrenar y evaluar con los datos de entrenamiento con la primera aproximación descrita en la sección 6.3.3 del capítulo anterior. Dado que existen secuencias muy cortas en el conjunto de datos, no es posible tener modelos que contenga un número muy grande de estados, por lo que se decidió hacer la evaluación hasta 30 estados como máximo, a pesar de tener algunas acciones que tengan menor número de secuencias a comparación.



(a) Con datos de entrenamiento

(b) Con datos de prueba

Figura 7-1: Ejemplo de la matriz de confusión de los resultados de la inferencia.

Como se mencionó en el capítulo anterior, mediante el uso de matrices de confusión (figura 7-1) se hace una evaluación del desempeño de los modelos. Dado que los resultados en la diagonal de esta matriz indican que la inferencia fue correcta, para poder tener una calificación del desempeño, la suma de los elementos de la diagonal entre el total de la suma de la matriz de confusión (A) indicará la precisión, es decir

$$\text{precisión} = \frac{\sum_{i}^{n} A_{ii}}{\sum_{i}^{n}\sum_{j}^{m} Aij} \tag{7-1}$$

Este valor de precisión también se guardará para los valores establecidos en los hiperparámetros mencionados anteriormente, por lo que también se hará un registro de precisiones por cada hiperparámetro variado. De manera más concreta, se estableció un número fijo de alfabeto de símbolos y se hizo variar el número de estados del HMM; desde 2 estados hasta un número mayor, como se ve en la figura 7-2, para poder obtener una gráfica de precisiones y poder así visualizar el desempeño para una prueba en específico. Adicional a esto, de manera como se suelen evaluar el entrenamiento en algunas pruebas del estado del arte, se muestra el desempeño para la precisión si se toma en cuenta que la clase verdadera se encuentra entre las mejores 2

verosimilitudes máximas.

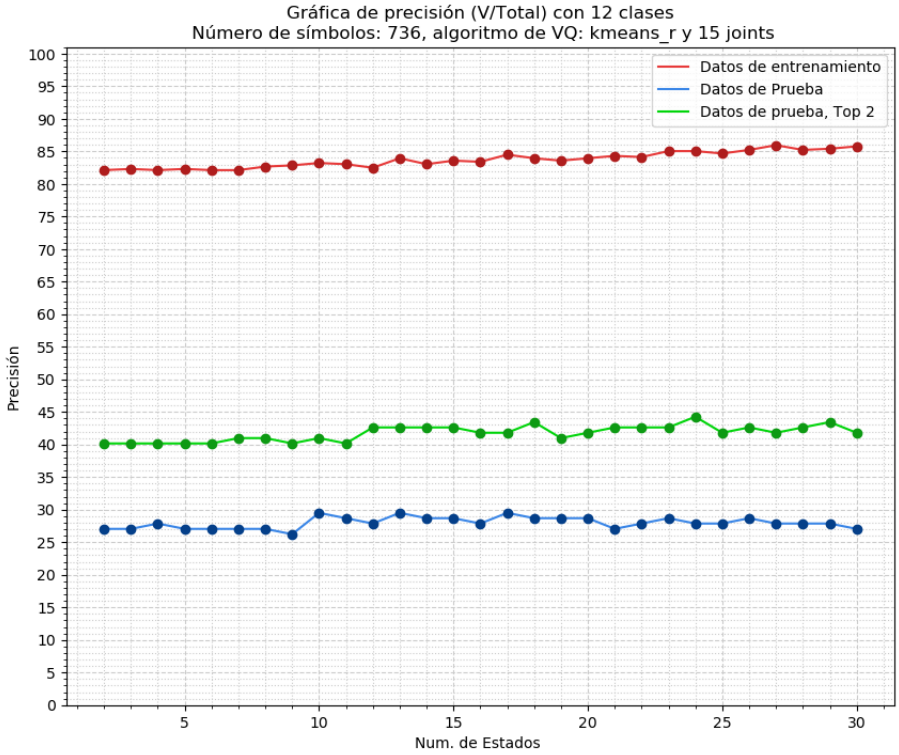
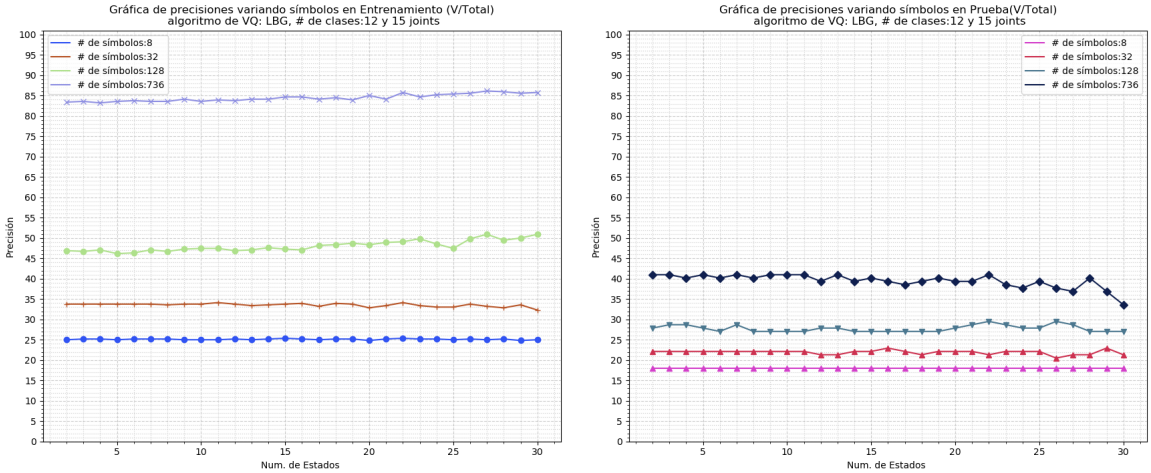


Figura 7-2: Gráfica de precisiones para una prueba, estableciendo un número fijo de símbolos, variando el número de estados (eje X) y la precisión correspondiente (eje Y).

También se muestra en la figura 7-3 la comparativa entre el entrenamiento para distintos tamaños en el alfabeto de símbolos para los datos de entrenamiento (a)) y para los datos de prueba (b). Se aprecia un aumento gradual entre más cantidad de símbolos haya, lo que significa mayor cantidad de centroides que pueden describir una secuencia de acciones, ya que una cantidad pequeña de centroides no puede describir tantas acciones "distintas" y la secuencia para realizarlas.

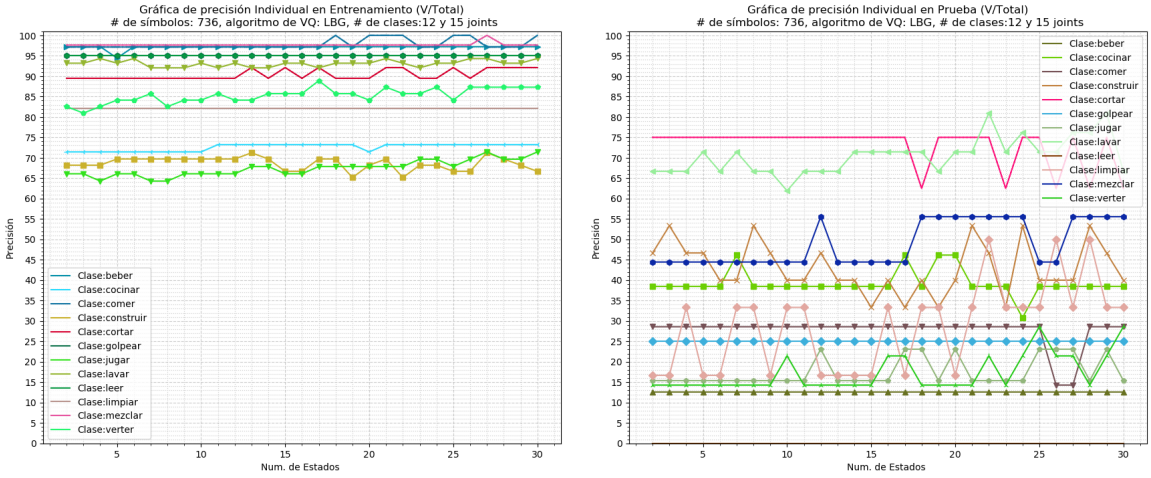


(a) Con datos de entrenamiento

(b) Con datos de prueba

Figura 7-3: Ejemplo del cambio en el desempeño al aumentar o disminuir el número de símbolos al entrenar los modelos.

También se muestra la precisión de los modelos por separado para un tamaño específico de alfabeto de símbolos y para un distinto número de estados en la figura 7-4.



(a) Con datos de entrenamiento

(b) Con datos de prueba

Figura 7-4: Ejemplo de las precisiones individuales por cada modelo al variar el número de símbolos en el entrenamiento.

En la siguiente sección se hará un análisis más detallado de estos resultados.

7.1.2. Entrenamiento con la segunda metodología propuesta

Dado que no existe una única manera de entrenar modelos ocultos de Markov [33], también se muestra una segunda aproximación para entrenar estos modelos, la cual utiliza en general conceptos similares al anterior método pero con algunas variaciones especiales y que fueron mencionados anteriormente y en la sección 6.3.3 del anterior capítulo.

De igual manera para poder evaluar los modelos entrenados, se hicieron variaciones en hiperparámetros, intentando obtener la combinación que mejor desempeño tenga 7-5. La manera de evaluar su desempeño es igual que con el método utilizado en la sección anterior. Es de mencionar que en ocasiones la inferencia de alguna secuencia daba como resultado probabilidades cero para todas las clases, por lo que al ocurrir esto no se modificaba la matriz de confusión y por tanto al calcular el desempeño con la ecuación 7-1, el valor del denominador será todavía mayor que el del numerador en el mejor de los casos para las demás secuencias.

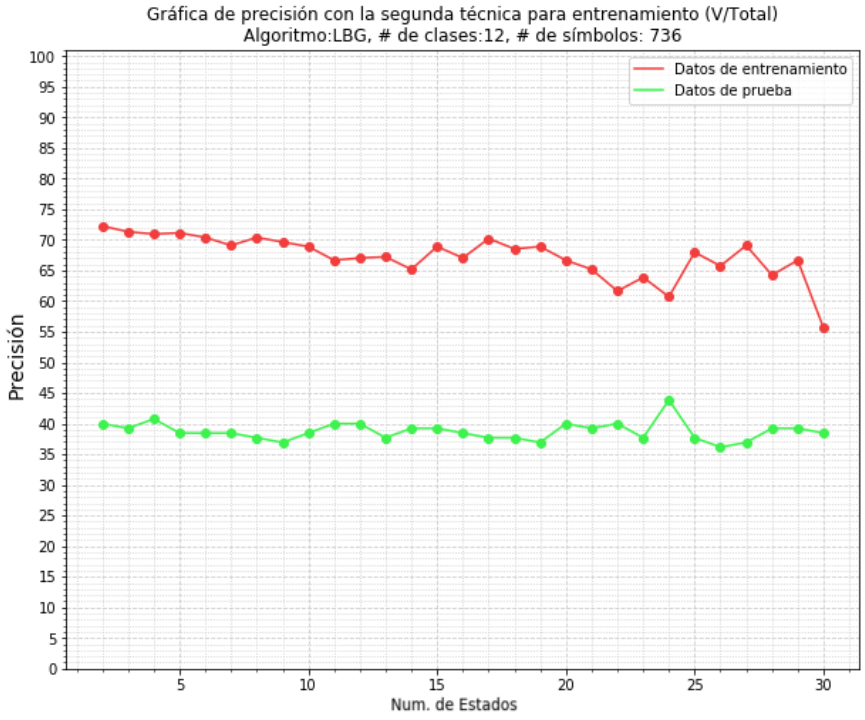


Figura 7-5: Gráfica de desempeño variando el número de estados, utilizando la segunda aproximación. Sin añadir los resultados de los mejores 2 (Top 2).

Algunas estadísticas del desempeño de los modelos

Por último se muestra en la tabla 7-1 que el tiempo de entrenamiento para los modelos estableciendo los hiperparámetros con el mejor desempeño (para datos de validación) es muy pequeño para ambas metodologías propuestas, contrastando de gran manera el tiempo que requieren los algoritmos de entrenamiento para redes neuronales (metodología utilizada en [40]) y también que no es necesario tener una computadora con grandes características en software y hardware para entrenar estos modelos. También se incluye desde el tiempo que se necesitó para poder obtener el alfabeto de símbolos *codebook* de un tamaño específico y con dos algoritmos de cuantización vectorial (VQ), siendo el primero *LBG* desarrollado a mano y el segundo *K-medias* utilizando librerías gratuitas especializadas para desarrollo de procesos de *Data Science* llamada **Scikit-Learn** en lenguaje de programación **python**.

Tabla 7-1: Algunas Estadísticas de esta propuesta

Tiempos de entrenamiento u obtención de datos con hiperparámetros específicos								
Especificaciones del software de la computadora:								
GPU: Nvidia Quadro P1000				RAM: 16 Gb				
Procesador: AMD Ryzen 5				S.O.: Linux Ubuntu 20				
# de clases	# de estados	Tamaño del Codebook	Obtención de símbolos (LBG)	Obtención de símbolos (K-medias)	Entrenamiento de los HMM (segundos)		Precisión máxima encontrada (Datos de prueba)	
					Método 1	Método 2	Met. 1	Met. 2
5	14	128	287.41 s.	4.53 s.	8.15 s.	49.98 s.	56.75 %	45 %
		160*	*49.01 s.	5.37 s.	18.31 s.	56.05 s.	75.67 %	60 %
12	24	128	2270.96 s.	42.81 s.	56.90 s.	1147.86 s.	30.32 %	25.38 %
		736*	*589.85 s.	131.53 s.	184.14 s.	1295.43 s.	44.26 %	44.84 %

*: utilizando la propuesta de obtener centroides por separado y luego juntarlos

7.2. Justificación de los resultados

Como se pudo observar en las distintas gráficas de la sección anterior, el desempeño no es bueno, aunque se aprecie un aumento en la precisión al incrementar el tamaño de símbolos para el conjunto de entrenamiento, para datos nuevos, es decir, para los datos de prueba, este

desempeño se mantiene casi igual o es mínimo su mejora proporcional a como es el desempeño de los datos de entrenamiento. Es de esperarse que estos métodos no tengan una precisión cercana al 100 % para esta cantidad de información y de clases, y si se hace la comparación con el desempeño del trabajo en [40], se esperaba como máximo ese desempeño obtenido intentando utilizar la misma información y el autor utilizando redes neuronales profundas.

Otra razón por la cual el desempeño de este trabajo es al comparar con el estado del arte, utilizando HMM y un conjunto de datos distinto en general, pero con una manera similar al procesar las secuencias de acciones obteniendo la información del esqueleto de la persona mediante distintos métodos, el desempeño de esta propuesta no es cercano a los obtenidos en esos trabajos.

Para poder dar una explicación de la causa de este bajo desempeño se explicará a continuación algunos factores que pueden limitar el desempeño de la propuesta de este trabajo, y por último se hará una comparativa la implementar este mismo trabajo pero con otros datos y así justificar el por qué los resultados son los mostrados anteriormente.

7.2.1. Comparación de símbolos en las acciones

La principal razón por la que se cree que este este trabajo tuvo el desempeño obtenido es por la naturaleza del conjunto de datos utilizado. Aunque este *dataset* fue creado para poder obtener esqueletos de personas en un ambiente controlado, la naturaleza de las acciones que se presentan no "ayudan" a los HMM para poder diferenciar de una acción y otra, contrastando con las redes neuronales utilizadas [40] que procesan muchos datos más que estos modelos y de alguna manera sí lograron inferir mejor estas acciones. Otra razón que influye y que incluso se menciona desde el trabajo en [40], es que el conjunto de datos no está balanceado, influyendo esto incluso en algoritmos de entrenamiento para redes neuronales profundas y claramente para técnicas más "pequeñas" como lo son los HMM. Claramente el número de cálculos en una red neuronal profunda es de millones de parámetros mientras que en un modelo oculto de Markov son solo proporcional al tamaño de los símbolos y estados mediante un cálculo descrito en [33] de forma que para un algoritmo, el cálculo es del orden de N^2T cálculos, siendo N el número de

estados y T el tamaño de la secuencia o el tiempo t que tarda la observación O, claramente este número difícilmente será cercano a los millones de cálculos que hace una red neuronal profunda.

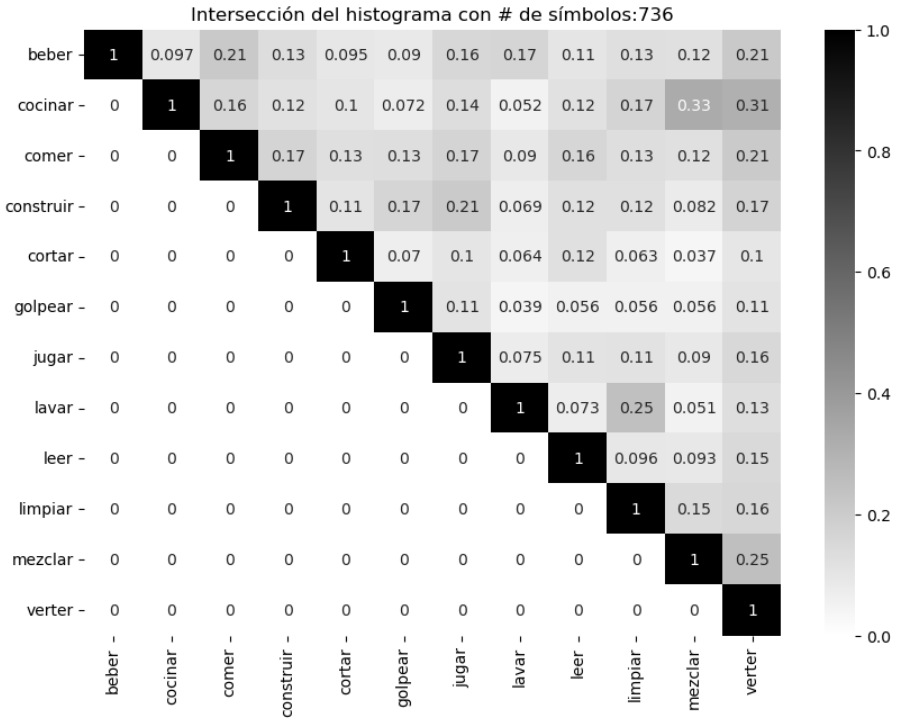


Figura 7-6: Matriz de confusión de la intersección de histogramas con un número determinado de símbolos.

Dado que se concluye que mayormente fue problema del conjunto de datos utilizado, se hace una comparativa en la figura 7-6 para justificar esto. Esta matriz de confusión muestra la dificultad del HMM para diferenciar las acciones dado que algunos símbolos utilizados en cada acción son utilizados en otras acciones, por lo que es posible que el HMM se "confunda" con secuencias de acciones que utilicen alguna cantidad de símbolos que también usan otras. Esta matriz de confusión se obtuvo de hacer un histograma de los símbolos utilizados en cada clase por separado y al hacer la intersección entre cada una de ellas.

Para intentar dar solución al parecido de estos símbolos, se hizo mediante una propuesta de obtención de centroides mediante VQ pero con todos los datos de cada clase por separado, es decir se obtienen N cantidad de centroides por cada clase y para el entrenamiento de los HMM, estos $N \times 12$ centroides se unen, así al obtener los símbolos por cada secuencia, en teoría

deberían de tender a usar los centroides obtenidos en su clase correspondiente en parte debido a que los algoritmos usados en esta cuantización son algoritmos *frecuentistas* por lo que es posible que algunos datos sean dominantes y no permitan diferenciar a otros datos más reducidos. Se muestra en la figura 7-7 una matriz de confusión en la cual se obtuvieron $N = 64$ centroides por cada clase, dando un total de $N = 12 \times 64 = 768$ centroides al juntarlos en un único alfabeto de símbolos. Haciendo una comparativa con el obtener la misma cantidad de centroides pero con todos los datos de todas las clases se ve una reducción en la intersección de los histogramas de las clases.

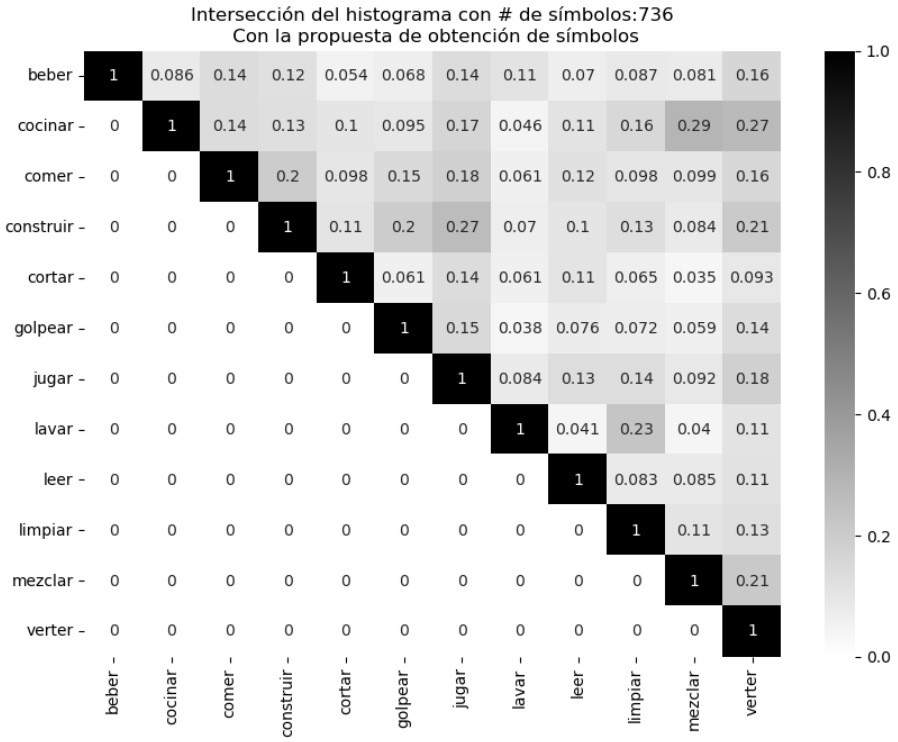


Figura 7-7: Matriz de confusión de la intersección de histogramas con la propuesta de obtención de símbolos (*codebook*) por separado.

Aunque se vio una reducción en el parecido de algunos de los símbolos utilizados en cada clase, al entrenar los modelos de Markov oculto mejoró un poco el desempeño del mismo utilizando la misma cantidad de centroides (figura 7-2 y 7-5) pero obtenido con todos los datos juntos, como se muestran en la figura 7-8).

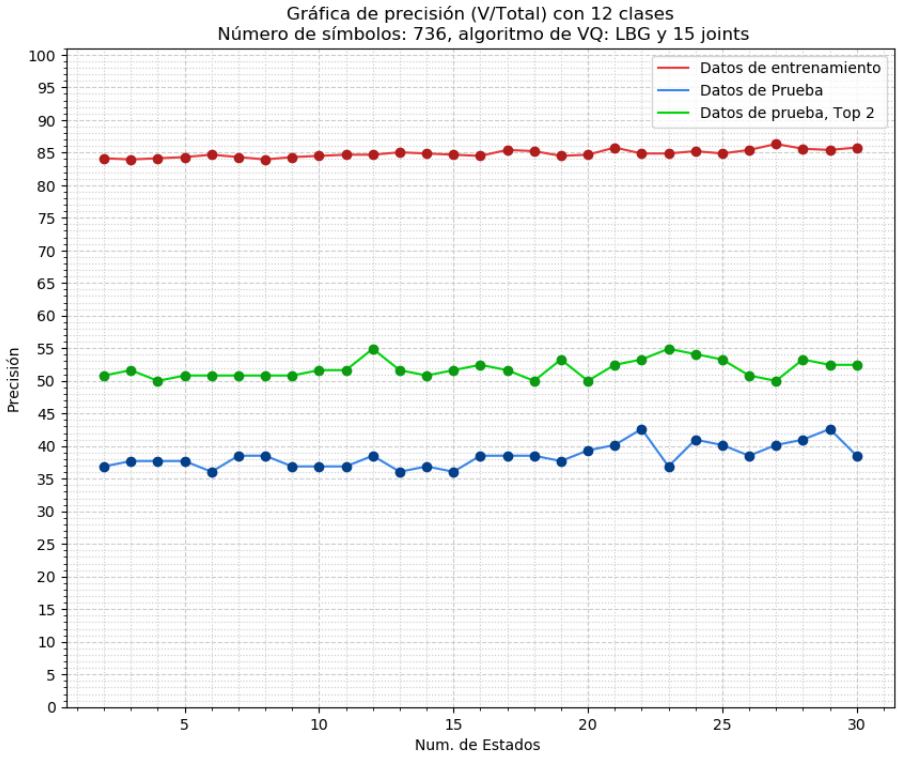


Figura 7-8: Gráfica del desempeño del modelo al juntar los centroides obtenidos por clase.

Dado lo anterior, se procede a explicar otra razón para la cual no tuvo buen desempeño este trabajo.

Comparativa entre los esqueletos en las acciones

A pesar de que se logró reducir el parecido de símbolos utilizados en cada clase, el desempeño no es suficiente para datos de prueba, por lo que otro de las posibles razones para que este método tuvo bajo desempeño, sigue siendo debido a la naturaleza del conjunto de datos utilizados, es decir las acciones realizadas en ellas. Como se describió en el Capítulo 5, el dataset consta de 12 acciones bien definidas, y que no son acciones muy complejas, por ejemplo una acción compuesta de otra más, sino acciones por separado y que fueron grabadas en ambiente controlado y tratando de acciones realizadas una a la vez.

Estas acciones, al observar la manera en que son realizadas y con los datos obtenidos en la cuantización vectorial, se ve que realmente no hay mucha diferencia apreciable en el esqueleto

obtenido al realizar una acción y otra. En efecto, los símbolos pueden ser distintos para cada clase, pero visualizando los centroides correspondientes a cada uno, se puede apreciar en muchos que la posición del esqueleto de un centroide y otro no varía tanto. Acciones como *lavar* y *limpiar* pueden ser muy parecidas entre sí, o acciones como *cortar* y *cocinar* pueden tener poses que son parecidas a las dos anteriores mencionadas, y toda una secuencia de una acción puede utilizar una cantidad mínima de símbolos que no permiten describir todo lo que la acción realiza y su diferencia con otra acción. Para dar un ejemplo concreto, la acción *cortar* (en este conjunto de datos) implica solo subir y bajar un poco una mano y la otra muy cerca agarrando el objeto, estando el resto del cuerpo en la misma posición en general. Esta misma posición de la mano y el cuerpo puede ser muy parecida en la acción *lavar* al tener las manos en una posición cercana, pero con otro objeto claramente y durante toda la acción no realizar tantos movimientos distintos. Todo lo anterior puede influir para que no ayude al HMM por la naturaleza del mismo al diferenciar la acción realizada. Se muestra una pequeña secuencia de dos acciones (figura 7-9) que aunque el símbolo puede ser distinto, el esqueleto es casi invariante entre la secuencia o es muy parecido entre las acciones.

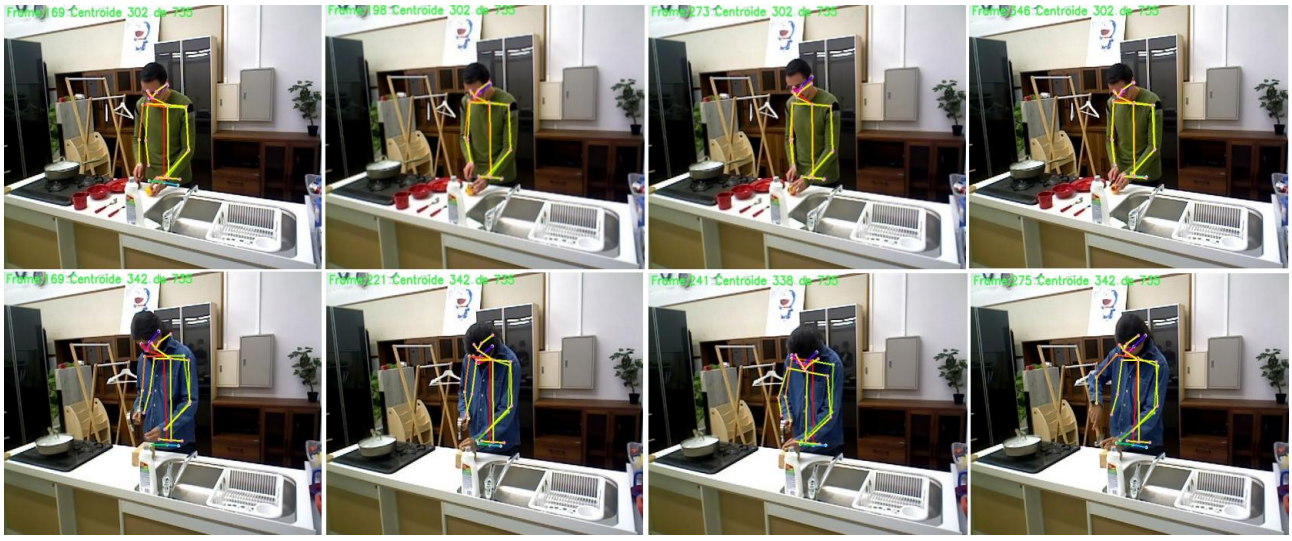


Figura 7-9: Ejemplo de secuencias para dos acciones (cortar y golpear, respectivamente), mostrando la mínima diferencia entre ellas e incluso entre la propia secuencia al dibujar el esqueleto obtenido y el centroide correspondiente (esqueleto amarillo).

Reducción de las clases

Para poder comprobar que la cantidad de las acciones y la naturaleza de las mismas influyen en el desempeño, se hizo una prueba con un número menor de acciones, intentando utilizar las acciones más distintas entre ellas y con un conjunto un poco más balanceado. En la figura 7-10 se muestra el desempeño de este HMM con un subconjunto de datos con 5 clases (acciones) pero manteniendo el mismo tamaño de alfabeto de símbolos.

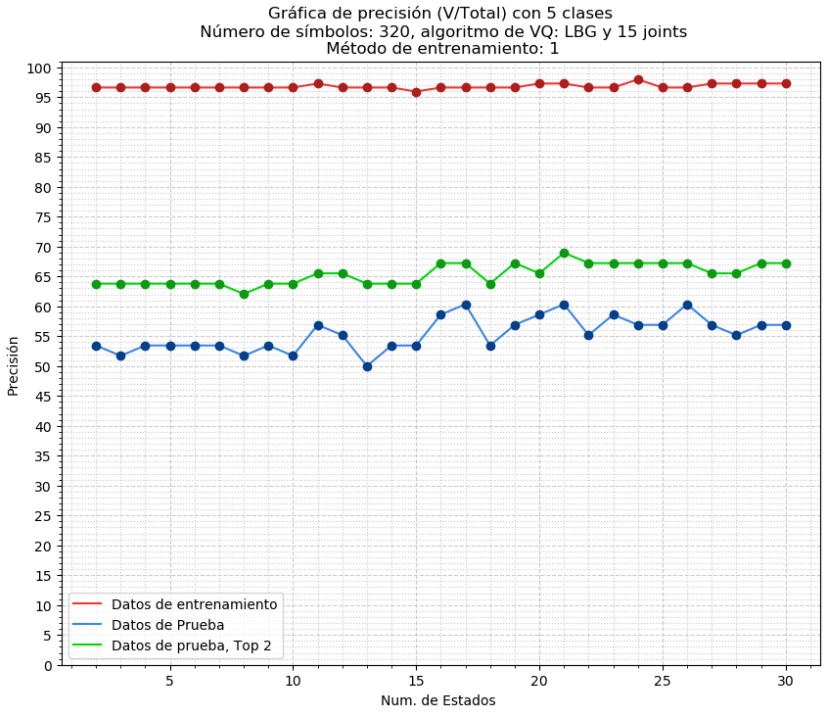


Figura 7-10: Gráfica de desempeño con menos clases (acciones).

Por lo mostrado en la anterior figura, se observa un ligero mejor desempeño, demostrando que la naturaleza de muchas acciones influye mayormente con estos modelos. Aun con lo mostrado con un conjunto reducido de clases, se mostrará en la siguiente sección una comparativa con otro conjunto de datos creado específicamente para demostrar la influencia de la naturaleza de las acciones en un conjunto de datos.

7.2.2. Sistema con un conjunto de datos diferente

Dado los resultados vistos en este capítulo, y con lo descrito en uno de los objetivos particulares de este trabajo, se creó un conjunto de datos con otras acciones, pero orientado especialmente para demostrar que la naturaleza del primer conjunto de datos influye en el bajo desempeño.

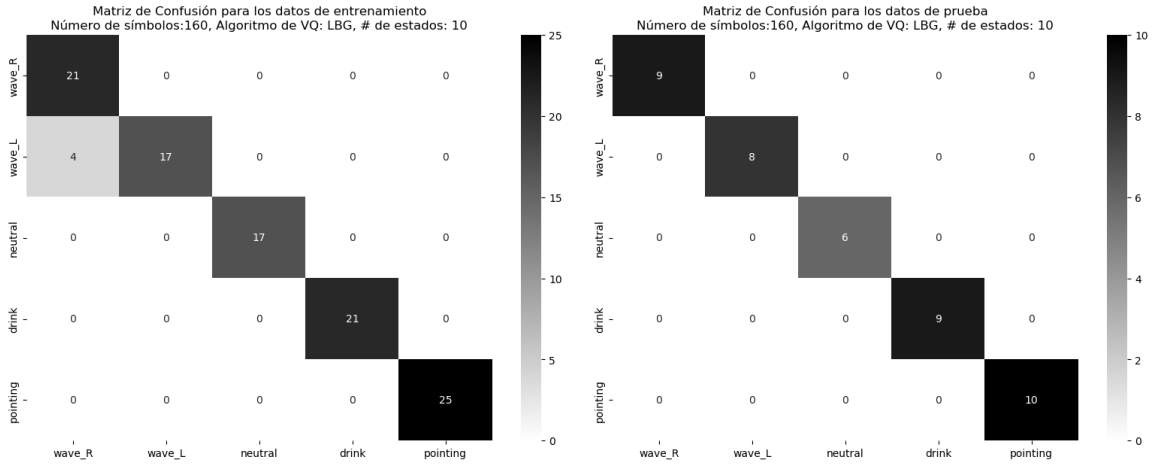
De manera similar, se realizó la grabación de acciones en un ambiente controlado y con acciones que sean lo más distintas posibles. En este caso se crearon una serie de videos con distintas personas realizando acciones pensadas con el objetivo de que a primera vista fuesen lo más distintas posibles, así como tener un conjunto de datos más balanceado, es decir que exista un número similar de secuencias para cada acción. En la tabla 7-2 se aprecia un número menor de acciones para la clase *Neutral* y un número mayor para la acción *Apuntar*, debido a que se trató de tener una cantidad de cuadros total similar y la cantidad de cuadros total es aún menor que la mayoría de las otras acciones, respectivamente. Para demostrar la diferencia, se creó el conjunto de datos con únicamente 4 acciones y una "acción" adicional el cual indicará que no esta haciendo ninguna acción (neutral) como se muestra en la tabla 7-2, también para poder decirle al modelo que si detecte una *no acción* y por tanto tenga una consecuencia distinta para la aplicación que se le dé. De manera similar se procesó este conjunto de datos a través del programa **OpenPose** para obtener los datos de los esqueletos detectados y crear un conjunto de datos con únicamente la información de cada esqueleto por cada imagen y cada acción.

Tabla 7-2: Información del segundo dataset

Acción	No. de acciones	Cantidad de cuadros total
Saludar (mano derecha)	30	2590
Saludar (mano izquierda)	30	2695
Beber	30	1765
Apuntar	36	1845
Neutral	25	2076
Total	151	10971

Igualmente se obtienen alfabetos de símbolos (*codebooks de centroides*) de distintos tamaños

a través de procesar esto datos con las técnicas de cuantización vectorial (VQ) mencionadas en este proyecto. Con lo anterior se aplica un subconjunto de datos (70 %) para entrenar los modelos y el restante para validarlos. En la figura 7-11 se muestra una matriz de confusión con los datos para el entrenamiento (a)) y para los datos de validación (b)) con los hiperparámetros establecidos que muestren el mejor desempeño.



(a) Con datos de entrenamiento (b) Con datos de prueba

Figura 7-11: Desempeños de los modelos para el segundo *dataset*.

También se muestra la gráfica del entrenamiento (figura 7-12) al variar el número de estados, mostrando la precisión para los datos de entrenamiento y los datos de prueba, utilizando la propuesta de obtener los centroides por cada clase por separado y juntarlos para el entrenamiento de los modelos.

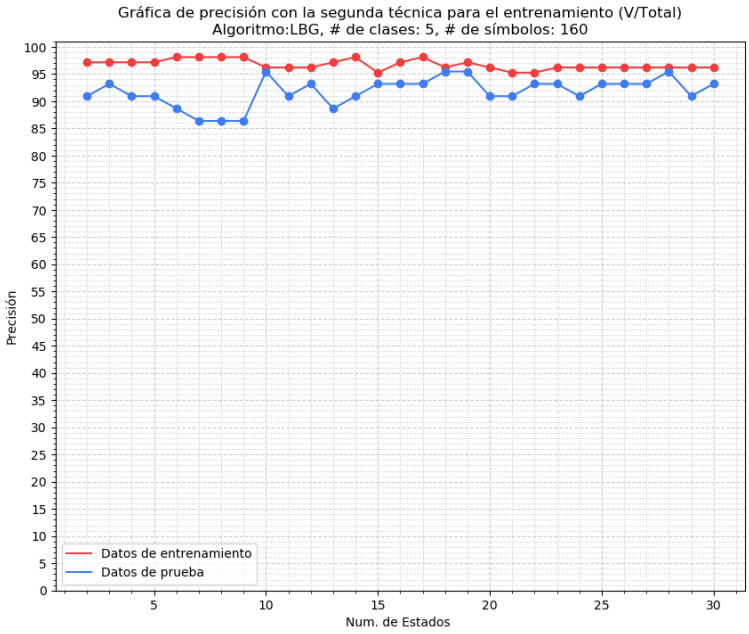


Figura 7-12: Gráfica de la precisión global de los modelos entrenados para este conjunto de datos.

Por último se muestra la intersección de los histogramas de este conjunto de datos en la figura 7-13, mostrando la diferencia entre los símbolos que se emplea en cada clase.

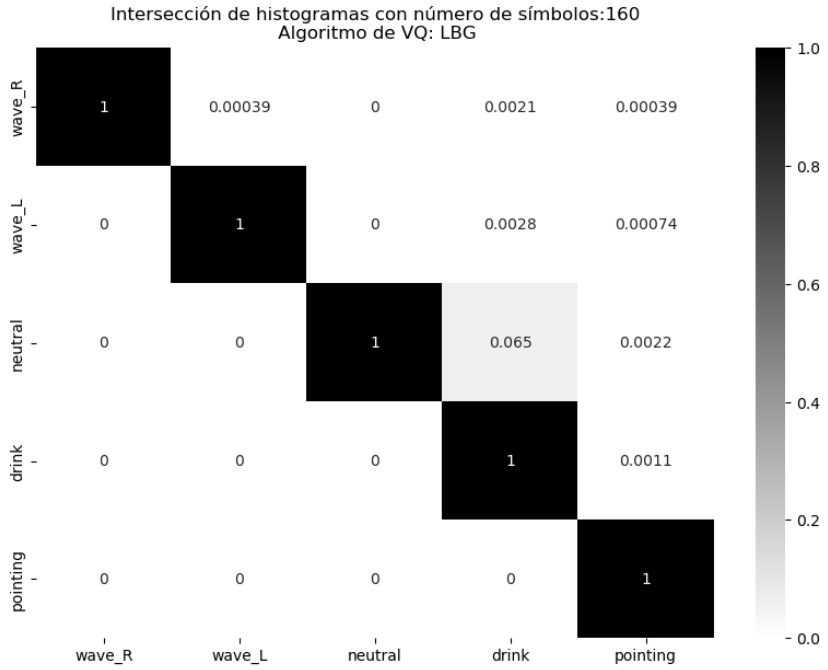


Figura 7-13: Intersección de histogramas con el segundo conjunto de datos.

Haciendo una comparación de la gráfica de desempeño entre el primer conjunto de datos con un número menor de acciones (fig. 7-10) y con el segundo conjunto de datos (fig. 7-12) se puede visualizar un notable incremento en la precisión para el segundo dataset a pesar de reducir el número de acciones para el primero e intentando escoger las acciones mutuamente más excluyentes. Lo anterior puede deberse a lo mencionado sobre la posible causa del poco desempeño del conjunto de datos completo ya que ni siquiera reduciendo el mismo conjunto de datos, los modelos pueden diferenciar bien las secuencias de acciones.

Este conjunto de datos sigue en construcción, por lo que se planea tener más clases (acciones) y que la cantidad de cuadros sean lo más parecido posible.

Capítulo 8

Conclusiones y Trabajo Futuro

8.1. Conclusiones

Se presentaron los resultados de esta propuesta utilizando la información creada en el trabajo de Vázquez [40], intentando continuar con la misma base establecida, ya que como se mencionó en capítulos previos, se volvió a hacer el etiquetado del conjunto de datos. Con los resultados del capítulo anterior se dio como conclusión de que esta propuesta, utilizando modelos de Markov ocultos y con este conjunto de datos, no dieron los desempeños deseados. Con lo visto en las gráficas de los resultados es posible mencionar que las precisiones de los modelos no alcanzan a generalizar bien las acciones "nuevas" o desconocidas, al quedarse con resultados bajos, a comparación de los datos de entrenamiento que si logra inferir de manera aceptable.

Se da por hecho que estos modelos no tendrían por que tener los mismos desempeños que al utilizar técnicas robustas de inteligencia artificial como lo son las redes neuronales profundas, utilizadas en [40], y haciendo comparación con los resultados para el sistema detector de acciones, es todavía más bajo su desempeño de lo que se pensaba que pudiera llegar.

Se intentaron diversas aproximaciones para lograr mejorar este módulo detector de acciones con modelos de Markov oculto (HMM), por ejemplo con la variación de tamaños del alfabeto, la obtención de centroides en la cuantización vectorial de manera separada, así como el entrenamiento para distinto número de estados, pero se comprobó que aunque se optimizó ligeramente

el desempeño de los modelos (figura 7-8), no fue lo suficiente para considerar que el modelo tiene el desempeño esperado y deseado.

Desde el Capítulo 6 se quiso mostrar la naturaleza del conjunto de datos con el que se trabajó, queriendo dar argumentos para poder justificar el desempeño que este sistema obtuvo en el Capítulo 7 y las razones por las que pudiera verse afectado el mismo.

Debido a todo lo anterior, no se dio seguimiento al resto del sistema general propuesto, es decir, el módulo detector de objetos y el módulo centralizador 5-3, debido por una parte a que el desempeño general claramente se vería afectado por el del módulo detector de acciones y no habría necesidad de hacer una comparación de la que ya se sabe que sería diferente.

Como alternativa para justificar que el conjunto de datos con esta propuesta no es la adecuada, se creó un nuevo conjunto de datos reducido a comparación, para poder mostrar que en general la propuesta de utilizar HMM para detectar acciones sí es válida y buena, comprobando esto en la figura 7-12. Con lo anterior es posible expresar que estas técnicas pueden servir para inferir un número reducido de acciones, afectando su desempeño al aumentar el número de éstas y el tipo de acciones que se realizan (si son muy similares entre ellas).

También es posible expresar que para poder diferenciar las acciones utilizando únicamente la información espacial del esqueleto de una persona en una imagen no sea suficiente para las técnicas de entrenamiento de los modelos ocultos de Markov, aun con más razón si no todas las extremidades del cuerpo (para este conjunto de datos) aportan información relevante, ya que las extremidades inferiores como los pies y rodillas no aparecen si quiera en este conjunto de datos.

Con todo lo anteriormente explicado y de acuerdo con la hipótesis planteada al inicio de este trabajo, se da por entendido que: para este conjunto de datos y la metodología específica planteada, no resulta un mejor desempeño utilizar conjuntamente redes neuronales y modelos ocultos de Markov que al utilizar únicamente redes neuronales [40]. Nuevamente se hace hincapié que con un conjunto de datos distinto el cual su diseño es más específico para esta metodología, por ejemplo la inferencia de menos acciones, da resultados muy buenos (figura 7-12).

8.2. Trabajo Futuro

Con lo anteriormente mencionado, se propone como un posible trabajo futuro utilizar más datos de la imagen que aporten información a los modelos ocultos de Markov y que no se utilice únicamente los vectores de las coordenadas de las extremidades. Para esto se puede realizar mediante una técnica de procesamiento de imágenes la cual permite "reconocer" el movimiento aparente de objetos, personas, superficies, etc. en una secuencia de imágenes ([9], [11]), llamada *Flujo Óptico* (**Optical Flow** en inglés). Esto permitiría no solo tener la información espacial de la persona haciendo una acción, sino además tener una idea de cómo se realiza el movimiento de la persona mediante una orientación del cambio de los píxeles en una secuencia de imágenes. Este se obtiene mediante un campo de vectores bidimensional donde cada uno es un vector de desplazamiento el cual muestra el movimiento de píxeles de un tiempo a otro [39].

Figura 8-1: Ejemplo visual de utilizar *Optical Flow* en una imagen al mostrarse en ella el movimiento de objetos como los coches. Imagen obtenida de [39].

Al obtener estos flujos de movimientos, es posible obtener el comportamiento de las acciones realizadas en este conjunto de datos y así tener más información para el HMM que ayude a diferenciar entre acciones.

Bibliografía

[1] Aggarwal, J. y Ryoo, M. Human Activity Analysis: A Review **43**(3) (2011). https://doi.org/10.1145/1922649.1922653

[2] Bernardic, V. Los robots japoneses toman la industria del automóvil (2019). https://www.abc.es/tecnologia/informatica/soluciones/abci-robots-japoneses-toman-industria-automovil-201904100211_noticia.html. Consultado el 20 de octubre de 2022

[3] Brand, M., Oliver, N., y Pentland, A. Coupled Hidden Markov Models for complex action recognition. *2012 IEEE Conference on Computer Vision and Pattern Recognition* **0**:994 (1997)

[4] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., y Sheikh, Y.A. OpenPose 1.7.0 The first real-time multi-person system to jointly detect human body, hand, facial, and foot keypoints (2019). https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/index.html

[5] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., y Sheikh, Y.A. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields (2019). 1812.08008

[6] Chandra, A. McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron (2018). https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1. Consultado el 26 Octubre de 2021

[7] Damen, D., Doughty, H., Farinella, G.M., Fidler, S., Furnari, A., Kazakos, E., Moltisanti, D., Munro, J., Perrett, T., Price, W., y Wray, M. The EPIC-KITCHENS Dataset: Collection, Challenges and Baselines. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2020)

[8] Damen, D., Fragomeni, A., Munro, J., Perrett, T., Whettam, D., Wray, M., Furnari, A., Farinella, G.M., y Moltisanti, D. EPIC-KITCHENS-100- 2021 Challenges Report. Informe técnico, University of Bristol (2021)

[9] David H. Warren, E.R.S. *Electronic Spatial Sensing for the Blind.* 1ª edición. Springer Dordrecht (1985)

[10] Fink, G. *Markov Models for Pattern Recognition: From Theory to Applications.* Springer e-Books (2007)

[11] Fleet, D.J. y Weiss, Y. Optical Flow Estimation. En *Handbook of Mathematical Models in Computer Vision* (2006)

[12] Fuentes, O., Savage, J., y Contreras, L. A SLAM system based on Hidden Markov Models. *Informatics and Automation* **21**(1):181–212 (2022)

[13] Gray, R. Vector quantization. *IEEE ASSP Magazine* **1**(2):4–29 (1984)

[14] He, K., Gkioxari, G., Dollár, P., y Girshick, R. Mask R-CNN (2018). 1703.06870

[15] Hochreiter, S. y Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **9**(8):1735–1780 (1997). https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf

[16] Hochreiter, S. y Schmidhuber, J. Long Short-term Memory. *Neural computation* **9**:1735–80 (1997)

[17] Ijjina, E.P. y Mohan, C.K. Human Action Recognition Based on Recognition of Linear Patterns in Action Bank Features Using Convolutional Neural Networks. En *2014 13th International Conference on Machine Learning and Applications*, págs. 178–182 (2014)

[18] KOBIAN60. Robots de servicio (2015). https://kobian60.webnode.es/apicacion-de-la-robotica/robots-de-servicio/. Consultado el 30 de Febrero de 2021

[19] KRIZHEVSKY, A., SUTSKEVER, I., Y HINTON, G.E. ImageNet Classification with Deep Convolutional Neural Networks. En F. Pereira, C.J.C. Burges, L. Bottou, y K.Q. Weinberger (editores), *Advances in Neural Information Processing Systems*, tomo 25. Curran Associates, Inc. (2012)

[20] KROGH, A., BROWN, M., MIAN, I., SJÖLANDER, K., Y HAUSSLER, D. Hidden Markov Models in Computational Biology: Applications to Protein Modeling. *Journal of Molecular Biology* **235**(5):1501–1531 (1994)

[21] LEVINSON, S.E., RABINER, L.R., Y SONDHI, M.M. An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. *Bell System Technical Journal* **62**(4):1035–1074 (1983). https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1983.tb03114.x

[22] LIU, T., SONG, Y., GU, Y., Y LI, A. Human Action Recognition Based on Depth Images from Microsoft Kinect. En *2013 Fourth Global Congress on Intelligent Systems*, págs. 200–204 (2013)

[23] LLOYD, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory* **28**(2):129–137 (1982)

[24] LU, T., PENG, L., Y MIAO, S. Human Action Recognition of Hidden Markov Model Based on Depth Information. En *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, págs. 354–357 (2016)

[25] MANNING, C.D. Y SCHUTZE, H. Foundations of Statistical Natural Language Processing. *Natural Language Engineering* (2000)

[26] MITCHEL, T. *Machine Learning.* McGraw Hill Science (1997)

[27] Muybridge, E. *Animal locomotion*, tomo VII. University of Pennsylvania (1887). https://www.flickr.com/photos/boston_public_library/4326303094. Consultado el 15 de marzo de 2021

[28] Olah, C. Long Short-Term Memory (2015). https://colah.github.io/posts/2015-08-Understanding-LSTMs/, Consultado el 10 Octubre de 2021

[29] Osorio-Comparan, R., de J. Vázquez, E., López-Juárez, I., Peña-Cabrera, M., Bustamante, M., y Lefranc, G. Object Detection Algorithms and Implementation in a Robot of Service. En *2018 IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, págs. 1–7 (2018)

[30] Postawka, A. y Śliwiński, P. Action Recognition by Weighted Averaged Hidden Markov Models. En *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, págs. 401–405 (2019)

[31] Rabiner, L. y Juang, B. An introduction to hidden Markov models. *IEEE ASSP Magazine* **3**(1):4–16 (1986)

[32] Rabiner, L.R. y Juang, B.H. Hidden Markov Models for Speech Recognition. *Technometrics* **33**(3):251–272 (1991)

[33] Rabiner, L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* **77**(2):257–286 (1989)

[34] Rangel Ortiz, J. *Scene Understanding for Mobile Robots exploiting Deep Learning Techniques*. Tesis Doctoral, Universidad Tecnológica de Panamá (2017)

[35] Rincón, L. *Introducción a los procesos estocásticos*. Biblioteca Nacional de México (2012)

[36] Rumelhart, D.E., Hinton, G.E., y Williams, R.J. Learning internal representations by error propagation (1986)

[37] Savage, J. Notas de clase, lección 5, 6 y 12: Cuantización Vectorial, Modelos de Markov Oculto, Redes Neuronales Artificiales (-). https://biorobotics.fi-p.unam.mx/reconocimiento-de-patrones/. Consultado el 21 de octubre de 2022

[38] Sultanov, J. Convolutional Neural Networks (11 de agosto de 2019). https://medium.com/ai^3-theory-practice-business/convolutional-neural-networks-2bd6c95528b4

[39] Vision, O.S.C. OpenCV: Optical Flow tutorial (-). https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html. Consultado el 14 de Septiembre de 2022

[40] Vázquez Silva, E.J. *Categorización semántica de objetos a partir de la interacción humano-robot*. Proyecto Fin de Carrera, U.N.A.M. (2021)

[41] Yamato, J., Ohya, J., y Ishii, K. Recognizing human action in time-sequential images using hidden Markov model. En *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, págs. 379–385 (1992)

[42] Zhao, L., Gao, X., Tao, D., y Li, X. Tracking Human Pose Using Max-Margin Markov Models. *IEEE Transactions on Image Processing* **24**(12):5274–5287 (2015)