Universidad de La Habana
Facultad de Matemática y Computación

Generación de Lengua de Señas Cubana

Autor:
Luis Enrique Dalmau Coopat

Tutores:

Leynier Gutiérrez González
Yudivian Almeida Cruz

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

Fecha

github.com/lukedalmau/computer-science-bachelor-thesis

Dedico este trabajo a mis padres, Liliana y Enrique, quienes con su amor y apoyo me han permitido lograr este sueño y a mis abuelos que ya no me acompañan en este plano terrenal , Benita, Luis, Raciel, Elsa y Kim, quienes estuvieran muy felices de verme cumplir este reto. Especial dedicación tambíen a toda la comunidad sordo-muda de Cuba por perseverar a pesar de todas las adversidades.

Agradecimientos

A mi familia, en especial a mis padres; mis abuelos; Vilma, Ada y Enrique; y a Moisés por siempre apoyarme y guiarme a pesar de lo duro que ha sido el camino. A mi prometida por acompañarme en estos años tan duros y ser uno de los pilares de mi vida; a sus padres y demás familia política, en especial a mi suegrita por ayudarme siempre que lo necesitaba. A mis amigos, Luiso, Miguel, Laurita, Wata, Yasmín, Jessy, Claudia, Marcos, Menor, Pollín, Betina, Karl Lewis, Raúl, Buti, Tigre por alumbrarme el camino cuando no era el mismo claro y por simpre lograr sacarme una risa y momentos de diversión. A mis demás compañeros y profesores por todos los conocimientos y experiencias que hacen de mi la persona que soy. A mis tutores, Yudivian y Leynier por abrirme las puertas a este mágnifico tema de investigación. A todas aquellas personas que de una forma u otra han aportado su granito de arena para que esta investigación sea hoy lo que es.

Opinión del tutor

Opiniones de los tutores

Resumen

Este trabajo sigue un enfoque nunca antes probado en el contexto de las investigaciones de la Lengua de Señas Cubana. Muchos países han avanzado en el aspecto de la producción de las lenguas de señas, probándose las redes convolucionales, las redes recurrentes, los transformadores y enfoques no autorregresivos, los cuales han brindado excelentes resultados en los respectivos países de dichas investigaciones. Se presenta una investigación acerca del uso de un modelo secuencia a secuencia basado en capas LSTM y capas deconvolucionales para la generación de avatares para la Lengua de Señas Cubana utilizando el corpus brindado por el CENDSOR.

Abstract

This work follows an untested approach in the context of Cuban Sign Language research. Many countries have some advance in the aspects of the production of sign languages, testing convolutional networks, recurrent networks, transformers and non-autoregressive approaches, which have provided excellent results in the respective countries of those investigations. An investigation about the use of a sequence-to-sequence model based on LSTM layers and deconvolutional layers for the generation of avatars for the Cuban Sign Language using the corpus provided by CENDSOR is now presented.

Índice general

Introducción **1**
0.1. Problemática 2
0.2. Objetivos del Trabajo 2
0.2.1. Objetivo general 2
0.2.2. Objetivos específicos 2
0.3. Propuesta de solución 3
0.4. Estructura del trabajo 3

1. Estado del Arte **4**
1.1. Lengua de Señas 4
1.2. Escritura de señas Sutton 4
1.2.1. Símbolos 4
1.2.2. Orientación 5
1.2.3. Forma de la mano 6
1.2.4. Movimiento de los dedos 8
1.2.5. Movimiento de las manos 9
1.2.6. Movimiento de los hombros, cabeza y ojos 9
1.2.7. Símbolos de Contacto 9
1.2.8. Ubicación 10
1.2.9. Expresión 11
1.2.10. Movimiento corporal 11
1.2.11. Prosodia 11
1.2.12. Puntuación 11
1.3. Traducción automática 11
1.3.1. Basada en métodos semánticos 11
1.3.2. Basada en métodos estadísticos 12
1.3.3. Basada en métodos de inteligencia artificial y aprendizaje de máquinas 13
1.4. Tecnología de avatares 14
1.5. Producción de Lengua de Señas 15

1.5.1. Redes Neuronales Convolucionales 15
1.5.2. Redes Recurrentes 16
1.6. Reconocimiento de la Lengua de Señas 16
1.7. Investigaciones y proyectos relacionados con la Lengua de Señas Cubana 17

2. Propuesta **19**
2.1. Alternativa Step by Step 19
2.1.1. Extraer los datos de la nube 20
2.1.2. Limpieza de los datos 20
2.1.3. Utilizando Embedding de palabra 20
2.1.4. Extrayendo la matriz de cada frase 21
2.1.5. Encontrando similitud de palabras 21
2.1.6. Arreglando errores ortográficos 21
2.1.7. Limitando a una sola seña por frase 21
2.1.8. Inicialización del modelo 22
2.1.9. Graficación y Animación 22

3. Detalles de Implementación y Experimentos **23**
3.1. Herramientas y tecnologías utilizadas 23
3.1.1. Lenguaje de programación Python 23
3.1.2. Numpy 24
3.1.3. Matplotlib 24
3.1.4. Tensorflow, Keras y Addons 24
3.1.5. Scipy 25
3.1.6. Google Colaboratory 25
3.1.7. Gensim 25
3.1.8. Word2Vec 25
3.1.9. difflib 26
3.1.10. Boto3 27
3.1.11. pathlib 27
3.2. Implementación de un prototipo 27
3.2.1. Extracción de los Datos 27
3.2.2. Limpieza de los datos 29
3.2.3. Cargando el embedding de palabra 31
3.2.4. Encontrando similitud de palabras 31
3.2.5. Extrayendo la matriz de cada frase 32
3.2.6. Creando los conjuntos 33
3.2.7. Métodos para la animación 33
3.3. Entrenamiento 37
3.3.1. Experimentos 40

3.3.2. Discusión 42

Conclusiones **43**

Recomendaciones **44**

Bibliografía **45**

Índice de figuras

1.1. Ejemplo de una secuencia de glifos 5
1.2. Formas de las manos y su representación en SignWriting 7
1.3. Ejemplo de movimiento de los dedos en SignWriting 8
1.4. Ejemplo de símbolos de contacto 10
1.5. Métodos de Traducción Semánticos 13
1.6. Software MediaPipe Holistic de Google 17
1.7. Aplicación Android para la LSC [Almarales-Wilson 2021] 18

2.1. Todos los puntos obtenidos sin obviar el tren inferior 20

3.1. Grado de aprendizaje de la semántica de Word2Vec en español 26
3.2. Resumen de las capas y parametros del modelo 40
3.3. Resultado de las métricas de los últimos epochs del modelo durante el entrenamiento 41
3.4. Comparativa de la frase "abandonar" . Fila superior los valores predichos. Fila inferior los valores reales 41
3.5. Comparativa de la frase "por la mañana" . Fila superior los valores predichos. Fila inferior los valores reales 42

Ejemplos de código

3.1. Instanciar cliente s3 27
3.2. Descargar usando el cliente s3 28
3.3. Cargar los json y limpiar las llaves 29
3.4. Descargar datos del modelo KeyedVectors de su sitio web 31
3.5. Cargar modelo de KeyedVectors 31
3.6. Método para encontrar palabras similares en un vocabulario 31
3.7. Obtener el vector dada una palabra 32
3.8. Crear los conjuntos X e y 33
3.9. Instanciar datos necesarios para animar 33
3.10. Extraer las partes en 3d 34
3.11. Método base para animar 34
3.12. Graficar la animación 37
3.13. Diseño del modelo 37
3.14. Graficar la animación 38

Introducción

La población sordomuda es una minoría no despreciable, en cuanto a desempeño social, a la cual se debe prestar principal atención. Muchas de estas personas sordomudas esconden un verdadero potencial humano el cual no pueden desarrollar en su máxima capacidad debido a los impedimentos sociales en que incurren, a causa del problema comunicativo mayoritariamente. El autor quiere con este trabajo lograr disminuir esa brecha comunicativa lo máximo posible para alcanzar un mundo mejor, en el que ser sordomudo no represente una limitación en el proceso de la comunicación social, ni en su desarrollo individual.

Alrededor de 72 millones, de las 8 mil millones de personas que vivimos en este mundo, son personas sordas que utilizan lenguas de señas para comunicarse, según la Federación Mundial de Sordos (WFD por sus siglas en inglés)[Kozik 2019].

Aproximadamente más del 80 % de esos 72 millones viven en países en desarrollo, donde las autoridades locales no promueven ni apoyan políticas o acciones para suplir sus necesidades o incluirlos en la sociedad.

En Cuba, la Asociación Nacional de Sordos de Cuba (ANSOC) cuenta con alrededor de 25 mil asociados y aproximadamente 400 intérpretes registrados, de un total de 11 millones aproximadamente de habitantes, según especifica Yoel Moya, Subdirector de Investigación del Centro Nacional de Superación y Desarrollo del Sordo (CENDSOR). Sería asombroso imaginar la cantidad de tiempo, dinero y preparación que se ahorraría el país si se tuviera, en tiempo real, una traducción de texto a lengua de señas(análogamente de audio a lengua de señas se puede realizar con un paso extra de llevar de audio a texto).

Como es una proporción abrumadora, las cifras del número de personas sordomudas representan menos del 1 % del total de personas, tanto a nivel mundial, como nacional. Por tanto sería prácticamente imposible para una persona sordomuda llegar a entenderse con personas en una actividad del día a día.

Una solución es lograr traducir del lenguaje natural a la lengua de señas. La solución que se propone es generar lengua de señas en forma de video con un avatar para que las personas sordomudas que sepan lengua de señas puedan entenderse con la otra parte de la población que no habla dicha lengua. De ser lo suficientemente preciso puede servir además a modo de tutorial para aquellas personas que no sepan

lengua de señas y puedan aprender mediante la repetición de los gestos del avatar.

En la actualidad este problema de la comunicación con personas sordomudas es abarcado por disímiles campos de investigación, como la interpretación automática de las lenguas de señas, la generación de avatares señantes, entre otros. Si bien este proceso se puede entender como una traducción interlingual, las características específicas de las lenguas de señas hacen que su traducción e interpretación constituyan un proceso no tan sencillo que involucra componentes visuales. Es por dichos componentes que intervienen además varios campos de investigación como el procesamiento de lenguaje natural y la visión por computadora para asistir al entendimiento y procesamiento correcto de las lenguas de señas [Gutiérrez-González 2021].

0.1. Problemática

En lo que se refiere a la generación de avatares para la lengua de señas existen avances significativos a nivel mundial. La Lengua de Señas Cubana es diferente de otras lenguas de señas utilizadas en el mundo y de cualquier idioma hablado conocido. Nuestra lengua se utiliza en muchos países con algunas modificaciones en cada país pero somos capaces de comunicarnos entre todos. Por el contrario la lengua de señas cubana es única y por tanto Cuba es el único país responsable, en realizar estudios, investigaciones y herramientas para impulsar su aceptación.

Aunque en los últimos años en el país se han impulsado las tecnologías de la información y las telecomunicaciones, no se ha utilizado estos avances en pos de la generación de avatares para lograr una mayor inclusión de la sociedad hipoacústica . Es debido a ello que se hace importante fomentar estudios e implementar plataformas que apoyen e inciten a una mejor inserción en la sociedad de las personas que dependan de la Lengua de Señas Cubana para comunicarse.

0.2. Objetivos del Trabajo

0.2.1. Objetivo general

- Proponer un modelo para la generación automática de avatares gráficos para la traducción automática de la lengua de señas cubana

0.2.2. Objetivos específicos

- Estudiar el estado del arte en la generación de avatares para las lenguas de señas
- Identificar el conjunto de símbolos y acciones que se conocen por investigaciones previas de la lengua de señas cubana

- Proponer un modelo para la generación de avatares gráficos en la lengua de señas
- Proponer un modelo basado en aprendizaje de máquina que permita dotar a los avatares de rasgos que permitan la identificación del hablante en relación al avatar
- Realizar un conjunto de experimentos que permitan probar la viabilidad de su propuesta

0.3. Propuesta de solución

Como método de solución se propone un sistema por partes, utilizando redes neuronales recurrentes, en específico un modelo basado en Memorias Grandes de Corto Plazo (LSTM por sus siglas en inglés) y además, de utilizar capas convolucionales transpuestas (1DConvT) para el incremento de dimensionalidad. Se utiliza el corpus disponible por trabajos anteriores en el tema, se le hace una limpieza a las palabras y se equiparan los frames de la secuencia de puntos del corpus. Además, se utiliza un embedding de palabra entrenado en el idioma español para aportarle generalización al modelo para luego usar dicho vector como entrada del modelo de las capas anteriormente mencionadas. Luego se procesa el resultado obtenido en forma de matriz para transformarlo a la dimensión que permita efectuar el dibujo.

0.4. Estructura del trabajo

Este trabajo se divide en 6 partes. El capítulo introductorio plantea la problemática y sus consecuencias en el contexto social correspondiente, describiendo posteriormente, los objetivos y un breve resumen de la propuesta del trabajo, terminando con la explicación de su estructura. El capítulo 1 aborda un estudio realizado sobre el estado del arte del campo, mostrando varios conceptos e ideas que se aplican en los capítulos siguientes. El capítulo 2 presenta el diseño del sistema con el modelo de capas recurrentes y convolucionales propuesto para la generación automática de avatares para la lengua de señas cubana. En el capítulo 3 se explica la implementación de un prototipo del modelo primario propuesto, los experimentos para validar la propuesta y los resultados obtenidos. Seguidamente se exponen las Conclusiones del trabajo y un grupo de Recomendaciones para investigaciones y desarrollos futuros. Culmina con la lista de las referencias bibliográficas utilizadas para su elaboración.

Capítulo 1

Estado del Arte

1.1. Lengua de Señas

La lengua de señas es el principal medio de comunicación para los sordos. Es una lengua que utiliza signos en lugar de sonidos para comunicarse. Los signos se producen utilizando las manos, la cara y el cuerpo. Los mismos se usan en todo el mundo y existen más de 300 variantes. Las lenguas de señas son todas diferentes, no son inteligibles entre sí, no son universales y tienen su propia gramática y sintaxis las cuales difieren de las del lenguaje hablado.

1.2. Escritura de señas Sutton

La escritura de señas Sutton (Sutton SignWriting), comúnmente conocida como escritura de señas (SignWriting), es un sistema de escritura de lenguas de señas. Es muy especifíco y visualmente icónico, tanto en las formas de los personajes, que son imágenes abstractas de las manos, la cara y el cuerpo, como en su disposición espacial en la página, que no sigue un orden secuencial como las letras que forman las palabras escritas. Fue desarrollado en 1974 por Valerie Sutton, una bailarina que, dos años antes, había desarrollado la escritura danzaria (DanceWriting). Algunas formas estandarizadas más nuevas se conocen como el Alfabeto internacional de escritura por señas (ISWA por sus siglas en inglés).

1.2.1. Símbolos

La cantidad de símbolos es extensa y, a menudo, brinda múltiples formas de escribir un solo signo. Así como tomó muchos siglos para que la ortografía de los diversos idiomas se estandarizara, la ortografía en SignWriting aún no está estandarizada para ninguna lengua de señas.

En SignWriting, se utiliza una combinación de símbolos icónicos para formas de manos, orientación, ubicaciones corporales, expresiones faciales, contactos y movimiento[Thiessen 2011][Everson y col. 2013] para representar palabras en lengua de señas.

Sutton originalmente diseñó el guión para que se escribiera horizontalmente (de izquierda a derecha), como en su idioma nativo, y desde el punto de vista del observador, pero luego lo cambió a vertical (de arriba a abajo) y desde el punto de vista de el firmante, para ajustarse a los deseos de los escritores sordos.

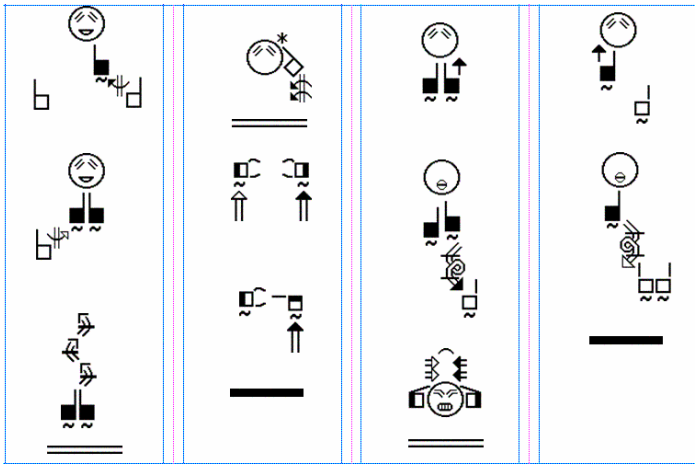

Figura 1.1: Ejemplo de una secuencia de glifos

Dado que SignWriting representa la formación física real de los signos en lugar de su significado, no se requiere ningún análisis fonético o semántico de un idioma para escribirlo. Una persona que ha aprendido el sistema puede ”sentir” un signo desconocido de la misma manera que una persona que habla inglés puede ”pronunciar” una palabra desconocida escrita en el alfabeto latino, sin siquiera saber qué significa el signo.

Las palabras pueden estar escritas desde el punto de vista del firmante o del espectador. Sin embargo, casi todas las publicaciones utilizan el punto de vista del firmante y asumen que la mano derecha es dominante.

1.2.2. Orientación

La orientación de la palma se indica rellenando el glifo de la forma de la mano. Un glifo de contorno hueco (blanco) indica que uno está mirando hacia la palma de la mano, un glifo relleno (negro) indica que uno está mirando hacia el dorso de la mano y un sombreado dividido indica que uno está viendo la mano desde un lado. Aunque en realidad la muñeca puede girar a posiciones intermedias, en SignWriting

solo se representan las cuatro orientaciones de palma, dorso y ambos lados, ya que son suficientes para representar las lenguas de señas.

Si se usa un glifo continuo, entonces la mano se coloca en el plano vertical (pared o cara) frente al firmante, como ocurre al deletrear con los dedos. Una banda borrada a través del glifo a través de los nudillos muestra que la mano se encuentra en el plano horizontal, paralela al suelo. (Si se usa uno de los glifos básicos de la forma de la mano, como el cuadrado o el círculo simple, esta banda lo divide en dos; sin embargo, si hay líneas para los dedos que se extienden desde la base, entonces se separan de la base, pero la base en sí permanece intacta).

El diagrama de la izquierda muestra una mano BA (mano plana) en seis orientaciones. Para las tres orientaciones verticales en el lado izquierdo, la mano se sostiene frente al firmante, con los dedos apuntando hacia arriba. Los tres glifos se pueden girar, como las manecillas de un reloj, para mostrar los dedos apuntando en ángulo, hacia un lado o hacia abajo. Para las tres orientaciones horizontales en el lado derecho del diagrama, la mano se sostiene hacia afuera, con los dedos apuntando en dirección opuesta al firmante y presumiblemente hacia el espectador. También se pueden rotar para mostrar los dedos apuntando hacia un lado o hacia el firmante. Aunque se puede representar un número indefinido de orientaciones de esta manera, en la práctica solo se usan ocho para cada plano, es decir, solo se encuentran múltiplos de 45°.

1.2.3. Forma de la mano

Hay más de cien glifos para las formas de las manos, pero todos los que se usan en la lengua de señas americana(ASL) se basan en cinco elementos básicos:

- Un cuadrado representa un puño cerrado, con los nudillos de los dedos flexionados doblados 90° para que los dedos toquen la palma y el pulgar quede sobre los dedos. Sin adornos, este cuadrado representa la mano S del deletreo manual. Modificado como se describe a continuación, indica que al menos uno de los cuatro dedos toca la palma de la mano.

- Un círculo representa un "puño abierto", una mano donde el pulgar y los dedos están flexionados para tocarse en sus puntas. Sin adornos, esta es la mano O del deletreo manual. Modificado, indica que al menos un dedo toca el pulgar de esta manera.

- Un pentágono (triángulo encima de un rectángulo), como en la ilustración utilizada para la sección de Orientación anterior, representa una mano plana, donde todos los dedos están rectos y en contacto. Esto es similar a la mano B del deletreo manual, aunque sin que el pulgar cruce la palma.

- Una forma de 'C' representa una mano donde el pulgar y los dedos están curvados, pero no lo suficiente como para tocarlos. Esto se usa para la mano C del deletreo manual y se puede modificar para mostrar que los dedos están separados.
- Una forma en ángulo, como una L gorda, muestra que los cuatro dedos son planos (derechos y en contacto), pero doblados a 90° del plano de la palma. No aparece como una forma simple, sino que debe incluir una indicación de dónde está el pulgar, ya sea hacia un lado o tocando las puntas de los dedos.

Figura 1.2: Formas de las manos y su representación en SignWriting

Una línea a la mitad del cuadrado o pentágono muestra el pulgar en la palma de la mano. Estas son la E, B y (con los dedos separados) 4 manos de deletreo manual.

Estas formas básicas se modifican con líneas que sobresalen de sus caras y esquinas para representar dedos que no están colocados como se describe arriba. Las líneas

rectas representan dedos rectos (estos pueden estar en ángulo para indicar que no están alineados con la palma; si apuntan hacia o lejos del firmante, tienen forma de diamante en la punta); líneas curvas para dedos curvos (en forma de copa); líneas ganchudas para dedos ganchudos; líneas de ángulo recto, para dedos doblados en una sola articulación; y líneas cruzadas, para dedos cruzados, como se muestra en el gráfico de la derecha. El pentágono y C solo se modifican para mostrar que los dedos están separados en lugar de estar en contacto; el ángulo solo se modifica para mostrar si el pulgar toca las puntas de los dedos o sobresale hacia un lado. Aunque se pueden hacer algunas generalizaciones para las docenas de otros glifos, que se basan en el círculo y el cuadrado, los detalles son algo idiosincrásicos y cada uno debe memorizarse.

1.2.4. Movimiento de los dedos

Solo hay unos pocos símbolos para el movimiento de los dedos. Pueden duplicarse para mostrar que el movimiento se repite.

Una bala sólida representa flexionar la articulación media de un dedo o dedos, y una bala hueca representa enderezar un dedo flexionado. Es decir, una mano 'D' con una bala sólida significa que se convierte en una mano 'X', mientras que una mano 'X' con una bala hueca significa que se convierte en una mano 'D'. Si los dedos ya están flexionados, entonces una bala sólida muestra que aprietan. Por ejemplo, un cuadrado (puño cerrado, mano 'S') con viñetas sólidas dobles es el signo de 'leche' (icónicamente apretando una ubre).

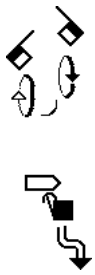

Figura 1.3: Ejemplo de movimiento de los dedos en SignWriting

Un cheurón que apunta hacia abajo representa la flexión de los nudillos, mientras que un cheurón que apunta hacia arriba ($\wedge$) muestra que los nudillos se enderezan. Es decir, una mano 'U' con un cheurón hacia abajo se convierte en una mano 'N', mientras que una mano 'N' con un cheurón hacia arriba se convierte en una mano 'U'.

Un zigzag como dos cheurones ($\wedge\wedge$) unidos significa que los dedos se flexionan repetidamente y sincronizados. Un zigzag de doble línea significa que los dedos se retuercen o aletean sin sincronización.

1.2.5. Movimiento de las manos

Cientos de flechas de varios tipos se utilizan para indicar el movimiento de las manos a través del espacio. La notación de movimiento se vuelve bastante compleja, y debido a que es más exacta de lo que necesita ser para cualquier lengua de señas, diferentes personas pueden optar por escribir el mismo signo de diferentes maneras.

Para el movimiento con la mano izquierda, la punta de flecha en forma de $\triangle$ es hueca (blanca); para el movimiento con la mano derecha, es sólido (negro). Cuando ambas manos se mueven como una sola, se usa una punta de flecha abierta (en forma de $\wedge$).

Al igual que con la orientación, las flechas de movimiento distinguen dos planos: el movimiento en el plano vertical (arriba y abajo) está representado por flechas con dos ejes, como en la parte inferior del diagrama de la izquierda, mientras que las flechas de un solo eje representan el movimiento paralelo al suelo (de ida y vuelta). Además, el movimiento en un plano diagonal utiliza flechas de doble tallo modificadas: una barra transversal en el tallo indica que el movimiento es hacia arriba o hacia abajo, y un punto sólido indica un movimiento de aproximación. El movimiento de ida y vuelta que también pasa por encima o por debajo de algo utiliza flechas modificadas de un solo tallo, con la parte de la flecha que representa el movimiento cercano más gruesa que el resto. Estos son icónicos, pero convencionalizados, por lo que deben aprenderse individualmente.

Los movimientos rectos son en una de las ocho direcciones para cualquier plano, como en las ocho direcciones principales de una brújula. Una flecha recta larga indica movimiento desde el codo, una flecha corta con una barra transversal detrás indica movimiento desde la muñeca y una flecha corta simple indica un movimiento pequeño. (Duplicados, en direcciones opuestas, estos pueden mostrar movimientos de cabeza desde la muñeca). Una flecha curva secundaria que cruza la flecha principal muestra que el brazo gira mientras se mueve. (Duplicadas, en direcciones opuestas, pueden mostrar el temblor de la mano). Las flechas pueden girar, curvarse, zigzaguear y hacer bucles.

1.2.6. Movimiento de los hombros, cabeza y ojos

Las flechas en la cara a la altura de los ojos muestran la dirección de la mirada.

1.2.7. Símbolos de Contacto

Seis glifos de contacto muestran el contacto de la mano con la ubicación de la señal. Es decir, un glifo de forma de mano ubicado al costado de la cara, junto con un glifo de contacto, indica que la mano toca el costado de la cara. La elección del glifo de contacto indica la forma del contacto:

- un asterisco ($*$ o $*$) por simplemente tocar el lugar;
- un signo más ($+$) para agarrar el lugar (generalmente la otra mano);
- un signo de hashtag o almohadilla ($\sharp$) para marcar el lugar;
- un círculo con un punto dentro ($\odot$) para cepillar el lugar y luego dejarlo;
- una espiral (o puede aproximarse con @) para frotar el lugar y no salir; si no hay flecha adicional, se entiende que está en círculos; y
- dos barras a cada lado de un símbolo de contacto ($|*|$) para indicar que el contacto ocurre entre elementos del lugar de contacto; generalmente entre los dedos, o dentro de una forma de mano circular. (Un contacto que no sea el asterisco básico rara vez se usa entre barras).

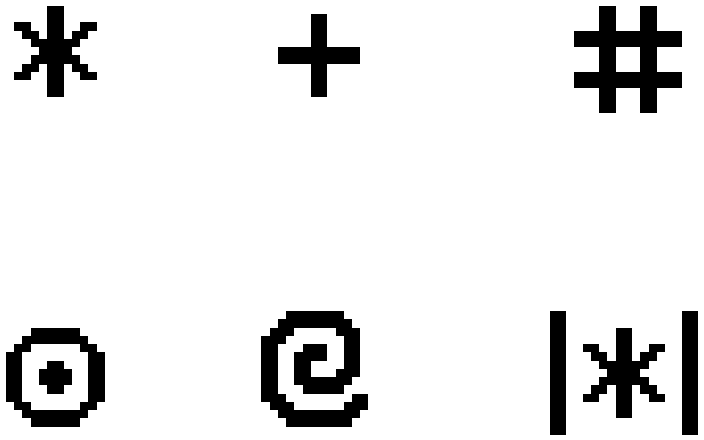

Figura 1.4: Ejemplo de símbolos de contacto

1.2.8. Ubicación

Si la mano que firma se encuentra en la otra mano, el símbolo de la misma es una de las formas de mano anteriores. En la práctica, solo ocurre un subconjunto de las formas de manos más simples.

Se utilizan símbolos adicionales para representar ubicaciones de signos en la cara o partes del cuerpo distintas de las manos. Un círculo muestra la cabeza.

1.2.9. Expresión

Hay símbolos para representar los movimientos faciales que se usan en varias lenguas de señas, incluidos los ojos, las cejas, los movimientos de la nariz, las mejillas, los movimientos de la boca y los cambios en la respiración. También se puede mostrar la dirección del movimiento de la cabeza y la mirada.

1.2.10. Movimiento corporal

Los hombros se muestran con una línea horizontal. Se pueden agregar flechas pequeñas para mostrar el movimiento del hombro y el torso. Se pueden agregar brazos e incluso piernas si es necesario.

1.2.11. Prosodia

También hay símbolos que indican la velocidad del movimiento, si el movimiento es simultáneo o alterno, y la puntuación.

1.2.12. Puntuación

Existen varios símbolos de puntuación que corresponden a comas, puntos, signos de interrogación y exclamación y otros símbolos de puntuación de otras escrituras. Estos se escriben entre signos, y las líneas no se rompen entre un signo y su siguiente símbolo de puntuación.

1.3. Traducción automática

La generación automática de avatares para las lenguas de señas como problema en si son un subconjunto de los problemas de traducción automática y de procesamiento de lenguajes naturales. Existen tres tipos de métodos esenciales en el ámbito de la traducción automática.

1.3.1. Basada en métodos semánticos

Los métodos basados en este acercamiento realizan análisis sintácticos y semánticos utilizando bases de conocimientos, ontologías y reglas definidas según el lenguaje que se utilice [Knight y Luk 1994]. Son presentadas tres ramas principales de esta familia:

- las basadas en traducciones directas

- las basadas en transferencia
- las basadas en Interlingua [Klueva 2007].

Los métodos de traducción automática directa consisten en reemplazar las palabras de un lenguaje a otro sin considerar aspectos sintácticos ni semánticos del lenguaje origen o destino, además de no utilizar información contextualizada [Okpor 2014].

Los basados en transferencia realizan transformaciones para obtener textos correctos del lenguaje destino aplicando reglas específicas a los lenguajes de origen y destino.

Generalmente, utilizan análisis sintácticos y semánticos en algún grado para realizar la traducción [Gehlot y col. 2015].

Los que son basados en Interlingua transforman la entrada de un lenguaje, el cual sería el de origen a una Interlingua. Este último no es más que un lenguaje artificial, diseñado para la traducción automática. Tomándose en cuenta ciertas distinciones necesarias para una traducción exitosa a un lenguaje de destino a pesar de no evidenciarse en el lenguaje de origen [Zhu y col. 2020].

La traducción por Interlingua puede ser tratada como un caso particular de los métodos de transferencia, solo que pasando por un lenguaje "común". Esto presenta desafíos, puesto que el lenguaje común debe ser capaz de captar las características de ambos lenguajes a traducir, sumándole también la posibilidad de captar características de terceros lenguajes. Esto es de gran utilidad cuando se quiere incoporar un nuevo lenguaje al flujo de trabajo ya que al poseer dos o más lenguajes conectados a una Interlingua, el trabajo necesario para agregar un nuevo lenguaje a la traducción recae solamente en realizar la correcta transformación del nuevo lenguaje a la Interlingua [Fig 1.5].

1.3.2. Basada en métodos estadísticos

Entre las categorías de estos métodos podemos encontrar las traducciones basadas en:

- palabra o traducción léxica, asignando una palabra a una o varias palabras del lenguaje objetivo en conjunto con cierta probabilidad de que sea la traducción correcta [Koehn 2009].
- frases. Similar a la basada en palabras, pero utiliza una secuencia de ellas como unidad de traducción. Se entrena a partir de cada par de oraciones de un corpus paralelo para así seleccionar la frase de manera exitosa [Koehn y col. 2003].

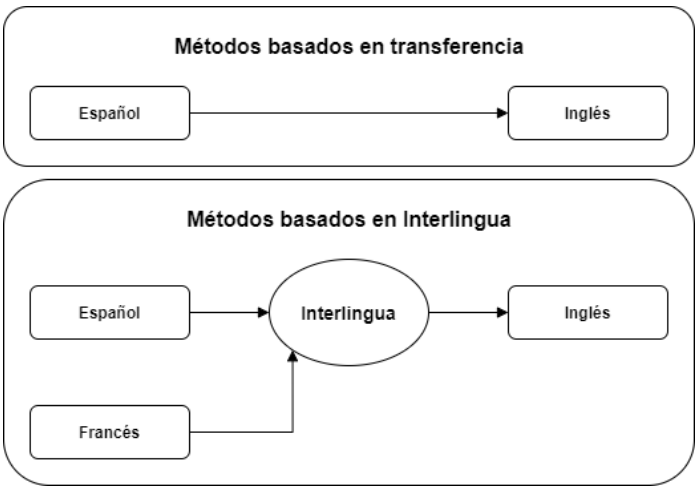

Figura 1.5: Métodos de Traducción Semánticos

- sintaxis. Comparte la misma naturaleza de la anterior pero utilizando frases sintácticamente corrrectas mediante árboles parciales de análisis sintácticos[Williams y col. 2016].
- jerarquía de frases. Una combinación de las dos anteriores [Chiang 2005].

1.3.3. Basada en métodos de inteligencia artificial y aprendizaje de máquinas

Dado que estos métodos son ampliamente aplicados en el procesamiento del lenguaje natural no es de extrañar que los sistemas actuales de traducción automática contengan modelos de redes neuronales artificiales [Koehn 2017].

Los basados en redes neuronales recurrentes convencionales (RNN por sus siglas en inglés) y las redes de gran memoria de corto plazo (LSTM por sus siglas en inglés), como se verá más adelante en este capítulo son muy usados en los sistemas de traducción automática. La gran virtud de este tipo de redes neuronales es la habilidad de capturar el contexto de las oraciones debido a sus estructuras de memoria [Sutskever y col. 2014].

A pesar de la gran relevancia que han tomado estos métodos basados en redes neuronales [Garje y Kharate 2013], los relacionados con las traducciones automáticas basadas en métodos estadísticos y/o semánticos son aún los más abundantes.

1.4. Tecnología de avatares

La traducción automática de lenguas de señas, aplicando la tecnología de generación de avatares, ha sido empleada en proyectos como ViSiCAST y eSIGN, apoyados por la Unión Europea [2021] [2021].

Se utilizan para traducir de las lenguas habladas a las lenguas de señas, el problema que se aborda en este trabajo.

Las animaciones para las señas individuales son generadas basándose en las señas no manuales de datos complementarios y en señas individuales.

Los procesos para generar los avatares incluyen una serie de pasos bastante complejos.

Los aspectos referentes a la velocidad, el ritmo y el tamaño deben ser tomados en consideración utilizando reglas para controlarlos, ser configurados por un humano o como resultado de un modelo de aprendizaje de máquina [Nguyen y col. 2021].

Otro de los usos que tienen estos métodos es lograr el anonimato en los videos y cabe remarcar que son nuevos comparándolos relativamente con otros métodos de traducción automática de lenguas de señas [Kang y Gratch 2010] [Saragih y col. 2011].

Uno de los más grandes desafíos es que los avatares luzcan como humanos. En diversos estudios se comparan los gestos de los avatares con los gestos de los humanos incluyendo algunos trabajos que han obtenido buenos resultados en Reino Unido en cuanto a generar dichos avatares con rasgos humanos[Stoll y col. 2018] y usando el mismo corpus del trabajo anterior en Corea se han logrado avances en un enfoque no autorregresivo para lograr resultados incluso mejores[Hwang y col. 2021].

Entre las tecnologías más utilizadas para la generación de avatares por animación se encuentran:

- Unity
- Unreal 4
- Blender
- Maya3D
- DirectX

Se han desarrollado lenguajes de marcado para automatizar la generación de avatares de una forma dinámica y flexible [Latoschik y col. 2017] [Aneja y col. 2019].

El estado del arte en la generación de avatares no está completamente automatizado, la mayoría de las partes del proceso de generación incluyen intervención humana para lograr buenos resultados.

Gran parte de las investigaciones miden la calidad de las animaciones de avatares de forma perceptual y de estudios comprensivos con participantes sordos-mudos,

incluyendo investigaciones metodológicas y recursos compartidos [Huenerfauth 2006] [Kacorri y col. 2017].

1.5. Producción de Lengua de Señas

La producción de lengua de señas(o SLP por sus siglas en inglés) ha tenido una adecuada trayectoria en cuanto a proyectos basados en avatares se refiere, los cuales generan señas parecidas a las realizadas por los humanos, pero se basan en búsqueda de frases y diccionarios de movimientos predefinidos [McDonald y col. 2015], o requieren captura de movimiento costosa o frases pregrabadas [Lu y Huenerfauth 2011]. Con los avances recientes en el aprendizaje profundo, Stoll et al. Alabama. [Stoll y col. 2018] proponen el primer modelo SLP para traducir texto en glosario de palabras y mapearlas a las poses de signos correspondientes. Zelinka y Kanis [Zelinka y Kanis 2020] proponen el primer modelo SLP de extremo a extremo de longitud fija y también proponen un método de descenso de gradiente para el refinamiento de los esqueletos generados. Recientemente, [Jiang y col. 2021] proponen un transformador(transformer en inglés), basado en estos modelos de Zelinka, el cual refina el esqueleto levantado en 3D con un enfoque de entrenamiento parecido a BERT . [Saunders y col. 2020] proponen Transformador Progresivo (PT), que utiliza un esquema de contracodificación para aprender directamente el mapeo entre lenguaje hablado y secuencia de poses de señas.

1.5.1. Redes Neuronales Convolucionales

Uno de los bloques básicos de los métodos utilizados en el razonamiento visual son las capas convolucionales. Usando estas capas, las redes neuronales convolucionales(CNN por sus siglas en íngles) modelan de manera efectiva la estructura de las imágenes[LeCun y col. 1998]. En cuanto a la producción de lengua de señas, las convolucionales son la mismísima base de los modelos propuestos. Sin embargo, el desarrollo de las CNN enfrenta inconvenientes con respecto al campo receptivo limitado, introducido como el tamaño del núcleo. Como solución, ideas como usar muchas más capas convolucionales, aumentar el tamaño del núcleo, entre otras. Otro inconveniente importante de las CNNs es la falta de aprendizaje temporal correspondiente a secuencias de imágenes. Lo cual es propiamente resuelto utilizando a las convolucionales de tres dimensiones(3DCNN por sus siglas) como alternativa ante el modelado de tipo recurrente. Variedad de modelos han sido propuestos para problemas relacionados con la lengua de señas en los cuales se utilizan las 3DCNN [Sharma y Kumar 2021] [Al-Hammadi y col. 2020] [«Proceedings of the 7th International Conference on Computer and Communications Management» 2019]. Sin embargo esta última variante de las convolucionales no es generalmente tan poderosa como los modelos específicos

para aprendizaje secuencial como las Redes Neuronales Recurrentes, Gran Memoria de Corto Plazo y las Unidades con Escotilla Recurrentes(GRU).

1.5.2. Redes Recurrentes

La primera intuición acerca de los modelos recurrentes es que modelan la representación temporal de datos secuenciales tales como vídeos(secuencia de imágenes). Las redes neuronales recurrentes profundas han demostrado ser exitosas en diversas tareas de aprendizaje de secuencias, como lo son en la traducción automática (anteriormente tratada en el capítulo), reconocimiento del habla, subtitulado de vídeo, predicción de vídeo, reconocimiento de lengua de señas [Gutiérrez-González 2021] y, como no, en la tarea que nos deviene que es el SLP. Sin embargo, existen limitaciones para estas redes, como es el caso del gradiente explosivo y desvaneciente. Para evitar este tipo de fallas, las redes neuronales recurrentes fueron extendidas a unos modelos más sofisticados como lo son las LSTM y las GRU. Diferentes trabajos han explorado modificaciones de los anteriores modelos extendidos, tales como:

- aplicar los modelos basados en LSTM al espacio de las imágenes[Shi y col. 2015]
- usando LSTM multidimensionales(MD-LSTM)[Graves y col. 2007]
- apilando capas de LSTM para incluir correlaciones espacio-temporales abstractas[Finn y col. 2016]

Aunque últimamente los modelos transformadores han obtenido muy buenos resultados en el ultimo lustro dado su mecanismo de auto-atención y cómputo paralelo. Cabe destacar que en la mayoría de los modelos para SLP se emplea un modelo recurrente por la necesidad de una representacion temporal de la secuencia de datos

1.6. Reconocimiento de la Lengua de Señas

El reconocimiento de la lengua de señas es ampliamente usado. Se emplea en la identificación de gestos de las lenguas de señas usando conjuntos de datos existentes previamente. Las dos manos, la cabeza, los ojos, los labios y el cuerpo son, en esencia, lo utilizado en las lenguas de señas para realizar infinidad de gestos dedicados a funciones lingüísticas [Sandler 2012]. Al ser los gestos una parte fundamental de este tipo de lenguas, se requiere de su reconocimiento específico de ellos. Los métodos de reconocimiento de gestos se pueden dividir en dos grandes grupos, en dependencia si utilizan un enfoque basado en hardware como Kinect u otros tipos de sensores, o un enfoque basado en software [Mitra y Acharya 2007].

Dentro de este segundo enfoque destaca la detección de gestos por MediaPipe Holistic de Google logrando muy buenos resultados en dichas tareas [Google 2020] [Fig 1.6].

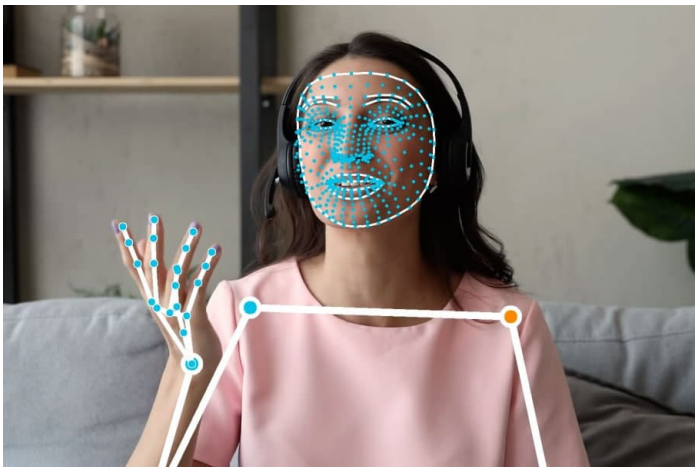

Figura 1.6: Software MediaPipe Holistic de Google

1.7. Investigaciones y proyectos relacionados con la Lengua de Señas Cubana

En el área de las tecnologías de la información y las comunicaciones sobre la Lengua de Señas Cubana el autor refiere otros 2 proyectos y/o trabajos.

El desarrollo de una aplicación móvil [Fig. 1.7] para dispositivos con sistema operativo Android. Una aplicación multimedia enfocada en las familias con niños sordo-mudos, para brindarles el vocabulario necesario para interactuar con sus hijos [Almarales-Wilson 2021] y la tesis de grado referente al reconocimiento de la lengua de señas cubana usando métodos de traducción basados en inteligencia artificial y aprendizaje de maquina [Gutiérrez-González 2021]

Se constata la existencia de una única investigación previa relacionada con la interpretación automática de la Lengua de Señas Cubana [Gutiérrez-González 2021] pero ninguna en su contraparte de SLP.

Figura 1.7: Aplicación Android para la LSC [Almarales-Wilson 2021]

Capítulo 2

Propuesta

El enfoque utilizado en este trabajo es la traducción automática basada en aprendizaje de máquinas, abordada en la subsección 1.3.3 y la combinación de redes convolucionales y redes LSTM mencionado en las subsecciones 1.5.1 y 1.5.2 . Se decidió utilizar dicha combinación puesto que los métodos del estado del arte basados en transformadores, pese a dar buenos resultados, requieren una cantidad exhorbitante de datos, de buena calidad y bien anotados, a diferencia de los datos que el autor emplea en este trabajo, de los cuales se abordará más adelante. Compartimos el objetivo del trabajo que precedió al nuestro de lograr que en un futuro pueda ser utilizado un modelo bilateral de reconocimiento y producción de lengua de señas a través de un dispositivo móvil y/o una web.

Por otra parte, el enfoque de utilizar un método de traducción automática basado en aprendizaje de máquinas porque es el campo de investigación principal de los autores, el cual es el de la inteligencia artificial. Para realizar la generación de avatares a priori se presenta la alternativa que se explica a continuación.

2.1. Alternativa Step by Step

Se decidió esta única alternativa Step by Step puesto que el conjunto de datos era muy pequeño, poco generalizado, con errores tanto en la escritura como en la detección de, alguno que otro, cuadro(FPS) del gesto captado. Además, por la investigación de [Gutiérrez-González 2021], conocemos que permite la re-utilización de anteriores trabajos desarrollados, explícitos, o no, para la traducción de la Lengua de Señas Cubana,la separación en sub-sistemas permite evaluar de forma más precisa cada paso del proceso y permite entrenamientos más cortos y con menos poder de cómputo, justo lo requerido dada la naturaleza de los datos con los que se trabaja.

Por lo anteriormente establecido, contamos primero con el trabajo con los datos, los cuales fueron procesados al ser cargados del almacenamiento en la nube donde

estaban guardados, y siguiendo la idea de [Gutiérrez-González 2021] utilizamos solamente los puntos del cuerpo y de las manos obtenidos a través de su invetigación. Se constan de 67 puntos espaciales siendo 25 del cuerpo(obviando el tren inferior de puntos), 21 para la mano derecha e idém para la mano izquierda [Fig 2.1].

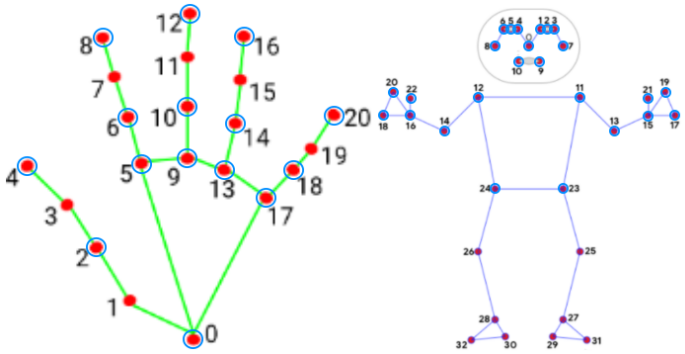

Figura 2.1: Todos los puntos obtenidos sin obviar el tren inferior

2.1.1. Extraer los datos de la nube

Utilizando una biblioteca especializada se extraen del almacenamiento en la nube de [Gutiérrez-González 2021]. Por tanto este paso no representa un problema y se logran obtener los datos de dicha investigación anterior para su ulterior uso.

2.1.2. Limpieza de los datos

Los cuadros(frames) del conjunto de datos son variados así que son todos llevados a 24 frames para una mayor homogeneidad del corpus,para agilizar el entrenamiento y porque la estructura neuronal a utilizar requiere de una entrada y una salida de un tamaño fijo.

A todas las palabras se le hace limpieza con expresiones regulares para eliminar la mayoría de imperfecciones posibles en las palabras, el uso de ”nn” en lugar de la ”ñ”, presencia de números puesto que al agrupar las frases de varios léxicos distintos, estos se renombraban.

2.1.3. Utilizando Embedding de palabra

A pesar de la limpieza se escapaban algunas faltas de ortografía o errores de escritura originados a la hora de teclear dicha frase durante su creación. Es por esto que se recurre a utilizar un modelo de incrustación (embedding) de palabra para exprimir lo máximo posible la información semántica de las frases del corpus [Fig 3.1],

dada la vérsatilidad del embedding de palabra utilizado el cuál es un modelo cargado pre-entrenado de más de 3 billones de vocablos y caracteres en español cortesía de [Almeida y Bilbao 2018].

2.1.4. Extrayendo la matriz de cada frase

Posteriormente de que se carga el modelo en español del embeddings, se le halla la matriz de tamaño 5x400 a cada frase del corpus. Cada frase contiene un máximo de 5 palabras y como en su mayoría son frases de una sola palabra pues se rellenaba con ceros las filas que no correspondían. Los vectores del embedding son de dimensión 400 por lo que se crea un arreglo de numpy de ceros antes de asignarle los vectores devueltos.

2.1.5. Encontrando similitud de palabras

Pese a ser un embedding tan potente, en cuanto a cantidad de vocablos con que se entrenó, al ser el idioma tan dependiente de la zona geográfica, como es nuestro caso con el Español, pues en el corpus se encontraron palabras que no tenían representación en el modelo de Word2Vec.

Para solventar dicha situación se recurrió una biblioteca especializada en similitud de palabras para encontrar la palabra más parecida dentro del vocabulario del modelo de embedding.

2.1.6. Arreglando errores ortográficos

Aún así por errores ortográficos (y por inexistencia de algunas) hubo palabras que, al no poseer tilde o estar escritas con mucha falta de ortografía, se tuvieron que codificar a mano en un diccionario de palabras mal escritas dado que eran pocas. Este paso de la solución, dista mucho de ser ideal, pero la similitud de palabras que fueron encontradas pues poseían diferencia semántica muy importante, lo cual afectaría muchísimo al modelo propuesto.

2.1.7. Limitando a una sola seña por frase

Luego de tener el conjunto de frases curadas y limpias, quedaba aún el problema de las palabras repetidas que tienen más de una seña, en dependencia de qué léxico del corpus brindado por el CENDSOR se revise. Para dicho problema se optó por escoger solamente la primera que apareciera puesto que solo complejizaría más la tarea presentada dado que para una misma frase en el corpus podía incurrir en una variación bastante considerable, es decir la señas para una misma palabra de un

intérprete a otro variaban mucho para el caso de la palabra paz con 4 tipos de señas distintas(como si de una ”falta de ortografía” se tratase).

2.1.8. Inicialización del modelo

Tensorflow y Keras fueron utilizados Una vez efectuada la asociación se inicializa el modelo secuencial que consta de 3 capas LSTM, luego a continuación de las anteriores se incluye una capa convolucional de una dimensión, una LSTM, otra 1DConv, otra LSTM y una capa de Reshape para obtener los resultados en las dimensiones esperadas.

El enfoque utilizado en este trabajo para realizar la predicción fue el de utilizar una RNN, específicamente una red de LSTM y además con capas convolucionales. Como se explicó en la sección 1.5.2, estas son muy utilizadas por su capacidad de recoger información del contexto y las deconvolucionales (convolucionales transpuestas) para ampliar las dimensiones y poder llevar del espacio menor a uno mayor.

No se tuvieron en cuenta los enfoques basados en métodos estadísticos debido a que no se disponía con un corpus suficientemente preparado y organizado, con sus respectivas probabilidades para atacar el problema con dichos métodos.

Fueron descartados los enfoques basados en métodos semánticos debido a que no es del dominio de los autores del trabajo, la estructura gramatical y semántica de la Lengua de Señas Cubana, siendo necesaria la ayuda de otros colaboradores, lo cual pertenecía a los objetivos del trabajo en cuestión.

2.1.9. Graficación y Animación

Posteriormente a obtener la matriz de 24 frames con los 67 puntos, de 3 dimensiones cada uno, se utiliza un graficador que está incorporado en el lenguaje utilizado, así como la función utilizada para lograr la animación. Al ser puntos espaciales , la graficación se realizaba en un ángulo que no era el ideal, para lo cual se realizaron transformaciones para lograr una visualización ideal de la animación con una vista casi frontal,puesto que tiene 1 grado de inclinación, lo cual es despreciable al evaluar el resultado visual.

Dicho resultado se guarda en video en formato mp4 para su ulterior comparación con valores reales similares.

Capítulo 3

Detalles de Implementación y Experimentos

Para poder valorar la viabilidad de la propuesta realizada en este trabajo, es necesaria la implementación de un prototipo del modelo explicado en el capítulo anterior.

3.1. Herramientas y tecnologías utilizadas

3.1.1. Lenguaje de programación Python

Python es un lenguaje de programación de propósito general y alto nivel desarrollado por Guido van Rossum en 1991. Su filosofía de diseño enfatiza la legibilidad del código con el uso de sangría significativa. Python se tipifica dinámicamente y tiene incorporado un recolector de basura. Admite múltiples paradigmas de programación, incluida la programación estructurada, orientada a objetos y funcional. La más reciente versión liberada al momento de realizarse este trabajo es la 3.11 [PSF 2022].

Es altamente empleado para ingeniería y análisis de datos, aprendizaje de máquina e inteligencia artificial gracias a sus infinidad de bibliotecas como NumPy, Tensorflow, Keras, Pytorch, SciPy, Pandas y Matplotlib, entre otras creadas tanto por el mismo equipo de trabajo de Python como la propia comunidad. Para desarrollo web cuenta con marcos de trabajo (frameworks) como Django. Es altamente utilizado en la educación al ser de fácil aprendizaje y asimilación logrando así reducir la barrera de entrada al mundo de la programación a todo aquel interesado.

En la actualidad sigue manteniendo el primer puesto de los índices de TIOBE y PYPL ratificando el interés de gran parte de la población y de los empleadores por este lenguaje y las ventajas que ofrece. Grandes organizaciones como Google [PSF

2021a], el CERN [CERN 2014], la NASA [PSF 2021b], Yahoo [PSF 2020], Wikipedia, Amazon, Facebook, Instagram [Facebook 2018], Spotify, entre otros. Para este trabajo se utilizó la versión 3.7 para lograr una retro-compatibilidad alta.

3.1.2. Numpy

Constituye una biblioteca de Python de código abierto, la cual permite generar, tanto vectores como matrices, de grandes dimensiones y operar de manera cómoda y sencilla con ellos [Numpy 2012]. Consigue esto gracias a que utiliza internamente el lenguaje C para lograr efectuar de forma rápida operaciones muy costosas entre elevadas dimensiones. En las etapas posteriores a la limpiezas de los datos explicada en el capítulo anterior se utiliza Numpy para el trabajo con los datos.

3.1.3. Matplotlib

Matplotlib es una biblioteca de gráficos creada en 2003 por John D. Hunter para el lenguaje de programación Python y su extensión matemática numérica NumPy. Proporciona una API orientada a objetos para incrustar gráficos en aplicaciones utilizando kits de herramientas GUI de uso general como Tkinter, wxPython, Qt o GTK. Es una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas. Matplotlib hace que las cosas fáciles, pues sigan siendo fáciles y las difíciles sean posibles, como lo es crear gráficos con calidad de una publicación y hacer figuras interactivas que puedan hacer zoom, desplazarse, actualizar.

3.1.4. Tensorflow, Keras y Addons

Tensorflow es una biblioteca de software gratuita y de código abierto para el aprendizaje automático de extremo a extremo y la inteligencia artificial desarrollada por el equipo de Google Brain. Se puede usar en una gran variedad de tareas, pero tiene un enfoque particular en el entrenamiento y la inferencia de redes neuronales profundas. Además brinda interfaces de forma oficial para C + +, Haskell, Java, Go, Rust y Python.

Keras es una interfaz de alto nivel de Tensorflow gratuita y altamente productiva para resolver problemas enfocados en el aprendizaje profundo.

Addons es un repositorio de contribuciones que se ajustan a patrones de API bien establecidos, pero implementan nuevas funciones que no están disponibles en el núcleo de TensorFlow. TensorFlow admite de forma nativa una gran cantidad de operadores, capas, métricas, pérdidas y optimizadores.

Para la creación y entrenamiento del modelo de generación de avatares propuesto en la sección 2.1 se utilizaron las tecnologías anteriormente mencionadas así como

para salvar los pesos del modelo una vez entrenado.

3.1.5. Scipy

SciPy es una biblioteca de Python gratuita y de código abierto que se utiliza para la computación científica y la informática técnica. SciPy contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales e imágenes, solucionadores de ODE y otras tareas comunes en ciencia e ingeniería. Esta creada encima de la biblioteca Numpy anteriormente mencionada y fue desarrollada por la compañía Enthought en el año 2001.

3.1.6. Google Colaboratory

Google provee un servicio gratuito en el navegador el cual brinda, de manera temporal, recursos computacionales (CPU,TPU,GPU,RAM,HDD) mientras se haga uso activo de los mismos. En dicha plataforma se le permite al usuario escribir y ejecutar codigo Python en un entorno basado en Jupyter Notebook.Siendo así una herramienta ideal para el aprendizaje de máquinas, el análisis de datos y la educación al posibilitar que personas de pocos recursos puedan acceder de forma fácil a herramientas para el aprendizaje de máquina las cuales muchas de ellas ya vienen incluidas en el entorno o son muy sencillas de instalar en el mismo. Cabe destacar que además del plan gratuito que incluye CPU de última generación, 13 Gb de RAM y 110 Gb de almacenamiento, nos da la posibilidad de utilizar Google Drive para ampliar el almacenamiento y varios planes de pago para aumentar la capacidad computacional del entorno.

Todo el trabajo fue llevado a cabo utilizando este servicio para dejar de manera accesible todo el flujo de trabajo.

3.1.7. Gensim

Gensim es una biblioteca gratuita de Python de código abierto para representar documentos como vectores semánticos, de la manera más eficiente (computacionalmente) y sin dolor (humanamente) posible. Está diseñado para procesar textos digitales crudos y sin estructura ("texto sin formato") utilizando algoritmos de aprendizaje automático no supervisados, entre los que se encuentran implementados Word2Vec [Řehůřek y Sojka 2010].

3.1.8. Word2Vec

Word2vec es una técnica para el procesamiento del lenguaje natural publicada en 2013. El algoritmo word2vec utiliza un modelo de red neuronal para aprender asocia-

ciones de palabras de un gran corpus de texto. Una vez entrenado, dicho modelo puede detectar palabras sinónimas o sugerir palabras adicionales para una oración parcial. Como su nombre lo indica, word2vec representa cada palabra distinta con una lista particular de números llamada vector. Los vectores se eligen cuidadosamente de modo que capturen las cualidades semánticas y sintácticas de las palabras; como tal, una simple función matemática (similitud de coseno) puede indicar el nivel de similitud semántica entre las palabras representadas por esos vectores. Para este trabajo se utilizará el modelo entrenado en 3 mil millones de vocablos en español. Esto nos dota de una gran herramienta que protege la semántica de las palabras ,como se puede evidenciar en la Fig 3.1.

```
vect = wv["rey"] - wv["hombre"] + wv["mujer"]
wv.similar_by_vector(vect,topn=1)

[('reina', 0.7074883580207825)]
```

Figura 3.1: Grado de aprendizaje de la semántica de Word2Vec en español

3.1.9. difflib

Este módulo de Python proporciona clases y funciones para comparar secuencias. Se puede usar, por ejemplo, para comparar archivos y puede producir información sobre diferencias de archivos en varios formatos, incluidos HTML y contexto y diferencias unificadas.

clase SequenceMatcher

Esta es una clase flexible para comparar pares de secuencias de cualquier tipo, siempre que los elementos de la secuencia se puedan modificar. El algoritmo básico es anterior a un algoritmo publicado a fines de la década de 1980 por Ratcliff y Obershelp con el nombre hiperbólico de ”coincidencia de patrones gestálticos”, y es un poco más elegante. La idea es encontrar la subsecuencia coincidente contigua más larga que no contenga elementos ”basura”; estos elementos ”basura” son los que no son interesantes en algún sentido, como líneas en blanco o espacios en blanco. (El manejo de basura es una extensión del algoritmo de Ratcliff y Obershelp). Luego, la misma idea se aplica recursivamente a las partes de las secuencias a la izquierda y a la derecha de la subsecuencia coincidente. Esto no produce secuencias de edición mínimas, pero tiende a generar coincidencias que ”parecen correctas” para las personas.

get_close_matches

Método para seleccionar la secuencia más similar a una secuencia dada dentro de un vocabulario.

3.1.10. Boto3

Se utiliza el Kit de desarrollador de software (SDK por sus siglas en inglés) de AWS para Python (Boto3) para crear, configurar y administrar servicios de AWS, como Amazon Elastic Compute Cloud (Amazon EC2) y Amazon Simple Storage Service (Amazon S3). El SDK proporciona una API orientada a objetos, así como acceso de bajo nivel a los servicios de AWS.

Este SDK es empleado para acceder a los datos.

3.1.11. pathlib

Este es un módulo de Python que ofrece clases que representan rutas de sistemas de archivos con semántica apropiada para diferentes sistemas operativos. Las clases de ruta se dividen entre rutas puras, que proporcionan operaciones puramente computacionales sin Entrada/Salida (I/O por sus siglas en inglés), y rutas concretas, que heredan de rutas puras pero también proporcionan operaciones de I/O.

La clase Path de esta biblioteca es ampliamente usada en este trabajo.

3.2. Implementación de un prototipo

El prototipo implementado realiza todos los pasos mencionados en la sección 2.1

Primeramente se realizó todo el trabajo en Google Colaboratory (Colab para abreviar).

3.2.1. Extracción de los Datos

Para la extracción de los datos se utilizó la biblioteca boto3 con acceso al almacenamiento en la nube donde se encontraban los datos. Para ello se importa el método client de boto3, el cual es instanciado en la url donde se encuentra el conjunto de datos, además de las claves de acceso, las cuales por privacidad del dueño de las mismas se sustituyen por valores de "X".

```
1 from boto3 import client
2
3 s3_client = client(
4     "s3",
```

```
    config=Config(
        signature_version="s3v4",
        retries={"max_attempts": 10},
        s3={"addressing_style": "path"},
    ),
    region_name="ams3",
    endpoint_url="https://ams3.digitaloceanspaces.com",
    aws_access_key_id="XXXXXXXXXXXXXXX",
    aws_secret_access_key="XXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
)
```

Ejemplo de código 3.1: Instanciar cliente s3

Una vez obtenido el cliente configurado se descargan los datos del mismo, conociéndose que se encuentran en el bucket "lsc-corpus", los de las señas aisladas, puesto que son los únicos datos con seguridad de la seña que corresponde a cada frase.

```
from pathlib import Path

root=Path("/content")
bucket="lsc-corpus"

paginator = s3_client.get_paginator('list_objects_v2')
pages = paginator.paginate(Bucket=bucket)

for page in pages:
    for obj in page['Contents']:
        extension = obj['Key'].split('.')[-1]
        name=obj['Key'].split('.')[0]
        if extension  in ['json']:
          drivepath=(root/f"tesis-generacion-lsc/{obj['Key']}").
   resolve()
          (drivepath/'..').resolve().mkdir(parents=True,exist_ok=
   True)
          drivepath.touch()
          s3_client.download_file(bucket,obj['Key'],str(drivepath))
```

Ejemplo de código 3.2: Descargar usando el cliente s3

Aquí se realiza un paginado para poder recorrer de manera adecuada los elementos del bucket para luego chequear la extensión de los archivos contenidos en cada página. Se conocen previamente que los datos anotados de los puntos de interés estaban recogidos en archivos con extensión json por cada frase. En la primera parte del código se importa la clase Path de pathlib para trabajar de manera rápida y efectiva con los directorios que se van creando.

3.2.2. Limpieza de los datos

Una vez ya poseemos los archivos json referentes a cada frase del corpus podemos entonces cargarlos para limpiarlos y organizarlos en un dataset mejor estructurado donde estén todas las frases juntas.

En este código utilizamos expresiones regulares para la limpieza de las llaves, ya que incurrían en faltas de ortografía, errores tipográficos y de copiado. La primera expresión elimina todo lo que esté entre paréntesis en las llaves, puesto que esto no aporta nada a la información semántica de los mismos. Luego se reemplaza todas las ”nn” por una ”ñ” y se eliminan los dígitos que hayan quedado de manera residual del primer método.Para a modo de cierre de la limpieza de las llaves eliminar todos los guiones entre palabras para que esten de manera individual.

```
1
2  import numpy as np
3  import json
4  import re
5
6  root_tesis = (root/f"tesis-generacion-lsc/").resolve()
7  cendsor_path = "cendsor-corpus/keypoints/"
8  fps=24
9
10 word_poses_dict={}
11
12 for dirpath,dirnames,filenames in os.walk(root_tesis/cendsor_path):
13   for filename in filenames:
14     key,extension=filename.split('.')
15
16     ############################
17     # Limpieza de las llaves
18     ############################
19
20     key=re.sub(r"\s*\(.*\)","",key)
21     key= key.replace("nn",'ñ')
22     key=re.sub(r"\s?\d","",key)
23     key = key.replace("-"," ")
24
25     ############################
26
27     with open((root_tesis/cendsor_path/filename).resolve(), 'r',
       encoding='utf-8') as json_list:
28       poseframes=np.array(json.load(json_list))
29       json_list.close()
30
31       ##########################################################
32       # Limpieza y reajuste de los frames de los gestos
```

```
33        ##########################################################
34        filtro = [m!=0.0 for m in  np.mean(poseframes, axis=1)]
35
36        poseframes=poseframes[filtro]
37
38        total_frames = poseframes.shape[0]
39
40        n = math.floor(total_frames/fps)
41
42        poseframes= poseframes[::n,:]
43
44        while poseframes.shape[0]>24:
45          if poseframes.shape[0]%2:
46            poseframes=poseframes[1:,:]
47          else:
48             poseframes=poseframes[:-1,:]
49        ##########################################################
50
51        single_gesture = poseframes.tolist()
52        try:
53          word_poses_dict[key].append(single_gesture)
54        except:
55          word_poses_dict[key]=[single_gesture]
```

Ejemplo de código 3.3: Cargar los json y limpiar las llaves

En el caso de los gestos existen en el corpus frames en los cuales los modelos no detectaban nad y por tanto estos devolvian (0,0,0) para todos los puntos de interés de ese frame. Por tanto se hace uso de la biblioteca json para cargar el archivo y de la biblioteca numpy para realizar la limpieza de una manera más cómoda y rápida. Luego de obtener la máscara (filtro) con la información de los frames vacíos, simplemente se indexa en el arreglo numpy guardando su resultado como si fuera ahora el arreglo original. Pero proseguía un problema más, el tamaño de los frames no era parejo. Algunos archivos tenían más frames que otros, por lo tanto se decidió llevarlos todos a 24 frames utilizando la función floor de la biblioteca math para obtener un valor que al dividir por la cantidad de frames no diera superior y fuera entero. Posteriormente, como no van a quedar exactamente de tamaño 24, sino mayor, se va reduciendo en 1 la cantidad de frames por los extremos hasta llegar a la cantidad deseada.

Como información adicional se guarda el diccionario tanto en la nube como en local para su mejor aprovechamiento y eliminarnos pasos extras a la hora de probar el modelo.

3.2.3. Cargando el embedding de palabra

Primeramente se debe descargar el archivo de los KeyedVectors el cual pesa alrededor de los 3Gb, razón por la cual se ha estado desarrollando todo este flujo de trabajo directamente en Colab puesto que es un gran modelo para capturar significado semántico dado lo limitado del corpus en ese sentido.

```
1
2  $ wget "https://zenodo.org/record/1410403/files/keyed_vectors.zip"
3  $ unzip keyed_vectors.zip
```

Ejemplo de código 3.4: Descargar datos del modelo KeyedVectors de su sitio web

Luego de descargado y descomprimido podemos entonces cargar el modelo.

```
1  from gensim.models import KeyedVectors
2  wv = KeyedVectors.load('complete.kv', mmap='r')
```

Ejemplo de código 3.5: Cargar modelo de KeyedVectors

Como se aprecia, KeyedVectors es un modelo perteneciente a la biblioteca gensim, el cual es uno de los elementos que componen a Word2Vec y justamente el único que necesitamos para nuestro propósito de capturar la semántica de las frases. Cargar completamente el modelo Word2Vec exige mucha memoria RAM puesto que es un modelo de más de 7 Gb en su completitud y es esa la razón por la que se escoge solamente el diccionario de los vectores claves (KeyedVectors).

.El archivo complete.kv es el resultado de descomprimir el modelo descargado.

3.2.4. Encontrando similitud de palabras

Como se plantea en la propuesta primeramente debemos asegurarnos de que se logre encontrar una palabra dentro del vocabulario de nuestro embedding, por eso se declara el siguiente método a continuación.

```
1  def find_similar(word, vocab):
2    import difflib
3    best_match=difflib.get_close_matches(word,vocab)
4    if len(best_match) != 0:
5        score = difflib.SequenceMatcher(None,word,best_match[0]).
   ratio()
6        return best_match[0],score
7    else:
8        return None
9
```

Ejemplo de código 3.6: Método para encontrar palabras similares en un vocabulario

En este metodo se utiliza el metodo get_close_matches para encontrar la palabra más parecida sintácticamente a la proporcionada, en el vocabulario dado. Luego si encuentra que existe alguna palabra que sea similar, pues se instancia una clase SequenceMatcher para hallar que tan similares son. Por supuesto este ultimo paso solo es necesario en caso de necesitarse cuantificar la similitud.

3.2.5. Extrayendo la matriz de cada frase

El siguiente método genera una matriz de *num_vecs* x *vec_dim* que es la dimensión de los vectores. Primero se crea un diccionario de palabras mal escritas (posible en este escenario dado que solo se cuentan con poco más de 1050 vocablos) y además para no arrastrar el error semántico de unos datos erróneos se escogen palabras de igual significado semántico.

Luego se crea un arreglo numpy de ceros con las dimensiones específicadas para posteriormente conformarlo con los vectores de cada palabra de la frase de 5 vocablos en el caso del corpus utilizado.

```
def get_vector(key,word_vectors,num_vecs=5):
wrong_words={'antonimo': 'antónimo',
             'audiologia':'audiología',
             'barsura':'basura',
             'defectologia':'defectología',
             'empanizar':'empanar',
             'insipido': 'insípido',
             'investar':'inventar',
             'lexicologia': 'lexicología',
             'ostion':'ostra',
             'panciencia':'paciencia',
             'policlinica': 'policlínica',
             'querologia':'querología',
             'rehablilitacion':'rehabilitación',
             'sacapunta':'sacapuntas',
             'señacionario':'diccionario',
             'señario':'libro',
             'zapia':'compartición'}
vector=np.zeros((num_vecs,word_vectors.vector_size))

for i,word in enumerate(key.split(" ")):
  if word in word_vectors.vocab.keys():

    wordvec = word_vectors[word]
  else:
    word=wrong_words[word]
    nword, similarity = find_similar(word,word_vectors.vocab.keys
 ())
    print(key,"Not Founded")
```

```
29        print(nword,similarity)
30
31        wordvec = word_vectors[nword]
32      vector[i:]=wordvec
33    return vector
```

Ejemplo de código 3.7: Obtener el vector dada una palabra

Luego de creado el arreglo de numpy, se verifica que la palabra esté en el vocabulario del embedding antes de indexar la palabra en busca de su vector. En caso de no existir la palabra pues se utiliza el método descrito en la subsección anterior para hallar un vocablo sintácticamente similar que si pertenezca al vocabulario del embedding.

Finalmente, después de rellenado el arreglo de numpy, este es retornado.

3.2.6. Creando los conjuntos

En este código creamos y llenamos los conjuntos X e y para su ulterior uso en el modelo. *num_vecs* es escogido como el mayor número de palabras de todo el conjunto de datos, el cual en exploraciones a los datos fue revelado que es 5.

```
1
2 X=[]
3 y=[]
4
5 num_vecs=max([len(key.split(" ")) for key in dataset.keys()])
6
7 for key,gestures in dataset.items():
8   vector=get_vector(key,word_vectors,num_vecs)
9   X.append(vector)
10  y.append(np.array(gestures[0]).reshape((fps,sum([dim[0] for dim in
      dims]),dims[0][1])))
```

Ejemplo de código 3.8: Crear los conjuntos X e y

Para evitar el problema de que a cada frase le correspondieran varias señas, con alta probabilidad de que fueran distintas, se optó por solamente escoger la primera de las señas como representativa de las demás.

3.2.7. Métodos para la animación

Primero instanciamos los valores necesarios para animar correctamente un conjunto de puntos con estructura igual a la de los datos .

```
1 body_dim = 25*3
2 left_dim = 21*3
3 right_dim = 21*3
4
```

```
dims=[(25,3),(21,3),(21,3)]

bodyjoints=[[1,7],[4,8],[9,10],[11,12],[11,13],
[12,14],[11,23],[12,24],[23,24],[13,15],[14,16]]
leftjoints=[[0,1],[1,2],[2,3],[3,4],
            [0,5],[5,6],[6,7],[7,8],
            [5,9],[9,10],[10,11],[11,12],
            [9,13],[13,14],[14,15],[15,16],
            [13,17],[17,18],[18,19],[19,20],[0,17]]
rightjoints=leftjoints

fullbodyjoints=[bodyjoints,leftjoints,rightjoints]

parts_names=["body","left","right"]
```

Ejemplo de código 3.9: Instanciar datos necesarios para animar

Utilizamos el siguiente metodo en caso de que los puntos no vengan en formato (24,67,3) y simplemente vengan en formato (24,201)

```
def extract_3d_shape(array,dims):
      init=0
      ending=0
      parts=[]
      for dim in dims:
        ending+=dim[0]*dim[1]
        parts.append(array[:,init:ending].reshape((array.shape[0],
   dim[0],dim[1])))
        init+=dim[0]*dim[1]

      single_gesture={parts_names[i]:parts[i] for i in range(len(
   dims))}

      return single_gesture
```

Ejemplo de código 3.10: Extraer las partes en 3d

Utilizamos la lista con las dimensiones para de poder, en caso de ser necesario, incluir graficaciones de otras partes del cuerpo que fueron excluidas.

El siguiente método contituye la base de todos los métodos para las animaciones de este trabajo. Se utilizan varias clases y métodos de la biblioteca matplotlib para realizar la animación.

FuncAnimation recibe la figura en la que trabajar,la función de actualización, que es la encargada de la transición entre frames, y por último la cantidad de frames a animar.

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib import rc
```

```
import matplotlib.cm as cm

def transform_data(Data,i):
    from scipy.spatial.transform import Rotation as R

    r = R.from_rotvec(np.pi/2 * np.array([0,0,1]))

    X=np.array(Data[i,:,0])
    Y=np.array(-Data[i,:,1])
    Z=np.array(Data[i,:,2])
    meanpoint=(X[X!=0].mean(),Y[Y!=0].mean(),Z[Z!=0].mean())

    D = np.array([x if x!=(0.0,0.0,0.0)
                    else meanpoint
                for x in zip(X, Y, Z) ])
    D = r.apply(D)
    return D

def animate(
            Data,
            frames=None,
            figsize=(7.0,3.5),
            elev=None,
            angle=None,
            dims=dims,
            text=False,
            joints=False):

  rc('animation',html='jshtml')
  fig_size_x,fig_size_y=figsize

  plt.rcParams["figure.figsize"]= [fig_size_x,fig_size_y]
  plt.rcParams['figure.autolayout']= True

  fig = plt.figure()
  ax = fig.add_subplot(projection='3d')
  colors=['b','y','r','k']

  def update(i):
    ax.clear()

    D = transform_data(Data,i)

    s=0
    e=0
    for j, dim in enumerate(dims):
      dim = dim[0]
      e+=dim
```

```
52
53        x=D[s:e,0]
54        y=D[s:e,1]
55        z=D[s:e,2]
56
57        # print((x.mean(),y.mean(),z.mean()) != meanpoint )
58        if (x.mean(),y.mean(),z.mean()) != meanpoint:
59          # print(x.mean(),y.mean(),z.mean())
60          ax.plot(D[s:e,0],D[s:e,1],D[s:e,2],f'{colors[j]}.')
61        if text:
62          for p in range(s,e):
63            ax.text3D(D[p,0],D[p,1],D[p,2],str(p))
64        if joints:
65          for h,k in fullbodyjoints[j]:
66            ax.plot([D[s+h,0],D[s+k,0]],[D[s+h,1],D[s+k,1]],[D[s+h,2],
    D[s+k,2]],f'{colors[j]}')
67        s+=dim
68
69      ax.set_axis_off()
70      if elev and angle:
71        ax.view_init(elev,angle)
72
73
74      return ax
75
76    if frames is None:
77      frames=24
78    return FuncAnimation(fig,update,frames=frames,repeat=True)
```

Ejemplo de código 3.11: Método base para animar

Se utiliza la clase Rotation del módulo transform de scipy para rotar de manera efectiva la figura para que sea visualmente atractiva la animación desde una vista frontal. Al igual que si existen muchos puntos espaciales en el (0,0,0) se halla la media de los puntos distintos del cero del espacio para no distorsionar el graficado de la animación, esto ocurre por fallos del método que creó el corpus inicialemente al no detectar alguna región del cuerpo, es decir ya los datos con los que se cuentan vienen dados con cierto grado de error. Luego, para que sea visualmente llamativo, se utilizan las dimensiones de las distintas partes del cuerpo para graficar de un color diferente cada una de las partes y se utilizan las uniones declaradas inicialmente.

En caso de ser necesario puede agregarse el texto del número correspondiente al punto para mejor visualización.

También se incorpora la posibilidad de modificar las dimensiones de la figura, así como el angl¡ulo de azimut y de elevación correspondiente.

En cuanto al último método, este solo representa una composición de los métodos anteriores con una organización intermedia de los arreglos correspondientes a cada

parte en caso de que la entrada no tenga la estructura idónea para la graficación.

```
1
2  def plotanimation(result):
3      if result.shape[2]==201:
4          dict_parts=extract_3d_shape(result[0],dims)
5
6          body_frames = dict_parts["body"]
7          left_frames = dict_parts["left"]
8          right_frames = dict_parts["right"]
9
10         result = np.hstack((body_frames,left_frames,right_frames))
11
12
13   a=animate(result,result.shape[0],(12,10),90,1,dims,text=False,
       joints=True)
14
15   return a
```

Ejemplo de código 3.12: Graficar la animación

3.3. Entrenamiento

El modelo presentado, como se mencionó en capítulos anteriores, necesita ser debidamente entrenado. Dado que el dominio e imagen del conjunto disponible para el entrenamiento es alarmantemente corto en comparación con los vocablos utilizados comúnmente en el día a día por una persona promedio. Hablamos de que alrededor de 20 mil vocablos son empleados a diario en una jornada normal, mientras que el corpus brindado por el CENDSOR solo cuenta con poco más de 1050 vocablos(con algunos teniendo más de 1 respresentación visual diferente)

Con estas condiciones del corpus, el enfoque adoptado para el entrenamiento es el de generar sobreadecuación (overfitting) sobre los únicos vocablos conocidos para que así al componerse con el uso del embedding de palabra puedan encontrarse similaridades semánticas entre una palabra desconocida y una del vocabulario disponible. subsectionDefinición del modelo

Como se explicó en la propuesta, el enfoque adoptado para este modelo es el de utilizar redes neuronales recurrentes, más específicamente LSTM multicapas con capas deconvolucionales para ampliar las dimensiones.

```
1
2  from tensorflow.keras.layers import InputLayer, LSTM ,
       Conv1DTranspose, Reshape
3  from tensorflow.keras.models import Sequential
4  from tensorflow_addons.metrics.r_square import RSquare
5
```

```
def get_model(X, y):
    model = Sequential()
    model.add(InputLayer(X.shape[1:]))
    model.add(LSTM(128, return_sequences=True, activation="relu"))
    model.add(LSTM(256, return_sequences=True, activation="relu"))
    model.add(Conv1DTranspose(filters=100,kernel_size=10))
    model.add(LSTM(128, return_sequences=True, activation="relu"))
    model.add(Conv1DTranspose(filters=150,kernel_size=11))
    model.add(LSTM(201,return_sequences=True))
    model.add(Reshape(y.shape[1:]))
    loss = "mae"
    metrics = RSquare()
    model.compile(optimizer='Adam', loss=loss, metrics=metrics)
    return model
```

Ejemplo de código 3.13: Diseño del modelo

Las cantidad de unidades fueron escogidas basándonos en la unidades que reportaron beneficios para el trabajo precedente a este [Gutiérrez-González 2021]. Para la función de pérdida fue seleccionado el Error Medio Absoluto(MAE por sus siglas en inglés) puesto que compara efectivamente la diferencia de posición espacial dados los datos con que se cuentan. Además se usa el R Cuadrado como métrica de puntuación para poder entrenar de manera efectiva.

Las opciones basadas en transformadores requerían de cantidades de datos superiores a las que se contaban así que se tomó solo en cuenta los estudios referentes a las Redes neuronales recurrentes LSTM.

Se utilizó un método supervisado de entrenamiento usando un enfoque regresivo dada la no existencia de categorías calaramente definidas en el espacio de todas las posibles señas.

```


from tensorflow.keras.callbacks import EarlyStopping,
    ModelCheckpoint
from sklearn.model_selection import KFold,cross_val_score

X,y=(np.array(X),np.array(y))
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,
    random_state=42)

checkpoint_path= (droot_tesis/"generacion-lsc_0001.ckpt").resolve()

num_folds=5
batch_size=32
epochs=1000

kfold = KFold(n_splits=num_folds, shuffle=True)
```

```
16
17 # K-fold Cross Validation model evaluation
18 acc_per_fold=[]
19 loss_per_fold=[]
20 fold_no = 1
21 for train, val in kfold.split(X, y):
22
23   model = get_model(X, y)
24   if checkpoint_path.exists():
25     model.load_weights(checkpoint_path)
26
27   ckpt_callback= ModelCheckpoint(
28       filepath=str(checkpoint_path).format(epoch=0),
29       save_weights_only=True,
30       verbose=1,
31       save_freq=5*batch_size)
32
33   callback = EarlyStopping(
34       monitor= "loss",
35       patience=10)
36
37   print(model.summary())
38   # Generate a print
39   print('------------------------')
40   print(f'Training for fold {fold_no} ...')
41
42   # Fit data to model
43   history = model.fit(
44       X[train],
45        y[train],
46        batch_size=batch_size,
47        epochs=epochs,
48        callbacks=[callback,ckpt_callback])
49
50   # Generate generalization metrics
51   scores = model.evaluate(X[val], y[val],)
52   print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {
      scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
53   acc_per_fold.append(scores[1] * 100)
54   loss_per_fold.append(scores[0])
55
56   # Increase fold number
57   fold_no = fold_no + 1
```

Ejemplo de código 3.14: Graficar la animación

Se utiliza un entrenamiento con Kfold para iterar por conjuntos de datos separados de forma aleatoria y así lograr evaluar para todos los valores además de 1000 epochs. Se utilizan callbacks de parada temprana para en caso de que los valores de las

funciones de puntuación y de pérdida no varían mucho luego de un cierto tiempo y además se usa un callback para salvar los pesos del modelo durante el entrenamiento en caso de colapso del servicio de Colab o de fallos eléctricos. Dicha salva de los pesos del modelo es retomada en cada iteración del método Kfold para garantizar que se haga overfitting sobre los datos.

Podrá parecer extraño que se quiera sobreadecuar el modelo al conjunto de datos pero no es la primera vez que se hace esto sobre el vocabulario disponible de cierta tarea o problema, como se hizo en el caso de GPT3, el cual sobreadecúa los datos de todo internet logrando así dominar el vocabulario dado.

3.3.1. Experimentos

Como desenlace del entrenamiento del modelo se logró evidenciar buenos resultados dada la simpleza del método presentado, las limitantes y problemáticas del conjunto de datos.

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_68 (LSTM)              (None, 5, 128)            270848

 lstm_69 (LSTM)              (None, 5, 256)            394240

 conv1d_transpose_34 (Conv1D  (None, 14, 100)          256100
 Transpose)

 lstm_70 (LSTM)              (None, 14, 128)           117248

 conv1d_transpose_35 (Conv1D  (None, 24, 150)          211350
 Transpose)

 lstm_71 (LSTM)              (None, 24, 201)           283008

 reshape_17 (Reshape)        (None, 24, 67, 3)         0

=================================================================
Total params: 1,532,794
Trainable params: 1,532,794
Non-trainable params: 0
```

Figura 3.2: Resumen de las capas y parametros del modelo

Durante el entrenamiento se utilizaron las métricas Error Promedio Absoluto (MAE por sus siglas en inglés) como función de pérdida y R Cuadrado (R Squared) como función de puntuación dado que este es un problema de regresión.

Aquí se puede presenciar una comparativa de los resultados obtenidos para la primera frases del conjunto de datos.

```
Epoch 90/100
27/27 [==============================] - 6s 215ms/step - loss: 0.0332 - r_square: -892234.0625
Epoch 91/100
27/27 [==============================] - 6s 215ms/step - loss: 0.0330 - r_square: -231.8048
Epoch 92/100
27/27 [==============================] - 5s 189ms/step - loss: 0.0332 - r_square: -4670.8433
Epoch 93/100
27/27 [==============================] - 5s 190ms/step - loss: 0.0330 - r_square: -139.5199
Epoch 94/100
27/27 [==============================] - 5s 197ms/step - loss: 0.0327 - r_square: -3528.9436
Epoch 95/100
21/27 [====================>.......] - ETA: 1s - loss: 0.0321 - r_square: -262.0040
Epoch 95: saving model to /content/drive/MyDrive/tesis-generacion-lsc/generacion-lsc_0001.ckpt
27/27 [==============================] - 5s 193ms/step - loss: 0.0324 - r_square: -228.5174
Epoch 96/100
27/27 [==============================] - 5s 195ms/step - loss: 0.0324 - r_square: -346.2374
Epoch 97/100
27/27 [==============================] - 6s 240ms/step - loss: 0.0320 - r_square: -183.9487
Epoch 98/100
27/27 [==============================] - 6s 231ms/step - loss: 0.0317 - r_square: -678.0912
Epoch 99/100
27/27 [==============================] - 5s 190ms/step - loss: 0.0315 - r_square: -462.0569
Epoch 100/100
27/27 [==============================] - 5s 196ms/step - loss: 0.0310 - r_square: -288.4684
7/7 [==============================] - 2s 95ms/step - loss: 0.1394 - r_square: -125.9380
Score for fold 5: loss of 0.1393798589706421; r_square of -12593.804168701172%
```

Figura 3.3: Resultado de las métricas de los últimos epochs del modelo durante el entrenamiento

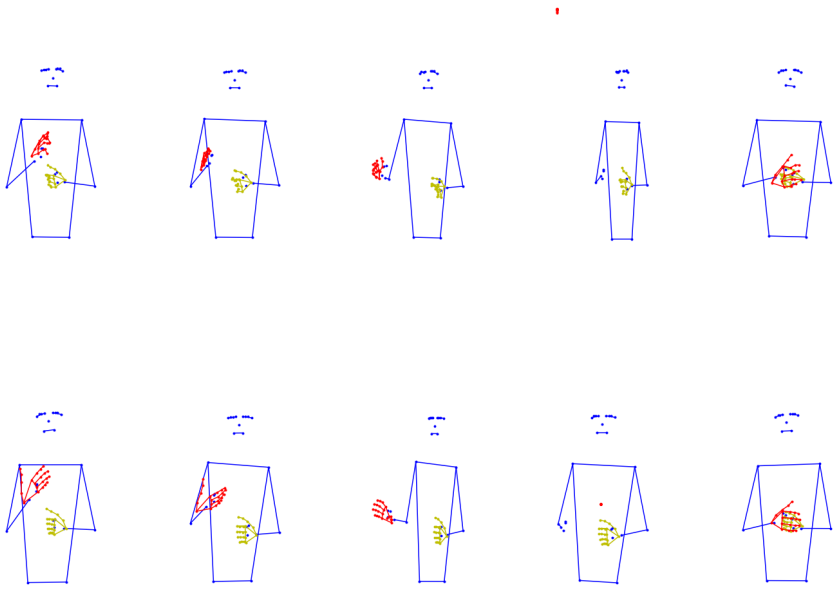

Figura 3.4: Comparativa de la frase ”abandonar” . Fila superior los valores predichos. Fila inferior los valores reales

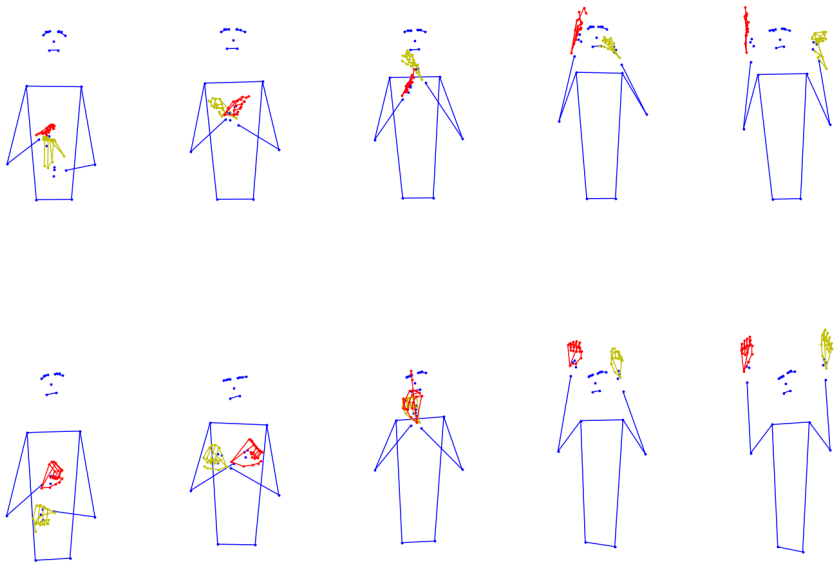

Figura 3.5: Comparativa de la frase ”por la mañana” . Fila superior los valores predichos. Fila inferior los valores reales

3.3.2. Discusión

Visualmente se obtuvieron buenos resultados, pero sin una medida específica salvo la percepción del autor. Muchos de los puntos referentes a los dedos no se detectaron como se correspondía y no se tuvo una métrica que describiera bien la puntuación del modelo de regresión. Esto se debe en gran parte a que los datos con que se contaban presentaban errores y, al ser pocos, se transmitían de manera fácil al modelo durante el entrenamiento, a pesar de emplear la sobreadecuación del mismo al corpus brindado.

A pesar de ello se obtuvieron resultados bastante aceptables y una puntuación del R Cuadrado bastante cercana al máximo posible que es 1.

Conclusiones

La generación de avatares para la Lengua de Señas Cubana representa el principar objetivo del trabajo. Durante la investigación se generan nuevos métodos, estructuras y corpus que posibilitan una mejor comprensión de esta temática, como a su vez, amplia la base de conocimiento actualmente disponible. Se introduce la problemática presente en el contexto social correspondiente y se profundiza en el estado del arte de diversos campos como la escritura de señas, haciendo enfásis en su estructura y composición, traducción automática, tecnología de avatares, producción de lengua de señas y reconocimiento de la misma, detectándose un crecimiento de la aceptación de los modelos que utilizan inteligencia artificial y aprendizaje de máquina. Destacar entre estos aspectos que los métodos por partes obtuvieron mejores resultados cuando se contaban con pocos datos al respecto, mientras que los enfoques extremo a extremo utilizando transformers son los que mejores resultados han obtenido a nivel mundial utilizando bases de datos bastante ricas y abundantes. También referir a los únicos 2 trabajos encontrados que tratan de alguna forma u otra la Lengua de Señas Cubana. Con todos estos ingredientes se escogió una propuesta de un modelo secuencia a secuencia, por partes, que utiliza un modelo secuencial de varias capas de redes recurrentes y de redes deconvolucionales para el ensanchamiento de dimensiones. Esta propuesta para la generación de avatares para nuestra lengua de señas utilizando el corpus generado por anteriores investigaciones fue concretada en un prototipo que utiliza un modelo de incrustaciones de palabras entrenado en un enorme corpus de español para capturar de manera efectiva el significado de las frases del corpus para así generalizar más el modelo. Se compararon las secuencias de imágenes obtenidas en las predicciones con las reales, obteniéndose buenos resultados a apreciación del autor. Dado que no existe un dominio del tema, la única falla grave es el hecho de que los dedos de las manos no se definen.

Recomendaciones

Con este trabajo se inicia una nueva rama de investigación y desarrollo referente a la producción de lengua de señas cubana. Al ser este un trabajo novedoso en el campo de la producción de señas, se deja como pauta inicial para futuros trabajos y una guía del proceder ante los limitados datos disponibles. Por lo que se seguirá trabajando en cumplirse los objetivos sin resolver en este trabajo y en el desarrollo de otros modelos para compararse los resultados. Se recomienda, a los investigadores de Cuba y a toda la comunidad, el crear conciencia sobre la importancia y necesidad de este tipo de estudios y herramientas para el correcto desarrollo de la comunidad sordo-muda cubana. Se incita a los desarrolladores cubanos, lingüistas y especialistas del lenguaje de señas a trabajar unánimemente en pos de lograr un corpus correctamente anotado y tanto extenso, como generalizado, para así obtener una herramienta de utilidad para que, en manos de los hipoacúsicos, sea la llave para romper las barreras comunicativas que les impiden desenvolverse y desarrollarse en nuestro país actualmente.

Bibliografía

(2021). https://www.visicast.cmp.uea.ac.uk/Visicast_index.html. (Vid. pág. 14)

(2021). https://www.visicast.cmp.uea.ac.uk/eSIGN/index.html. (Vid. pág. 14)

Al-Hammadi, M. H., Muhammad, G., Abdul, W., Alsulaiman, M., Bencherif, M. A. & Mekhtiche, M. A. (2020). Hand Gesture Recognition for Sign Language Using 3DCNN. *IEEE Access*, *8*, 79491-79509 (vid. pág. 15).

Almarales-Wilson, Y. (2021). Aplicación Androide para favorecer la comunicación en Lengua de Señas Cubana. *Evento "Pedagogía 2021"* (vid. págs. 17, 18).

Almeida, A. & Bilbao, A. (2018). *Spanish 3B words Word2Vec Embeddings* (Ver. 1.0). Zenodo. https://doi.org/10.5281/zenodo.1410403. (Vid. pág. 21)

Aneja, D., McDuff, D. & Shah, S. (2019). A high-fidelity open embodied avatar with lip syncing and expression capabilities. *2019 International Conference on Multimodal Interaction*, 69-73 (vid. pág. 14).

CERN. (2014). http://cdsweb.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974627?ln=en. (Vid. pág. 24)

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. *Proceedings of the 43rd annual meeting of the association for computational linguistics (acl'05)*, 263-270 (vid. pág. 13).

Everson, M., Slevinski, S. & Sutton, V. (2013). Proposal for encoding Sutton SignWriting in the UCS. https://www.unicode.org/L2/L2012/12321-n4342-signwriting.pdf (vid. pág. 5)

Facebook. (2018). https://developers.facebook.com/blog/post/301. (Vid. pág. 24)

Finn, C., Goodfellow, I. J. & Levine, S. (2016). Unsupervised Learning for Physical Interaction through Video Prediction. *ArXiv*, *abs/1605.07157* (vid. pág. 16).

Garje, G. V. & Kharate, G. (2013). Survey of machine translation systems in India. *International Journal on Natural Language Computing (IJNLC) Vol*, *2*, 47-67 (vid. pág. 13).

Gehlot, A., Sharma, V., Singh, S. P. & Kumar, A. (2015). Hindi to English transfer based machine translation system. *arXiv preprint arXiv:1507.02012* (vid. pág. 12).

Google. (2020). https://mediapipe.dev/. (Vid. pág. 17)

Graves, A., Fernández, S. & Schmidhuber, J. (2007). Multi-dimensional Recurrent Neural Networks. *International Conference on Artificial Neural Networks* (vid. pág. 16).

Gutiérrez-González, L. (2021). Plataforma y modelos para la interpretación automática de la Lengua de Señas Cubana. https://github.com/leynier/computer-science-bachelors-thesis (vid. págs. 2, 16, 17, 19, 20, 38)

Huenerfauth, M. (2006). *Generating American Sign Language classifier predicates for English-to-ASL machine translation* (Tesis doctoral). Citeseer. (Vid. pág. 15).

Hwang, E. J., Kim, J.-H. & Park, J. C. (2021). Non-Autoregressive Sign Language Production with Gaussian Space. *The 32nd British Machine Vision Conference (BMVC 21)* (vid. pág. 14).

Jiang, T., Camgoz, N. C. & Bowden, R. (2021). Skeletor: Skeletal Transformers for Robust Body-Pose Estimation. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 3389-3397 (vid. pág. 15).

Kacorri, H., Huenerfauth, M., Ebling, S., Patel, K., Menzies, K. & Willard, M. (2017). Regression analysis of demographic and technology-experience factors influencing acceptance of sign language animation. *ACM Transactions on Accessible Computing (TACCESS)*, *10*(1), 1-33 (vid. pág. 15).

Kang, S.-H. & Gratch, J. (2010). The effect of avatar realism of virtual humans on self-disclosure in anonymous social interactions. *CHI'10 Extended Abstracts on Human Factors in Computing Systems* (pp. 3781-3786). (Vid. pág. 14).

Klueva, N. (2007). Semantics in Machine Translation. *WDS'07 Proceedings of Contributed Papers, Part I*, 141-144 (vid. pág. 12).

Knight, K. & Luk, S. K. (1994). Building a large-scale knowledge base for machine translation. *AAAI*, *94*, 773-778 (vid. pág. 11).

Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press. (Vid. pág. 12).

Koehn, P. (2017). Neural machine translation. *arXiv preprint arXiv:1709.07809* (vid. pág. 13).

Koehn, P., Och, F. J. & Marcu, D. (2003). *Statistical phrase-based translation* (inf. téc.). UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST. (Vid. pág. 12).

Kozik, K. (2019). https://www.hrw.org/news/2019/09/23/without-sign-language-deaf-people-are-not-equal#. (Vid. pág. 1)

Latoschik, M. E., Roth, D., Gall, D., Achenbach, J., Waltemate, T. & Botsch, M. (2017). The effect of avatar realism in immersive social virtual realities. *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, 1-10 (vid. pág. 14).

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, *86*, 2278-2324 (vid. pág. 15).

Lu, P. & Huenerfauth, M. (2011). Data-Driven Synthesis of Spatially Inflected Verbs for American Sign Language Animation. *ACM Trans. Access. Comput.*, *4*, 4:1-4:29 (vid. pág. 15).

McDonald, J. C., Wolfe, R. J., Schnepp, J., Hochgesang, J. A., Jamrozik, D. G., Stumbo, M., Berke, L., Bialek, M. & Thomas, F. (2015). An automated technique for real-time production of lifelike animations of American Sign Language. *Universal Access in the Information Society*, *15*, 551-566 (vid. pág. 15).

Mitra, S. & Acharya, T. (2007). Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *37*(3), 311-324 (vid. pág. 16).

Nguyen, L. T., Schicktanz, F., Stankowski, A. & Avramidis, E. (2021). Automatic generation of a 3D sign language avatar on AR glasses given 2D videos of human signers. *Proceedings of the 1st International Workshop on Automatic Translation for Signed and Spoken Languages (AT4SSL)*, 71-81 (vid. pág. 14).

Numpy. (2012). https://numpy.org/about/. (Vid. pág. 24)

Okpor, M. (2014). Machine translation approaches: issues and challenges. *International Journal of Computer Science Issues (IJCSI)*, *11*(5), 159 (vid. pág. 12).

Proceedings of the 7th International Conference on Computer and Communications Management. (2019). *Proceedings of the 7th International Conference on Computer and Communications Management* (vid. pág. 15).

PSF. (2020). https://wiki.python.org/moin/OrganizationsUsingPython. (Vid. pág. 24)

PSF. (2021a). https://www.python.org/about/quotes/. (Vid. pág. 23)

PSF. (2021b). https://www.python.org/about/success/usa/. (Vid. pág. 24)

PSF. (2022). https://www.python.org/doc/essays/blurb/. (Vid. pág. 23)

Řehůřek, R. & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora [http://is.muni.cz/publication/884893/en]. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45-50 (vid. pág. 25).

Sandler, W. (2012). Dedicated gestures and the emergence of sign language. *Gesture*, *12*(3), 265-307 (vid. pág. 16).

Saragih, J. M., Lucey, S. & Cohn, J. F. (2011). Real-time avatar animation from a single image. *2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG)*, 117-124 (vid. pág. 14).

Saunders, B., Camgoz, N. C. & Bowden, R. (2020). Progressive Transformers for End-to-End Sign Language Production. *Proceedings of the European Conference on Computer Vision (ECCV)* (vid. pág. 15).

Sharma, S. & Kumar, K. (2021). ASL-3DCNN: American sign language recognition technique using 3-D convolutional neural networks. *Multim. Tools Appl.*, *80*, 26319-26331 (vid. pág. 15).

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K. & Woo, W.-c. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *NIPS* (vid. pág. 16).

Stoll, S., Camgoz, N. C., Hadfield, S. & Bowden, R. (2018). Sign Language Production using Neural Machine Translation and Generative Adversarial Networks. *British Machine Vision Conference (BMVC)* (vid. págs. 14, 15).

Sutskever, I., Vinyals, O. & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 3104-3112 (vid. pág. 13).

Thiessen, S. (2011). *A Grammar of SignWriting* (M.A thesis). (Vid. pág. 5).

Williams, P., Sennrich, R., Post, M. & Koehn, P. (2016). Syntax-based statistical machine translation. *Synthesis Lectures on Human Language Technologies*, *9*(4), 1-208 (vid. pág. 13).

Zelinka, J. & Kanis, J. (2020). Neural Sign Language Synthesis: Words Are Our Glosses. *The IEEE Winter Conference on Applications of Computer Vision (WACV)* (vid. pág. 15).

Zhu, C., Yu, H., Cheng, S. & Luo, W. (2020). Language-aware interlingua for multilingual neural machine translation. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 1650-1655 (vid. pág. 12).