Escuela de Computación

Maestría en Computación con énfasis en Ciencias de la Computación

Análisis comparativo de clasificadores para gestos descritos por LaGeR

Tesis
para el grado de

Magister Scientiæ en Computación

Autor
Milena Mesén Solórzano

Asesor
Erick Mata Montero, Ph.D.

diciembre 2018

Dedicada a mi padre fallecido, mi esposo Andrés y mi madre.

Agradecimientos

Agradezco a mi asesor y mi esposo por su soporte, paciencia y motivación en este proceso tan largo y lleno de pruebas. Sin su apoyo esta investigación no hubiese sido posible.

Índice general

Índice de cuadros

Índice de figuras

Abreviaturas

API: Application Programming Interface, es un conjunto de interfaces que permite integrar un sistema informático con otro.

GLC: Gramática Libre de Contexto, es una gramática con estructura de frase donde cada símbolo no terminal produce una cadena de terminales y/o no terminales.

JSON: JavaScript Object Notation, es un formato de intercambio de datos liviano y legible por humanos. Es un estándar abierto ampliamente utilizado..

LaGeR: Language for Gesture Representation, es un lenguaje para la representación de gestos por medio de una secuencia de movimientos en 26 intervalos de 45 grados en espacio tridimensional..

MNIST: Modified National Institute of Standards and Technology, es una instituto científico de los Estados Unidos cuya base de datos de imágenes es comúnmente utilizada para entrenar y evaluar clasificadores tales como redes neuronales..

OSVR: Open Source Virtual Reality, es un middleware que provee APIs que abstraen dispositivos de entrada y salida de realidad virtual para su uso con programas de software..

Resumen

La presente tesis realiza un análisis comparativo entre el algoritmo Damerau-Levenshtein y un algoritmo de deep learning y confirma que, utilizando deep learning, se puede obtener mejores resultados en el reconocimiento de gestos descritos por LaGeR, tanto en tiempo de ejecución como en exactitud. Además, demuestra que dicho método es efectivo con varios dispositivos de entrada.

Abstract

The present thesis is a comparative analysis between the Damerau-Levenshtein algorithm and a deep learning algorithm and confirms that, by using deep learning techniques, one can obtain better results when recognizing gestures described by LaGeR, in terms of both running time and accuracy. Additionally, it shows that said method is effective with several input devices.

www.tec.ac.cr

TEC | Tecnológico de Costa Rica
Programa de Maestría en Computación
Teléfono: 2550-2402

ACTA DE APROBACION DE TESIS

Con fundamento en lo que establecen los Artículos **22-24-25** del "Manual de Normas y Procedimientos para optar por el título de "MAGÍSTER SCIENTIAE EN COMPUTACION", el Tribunal Examinador de Tesis (TET), nombrado con el propósito de evaluar la tesis de grado.

Análisis Comparativo de clasificadores para gestos descritos por LaGeR

Habiendo analizado el resultado general del trabajo presentado por el estudiante:

Primer Apellido	Segundo Apellido	Nombre	No. de carné
MESEN	SOLÓRZANO	MILENA MARIA	201281977

Emite el siguiente dictamen:

☒ APROBADO

El TET, considerando que el trabajo realizado por el estudiante es SOBRESALIENTE, le otorga la siguiente MENCION HONORIFICA:

☒ CUM LAUDE ○ MAGNA CUM LAUDE ○ SUMMA CUM LAUDE

○ REPROBADO

○ SE RECOMIENDA ○ NO SE RECOMIENDA

Brindarle una nueva oportunidad para la DEFENSA PUBLICA de su Tesis

NUEVA FECHA: _____

Dando fe de lo aquí expuesto firmamos (*IDEM*: HOJAS DE APROBACION DE TESIS)

Dr. Erick Mata Montero
Profesor Asesor

Máster Armando Arce Orozco
Profesor Lector

Dr. Franklin Hernández Castro
Profesor Externo

Dr. Roberto Cortés Morales
Coordinador del Programa de Maestría en Computación

11 de Diciembre del 2018

TEC | Tecnológico de Costa Rica
Sello Maestría en Computación

FT-01-MC

www.tec.ac.cr

TEC | Tecnológico de Costa Rica
Programa de Maestría en Computación
Teléfono: 2550-2402

SINAES POST-GRADO ACREDITADO COSTA RICA

APROBACIÓN DE PROYECTO

"Análisis Comparativo de clasificadores para gestos descritos por LaGeR"

TRIBUNAL EXAMINADOR

Dr. Erick Mata Montero
Profesor Asesor

Máster Armando Arce Orozco
Profesor Lector

Dr. Franklin Hernández Castro
Profesor Externo

Dr. Roberto Cortés Morales
Coordinador del Programa
de Maestría en Computación

Diciembre, 2018

Capítulo 1

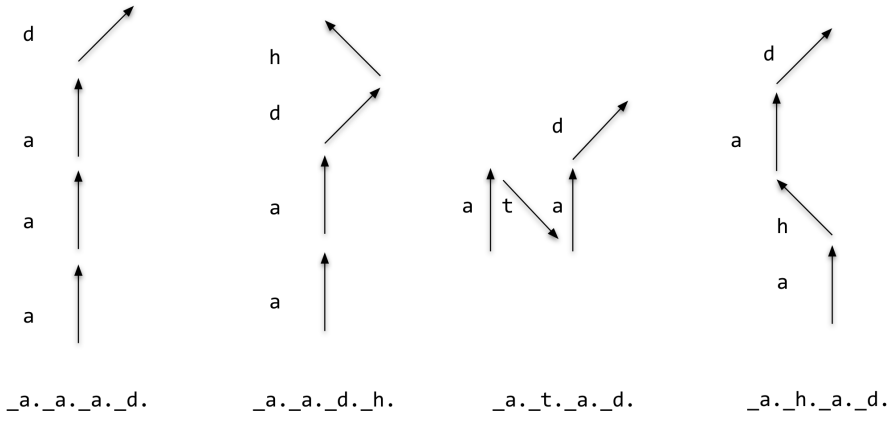
Introducción

1.1. Planteamiento del Problema

Esta investigación continúa y mejora la eficiencia y exactitud lograda en el trabajo de la tesis de maestría de Andrés Odio Vivi denominado "LaGeR: Lenguaje para descripción de gestos bidimensionales y tridimensionales" Odio-Vivi, 2015; Mata-Montero y Odio-Vivi, 2015. El problema a resolver sigue siendo en el campo de aplicaciones con interfaces gestuales. Específicamente, cómo facilitar el diseño, programación y mantenimiento de este software de manera modular, eficiente, confiable, y, sobre todo, de la forma más independiente posible del dispositivo utilizado para capturar los gestos. El objetivo principal de esa investigación fue definir e implementar un lenguaje Language for Gesture Representation (LaGeR) y un *workbench* para la representación e interpretación de gestos bidimensionales y tridimensionales mediante el cual los desarrolladores pueden definir gestos que se detecten de manera agnóstica a los dispositivos usados, es decir, sin importar el dispositivo o las Application Programming Interface (API) de por medio.

El método original para la detección de gestos con LaGeR se basa en la distancia de Damerau-Levenshtein Bard, 2007 pero tiene varios inconvenientes. Por ejemplo, la forma en la que se expanden las hileras para obtener invariabilidad de escala es — en un escenario de peor caso — muy lento, por lo que resulta en largos tiempos de ejecución bajo ciertas circunstancias. De igual manera, no es capaz de proveer invariabilidad de rotación. Además, su tolerancia a gestos imprecisos se ve limitada por el hecho de que no considera que algunos literales corresponden a direcciones más cercanas que otras.

FIGURA 1.1: Distancia entre letras de gestos



Una de las características principales de LaGeR es que representa gestos por medio de una secuencia de literales $a - z$ que corresponden a movimientos en diferentes direcciones (ver Subsección 3.3.2). El algoritmo de reconocimiento antiguo

simplemente compara la hilera de LaGer del gesto de entrada con las de gestos candidatos, pero no toma en cuenta la relación espacial real entre cada literal (ver Sección 2.4). Esto se ilustra en la Figura 1.1, donde la primera hilera corresponde al gesto de entrada y las tres hileras restantes corresponden a los gestos candidatos entre los cuáles el LaGeR Workbench original debe elegir. Para el algoritmo Damerau-Levenshtein, la tercera y la cuarta hilera tendrían la misma probabilidad de ser la respuesta correcta a pesar de que el movimiento correspondiente al literal 'h' sea más cercano al del literal 'a' que al del literal 't'.

Por lo tanto, se desea implementar y evaluar algoritmos con otros enfoques que teóricamente tienen amplio potencial tales como los basados en *deep learning*. Para tal fin, se necesitará un amplio y variado conjunto de datos de entrenamiento, lo cual implica explorar formas de generar datos de entrenamiento a gran escala (*data augmentation*).

1.2. Impacto

Actualmente, hay una proliferación de dispositivos de entrada para la captura de gestos por parte de usuarios. Por ejemplo, los Oculus Touch (xinreality, 2018), los HTC Vive controllers (HTC, 2017), el SensIR (McIntosh, Marzo & Fraser, 2017) y, más recientemente, el proyecto Soli de Google (Google, 2017). Este tipo de interfaz, es no solamente muy útil en la billonaria industria de los juegos de vídeo, sino también en aplicaciones de realidad virtual y para el apoyo a personas con discapacidad. Desde el punto de vista de ingeniería de software, no es aceptable que los desarrolladores se vean obligados a reprogramar el procesamiento de los gestos de forma diferente para cada dispositivo. La primera versión del LaGeR Workbench logró una primera aproximación exitosa. Sin embargo, para que esto sea realmente útil en un contexto práctico es necesario que el reconocimiento de dichos gestos, especialmente para gestos complejos, sea rápido y confiable. Al introducir técnicas modernas de reconocimiento de patrones como deep learning, este trabajo logra mejorar lo alcanzado por el LaGeR Workbench original.

1.3. Fundamentos

Esta tesis busca mejorar la plataforma LaGeR mediante el uso de técnicas de inteligencia artificial. Por lo tanto, es importante presentar, aunque sea de forma resumida, el marco conceptual y terminología básica relevante de la inteligencia

artificial. En el Capítulo 2 se discutirá más profundamente los conceptos introducidos en esta sección.

La inteligencia artificial (IA, o AI según sus siglas en inglés), comenzó a desarrollarse después de la Segunda Guerra Mundial. Sin embargo, no fue hasta 1956 cuándo se le otorgó el nombre por el que hoy conocemos a esta área de estudio (McCarthy, Minsky, Rochester & Shannon, 1955). A continuación presentamos algunas definiciones del campo de la IA que son relevantes para este trabajo (Russell & Norvig, 2009):

Machine learning (aprendizaje de máquina) es un área de la IA que mejora los primeros algoritmos de IA al plantear mecanismos capaces de aprender e inferir reglas sin necesidad de programar directamente esas reglas. Machine learning es ir un paso adelante de lo que era la IA en sus principios. Antes de machine learning existían retos para la IA que probaron ser demasiado difíciles, por ejemplo, porque las reglas lógicas que gobiernan muchos sistemas complejos son muy difíciles de inferir y codificar por parte de programadores.

Sin embargo, conforme ha pasado el tiempo, la cantidad de datos digitales existentes aumentó considerablemente y esto ha posibilitado otras opciones. Además el poder computacional del hardware ha crecido exponencialmente. Para el análisis inteligente de grandes cantidades de datos (*big data*), ha surgido otra rama de la IA llamada deep learning (Deng & Yu, 2014). Sus características principales son (Deng & Yu, 2014):

- Hace uso de una cascada de múltiples capas de procesamiento de unidades no lineales para extracción y transformación de características. Cada capa sucesiva usa la salida de la capa previa como entrada.
- Aprende de forma supervisada (e.g., clasificación) y/o no supervisada (e.g., analizando patrones).
- Aprende múltiples niveles de representaciones que corresponden a diferentes niveles de abstracción; los niveles forman una jerarquía de conceptos.
- Usa alguna forma de descenso de gradiente para entrenamiento vía retropropagación.

En resumen, la IA es un esfuerzo para lograr que las máquinas sean capaces de desarrollar comportamientos previamente solo exhibidos por humanos. Como resultado de estos esfuerzos, se ha logrado el avance en muchos temas tales como

los vehículos autónomos, la detección y extracción de elementos en imágenes, y el desarrollo de agentes inteligentes en videojuegos, entre otros.

Para el desarrollo de esta tesis contamos con la exigencia de que el entrenamiento de modelos de deep learning requiere de gran cantidad de datos. Nos tomaría demasiado tiempo y esfuerzo recopilar esa cantidad de datos para LaGeR, lo cual pondría en riesgo la viabilidad del estudio. Sin embargo es posible hacer modificaciones leves en los datos capturados y de esa forma aumentar el número de mediciones captadas. Para este fin haremos uso del algoritmo de Monte Carlo, que ha sido usado exitosamente en otras investigaciones, tal y como se puede corroborar en (Goodfellow, Bengio & Courville, 2016) y en (Deviant, 2011) .

1.4. Objetivos

1.4.1. Objetivo General

Definir, implementar y probar un sistema de reconocimiento de gestos basado en LaGeR que mejore la forma en la que LaGeR Workbench actualmente compara literales, para distinguir y reconocer los gestos de forma más exacta, sin impactar significativamente el tiempo de ejecución.

1.4.2. Objetivos Específicos

1. Desarrollar e implementar algoritmos de deep learning como mejoras del LaGeR workbench.

2. Comparar los algoritmos implementados con el algoritmo original de LaGeR Workbench (Damerau-Levenshtein), enfocándose en la exactitud de la detección de gestos, la invariabilidad de escala, la invariabilidad de rotación, y el tiempo de ejecución.

3. Utilizar al menos un dispositivo más de entrada para la prueba del funcionamiento de LaGeR Workbench.

1.4.3. Hipótesis

Es posible encontrar un algoritmo de deep learning para mejorar el enfoque actual de reconocimiento de gestos con LaGeR, de forma que sea más rápido y con mayor exactitud.

1.5. Estructura del documento

- **Capítulo 2:** Se presenta un resumen de conceptos fundamentales para entender la importancia y el impacto del presente trabajo. La primera sección plantea antecedentes y terminología sobre técnicas de machine learning desarrolladas antes del advenimiento de deep learning. La segunda sección se concentra en deep learning. La tercera y cuarta secciones son muy cortas, definen respectivamente los algoritmos de Márkov y el algoritmo de Damerau-Levenshtein.

- **Capítulo 3:** Se resume el trabajo relacionado en el reconocimiento de gestos usando técnicas de deep learning y modelos ocultos de Márkov. En la última sección se introduce el lenguaje LaGeR y el LaGer Workbench.

- **Capítulo 4:** Se describe la metodología utilizada para las modificaciones realizadas al LaGeR Workbench haciendo uso de algoritmos de deep learning. Además, se describen los experimentos llevados a cabo para probar el nuevo enfoque y compararlo con el original de LaGer.

- **Capítulo 5:** En este capítulo se detalla la estructura de directorios utilizada en la implementación de LaGeR deep learning. Además de esto se explica la red neuronal utilizada y como se implementó el algoritmo de Monte Carlo.

- **Capítulo 6:** Se analizan los resultados de los experimentos descritos en el capítulo 4.

- **Capítulo 7:** Este capítulo contiene las conclusiones y recomendaciones para trabajos futuros de la investigación.

Capítulo 2

Antecedentes

2.1. Machine Learning

Machine learning es un campo de estudio dentro de la IA que busca darle la habilidad a las computadoras de aprender sin especificar de forma explícita en el programa cómo llegar a ese objetivo. En el campo de machine learning, el programa tiene la capacidad de mejorar las tareas que realiza conforme va adquiriendo experiencia en dichas áreas.

A continuación algunos ejemplos de aplicaciones de machine learning que han sido exitosas:

- Aprender a reconocer palabras habladas: Los más exitosos sistemas de reconocimiento de voz emplean machine learning desde hace casi tres décadas. Por ejemplo el sistema Sphinx (Lee, Hon, Hwang, Mahajan & Reddy, 1989) permitió aprender estrategias específicas para el reconocimiento de sonidos primitivos (fonemas) y palabras de la señal de voz observada. Las redes neuronales (Waibel, Hanazawa, Hinton, Shikano & Lang, 1989) y métodos de Modelos ocultos de Markov (Lee et al., 1989) fueron efectivos personalizando para hablantes individuales, vocabularios, diferentes tipos de micrófonos, ruidos, entre otros. Más recientemente, investigadores de Microsoft (Xiong et al., 2016) y Google (Li et al., 2017) han logrado crear algoritmos basados en deep learning capaces de detectar el habla con tasas de exactitud del habla que sobrepasan la del humano promedio ($\sim$94 %). Técnicas similares tienen aplicaciones en muchas áreas de interpretación de señales.

- Aprender a conducir vehículos autónomos: Métodos de machine learning han sido utilizados para entrenar computadores de vehículos para conducir en diferentes tipos de rutas. Por ejemplo, ya desde 1989 el sistema ALVINN (Pomerleau, 1989) permitió conducir sin asistencia a una velocidad de 70 millas por hora por 90 millas de autopista pública acompañado por otros vehículos. Más recientemente, las redes neuronales convolucionales han permitido manejar vehículos autónomos con mínimo entrenamiento y valiéndose únicamente de una cámara frontal que provee píxeles al algoritmo, el cual manipula directamente el sistema de dirección del automóvil (Bojarski et al., 2016). Este tipo de tecnología ya está siendo comercializada por empresas como Tesla y Waymo. Más allá de los automóviles, técnicas similares tienen posibles aplicaciones en muchos otros sistemas de control basados en sensores.

- Aprender a clasificar nuevas estructuras astronómicas: Los métodos de machine learning también han sido aplicados a diversas bases de datos con el objetivo de aprender regularidades implícitas en esos datos. Por ejemplo, astroML es un modulo de machine learning y *data mining* de Python para el análisis de datos astronómicos. "El objetivo de astroML es proporcionar un repositorio comunitario para implementaciones rápidas de Python de herramientas y rutinas comunes utilizadas para el análisis de datos estadísticos en Astronomía y Astrofísica, para proporcionar una interfaz uniforme y fácil de usar a los conjuntos de datos astronómicos disponibles gratuitamente." (Ivezić, Connolly, Vanderplas & Gray, 2014).

- Aprender a jugar *Go* (juego milenario chino): Un juego chino cuya complejidad es muy superior al del ajedrez es ahora exitosamente ganado mediante el uso de algoritmos de machine learning. Recientemente, un grupo de ingenieros de Google creó AlphaGo, un algoritmo cuyas "...redes neuronales juegan Go a nivel de los programas de búsqueda de árboles de Monte Carlo de última generación que simulan miles de juegos aleatorios de autoaprendizaje. También se presenta un nuevo algoritmo de búsqueda que combina la simulación Monte Carlo con redes de valores y políticas. Utilizando este algoritmo de búsqueda, AlphaGo logró una tasa de ganancia del 99.8 % contra otros programas de Go, y derrotó al campeón (humano) de *European Go* por 5 juegos a 0. Esta es la primera vez que un programa de computadora ha ganado el juego Go, una hazaña que se creía que estaría al menos a una década de distancia." (Silver et al., 2016).

Se distinguen dos tipos generales de Machine Learning: aprendizaje supervisado y aprendizaje no-supervisado.

2.1.1. Aprendizaje Supervisado

En este tipo de aprendizaje, somos nosotros (los programadores), los encargados de darle al algoritmo la respuesta correcta. Por ejemplo, podríamos crear un algoritmo que asigne precios a casas basándose en el tamaño de la casa a partir de ejemplos en los cuales se asocia tamaños con precios. A este tipo de problemas le llamaremos ejercicios de regresión. Otro ejemplo seria decidir cuándo un e-mail es *spam* o no.

Dentro de los algoritmos de aprendizaje supervisado tenemos: Regresiones, Naive Bayes, Máquinas de Soporte Vectorial y Redes Neuronales, entre otros.

Es importante considerar que cuando utilizamos aprendizaje supervisado debemos tener un conjunto de datos (que se dividirá en conjunto de entrenamiento y conjunto de prueba) que serán usados para entrenar al programa y probarlo. De esa forma se entrena el algoritmo utilizando el conjunto de entrenamiento y después se corre el algoritmo con el conjunto de prueba para saber en cuáles de los puntos algoritmo funciona correctamente y en cuáles no. Normalmente el conjunto de datos está constituido por un 80 % de datos de entrenamiento y un 20 % de datos de prueba.

El algoritmo es capaz de modificar sus parámetros internos para poder generalizar los datos y de esa manera encontrar las respuestas a posibles futuras preguntas. El conjunto de datos de prueba se separa para ver cuán bien se ha generalizado y de esa forma verificar que no ocurra *overfitting* (sobreadjuste).

A continuación se explica el funcionamiento de algunos algoritmos de machine learning supervisados.

Regresiones Lineales

Este tipo de algoritmos lo que busca es modelar la relación entre una variable dependiente Y y variables independientes X. Si solo poseemos una variable independiente X, tendremos una regresión lineal simple (e.g., si queremos aproximar el precio de casas (Y) sabiendo el tamaño de las mismas (X)).

También se puede dar el caso en el que tengamos varias variables independientes. Por ejemplo, queremos aproximar el precio de las casas, pero ahora además de saber el tamaño, también sabemos: el número de cuartos, la calidad de los materiales utilizados en el diseño y el tipo de diseño, entre otros.

El nombre de regresiones lineales se debe a que nos permiten "predecir" el comportamiento de una variable por medio de funciones lineales.

Nuestro objetivo será encontrar una función gradiente $H(X) = a + bX$ (encontrar a y b), que describa de la mejor manera posible el comportamiento del conjunto de entrenamiento que nosotros le vamos a dar al algoritmo. Necesitamos elegir los valores a y b, de manera que obtengamos un valor mínimo de $[H(X)-Y]^2$.

Al proceso de ir optimizando iterativamente los valores de los coeficientes (a y b) para minimizar el error del modelo en nuestro conjunto de entrenamiento se le denomina *gradient descent* (descenso de gradiente). Se dan valores aleatorios a los coeficientes al principio. La suma de los errores al cuadrado se calcula para cada par de valores de entrada y de salida. Como valor de escala se utiliza un ritmo de aprendizaje denominado *learning rate* y en cada ocasión los coeficientes son

actualizados de forma que se minimice el error. Este proceso se repite hasta que el error alcanzado por la suma de los cuadráticos sea mínimo (Brownlee, 2016b).

Regresión Logística

En algunos casos no es posible utilizar regresiones lineales. Consideremos un ejemplo en el que el conjunto de entrenamiento es el salario de las personas y si estas aplican o no para un préstamo basado únicamente en su salario. Al ser las posibilidades verdadero o falso (1 ó 0), no existe una función lineal que se ajuste a lo antes descrito.

En regresión logística vamos a tratar de calcular la probabilidad de que un ejemplo dado pertenezca a la clase verdadero (1) o a la clase falso (0). Para esto utilizaremos la función sigmoide:

$$P(y = 1|x) = h_\theta(x) = \frac{1}{1+exp(-\theta^\top x)} \equiv \sigma(\theta^\top x),$$
$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - h_\theta(x).$$

La función sigmoide es una función con forma de 'S', que trata de buscar el valor $\theta^T x$ para el cual y se mueve de 0 a 1. Esto es similar a cuando en la regresión lineal buscamos el valor de a y b. La función sigmoide tiene un valor de 0 a 1.

Clasificadores Bayesianos Ingenuos (*Naive Bayes*)

Con este enfoque se permite manejar datos con una gran cantidad de dimensiones. El algoritmo consiste en tener un conjunto de entrenamiento tal que, al recibir un nuevo elemento en el sistema, este sea capaz de clasificarlo tomando en cuenta la probabilidad de que ocurran sus vecinos. Por ejemplo, queremos clasificar si un elemento es rojo o amarillo, y el elemento está rodeado de cuatro vecinos rojos y solo uno amarillo. En este caso, al calcular la probabilidad de que el elemento sea rojo versus la probabilidad de que sea amarillo, obtenemos una mayor probabilidad para rojo y clasificamos el elemento como tal. Naives Bayes tiene como ventaja que es un algoritmo sencillo de entender y de implementar. Sin embargo, se basa únicamente en la cantidad de vecinos de un tipo u otro y esta metodología no es aplicable a todos los casos.

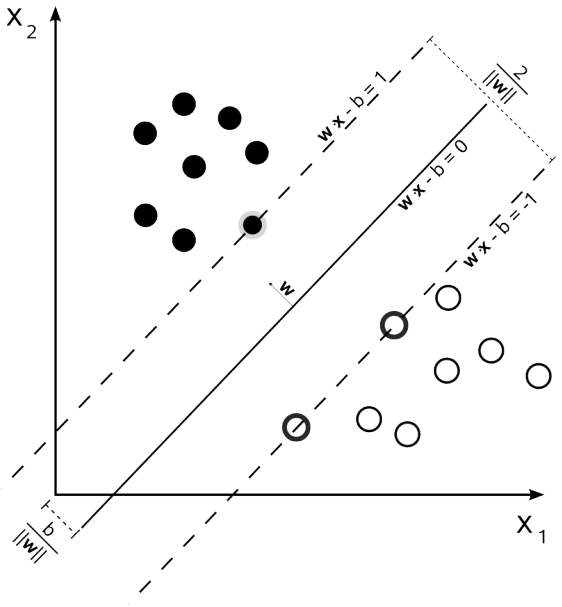
Máquinas de Soporte Vectorial (*Support Vector Machines*)

SVM es un método de machine learning supervisado que cuenta con la ventaja de que se puede implementar para un número arbitrario de dimensiones. Consiste en medir la distancia mínima entre los diferentes grupos o cúmulos del conjunto de entrenamiento, buscando maximizar el "margen" entre ellos a través de una

función *kernel* (semilla) que los traslada a un espacio de mayor dimensionalidad. Por ejemplo, en el caso de que un conjunto de datos no pueda ser claramente definido en dos dimensiones, se acude a una tercera para de esta forma tener más información de los datos y así poderlos separar en distintas clasificaciones.

La figura 2.1 muestra gráficamente el funcionamiento del algoritmo.

FIGURA 2.1: Descripción gráfica de SVM (Wikipedia, s.f.-d)



Este algoritmo es sumamente útil para clasificar datos que no pueden ser divididos por métodos más sencillos, pero es complejo de utilizar e implementar, con tiempos de ejecución relativamente largos.

Redes Neuronales Artificiales

"Las redes neuronales artificiales (ANN, por sus siglas en inglés) proporcionan un método general y práctico para el aprendizaje de funciones con valores reales, valores discretos y valores de vectores a partir de ejemplos. Algoritmos tales como *backpropagation* (retropropagación) usan una gradiente descendente para ajustar los parámetros de red para mejor adaptar un conjunto de entrenamiento de pares de entrada-salida. El aprendizaje ANN es robusto a los errores en los datos de entrenamiento y se ha aplicado con éxito desde hace muchos años a problemas tales como la interpretación de escenas visuales, el reconocimiento de voz y el aprendizaje de estrategias de control de robots." (Mitchell, 1997)

Las redes neuronales interconectan capas de nodos de forma análoga a las conexiones sinápticas entre neuronas del cerebro humano. El investigador establece parámetros de interconexión iniciales y le provee datos de entrenamiento para que

se ajuste automáticamente a ellos. De forma similar al cerebro humano, no es fácil establecer qué rol juega cada parte de la red final, pero el agregado suele ser un clasificador muy efectivo.

Los algoritmos de ANN se han utilizado para resolver problemas complejos de visión por computador y reconocimiento de voz, que eran difíciles de resolver por métodos de programación convencionales como los que hemos descrito en las secciones anteriores. Otra ventaja que presentan es que, aunque requieren de muchos datos de entrenamiento, una vez entrenados son muy rápidas de ejecutar.

2.1.2. Aprendizaje no supervisado

En este tipo de aprendizaje no tenemos un conjunto de datos previamente etiquetado con las respuestas correctas que el algoritmo debería dar. En el aprendizaje no-supervisado lo que tenemos son solamente "datos"sin ningún tipo de etiquetas. Por lo tanto, lo que se busca es encontrar patrones, *clusters* o distancias entre los datos, que nos ayuden a clasificar cualquier tipo de información nueva.

Entre los ejemplos utilizados para aprendizaje no-supervisado tenemos: *k-means clustering*, *agrupamiento jerárquico* y *Análisis de Componentes Principales (ACP)*. Google News es un ejemplo conocido que se basa fuertemente en este tipo de técnicas. Estas también son usadas con frecuencia para la clasificación de genes, segmentación de mercados y redes sociales.

K-means Clustering

Es un tipo de aprendizaje no supervisado que consiste en agrupar los datos en k *clusters*. Por ejemplo, si tenemos un conjunto de elementos que son alimentos y queremos clasificarlos como proteínicos, grasosos o harinosos, en este caso tendríamos $k = 3$.

Basado en ese k, el algoritmo elige k puntos iniciales en el espacio de datos y calcula su centroide (distancia Euclidiana) respecto a los datos más cercanos. La ubicación de esos nuevos puntos se va a ir depurando en cada iteración de manera que se encuentre lo más céntrico posible respecto a los datos que lo rodean. El error va disminuyendo en cada iteración hasta que finalmente el *cluster* quede formado y tengamos la clasificación de los datos. No es inusual que el algoritmo no converja, por lo que a menudo se corre con un límite máximo de iteraciones.

K-means es un algoritmo pesado y difícil de escalar para grandes cantidades de datos (NP-hard). Sin embargo, es ampliamente utilizado en áreas como segmentación de mercados, visión por computador, geoestadística, astronomía y minería de datos en agricultura (Dabbura, 2018).

Agrupamiento Jerárquico

A diferencia de K-means, el agrupamiento jerárquico busca no solo agrupar sino también establecer jerarquías entre los grupos. Los resultados usualmente son presentados en un dendograma (representación gráfica de datos en forma de árbol).

La complejidad de estos algoritmos generalmente es de $O(n^3)$, lo cual hace que este algoritmo no sea ideal cuando se tienen grandes cantidades de datos por analizar.

Análisis de Componentes Principales

El PCA (por sus siglas en inglés), es una técnica estadística utilizada para disminuir la dimensionalidad de un conjunto de datos.

"El ACP construye una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje (llamado el Primer Componente Principal), la segunda varianza más grande es el segundo eje, y así sucesivamente. Para construir esta transformación lineal debe construirse primero la matriz de covarianza o matriz de coeficientes de correlación. Debido a la simetría de esta matriz existe una base completa de vectores propios de la misma. La transformación que lleva de las antiguas coordenadas a las coordenadas de la nueva base es precisamente la transformación lineal necesaria para reducir la dimensionalidad de datos. Además las coordenadas en la nueva base dan la composición en factores subyacentes de los datos iniciales.

Una de las ventajas del ACP para reducir la dimensionalidad de un grupo de datos, es que retiene aquellas características del conjunto de datos que contribuyen más a su varianza, manteniendo un orden de bajo nivel de los componentes principales e ignorando los de alto nivel. El objetivo es que esos componentes de bajo orden a veces contienen el aspecto "más importante" de esa información." (Wikipedia, s.f.-a)

2.2. Deep Learning

La caja de herramientas de machine learning ha ido evolucionando con los años, haciéndose cada vez más efectiva. En la actualidad existen metodologías que permiten analizar enormes cantidades de datos para obtener clasificadores muy detallados y poderosos. El análisis de datos por medio de redes neuronales con muchas capas y unidades neuronales es la implementación más usual usada en deep learning. Algunos autores lo ven como una sub-rama de machine learning ideal para analizar grandes cantidades de datos, mientras que otros lo consideran como una rama independiente.

Para el presente trabajo utilizamos deep learning (aprendizaje profundo) para optimizar el funcionamiento de LaGeR. La cantidad de herramientas aptas para implementar deep learning en este contexto es relativamente limitada, debido a que esta metodología está diseñada para aplicaciones muy específicas donde el conjunto de datos disponible es muy amplio. Ejemplo de lo anterior son la visión por computadora, la traducción automática, los vehículos autónomos y el subtitulado de videos en tiempo real. Afortunadamente, la mayoría de los *frameworks* existentes para deep learning son de código abierto, debido a su origen en la academia y en empresas de tecnología (e.g., Google) comprometidas tanto con la difusión del conocimiento como con su aplicación práctica:

Según Parvat et al (Parvat, Chavan, Kadam, Dev & Pathak, 2017) "Comparado a los algoritmos tradicionales de machine learning, los modelos de deep learning son capaces de proveer una mejora significativa en áreas como reconocimiento del habla y traducciones de lenguaje, lo cual queda evidenciado por la mejora significativa de Google Translate después de cambiar Phrase Based Machine Translation (PBMT) por Neural Machine Translation (NMT)" .

Deep learning está inspirado en el funcionamiento de las redes neuronales biológicas. En la sección anterior se explicó brevemente el funcionamiento de los algoritmos de redes neuronales artificiales supervisados. Sin embargo, las redes neuronales se pueden utilizar en ambos ambientes (supervisados y no supervisados). En el caso específico de deep learning, se utilizan redes neuronales con un alto número de capas ocultas (de ahí su "profundidad"). Esto significa que el número de parámetros a afinar es mayor, por lo que deep learning requiere de grandes cantidades de datos de entrenamiento. En nuestro caso, haremos uso de un algoritmo de Monte Carlo para engrosar nuestro conjunto de entrenamiento con suficientes datos como para poder utilizar deep learning en el contexto de reconocer gestos representados por LaGeR.

2.2.1. Arquitecturas de Deep Learning

Existe una amplia gama de clasificaciones para las arquitecturas existentes de deep learning. Como se describió anteriormente, el aprendizaje profundo se refiere a un conjunto de técnicas y arquitecturas de aprendizaje automático. A grandes rasgos, las podemos clasificar en tres clases principales: arquitecturas profundas generativas, arquitecturas profundas discriminatorias y arquitecturas profundas híbridas.

Arquitecturas Generativas

"Son arquitecturas que están destinadas a caracterizar las propiedades de correlación de alto orden de los datos observados o visibles para análisis de patrones o propósitos de síntesis, y/o caracterizar las distribuciones estadísticas conjuntas de los datos visibles y sus clases asociadas. En este último caso, el uso de la regla de Bayes puede convertir este tipo de arquitectura en una discriminatoria." (Deng, 2014).

Arquitecturas Basadas en Discriminatorios

Estas arquitecturas buscan proveer poder discriminatorio para la clasificación de patrones. En (Morgan, 2012) se pueden encontrar algunos modelos discriminatorios existentes para el reconocimiento de voz. Para trabajos más recientes en esta área, se pueden analizar (Pinto, Garimella, Magimai-Doss, Hermansky & Bourlard, 2011) y (Ketabdar & Bourlard, 2010).

Arquitecturas Híbridas Generativas-Discriminativas

El objetivo de este tipo de arquitecturas es la discriminación, pero asistida por las arquitecturas generativas para optimizarla. En resumen, es un enfoque que apalanca los otras dos. En (Hinton & Salakhutdinov, 2006; Mohamed, Yu & Deng, 2010; Dahl, Yu, Deng & Acero, 2011; Sainath, Kingsbury & Ramabhadran, 2012) se pueden ver algunos ejemplos de este tipo de arquitecturas.

2.3. Algoritmo de Monte Carlo

Para poder implementar un reconocedor de LaGeR utilizando técnicas de deep learning, necesitamos un conjunto de datos bastante grande. Dado que nuestro conjunto de entrenamiento es limitado, vamos a utilizar la técnica de Monte Carlo. Esta juega un papel importante pues, como se indica en (Schuck, s.f.): "La idea del análisis de Monte-Carlo es la generación de un gran número (por ejemplo,

100 - 1000) conjuntos de datos sintéticos que son similares al conjunto de datos experimentales, pero cada uno con diferente por el ruido aleatorio distribuido de manera uniforme. Cada uno de estos nuevos conjuntos de datos se analiza y las distribuciones se almacenan. El conjunto resultante de distribuciones se puede estudiar, punto por punto, y se pueden calcular los contornos de la media y la probabilidad".

2.4. Algoritmo Damerau-Levenshtein

Este algoritmo fue la base del workbench original LaGeR y fue la parte que cambiamos para introducir los algoritmos de deep learning. Este algoritmo esta basado en la llamada *distancia Damerau-Levenshtein* que se refiere al número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Debido a que el lenguaje LaGeR esta compuesto por hileras, el LaGeR Workbench original basado en Damerau-Levenshtein fue utilizado para comparar las hileras capturadas versus hileras previamente almacenadas (hileras canónicas). "Dicho algoritmo ha sido utilizado exitosamente para diversas aplicaciones tales como la corrección ortográfica (Bard, 2007)y la medición de variaciones en ADN (Majorek et al., 2014). En nuestro caso, el tipo de operaciones tomadas en cuenta por Damerau-Levenshtein se adecuó satisfactoriamente a las diferencias que pueden existir entre una hilera almacenada y una producida por un ser humano sosteniendo un sensor." (Odio-Vivi, 2015)

Capítulo 3

Trabajo relacionado

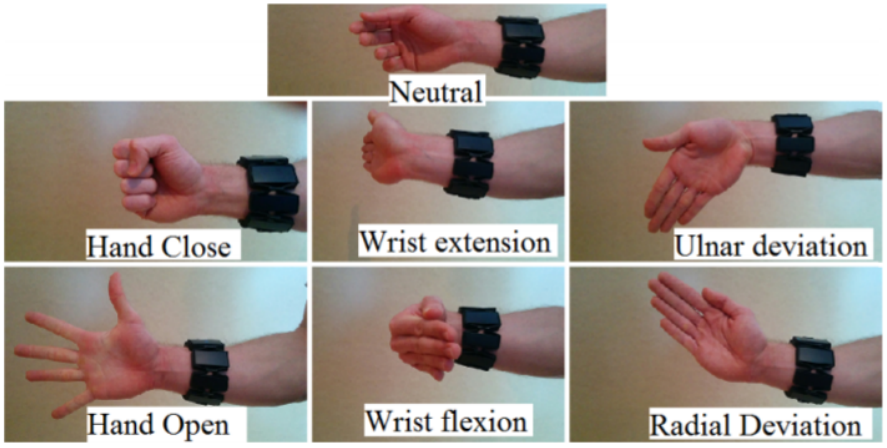
3.1. Deep Learning

Es importante hacer referencia a los trabajos previos que se han efectuado para el reconocimiento de gestos haciendo uso de deep learning. Un ejemplo de ellos es (Neverova, Wolf, Taylor & Nebout, 2015). En dicho artículo se muestra una estrategia que fue presentada en el "ChaLearn 2014, Looking at People Challenge" en el área de reconocimiento de gestos donde obtuvieron el primer lugar de 17 equipos que presentaron.

En (Côté Allard et al., 2017) se muestra otra investigación muy interesante donde se utilizan redes neuronales convolucionales para reconocer gestos de una mano por medio de sMEG, un procedimiento no-invasivo que detecta e interpreta la actividad eléctrica en los músculos. Mediante la información recibida por ese mecanismo y con el uso de algoritmos de redes neuronales, dicho proyecto presenta una propuesta efectiva de reconocimiento de patrones en una pulsera con sensores.

La figura 3.1 muestra los gestos que es capaz de reconocer el algoritmo creado por (Côté Allard et al., 2017). Si bien el objetivo de los proyectos es similar, a través de LaGeR se pretende crear un sistema que sea capaz de hacer uso de diferente tipos de dispositivos, en tanto el proyecto mencionado en (Côté Allard et al., 2017) se centra únicamente en dispositivos sMEG.

FIGURA 3.1: Gestos manuales de estudio con sEMG(Côté Allard et al., 2017)



3.2. Modelos Ocultos de Márkov

En la bibliografía se pueden encontrar bastantes ejemplos de usos de *HMM* para el reconocimiento de gestos (Dahl et al., 2011). Si bien esta tesis se limita al uso de técnicas de deep learning para mejorar el LaGer Workbench, es importante mencionar el enfoque HMM como una posibilidad de trbajo futuro.

Desde hace más de dos décadas se utilizan HMM para el reconocimiento de gestos, como es el caso de (Yang & Xu, 1994), donde se creó un modelo multidimensional de Márkov para tal fin. En ese caso, se convirtieron los gestos en símbolos geométricos que fueron utilizados por HMM para entrenar su sistema. De esa forma el algoritmo fue capaz de encontrar la mayor similitud para reconocerlos con una exactitud del 99.78 %, un logro notable especialmente si se toma en cuenta la época.

Existen varios otros trabajos relevantes que se han llevado a cabo en esta área. Por ejemplo, en (Williamson, s.f.) se puede ver una lista amplia de *papers* que han buscado implementar reconocimiento de gestos mediante el uso de HMM, sin embargo ninguno de ellos se ha centrado en hacerlo para diferentes dispositivos de entrada como es el caso de LaGeR.

3.3. La Tesis LaGeR

Esta sección es un resumen del capítulo 4 de la tesis que originó la presente investigación (Odio-Vivi, 2015). Como se ha descrito en capítulos anteriores, el presente trabajo esta basado en la oportunidad de mejorar los resultados obtenidos en esa tesis, en la cual se usó el algoritmo *Damerau-Levenshtein* para crear

un programa capaz de reconocer los gestos representados por medio del lenguaje LaGeR. "LaGeR es un lenguaje regular, pero se presenta una Gramática Libre de Contexto (GLC) para que sea más compacta y fácil de leer. En particular, se describe cómo se enfrenta la invariabilidad de escala en los movimientos, la tolerancia a gestos imprecisos, y la invariabilidad rotativa de gestos cerrados." (Odio-Vivi, 2015).

Nuestro objetivo fue mejorar lo obtenido por medio del algoritmo *Damerau-Levenshtein* haciendo uso de algoritmos de deep learning. Para entender mejor el contexto de dicho trabajo el presente capítulo se dedicará a explicar el lenguaje LaGeR.

3.3.1. LaGeR

LaGeR es el nombre del lenguaje definido en la tesis *LaGeR: Lenguaje para descripción de gestos bidimensionales y tridimensionales* que tenía como objetivo: "la representación de gestos producto de sensores de entrada bidimensionales y tridimensionales." (Odio-Vivi, 2015). En el mismo se hizo uso de datos dados por medio de funciones de la biblioteca VRPN para obtener las coordenadas cartesianas (x, y, z) de los sensores de forma periódica y así obtener una secuencia de vectores de movimiento con magnitud 1.

Por ejemplo, (0, 0, 0) $\rightarrow$ (1, 0, 0) $\rightarrow$ (1, 1, 0) "correspondería a un vector de magnitud 1 hacia la derecha, seguido por un vector de magnitud 1 hacia adelante.' (Odio-Vivi, 2015)

Conceptualmente, el lenguaje LaGeR hace uso de las 26 caras del poliedro regular llamado "rumbicuboctaedro" (Figura 3.2) para representar los vectores de movimiento tridimensional en intervalos de 45°.

Cada una de estas 26 direcciones se puede representar por medio de coordenadas esféricas $\{r, \theta, \phi\}$ donde r es el radio, θ el ángulo polar, y ϕ el ángulo azimutal (Figura 3.3).

3.3.1.1. Literales de movimiento

Las coordenadas antes descritas se convierten a la más cercana de las 26 direcciones predefinidas. A su vez, cada dirección corresponde a una de las 26 letras $a-z$ del abecedario inglés. En el Apéndice A de (Odio-Vivi, 2015) se encuentra una tabla con dicha asignación, la cual observamos de forma gráfica en las Figuras 3.4 y 3.5.

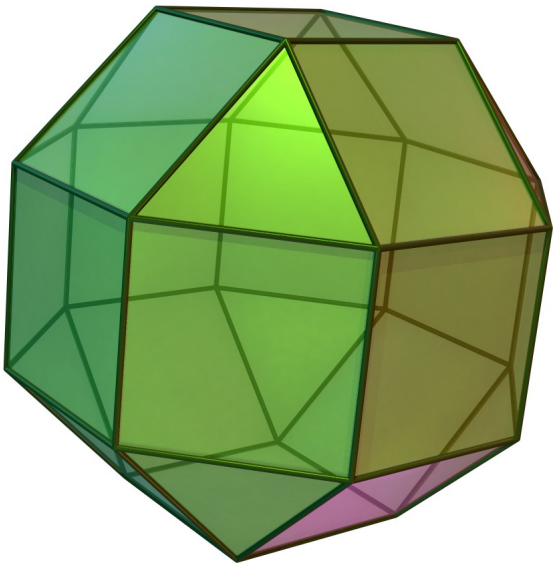


FIGURA 3.2: Rombicuboctaedro (Wikipedia, s.f.-b)

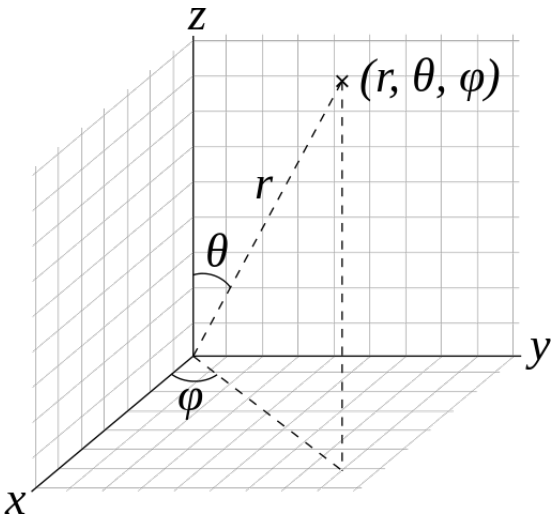


FIGURA 3.3: Coordenadas esféricas (Wikipedia, s.f.-c)

3.3.1.2. Literales de agrupamiento e inmovilidad

Los literales de movimiento se complementan por dos literales especiales:

- Literal de agrupamiento ('.'): Agrupa los movimientos simultáneos de uno o más sensores.

  Por ejemplo, para llevar a cabo el gesto *pinch to zoom* se mueven dos sensores simultáneamente en direcciones opuestas (llamémoslas 'p' y 'l'), lo cual se puede expresar por medio de la hilera 'pl.pl.pl.pl.pl.pl.pl.pl.' (ver Figura 3.6).

- Literal de inmovilidad ('_'): Representa la inmovilidad de un sensor en un intervalo de tiempo.

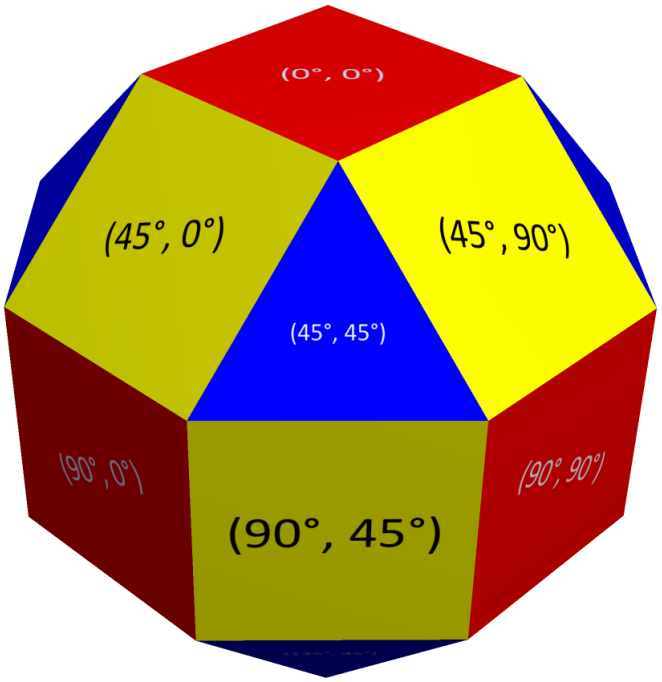


FIGURA 3.4: Coordenadas esféricas en caras de rombicuboctaedro (Odio-Vivi, 2015)

Por ejemplo, si deseamos representar un movimiento con el sensor izquierdo del Razer Hydra (sin mover el sensor derecho), lo podemos hacer con la hilera 'l_.n_.n_.' (ver Figura 3.7)

3.3.2. Sintaxis y Semántica de LaGeR

La sintaxis de LaGeR se puede representar por medio de una GLC compacta que se detalla en (Odio-Vivi, 2015).

En cuanto a la semántica, cada $< gesto >$ está compuesto de una o más instancias de $< grupo >$, que a su vez se componen de uno o más $< movimiento >$ simultáneos en una de las 26 direcciones del rombicuboctaedro. Para más detalles, ver (Odio-Vivi, 2015).

3.3.3. Aspectos pragmáticos

Hay tres aspectos importantes que se contemplan en (Odio-Vivi, 2015):

- Escala: LaGeR es un lenguaje que diferencia entre gestos grandes y pequeños por medio de la repetición de literales de movimiento (ver Figura 3.8).

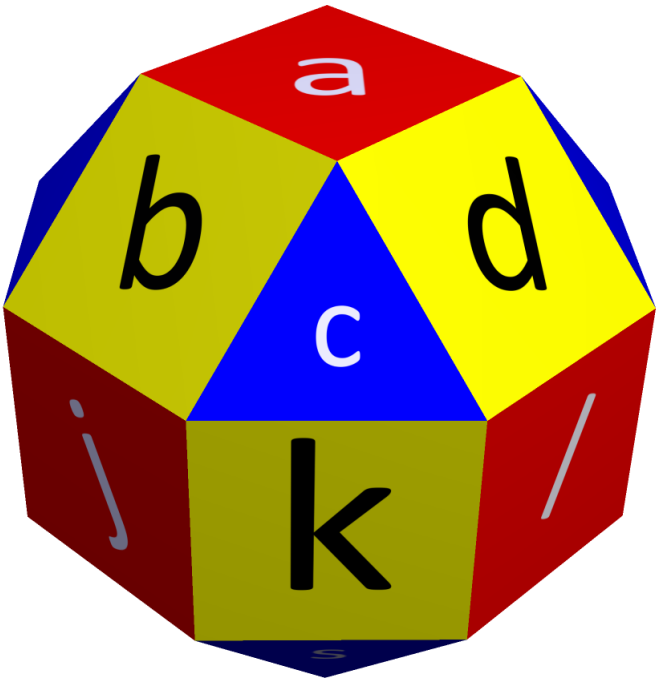


FIGURA 3.5: Literales de LaGeR en caras de rombicuboctaedro (Odio-Vivi, 2015)

FIGURA 3.6: *Pinch-to-zoom* representado por la hilera pl.pl.pl.pl.pl.pl.pl.pl. (Odio-Vivi, 2015)

En caso de que se desee reconocer de manera equivalente gestos que con la misma forma pero tamaños distintos, el ajuste se debe hacer a nivel del algoritmo de reconocimiento. En el caso del algoritmo Damerau-Levenshtein, el reconocimiento se ralentiza puesto que se deben expandir las hileras a un mínimo común múltiplo antes de compararlas (Odio-Vivi, 2015). En la presente investigación, todas las hileras se normalizan a una longitud estándar y la velocidad de reconocimiento con deep learning no se ve afectada.

- Imprecisión de gestos: LaGeR representa los gestos de entrada con la mayor fidelidad posible. Esto significa que errores en los movimientos hechos por el usuario pueden resultar en hileras diferentes a su representación ideal (ver Figura 3.9).

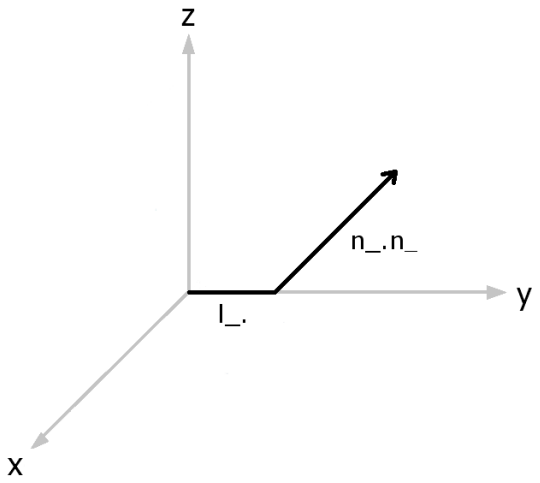


FIGURA 3.7: Flecha representada por la hilera l_.n_.n_. (Odio-Vivi, 2015)

En caso de que se desee crear un sistema de reconocimiento que tolere gestos imprecisos, la responsabilidad recae sobre el algoritmo de reconocimiento, no sobre el lenguaje. En (Odio-Vivi, 2015), esto se logra por medio de un umbral de distancia Damerau-Levenshtein entre la hilera del gesto de entrada y las hileras de los gestos candidatos. En la presente investigación se logra por medio de un algoritmo de deep learning que se entrena con múltiples ejemplos de cada gesto para lograr que generalice su clasificación.

- Orden de movimientos: El lenguaje LaGeR representa una secuencia de movimientos, no la imagen de un gesto completo. Esto significa que si existen múltiples formas de "dibujar" una misma figura, LaGeR va a representar cada una de ellas con hileras diferentes (ver Figura 3.10).

  En caso de que se desee reconocer gestos equivalentes sin importar el orden de los movimientos, esto recae sobre el algoritmo de reconocimiento. Mientras que el método utilizado en (Odio-Vivi, 2015) no tiene esa propiedad (e.g., invariabilidad de rotación), nuestro algoritmo de deep learning tiene un mayor grado de tolerancia. La razón es que la red neuronal considera el aporte de cada movimiento en la hilera de entrada de una forma que no le da tanta importancia a su orden en la misma.

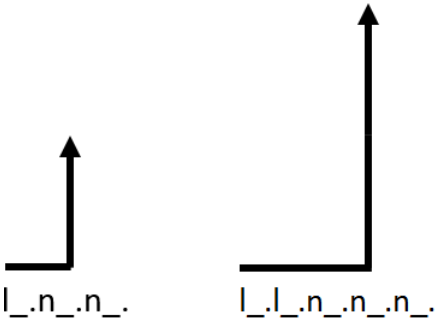


FIGURA 3.8: Gestos con escalas diferentes (Odio-Vivi, 2015)

FIGURA 3.9: Hilera de gesto almacenado vs. hilera de gesto impreciso de entrada (Odio-Vivi, 2015)

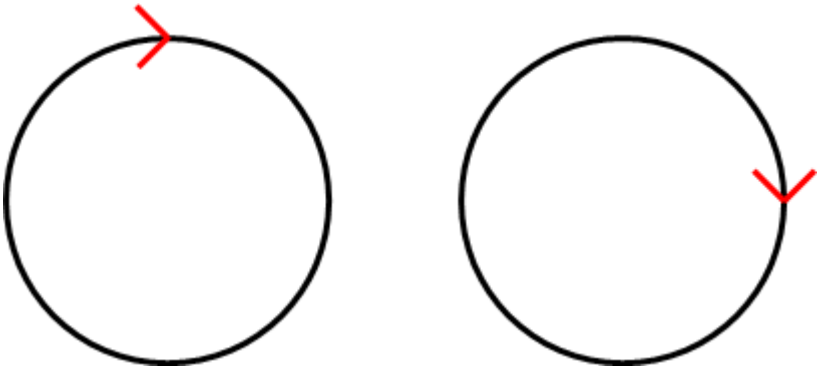


FIGURA 3.10: Gestos con trazos distintos (Odio-Vivi, 2015)

Capítulo 4

Marco Metodológico

4.1. Generalidades

La metodología consistió en primero que todo crear un conjunto de entrenamiento de gestos vasto, para poder hacer uso de los algoritmos de deep learning. Esto se debe a que los algoritmos de deep learning necesitan de una gran cantidad de datos para funcionar adecuadamente.

Seguidamente se creó un programa en Python para reconocimiento de gestos en LaGeR haciendo uso de algoritmos de deep learning. Para este fin se utilizó el framework "TensorFlow", el cuál provee APIs mediante los cuáles se facilita hacer uso de redes neuronales para el reconocimiento de imágenes.

Al principio se dificultó el reconocimiento de gestos LaGeR, ya que las hileras LaGeR distan de lo que comúnmente se procesa con TensorFlow. Sin embargo, variando el factor de aprendizaje, el número de neuronas y el número de capas, se logró el reconocimiento de los gestos.

Paralelamente a esto, se trabajó para mejorar el Workbench de LaGeR ya existente para comprobar su factibilidad en cuanto a utilizar diferentes tipos de dispositivos de entrada. Los dispositivos que utilizamos fueron: Razer Hydra, Leap Motion, Wacom Intuos, XBox Controller y *mouse*.

A continuación se detallan los dispositivos y *frameworks* utilizados para la elaboración de esta tesis.

4.1.1. Software

4.1.1.1. *Frameworks* de Deep Learning

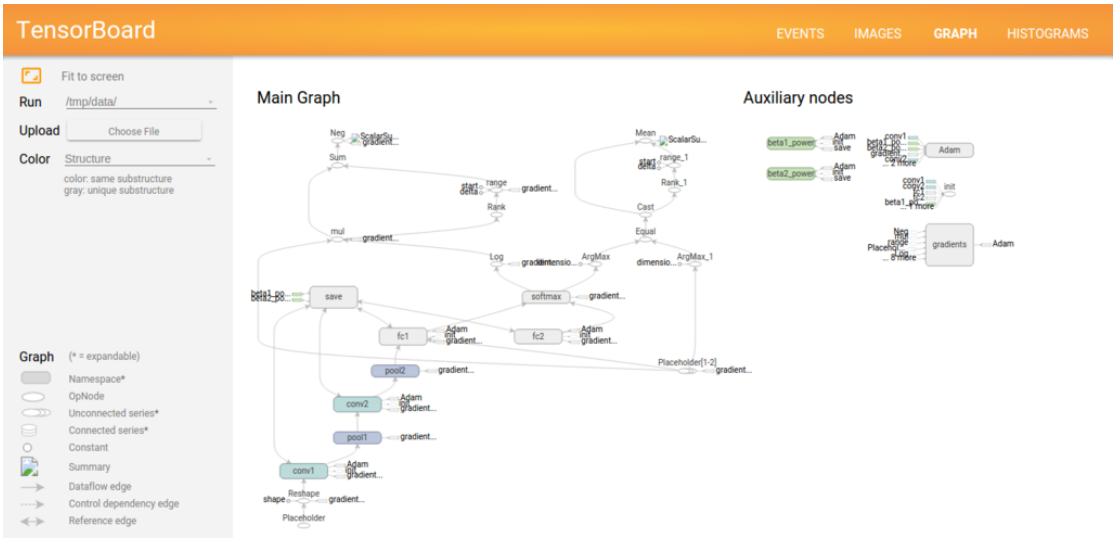
En la siguiente sección repasaremos algunos de los *frameworks* de código abierto existentes para deep learning.

TensorFlow

"TensorFlow fue desarrollado originalmente por investigadores e ingenieros que trabajaron en el equipo de Google Brain dentro de la organización de investigación de Inteligencia Artificial de Google con el objetivo de realizar investigaciones de aprendizaje automático y redes neuronales profundas y con el tiempo se ha descubierto que el framework es lo suficientemente general para ser aplicable en un una amplia variedad de otros dominios también. TensorFlow se implementa en Python y utiliza gráficos de flujo de datos para el cálculo numérico. En comparación con otros frameworks de deep learning, se puede ver que se considera

que TensorFlow es actualmente el framework de código abierto mejor documentado disponible. TensorFlow también hace alarde de contar con un front-end fácil de usar y modular en términos de arquitectura. [...] Además, TensorFlow también contiene TensorBoard, que es un conjunto de herramientas de visualización que facilita la comprensión, depuración y optimización de los programas que se ejecutan con el código TensorFlow." Parvat et al., 2017

FIGURA 4.1: TensorBoard: Visualizador de TensorFlow



Keras

Keras es un API para redes neuronales; está escrito en Python y provee capas de abstracción que facilitan el uso de otros *frameworks* como CNTK, Theano o TensorFlow. En nuestro caso se utilizó como una librería para facilitar el desarrollo con TensorFlow.

4.1.2. Hardware

Además del hardware utilizado en la tesis de LaGeR, esta tesis introdujo la utilización del **Leap Motion**, que es un dispositivo elaborado por *Leap Motion Inc.* que posee sensores capaces de reconocer los movimiento de las manos y dedos sin la necesidad de tocar el dispositivo. En el 2016 la compañía lanzó una nueva versión del producto para ser utilizada en realidad virtual. En la Figura 4.2 se muestra una fotografía del Leap Motion (dispositivo pequeño que está delante del teclado), así como una mano en la posición típica de uso para demostrar su funcionamiento.

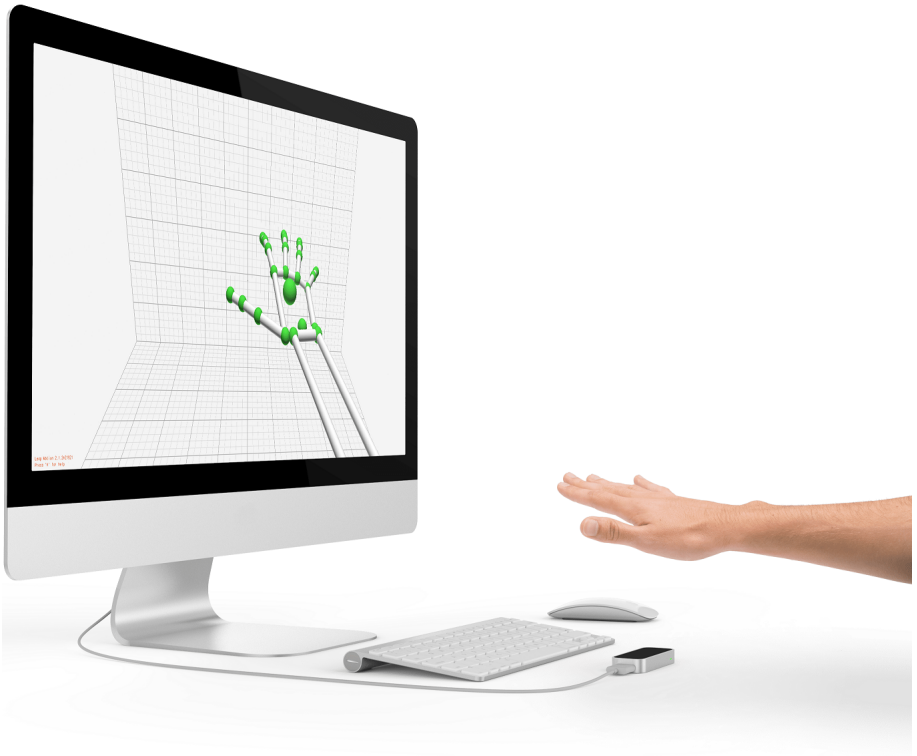


FIGURA 4.2: Leap Motion (Hardware.Info, 2011)

El **Wacom Intuos Pen and Touch** (otro de los dispositivos utilizados para la tesis) es un dispositivo tipo "tableta" que funciona con un lápiz digital inalámbrico pasivo que permite reconocer la posición de su punta así como también la variación en la presión del dispositivo. Gracias a ello permite que los lápices empleados sean ligeros y fáciles de manipular. La tecnología se basa en resonancia electromagnética. La base de la tarjeta contiene unos emisores de campo electromagnético, mientras que el lápiz posee un circuito resonante que oscila según el campo electromagnético que recibe. En la Figura 4.3 se puede observar el dispositivo Wacom utilizado.

El siguiente dispositivo utilizado fue un **Xbox 360 Controller**, que es el control estándar desarrollado para la consola Xbox 360 de Microsoft, y posteriormente lanzado para su uso en computadoras personales.

Además, también se hizo uso de un *mouse* de computadora para trazar gestos con el cursor.

Finalmente, al igual que en la tesis original de LaGeR, la mayoría de los experimentos se elaboraron haciendo uso de un Razer Hydra (Figuras 4.5 y 4.6).

FIGURA 4.3: Wacom (WebAntics, 2018)

FIGURA 4.4: Xbox Controller (Wikipedia, s.f.-b)

4.2. Componentes

En primer lugar, deseábamos definir, implementar y probar un sistema de reconocimiento de gestos basado en LaGeR y comprobar que mejora la forma en la que LaGeR Workbench comparaba literales. Para ello analizamos la literatura para encontrar un algoritmo que pudiéramos reaprovechar o utilizar como base para

FIGURA 4.5: Razer Hydra (Hardware.Info, 2011)

FIGURA 4.6: Los dos sensores del Razer Hydra en las manos de un usuario. (Castle, 2011)

definir un reconocedor nuevo que represente gestos formados a partir de esos movimientos. Luego analizamos algoritmos de deep learning utilizados normalmente en el reconocimiento de imágenes e implementamos un sistema para tratar las hileras de LaGeR de forma similar a como se trataban las imágenes en los algoritmos antes mencionados.

Seguidamente implementamos los siguientes componentes con el fin de cumplir los objetivos mencionados en el Capítulo 1 del presente documento:

FIGURA 4.7: Bobinas en la base del Razer Hydra (Matthews, 2013)

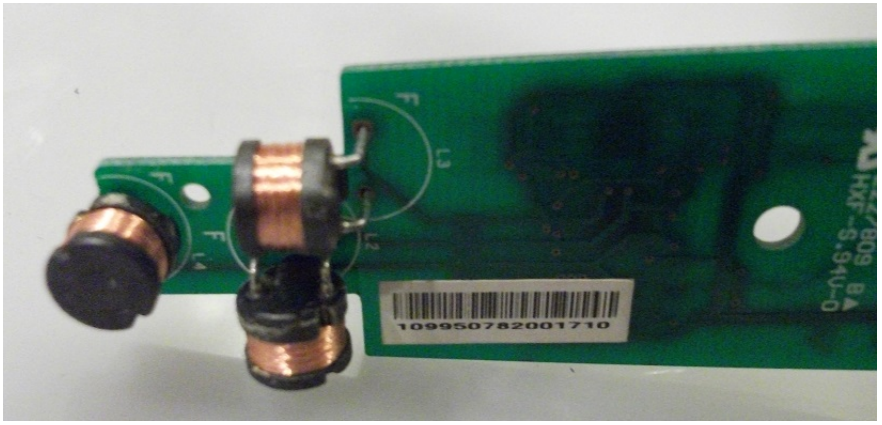


FIGURA 4.8: Bobinas en un sensor del Razer Hydra (WiredEarp, 2011)

1. Creamos un programa en Python (*lager_generator*) que toma de entrada hileras LaGeR y las modifica para lograr multiplicar el número de ejemplos de cada gesto disponibles para entrenar la red neuronal. Para ello nos basamos en el algoritmo de Monte Carlo descrito en el Capítulo 2 de este documento. Esto es necesario debido a que como se menciona en ese mismo capítulo, los algoritmos de deep learning requieren una cantidad considerable de datos para funcionar de forma adecuada.

2. lager_calibrator: Creamos este utilitario para calibrar los diferentes sensores de manera tal que la magnitud de sus movimientos obtenidos a través de Open Source Virtual Reality (OSVR) sea consistente — es decir, que las unidades del sistema de coordenadas sean equivalentes en relación al rango

de movimiento de cada dispositivo). Esto es importante para que el liblager_convert traduzca los movimientos de los sensores a hileras de tamaños comparables.

3. lager_ml: Creamos una red neuronal con Keras y TensorFlow para clasificar gestos de entrada descritos en LaGeR. Además creamos *scripts* en Python para expandir gestos de entrenamiento al tamaño de entrada de la red y convertirlos en arreglos numéricos.

4. Remplazo de VRPN por OSVR: El lager_converter original se enlazaba con el *middleware* VRPN, el cual provee "una interfaz transparente a la red entre programas de aplicaciones y el conjunto de dispositivos físicos (rastreador, etc.) utilizados en un sistema de realidad virtual" (Taylor II, Yang, Weber & Hudson, 2014). Dado que VRPN es poco flexible y extensible, modificamos nuestro código para utilizar las APIs del moderno OSVR y así poder soportar nuevos dispositivos de entrada. Las ventajas de OSVR se detallan en la Subsección5.2.1.

4.3. Experimentos

Llevamos a cabo pruebas con un conjunto predeterminado de gestos por medio de cinco dispositivos de entrada (Leap Motion, Razer Hydra, *mouse*, control de Xbox y tableta Wacom). El *lager_recognizer* se modificó de manera tal que cada vez que detecta que se hizo un gesto con un sensor invoca tanto al reconocedor antiguo basado en el algoritmo Damerau-Levenshtein, como el nuevo reconocedor basado en deep learning. En ambos casos se imprime el tiempo de ejecución y porcentaje de correspondencia de los gestos candidatos (cuán "seguro" esta el algoritmo de que el candidato corresponde al gesto de entrada).

Para comparar el desempeño del nuevo algoritmo, reclutamos seis personas para llevar a cabo una serie de tareas básicas de navegación web utilizando nuestro sistema. A continuación una descripción de los experimentos realizados para la presente tesis:

4.3.1. Experimento 1

Objetivo: Comparar el algoritmo utilizado en la tesis del LaGeR, Damerau-Levenshtein (en adelante llamado .ªlgoritmo D-L"), con el algoritmo de deep learning desarrollado en la presente tesis, en cuanto a exactitud y tiempo de ejecución

para detectar gestos usando los 10 gestos descritos en Mata-Montero y Odio-Vivi, 2015 y el dispositivo de entrada Razer Hydra. Con ello atendimos los objetivos 1 y 2 de esta tesis.

Pasos:

1. Se definió una red neural de dos capas ocultas con 100 neuronas cada una por medio de Keras y TensorFlow. Esta cumple con la definición de redes neuronales profundas ya que "Cuando una [red neuronal artificial] tiene dos o más capas ocultas, se le conoce como una red neuronal profunda" (Géron, 2018). Detalles de la implementación se presentan en la Sección 5.5.

2. Se siguió el enfoque de aprendizaje supervisado y se entrenó el sistema de la siguiente forma:

   a) **Etapa de entrenamiento:** Para esta etapa se hizo uso de el algoritmo de Monte Carlo, descrito en en Capítulo 2 del presente trabajo para generar los datos suficientes para entrenar el sistema. Se utilizaron 3/4 de los datos generados para entrenar el sistema y 1/4 para validar el modelo. En el Capítulo 5, Sección 5.4, el Algoritmo 1 detalla el proceso seguido para esta etapa.

   b) **Etapa de pruebas:** A cada sujeto S_j ($j \in \{1, 6\}$) se le presentó la tarea de operar el navegador web Google Chrome llevando a cabo la siguiente secuencia de gestos G_i ($i \in \{1, 8\}$):

      1) G_1: Abrir Google Chrome (trazar C mayúscula)
      2) G_2: Abrir nueva pestaña (trazar T mayúscula)
      3) G_3: Cargar CNN (trazar CNN en mayúsculas)
      4) G_4: *Zoom in* (apartar sensores horizontalmente)
      5) G_5: *Zoom out* (juntar sensores horizontalmente)
      6) G_6: Refrescar página (trazar círculo en dirección horaria)
      7) $G_{6'}$: Refrescar página con rotación inversa (trazar círculo en dirección anti-horaria)
      8) G_7: Cerrar pestaña (trazar X mayúscula)
      9) G_2: Abrir nueva pestaña (trazar T mayúscula)
      10) G_8: Cargar Google (trazar G mayúscula)
      11) G_7: Cerrar pestaña (trazar X mayúscula)
      12) G_7: Cerrar pestaña (trazar X mayúscula)

Cada usuario debió realizar la secuencia antes descrita dos veces con el dispositivo Razer Hydra, variando los movimientos de las siguiente manera:

1) G_i de tamaño "grande"
2) G_i de tamaño "pequeño"
   Para cada caso, contabilizamos la precisión y el tiempo de ejecución para cada uno de los algoritmos: el de D-L y el de deep learning. Luego capturamos los resultados para cada sujeto S_j.

4.3.2. Experimento 2

Objetivo: Demostrar que Lager Workbench también puede utilizar al menos dos dispositivos de entrada más, los cuales denominamos D_2 y D_3. Medimos la precisión y tiempo de ejecución para detectar gestos usando los 8 gestos descritos en Odio-Vivi, 2015. Dicho experimento atiende el objetivo 3 de esta tesis y además permite comparar los algoritmos D-L y de deep learning en nuevos contextos.

Pasos:

1. **Etapa de entrenamiento:** Se usó la misma red creada para el Experimento 1, por lo que no se hizo más entrenamiento.

2. **Etapa de entrenamiento:** Se realizó la misma etapa de pruebas del Experimento 1, pero utilizando los dispositivos D_2 y D_3 en vez del Razer Hydra.

4.3.3. Experimento 3

A pesar de que con el Experimento 1 y Experimento 2 se atendieron los tres objetivos específicos de esta tesis, se desarrolló un experimento adicional.

Objetivo: Medir, al menos de manera preliminar, otro aspecto importante de un algoritmo de identificación de gestos: la escalabilidad con respecto al número de gestos en el repertorio de gestos a identificar.

Pasos:

1. Se aumentó el número de gestos y se midió la exactitud y tiempo de ejecución. Solamente se utilizó el dispositivo Razer Hydra. El repertorio incluye los ocho gestos previamente definidos, más los siguientes ocho nuevos gestos:

   - G_9: Maximizar la ventana (mover dos sensores hacia arriba y hacia afuera simultáneamente)

- G_{10}: Restaurar el tamaño original de la ventana (mover dos sensores hacia adentro y hacia abajo simultáneamente)
- G_{11}: Abrir Google Play Music (trazar M mayúscula)
- G_{12}: Abrir Netflix (trazar N mayúscula)
- G_{13}: Abrir YouTube (trazar Y mayúscula)
- G_{14}: Abrir el pronóstico del tiempo (trazar W mayúscula)
- G_{15}: Hacer *scroll* hacia abajo (trazar línea hacia arriba)
- G_{16}: Hacer *scroll* hacia arriba (trazar línea hacia abajo)

2. **Etapa de entrenamiento:** Se utilizó la misma arquitectura de red neuronal de los experimentos anteriores pero la red se entrenó, desde cero, con el nuevo repertorio de gestos G_i, donde $1 \leqslant i \leqslant 16$ y siguiendo los pasos descritos en la etapa de entrenamiento del Experimento 1.

3. **Etapa de pruebas:** Se siguieron los mismos pasos del Experimento 1, pero con el nuevo repertorio de gestos G_i, donde $1 \leqslant i \leqslant 16$. Además, con el fin de acotar la duración de la tesis al cronograma acordado, se realizó con solamente dos sujetos a cuyos gestos se les aplicó Monte-Carlo como técnica de data augmentation.

Capítulo 5

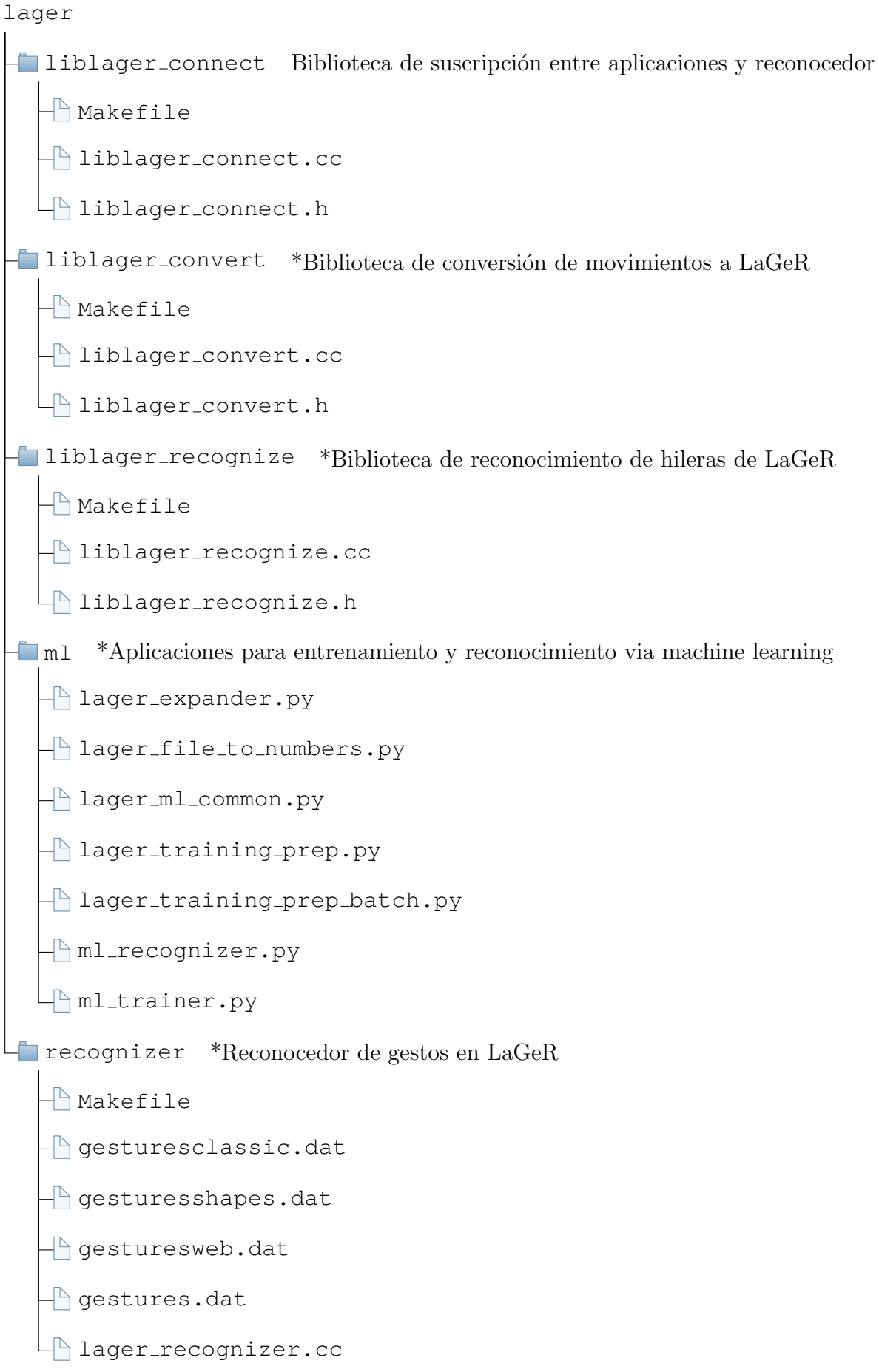
Implementación

5.1. Estructura del código de LaGeR Workbench

Tal y como mencionamos en el Capítulo 4, la presente tesis se implementó modificando el código original del LaGeR Workbench para crear un generador de gestos, añadir un reconocedor basado en deep learning y soportar dispositivos de entrada adicionales. El LaGeR Workbench consiste de código en C++ y Python estructurado de la siguiente manera:

(Las principales modificaciones se marcan con asterisco.)

- `lager`
  - `calibrator` *Calibra escala de conversión a literales según el dispositivo
    - `Makefile`
    - `lager_calibrator.cc`
  - `common` Bibliotecas para mapeo de coordenadas a hileras de LaGeR
    - `coordinates_letter.h`
    - `letter_coordinates.h`
    - `spherical_coordinates.h`
    - `string_tokenizer.h`
  - `gesture_manager` Utilitario para manejo de archivos de gestos
    - `Makefile`
    - `gestures.dat`
    - `lager_gesture_manager.cc`
  - `injector` Aplicación que se suscribe a gestos e inyecta comandos al sistema
    - `Makefile`
    - `lager_injector.cc`
  - `lager_generator` *Generador de gestos basado en Monte Carlo
    - `lager_generator.py`


lager
liblager_connect Biblioteca de suscripción entre aplicaciones y reconocedor
Makefile
liblager_connect.cc
liblager_connect.h
liblager_convert *Biblioteca de conversión de movimientos a LaGeR
Makefile
liblager_convert.cc
liblager_convert.h
liblager_recognize *Biblioteca de reconocimiento de hileras de LaGeR
Makefile
liblager_recognize.cc
liblager_recognize.h
ml *Aplicaciones para entrenamiento y reconocimiento via machine learning
lager_expander.py
lager_file_to_numbers.py
lager_ml_common.py
lager_training_prep.py
lager_training_prep_batch.py
ml_recognizer.py
ml_trainer.py
recognizer *Reconocedor de gestos en LaGeR
Makefile
gesturesclassic.dat
gesturesshapes.dat
gesturesweb.dat
gestures.dat
lager_recognizer.cc

- lager
  - viewer *Visualizador tridimensional de gestos en LaGeR
    - common
    - controls.cc
    - controls.h
    - shader.cc
    - shader.h
    - texture.cc
    - texture.h
    - external
    - src
      - TextureFragmentShader.fragmentshader
      - TransformVertexShader.vertexshader
      - gesture.dat
      - lager_viewer.cc
      - lager_viewer.cc
      - uvtemplate.DDS
    - CMakeLists.txt
  - Doxyfile Configuración de auto-generador de documentación
  - LICENSE Licencia MIT
  - README.md Guía rápida
  - buildall Utilitario para construir el proyecto
  - dependencies.txt Instrucciones para configurar dependencias externas

5.2. Middlewares para Dispositivos de Entrada

5.2.1. Modificaciones Generales

El LaGeR Workbench original fue creado con el objetivo de soportar varios dispositivos de entrada. Si bien el lenguaje LaGeR es agnóstico al dispositivo,

liblager_convert – el convertidor de movimientos de sensor a literales de LaGeR – dependía de VRPN, un middleware poco flexible y con un soporte limitado para dispositivos más recientes.

El principal cambio que permitió soportar dispositivos adicionales en esta investigación fue el uso de OSVR. OSVR es un middleware que se basa en VRPN y provee dos ventajas significativas:

1. Soporta plugins, lo cual permite extender su funcionalidad a nuevos dispositivos.

2. Provee una capa de abstracción semántica descrita en JavaScript Object Notation (JSON) por encima de los módulos de OSVR y VRPN.

Lo primero es muy valioso puesto que el Leap Motion inicialmente no era soportado por OSVR ni por VRPN, pero logramos encontrar un plugin de OSVR que añadió el soporte.

Lo segundo fue fundamental para lograr que el liblager_convert en sí fuera agnóstico al dispositivo de entrada, ya que nos permitió representar los diferentes dispositivos de entrada por medio de mapeos semánticos consistentes:

"OSVR mantiene un "árbol de ruta" – similar a un URL o ruta de sistema de archivos – en el cual todos los datos de sensado y representación se hacen disponibles. Los aliases se configuran en el servidor para esencialmente redireccionar desde una ruta semántica (una ruta con un nombre significativo) de vuelta a los detalles de hardware específicos al sistema. [...] Esto permite casos de uso donde el hardware [...] es modificado durante la operación sin ningún impacto al desarrollador de aplicaciones." (Boger & Pavlik, 2015)

Por ejemplo, liblager_convert solo tiene que suscribirse a notificaciones de OSVR para "/me/hands/left" y "/me/hands/right/, y la configuración de OSVR se encarga de ligar cada "mano" a los rastreadores correspondientes de cada dispositivo de entrada (e.g., movimientos del mouse, manos con el Leap Motion, bastones del Razer Hydra, lápiz del Wacom y bastones del Xbox Controller).

De la misma forma, "/controller/left/1" y "/controller/right/1" representan los botones principales de los dispositivos (e.g., botones del mouse, Razer Hydra, lápiz del Wacom y Xbox Controller) mientras que "/controller/left/trigger" y "/controller/right/trigger" representan los gatillos analógicos (e.g., pinchazo de dedos / cierre de puños con el Leap Motion, gatillos del Xbox Controller).

A continuación vemos el código de suscripción vía mapeo semántico que se utilizó en liblager_convert, junto con la configuración de OSVR correspondiente para el Xbox Controller:

[liblager_convert]

```
void LagerConverter::InitializeTrackers() {
// Initialize the tracker handlers

left_tracker_ = context_.getInterface("/me/hands/left");
right_tracker_ = context_.getInterface("/me/hands/right");

left_tracker_.registerCallback(&HandleTrackerChangeLeft, NULL);
right_tracker_.registerCallback(&HandleTrackerChangeRight, NULL);

if (use_buttons_) {
    // Initialize the button and pinch handlers
    left_button_ = context_.getInterface("/controller/left/1");
    right_button_ = context_.getInterface("/controller/right/1");

left_button_.registerCallback(&HandleButtonChangeLeft, NULL);
right_button_.registerCallback(&HandleButtonChangeRight, NULL);
```

[osvr_server_config.json]

```
"externalDevices": {
    "/Xbox_360_LeftTracker": {
        "deviceName": "Tracker0",
        "server": "localhost:3884",
        "descriptor": {
            "interfaces": {
                "tracker": {
                    "count": 1
                }
            },
            "automaticAliases": {
                "/me/hands/left": "/Xbox_360_LeftTracker/tracker/0"
            }
        }
    },
    "/Xbox_360_RightTracker": {
        "deviceName": "Tracker1",
        "server": "localhost:3884",
        "descriptor": {
            "interfaces": {
                "tracker": {
                    "count": 1
                }
            },
            "automaticAliases": {
                "/me/hands/right": "/Xbox_360_RightTracker/tracker/0"
            }
        }
    },
    "/Xbox_360_Buttons": {
```

```
        "deviceName": "Joylin0",
        "server": "localhost:3884",
        "descriptor": {
            "interfaces": {
                "button": {
                    "count": 14
                }
            },
            "automaticAliases": {
                "/controller/left/1": "/Xbox_360_Buttons/button/4",
                "/controller/right/1": "/Xbox_360_Buttons/button/5"
            }
        }
    }
},
```

5.2.2. Configuración para Leap Motion

Para configurar el soporte para el Leap Motion, los pasos a seguir fueron:

1. Descargar e instalar el SDK de Leap Motion para Linux
2. Descargar e instalar el plugin de OSVR para Leap Motion
3. Iniciar el servicio de sistema de Leap Motion
4. Iniciar el servidor de OSVR

Para más detalles, favor consultar el Apéndice A.

5.2.3. Configuración para Mouse

Para configurar el soporte para el mouse, los pasos a seguir fueron:

1. Obtener el nombre del dispositivo de entrada correspondiente de Linux
2. Asociarlo a un dispositivo ligado a un *tracker* en la configuración de VRPN
3. Añadir un dispositivo correspondiente en la configuración de OSVR
4. Iniciar el servidor de VRPN
5. Iniciar el servidor de OSVR

Para más detalles, favor consultar el Apéndice A.

5.2.4. Configuración para Razer Hydra

Para configurar el soporte para el Razer Hydra, los pasos a seguir fueron:

1. Iniciar el servidor de OSVR

(No fue necesario configurarlo ya que OSVR soporta el Razer Hydra por defecto.)

5.2.5. Configuración para Wacom

Para configurar el soporte para tabletas Wacom, los pasos a seguir fueron:

1. Obtener el nombre del dispositivo de entrada correspondiente de Linux
2. Asociarlo a un dispositivo ligado a un *tracker* en la configuración de VRPN
3. Añadir un dispositivo correspondiente en la configuración de OSVR
4. Iniciar el servidor de VRPN
5. Iniciar el servidor de OSVR

Para más detalles, favor consultar el Apéndice A.

5.2.6. Configuración para XBox Controller

Para configurar el soporte para controles de Xbox, los pasos a seguir fueron:

1. Descargar e instalar el driver del control de Xbox para Linux
2. Cargar el driver y obtener el nombre del dispositivo de Linux correspondiente
3. Asociarlo a un dispositivo ligado a un *tracker* en la configuración de VRPN
4. Añadir un dispositivo correspondiente en la configuración de OSVR
5. Iniciar el servidor de VRPN
6. Iniciar el servidor de OSVR

Para más detalles, favor consultar el Apéndice A.

5.3. Calibrador de sensores

Como vimos anteriormente, el uso del middleware OSVR permite que liblager_convert se refiera a diferentes dispositivos de entrada a través de mapeos semánticos consistentes. Sin embargo, eso no resuelve la problemática de que cada dispositivo genera coordenadas con escalas diferentes, por lo que el convertidor podría llegar a generar hileras de tamaños muy distintos según el dispositivo que se utilice.

Para resolver esto, creamos un programa llamado lager_calibrator. El mismo permite que el usuario defina los bordes de su "área de trabajo" moviendo el dispositivo de lado a lado mientras presiona su botón principal. El número de unidades de esa distancia se convierte a un factor de escala que luego es utilizado por lager_convert para que el número de literales producido guarde la misma proporción para cada dispositivo. De esta manera, nos aseguramos de que el LaGeR Workbench sea agnóstico al dispositivo de entrada sin tener valores de escala predefinidos para cada uno.

5.4. Generador de Gestos Basado en Monte Carlo

El LaGeR Workbench original utilizó un reconocedor que requería pocos gestos canónicos para su funcionamiento, los cuales se crearon manualmente. Sin embargo, ese enfoque no iba a ser factible para esta investigación puesto que se requerirían miles de ejemplos para entrenar el nuevo clasificador basado en deep learning.

Eso se solucionó escribiendo lager_generator, un programa en Python que genera nuevas hileras (en este caso, 500) a partir de una hilera original en la cual se van remplazando literales de movimiento por sus vecinos en el espacio tridimensional. Esto se hace siguiendo una distribución estadística (enfoque de Monte Carlo). El proceso se formaliza en el Algoritmo 1.

5.5. Reconocedor basado en Deep Learning

Para crear el nuevo reconocedor nos basamos en el enfoque de la tesis original de LaGeR: traducir el dominio del problema. En Odio-Vivi, 2015, se convirtieron los movimientos a hileras para compararlas con un algoritmo comúnmente aplicado a la corrección ortográfica. De forma análoga, en esta tesis convertimos los literales de

Algoritmo 1 Algoritmo de generación de variantes de gestos en LaGeR

Require: Hilera de LaGeR para gesto a ser variado

1: **for all** Caracteres c_i en la hilera **do**
2: **if** $c_i \neq$ '_' **and** $c_i \notin \{a - z\}$ **then**
3: **continue** (solo se procesan literales de LaGeR)
4: **end if**
5: Obtener un valor aleatorio x de una distribución normal con $\mu = 5$, $\sigma = 1$
6: **if** $c_i \neq$ '_' **and** $|x - \mu| < \sigma$ **then**
7: **continue** (no se modifica el caracter)
8: **else if** $c_i =$ '_' **and** $|x - \mu| < 2\sigma$ **then**
9: **continue** (no se modifica el caracter)
10: **end if**
11: Remplazar c_i por uno de sus vecinos v_i en el rombicuboctaedro, elegido al azar. En el caso del caracter '_', todos los 26 literales de movimiento de LaGeR se consideran "vecinos".
12: **end for**
13: **return** Nueva hilera de LaGeR modificada

LaGeR a "píxeles" para alimentarlos a un tipo red neuronal comúnmente utilizado para clasificar imágenes.

5.5.1. Creación de la red neuronal

El nuevo reconocedor se escribió en Python utilizando las bibliotecas estándar de TensorFlow. Nuestro punto de partida fue una red neuronal profunda utilizada para clasificar imágenes de dígitos escritos a mano de la base de datos Modified National Institute of Standards and Technology (MNIST) (LLC, 2017). Dicha red toma una serie de imágenes cuadradas pequeñas y trata sus $n \times n$ píxeles como una matriz donde cada número corresponde al valor en escala de grises de cada píxel (ver Figura 5.1). Finalmente, la matriz se convierte a una sola hilera de n^2 números.

El resultado fue una red neuronal con dos capas ocultas de cien neuronas cada una, un *learning_rate* (tasa de aprendizaje) de 0.05, 1000 *epochs* (pasos) y un *batch_size* (tamaño de lotes) de 30.

Uno de los problemas de esta implementación es que el código era verboso y difícil de seguir. Por ello decidimos mantener el concepto original pero reescribir la implementación basándonos en (Authors, 2018), donde se utiliza la biblioteca tf.keras para crear un clasificador de imágenes de prendas de vestir. El resultado fue un código más compacto sobre el cual pudimos iterar más fácilmente.

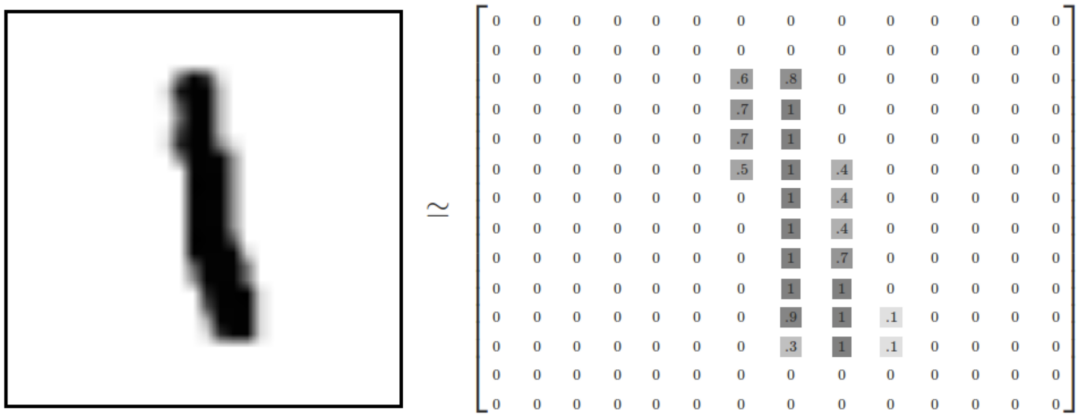


FIGURA 5.1: Representación de imágenes de dígitos

La capa de entrada era demasiado grande (784 neuronas para imágenes de $28X28 = 784$ píxeles), y además solo manejaba entradas unidimensionales (la escala de gris de cada píxel). En el caso de LaGeR, queríamos optimizar la red para las longitudes de hilera más comunes y además poder representar al menos dos dimensiones por movimiento ("pixel"), correspondientes a los movimientos de dos sensores.

Analizamos las hileras resultantes para el gesto más largo de nuestro conjunto experimental ("OpenCNN"), y vimos que rondaba un máximo de 200 movimientos. Añadimos un factor extra de 10 % y redondeamos a 224, un número cercano que es divisible por una potencia de 2 ($224 = 32/7$). Esto se basa en el hecho de que TensorFlow aprovecha el procesamiento paralelo del GPU, y los GPUs pueden operar más eficientemente con datos en potencias de 2 (Josh, 2012).

Luego, nos inspiramos en redes neuronales para el reconocimiento de imágenes a color (con tres canales RGB) para redefinir la entrada de nuestra red en términos de tuplas bidimensionales. Esto nos permitió representar cada movimiento con una tupla donde la primera dimensión representa el movimiento del sensor izquierdo, y la segunda dimensión representa el movimiento del sensor derecho.

5.5.2. Afinamiento de la red neuronal

Para afinar los hiperparámetros de la red neuronal, partimos de las redes de ejemplo antes mencionadas y seguimos un proceso sistemático de prueba y error:

1. Realizar gestos de los experimentos con un dispositivo de entrada

2. En caso de una clasificación errónea, examinar su representación gráfica por medio de lager_viewer para determinar si fue error del usuario (e.g. gesto

mal realizado), del dispositivo (e.g. imprecisión excesiva para detectar el comienzo, fin, o camino de los trazos) o del algoritmo (se clasificó mal un gesto fiel a su forma correcta).

3. En caso de que el error haya sido atribuible al algoritmo, anotar su hilera de LaGeR en un archivo de errores

4. Al final de la secuencia de gestos, modificar el hiperparámetro en cuestión, re-entrenar el clasificador y re-evaluar las hileras de gestos antes anotadas

5. En caso de mejora, mantener las modificaciones y continuar el proceso. En caso contrario, deshacer el cambio, realizar otra modificación y repetir el proceso.

Este proceso fue utilizado para afinar cada aspecto de la red en orden de lo más grueso a lo más fino:

1. Número y tipo de capas

2. Número de neuronas por capa

3. Optimizador

4. Learning rate

5. Epochs

A continuación vemos la arquitectura básica del clasificador de 8 gestos, descrito según el comando model.summary() de TensorFlow Keras:

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 448)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               44900
_________________________________________________________________
dense_2 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 808
=================================================================
Total params: 55,808
Trainable params: 55,808
Non-trainable params: 0
```

Y aquí tenemos la arquitectura básica del clasificador de 16 gestos:

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 448)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               44900
_________________________________________________________________
dense_2 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_3 (Dense)              (None, 16)                1616
=================================================================
Total params: 56,616
Trainable params: 56,616
Non-trainable params: 0
```

En ambos casos, la primera capa sirve para aplanar la entrada de 224 movimientos bidimensionales a $224*2 = 448$ salidas que son conectadas densamente con el resto de la red. Si bien las capas posteriores podrían recibir entradas bidimensionales, decidimos aplanar al comienzo de la red puesto que de lo contrario el número de parámetros entrenables crecía notablemente sin ninguna mejora aparente en la clasificación.

La información multidimensional sí se aprovecharía en capas convolucionales (Karpathy, 2018), pero decidimos utilizar capas densas puesto que consideramos que tanta complejidad no era necesaria para nuestro clasificador. Por ejemplo, el uso de capas densas fue suficiente para clasificar de forma muy exitosa imágenes de dígitos escritos a mano de 24 X 24 píxeles, mientras que nuestra entrada de 224 movimientos es equivalente a "imágenes" de tan solo 15 X 15 "píxeles". También experimentamos con el uso de capas "dropout"que desechan información aleatoriamente para reducir las probabilidades de overfitting (Brownlee, 2016a), pero no vimos un beneficio inmediato.

Para dichas capas, empezamos con la función de activación ReLu (Rectified Linear Unit). En la actualidad, ReLu es la función de activación más utilizada para deep learning (Sharma, 2017) ya que es muy sencilla y resulta en una convergencia múltiples veces más rápida que otras ya que no se satura (Krizhevsky, Sutskever & Hinton, 2017). Existe un riesgo de que parte de la red "muera" (neuronas con peso cero) en caso de gradientes muy grandes, pero esto se puede evitar con learning rates bajos (Karpathy, 2018), como en nuestro caso. De todas formas experimentamos brevemente con otras funciones de activación (sigmoid y tanh), pero el desempeño empeoró y decidimos mantener ReLu.

En cuanto al número de neuronas en las capas ocultas, empezamos con 100 en cada una ya que es lo que se utilizaba en el clasificador de dígitos. Luego de

probar con números significativamente más pequeños, vimos que la red perdía la capacidad de distinguir gestos consistentemente, probablemente porque no contaba con suficientes parámetros para capturar toda la complejidad del problema. Finalmente, decidimos dejar la cifra en 100 y no subir más para evitar el riesgo de overfitting.

Para la capa de salida, utilizamos un número de neuronas igual al número de gestos a clasificar (8 o 16) y la función de activación softmax. Esto nos permitió generar una salida adecuada para la función de pérdida sparse_categorical_crossentropy. Esta es la función recomendada para clasificación entre múltiples clases, ya que produce un arreglo donde cada clase recibe una probabilidad y todas suman a 1 (Zhang, 2018).

En cuanto al optimizador, inicialmente utilizamos Adam (Adaptive Moment Estimation), el cual se utilizó en el ejemplo de clasificación de dígitos. Adam es muy moderno y es eficiente para entrenar redes complejas con grandes cantidades de datos ruidosos (Ruder, 2016). Curiosamente, el mismo autor lleva a cabo un análisis comparativo de optimizadores que concluye que el SGD (Stochastic Gradient Descent) se está utilizando "en muchos papers recientes y usualmente logra encontrar un mínimo, pero le puede tardar significativamente más tiempo que con algunos de los otros optimizadores", mientras que se recomienda el uso de optimizadores adaptativos para "convergencia rápida y entrenar redes neuronales profundas o complejas" (Ruder, 2016).

Dado que nuestra red se entrena muy rápidamente, decidimos utilizar SGD junto con un learning rate bajo de 0.001. Esto también bajó el riesgo de usar de ReLu como función de activación de las capas ocultas, como mencionamos anteriormente.

Por último, como vimos que rápidamente convergía con alta exactitud y baja pérdida, redujimos el número de epochs a solo 25 para reducir la posibilidad de overfitting.

5.5.3. Entrenamiento de la red neuronal

5.5.3.1. Proceso

En la Figura 5.2 podemos observar el flujo completo para la creación del conjunto de entrenamiento que alimenta nuestra red neuronal.

Para cada gesto a ser entrenado, se hacen 10 repeticiones manualmente y se guardan en un archivo. Luego se invoca lager_training_prep_batch.py y se ejecuta la secuencia de la figura para cada archivo:

1. Se toma una hilera y se convierte a números a través de la función convert_lager_to_numbers(), la cual provee lager_ml_common.py. Esta convierte cada literal a un número de 0 a 26, donde 0 corresponde al literal de inmovilidad "_" y el $1 - 26$ a los literales $a - z$.

2. El paso anterior permite invocar lager_expander.py, el cual invoca la función expand_gesture_num_to_target() provista por lager_ml_common.py para expandir el arreglo de números como si se tratara de una imagen. Esto se logra por medio de la biblioteca skimage (scikit-image), y simplifica y acelera notablemente el proceso de redimensionar las hileras al tamaño deseado. Anteriormente, el LaGeR Workbench "expandía" las hileras de entrada a nivel de caracter (e.g., a_.b_. →a_.a_.b_.b_.) al MCM (mínimo común múltiplo) del de tamaño de las hileras canónicas. En casos donde el MCM era muy grande (e.g., números primos), esto causaba expansiones excesivas y ralentizaba todo el resto del proceso. En cambio, el nuevo método nos permite redimensionar todas las hileras a un único tamaño con un llamado a función sencillo.

3. Se convierten las hileras expandidas (o rara vez, encogidas) de vuelta a letras por medio de la función convert_numbers_to_lager(), provista por lager_ml_common.py.

4. El paso anterior nos permite utilizar lager_generator.py (descrito en la Sección 5.4) para generar un gran número de variantes (en este caso, 500) por cada una de las 10 hileras de ejemplo del gesto.

5. Por último, las variantes se convierten de vuelta a números con la función onvert_lager_to_numbers() y se almacenan con su respectiva etiqueta en un archivo junto con los demás ejemplos del conjunto de entrenamiento.

Al final del proceso, obtenemos un archivo de texto donde cada línea corresponde a un ejemplo etiquetado para cada gesto a ser entrenado. Los mismos se generan en orden secuencial, por lo que los reordenamos aleatoriamente ("shuffle") con el utilitario de Linux *shuf*.

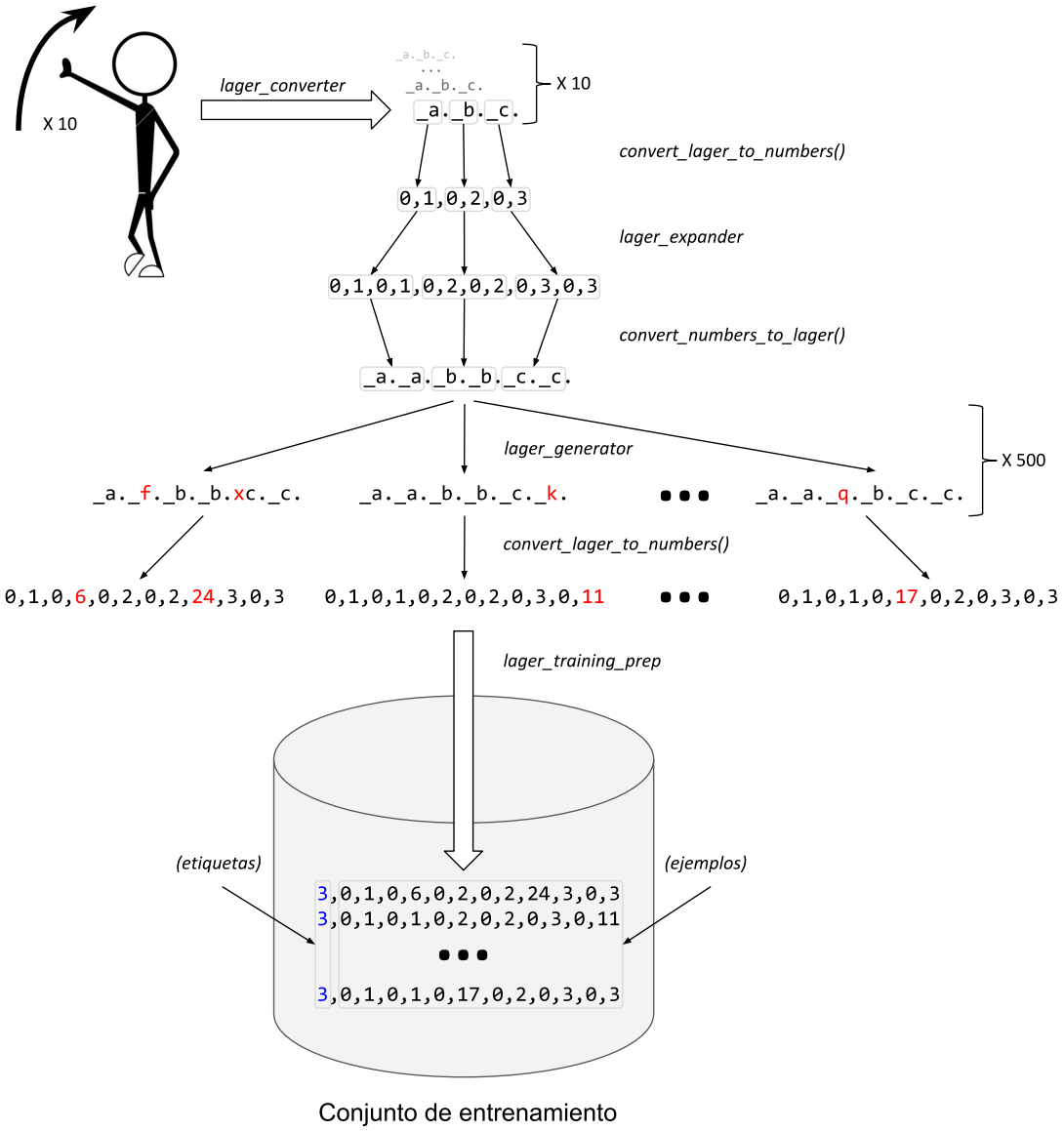


FIGURA 5.2: Creación de conjunto de entrenamiento. (Muñeco de palitos obtenido de Storyblocks, 2018)

Según vemos en los Cuadros 5.1 y 5.2, el conjunto de datos para el clasificador de 8 gestos contiene 40 mil entradas, mientras que el de 16 gestos contiene 80 mil entradas. Estos datos se le alimentan al programa *ml_trainer.py*, el cual usa el 75 % como conjunto de entrenamiento y el 25 % restante como conjunto de validación.

Gestos por clasificar	8
Ejemplos manuales por gesto	x 10
Variantes autogeneradas por ejemplo	x 500
Total de datos de entrenamiento	**40000**

CUADRO 5.1: Conjunto de datos para clasificador de 8 gestos

Gestos por clasificar	16
Ejemplos manuales por gesto	x 10
Variantes autogeneradas por ejemplo	x 500
Total de datos de entrenamiento	**80000**

CUADRO 5.2: Conjunto de datos para clasificador de 16 gestos

5.5.3.2. Desempeño

Una vez creado el conjunto de entrenamiento, entrenamos cada modelo y evaluamos su desempeño.

Como vemos en el Cuadro 5.3, ambas redes se entrenaron rápidamente y produjeron buenos resultados.

	Red para 8 gestos	**Red para 16 gestos**
Tiempo de entrenamiento (s)	39	77
Pérdida logarítmica promedio	0.0391	0.0942
Exactitud promedio	99.83 %	99.03 %

CUADRO 5.3: Valores promedio para tres corridas de entrenamiento de las redes neuronales

La red de 8 gestos fue casi dos veces más rápida entrenando, ya que su topología es más sencilla en la última capa y tiene un conjunto de entrenamiento más pequeño (40 mil ejemplos vs. 80 mil ejemplos).

La red de 8 gestos también tuvo una perdida menor y una exactitud mayor, lo cual se explica por el hecho de que tiene que discriminar entre menos clases, y por lo tanto las probabilidades de clasificación se distribuyen entre menos objetivos.

En las Figuras 5.3 y 5.4 se muestra el historial de la pérdida logarítmica durante el entrenamiento de ambos modelos. Vemos que desciende consistentemente tanto para el conjunto de entrenamiento ("train") como para el de validación ("test"). Esto significa que no llegó a haber overfitting, lo cual se habría visto como en la Figura 5.5. En tal circunstancia, el desempeño con el conjunto de validación empezaría a empeorar a medida que el modelo se adapta excesivamente al conjunto de entrenamiento y pierde la capacidad de generalizar.

Por último, en las Figuras 5.6 y 5.7 se muestra el historial de la exactitud durante el entrenamiento de ambos modelos. La evolución es similar a la que vimos para la pérdida logarítmica, con mejoras constantes tanto para el conjunto de entrenamiento como para el de prueba.

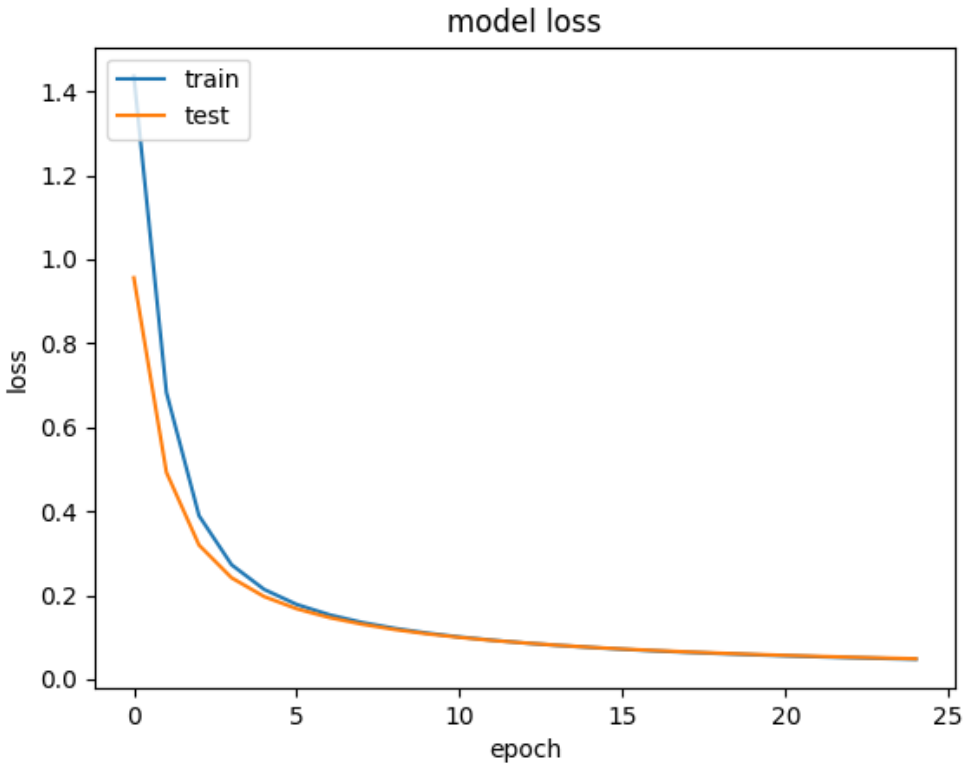


FIGURA 5.3: Historial de pérdida logarítmica para entrenamiento del modelo de 8 gestos

5.6. Proceso para Reconocimiento con Deep Learning

El proceso para reconocer gestos con el las redes neuronales es similar al del entrenamiento (ver Figuras 5.8 y 5.9.

1. Se convierte la hilera de entrada a números por medio de convert_lager_to_numbers()
2. Se redimensiona al tamaño estándar por medio de la biblioteca de procesamiento de imágenes scikit-image
3. Se redimensiona a un arreglo bidimensional y se le alimenta a la red neuronal
4. La red procesa los datos y su última capa produce un arreglo con probabilidades $[P1, ..., Pn]$, donde n es el número de gestos a clasificar y Pn es la probabilidad de que la hilera de entrada corresponda a ese gesto.
5. Por último, lager_recognizer decide si la probabilidad más alta está por arriba del umbral de detección ML_RECOGNITION_THRESHOLD_PCT, el cual

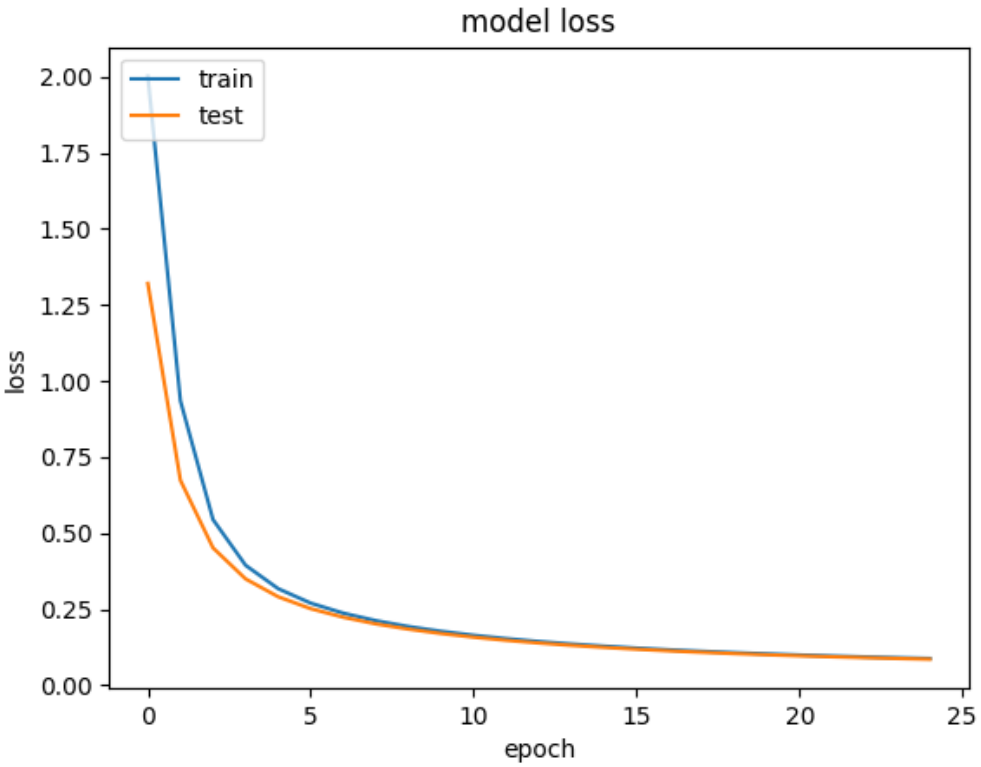


FIGURA 5.4: Historial de pérdida logarítmica para entrenamiento del modelo de 16 gestos

establecimos en 55 % según nuestro juicio a través de un proceso de prueba y error.

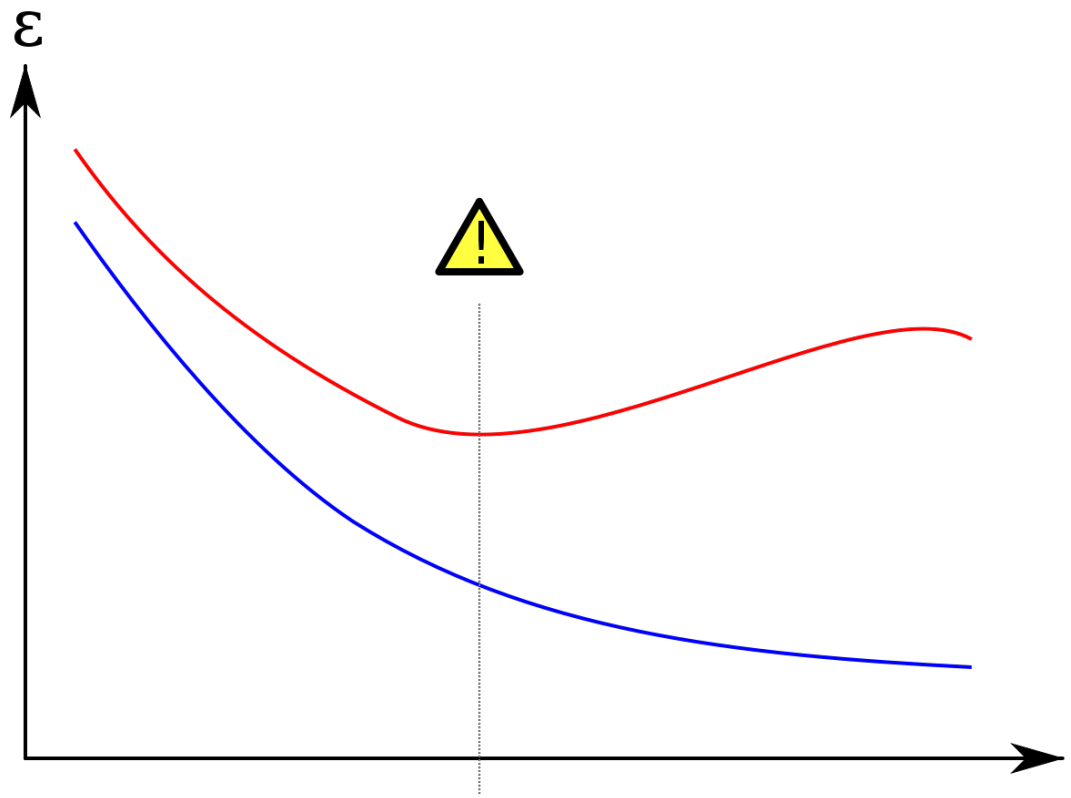


FIGURA 5.5: Ejemplo de overfitting

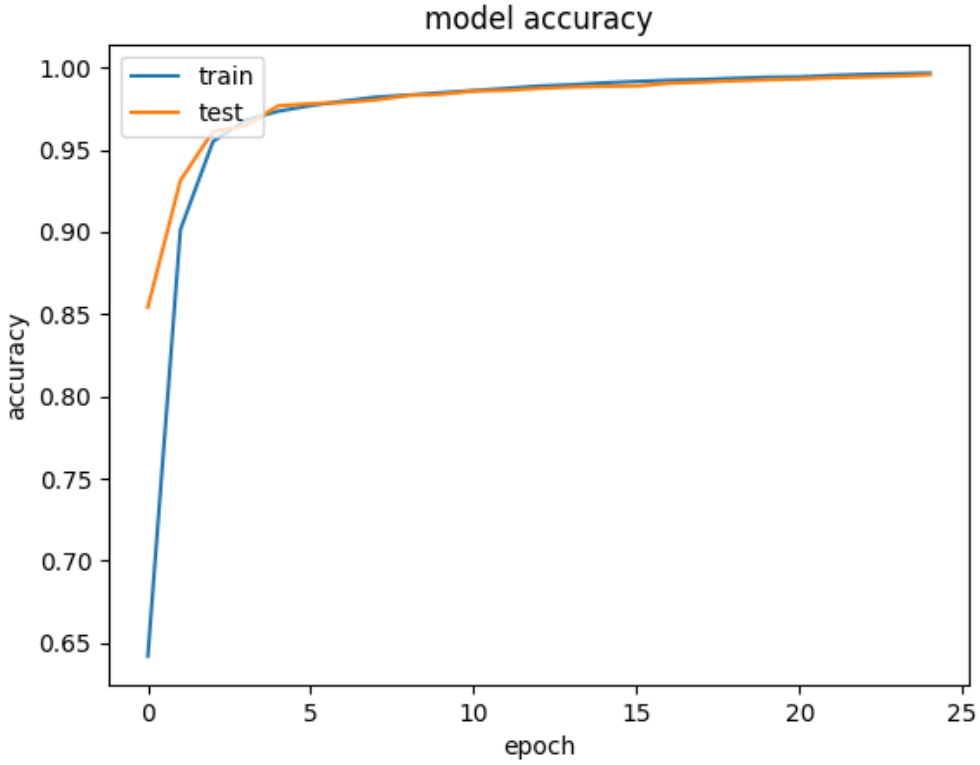


FIGURA 5.6: Historial de exactitud para entrenamiento del modelo de 8 gestos

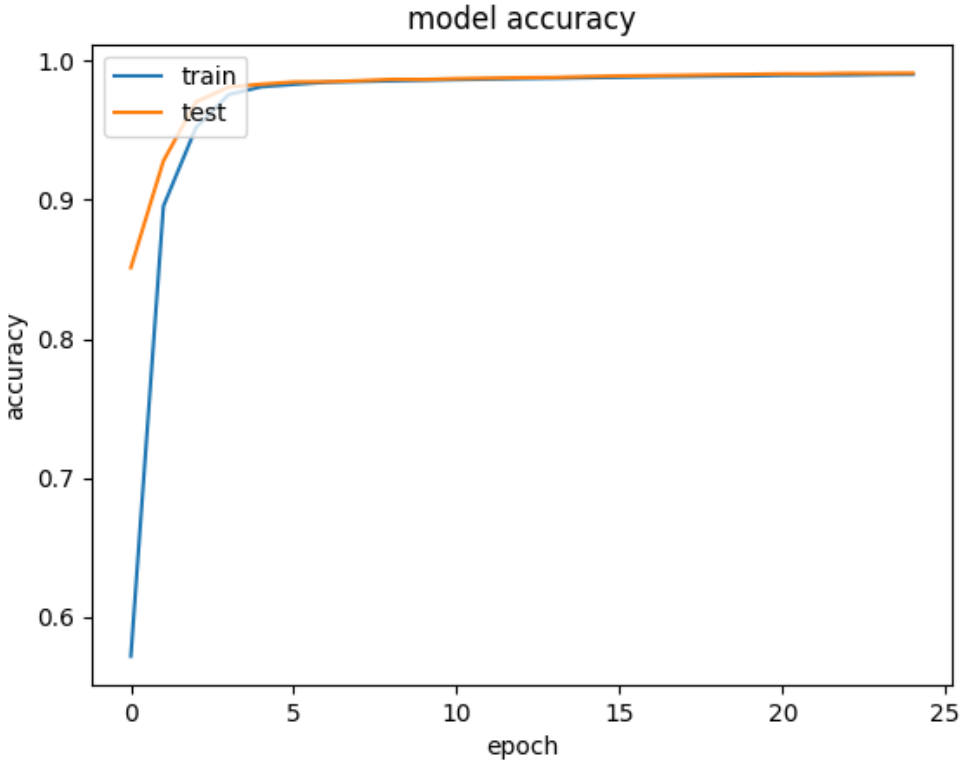


FIGURA 5.7: Historial de exactitud para entrenamiento del modelo de 16 gestos

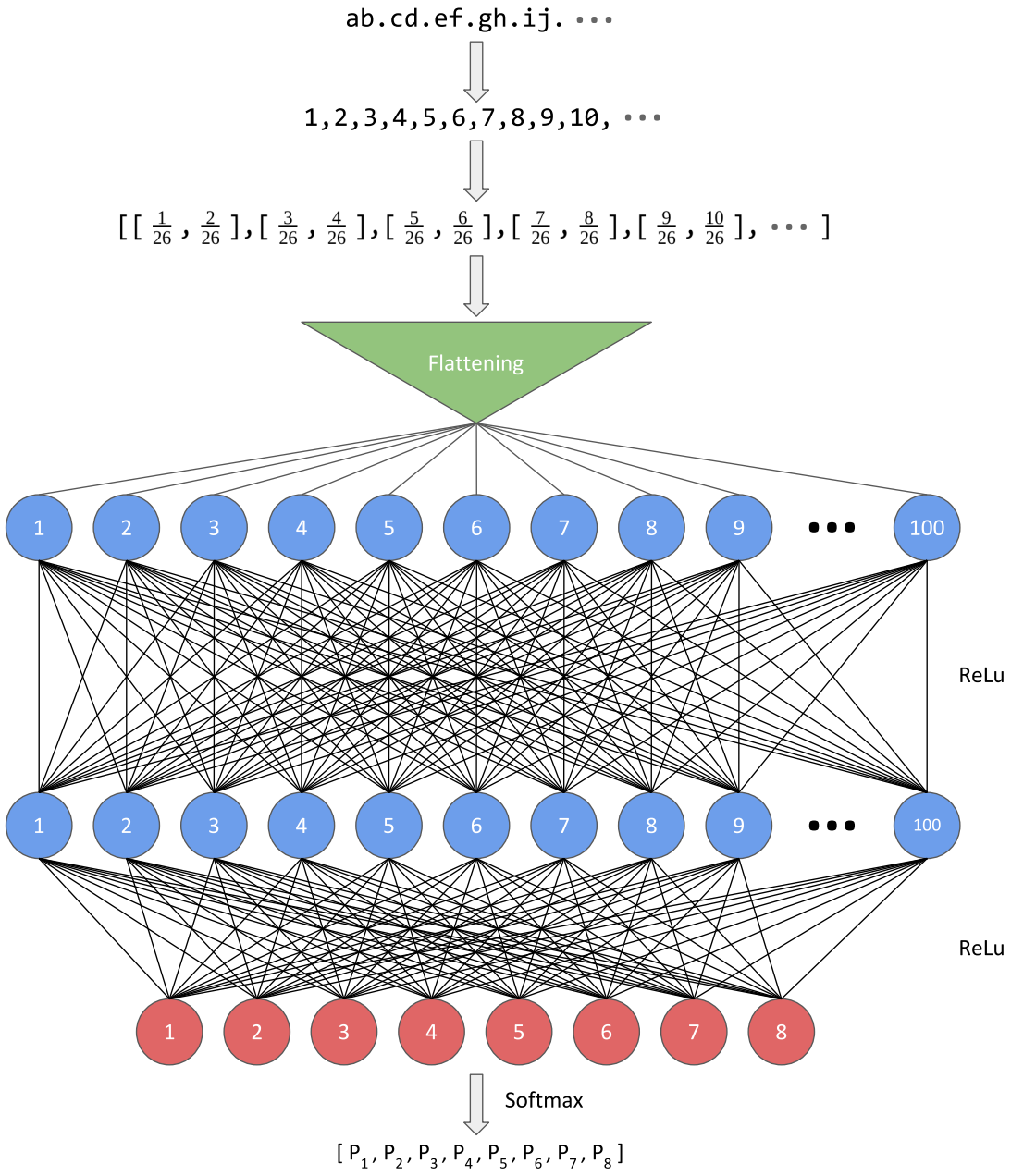


FIGURA 5.8: Red neuronal para 8 gestos

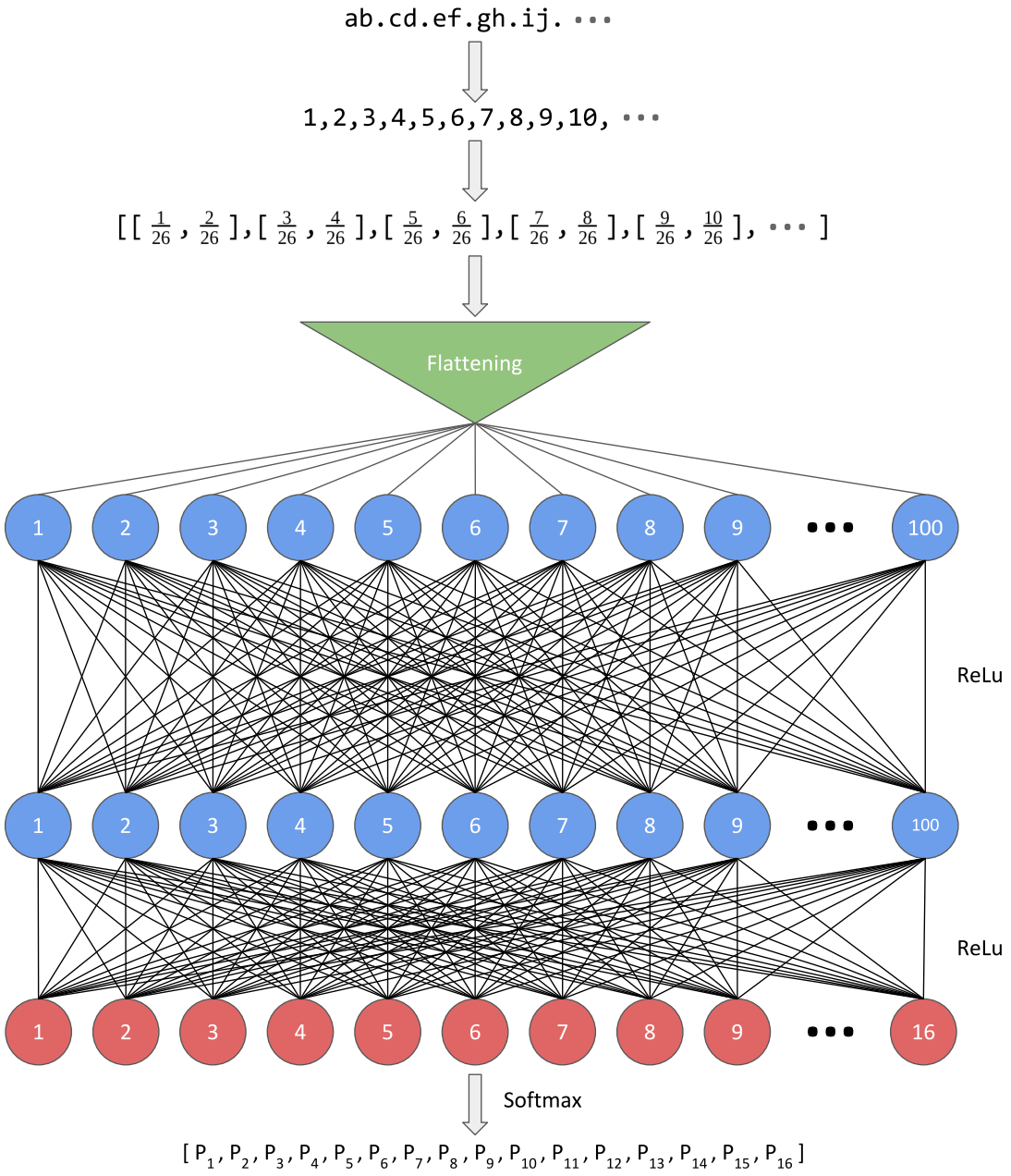


FIGURA 5.9: Red neuronal para 16 gestos

Capítulo 6

Análisis de Resultados

6.1. Experimento 1

El experimento 1 fue exactamente la misma que se realizó en la tesis LaGeR (Odio-Vivi, 2015), tal y como se describió en el Capítulo 4. Los resultados de esta prueba se muestran en el Cuadro 6.1, donde se puede observar un resultado según lo esperado.

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
88.89	610.58	93.06	6.13

Cuadro 6.1: Resultados de experimento 1 con gestos grandes

Mientras tanto, el mismo experimento con gestos pequeños mostró un resultado inferior (pero en todo caso satisfactorio) de 90.28 % de exactitud para el algoritmo de deep learning vs 84.72 % para el algoritmo Damerau-Levenshtein. Los resultados se pueden observar en el Cuadro 6.2.

Esta diferencia se debe a que al tratar de hacer gestos pequeños, las hileras son más cortas y por lo tanto hay menos literales y menos "resolución" para describir los trazos. Este pese a que el sistema diseñado trata de normalizar el tamaño de los datos obtenidos, ya que al expandir hileras pequeñas se repiten los literales originales (no se crea "resolución"), y aunado a esto para los usuarios es más difícil ser precisos al dibujar gestos pequeños con los diferentes dispositivos de entrada.

En todo caso, se puede observar que persiste la tendencia de mejor exactitud por parte del algoritmo deep learning respecto al Damerau-Levenshtein.

Al mismo tiempo, se evidencia una mejora en los tiempos de reconocimiento para ambos algoritmos al hacer los gestos pequeños. Esto también es esperado tomando en cuenta que las hileras más pequeñas son más fáciles de procesar. La diferencia es mucho mayor para el algoritmo Damerau-Levenshtein, ya que este es el más afectado por la longitud de las hileras debido al método de expansión (ver Subsección 3.3.3) y la subsiguiente comparación caracter por caracter. En cambio, el algoritmo de deep learning redimensiona las hileras con una operación eficiente de la biblioteca de imágenes scikit-image, y luego ejecuta un modelo pre-entrenado cuya desempeño es prácticamente de tiempo constante.

Respecto a la invariabilidad de rotación los resultados se pueden ver en el Cuadro 6.3. El algoritmo de deep learning fue capaz de tener una exactitud del 100 % con gestos "RefreshTab" ejecutados en dirección anti-horaria, en tanto el algoritmo Damerau-Levenshtein no fue capaz de reconocer ni uno solo.

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
84.72	204.18	90.28	5.29

CUADRO 6.2: Resultados de experimento 1 con gestos pequeños

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
0	506.08	100	4

CUADRO 6.3: Resultados de experimento 1 con variabilidad de rotación

6.2. Experimento 2

El experimento 2 tal y como se describió en el Capítulo 4 consistió en realizar los mismos gestos descritos en el experimento 1 pero utilizando 2 dispositivos diferentes al Razer Hydra: el Wacom y el Leap Motion.

En general, los resultados obtenidos continuaron la tendencia del experimento 1. Sin embargo, en este caso no solo queremos observar el comportamiento para gestos grandes y pequeños, sino también observar la adaptabilidad del algoritmo Damerau-Levenshtein vs. el algoritmo de deep learning.

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
84.72	479.53	95.83	5.49

CUADRO 6.4: Resultados de experimento 2 con el Leap Motion para gestos grandes

Como se observa en el Cuadro 6.4 los resultados obtenidos utilizando el dispositivo Leap Motion y gestos grandes tuvieron un 95.83 % de exactitud con el algoritmo de deep learning vs. 84.72 % con el de Damerau-Levenshtein. En tanto a los tiempos promedio de ejecución, tenemos un total de 479.53 ms para el Damerau-Levenshtein vs. 5.49 ms para el algoritmo de deep learning. En estos resultados se refleja la misma tendencia observada en los resultados del experimento 1.

Los resultados observados en el Cuadro 6.5 muestran cómo para gestos pequeños la exactitud y el tiempo de ejecución promedio bajan con ambos algoritmos. Cabe mencionar que persiste el fenómeno de que el tiempo de ejecución del algoritmo de deep learning es casi el mismo para todos los experimentos, debido a

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
83.33	237.5	88.89	4.44

CUADRO 6.5: Resultados de experimento 2 con el Leap Motion para gestos pequeños

que una red neuronal ya entrenada siempre tarda casi lo mismo en ejecutarse; la única variante es el tiempo que dure el proceso en espera del hilo ejecutor.

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
88.33	1278.47	93.33	4.35

CUADRO 6.6: Resultados de experimento 2 con el Wacom para gestos pequeños

Como se observa en el Cuadro 6.6, el Wacom mostró mejores resultados para gestos pequeños de los que mostró el Leap Motion. Esto era esperado ya que es más sencillo para el usuario hacer uso de un *stylus* para dibujar los gestos vs. dibujarlos en el aire a pequeña escala. Además hay que tomar en cuenta que con el Wacom se efectuaron menos gestos pues los gestos que requerían dos sensores (e.g., *ZoomIn* y *ZoomOut*) no se pudieron hacer ya que solo es capaz de rastrear el movimiento de un stylus.

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
90	1736.13	88.33	4.62

CUADRO 6.7: Resultados de experimento 2 con el Wacom para gestos grandes

Los resultados del experimento 2 con gestos grandes utilizando el Wacom se muestran en el Cuadro 6.7, donde se puede observar que son los mejores resultados de los tres dispositivos utilizados con el algoritmo Damerau-Levenshtein. Como se mencionó anteriormente, esto se debe a que fue un modo mucho más natural para los usuarios el utilizar un stylus para dibujar los gestos. Además, al dibujarlos con un tamaño mayor se les facilitó ser precisos con los detalles.

Finalmente, los Cuadros 6.8 y 6.9 muestran los resultados del experimento 2 a la invariabilidad de rotación. Nuevamente el algoritmo de Damerau-Levenshtein no fue capaz de reconocer los gestos, en tanto el de deep learning sí lo fue.

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
0	2242.67	91.67	4

CUADRO 6.8: Resultados de experimento 2 con el Wacom con invariabilidad de rotación

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
0	469.5	100	4.08

CUADRO 6.9: Resultados de experimento 2 con el Leap Motion con invariabilidad de rotación

6.3. Experimento 3

El experimento 3 consistió en probar la capacidad de ambos algoritmos para reconocer gestos nuevos a los descritos en la tesis LaGeR (Odio-Vivi, 2015). La lista completa de gestos hecha en este experimento se detalla en el Capítulo 4, donde además de dos de los gestos antiguos (*OpenChrome* y *CloseTab*), se incluyen 8 nuevos gestos.

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
80	5997.95	100	7.75

CUADRO 6.10: Resultados de experimento 3 para gestos grandes

Como se observa en el Cuadro 6.10, la diferencia en el porcentaje de exactitud de ambos algoritmos aumenta al introducir más gestos al sistema. Esto se debe a que el algoritmo Damerau-Levenshtein se basa en una simple comparación, por lo que es más susceptible a que la probabilidad de fallo aumente al tener más gestos con los cuáles comparar. En tanto, con el algoritmo de deep learning los buenos resultados para gestos grandes prevalecen. Sí hay que tomar en cuenta al hacer un análisis de estos resultados que los sujetos de prueba fueron únicamente 2 en lugar de 6, lo cuál disminuye el poder de los datos a analizar.

Los resultados de gestos pequeños que se muestran en el Cuadro 6.11 fueron menos satisfactorios que para gestos grandes. Esto fue especialmente evidente con el algoritmo Damerau-Levenshtein, tal y como fue esperado. Como se mencionó

anteriormente, los gestos pequeños son más difíciles de hacer para el usuario, contienen menos información y aunado a esto el LaGeR Workbench original normaliza las hileras de forma ineficiente.

6.4. Análisis de matrices de confusión

6.4.1. Matriz de confusión experimentos 1 y 2

Como se observa en la Figura 6.1 la mayoría de los errores cometidos por el algoritmo Damerau-Levenshtein se dieron con el gesto *RefreshTab*, que fue confundido con el *OpenGoogle*. Esto se debe a que cuando se hicieron los experimentos de invariabilidad de rotación, el circulo anti-horario de *RefreshTab* era confundido con una "G" de Google. Recordemos que el algoritmo Damerau-Levenshtein funciona por la distancia lexicográfica y al hacer el circulo anti-horario y no poseer invariabilidad de rotación, lo más cercano a ese gesto es la "G".

Para los otros gestos que no se pudieron reconocer, el algoritmo no fue capaz de clasificarlos como ninguno de los gestos almacenados.

	N/A	CloseTab	NewTab	OpenChrome	OpenCNN	OpenGoogle	RefreshTab	ZoomIn	ZoomOut
N/A	–	–	–	–	–	–	–	–	–
CloseTab	1	95	0	0	0	0	0	0	0
NewTab	0	0	64	0	0	0	0	0	0
OpenChrome	2	0	0	27	0	2	0	0	0
OpenCNN	0	0	0	0	32	0	0	0	0
OpenGoogle	0	0	0	0	0	32	0	0	0
RefreshTab	7	0	0	0	3	23	31	0	0
ZoomIn	5	0	0	0	0	0	0	15	0
ZoomOut	5	0	0	0	0	0	0	0	15

FIGURA 6.1: Matriz de confusión del experimento 1 y 2 para el algoritmo Damerau-Levenshtein

Con el algoritmo de deep learning, los resultados obtenidos fueron más satisfactorios. Sin embargo como se observa en la figura 6.2 en algunos casos los gestos *RefreshTab* y *OpenGoogle* se confundieron entre si. Esto se debe a que dos de los sujetos dibujaron la "G" de *OpenGoogle* muy cerrada y fue difícil para el algoritmo distinguir si los gestos correspondían a una "G" o a un círculo un poco abierto. Esto se podría mejorar usando más sujetos para crear el conjunto de entrenamiento del sistema.

	N/A	CloseTab	NewTab	OpenChrome	OpenCNN	OpenGoogle	RefreshTab	ZoomIn	ZoomOut
N/A	–	–	–	–	–	–	–	–	–
CloseTab	0	96	0	0	0	0	0	0	0
NewTab	0	0	63	1	0	0	0	0	0
OpenChrome	1	0	1	29	0	1	0	0	0
OpenCNN	2	0	0	0	30	0	0	0	0
OpenGoogle	2	0	0	0	0	22	8	0	0
RefreshTab	3	0	0	1	0	8	52	0	0
ZoomIn	0	0	0	0	0	0	0	19	1
ZoomOut	0	0	0	0	0	0	0	1	19

FIGURA 6.2: Matriz de confusión del experimento 1 y 2 para el algoritmo de deep learning

6.4.2. Matriz de confusión experimento 3

Como se observa en la Figura 6.3, el comportamiento del algoritmo Damerau-Levenshtein con nuevos gestos es algo caótico. Como se comentó en la Sección 6.3 para gestos grandes el algoritmo presentó una exactitud de 80 %. Es gracias a ese resultado que la matriz presenta una tendencia a la diagonal. Sin embargo, existen gestos como por ejemplo el de *MaximizeWindow* que el algoritmo no fue capaz de reconocer.

No existe una tendencia clara, como se vio en la Subsección 6.4.1, donde la confusión del algoritmo se daba con los gestos *RefreshTab* y *OpenGoogle*. En este caso no esta tan explicito, no obstante se podría concluir que el algoritmo confunde la "W" de *OpenWeather* con el *ScrollDown*, que era simplemente una línea vertical.

	N/A	CloseTab	MaximizeWindow	OpenChrome	OpenMusic	OpenNetflix	OpenWeather	OpenYouTube	RestoreWindow	ScrollDown	ScrollUp
N/A	–	–	–	–	–	–	–	–	–	–	–
CloseTab	0	4	0	0	0	0	0	0	0	0	0
MaximizeWindow	4	0	0	0	0	0	0	0	0	0	0
OpenChrome	0	0	0	3	0	1	0	0	0	0	0
OpenMusic	0	0	0	0	4	0	0	0	0	0	0
OpenNetflix	0	0	0	0	0	3	0	0	0	1	0
OpenWeather	0	0	0	0	0	0	1	0	0	3	0
OpenYouTube	0	0	0	0	0	0	0	3	0	0	1
RestoreWindow	3	0	0	0	0	0	0	0	1	0	0
ScrollDown	0	0	0	0	0	0	0	0	0	4	0
ScrollUp	0	0	0	0	0	0	0	0	0	0	3

FIGURA 6.3: Matriz de confusión del experimento 3 algoritmo Damerau-Levenshtein

En la Figura 6.4 se puede observar la matriz de confusión del experimento 3 para el algoritmo de deep learning. El comportamiento de este algoritmo para reconocer los gestos nuevos fue muy bueno y una de las dos veces que falló se debe a que a pesar de que el gesto que tenía la mayor probabilidad fue *OpenChrome*, el resultado fue inferior al umbral, entonces no lo consideró correcto. El segundo sí fue un error en el reconocimiento.

	N/A	CloseTab	laximizeWindo	OpenChrome	OpenMusic	OpenNetflix	OpenWeather	OpenYouTube	RestoreWindow	ScrollDown	ScrollUp
N/A	–	–	–	–	–	–	–	–	–	–	–
CloseTab	0	3	0	0	0	0	0	1	0	0	0
MaximizeWindo	0	0	4	0	0	0	0	0	0	0	0
OpenChrome	1	0	0	3	0	0	0	0	0	0	0
OpenMusic	0	0	0	0	4	0	0	0	0	0	0
OpenNetflix	0	0	0	0	0	4	0	0	0	0	0
OpenWeather	0	0	0	0	0	0	4	0	0	0	0
OpenYouTube	0	0	0	0	0	0	0	4	0	0	0
RestoreWindow	0	0	0	0	0	0	0	0	3	0	0
ScrollDown	0	0	0	0	0	0	0	0	0	4	0
ScrollUp	0	0	0	0	0	0	0	0	0	0	4

FIGURA 6.4: Matriz de confusión del experimento 3 algoritmo deep learning

Algoritmo Damerau-Levenshtein		**Algoritmo de deep learning**	
Exactitud (%)	Tiempo promedio (ms)	Exactitud (%)	Tiempo promedio (ms)
50	926.75	85	7.15

CUADRO 6.11: Resultados de experimento 3 para gestos pequeños

6.5. Análisis de Gestos Específicos

En esta sección veremos algunos gestos que fueron excluidos por distintos motivos del análisis de las secciones anteriores. Además veremos algunos gestos que fueron identificados como erróneos por el sistema para entender cómo la diversidad de sujetos y de diferentes caligrafías afectaron los resultados obtenidos. Muchas de estas situaciones ya fueron descritas en las secciones anteriores, sin embargo tener los gestos ayuda a entender las tendencias explicadas con antelación.

La curva de aprendizaje para el uso de los dispositivos de entrada ocasionó que las primeras veces que los sujetos hicieron ciertos gestos, estos fueron irreconocibles para ambos algoritmos. En la Figura 6.5 se muestran dos ejemplos de estos casos. El gesto de la izquierda es un intento de *OpenCNN* y el de la derecha es un intento de *RefreshTab* (círculo en dirección horaria). Debido a que inclusive para el ojo humano estos gestos no son fáciles de identificar, se tomó la decisión de excluirlos.

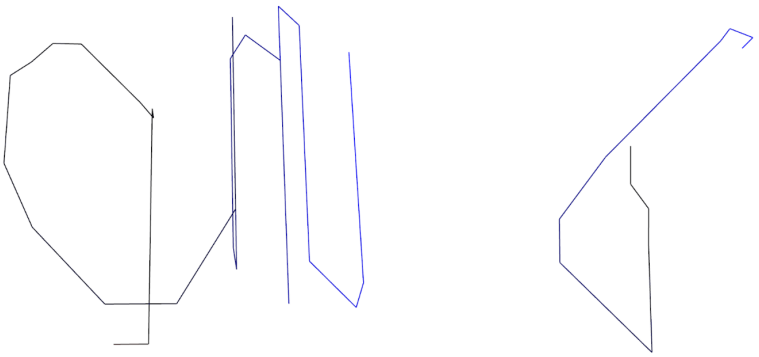


FIGURA 6.5: Gestos excluidos

También se observó que ninguno de los algoritmos fue capaz de reconocer gestos que variaban el punto inicial de donde se dibujaban. Específicamente estamos refiriéndonos al círculo, ya que este es el único gesto que puede variar su punto de inicio. Dos ejemplos de este fenómeno se muestran en la Figura 6.6 donde el punto inicial del circulo fue a la izquierda en lugar de arriba, que es como se definió el gesto. En el caso del circulo de la izquierda ninguno de los algoritmos fue capaz de reconocer este gesto como un *RefreshTab*. El de la derecha fue reconocido de forma

correcta por el algoritmo Damerau-Levenshtein. El algoritmo de deep learning, al poseer invariabilidad de rotación, confundió ese circulo con una "G" espejo abierta.

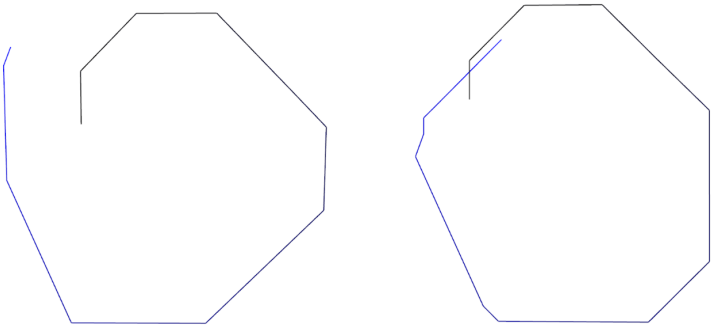


FIGURA 6.6: Gestos con punto inicial variable

En otros casos, hubo gestos cuya forma general fue correcta pero que tenía ruido al inicio y/o al final de la misma. Estos tendieron a ser identificados de forma correcta por el algoritmo Damerau-Levenshtein, pero no así por el algoritmo de deep learning. La Figura 6.7 muestra a la izquierda un gesto *RefreshTab* donde eso sucedió. El gesto de la derecha muestra un gesto *NewTab* con ruido al final que también fue reconocido únicamente por el algoritmo Damerau-Levenshtein.

De lo anterior concluimos que el algoritmo de deep learning presenta dificultad para reconocer gestos que presenten ruido al inicio o al final del mismo, en tanto el algoritmo de Damerau-Levenshtein es capaz de hacerlo sin ningún problema. Esto puede ser debido a que esas pequeñas desviaciones resultan en pocas letras de diferencia al calcular la distancia Damerau-Levenshtein, pero que en una red neuronal de caja negra pueden tener resultados inesperados.

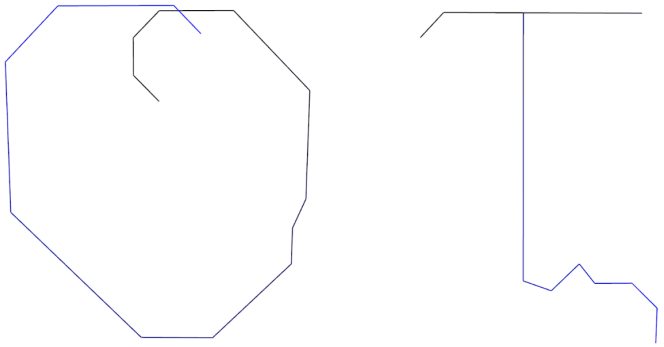


FIGURA 6.7: Gestos con ruido al inicio y al final

Por otra parte, sí hubo casos donde el algoritmo Damerau-Levenshtein no fue capaz de reconocer gestos ruidosos. La figura 6.8 muestra dos ejemplos del gesto

OpenChrome que no fueron reconocidos por ninguno de los algoritmos debido a su forma ruidosa. Pese a esto, en general el algoritmo Damerau-Levenshtein tuvo un mejor comportamiento con gestos ruidosos.

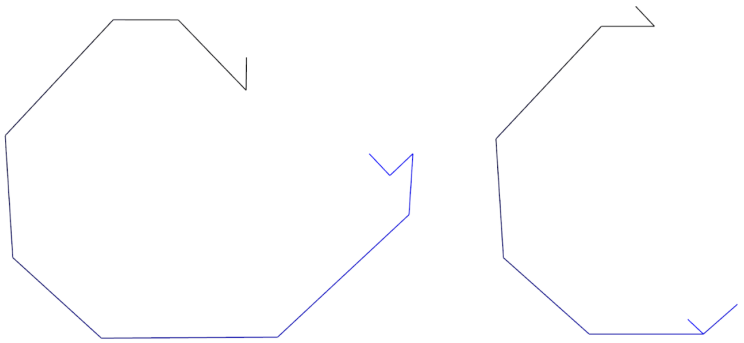


FIGURA 6.8: Gestos con ruido no reconocidos

La Figura 6.9 muestra un ejemplo de un gesto ruidoso que sí fue capaz de reconocer el algoritmo de deep learning pero no el de Damerau-Levenshtein. Este ejemplo sin embargo fue una excepción ya que al ser un gesto anti-horario el algoritmo Damerau-Levenshtein no es capaz de reconocer este tipo de gestos al no tener invariabilidad de rotación. No obstante y como se mencionó anteriormente, el algoritmo Damerau-Levenshtein suele ser mejor para reconocer gestos ruidosos.

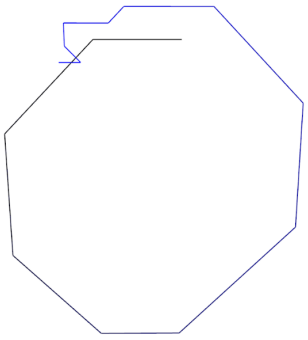


FIGURA 6.9: Gesto con ruido reconocido por el algoritmo de deep learning

Algunos de los errores cometidos por el algoritmo de deep learning que mencionamos en la Subsección 6.4.1 están relacionados con la invariabilidad de rotación y cómo en ocasiones, dependiendo de la "G" que se dibuje para el gesto *OpenGoogle*, el algoritmo de deep learning confunde esos gestos con un circulo anti-horario. En la Figura 6.10 se muestran 3 gestos de OpenGoogle ("G") que el algoritmo de deep learning confundió con gestos de *RefreshTab* (circulo anti-horario).

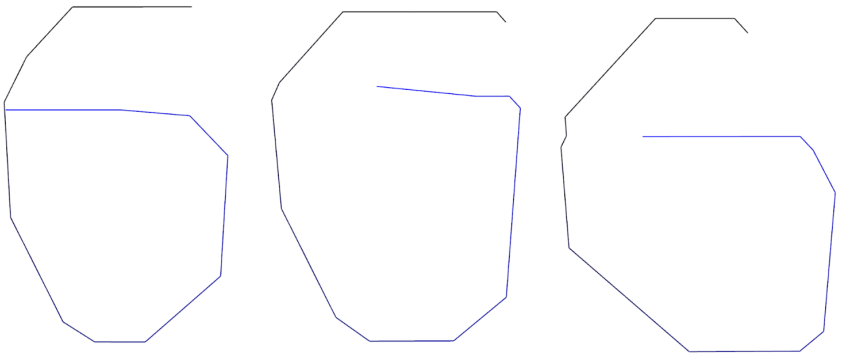


FIGURA 6.10: Gestos confundidos por el algoritmo de deep learning

Otro caso curioso que sucedió con el algoritmo de deep learning y que también mencionamos en la Subsección 6.4.1 es el de las "G" espejo. Al poder identificar gestos en sentido anti-horario, el algoritmo de deep learning identifico gestos de *RefreshTab* realizados por los sujetos como si fueran *OpenGoogle* ("G" espejo). En la Figura 6.11 se pueden observar tres gestos de *RefreshTab* que los sujetos de estudio realizaron como círculos y que sí fueron reconocidos como tales por el algoritmo de Damerau-Levenshtein pero no por el de deep learning, el cuál los identificó como *OpenGoogle* ("G"). Tal y como se observa en dicha figura, efectivamente estos gestos están más cercanos de ser una "G" espejo que a un circulo.

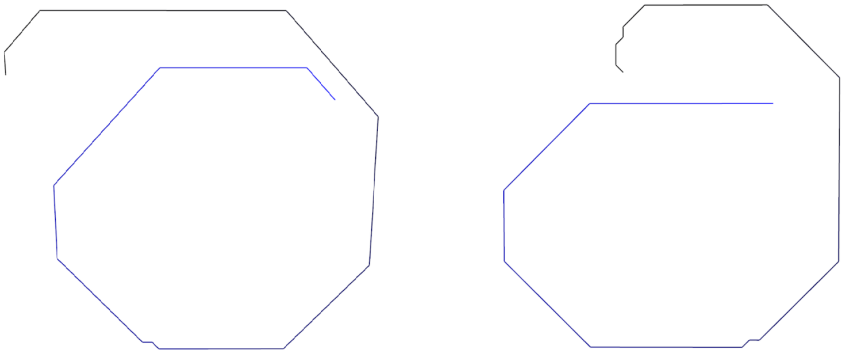


FIGURA 6.11: Gestos que el algoritmo deep learning confundió con "G" espejo

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

La presente investigación probó la hipótesis de que es posible encontrar un algoritmo de deep learning para mejorar el enfoque de reconocimiento de gestos con LaGeR logrado con el algoritmo Damerau-Levenshtein, mejorando el tiempo de ejecución y la exactitud. Dichos resultados se ven soportados por los datos mostrados en el Capitulo 6.

Los objetivos planteados se alcanzaron de la siguiente manera:

1. **Objetivos 1**:Desarrollamos e implementamos un algoritmo de deep learning que mejoró los resultados obtenidos por el LaGeR Workbench original.

2. **Objetivo 2**: Logramos comparar el algoritmo implementado con el original de LaGeR Workbench (Damerau-Levenshtein), enfocándonos en la exactitud de la detección de gestos, la invariabilidad de escala, la invariabilidad de rotación y el tiempo de ejecución (el cual disminuyó en al menos dos órdenes de magnitud). Este objetivo quedó comprobado en el resultado del experimento 1 (Sección 6.1).

3. **Objetivo 3**: Se logró utilizar más dispositivos de entrada para la prueba del funcionamiento del LaGeR Workbench. Esto quedó comprobado en el experimento 2 (Sección 6.2.

4. Adicionalmente, se desarrolló un tercer experimento que de manera preliminar nos sugiere que al incrementar el número de gestos a identificar, el algoritmo de deep learning logró desempeñarse con mayor velocidad y exactitud que el algoritmo Damerau-Levenshtein.

7.1.1. Trabajo Futuro

La investigación desarrollada logró cumplir los objetivos planteados en esta tesis, no obstante siempre existen áreas que se pueden desarrollar para futuros trabajos. A continuación una lista de posibles mejoras que se podrían trabajar en el futuro:

1. **Algoritmo Basado en Modelos Ocultos de Márkov**: Explorar utilizar un algoritmo basado en Modelos Ocultos de Márkov en lugar de los algoritmos de deep learning y de Damerau-Levenshtein. Tal y como se pudo leer en el Capítulo 3.2, los Modelos Ocultos de Márkov han sido ampliamente

utilizados en el reconocimiento de gestos. Es por este motivo que consideramos importante explorar utilizar un algoritmo basado en dicho enfoque en LaGeR Workbench.

2. **Automatización del afinamiento de la red neuronal**: Una de las recomendaciones de los expertos respecto a la creación de algoritmos de deep learning es automatizar el proceso de elección de los hiper-parámetros de la red neuronal.

3. **Optimización del algoritmo de Monte Carlo**: Mejorar la generación de variantes para el entrenamiento de la red neuronal. Podría ser mediante el ajuste de los parámetros del generador basado en Monte Carlo, o por medio de otros métodos. Esta posible mejora la concluimos de lo observado en la Sección 6.5, donde se pudo observar que algunos gestos con diferentes caligrafías no fueron reconocidos por el algoritmo de deep learning.

4. **Automatizar la creación de nuevos gestos**: Automatizar el proceso mediante el cual programas escritos por terceros puedan definir nuevos gestos a ser identificados por el nuevo reconocedor. (e.g., integrar proceso descrito en la Subsección 5.5.3.1 con el lager_gesture_manager). Esto se debe a que actualmente si utilizamos el algoritmo de Damerau-Levenshtein es sencillo para cualquier usuario añadir nuevos gestos a la lista de candidatos, sin embargo no es igual de sencillo si lo que deseamos es utilizar el algoritmo de deep learning. El algoritmo de deep learning requiere una gran cantidad de datos generados mediante Monte Carlo, un cambio a su topología y un re-entrenamiento, por lo que la creación de nuevos "gestos candidatos" es más compleja.

5. **Sistema de suscripciones**: Actualmente, el LaGeR Workbench provee un mecanismo de "suscripciones"para que los programas escritos por terceros soliciten ser notificados cuando se detecten gestos específicos. Sin embargo, el mecanismo es relativamente limitado ya que solo permite crear nuevas suscripciones. Tal y como se sugirió en la tesis original de LaGeR, "Se podría crear un mecanismo para cancelar las suscripciones [manualmente], así como [...] para cancelarlas automáticamente en caso de que el programa cliente muera inesperadamente." (Odio-Vivi, 2015).

Apéndice A

Apéndice A: Configuración de Dependencias y Dispositivos de Entrada

```
/**
 * Dependencies that need to be installed for lager to compile and run
    on
 * an Ubuntu 18.04 machine.
 */

-----------------
 Ubuntu Packages
-----------------

Install the missing packages:
~$ sudo apt-get install cmake doxygen git graphviz libboost-
    serialization-dev libboost-system-dev libboost-thread-dev libglew-dev
     libusb-1.0-0-dev libxrandr-dev libxt-dev libxtst-dev

Make sure to accept all extra packages suggested by apt-get.

-----------------
      OSVR
-----------------

Clone the OSVR-Core repository:
~$ git clone https://github.com/OSVR/OSVR-Core.git

Install OSVR-Core by following the instructions at:
https://github.com/OSVR/OSVR-Docs/blob/master/Getting-Started/Installing
    /Linux-Build-Instructions.md

If the compilation segfaults with GCC, try it with clang:
~/OSVR-Core/build$ CXX=/usr/bin/clang++ CC=/usr/bin/clang cmake ..
~/OSVR-Core/build$ CXX=/usr/bin/clang++ CC=/usr/bin/clang make

Finally, install OSVR:
~/OSVR-Core/build$ sudo make install

-----------------
   Razer Hydra
-----------------

The Razer Hydra is supported out of the box by OSVR.

Simply start the OSVR server:
~$ sudo osvr_server

-----------------
   Leap Motion
-----------------

Download the Leap Motion v2 SDK:
https://leapmotion.app.box.com/s/rlwzvt680i5r6zes75ge36ub2yt7ksx3

Extract it:
~$ tar xvzf LeapDeveloperKit_linux-hotfix_2.3.1+33747_linux.tgz
```

```
Move the LeapSDK to your home directory:
~$ mv LeapDeveloperKit_2.3.1+33747_linux/LeapSDK/ .

Copy the Leap library to your system:
~$ sudo cp LeapSDK/lib/x64/libLeap.so /usr/local/lib/

Clone the OSVR-Leap-Motion repository:
~$ git clone https://github.com/OSVR/OSVR-Leap-Motion.git

Checkout the following commit, which is the latest one that supports the
     Leap
Motion v2 SDK:
~$ cd OSVR-Leap-Motion
~/OSVR-Leap-Motion$ git checkout
    aa89b29454a6e717ff9860a29b7791518a690d3d

Cherry-pick the following commit, which adds retries to make sure the
    Leap
Motion controller is connected:
~$ cd OSVR-Leap-Motion
~/OSVR-Leap-Motion$ git cherry-pick 3810
    b93d908a943bb934fe739171f9f4e21e575c

Apply the following patch to fix a compilation error:
--- a/Tracker.cpp
+++ b/Tracker.cpp
@@ -15,7 +15,7 @@ Tracker::Tracker(const osvr::pluginkit::DeviceToken&
    pDeviceToken,

                            mTrackerInterface(NULL) {
      // note we have two skeleton sensors, one for each hand, each with
    their own spec
      // this is because the articulation specs for each hand are not
    connected by a parent joint
-     osvrDeviceSkeletonConfigure(pOptions, &mSkeletonInterface,
    com_osvr_LeapMotion_json, 2);
+     osvrDeviceSkeletonConfigure(pOptions, &mSkeletonInterface,
    com_osvr_LeapMotion_json);

         osvrDeviceTrackerConfigure(pOptions, &mTrackerInterface);

Compile and Install OSVR-Leap-Motion:
~/OSVR-Leap-Motion$ mkdir build
~/OSVR-Leap-Motion$ cd build
~/OSVR-Leap-Motion/build$ cmake -DLEAPMOTION_LEAP_LIBRARY_RELEASE=~/
    LeapSDK/lib/x64/libLeap.so -DLEAPMOTION_ROOT_DIR=~/ ..
~/OSVR-Leap-Motion/build$ make
~/OSVR-Leap-Motion/build$ sudo make install

Start the Leap Motion daemon:
~$ sudo service leapd start

Start the OSVR server:
~$ sudo osvr_server
```

```
-------------------------
   Xbox 360 Controller
-------------------------

Install the Xbox 360 controller Linux driver:
~$ sudo apt install xboxdrv

Clone the VRPN repo:
~$ git clone https://github.com/vrpn/vrpn.git

Install VRPN:
~$ cd vrpn
~/vrpn$ mkdir build && cd build && cmake .. && make && sudo make install

Start the Xbox 360 controller driver:
~$ sudo xboxdrv --detach-kernel-driver --daemon

Check the ID assigned by the system to the controller ("js0" or "js1"):
~$ cat /proc/bus/input/devices | grep js

Uncomment the following joystick support within the VRPN config file (/
    usr/local/etc/vrpn.cfg),
making sure that the ID matches the one obtained before ("js0" or "js1")
    :
vrpn_Joylin Joylin0 /dev/input/js0

Add virtual trackers to the VRPN configuration (/usr/local/etc/vrpn.cfg)
     that use the controller's analog sticks:
vrpn_Tracker_AnalogFly  Tracker0    60.0    absolute
X   *Joylin0    0   0.0 0.0 1.0 1.0
Y   *Joylin0    1   0.0 0.0 -1.0    1.0

vrpn_Tracker_AnalogFly  Tracker1    60.0    absolute
X   *Joylin0    2   0.0 0.0 1.0 1.0
Y   *Joylin0    3   0.0 0.0 -1.0    1.0

Add Xbox 360 tracker and button devices to the OSVR configuration (/usr/
    local/share/osvrcore/osvr_server_config.json):
    "externalDevices": {
        "/Xbox_360_LeftTracker": {
            "deviceName": "Tracker0",
            "server": "localhost:3884",
            "descriptor": {
                "interfaces": {
                    "tracker": {
                        "count": 1
                    }
                },
                "automaticAliases": {
                    "/me/hands/left": "/Xbox_360_LeftTracker/tracker/0"
                }
            }
        },
        "/Xbox_360_RightTracker": {
            "deviceName": "Tracker1",
```

```
            "server": "localhost:3884",
            "descriptor": {
                "interfaces": {
                    "tracker": {
                        "count": 1
                    }
                },
                "automaticAliases": {
                    "/me/hands/right": "/Xbox_360_RightTracker/tracker
    /0"
                }
            }
        },
        "/Xbox_360_Buttons": {
            "deviceName": "Joylin0",
            "server": "localhost:3884",
            "descriptor": {
                "interfaces": {
                    "button": {
                        "count": 14
                    }
                },
                "automaticAliases": {
                    "/controller/left/1": "/Xbox_360_Buttons/button/4",
                    "/controller/right/1": "/Xbox_360_Buttons/button/5"
                }
            }
        }
    },
```

```
Start the VRPN server on port 3884 so it doesn't clash with the OSVR
    server's default port:
~$ sudo vrpn_server -f /usr/local/etc/vrpn.cfg 3884

Start the OSVR server:
~$ sudo osvr_server /usr/local/share/osvrcore/osvr_server_config.json

------------
   Mouse
------------

Look for the name of your mouse device in the Linux input system:
~$ cat /proc/bus/input/devices | grep Name | grep -i mouse

Use that name to add a mouse device to the VRPN configuration (/usr/
    local/etc/vrpn.cfg). For example:
vrpn_DevInput           Mouse0          "Logitech G700s Rechargeable
    Gaming Mouse" relative -1

Add a virtual tracker to the VRPN configuration (/usr/local/etc/vrpn.cfg
    ) that uses the mouse device:
vrpn_Tracker_AnalogFly Tracker0 60.0 absolute
X  *Mouse0 0 0.5 0.0 1.0 1.0
Y  *Mouse0 1 0.5 0.0 -1.0 1.0
Z  NULL 2 0.0 0.0 1.0 1.0
```

```
RX NULL 3 0.0 0.0 1.0 1.0
RY NULL 4 0.0 0.0 1.0 1.0
RZ NULL 5 0.0 0.0 1.0 1.0
RESET NULL 0
CLUTCH NULL 0

Add mouse tracker and mouse button devices to the OSVR configuration (/
    usr/local/share/osvrcore/osvr_server_config.json):
    "externalDevices": {
        "/MouseTracker": {
            "deviceName": "Tracker0",
            "server": "localhost:3884",
            "descriptor": {
                "interfaces": {
                    "tracker": {
                        "count": 1
                    }
                },
                "automaticAliases": {
                    "/me/hands/right": "/MouseTracker/tracker/0"
                }
            }
        },
        "/MouseButtons": {
            "deviceName": "Mouse0",
            "server": "localhost:3884",
            "descriptor": {
                "interfaces": {
                    "button": {
                        "count": 7
                    }
                },
                "automaticAliases": {
                    "/controller/right/1": "/MouseButtons/button/4"
                }
            }
        }
    }

In the example above, we used mouse button 4, but you can use any button
     on your mouse.

Start the VRPN server on port 3884 so it doesn't clash with the OSVR
    server's default port:
~$ sudo vrpn_server -f /usr/local/etc/vrpn.cfg 3884

Start the OSVR server:
~$ sudo osvr_server /usr/local/share/osvrcore/osvr_server_config.json

------------
   Wacom
------------

Look for the name of your Wacom pen device in the Linux input system:
~$ cat /proc/bus/input/devices | grep Name | grep Wacom
```

```
Use that name to add a Wacom pen device to the VRPN configuration (/usr/
    local/etc/vrpn.cfg). For example:
vrpn_DevInput           Wacom0          "Wacom Intuos PT S Pen" absolute
     1920

Add a virtual tracker to the VRPN configuration (/usr/local/etc/vrpn.cfg
    ) that uses the Wacom device:
vrpn_Tracker_AnalogFly Tracker0 60.0 absolute
X  *Wacom0 0 0.0 0.0 1.0 1.0
Y  *Wacom0 1 0.0 0.0 -1.0 1.0
Z  NULL 2 0.0 0.0 1.0 1.0
RX NULL 3 0.0 0.0 1.0 1.0
RY NULL 4 0.0 0.0 1.0 1.0
RZ NULL 5 0.0 0.0 1.0 1.0
RESET NULL 0
CLUTCH NULL 0

Add Wacom tracker and Wacom button devices to the OSVR configuration (/
    usr/local/share/osvrcore/osvr_server_config.json):
    "externalDevices": {
        "/WacomTracker": {
            "deviceName": "Tracker0",
            "server": "localhost:3884",
            "descriptor": {
                "interfaces": {
                    "tracker": {
                        "count": 1
                    }
                },
                "automaticAliases": {
                    "/me/hands/right": "/WacomTracker/tracker/0"
                }
            }
        },
        "/WacomButtons": {
            "deviceName": "Wacom0",
            "server": "localhost:3884",
            "descriptor": {
                "interfaces": {
                    "button": {
                        "count": 80
                    }
                },
                "automaticAliases": {
                    "/controller/right/1": "/WacomButtons/button/59"
                }
            }
        }
    }

In the example above, we used Wacom button 59, which corresponds to our
    Wacom pen's main button, but you can use any button on your device.
```

```
Start the VRPN server on port 3884 so it doesn't clash with the OSVR
    server's default port:
~$ sudo vrpn_server -f /usr/local/etc/vrpn.cfg 3884

Start the OSVR server:
~$ sudo osvr_server /usr/local/share/osvrcore/osvr_server_config.json
```

Apéndice B

Apéndice B: Bitácoras de Experimentos

B.1. Experimento 1: Razer Hydra - Grandes

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	OpenChrome	1	417	OpenChrome	1	27
1	2	NewTab	Normal	NewTab	1	251	NewTab	1	4
1	3	OpenCNN	Normal	OpenCNN	1	1118	OpenCNN	1	4
1	4	ZoomIn	Normal	ZoomIn	1	76	ZoomIn	1	4
1	5	ZoomOut	Normal	ZoomOut	1	68	ZoomOut	1	4
1	6	RefreshTab	Normal	RefreshTab	1	343	RefreshTab	1	4
1	7	RefreshTab	Invertido	OpenGoogle	0	304	RefreshTab	1	4
1	8	CloseTab	Normal	CloseTab	1	591	CloseTab	1	4
1	9	NewTab	Normal	NewTab	1	167	NewTab	1	4
1	10	OpenGoogle	Normal	OpenGoogle	1	1225	OpenGoogle	1	4
1	11	CloseTab	Normal	CloseTab	1	366	CloseTab	1	4
1	12	CloseTab	Normal	CloseTab	1	658	CloseTab	1	4
2	1	OpenChrome	Normal	OpenChrome	1	264	OpenChrome	1	60
2	2	NewTab	Normal	NewTab	1	485	NewTab	1	4
2	3	OpenCNN	Normal	OpenCNN	1	461	OpenCNN	1	4
2	4	ZoomIn	Normal	ZoomIn	1	65	ZoomIn	1	4
2	5	ZoomOut	Normal	ZoomOut	1	99	ZoomOut	1	4
2	6	RefreshTab	Normal	RefreshTab	1	386	RefreshTab	1	4
2	7	RefreshTab	Invertido	OpenGoogle	0	470	RefreshTab	1	4
2	8	CloseTab	Normal	CloseTab	1	165	CloseTab	1	4
2	9	NewTab	Normal	NewTab	1	325	NewTab	1	4
2	10	OpenGoogle	Normal	OpenGoogle	1	1039	RefreshTab	0	4
2	12	CloseTab	Normal	CloseTab	1	152	CloseTab	1	4
2	13	CloseTab	Normal	CloseTab	1	366	CloseTab	1	4
3	1	OpenChrome	Normal	OpenChrome	1	249	OpenChrome	1	61
3	2	NewTab	Normal	NewTab	1	117	NewTab	1	4
3	3	OpenCNN	Normal	OpenCNN	1	671	OpenCNN	1	6
3	4	ZoomIn	Normal	ZoomIn	1	79	ZoomIn	1	4
3	5	ZoomOut	Normal	ZoomOut	1	23	ZoomOut	1	10
3	6	RefreshTab	Normal	RefreshTab	1	324	OpenGoogle	0	4
3	8	RefreshTab	Invertido	OpenGoogle	0	450	RefreshTab	1	4
3	9	CloseTab	Normal	CloseTab	1	189	CloseTab	1	4
3	10	NewTab	Normal	NewTab	1	193	NewTab	1	4
3	11	OpenGoogle	Normal	OpenGoogle	1	100	OpenGoogle	1	4
3	12	CloseTab	Normal	CloseTab	1	751	CloseTab	1	4
3	13	CloseTab	Normal	CloseTab	1	629	CloseTab	1	4
4	1	OpenChrome	Normal	OpenChrome	1	85	OpenChrome	1	4
4	2	NewTab	Normal	NewTab	1	85	NewTab	1	4
4	3	OpenCNN	Normal	OpenCNN	1	908	OpenCNN	1	4
4	4	ZoomIn	Normal	ZoomIn	1	71	ZoomIn	1	5
4	5	ZoomOut	Normal	ZoomOut	1	52	ZoomOut	1	4
4	6	RefreshTab	Normal	N/A	0	724	OpenChrome	0	4
4	11	RefreshTab	Invertido	OpenCNN	0	2711	RefreshTab	1	4
4	12	CloseTab	Normal	CloseTab	1	592	CloseTab	1	4
4	13	NewTab	Normal	NewTab	1	263	NewTab	1	4

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
4	14	OpenGoogle	Normal	OpenGoogle	1	3471	OpenGoogle	1	4
4	15	CloseTab	Normal	CloseTab	1	487	CloseTab	1	4
4	16	CloseTab	Normal	CloseTab	1	456	CloseTab	1	4
5	1	OpenChrome	Normal	OpenChrome	1	242	OpenChrome	1	4
5	2	NewTab	Normal	NewTab	1	746	NewTab	1	4
5	3	OpenCNN	Normal	OpenCNN	1	3838	OpenCNN	1	5
5	4	ZoomIn	Normal	ZoomIn	1	71	ZoomIn	1	4
5	5	ZoomOut	Normal	ZoomOut	1	33	ZoomOut	1	4
5	6	RefreshTab	Normal	RefreshTab	1	1559	RefreshTab	1	4
5	7	RefreshTab	Invertido	OpenGoogle	0	296	RefreshTab	1	4
5	8	CloseTab	Normal	CloseTab	1	754	CloseTab	1	4
5	9	NewTab	Normal	NewTab	1	1065	NewTab	1	4
5	10	OpenGoogle	Normal	OpenGoogle	1	1603	RefreshTab	0	4
5	13	CloseTab	Normal	CloseTab	1	263	CloseTab	1	4
5	14	CloseTab	Normal	CloseTab	1	458	CloseTab	1	4
6	1	OpenChrome	Normal	OpenChrome	1	604	OpenChrome	1	11
6	1	NewTab	Normal	NewTab	1	272	NewTab	1	4
6	2	OpenCNN	Normal	OpenCNN	1	941	OpenCNN	1	4
6	3	ZoomIn	Normal	N/A	0	177	ZoomIn	1	4
6	4	ZoomOut	Normal	ZoomOut	1	107	ZoomOut	1	4
6	5	RefreshTab	Normal	RefreshTab	1	456	OpenGoogle	0	4
6	7	RefreshTab	Invertido	OpenCNN	0	371	RefreshTab	1	4
6	8	CloseTab	Normal	CloseTab	1	3360	CloseTab	1	4
6	9	NewTab	Normal	NewTab	1	470	NewTab	1	4
6	10	OpenGoogle	Normal	OpenGoogle	1	861	OpenGoogle	1	4
6	11	CloseTab	Normal	CloseTab	1	1068	CloseTab	1	4
6	12	CloseTab	Normal	CloseTab	1	806	CloseTab	1	4

B.2. Experimento 1: Razer Hydra - Pequeños

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	CloseTab	0	55	OpenChrome	1	70
1	2	NewTab	Normal	NewTab	1	71	NewTab	1	3
1	3	OpenCNN	Normal	OpenCNN	1	118	OpenCNN	1	4
1	4	ZoomIn	Normal	ZoomIn	1	9	ZoomIn	1	4
1	5	ZoomOut	Normal	ZoomOut	1	30	N/A	0	6
1	7	RefreshTab	Normal	RefreshTab	1	284	RefreshTab	1	4
1	8	RefreshTab	Invertido	OpenGoogle	0	84	RefreshTab	1	4
1	9	CloseTab	Normal	CloseTab	1	86	CloseTab	1	4
1	10	NewTab	Normal	NewTab	1	132	NewTab	1	4
1	11	OpenGoogle	Normal	OpenChrome	0	83	OpenGoogle	1	4
1	12	CloseTab	Normal	CloseTab	1	69	CloseTab	1	4
1	13	CloseTab	Normal	CloseTab	1	362	CloseTab	1	4
2	1	OpenChrome	Normal	OpenChrome	1	232	OpenChrome	1	10
2	2	NewTab	Normal	NewTab	1	32	NewTab	1	4
2	3	OpenCNN	Normal	OpenCNN	1	343	OpenCNN	1	4
2	4	ZoomIn	Normal	ZoomIn	1	34	ZoomIn	1	4
2	5	ZoomOut	Normal	ZoomOut	1	24	ZoomOut	1	11
2	6	RefreshTab	Normal	RefreshTab	1	622	RefreshTab	1	4
2	7	RefreshTab	Invertido	OpenGoogle	0	126	RefreshTab	1	4
2	8	CloseTab	Normal	CloseTab	1	87	CloseTab	1	4
2	9	NewTab	Normal	NewTab	1	144	NewTab	1	4
2	10	OpenGoogle	Normal	OpenGoogle	1	174	N/A	0	4
2	12	CloseTab	Normal	CloseTab	1	400	CloseTab	1	4
2	13	CloseTab	Normal	CloseTab	1	75	NewTab	0	4
3	1	OpenChrome	Normal	OpenChrome	1	136	OpenChrome	1	4
3	2	NewTab	Normal	NewTab	1	73	NewTab	1	4
3	3	OpenCNN	Normal	OpenCNN	1	192	OpenCNN	1	4
3	4	ZoomIn	Normal	ZoomIn	1	22	ZoomIn	1	4
3	5	ZoomOut	Normal	ZoomOut	1	16	ZoomOut	1	4
3	6	RefreshTab	Normal	RefreshTab	1	80	RefreshTab	1	4
3	7	RefreshTab	Invertido	OpenGoogle	0	118	RefreshTab	1	4
3	8	CloseTab	Normal	CloseTab	1	204	CloseTab	1	4
3	9	NewTab	Normal	NewTab	1	163	NewTab	1	4
3	10	OpenGoogle	Normal	OpenGoogle	1	69	OpenGoogle	1	4
3	11	CloseTab	Normal	CloseTab	1	275	CloseTab	1	4
3	12	CloseTab	Normal	CloseTab	1	454	CloseTab	1	4
4	1	OpenChrome	Normal	Open	0	85	OpenChrome	1	5
4	2	NewTab	Normal	NewTab	1	258	NewTab	1	4
4	3	OpenCNN	Normal	OpenCNN	1	392	OpenCNN	1	4
4	4	ZoomIn	Normal	ZoomIn	1	83	ZoomIn	1	4
4	5	ZoomOut	Normal	ZoomOut	1	17	ZoomOut	1	4
4	6	RefreshTab	Normal	RefreshTab	1	165	OpenGoogle	0	4
4	8	RefreshTab	Invertido	OpenGoogle	0	520	RefreshTab	1	4
4	9	CloseTab	Normal	CloseTab	1	195	CloseTab	1	4
4	10	NewTab	Normal	NewTab	1	171	NewTab	1	4

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
4	11	OpenGoogle	Normal	OpenGoogle	1	1537	RefreshTab	0	4
4	14	CloseTab	Normal	CloseTab	1	264	CloseTab	1	4
4	15	CloseTab	Normal	CloseTab	1	301	CloseTab	1	4
5	1	OpenChrome	Normal	OpenChrome	1	14	OpenChrome	1	4
5	2	NewTab	Normal	NewTab	1	66	NewTab	1	4
5	3	OpenCNN	Normal	OpenCNN	1	518	OpenCNN	1	4
5	4	ZoomIn	Normal	N/A	0	39	ZoomIn	1	5
5	5	ZoomOut	Normal	N/A	0	41	ZoomOut	1	5
5	6	RefreshTab	Normal	RefreshTab	1	272	N/A	0	4
5	8	RefreshTab	Invertido	OpenGoogle	0	39	RefreshTab	1	4
5	9	CloseTab	Normal	CloseTab	1	77	CloseTab	1	4
5	10	NewTab	Normal	NewTab	1	66	NewTab	1	4
5	11	OpenGoogle	Normal	OpenGoogle	1	105	OpenGoogle	1	4
5	12	CloseTab	Normal	CloseTab	1	413	CloseTab	1	4
5	13	CloseTab	Normal	CloseTab	1	243	CloseTab	1	4
6	1	OpenChrome	Normal	OpenChrome	1	83	OpenChrome	1	4
6	2	NewTab	Normal	NewTab	1	236	NewTab	1	4
6	3	OpenCNN	Normal	OpenCNN	1	1064	OpenCNN	1	4
6	4	ZoomIn	Normal	ZoomIn	1	19	ZoomIn	1	5
6	5	ZoomOut	Normal	ZoomOut	1	17	ZoomOut	1	14
6	6	RefreshTab	Normal	RefreshTab	1	277	OpenGoogle	0	3
6	9	RefreshTab	Invertido	OpenGoogle	0	584	RefreshTab	1	4
6	10	CloseTab	Normal	CloseTab	1	377	CloseTab	1	4
6	11	NewTab	Normal	NewTab	1	86	NewTab	1	4
6	12	OpenGoogle	Normal	OpenGoogle	1	105	OpenGoogle	1	4
6	13	CloseTab	Normal	CloseTab	1	159	CloseTab	1	4
6	14	CloseTab	Normal	CloseTab	1	335	CloseTab	1	4

B.3. Experimento 2: Wacom - Grandes

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	OpenChrome	1	1227	OpenChrome	1	10
1	2	NewTab	Normal	NewTab	1	326	NewTab	1	4
1	3	OpenCNN	Normal	OpenCNN	1	5575	OpenCNN	1	4
1	4	RefreshTab	Normal	RefreshTab	1	360	RefreshTab	1	4
1	5	RefreshTab	Invertido	OpenCNN	0	1207	RefreshTab	1	4
1	6	CloseTab	Normal	CloseTab	1	494	CloseTab	1	4
1	7	NewTab	Normal	NewTab	1	2218	NewTab	1	4
1	8	OpenGoogle	Normal	OpenGoogle	1	474	N/A	0	4
1	10	CloseTab	Normal	CloseTab	1	895	CloseTab	1	4
1	11	CloseTab	Normal	CloseTab	1	2329	CloseTab	1	4
2	1	OpenChrome	Normal	OpenChrome	1	736	OpenChrome	1	32
2	2	NewTab	Normal	NewTab	1	1760	NewTab	1	4
2	3	OpenCNN	Normal	OpenCNN	1	1987	OpenCNN	1	4
2	4	RefreshTab	Normal	RefreshTab	1	3301	RefreshTab	1	4
2	5	RefreshTab	Invertido	OpenGoogle	0	815	RefreshTab	1	4
2	6	CloseTab	Normal	CloseTab	1	2626	CloseTab	1	4
2	7	NewTab	Normal	NewTab	1	436	NewTab	1	4
2	8	OpenGoogle	Normal	OpenGoogle	1	686	OpenGoogle	1	4
2	9	CloseTab	Normal	CloseTab	1	491	CloseTab	1	4
2	10	CloseTab	Normal	CloseTab	1	328	CloseTab	1	4
3	1	OpenChrome	Normal	OpenChrome	1	1235	OpenChrome	1	4
3	2	NewTab	Normal	NewTab	1	502	NewTab	1	4
3	3	OpenCNN	Normal	OpenCNN	1	1517	OpenCNN	1	4
3	4	RefreshTab	Normal	RefreshTab	1	364	RefreshTab	1	4
3	5	RefreshTab	Invertido	OpenGoogle	0	1349	RefreshTab	1	4
3	6	CloseTab	Normal	CloseTab	1	330	CloseTab	1	4
3	7	NewTab	Normal	NewTab	1	299	NewTab	1	4
3	8	OpenGoogle	Normal	OpenGoogle	1	2671	OpenGoogle	1	4
3	9	CloseTab	Normal	CloseTab	1	1684	CloseTab	1	4
3	10	CloseTab	Normal	CloseTab	1	561	CloseTab	1	4
4	1	OpenChrome	Normal	OpenChrome	1	777	OpenChrome	1	5
4	2	NewTab	Normal	NewTab	1	308	NewTab	1	4
4	3	OpenCNN	Normal	OpenCNN	1	8343	N/A	0	4
4	5	RefreshTab	Normal	RefreshTab	1	1409	OpenGoogle	0	5
4	8	RefreshTab	Invertido	OpenGoogle	0	2067	RefreshTab	1	4
4	9	CloseTab	Normal	CloseTab	1	473	CloseTab	1	4
4	10	NewTab	Normal	NewTab	1	1515	NewTab	1	4
4	11	OpenGoogle	Normal	OpenGoogle	1	1267	RefreshTab	0	4
4	13	CloseTab	Normal	CloseTab	1	1510	CloseTab	1	4
4	14	CloseTab	Normal	CloseTab	1	369	CloseTab	1	4
5	1	OpenChrome	Normal	OpenChrome	1	1060	OpenChrome	1	4
5	2	NewTab	Normal	NewTab	1	509	NewTab	1	4
5	3	OpenCNN	Normal	OpenCNN	1	4373	OpenCNN	1	4
5	4	RefreshTab	Normal	RefreshTab	1	623	RefreshTab	1	5
5	5	RefreshTab	Invertido	OpenGoogle	0	4152	N/A	0	4

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
5	7	CloseTab	Normal	CloseTab	1	881	CloseTab	1	4
5	8	NewTab	Normal	NewTab	1	1612	NewTab	1	4
5	9	OpenGoogle	Normal	OpenGoogle	1	2731	RefreshTab	0	4
5	12	CloseTab	Normal	CloseTab	1	1538	CloseTab	1	4
5	13	CloseTab	Normal	CloseTab	1	1931	CloseTab	1	4
6	1	OpenChrome	Normal	OpenChrome	1	332	OpenChrome	1	4
6	2	NewTab	Normal	NewTab	1	1747	NewTab	1	4
6	3	OpenCNN	Normal	OpenCNN	1	5263	OpenCNN	1	4
6	4	RefreshTab	Normal	RefreshTab	1	5201	OpenGoogle	0	4
6	7	RefreshTab	Invertido	OpenGoogle	0	6403	RefreshTab	1	4
6	8	CloseTab	Normal	CloseTab	1	369	CloseTab	1	4
6	9	NewTab	Normal	NewTab	1	1126	NewTab	1	4
6	10	OpenGoogle	Normal	OpenGoogle	1	4072	OpenGoogle	1	4
6	11	CloseTab	Normal	CloseTab	1	1088	CloseTab	1	4
6	12	CloseTab	Normal	CloseTab	1	2336	CloseTab	1	4

B.4. Experimento 2: Wacom - Pequeños

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	OpenChrome	1	194	OpenChrome	1	13
1	2	NewTab	Normal	NewTab	1	234	NewTab	1	4
1	3	OpenCNN	Normal	OpenCNN	1	5294	OpenCNN	1	4
1	4	RefreshTab	Normal	RefreshTab	1	353	RefreshTab	1	4
1	5	RefreshTab	Invertido	OpenGoogle	0	3130	RefreshTab	1	4
1	6	CloseTab	Normal	CloseTab	1	492	CloseTab	1	4
1	7	NewTab	Normal	NewTab	1	1607	NewTab	1	4
1	8	OpenGoogle	Normal	OpenGoogle	1	2906	OpenGoogle	1	4
1	9	CloseTab	Normal	CloseTab	1	1500	CloseTab	1	4
1	10	CloseTab	Normal	CloseTab	1	940	CloseTab	1	4
2	1	OpenChrome	Normal	OpenChrome	1	425	OpenChrome	1	11
2	2	NewTab	Normal	NewTab	1	203	NewTab	1	4
2	3	OpenCNN	Normal	OpenCNN	1	1445	OpenCNN	1	4
2	4	RefreshTab	Normal	RefreshTab	1	1118	RefreshTab	1	4
2	5	RefreshTab	Invertido	OpenGoogle	0	2602	RefreshTab	1	4
2	6	CloseTab	Normal	CloseTab	1	846	CloseTab	1	4
2	7	NewTab	Normal	NewTab	1	338	NewTab	1	4
2	8	OpenGoogle	Normal	OpenGoogle	1	3271	RefreshTab	0	5
2	10	CloseTab	Normal	CloseTab	1	490	CloseTab	1	4
2	11	CloseTab	Normal	CloseTab	1	457	CloseTab	1	4
3	1	OpenChrome	Normal	OpenGoogle	0	480	N/A	0	5
3	4	NewTab	Normal	NewTab	1	522	NewTab	1	4
3	5	OpenCNN	Normal	OpenCNN	1	1126	OpenCNN	1	4
3	6	RefreshTab	Normal	RefreshTab	1	652	RefreshTab	1	4
3	7	RefreshTab	Invertido	N/A	0	233	RefreshTab	1	4
3	8	CloseTab	Normal	CloseTab	1	961	CloseTab	1	6
3	9	NewTab	Normal	NewTab	1	520	NewTab	1	4
3	10	OpenGoogle	Normal	OpenGoogle	1	1248	OpenGoogle	1	4
3	11	CloseTab	Normal	CloseTab	1	419	CloseTab	1	4
3	12	CloseTab	Normal	CloseTab	1	411	CloseTab	1	4
4	1	OpenChrome	Normal	OpenChrome	1	1507	OpenChrome	1	4
4	2	NewTab	Normal	NewTab	1	1121	NewTab	1	4
4	3	OpenCNN	Normal	OpenCNN	1	3798	OpenCNN	1	4
4	4	RefreshTab	Normal	RefreshTab	1	5265	RefreshTab	1	4
4	5	RefreshTab	Invertido	N/A	0	1860	RefreshTab	1	4
4	6	CloseTab	Normal	CloseTab	1	480	CloseTab	1	4
4	7	NewTab	Normal	NewTab	1	2876	NewTab	1	4
4	8	OpenGoogle	Normal	OpenGoogle	1	1573	RefreshTab	0	4
4	10	CloseTab	Normal	CloseTab	1	1174	CloseTab	1	4
4	11	CloseTab	Normal	CloseTab	1	902	CloseTab	1	4
5	1	OpenChrome	Normal	OpenChrome	1	42	OpenChrome	1	5
5	2	NewTab	Normal	NewTab	1	166	NewTab	1	4
5	3	OpenCNN	Normal	OpenCNN	1	6993	OpenCNN	1	4
5	4	RefreshTab	Normal	RefreshTab	1	914	OpenGoogle	0	4
5	6	RefreshTab	Invertido	OpenGoogle	0	489	RefreshTab	1	4

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
5	7	CloseTab	Normal	CloseTab	1	520	CloseTab	1	4
5	8	NewTab	Normal	NewTab	1	591	NewTab	1	4
5	9	OpenGoogle	Normal	OpenGoogle	1	299	OpenGoogle	1	4
5	10	CloseTab	Normal	CloseTab	1	1211	CloseTab	1	4
5	11	CloseTab	Normal	CloseTab	1	233	CloseTab	1	4
6	1	OpenChrome	Normal	OpenChrome	1	166	OpenChrome	1	4
6	2	NewTab	Normal	NewTab	1	212	NewTab	1	4
6	3	OpenCNN	Normal	OpenCNN	1	4099	OpenCNN	1	4
6	4	RefreshTab	Normal	RefreshTab	1	629	RefreshTab	1	4
6	5	RefreshTab	Invertido	OpenGoogle	0	2605	RefreshTab	1	4
6	6	CloseTab	Normal	CloseTab	1	435	CloseTab	1	4
6	7	NewTab	Normal	NewTab	1	593	NewTab	1	4
6	8	OpenGoogle	Normal	OpenGoogle	1	941	OpenGoogle	1	4
6	9	CloseTab	Normal	CloseTab	1	299	CloseTab	1	4
6	10	CloseTab	Normal	CloseTab	1	298	CloseTab	1	4

B.5. Experimento 2: Leap Motion - Grandes

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	OpenChrome	1	203	OpenChrome	1	16
1	2	NewTab	Normal	NewTab	1	1007	NewTab	1	4
1	3	OpenCNN	Normal	OpenCNN	1	1809	OpenCNN	1	4
1	4	ZoomIn	Normal	ZoomIn	1	135	ZoomIn	1	4
1	5	ZoomOut	Normal	N/A	0	151	ZoomOut	1	4
1	6	RefreshTab	Normal	RefreshTab	1	775	RefreshTab	1	4
1	7	RefreshTab	Invertido	OpenGoogle	0	2364	RefreshTab	1	4
1	8	CloseTab	Normal	CloseTab	1	2016	CloseTab	1	4
1	9	NewTab	Normal	NewTab	1	1245	NewTab	1	4
1	10	OpenGoogle	Normal	OpenGoogle	1	2424	N/A	0	4
1	12	CloseTab	Normal	CloseTab	1	1593	CloseTab	1	4
1	13	CloseTab	Normal	CloseTab	1	2030	CloseTab	1	4
2	1	OpenChrome	Normal	N/A	0	259	OpenChrome	1	4
2	2	NewTab	Normal	NewTab	1	38	NewTab	1	5
2	3	OpenCNN	Normal	OpenCNN	1	3144	OpenCNN	1	4
2	4	ZoomIn	Normal	ZoomIn	1	30	ZoomIn	1	6
2	5	ZoomOut	Normal	ZoomOut	1	31	ZoomOut	1	5
2	6	RefreshTab	Normal	RefreshTab	1	951	RefreshTab	1	4
2	7	RefreshTab	Invertido	OpenGoogle	0	359	RefreshTab	1	4
2	8	CloseTab	Normal	CloseTab	1	1274	CloseTab	1	4
2	9	NewTab	Normal	NewTab	1	95	NewTab	1	4
2	10	OpenGoogle	Normal	OpenGoogle	1	194	OpenGoogle	1	4
2	11	CloseTab	Normal	CloseTab	1	1275	CloseTab	1	4
2	12	CloseTab	Normal	CloseTab	1	871	CloseTab	1	4
3	1	OpenChrome	Normal	N/A	0	122	NewTab	0	4
3	3	NewTab	Normal	NewTab	1	101	NewTab	1	5
3	4	OpenCNN	Normal	OpenCNN	1	375	OpenCNN	1	4
3	5	ZoomIn	Normal	ZoomIn	1	9	ZoomIn	1	7
3	6	ZoomOut	Normal	ZoomOut	1	23	ZoomOut	1	5
3	7	RefreshTab	Normal	RefreshTab	1	88	RefreshTab	1	4
3	8	RefreshTab	Invertido	N/A	0	31	RefreshTab	1	4
3	9	CloseTab	Normal	CloseTab	1	457	CloseTab	1	5
3	10	NewTab	Normal	NewTab	1	258	NewTab	1	4
3	11	OpenGoogle	Normal	OpenGoogle	1	93	OpenGoogle	1	4
3	12	CloseTab	Normal	CloseTab	1	527	CloseTab	1	10
3	13	CloseTab	Normal	CloseTab	1	120	CloseTab	1	4
4	1	OpenChrome	Normal	OpenChrome	1	50	OpenChrome	1	4
4	2	NewTab	Normal	NewTab	1	104	NewTab	1	4
4	3	OpenCNN	Normal	OpenCNN	1	1557	OpenCNN	1	4
4	4	ZoomIn	Normal	ZoomIn	1	25	ZoomIn	1	9
4	5	ZoomOut	Normal	ZoomOut	1	26	ZoomOut	1	5
4	6	RefreshTab	Normal	RefreshTab	1	531	RefreshTab	1	4
4	7	RefreshTab	Invertido	N/A	0	199	RefreshTab	1	4
4	8	CloseTab	Normal	N/A	0	85	CloseTab	1	4
4	9	NewTab	Normal	NewTab	1	122	NewTab	1	4

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
4	10	OpenGoogle	Normal	OpenGoogle	1	96	RefreshTab	0	4
4	15	CloseTab	Normal	CloseTab	1	101	CloseTab	1	4
4	16	CloseTab	Normal	CloseTab	1	264	CloseTab	1	4
5	1	OpenChrome	Normal	OpenChrome	1	274	OpenChrome	1	5
5	2	NewTab	Normal	NewTab	1	110	NewTab	1	4
5	3	OpenCNN	Normal	OpenCNN	1	767	OpenCNN	1	4
5	4	ZoomIn	Normal	ZoomIn	1	29	ZoomIn	1	5
5	5	ZoomOut	Normal	N/A	0	49	ZoomOut	1	4
5	6	RefreshTab	Normal	RefreshTab	1	210	RefreshTab	1	4
5	7	RefreshTab	Invertido	OpenGoogle	0	130	RefreshTab	1	4
5	8	CloseTab	Normal	CloseTab	1	110	CloseTab	1	4
5	9	NewTab	Normal	NewTab	1	47	NewTab	1	5
5	10	OpenGoogle	Normal	OpenGoogle	1	205	OpenGoogle	1	4
5	11	CloseTab	Normal	CloseTab	1	399	CloseTab	1	4
5	12	CloseTab	Normal	CloseTab	1	96	CloseTab	1	4
6	1	OpenChrome	Normal	OpenChrome	1	90	OpenChrome	1	73
6	2	NewTab	Normal	NewTab	1	92	NewTab	1	4
6	3	OpenCNN	Normal	OpenCNN	1	567	OpenCNN	1	4
6	4	ZoomIn	Normal	ZoomIn	1	39	ZoomIn	1	5
6	5	ZoomOut	Normal	ZoomOut	1	54	ZoomOut	1	4
6	6	RefreshTab	Normal	RefreshTab	1	218	RefreshTab	1	4
6	7	RefreshTab	Invertido	OpenGoogle	0	199	RefreshTab	1	4
6	8	CloseTab	Normal	CloseTab	1	402	CloseTab	1	4
6	9	NewTab	Normal	NewTab	1	267	NewTab	1	4
6	10	OpenGoogle	Normal	OpenGoogle	1	124	OpenGoogle	1	4
6	11	CloseTab	Normal	CloseTab	1	208	CloseTab	1	4
6	12	CloseTab	Normal	CloseTab	1	203	CloseTab	1	4

B.6. Experimento 2: Leap Motion - Pequeños

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	OpenChrome	1	65	OpenChrome	1	9
1	2	NewTab	Normal	NewTab	1	98	NewTab	1	4
1	3	OpenCNN	Normal	OpenCNN	1	1608	OpenCNN	1	4
1	4	ZoomIn	Normal	ZoomIn	1	3	ZoomIn	1	4
1	5	ZoomOut	Normal	ZoomOut	1	42	ZoomOut	1	5
1	6	RefreshTab	Normal	RefreshTab	1	635	OpenGoogle	0	4
1	8	RefreshTab	Invertido	OpenGoogle	0	778	RefreshTab	1	4
1	9	CloseTab	Normal	CloseTab	1	463	CloseTab	1	4
1	10	NewTab	Normal	NewTab	1	199	NewTab	1	4
1	11	OpenGoogle	Normal	OpenGoogle	1	213	OpenGoogle	1	4
1	12	CloseTab	Normal	CloseTab	1	136	CloseTab	1	4
1	13	CloseTab	Normal	CloseTab	1	617	CloseTab	1	4
2	1	OpenChrome	Normal	OpenChrome	1	790	OpenChrome	1	4
2	2	NewTab	Normal	NewTab	1	277	NewTab	1	4
2	3	OpenCNN	Normal	OpenCNN	1	631	OpenCNN	1	4
2	4	ZoomIn	Normal	ZoomIn	1	53	ZoomOut	0	4
2	6	ZoomOut	Normal	N/A	0	161	ZoomOut	1	4
2	7	RefreshTab	Normal	RefreshTab	1	144	RefreshTab	1	4
2	8	RefreshTab	Invertido	OpenGoogle	0	644	RefreshTab	1	4
2	9	CloseTab	Normal	CloseTab	1	332	CloseTab	1	4
2	10	NewTab	Normal	NewTab	1	182	NewTab	1	4
2	11	OpenGoogle	Normal	OpenGoogle	1	333	OpenGoogle	1	4
2	12	CloseTab	Normal	CloseTab	1	127	CloseTab	1	4
2	13	CloseTab	Normal	CloseTab	1	131	CloseTab	1	4
3	1	OpenChrome	Normal	OpenGoogle	0	66	OpenGoogle	0	4
3	3	NewTab	Normal	NewTab	1	94	OpenChrome	0	4
3	5	OpenCNN	Normal	OpenCNN	1	521	OpenCNN	1	4
3	6	ZoomIn	Normal	N/A	0	32	ZoomIn	1	4
3	7	ZoomOut	Normal	ZoomOut	1	5	ZoomOut	1	7
3	8	RefreshTab	Normal	RefreshTab	1	80	RefreshTab	1	4
3	9	RefreshTab	Invertido	OpenGoogle	0	188	RefreshTab	1	4
3	10	CloseTab	Normal	CloseTab	1	394	CloseTab	1	5
3	11	NewTab	Normal	NewTab	1	118	NewTab	1	4
3	12	OpenGoogle	Normal	OpenGoogle	1	103	OpenGoogle	1	4
3	13	CloseTab	Normal	CloseTab	1	258	CloseTab	1	4
3	14	CloseTab	Normal	CloseTab	1	109	CloseTab	1	4
4	1	OpenChrome	Normal	OpenChrome	1	23	OpenChrome	1	6
4	2	NewTab	Normal	NewTab	1	62	NewTab	1	4
4	3	OpenCNN	Normal	OpenCNN	1	193	OpenCNN	1	4
4	4	ZoomIn	Normal	ZoomIn	1	8	ZoomIn	1	7
4	5	ZoomOut	Normal	ZoomOut	1	16	ZoomIn	0	8
4	8	RefreshTab	Normal	RefreshTab	1	387	RefreshTab	1	4
4	9	RefreshTab	Invertido	OpenGoogle	0	393	RefreshTab	1	4
4	10	CloseTab	Normal	CloseTab	1	116	CloseTab	1	4
4	11	NewTab	Normal	NewTab	1	93	NewTab	1	4

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
4	12	OpenGoogle	Normal	OpenGoogle	1	406	RefreshTab	0	4
4	16	CloseTab	Normal	CloseTab	1	106	CloseTab	1	4
4	17	CloseTab	Normal	CloseTab	1	377	CloseTab	1	4
5	1	OpenChrome	Normal	OpenChrome	1	34	OpenChrome	1	5
5	2	NewTab	Normal	NewTab	1	32	NewTab	1	4
5	3	OpenCNN	Normal	OpenCNN	1	585	OpenCNN	1	4
5	4	ZoomIn	Normal	N/A	0	7	ZoomIn	1	15
5	5	ZoomOut	Normal	N/A	0	98	ZoomOut	1	4
5	6	RefreshTab	Normal	RefreshTab	1	182	N/A	0	4
5	8	RefreshTab	Invertido	N/A	0	262	RefreshTab	1	4
5	9	CloseTab	Normal	CloseTab	1	63	CloseTab	1	4
5	10	NewTab	Normal	NewTab	1	44	NewTab	1	4
5	11	OpenGoogle	Normal	OpenGoogle	1	268	OpenGoogle	1	4
5	12	CloseTab	Normal	CloseTab	1	252	CloseTab	1	4
5	13	CloseTab	Normal	CloseTab	1	169	CloseTab	1	4
6	1	OpenChrome	Normal	OpenChrome	1	51	OpenChrome	1	3
6	2	NewTab	Normal	NewTab	1	98	NewTab	1	4
6	3	OpenCNN	Normal	OpenCNN	1	340	N/A	0	4
6	5	ZoomIn	Normal	N/A	0	2	ZoomIn	1	5
6	6	ZoomOut	Normal	ZoomOut	1	56	ZoomOut	1	4
6	7	RefreshTab	Normal	RefreshTab	1	404	RefreshTab	1	4
6	8	RefreshTab	Invertido	N/A	0	87	RefreshTab	1	5
6	9	CloseTab	Normal	CloseTab	1	632	CloseTab	1	4
6	10	NewTab	Normal	NewTab	1	120	NewTab	1	4
6	11	OpenGoogle	Normal	OpenGoogle	1	279	OpenGoogle	1	4
6	12	CloseTab	Normal	CloseTab	1	123	CloseTab	1	4
6	13	CloseTab	Normal	CloseTab	1	102	CloseTab	1	4

B.7. Experimento 3: Razer Hydra - Grandes

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	OpenChrome	1	1553	OpenChrome	1	15
1	2	MaximizeWinc	Normal	N/A	0	741	MaximizeWinc	1	4
1	3	RestoreWindo	Normal	RestoreWindo	1	733	RestoreWindo	1	4
1	4	OpenMusic	Normal	OpenMusic	1	2227	OpenMusic	1	4
1	5	OpenNetflix	Normal	OpenNetflix	1	3342	OpenNetflix	1	4
1	6	OpenYouTube	Normal	OpenYouTube	1	6953	OpenYouTube	1	4
1	7	OpenWeather	Normal	OpenWeather	1	6725	OpenWeather	1	4
1	8	ScrollDown	Normal	ScrollDown	1	200	ScrollDown	1	4
1	9	ScrollUp	Normal	ScrollUp	1	1164	ScrollUp	1	4
1	10	CloseTab	Normal	CloseTab	1	2224	CloseTab	1	4
2	1	OpenChrome	Normal	OpenChrome	1	15478	OpenChrome	1	68
2	2	MaximizeWinc	Normal	N/A	0	1676	MaximizeWinc	1	4
2	3	RestoreWindo	Normal	N/A	0	1538	RestoreWindo	1	4
2	4	OpenMusic	Normal	OpenMusic	1	24957	OpenMusic	1	4
2	5	OpenNetflix	Normal	OpenNetflix	1	13235	OpenNetflix	1	4
2	6	OpenYouTube	Normal	OpenYouTube	1	6400	OpenYouTube	1	4
2	7	OpenWeather	Normal	ScrollDown	0	11934	OpenWeather	1	4
2	8	ScrollDown	Normal	ScrollDown	1	2057	ScrollDown	1	4
2	9	ScrollUp	Normal	ScrollUp	1	1111	ScrollUp	1	4
2	10	CloseTab	Normal	CloseTab	1	15711	CloseTab	1	4

B.8. Experimento 3: Razer Hydra - Pequeños

				Damerau-Levenshtein			Deep Learning		
Sujeto	**Secuencia**	**Gesto**	**Orientación**	**Predicción**	**Correcto**	**Tiempo (ms)**	**Predicción**	**Correcto**	**Tiempo (ms)**
1	1	OpenChrome	Normal	OpenChrome	1	1303	N/A	0	10
1	3	MaximizeWinc	Normal	N/A	0	344	MaximizeWinc	1	4
1	4	RestoreWindo	Normal	N/A	0	359	RestoreWindo	1	4
1	5	OpenMusic	Normal	OpenMusic	1	394	OpenMusic	1	4
1	6	OpenNetflix	Normal	OpenNetflix	1	1200	OpenNetflix	1	4
1	7	OpenYouTube	Normal	OpenYouTube	1	1196	OpenYouTube	1	4
1	8	OpenWeather	Normal	ScrollDown	0	1263	OpenWeather	1	4
1	9	ScrollDown	Normal	ScrollDown	1	82	ScrollDown	1	4
1	10	ScrollUp	Normal	ScrollUp	1	96	ScrollUp	1	4
1	11	CloseTab	Normal	CloseTab	1	392	CloseTab	1	4
2	1	OpenChrome	Normal	OpenNetflix	0	4592	OpenChrome	1	61
2	2	MaximizeWinc	Normal	N/A	0	702	MaximizeWinc	1	4
2	3	RestoreWindo	Normal	N/A	0	157	ZoomOut	0	4
2	5	OpenMusic	Normal	OpenMusic	1	1677	OpenMusic	1	4
2	6	OpenNetflix	Normal	ScrollDown	0	434	OpenNetflix	1	4
2	7	OpenYouTube	Normal	ScrollUp	0	429	OpenYouTube	1	4
2	8	OpenWeather	Normal	ScrollDown	0	1113	OpenWeather	1	4
2	9	ScrollDown	Normal	ScrollDown	1	406	ScrollDown	1	4
2	10	ScrollUp	Normal	ScrolUp	0	58	ScrollUp	1	4
2	11	CloseTab	Normal	CloseTab	1	2338	OpenYouTube	0	4

Referencias Bibliográficas

Authors, T. T. (2018). Train your first neural network: basic classification. Recuperado desde https://github.com/tensorflow/docs/blob/master/site/en/tutorials/keras/basic_classification.ipynb

Bard, G. V. (2007, enero). Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric. *Proceedings of the Fifth Australasian Symposium on ACSW Frontiers*. Recuperado desde http://crpit.com/confpapers/CRPITV68Bard.pdf

Boger, Y. & Pavlik, R. A. (2015, mayo). An Introduction to OSVR. Recuperado desde https://osvr.github.io/whitepapers/introduction_to_osvr/

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... Zieba, K. (2016). End to End Learning for Self-Driving Cars. *CoRR*, *abs/1604.07316*. arXiv: 1604.07316. Recuperado desde http://arxiv.org/abs/1604.07316

Brownlee, J. (2016a, 20 de junio). Dropout Regularization in Deep Learning Models With Keras. Recuperado desde https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Brownlee, J. (2016b, 25 de marzo). Linear Regression for Machine Learning. Recuperado desde https://machinelearningmastery.com/linear-regression-for-machine-learning/

Castle, A. (2011, 12 de septiembre). Razer Hydra Review. MaximumPC. Recuperado desde http://www.maximumpc.com/article/reviews/razer_hydra_review

Côté Allard, U., Fall, C. L., Campeau-Lecours, A., Gosselin, C., Laviolette, F. & Gosselin, B. (2017). Transfer Learning for sEMG Hand Gesture Recognition Using Convolutional Neural Networks. En *2011 IEEE International Conference On Systems, Man and Cybernetics*. doi:10.1109/SMC.2017.8122854

Dabbura, I. (2018, 17 de septiembre). K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks. Recuperado desde https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

Dahl, G. E., Yu, D., Deng, L. & Acero, A. (2011). Large vocabulary continuous speech recognition with context-dependent DBN-HMMS. En *2011*

IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 4688-4691). doi:10.1109/ICASSP.2011.5947401

Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, *3*. doi:10.1017/atsip.2013.9

Deng, L. & Yu, D. (2014). *Deep Learning: Methods and Applications*. NOW Publishers. Recuperado desde https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/

Deviant, S. (2011). *The Practically Cheating Statistics Handbook – 3rd Edition*. Andale Publishing LLC. Recuperado desde https://books.google.nl/books?id=PZ2JAwAAQBAJ

Géron, A. (2018). *Neural Networks and Deep Learning*. O'Reilly. Recuperado desde https://books.google.nl/books?id=5pm6tQEACAAJ

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. The MIT Press.

Google. (2017). Soli. Recuperado desde https://atap.google.com/soli/

Hardware.Info. (2011). Razer Hydra + Portal 2. Hardware.Info. Recuperado desde http://us.hardware.info/productinfo/131486/razer-hydra-+-portal-2

Hinton, G. & Salakhutdinov, R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504-507.

HTC. (2017). Valve Provides Details on Upcoming Knuckles Vive Controllers. Recuperado desde https://www.vrfocus.com/2017/06/valve-provides-details-on-upcoming-knuckles-vive-controllers/

Ivezić, Ž., Connolly, A., Vanderplas, J. & Gray, A. (2014). *Statistics, Data Mining and Machine Learning in Astronomy*. Princeton, NJ: Princeton University Press.

Josh. (2012, 26 de marzo). Why are textures always square powers of two? What if they aren't? Recuperado desde https://gamedev.stackexchange.com/a/26189

Karpathy, A. (2018). CS231n Convolutional Neural Networks for Visual Recognition. Recuperado desde http://cs231n.github.io/convolutional-networks/

Ketabdar, H. & Bourlard, H. (2010). Enhanced Phone Posteriors for Improving Speech Recognition Systems. *IEEE Transactions on Audio, Speech, and Language Processing*, *18*(6), 1094-1106. doi:10.1109/TASL.2009.2023162

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2017). ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM*, *60*(6), 84-90. doi:10.1145/3065386

Lee, K.-F., Hon, H.-W., Hwang, M.-Y., Mahajan, S. & Reddy, R. (1989). The SPHINX speech recognition system. En *1989 International Conference on Acoustics, Speech, and Signal Processing (ICASSP '89)* (Vol. 1, pp. 445-448).

Li, B., Sainath, T., Narayanan, A., Caroselli, J., Bacchiani, M., Misra, A., ... Shannon, M. (2017). Acoustic Modeling for Google Home. Recuperado desde http://www.cs.cmu.edu/~chanwook/MyPapers/b_li_interspeech_2017.pdf

LLC, G. (2017). Classifying Handwritten Digits with Neural Networks. Recuperado desde https://colab.research.google.com/notebooks/mlcc/multi-class_classification_of_handwritten_digits.ipynb

Majorek, K. A., Steczkiewicz, K., Muszewska, A., Nowotny, M., Ginalski, K. & Bujnicki, J. M. (2014, abril). The RNase H-like superfamily: new members, comparative structural analysis and evolutionary classification. *Nucleic Acids Research*, *42*(7), 4160-79. Recuperado desde http : / / nar . oxfordjournals.org/content/42/7/4160.long

Mata-Montero, E. & Odio-Vivi, A. (2015). LaGeR Workbench: En J. M. García-Chamizo, G. Fortino & S. F. Ochoa (Eds.), *Ubiquitous Computing and Ambient Intelligence. Sensing, Processing, and Using Environmental Information: 9th International Conference, UCAmI 2015, Puerto Varas, Chile, December 1-4, 2015, Proceedings* (pp. 334-345). Cham: Springer International Publishing. doi:10.1007/978-3-319-26401-1_31

Matthews, H. (2013, 29 de mayo). Razer Hydra teardown Part 2 [h's blog]. Recuperado desde http://howiem.com/wordpress/index.php/2013/05/29/razer-hydra-teardown-part-2/

McCarthy, J., Minsky, M. L., Rochester, N. & Shannon, C. E. (1955). A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE. http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html. Recuperado desde http : / / www - formal . stanford . edu / jmc / history / dartmouth / dartmouth.html

McIntosh, J., Marzo, A. & Fraser, M. (2017). SensIR: Detecting Hand Gestures with a Wearable Bracelet using Infrared Transmission and Reflection. En *UIST*.

Mitchell, T. M. (1997). *Machine Learning* (1.ª ed.). New York, NY, USA: McGraw-Hill, Inc.

Mohamed, A.-r., Yu, D. & Deng, L. (2010). Investigation of Full-Sequence Training of Deep Belief Networks for Speech Recognition. En *Interspeech 2010*. International Speech Communication Association. Recuperado desde https://www.microsoft.com/en-us/research/publication/investigation-of-full-sequence-training-of-deep-belief-networks-for-speech-recognition/

Morgan, N. (2012). Deep and Wide: Multiple Layers in Automatic Speech Recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(1), 7-13. doi:10.1109/TASL.2011.2116010

Neverova, N., Wolf, C., Taylor, G. W. & Nebout, F. (2015). Multi-scale Deep Learning for Gesture Detection and Localization. En L. Agapito, M. M. Bronstein & C. Rother (Eds.), *Computer Vision - ECCV 2014 Workshops: Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part I* (pp. 474-490). Cham: Springer International Publishing. doi:10.1007/978-3-319-16178-5_33

Odio-Vivi, A. (2015). *LaGeR: Lenguaje para descripción de gestos bidimensionales y tridimensionales* (Tesis doctoral, Instituto Tecnológico de Costa Rica). Recuperado desde http://hdl.handle.net/2238/6700

Parvat, A., Chavan, J., Kadam, S., Dev, S. & Pathak, V. (2017). A survey of deep-learning frameworks. En *2017 International Conference on Inventive Systems and Control (ICISC)* (pp. 1-7). doi:10.1109/ICISC.2017.8068684

Pinto, J., Garimella, S., Magimai-Doss, M., Hermansky, H. & Bourlard, H. (2011). Analysis of MLP-Based Hierarchical Phoneme Posterior Probability Estimator. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(2), 225-241. doi:10.1109/TASL.2010.2045943

Pomerleau, D. (1989). ALVINN: An Autonomous Land Vehicle In a Neural Network. En *Advances in Neural Information Processing Systems 1*. Morgan Kaufmann.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv: 1609.04747 [cs.LG]

Russell, S. & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd). Upper Saddle River, NJ, USA: Prentice Hall Press.

Sainath, T. N., Kingsbury, B. & Ramabhadran, B. (2012). Improving training time of deep belief networks through hybrid pre-training and larger batch sizes. En *Proc. NIPS Workshop Log-linear Models*.

Schuck, P. (s.f.). Monte-Carlo Simulations. Recuperado desde http://www.analyticalultracentrifugation.com/sedfit_help_monte_carlo_simulations.htm

Sharma, S. (2017, 6 de septiembre). Activation Functions: Neural Networks. Recuperado desde https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484-503. Recuperado desde http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

Storyblocks. (2018, 15 de noviembre). Cartoon Stick Figure Gestures. Recuperado desde https://www.storyblocks.com/stock-image/cartoon-stick-figure-gestures-bp9odvhm_-j6gobsbe

Taylor II, R. M., Yang, R., Weber, H. & Hudson, T. (2014). Virtual Reality Peripheral Network. Recuperado desde http://www.cs.unc.edu/Research/vrpn/

Waibel, A. H., Hanazawa, T., Hinton, G. E., Shikano, K. & Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoustics, Speech, and Signal Processing*, *37*(3), 328-339. doi:10.1109/29.21701

WebAntics. (2018). WACOM INTUOS PEN AND TOUCH GRAPHICS TABLET - MEDIUM (+/-A5). Recuperado desde https://www.webantics.com/wacom-intuos-pen-and-touch-graphics-tablet-medium-a5

Wikipedia. (s.f.-a). *Análisis de componentes principales*. En *Wikipedia*. Recuperado desde https://es.wikipedia.org/wiki/An%5C%C3%5C%A1lisis_de_componentes_principales

Wikipedia. (s.f.-b). *Rhombicuboctahedron*. En *Wikipedia*. Recuperado desde https://en.wikipedia.org/wiki/Rhombicuboctahedron

Wikipedia. (s.f.-c). *Spherical coordinate system*. En *Wikipedia*. Recuperado desde http://en.wikipedia.org/wiki/Spherical_coordinate_system

Wikipedia. (s.f.-d). *Support vector macine*. En *Wikipedia*. Recuperado desde https://en.wikipedia.org/wiki/Support_vector_machine

Williamson, J. (s.f.). Hidden Markov Model gesture recognition. Recuperado desde http://www.dcs.gla.ac.uk/~jhw/gesturehmm.html

WiredEarp. (2011, 28 de noviembre). Razer Hydra teardown. [Meant to be Seen]. Recuperado desde http://www.mtbs3d.com/phpBB/viewtopic.php?f=120%5C&t=14036#p67251

xinreality. (2018). Oculus Touch. Recuperado desde https://xinreality.com/wiki/Oculus_Touch

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., ... Zweig, G. (2016). Achieving Human Parity in Conversational Speech Recognition. *CoRR, abs/1610.05256*. arXiv: 1610.05256. Recuperado desde http://arxiv.org/abs/1610.05256

Yang, J. & Xu, Y. (1994). *Hidden Markov Model for Gesture Recognition* (inf. téc. N.º CMU-RI-TR-94-10). Robotics Institute , Carnegie Mellon University. Pittsburgh, PA.

Zhang, C. (2018, octubre). How to use Keras sparse_categorical_crossentropy. Recuperado desde https://www.dlology.com/blog/how-to-use-keras-sparse_categorical_crossentropy/