UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de un chatbot a través de reconocimiento de voz en tiempo real entre el usuario y el rostro animatrónico de la Universidad del Valle de Guatemala

Trabajo de graduación presentado por Daniel Eduardo Fuentes Oajaca
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

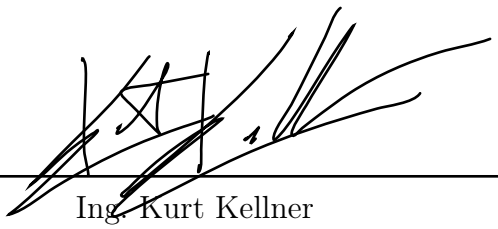
Implementación de un chatbot a través de reconocimiento de voz en tiempo real entre el usuario y el rostro animatrónico de la Universidad del Valle de Guatemala

Trabajo de graduación presentado por Daniel Eduardo Fuentes Oajaca
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

Vo.Bo.:

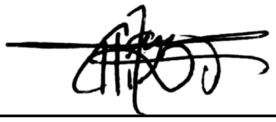
(f) 

Ing. Kurt Kellner

Tribunal Examinador:

(f) 

Ing. Kurt Kellner

(f) 

MSc. Carlos Esquit

(f) 

MAEB. Pablo Mazariegos

Fecha de aprobación: Guatemala, 29 de Junio de 2022.

Prefacio

En este trabajo dejó plasmada la combinación de una gran variedad de conocimientos adquiridos a lo largo de mi carrera. Trabajar en el rostro animatrónico es un gran reto y poner en práctica todo lo que he aprendido hace de este trabajo algo muy interesante. Para mí la mejor parte de un proyecto es la innovación y dotar a este proyecto un enfoque más moderno es de las cosas que más he disfrutado.

Agradezco el apoyo de las personas que estuvieron conmigo a lo largo de mi carrera universitaria. Especialmente agradezco a mis padres, Eduardo Fuentes y Vanessa Oajaca, que me brindaron el apoyo económico para desarrollar al máximo mis habilidades durante mi carrera universitaria. A Jorge Lorenzana, por brindarme amistad, apoyo y consejo incondicional y en todo momento. A mi asesor, el ingeniero Kurt Kellner, por guiarme no solo en el proyecto sino a lo largo de mi carrera despertando en mi el gusto por la innovación y creación. A Bárbara Uzcátegui, por apoyo moral y emocional, además de brindarme contenido y experiencia en diseño para la interfaz gráfica del proyecto. A Gabriela Porras, quien me enseñó el valor de luchar por lo que deseas y de apreciar lo que se tiene. Finalmente, a la Universidad del Valle de Guatemala por darme la oportunidad de desarrollar una carrera profesional y brindarme el material necesario para ello.

Dedico este trabajo a mi persona. Porque aunque hubo tropiezos, tuve la fuerza para volver a levantarme. Por haber aprendido y superado todos los obstáculos durante el viaje y por haber demostrado toda la preparación que he adquirido a lo largo de mi carrera.

Índice

Lista de figuras

Lista de cuadros

Resumen

El siguiente trabajo de graduación tiene como objetivo principal el desarrollo e implementación de un sistema de interacción verbal en tiempo real entre un usuario y el rostro animatrónico de la Universidad del Valle de Guatemala. Luego de un trade study, se seleccionó TensorFlow, específicamente la librería de Keras, para el desarrollo de un modelo capaz de reconocer y clasificar una entrada de texto basandose en la temática de la oración. Se implementa un sistema de reconocimiento de voz, un sistema de sintetización de voz para responder y movimientos del rostro al momento de responder.

Para el reconocimiento de voz y conversión a texto se utilizó la librería de Python "`Speech_Recognition`" y se utiliza específicamente la API de Google para este procedimiento, de manera que la consulta y generación del texto reconocido es bastante rápida y ofrece dicho servicio tanto para inglés como español latino.

El desarrollo del modelo de reconocimiento y clasificación de temas constó de varias partes. La primera es la estructuración y desarrollo de una base de datos que funje como Dataset para el entrenamiento del modelo. Esta se desarrolló en formato JSON y cuenta con una estructura tipo diccionario, de modo que resulta sencillo acceder a la información. Cada diccionario cuenta con un identificador; patrones, los cuales se usarán para entrenar al modelo y respuestas para dar al usuario.

Debido a la complejidad de la tarea, es necesario procesar los datos tanto para el entrenamiento del modelo como para recibir los datos del usuario. Por ello, se usaron técnicas de Natural Language Processing para procesar dichos datos; una de ellas fue la lematización de palabras y una forma inteligente de entrenamiento del modelo a base de listas codificadas. Con ello, se diseña el modelo de una red neuronal entrenada a partir de los datos encontrados en la base de datos para reconocer y clasificar temáticas de conversación. Se utilizaron técnicas, como algoritmos de optimización, para mejorar los resultados del modelo y evitar overfitting. Se desarrolló una serie de funciones capaz de implementar el modelo para poder ser utilizado en algún proceso.

Finalmente, se desarrolló por medio de la librería de Kivy una interfaz gráfica capaz de mostrar al operador el flujo de la conversación que existirá entre el usuario y el animatrónico. Por medio de un proceso de diseño, se obtiene un modelo atractivo y con base en este se

desarrolla el apartado gráfico del proyecto. Para poder implementar el modelo en la interfaz, se hace uso de la técnica "Multithreading" que nos permite ejecutar una parte del código en paralelo con la ejecución de la interfaz gráfica, de modo que ambas partes comparten información, pero sus procesos no se ven interrumpidos entre sí. Se hace uso de las funciones desarrolladas para la implementación del modelo y se desarrolla un bucle, el cual siempre escuchando activamente todo el tiempo, pero no responderá si no se le saluda primero, de modo que el bot no responderá a personas que no le están dirigiendo la palabra a él. Cada vez que el bot comience a hablar mandará una señal por medio de comunicación serial al microcontrolador para que este mueva la boca; este proceso se hace de manera muy exacta y síncrona, por lo que la sincronización entre el tiempo de habla y el movimiento de la boca es bastante atractiva.

Abstract

The main objective of this graduation project is to develop and implement a real-time verbal interaction system between the user and the animatronic face of the Universidad del Valle de Guatemala. After a trade study, I chose TensorFlow and Keras as the best environment to develop a model capable of recognizing and determining the topic of a sentence based on its words. I also implemented a voice recognition, a text-to-speech, and a mouth movement system to interact with the user.

For the speech-to-text recognition, I used the "`Speech_Recognition`" library, specifically the Google API, for a fast query system and good Spanish voice recognition.

The model development counts on many parts. The first one is the structuration and development of the database, which is used as the dataset to train the model. It was developed in JSON format with a dictionary structure to ease access to the information. Each of the dictionaries has a tag, patterns to train the model, and a variety of answers to the conversation.

In order to facilitate the training process and simplify the user data, I used Natural Language Processing techniques such as lematizing and a smart way to train the model with coded lists. This results in a neural network capable of recognizing conversation topics based on the database patterns. I also used optimization algorithms to perform better results and avoid overfitting. Finally, I created functions to implement the model across the project.

Finally, I used the Kivy library to develop a graphic user interface to show the flow of the conversation between the user and the bot. Through the design thinking process, I obtained a mockup to guide the visual aspect of the program. For the implementation of the model in the user interface, I used multithreading techniques to execute the user interface and the recognizing and answering process in parallel. This way, both processes share information without interrupting each other. The recognizing and answering process is made by a loop that constantly hears every voice request, but it doesn't answer until it recognize that you're talking to him.

CAPÍTULO 1

Introducción

La parte final del proceso de diseño es la iteración. Siempre existe algún aspecto a mejorar y siempre se puede innovar en cualquier proyecto. Por ello, en este documento se describe detalladamente la modificación del rostro animatrónico de la Universidad del Valle de Guatemala con el fin de mejorar la estructura e implementar un sistema de preguntas y respuestas por medio de Machine Learning con el fin de "humanizar" la interacción que existe entre el sistema y el usuario.

El desarrollo de proyectos que impliquen el uso de Machine Learning es algo muy popular y que está al alcance de cualquiera en la actualidad; por ello, existen muchísimas herramientas para el desarrollo e implementación de Machine Learning que facilitan el proceso y se especializan en una gran cantidad de aspectos y áreas abarcadas por la misma técnica. Debido a esto, es crucial para el proyecto de desarrollar un Trade Study para definir qué herramientas se adecúan y especializan en las funciones de interés del proyecto para facilitar su desarrollo.

Ya con las herramientas para el desarrollo de la red neuronal, es crucial separar el proyecto en módulos. Por ello, este se divide en:

- Módulo de entrenamiento de la red neuronal.
- Módulo de predicción y generación de respuesta.
- Módulo de reconocimiento de voz.
- Módulo de habla.
- Módulo de estructura y organización gráfica.
- Módulo multiproceso.
- Módulo de movimiento facial.

CAPÍTULO 2

Antecedentes

El primer programa diseñado para responder de forma automática y coherente lo que se le pregunta fue inventado en Estados Unidos en 1966 y fue llamado Eliza. Su creador, Joseph Weizenbaum, fue un profesor de informática del Instituto Tecnológico de Massachusetts, y lo creó a modo de burla de las preguntas que hacían los psicoterapeutas que seguían los lineamientos del psicólogo Carl Rogers, creador de la llamada "terapia centrada en el cliente". El objetivo era que Eliza conversara con el usuario de modo que pareciese como si estuviese escuchándolo y empatizando con él. Este era un modo de demostrar lo superficial que puede ser la conversación entre el hombre y la máquina. [1]

El algoritmo de Eliza se basa en el reconocimiento de palabras clave al momento de recibir la información para luego hacer preguntas referentes al tema. También tenía almacenadas frases hechas sobre una variedad de temas, que utilizaba al reconocer palabras clave, así como frases empáticas y de continuidad o daba vuelta la frase que el usuario le decía y la repetía, transformándola en una pregunta. En el peor de los casos, si Eliza no obtiene palabras clave se apoya en frases como "¿Por qué dices eso?" o "¿Estás seguro?" para darle continuidad y fluidez a la conversación.

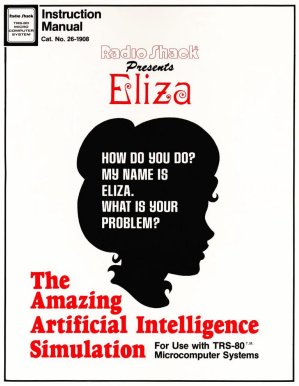


Figura 1: Portada del instructivo de Eliza. [2]

Sin embargo, Eliza no es capaz de retener información o aprender de la conversación. Esto nos deja ver el enorme salto que ha existido con el paso de los años en el desarrollo de inteligencia artificial pues, actualmente, existen algoritmos capaces de aprender muchísima información sobre el usuario y generar una conversación completamente fluida y "casi humana". [3]

Actualmente, Google Meena se considera como uno de los chatbots más avanzados que existen. Se trata de un chatbot potenciado mediante inteligencia artificial que se ha integrado en una red neuronal compleja que es capaz, en teoría, de mantener una conversación sobre cualquier cosa, y de hacerlo al nivel de un ser humano. Dicha red está formada por 2.600 millones de parámetros y fue entrenada con 40.000 millones de palabras y 341 GB de datos en texto, incluyendo conversaciones extraídas de redes sociales, una variación del Google Translate que trabaja como una red neuronal que compara palabras en un párrafo entre sí para comprender la relación entre ellas. Podemos ver que el robot está capacitado para mantener una conversación sobre cualquier tema con cualquier persona debido al avanzado sistema de inteligencia artificial que se desarrolló para este y la amplia base de datos que tiene para generar respuestas; definitivamente se tiene una capacidad casi humana de conversar. [4]

El desarrollo de animatrónicos va muy de la mano con el desarrollo de autómatas, con la necesidad del humano de crear algo semejante a la vida. En 1961 Walt Disney acuña el término audio-animatrónico y comienza a desarrollar tecnología animatrónica moderna. Desde pájaros animatrónicos en la película de Mary Poppins hasta la primera figura animatrónica de una persona (Abraham Lincoln), Disney ha sido un pionero en este campo con una visión futurista del entretenimiento. En la actualidad, el uso de animatrónicos está muy ligado al entretenimiento y al campo de robótica, pues las técnicas modernas de robótica permiten generar modelos más cercanos al movimiento y acción de cualquier organismo. Generalmente, los animatrónicos de entretenimiento se enfocan de mayor forma en el aspecto estético y en la naturaleza de los movimientos. [5]

Finalmente, en años pasados se han trabajado distintos modelos en la Universidad del Valle de Guatemala y se ha buscado el desarrollo de un rostro y mano animatrónicos con el fin de mostrar en ferias y otras exposiciones las capacidades y el alcance de la carrera de Ingeniería Mecatrónica. En el año 2018 se desarrolla un prototipo con ojos y dientes que contaba con diez grados de libertad y se construyó con piezas cortadas por láser y algunas impresas en 3D. En este año, también se trabajó en el desarrollo de un sistema interactivo de reconocimiento de comandos por voz con análisis de Emociones para Robot Animatrónico. En el año 2020 el, en ese entonces, estudiante Christopher Jenatz Melgar rediseñó, desarrolló y construyó un rostro animatrónico más complejo, dándole un enfoque muy grande al diseño y desarrollando una pequeña aplicación para el uso del proyecto, el cual cuenta con dieciocho grados de libertad.

CAPÍTULO 3

Justificación

Los objetivos, general y específicos, del proyecto surgen de la necesidad de poder desarrollar algo más complejo y de pulir y aprovechar el potencial del hardware con el que se cuenta del tema de Animatrónicos en la Universidad del Valle de Guatemala, pues este proyecto representará no solo algo entretenido para las ferias o exposiciones en las que participará, sino también para demostrar las capacidades y el alcance que se puede tener con los conocimientos adquiridos en la carrera de Ingeniería Mecatrónica. La tecnología avanza constantemente a pasos agigantados y la carrera exige un constante aprendizaje y actualización, por lo que el hecho de generar una interacción más humana entre el usuario y el proyecto representa un avance en las capacidades y complejidad actuales del mismo. Convertir el reconocimiento de comandos a una conversación con un chatbot representa un gran salto en el desarrollo del software del proyecto. De igual forma, es muy importante pulir no solo el movimiento suavizado del rostro sino también mejorar la parte estética y funcional del sistema, reemplazar o mejorar partes del diseño original es parte del proceso de diseño (el cual es iterativo) y ayudará a presentar algo de muchísima calidad en las ferias, el cual es un aspecto de suma importancia para demostrar el alto nivel en el que se encuentra la carrera y sus estudiantes.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar e implementar un sistema de interacción verbal en tiempo real entre un usuario y el rostro animatrónico de la Universidad del Valle de Guatemala.

4.2. Objetivos específicos

- Implementar un sistema de reconocimiento de voz y *Speech-to-Text* con la herramienta seleccionada.
- Crear un sistema de chatbot con un modelo basado en recuperación mediante machine learning.
- Implementar una interfaz gráfica que funja de puente entre el hardware y los sensores.
- Generar una base de datos con información amplia sobre la universidad para que el sistema de chatbot pueda generar respuesta.
- Implementar un sistema de sintetización de voz para las respuestas del robot.
- Implementar un sistema de movimientos expresivos del rostro animatrónico al hablar.

CAPÍTULO 5

Alcance

El proyecto del rostro animatrónico tiene un enfoque claro en la innovación y el alcance de este trabajo abarca la investigación e implementación de redes neuronales para el desarrollo de un sistema de reconocimiento de frases capaz de recibir datos del usuario por medio de reconocimiento de voz y generar una respuesta. Para ello, se requiere entrenamiento y validación de la red neuronal mediante una base de datos con frases que esta pueda reconocer durante la conversación con el usuario tomando en cuenta el contexto en el que se encuentra. Por ello, también se le dota de respuestas variadas a los temas que esta pueda abarcar.

También abarcará la investigación, proceso de diseño y desarrollo de una interfaz gráfica capaz de mostrar al operador del sistema el flujo de la conversación entre el usuario y el modelo de forma sencilla y poder comunicar esta con el microcontrolador para representar los movimientos de la boca del rostro mientras habla.

CAPÍTULO 6

Marco teórico

6.1. Comunicación

Según la Fundación de la Universidad Autónoma de Madrid, la comunicación es el proceso mediante el cual transmitimos y recibimos datos, ideas, opiniones y actitudes para lograr comprensión y acción. Este concepto es importante ya que nos ayuda a entender de forma básica en qué consiste este proceso y cómo es posible implementarlo de forma activa para crear una interacción más orgánica entre el usuario y el animatrónico, pues el objetivo es generar cierta semejanza con la interacción humana. [6]

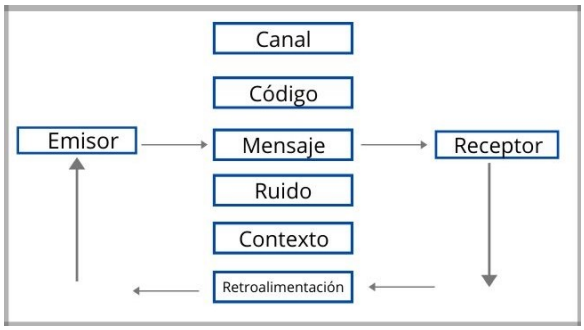


Figura 2: Estructura básica de la comunicación.[7]

6.2. Chatbot

Un chatbot es una pieza de software con inteligencia artificial en un dispositivo, aplicación, sitio web u otras redes que intentan medir las necesidades de los consumidores y luego ayudarlos a realizar una tarea particular. En general, existen dos variantes de chatbots: basado en reglas y autoaprendizaje. El primero responde preguntas basadas en algunas reglas que previamente han sido entrenadas. Los de autoaprendizaje utilizan técnicas de aprendi-

zaje automático y son mucho más eficientes; existen de dos tipos: basados en recuperación y generativos. En los modelos basados en recuperación, estos bots utilizan heurística para seleccionar de una base de datos una respuesta, por lo que su conocimiento es limitado y controlable; utiliza el mensaje y el contexto de la conversación para seleccionar la mejor respuesta. En los modelos generativos, el bot es capaz de tomar palabra por palabra y generar una respuesta que no siempre viene del conjunto de datos previos. Esto los hace más inteligentes y con una capacidad amplia de aprendizaje. [8]

6.3. Redes neuronales

Una red neuronal es un modelo simplificado que emula el modo en que el cerebro humano procesa la información: Funciona simultaneando un número elevado de unidades de procesamiento interconectadas que parecen versiones abstractas de neuronas. Estas unidades de procesamiento de información se organizan en capas de las cuales normalmente existen una de entrada, una o varias ocultas y una de salida. Existe una gran variedad de capas con funciones y tipos de activación distintas, así como una gran variedad de modelos que estructuran de distinta forma las capas. [9]

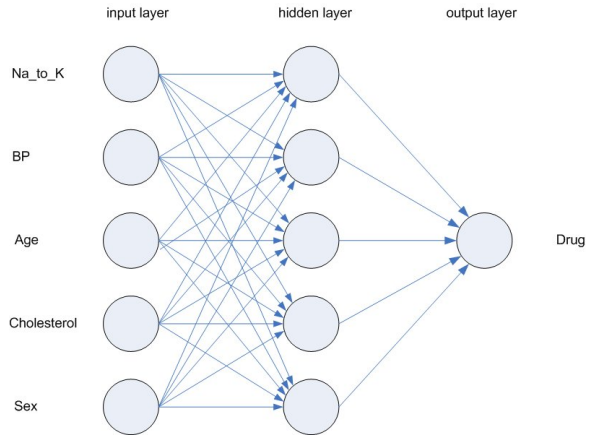


Figura 3: Ejemplo gráfico de una red neuronal. [9]

El esquema de desarrollo de redes neuronales asume que existe un modelo por encontrar el cual es demasiado complejo para identificar o describir a partir de principios básicos matemáticos. Logra encontrar el modelo con base en la información o "data" disponible para su entrenamiento.

Podemos pensar que muchos problemas no requieren de machine learning puesto que existen muchos escenarios que son fácilmente descritos por leyes o fundamentos físicos o matemáticos y encontrar modelos que se comporten de esa forma resulta una tarea sencilla. Sin embargo, resulta muy útil implementarlo en cualquier diseño basado en heurística; que significa "hallar" o "inventar". Existe una gran cantidad de aplicaciones para machine learning como el reconocimiento de escritura o de caracteres, reconocimiento facial, procesamiento natural de lenguaje, traducción, sistemas de recomendaciones personales, visión por computadora, etc. Las redes neuronales se entrenan a base de Datasets, de modo que necesitan información previa para poder generar predicciones con base en el modelo seleccionado. El

proceso de esta información es un trabajo complejo, pues es necesario representar de manera inteligente la información para que el modelo capte la tendencia. En ciertas situaciones, Machine Learning corresponde a inferencia estadística.

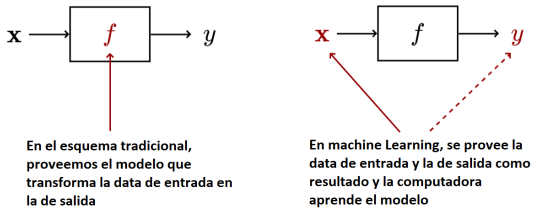


Figura 4: Ejemplo gráfico del esquema de Machine Learning.

6.3.1. Tipos de Machine Learning

Aprendizaje supervisado

En este tipo de machine learning contamos con la data de entrada, en la cual cada uno de los elementos representa una medición u observación de algún fenómento; y la data de salida, la cual nos muestra el resultado de la "transformación" o la "respuesta" concebida a los datos de entrada (es la variable dependiente). A este conjunto de data se le conoce como data o set de entrenamiento y no debemos descartar la idea de que este puede presentar ruido.

El objetivo principal de este tipo de aprendizaje es generalizar la relación que existe entre la data de entrada y la de salida para poder predecir la salida asociada a una data de entrada que no haya sido utilizada para el entrenamiento. Cabe mencionar que este tipo de aprendizaje intenta resolver problemas de clasificación y de regresión. Los problemas de regresión se caracterizan en que la variable de respuesta es cuantitativa y puede ser flexiblemente determinada por las entradas al modelo; por otro lado, los problemas de clasificación se caracterizan por tener una variable de respuesta cualitativa o categórica y es muy común utilizar como método de clasificación predecir la probabilidad de una observación de pertenecer a cada una de las categorías. [10]

Aprendizaje no supervisado

Este tipo de machine learning cuenta con una data de entrada que no tiene etiquetas asignadas. El objetivo principal de este aprendizaje, a diferencia del supervisado, no se encuentra relacionado con observaciones futuras sino con la idea de comprender la estructura de la data o inferir características sobre la misma y/o su distribución. Algunos de los problemas más comunes de este tipo de aprendizaje son clustering, estimasión de densidad y visualización. El clustering consiste en el agrupamiento de conjunto de objetos no etiquetados para construir subconjuntos de datos. Sirve para segmentar datos en grupos de dimensiones similares con base en características para facilitar el proceso mismo. [11]

Aprendizaje por refuerzo

Las entradas de este tipo de aprendizaje se observan de forma secuencial y, luego de observarlas, se debe tomar una acción. Luego de cada acción, el agente recibe recompensas y el objetivo es determinar una "política" o "razonamiento" que maximice la recompensa a largo plazo. Vemos que no se tiene una "salida correcta" sino una "recompensa para esta salida". Este tipo de aprendizaje sigue en un proceso de investigación fuerte.

6.3.2. Clasificación de redes neuronales según la topología de la red

Red neuronal monocapa - Perceptrón simple

Esta es una red neuronal simple y está compuesta por una sola capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan los cálculos. [12]

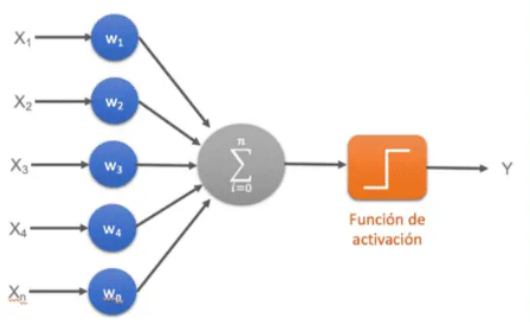


Figura 5: Ejemplo gráfico del una red neuronal monocapa. [12]

Red neuronal multicapa - Perceptrón multicapa

Es una generalización de la red monocapa, pero esta cuenta con un conjunto de capas intermedias entre las capas de entrada y salida, las cuales también podemos llamar "capas ocultas". Esta puede estar total o parcialmente conectada dependiendo del número de conecciones que presente. [12]

Red neuronal convolucional (CNN)

Este tipo de red difiere de la red multicapa por el hecho de que cada neurona no se conecta con todas y cada una de las capas siguientes, sino que con un subgrupo de estas. De esta forma, se reduce el número de neuronas necesarias y la complejidad computacional necesaria para su ejecución. [12]

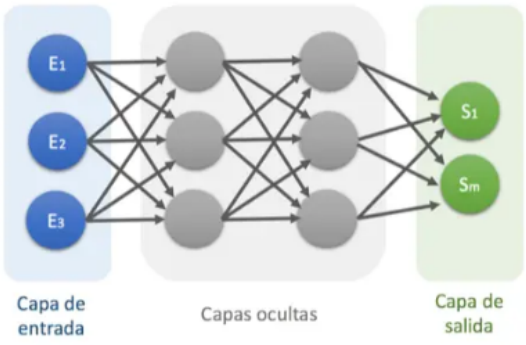


Figura 6: Ejemplo gráfico del una red neuronal multicapa. [12]

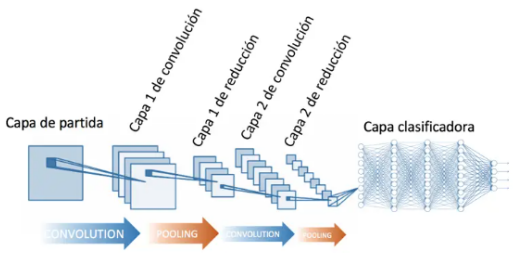


Figura 7: Ejemplo gráfico del una red neuronal convolucional. [12]

Red neuronal recurrente (RNN)

Estas se caracterizan por no tener una estructura de capas, sino que permiten conexiones arbitrarias entre neuronas incluyendo ciclos y, de esta forma, creando temporalidad y permitiendo que la red tenga memoria. Los datos introducidos en la entrada se transforman y circulan por la red incluso en las entradas de los siguientes datos. [12]

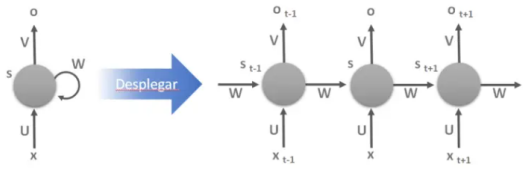


Figura 8: Ejemplo gráfico del una red neuronal recurrente. [12]

Redes de base radial (RBF)

Calculan la salida del modelo en función de la distancia a un punto denominado como centro. Es una combinación lineal de las funciones de activación radiales utilizadas por las neuronas individuales. Cada neurona de la capa posee un caracter local, por lo que cada una se activa en una región diferente en el espacio de patrones de entrada [12]

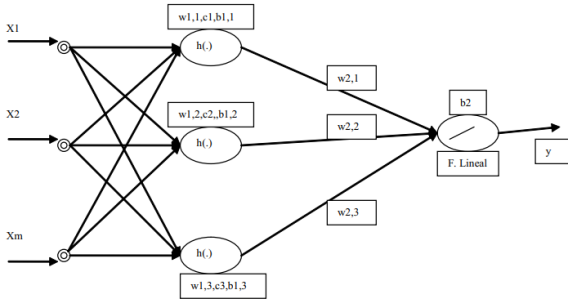


Figura 9: Ejemplo gráfico del una red de base radial. [13]

6.3.3. Consolidación de conceptos

Con la información previa podemos pensar que resulta indispensable elegir. Sabiendo que el desarrollo de una red neuronal capaz de responder al usuario (chatbot) es primordial, podemos ver que convenientemente el tipo de Machine Learning apropiado para la tarea es el aprendizaje supervisado pues la tarea que deseamos realizar es un problema de clasificación y una estructura multicapa sería la adecuada debido a que se tiene una entrada con múltiples datos y se debe comparar con varios patrones de distintas categorías. Podemos concluir entonces que el uso del modelo por el cual estamos interesados involucra implícitamente la solución a un problema de optimización y se denomina como "optimización de pérdida". Las funciones de pérdida se seleccionan según el problema a resolver. Una característica muy importante a tener en cuenta es la función de activación de cada capa, pues esta es la encargada de devolver una salida a partir del valor en la entrada y normalmente se buscan funciones que las derivadas sean simples para minimizar el coste computacional y su idea fundamental es introducir no linealidades al modelo para hacer posible la clasificación. Las funciones de activación más comunes son:

- **Sigmoid:** Transforma los valores introducidos a una escala (0,1), donde los valores altos tienden de forma asintótica a 1 y los bajos de manera asintótica a 0. Esta función satura y mata los gradientes, es lenta en converger, no está centrada en cero y tiene un buen rendimiento en la última capa.

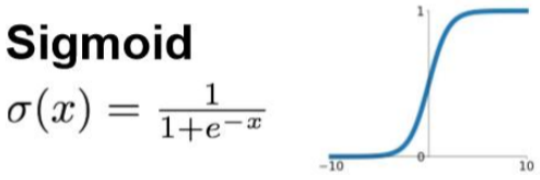


Figura 10: Función de activación Sigmoid.

- **Tanh:** Tangente hiperbólica, transforma los valores introducidos a una escala (-1,1), donde los valores altos tienden de forma asintótica a 1 y los bajos de manera asintótica a -1. Es muy similar a la función Sigmoid, pero esta sí se encuentra centrada en 0 y se

utiliza para decidir entre una opción y la contraria y tiene buen rendimiento en redes recurrentes.

Figura 11: Función de activación Tanh.

- **ReLU:** Transforma los valores introducidos anulando los valores negativos y dejando los positivos sin modificar. Esto implica que solo se activa si los valores son positivos, no se encuentra acotada por ningún rango, funciona muy bien con imágenes y tiene buen rendimiento en redes convolucionales.

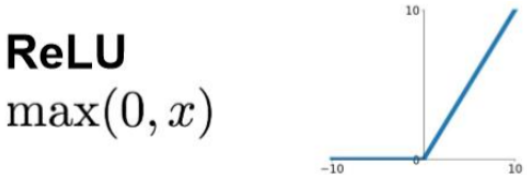


Figura 12: Función de activación ReLU.

- **Leaky ReLU:** Transforma los valores introducidos multiplicando los negativos por un coeficiente rectificativo y dejando los positivos sin modificar. Es muy similar a la función ReLU, incluyendo en características y rendimiento.

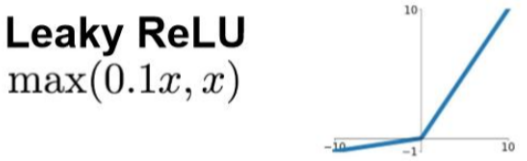


Figura 13: Función de activación Leaky ReLU.

- **Softmax:** Transforma las salidas a una representación en forma de probabilidades, por lo que la suma de las probabilidades de las salidas es 1. Normalmente se utiliza cuando buscamos predecir por medio de probabilidades y para normalizar tipo multiclase; tiene un buen rendimiento en las últimas capas.

[14]

Softmax

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

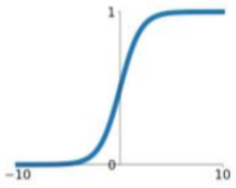


Figura 14: Función de activación Softmax.

6.3.4. Funciones de pérdida

Como ya se mencionó anteriormente, el uso del modelo involucra la solución a un problema de optimización y por ello es importante escoger la función de pérdida que se adecúe al modelo según el problema que se desea resolver con el mismo. Esta función nos dirá durante el proceso de entrenamiento de la red neuronal cuán lejos está de un momento dado, su salida y el resultado que se considera correcto. Debido a que los algoritmos de aprendizaje calculan la derivada de la función de pérdida, esta debe ser continua y derivable. Existe una gran cantidad de funciones de pérdida, algunas de ellas que resultan bastante útiles son:

- **Mean Square Error (MSE):** Esta función calcula la distancia "geométrica" al cuadrado al valor objetivo. Existe una variante a esta que calcula el valor absoluto de dicha distancia (Error absoluto). Se usa en problemas de regresión a valores arbitrarios y con una última capa sin función de activación.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Figura 15: Función de pérdida MSE.

- **Entropía cruzada categórica (Categorical Cross Entropy):** Es una medida de la distancia entre distribuciones de probabilidad. Suele ser adecuada para modelos con una salida que representa una probabilidad cuando se hace una clasificación categórica como con la función de activación Softmax.

$$L_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Figura 16: Función de pérdida CCE.

- **Entropía cruzada binaria (Binary Cross Entropy):** Es una variante de la entropía cruzada categórica pero con clasificación binaria, por lo que la función de activación utilizada sería Sigmoid.

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

Figura 17: Función de pérdida BCE.

- **Entropía cruzada categórica dispersa (Sparse Categorical Cross Entropy):** Esta función, también variante de la CCE, se suele usar en aplicaciones que trabajan con número enteros.

[15]

Algo muy importante a tomar en cuenta para la estructuración de redes neuronales es el teorema de aproximación universal el cual establece que, con suficientes nodos, la red neuronal de una capa es capaz de aproximar cualquier mapa continuo. Cabe aclarar que el número de nodos cambia dependiendo de la aplicación. La idea de emplear múltiples capas ocultas con múltiples nodos ocultos en vez de aumentar el número de nodos de una única capa se conoce como "deep learning". En vez de requerir un proceso de caracterización de la data antes de introducirla al modelo, los sistemas de deep learning efectúan por sí mismos la extracción de los rasgos a partir de la data.

El proceso de entrenamiento también puede presentar problemas debido a los datos provistos para el entrenamiento. Estos problemas pueden ser Underfitting u Overfitting y ocurre cuando el modelo generaliza el conocimiento que se pretende que adquiera. El underfitting se presenta cuando los datos de entrenamiento no son suficientes como muestra o son tan específicos que no representan al resto de la población. El overfitting se presenta cuando al entrenar el modelo se presentan características tan específicas que el modelo se vuelve estricto y presenta fallos al momento de intentar reconocer alguna muestra que presente características un poco diferentes. [16]

Existe una gran variedad de métodos para evitar estos problemas: cantidad mínima de muestras, ajuste de parámetros, ajuste de dimensiones o incluso algoritmos de optimización. Las herramientas más prácticas para ello son el ajuste de parámetros y la implementación de algoritmos de optimización. El ajuste de parámetros consiste en un proceso iterativo de cambio de parámetros de entrenamiento (tiempo, iteraciones, densidad de capas, etc.) con el objetivo de encontrar un equilibrio para el proceso de entrenamiento. Por su parte, la implementación de un algoritmo de optimización es un método más controlado y genera buenos resultados. Existen varios algoritmos de optimización pero uno de los más conocidos es el decenso de gradiente estocástico (SGD).

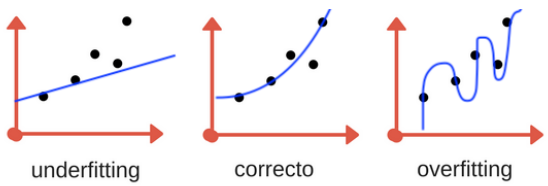


Figura 18: Ejemplo gráfico de los problemas underfitting y overfitting.[16]

6.4. Plataforma de desarrollo

Python se ha convertido en una de las herramientas más populares no solo de programación general sino también en el campo de data science. Es muy común en este campo el uso de MATLAB y R; Python cuenta con librerías para carga de datos, visualización, estadística, procesamiento de lenguage natural y de imágenes y muchas más herramientas gracias a su enorme comunidad y además presenta una interacción directa con el código, por lo que es una opción ideal para el trabajo de machine learning que, fundamentalmente, es un proceso iterativo. [17]

Figura 19: Logo de Python.

CAPÍTULO 7

Diseño 3D

Como se menciona en el capítulo 2, el hardware del proyecto se ha trabajado en años anteriores y el diseño que se utiliza para trabajar es el entregado en el año 2020 por Cristopher Jenatz Melgar, el cual cuenta con dieciocho grados de libertad y su respectiva unidad de alimentación y control de motores. Sin embargo, al inicio del proyecto en el momento en que recibí el hardware, algunas de las piezas de la base, párpados y cejas estaban dañadas y/o no era un diseño adecuado, por lo que la persona que también hizo uso del hardware para su proyecto de graduación y yo nos dimos a la tarea de rediseñar algunas de las piezas e imprimir otras que simplemente estaban rotas. Se hizo un cambio radical de la base y se hizo un ajuste del diseño de los párpados para darle mayor solidez a la parte en que encaja con el eslabón que le transmite el movimiento del servo, pues se encontraba rota y era muy flexible debido a su grosor. Finalmente, se reimprimieron algunas piezas del mecanismo que mueve las cejas para poder montar nuevamente el hardware. Otro de los cambios importantes que se realizó fue el armado de la base de los ojos, pues el diseño anterior era una única pieza sólida. Esto representa un problema porque el método de fabricación con filamento fundido (FFF), resultando en un sólido quebradizo si se le aplica una fuerza paralela al plano en el que fueron impresas las capas. El modelo de la parte inferior no presenta este problema debido a sus dimensiones, pero los "salientes" que sostienen el resto de la estructura sí. Por ello, decidí imprimir la parte inferior como una base con agujeros guías y sin salientes, mientras que el resto de piezas se imprimen estratégicamente con la superficie más grande del sólido como base, de esta forma se evita la facilidad de separación de las capas.

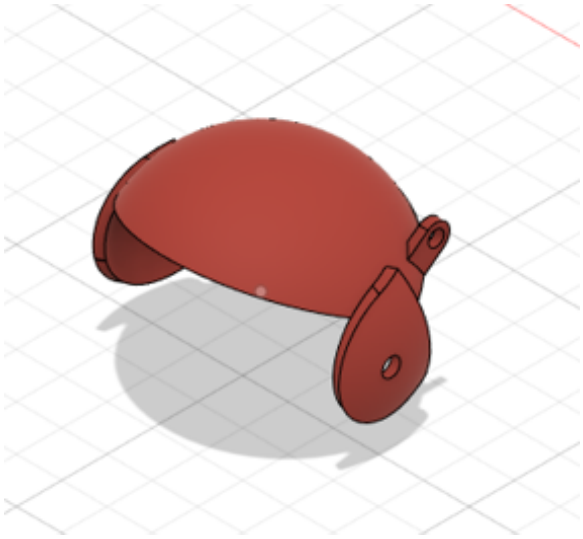


Figura 20: Diseño anterior del párpado.

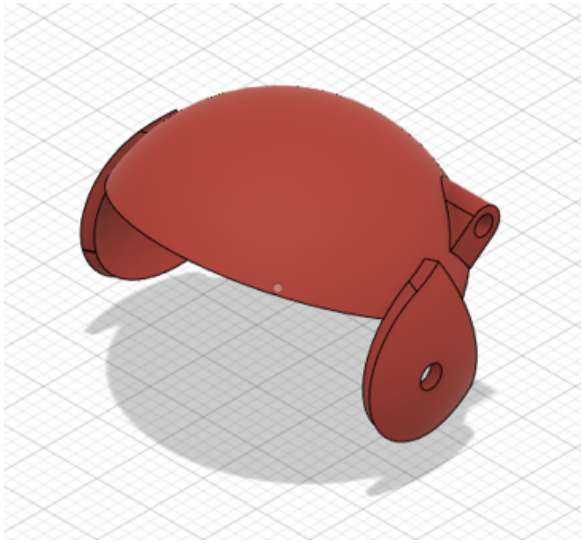


Figura 21: Nuevo diseño del párpado.

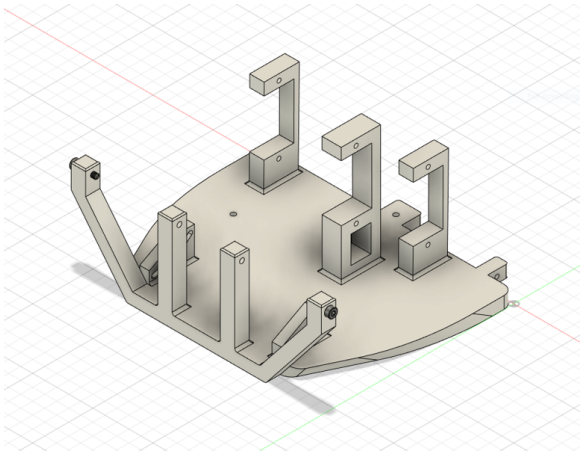


Figura 22: Diseño anterior de la base para los ojos.

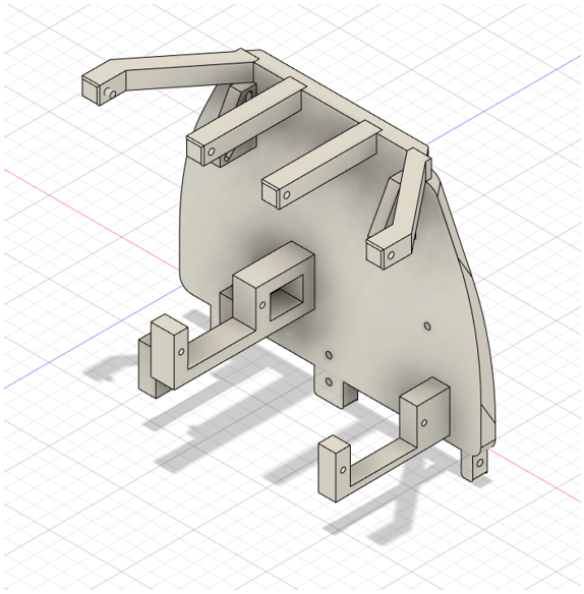


Figura 23: Nuevo diseño de la base para los ojos.

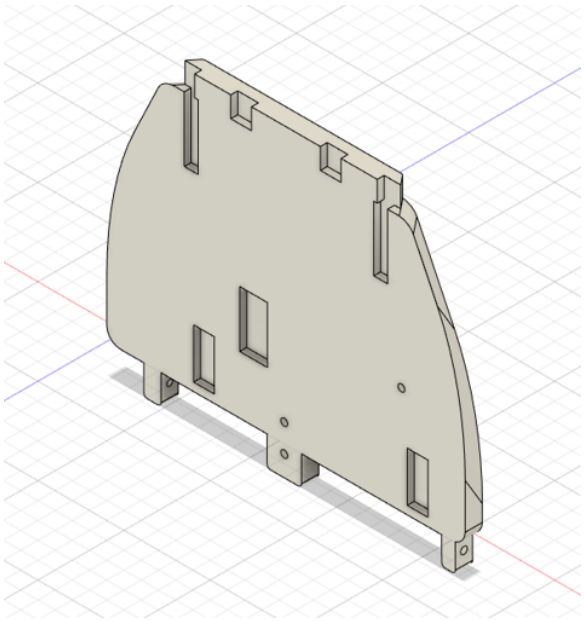


Figura 24: Pieza inferior de la base de los ojos.

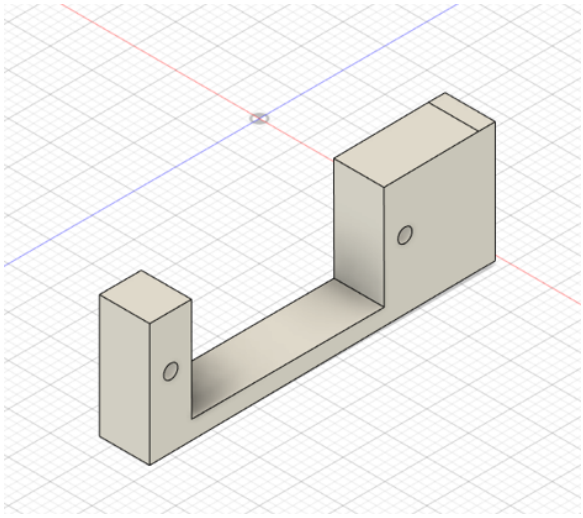


Figura 25: Soporte trasero de servo de la base de los ojos.

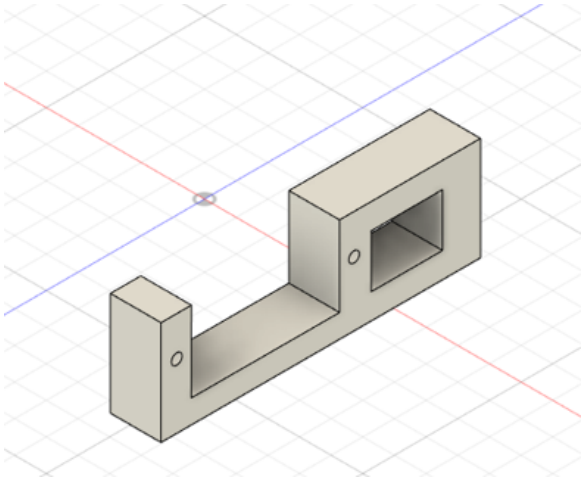


Figura 26: Soporte alto de servo de la base de los ojos.

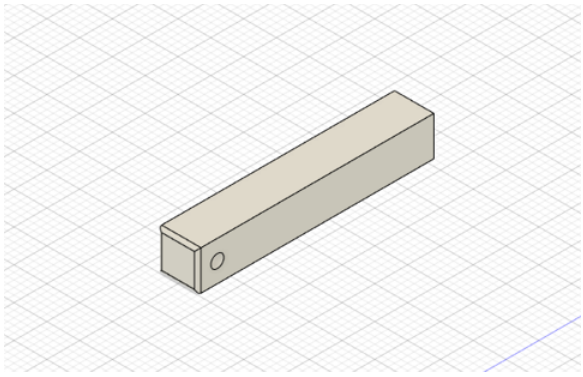


Figura 27: Soporte ocular frontal.

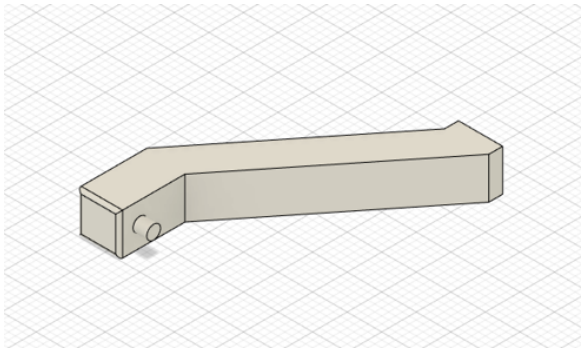


Figura 28: Soporte ocular lateral.

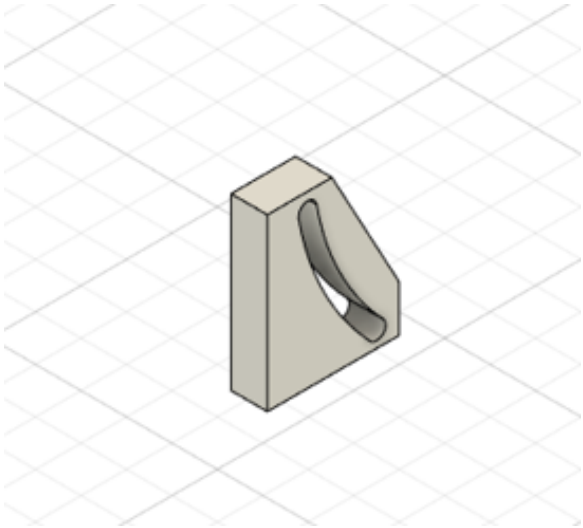


Figura 29: Slider para la base de los ojos.

CAPÍTULO 8

Reconocimiento de voz

Para el reconocimiento de voz, se utilizará el módulo llamado “speech recognition” de Python, pues es el lenguaje de programación donde se desarrolla el resto del proyecto y es una librería muy fácil de utilizar. Primero, es necesario instalar la librería y luego incluirla en el proyecto.

```
with sr.Microphone() as source:
    audio = r.listen(source)
    try:
        text = r.recognize_google(audio, language='es-GT')
```

Figura 30: Implementación del módulo y obtención del texto.

Usar la librería es muy sencillo, con el método “Microphone()” de la librería como fuente lo utilizamos en el la función "listen()"del método “Recognizer()” para obtener la información que viene del micrófono de la computadora. Cabe aclarar que en el momento en el que se llama a esta función se abre el micrófono y comienza a grabar hasta escuchar un silencio.

Finalmente, para convertir la grabación del sonido a texto, utilizaremos el módulo de Google Speech Recognition del método “Recognizer()”. A este le debemos indicar en los parámetros lo que deseamos convertir (el audio previamente obtenido) y el lenguaje en el que se habló (en este caso, español de Guatemala). Esto devolverá una cadena de texto con el mensaje del audio el cual almacenaremos en una variable. Para poder ver el resultado vea el siguiente video **Módulo de reconocimiento de voz**.

```
import speech_recognition as sr
r = sr.Recognizer()
```

Figura 31: Inclusión y renombramiento de la librería Speech recognition.

CAPÍTULO 9

Herramientas para machine learning

9.1. Sickit Learn

Una de las herramientas que podría ser útil para este proceso es Scikit Learn, el cual es un proyecto open source. Es una herramienta simple y eficiente para análisis de predicción de datos; es accesible y se construye a base de varias librerías básicas de Python y cuenta con una basta y comprensible cantidad de documentación para su uso como librería de Python. Es la librería más prominente para machine learning en Python. [18]

Figura 32: Logo de Sickit Learn.

9.2. TensorFlow

Es una librería que facilita tanto la creación como la implementación de modelos de redes neuronales. También cuenta con su propia plataforma y la creación de modelos es sencilla. Ofrece varios niveles de abstracción para cubrir distintas necesidades y permite el entrenamiento y la implementación del modelo de forma fácil, ya sea en servidores, dispositivos

perimetrales o en la web, sin importar el lenguaje o la plataforma que se utilice. Finalmente, TensorFlow cuenta con una API llamada Keras que facilita la creación y entrenamiento de modelos de alto nivel y facilita el aprendizaje automático. Finalmente, TensorFlow está disponible como una librería de Python de facil acceso e instalación y cuenta con muchísima documentación (Incluyendo la API de Keras) y una comunidad muy activa puesto que se utiliza incluso en grandes empresas como Google, Intel, Twitter, entre otras. [19]

Figura 33: Logo de TensorFlow.

9.3. Pytorch

Es una herramienta open source para el trabajo de tensores para la creación de redes neuronales. Una de las características principales de esta librería es la capacidad de cómputo de tensores por medio de GPU (Graphics Processing Unit) lo cual acelera el proceso de entrenamiento. Cuenta con un sistema robusto y complejo. La documentación en su sitio web es bastante completa pero no muy amigable para personas que inician en el desarrollo de herramientas que hacen uso de machine learning. Se podría decir que la robustez y complejidad de su sistema lo hace una herramienta para conocedores y/o profesionales. [20]

Figura 34: Logo de Pytorch.

9.4. XGBoost

Es un algoritmo entrenador de modelos basado en estructura de árbol que usa Boosting gradients para optimizar el proceso, alcanzando así una secuencia de modelo óptima. Lo que hace es entrenar un árbol sobre otro para aprender de las debilidades del último. De esta forma, se garantiza un modelo más robusto. XGBoost es ideal para trabajar con datasets de entrenamiento enormes.[21]

Figura 35: Logo de XGBoost.

Característica	Sickit Learn	TensorFlow	PyTorch	XGBoost
Sintaxis	Simple	Compleja. Puede Simplificarse	Compleja	Simple
Documentación	Mucha. Es muy popular	Mucha. Es muy popular	Mucha.No es muy popular	Media
Datasets	Pequeños	Flexible	Grandes	Grandes
Velocidad	Lento	Rápido	Rápido	Rápido
Especialidad	Machine Learning amateur	Redes neuronales y tensores	Open Source para tensores	Boosting Gradients
Curva de aprendizaje	Para principiantes	Principiantes y profesionales	Profesionales y entusiastas	Conocimiento medio
Lenguaje	Python	C++, Python	C++, Python	C++, Python, R, Perl, ...

Cuadro 1: Trade Study de software para desarrollo de Machine Learning.

El Cuadro 1 presenta las características más importantes de cada uno de los software candidatos para el desarrollo del proyecto. Existen algunas características que puedes significar una ventaja o desventaja para el proyecto y sus objetivos. Por ello, es necesario asignar un valor numérico entre 1 y 5 a cada característica según su prioridad y de qué forma se adecúan mejor al proyecto para poder tomar una decisión.

Característica	Sickit Learn	TensorFlow	PyTorch	XGBoost
Sintaxis	4	4	2	4
Documentación	4	5	3	3
Datasets	4	5	2	2
Velocidad	3	4	4	5
Especialidad	4	5	5	2
Curva de aprendizaje	5	5	3	4
Lenguaje	5	5	5	5
Total	29	33	24	22

Cuadro 2: Trade Study, tabla de puntajes.

Como podemos ver en el Cuadro 2, las herramientas con mayor puntaje es Tensorflow. Dentro de las características con más peso se encuentra la documentación, en la cual no solo se toma en cuenta la información dada por los desarrolladores sino también su popularidad o tamaño de su comunidad, TensorFlow cuenta con una comunidad muy grande y una grán cantidad de ejemplos y proyectos por parte de de la misma. Otro aspecto de gran importancia para el proyecto es la especialidad, pues se tienen objetivos bastante claros a cerca del uso de la herramienta. TensorFlow es una de las mejores herramientas para el desarrollo de redes

neuronales y tensores. Finalmente, otro aspecto a tomar en cuenta es el manejo de Datasets según su tamaño. TensorFlow es flexible en este aspecto y no es necesario un proceso de agilización de entrenamiento de la red neuronal (como el que ofrece XGBoost por medio de Boosting Gradients o el uso de GPU para el procesamiento de esto como el que ofrece PyTorch). Cabe destacar que el módulo de Keras ayudará a facilitar el desarrollo del proyecto ya que ofrece facilidad para la creación de modelos de redes neuronales.

CAPÍTULO 10

Chatbot

En el capítulo anterior se define por medio de un trade study la herramienta a utilizar para el desarrollo del chatbot; debido a que la mejor opción fue Tensorflow y Keras. Se utilizó el lenguaje de programación Python debido a su grán versatilidad para el desarrollo de este tipo de proyectos y la basta cantidad de documentación que existe del mismo.

Para el desarrollo de este proyecto se realizó un proceso iterativo, por lo que se comenzó con un modelo de redes neuronales que pudiese escoger una respuesta en inglés de una base de datos, basándose en el tema reconocido dependiendo de la entrada de texto que recibe.

10.1. Preparación de la data

10.1.1. Base de datos

Para el proceso de desarrollo de redes neuronales es muy importante procesar la data que recibirá la red para ser entrenada debido a que necesitamos que sea capaz de predecir basándose en la misma. Por ello, se optó por la facilidad de tener un archivo como base de datos que contenga tanto las respuestas que puede generar el bot como la data con la cual se entrenará; este archivo tendrá un formato JSON por su facilidad de lectura para la máquina y cuenta con la estructura de un diccionario en python, por lo que resulta óptimo y eficiente extraer datos del archivo. La estructura de este documento será de la siguiente manera: El nombre del diccionario será “intents”, el cual tiene asignado un array que contiene varios diccionarios; cada uno cuenta con los items “tag” que indica el tema reconocido por el robot, “patterns” que contiene un array con varias frases referentes y recurrentes al tema indicado en “tag” y finalmente ’responses’ que contiene un array con varias respuestas referentes al tema. Vemos que esta estructura nos permite definir de forma clara el tema que debe predecir la red

neuronal por medio del item “tag” basándose en el entrenamiento recibido por la información percibida en el item “patterns” y, ya teniendo reconocido el tema del cual recibió información del usuario, escoge una de las respuestas encontradas en el item “responses” dándole incluso variedad a la conversación si es que se le provee de más de una respuesta. Al momento de obtener estos datos para el entrenamiento del modelo, se dividirán en clases según su “tag” y se ignoran los caracteres como símbolos de interrogación y admiración, puntos y comas debido a que esto no agrega información concreta a la predicción.

10.1.2. Lematización

El proceso de lematización de la información requerida para el entrenamiento del modelo es crucial para hacer eficiente el proceso de aprendizaje de la red neuronal, pues sabemos que el lenguaje natural (sea inglés o español) es complejo y vasto, por lo que surge la necesidad de simplificarlo para generar un recurso que no tarde demasiado en construirse y que tenga una precisión óptima al momento de predecir el tema de la conversación. Por ello, la lematización es la herramienta perfecta; los lematizadores son programas simples que intentan adivinar la raíz de una palabra. Por ejemplo, el lematizador de Porter, un algoritmo de lematización muy conocido, encontrará que ’universidad’ y ’universidades’ comparten la misma raíz y por lo tanto pertenecen a la misma clase de raíz.

Como se dijo al inicio del capítulo, el desarrollo se comienza por medio de una red neuronal que entrena con una base de datos en inglés, por lo que inicialmente se utilizó de la librería de Python “Natural Language Toolkit” un lematizador llamado “WordNetLemmatizer” el cual obtiene de manera sencilla el lema de cualquier cualquier palabra del idioma inglés. Este lematizador es reemplazado por uno de la librería “spaCy” al momento de utilizar la base de datos con la información en español, pues WordNetLemmatizer no cuenta con su función en español a diferencia de spaCy; cabe aclarar que durante esta transición es importante obtener la data de la base codificada en el formato “utf-8” debido a las tildes (las cuales influyen en la función del lematizador).

```
nlp = spacy.load('es_core_news_sm') #se carga el lematizador
nltk.download('wordnet')

# Funcion para lematizar palabras
def lemmaSP(word):
    doc = nlp(word)
    for token in doc:
        lemma = token.lemma_
    return lemma
```

Figura 36: Función de lematización de palabras.

10.1.3. Serialización de objetos

Ya que obtuvimos los temas con sus respectivas respuestas de la base de datos, es necesario ponerlos en listas de python para tener un manejo mas sencillo de la data, por lo que se generan listas con las clases (tag) y una lista con los patrones y sus respectivas clases.

Esta es información necesaria aun después del entrenamiento del modelo, por lo que la mejor opción para obtener estas listas en vez de realizar nuevamente el proceso de la base de datos es la serialización de estos objetos (las listas). Con ayuda del módulo de serialización de objetos de Python llamado "Pickle", se generan dos archivos ".pkl" los cuales contienen dichos objetos convertidos en un byte stream los cuales pueden ser de-serealizados en cualquier otro momento por medio de python cuando se necesiten. En este caso, se genera uno para la lista de clases y otro para la que contiene las clases con sus respectivos patrones.

10.1.4. Listas de entrenamiento

El proceso de entrenamiento del chatbot es crucial para los resultados que este puede generar. Hay que recordar que el tipo de chatbot que se definió para este proyecto es uno que su conocimiento se limita por su base de datos pero que no necesita comandos exactos para generar una respuesta coherente. Esto genera una necesidad de poder agilizar el proceso de entrenamiento del modelo predictivo sin importar la cantidad de texto que contenga la base de datos. Por ello, la lista de entrenamiento se genera de forma ingeniosa de la siguiente manera: Se lee y se carga el contenido de la base de datos (JSON) a una variable y se crean listas para almacenar las palabras, clases y una con caracteres que deseamos ignorar, como las comas, puntos, signos de interrogación y admiración. Ahora, por medio de un ciclo generamos una lista con cada palabra lematizada y en minúsculas que se encuentran dentro de los patrones de la base de datos.

```
intents = json.loads(open('intentsUVG.json', encoding="utf-8").read()) # Se lee el archivo JSON

words = []
classes = []
documents = []
ignore_letters = ['?', '!', '.', ',']

# Se extraen los datos del diccionario intents para formar las listas que contienen las clases y palabras de los patrones
for intent in intents['intents']:
    for pattern in intent['patterns']:
        word_list = nltk.word_tokenize(pattern)
        words.extend(word_list)
        for i in ignore_letters:
            if i in words:
                words.remove(i)
        documents.append((word_list, intent['tag']))
        if intent['tag'] not in classes:
            classes.append(intent['tag'])


mc = []
list1 = words

# Se convierten todas las palabras de la lista en minusculas
for i in list1:
    mc.append(lemmaSP(i.lower()))

# Se ordenan y se eliminan palabras repetidas
words = mc
words = sorted(set(words))
classes = sorted(set(classes))
```

Figura 37: Ciclo para generar lista de palabras y clases.

Es necesario serializar estas dos listas pues se utilizarán al momento de realizar la predicción. Ahora, se utiliza una serie de ciclos para poder generar una matriz con dos columnas y un número de filas igual al numero de frases contenidas en cada patrón de la base de datos. en la primera columna se tendrán listas de "1" y "0" en la misma posición en la que se encuentra cada palabra de la frase que estamos transformando en la lista de palabras que generamos anteriormente; esto quiere decir que, si el primer patrón es la palabra "hola" y en nuestra lista de palabras la palabra "hola" se encuentra en la posición 4 en una lista de 10 palabras,

esta lista se verá como: “[0,0,0,1,0,0,0,0,0,0]”. En la segunda columna tendremos una lista similar a la anterior, pero indicando a qué clase pertenece la frase; esto quiere decir que si la frase pertenece a la clase “Saludo” y en la lista de clases esta será la posición 2 de 5 clases entonces la lista se verá como: “[0,1,0,0,0]”. Este proceso es claramente complejo pero agiliza el entrenamiento debido a que no se analiza letra por letra para un lenguaje complejo sino una serie de listas que, dependiendo de su contenido, pertenecen a una categoría codificada. Finalmente, desordenamos aleatoriamente esta matriz para no generar ningún tipo de sesgo en el modelo.

```
# Se serializan las listas "words" y "classes" para utilizarlas en el chatbot
pickle.dump(words, open('wordsUVG.pkl', 'wb'))
pickle.dump(classes, open('classesUVG.pkl', 'wb'))

training = []
output_empty = [0]*len(classes)

# Se crea la matriz de entrenamiento del modelo
for document in documents:
    bag = []
    word_patterns = document[0]
    word_patterns = [lemmaSP(word.lower()) for word in word_patterns] # Lista con las palabras de cada patron lematizadas y en minusculas
    print(word_patterns)
    for word in words:
        bag.append(1) if word in word_patterns else bag.append(0) # Lista que indica las palabras encontradas en cada patron de la lista de palabras
    print(bag)
    output_row = list(output_empty)
    output_row[classes.index(document[1])] = 1 # Lista que indica a que clase pertenece la frase anterior
    print(output_row)
    training.append([bag, output_row]) # Lista de entrenamiento con las frases convertidas y la clase a la que pertenece
```

Figura 38: Ciclo para generar la matriz de entrenamiento.

10.2. Red neuronal

Debido a que el chatbot únicamente tiene un tensor de entrada y un tensor de salida, el modelo de red neuronal del tipo secuencial es apropiado debido a su estructura, pues es una pila plana de capas. Para el proceso de entrenamiento del modelo se genera un modelo del tipo secuencial con varias capas distintas.

10.2.1. Definición de las capas

La primera capa es una capa densa de 128 nodos, con una entrada de datos del mismo tamaño que la lista que contiene los patrones codificados con activación del tipo ReLU. Se escoge este tipo de activación ya que presenta facilidad de entrenamiento pues no presenta “no linealidades” y alcanza un mejor rendimiento con el tipo de tarea que va a realizar, pues hay que recordar que este tipo de función de activación únicamente se activa si los valores son positivos. [22]

Se usa una capa del tipo “Dropout” entre capas ya que estas de forma aleatoria convierten unidades de la entrada en cero dependiendo de la frecuencia que definamos durante el entrenamiento. Esto se hace con el fin de ayudar a prevenir el overfitting.[23]

Luego, se utiliza otra capa densa de 64 nodos, con función de activación del tipo ReLU. Finalmente, se agrega una capa densa con la cantidad de nodos del tamaño de los elementos de salida y activación de tipo Softmax. Normalmente se utiliza este tipo de activación en la última capa de una red de clasificación (que es el tipo de red que se está desarrollando) porque el resultado puede interpretarse como una distribución de probabilidad.[22]

Cabe recalcar que los valores especificados para la cantidad de nodos de las capas están dados por un proceso iterativo en el que se fue ajustando dichos valores hasta conseguir un resultado óptimo del modelo. Este criterio es correcto según el teorema de aproximación universal, que dicta que con suficientes nodos en la capa, esta será capaz de aproximar cualquier mapa continuo.

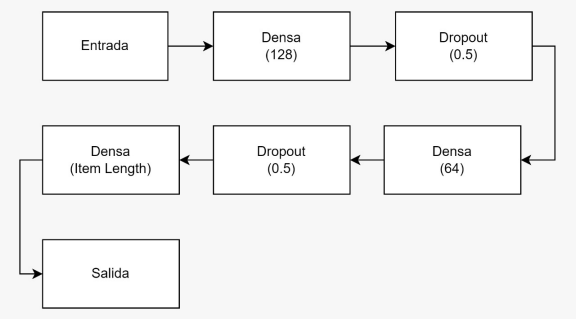


Figura 39: Diagrama de bloques del modelo (secuencial) por capas.

10.2.2. Compilación del modelo

Con la finalidad de optimizar el modelo, se implementa un algoritmo de optimización pues hará que la red neuronal tenga una mayor eficiencia en un menor tiempo. Para ello, se utiliza el algoritmo de optimización SGD (Stochastic Gradient Descent) que en cada iteración utiliza un único punto de datos para calcular el valor del gradiente, por lo que se tiene una convergencia mucho más rápida. [24]

Ahora, para poder implementar el algoritmo debemos establecer los hiper parámetros del mismo, estos pueden ir cambiándose a prueba y error al momento de entrenar el modelo con el fin de encontrar un set de hiper parámetros que presenten resultados óptimos. Finalmente, hay que definir algunos parámetros para compilar el modelo. Primero, se utiliza la función de pérdida "categorical crossentropy" pues es la que se usa en modelos que clasifican varias clases distintas (Se sabe desde la base de datos que existen varias categorías de la temática de la conversación que intentará predecir el modelo). Se le indica que el optimizador a utilizar será el definido previamente (SGD) y la métrica a utilizar será "accuracy" que calcula y da como resultado el porcentaje de exactitud con la que el elemento se asemeja a alguna de las frases de entrenamiento de las categorías. Tanto el algoritmo de optimización como las listas de entrenamiento y el número de iteraciones (epochs) se introducen en el modelo con el método "fit" de la clase "model" de la librería de Keras. Es importante guardar este modelo en un formato ".h5" para utilizarlo al momento de realizar predicciones.

Vemos que una gran parte de las decisiones tomadas para el desarrollo del modelo de la red neuronal están enfocadas a evitar el overfitting. Esto es debido a que este tipo de aplicación en específico presentará una gran cantidad de cambios en los datos de entrada respecto a los utilizados durante el entrenamiento, por lo que tener un modelo con overfitting resultaría fatal para el proyecto teniendo en cuenta que el modelo sería demasiado riguroso con las palabras que utilice el usuario al momento de categorizar la conversación cuando el resultado que estamos buscando es un modelo más generalizado en ese aspecto.

```
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbotmodelUVG.h5', hist)
print("Done")
```

Figura 40: Código de compilación del modelo de la red neuronal.

Parámetro	**Valor**
Learning rate	0.01
Decay	1e-06
Momentum	0.9
Epochs	200

Cuadro 3: Valores de hiper parámetros utilizados para el entrenamiento del modelo.

10.3. Tiempo de entrenamiento del modelo

El tiempo que tarda el modelo en entrenarse varía según la cantidad de información en la base de datos utilizada para su entrenamiento y la potencia del hardware en el que se realiza. A continuación, se presentan resultados para generar una pauta o referencia de cuánto tiempo (en segundos) tarda el hardware utilizado durante el desarrollo del proyecto dependiendo de la cantidad de tópicos en la base de datos (estos contendrán al menos 4 patrones distintos cada uno):

- **Procesador:** Intel Core i7-4720HQ.
- **GPU:** GTX 960M con 2GB VRAM.
- **Memoria RAM:** 8GB.
- **Sistema:** Windows 10 de 64 bits.
- **Entorno de desarrollo:** Python 3.9.6 de 64 bits

No. de tópicos entrenados	**Tiempo de entrenamiento (s)**
7	6.0799313
14	6.8264664
18	6.9222719
24	8.2695284

Cuadro 4: Tiempo que requirió el hardware para entrenar el modelo.

10.4. Implementación del modelo

En la sección anterior (Compilación del modelo) se obtiene un archivo ".h5" que contiene el modelo entrenado. De la sección 10.1. (Preparación de la data) se obtienen un archivo tipo JSON que contiene el diccionario de la base de datos, una lista con todas las palabras únicas

de la base de datos y otra con las clases de conversación (Estas en formato ".pkl"). Se crea un nuevo archivo de Python con el objetivo de ser un módulo contenedor de funciones útiles para la implementación del modelo. Para este proceso también es importante la lematización de los datos que introduce el usuario, por lo que es importante declarar la función "lemmaSP" mostrada en la Figura 36. A continuación se presenta un listado de las librerías a incluir en el archivo para poder implementarlo:

- Random
- Json
- Pickle
- Numpy
- NLTK
- La función "`load_model`" del método "models" de la librería de Keras de TensorFLow
- Spacy
- `speech_recognition`
- Pyttsx3

Por comodidad, se hace la inclusión y renombramiento de la librería Speech Recognition como se muestra en la Figura 31.

```
# Recuperación de listas, diccionario y modelo entrenado
intents = json.loads(open('TF/intentsUVG.json', encoding="utf-8").read())
words = pickle.load(open('TF/wordsUVG.pkl', 'rb'))
classes = pickle.load(open('TF/classesUVG.pkl', 'rb'))
model = load_model('TF/chatbotmodelUVG.h5')
```

Figura 41: Recuperación de objetos necesarios para la implementación.

10.4.1. De texto a voz

Parte del proyecto también es hacer que el "chatbot" pueda responder de forma humana, por lo que se utiliza la librería Pyttsx3 capaz de reproducir, por medio de las bocinas del dispositivo, una entrada de texto de manera sencilla. Para ello, definimos una función llamada "tts" con un solo parámetro que será la cadena de texto de la frase que deseamos reproducir. Dentro de la función, se inicia la librería, se indica la velocidad con la que deseas que lea el texto, se le indica el texto que se necesita decir y se corre el audio. Para poder ver el resultado de este módulo vea el siguiente video **Módulo de sinterización de voz**.

10.4.2. Procesamiento de entrada

Hay que recordar que los datos que ingresa el usuario, a pesar de que gracias al sistema de reconocimiento de voz desarrollado en el capítulo 8 es convertido de un audio a texto, son

```
# Función para convertir de texto a audio
def tts(phrase):
    engine = pyttsx3.init()
    engine.setProperty('rate', 125)
    engine.say(phrase)
    engine.runAndWait()
    return
```

Figura 42: Código de la función "tts".

una cadena de texto sin procesar. Para poder realizar la predicción es necesario procesar este texto convirtiéndolo en una lista de palabras lematizadas. Primero, generamos una función llamada "`clean_up_sentence`" con un parámetro que recibirá una cadena de texto que será la entrada dada por el usuario. Esta función se encargará de convertir esa cadena de texto en una lista de cadenas de texto (cada elemento será una palabra) y lematizará cada uno de los elementos por medio de la función "lemmaSP" de la Figura 36. Finalmente, generamos una función llamada "`bag_of_words`" que utilizará la función "`clean_up_sentence`" y eliminará las palabras que se repitan en la oración.

```
# Genera una lista con las palabras de la entrada lematizadas
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmaSP(word.lower()) for word in sentence_words]
    return sentence_words

# Limpia la oración de entrada y elimina los elementos repetidos
def bag_of_words(sentence):
    sentence_words = clean_up_sentence(sentence)
    bag = [0]*len(words)
    for w in sentence_words:
        for i, word in enumerate(words):
            if word == w:
                bag[i] = 1
    return np.array(bag)
```

Figura 43: Funciones "`clean_up_sentence`" y "`bag_of_words`" para el procesamiento de texto.

10.4.3. Predicción de clase y generación de respuesta

Con una entrada de datos limpia podemos comenzar con la predicción. Se define una función llamada "`predict_class`" con un parámetro que contendrá la entrada del usuario como cadena de texto. Luego de obtener el resultado de la función "`bag_of_words`" se utiliza el método "predict" del módulo "model" de la librería de Keras de TensorFLow ya que este realiza una predicción con la regresión del modelo. Finalmente, como resultado se obtiene la clase predicha por el modelo y la probabilidad de que sea esta. Estos datos son primordiales para conocer si el modelo realmente identificó correctamente la clase de conversación y no solamente tomó la opción más cercana. También se genera una nueva función llamada "`get_response`" con los parámetros "`intents_list`" que contendrá la lista con todas las clases y "`intents_json`" que contiene el diccionario completo. Esta función servirá para hacer una selección al azar de la lista de respuestas que existe en la base de

datos sobre la clase de conversación que se haya predicho. Para poder ver el resultado de la lematización, entrenamiento e implementación del modelo vea el siguiente video **Prototipo de entrenamiento del modelo e implementación**.

```
# Predicción de clase
def predict_class(sentence):
    bow = bag_of_words(sentence)
    res = model.predict(np.array([bow]))[0]
    ERROR_TRESHOLD = 0.2
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_TRESHOLD]
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({'intent': classes[r[0]], 'probability' : str(r[1])})
    return return_list

# Obtención de una respuesta
def get_response(intents_list, intents_json):
    tag = intents_list[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if i['tag'] == tag:
            result = random.choice(i['responses'])
            break
    return result
```

Figura 44: Funciones "`predict_class`" y "`get_response`" para la predicción.

CAPÍTULO 11

Interfaz gráfica

En este punto del proyecto, una parte crucial para finalizar la implementación del modelo y darle una respuesta al usuario es una interfaz gráfica capaz de reunir los elementos de funcionalidad que convierten el procedimiento de introducción de datos en una conversación natural y ofrecer al operador del programa un aspecto visual simplificado de lo que está sucediendo.

Esta parte del proyecto es un proceso de diseño creativo, puesto que es importante generar un diseño agradable para el operador. Como primer paso, se pensó en qué tipo de interfaz sería la adecuada para una aplicación de este tipo. Me pareció una gran idea representar la interacción entre el usuario y el bot como una "Aplicación de chat" ya que cuenta con una vista amigable para el operador y este es capaz de visualizar el flujo de la "conversación".

Ahora, se crea un mockup de la vista general del resultado final de la interfaz gráfica. Luego de varios cambios de estilo y discusiones con una estudiante de diseño gráfico con experiencia en el diseño de aplicaciones, se llega al siguiente resultado mostrado en la Figura 45.

11.1. Desarrollo de la interfaz gráfica

El primer paso para el desarrollo de la interfaz gráfica es decidir el entorno y herramientas que se utilizará para desarrollarla. Como la mayor parte del proyecto fué desarrollada en Python entonces se decidió que se utilizará este mismo entorno. La librería Tkinter es una librería que viene por defecto al instalar Python sin embargo, esta librería es muy básica y requiere de mucho más dedicación para alcanzar resultados atractivos. Por ello, Se utiliza una librería llamada Kivy, la cual está basada enfocada al desarrollo de interfaz gráfica para aplicaciones en dispositivos multi-touch (Smartphones, tablets, etc). Esto último implica que

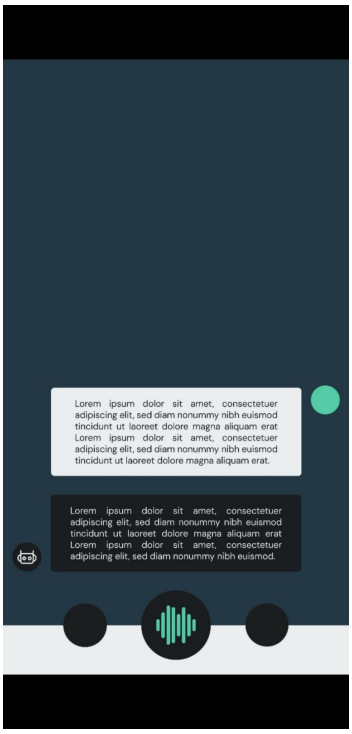


Figura 45: Diseño guía para la interfaz gráfica de la aplicación.

las animaciones y recursos gráficos generales de la librería son bastante fluidos y atractivos, a diferencia de los recursos de Tkinter que están enfocados en el estilo gráfico clásico de Windows.

11.1.1. Distribución del layout

Antes de comenzar a dar estilo a lo que se muestra en pantalla, es primordial planear una estructura en la pantalla. Kivy provee de una grán variedad de layouts que permiten organizar los objetos que se muestren dentro de los mismos. Cada uno tiene diferentes características y, con el objetivo de no generar conflicto entre los distintos objetos y texto que estarán cambiando su tamaño constantemente durante el uso de la aplicación, se decide optar por usar Layouts con propiedades de posición y tamaño relativos. Kivy ofrece la oportunidad al desarrollador de poder anidar distintos tipos de Layout lo cual facilitará el posicionamiento y organización de los objetos siempre y cuando utilicemos posiciones y tamaños relativos.Entonces, se creó un boceto para planear e indicar la forma en que se distribuyen los objetos en la ventana de la aplicación; este se muestra en la Figura 46. También se incluye un pequeño ejemplo, del cómo distribuye los objetos en su interior, de cada tipo de Layout utilizados en el proyecto.

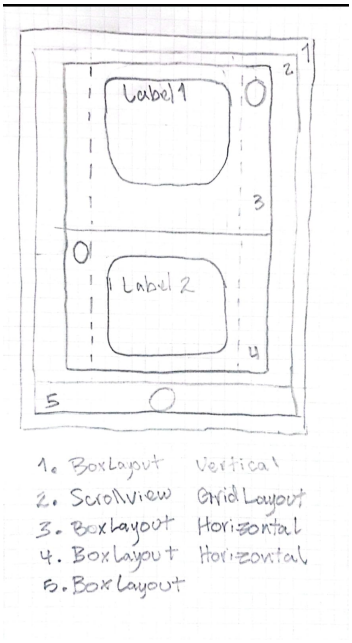


Figura 46: Boceto de distribución de layouts en la aplicación.

11.1.2. Declaración de la App

Para poder comenzar con la ejecución de la interfaz gráfica es necesario importar la librería de Kivy. De esta, se importan los Objetos que se utilizarán: App, Window, BoxLayout, Widget, Button y Clock. Adicionalmente, se importa la función "sleep" de la librería "Time" de Python; se importa una librería llamada "Threading" y el módulo de funciones desarrollado en el capítulo anterior (Generado en la parte 10.3).

El desarrollo de interfaz gráfica por medio de Kivy cuenta con una estructura que facilita la organización del código: Declaración y ejecución de la ventana, funciones y hoja de estilos. Primero, se define una clase del tipo "BoxLayout" que contendrá el tipo de Layout que contendrá todo en la pantalla. Dentro de esta clase se definen funciones que pueden cambiar de forma activa elementos dentro de nuestra aplicación; esto se define más tarde. Luego, se define una clase la cual será la ventana de tip "App" que muestra todo el contenido de la aplicación y es quien la construye; se debe indicar que esta clase debe regresar el Layout definido previamente. Finalmente, debemos indicar en el archivo que debe comenzar a ejecutar la clase que contiene nuesta ventana del tipo "App". Esta es la estructura básica de una aplicación en Kivy pero claramente está vacía, por lo que comenzaremos a llenarla por medio de la hoja de estilos.

11.1.3. Hoja de estilos

Para comenzar con la hoja de estilos se debe crear un archivo de texto con el mismo nombre de la clase de tipo "App" que se declaró en el archivo de ejecución de la ventana. Este archivo debe llevar una extensión ".kv", la primera linea debe llevar el símbolo numeral, seguido de ":kivy " y la versión exacta de kivy instalada en el equipo (Para este caso, esta

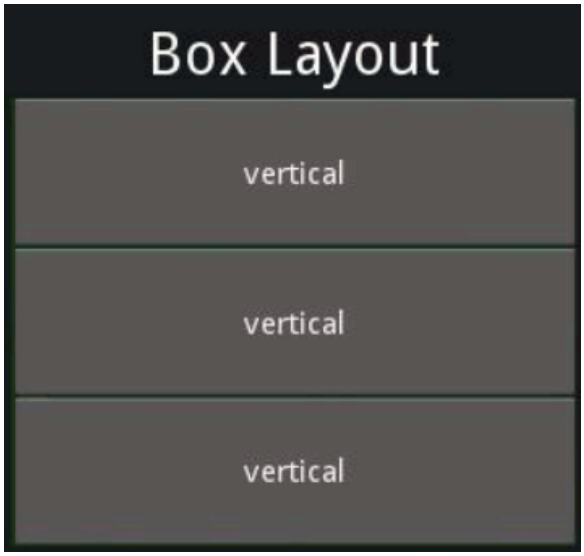


Figura 47: Ejemplo de distribución de BoxLayout.

linea de código se vera como ":kivy 2.0.0") y el archivo debe encontrarse en la misma carpeta que el archivo en el que se ejecuta la App. La estructura y sintaxis de este archivo es muy similar a una hoja de estilos del lenguaje CSS.

Una vez se tiene el archivo, es momento de utilizar el boceto mostrado en la Figura 46 para comenzar a estructurar de manera adecuada los objetos en la ventana de la aplicación. Declaramos la etiqueta principal con el mismo nombre de la clase definida como "BoxLayout" en el archivo de ejecución (En este caso tendrá el nombre de "MyWidget"). Dentro de esta etiqueta declaramos el primer BoxLayout (No. 1 en el boceto) con orientación vertical. Este es quien contendrá los elementos de la barra de color blanco fija (No. 5 del boceto) y el ScrollView donde se desplegará el chat (No. 2 del boceto). El ScrollView cuenta con la característica de ser un espacio que puede desplazarse hacia abajo y hacia arriba de forma indefinida y con una animación muy fluida. Sin embargo, la desventaja de utilizar ScrollView es que únicamente podemos utilizar una GridLayout dentro de esta, por que declaramos una dentro que cuenta con una sola columna para poder posicionar de manera vertical los elementos dentro de la misma. Dentro, declaramos dos BoxLayout, una identificada como el espacio de información del usuario y la otra identificada como el espacio de información del chatbot (Cajas No. 3 y 4 del boceto). Ambas cajas tienen orientación horizontal y cuentan con tres elementos en su interior: Espacio vacío, Label con el mensaje y figura con el icono del remitente. El espacio vacío y el icono intercambian de lugar dependiendo de si la caja de información pertenece al usuario o al chatbot. Para Introducir el icono de cada remitente se puede usar una Boxlayout con la propiedad Canvas que permite ingresar una figúra, como una elipse o un rectángulos, que puede contener una imagen (en este caso utilizamos una elipse). Todos los elementos se acomodan y se les asigna un tamaño cantidades relativas, esto con el objetivo de que la aplicación no se deforme si es ejecutada en una ventana con mayor o menor tamaño. El espacio vacío debe contener la misma cantidad de espacio que el del icono del remitente para tener una buena relación de espacio con el texto de abajo. Finalmente, al label se le dá un identificador (uno para el chatbot y otro distinto para el usuario) ya que esta propiedad será la que nos permita cambiar el texto a medida que fluye la conversación entre el usuario y el chatbot. Se le da un fondo rectangular con las esquinas redondeadas

Figura 48: Ejemplo de distribución de GridLayout.

```
import kivy
kivy.require('2.1.0') # replace with your current kivy version !

from kivy.app import App
from kivy.uix.label import Label


class MyApp(App):

    def build(self):
        return Label(text='Hello world')


if __name__ == '__main__':
    MyApp().run()
```

Figura 49: Ejemplo de estructura básica de declaración y ejecución de una ventana en Kivy.

para captar el estilo del Mockup mostrado en la Figura 45 y también se utiliza el tipo de fuente que se incluye dentro de los archivos de la aplicación. A medida que avanzamos con los elementos del estilo se cambian las propiedades para tener un resultado que concuerde con el Mockup de la Figura 45.

11.1.4. Funcionalidad

En la sección anterior se obtuvo una interfaz visual muy similar en cuestión visual y estructural al Mockup. Sin embargo, es necesario introducir el mensaje que envía el usuario y el de respuesta del chatbot en los Labels asignados. Para ello, se utiliza el método "ids" seguido del identificador del objeto y la propiedad que deseamos cambiar del mismo.

```
self.count = self.count + 1
self.ids.userMsg.text = "Updated % d...times"% self.count
```

Figura 50: Ejemplo de sintaxis para el cambio de texto de labels.

Ahora conocemos cómo cambiar el contenido de labels sin embargo, Kivy presenta una interfaz gráfica con propiedades ya cargadas pero no presenta ningún "Loop" que cargue

nueva información. Para poder actualizar propiedades Nos aprovechamos de la propiedad de "Clock" que presenta la interfaz en segundo plano por parte de Kivy. Por ello, se programa una "alarma" que le indicará cada cierto tiempo a la interfaz que debe actualizar el texto de las labels basándose en el contenido de dos variables globales (En mi caso llamadas "text" que contiene el mensaje enviado por el usuario y "res" que contiene la respuesta generada por el chatbot). Para declarar esta alarma, debemos declarar una función llamada "`__init__(self, **kwargs)`" dentro de la clase "MyWidget". Dentro de esta función utilizaremos una función "super()" para llamarse a si misma, de manera que esta se ejecutara en el momento en el que se ejecute la aplicación. Luego, del módulo "Clock" utilizamos la función "`schedule_interval`" para poder declarar la "alarma". Esta función recibe dos parámetros: La acción o función que deseamos que se ejecute al momento de sonar la "alarma" y la cantidad de tiempo (en segundos) que tardará en ejecutar la siguiente "alarma". Cabe aclarar que estas seguirán "disparándo" la alarma indefinidamente hasta acabar el programa. Colocaremos dos alarmas, una para cambiar el mensaje del usuario y otra para cambiar el mensaje del chatbot y los cambiaremos cada medio segundo, pues tampoco serán conversaciones extremadamente rápidas. Definimos dos funciones: una para cambiar el mensaje del label del usuario y otra para cambiar el mensaje del label del chatbot. Para poder ver el resultado de este apartado del proyecto vea el siguiente video **Prototipo de la interfaz gráfica del proyecto**.

```
# Almacenan los mensajes del chatbot y el usuario
global text
global res

# Se ejecuta al inicio de ejecución de la ventana y programa dos alarmas
def _init_(self, **kwargs):
    super(MyWidget, self)._init_(**kwargs)
    Clock.schedule_interval(self.changeUser, 0.5)
    Clock.schedule_interval(self.changeBot, 0.5)

# Actualiza el label que contiene el mensaje dsel usuario
def changeUser(self, dt):
    self.ids.userMsg.text = text

# Actualiza el label que contiene el mensaje dsel usuario
def changeBot(self,dt):
    self.ids.botMsg.text = res
```

Figura 51: Código de actualización de los mensajes del chatbot y el usuario.

11.1.5. Threading

Ya contamos con una interfaz gráfica capaz de mostrar de manera ordenada los mensajes del usuario y el chatbot dependiendo del valor contenido en dos variables globales llamadas "text" y "res". Sin embargo, ¿cómo actualizar el valor de estas dos variables? En el capítulo 8, en las Figuras 30 y 31. Desarrollamos la estructura básica para poder recibir y convertir la voz del usuario en una cadena de texto y el el capítulo 10, sección 3, obtuvimos varias funciones que nos dejan utilizar esa cadena de texto para predecir el tema de conversación, generar una respuesta y reproducirla mediante bocinas. Por desgracia, todas estas acciones no se ejecutan en segundo plano y la ejecución de una interfaz gráfica por medio de Kivy (y cualquier otra librería similar) se vé interrumpida por estos procesos y no permite que el reloj ejecutado para la actualización de datos siga corriendo y si el usuario intenta interactuar

con la interfaz esta detiene por completo todo el proceso. Por ello, es necesario utilizar una estrategia de "threading". Esto permite utilizar otro hilo del procesador para ejecutar otra acción al mismo tiempo en el que se ejecuta nuestra aplicación. Durante la declaración de la App (11.1.2) se importa la librería de "Threading" y es momento de utilizarla. Dentro de la definición de la clase de nuestra App (Window) declara una función que contendrá todo el proceso que se ejecutará en paralelo al despliegue de la interfaz gráfica (en mi caso es llamada "chatBotDo()").

Dentro de "chatBotDo()" el primer paso es declarar dentro las variables globales utilizadas para el cambio de texto en los labels del mensaje del usuario y el chatbot. Como señal de inicio, hacemos que el bot diga una frase de bienvenida y comenzamos con un bucle que repetirá constantemente todo el proceso ya que necesitamos que el chatbot este constantemente escuchando. Por ello, utilizamos el código mostrado en la Figura 30 para obtener la entrada del usuario y utilizamos la función "`predict_class`" de la Figura 44 con el texto recuperado del usuario como parámetro. Como este proyecto se encontrará en un ambiente en el que puede escuchar distintas conversaciones, se agrega un condicional el cual hará entrar al bot en un bucle de preguntas y respuestas únicamente si lo primero que identificó fue un saludo. Si el bot no se encuentra en este "modo de preguntas y respuestas" ignorará completamente las preguntas. Ya dentro de este modo, el bot estará esperando una entrada del usuario y si la recibe y la reconoce, intentará predecir la clase de conversación con la función "`predict_class`", obtendrá y almacenará una posible respuesta al tema utilizando la función "`get_response`" y la reproducirá por las bocinas luego de obtenerla por medio de la función "tts" de la Figura 42. Automáticamente después de responder a la pregunta, el bot seguirá escuchando a las preguntas del usuario y seguirá respondiéndolas a no ser que el último tema reconocido haya sido una despedida; si es ese el caso, el bot se despedirá y volverá a su estado anterior (Sigue escuchando, pero no responde hasta que se identifique como un saludo). Hay que recordar que el mensaje en reconocido y convertido a texto del usuario debe almacenarse en la variable "text" y cualquier respuesta generada por el chatbot debe ponerse en la variable "res" ya que, en otro hilo del procesador, estas están siendo mostradas en la interfaz gráfica y debe ser actualizado su valor cada vez que cambia. Finalmente, esta función está preparada para ejecutar el proceso deseado, por lo que para ejecutarlo en un hilo en paralelo con la ejecución de la interfaz gráfica utilizaremos la función "Thread" de la librería "threading". Pondremos la función "chatBotDo" mostrada en las Figuras 52 y 53 como "target" e indicamos que este no será un "daemon thread", pues este tipo de hilos se cierran únicamente al momento de apagar la computadora y no es lo que queremos. Un pequeño video complementario con el análisis del principio que utilicé para desarrollar este apartado es **Multithreading video complementario**.

11.1.6. Herramienta de ingreso de nuevos conocimientos

Como uno de los objetivos es generar una base de datos amplia, una forma de alcanzarlo es crear una herramienta que el operador sea capaz de manejar de forma sencilla para poder ingresar nuevos conocimientos al modelo del chatbot. Para ello, utilicé los mismos principios para crear una interfaz gráfica sencilla en la que el operador puede ingresar el tag, 4 patrones y 2 respuestas y presionar un botón que comenzará a entrenar nuevamente el modelo para que adquiera estos nuevos conocimientos. Es importante mencionar que para que funcione de manera adecuada todos los campos deben estar llenos y la aplicación del chatbot debe estar

```
def chatBotDo(self):

    # Variables utilizadas para actualizar los globos de texto del usuario y el chatbot
    global text
    global res
    global mouth

    # El bot saluda al estar iniciado por primera vez
    tts("Bienvenido")
    res = "Bienvenido"
    text = "..."
    m = 0

    # Bucle de reconocimiento de saludo
    # El bot no responde a ninguna pregunta hasta que lo saluden
    while(1):
        with sr.Microphone() as source:
            audio = r.listen(source)
            try:
                text = r.recognize_google(audio, language='es-GT')
                ints = predict_class(text)
                print(ints)
                if ints[0]['intent'] == 'Saludo' and float(ints[0]['probability']) >= 0.7:
                    m = 1                   # Bandera para entrar al bucle "Modo Conversacion"
                    res = get_response(ints, intents)
                    mouth = 1
                    tts(res)
                    mouth = 0

                    print(res)
            except:
                print("Lo siento, no pude entenderte")
```

Figura 52: Bucle principal del proceso de conversación y modo “reconocimiento de saludo”.

```
# Modo Conversacion
# El bot responde preguntas automaticamente una detras de otra hasta que se despidan de el
while m == 1:
    with sr.Microphone() as source:
        audio = r.listen(source)
        try:
            text = r.recognize_google(audio, language='es-GT')
            ints = predict_class(text)
            if ints[0]['intent'] == 'Despedida' and float(ints[0]['probability']) >= 0.3:  # Si se despiden del bot...
                m = 0                       # Bandera para regresar al modo de reconocimiento de saludo
                res = get_response(ints, intents)
                mouth = 1
                tts(res)
                mouth = 0
                print(res)
            elif ints[0]['intent'] != 'Despedida' and float(ints[0]['probability']) >= 0.3:
                res = get_response(ints, intents)
                mouth = 1
                tts(res)
                mouth = 0
                print(res)
            else:
                mouth = 1
                tts("Lo siento, no pude entenderte")
                mouth = 0
                res = "Lo siento, no pude entenderte"
        except:
            mouth = 1
            tts("Lo siento, no pude entenderte")
            mouth = 0
            res = "Lo siento, no pude entenderte"
```

Figura 53: Bucle del proceso de conversación en modo “conversación”.

cerrada al momento de utilizar la herramienta y abrirla luego de que esta termine su proceso. Para ver una demostración y una breve explicación vea el siguiente video **Herramienta de actualización de conocimiento del chatbot**.

CAPÍTULO 12

Movimiento del rostro

Ya que se cuenta con un set de servomotores para representar por medio de una estructura el rostro animatrónico se programan algunos movimientos para representar el momento en el que habla el bot. El rostro únicamente debe mover la boca mientras habla, por lo que se envía una señal por medio de comunicación serial al microcontrolador para que este sepa cuándo debe hablar y cuándo no. Por ello, agregamos una variable global al archivo donde se ejecuta la aplicación (esta cambiará su valor entre 0 y 1 dependiendo de si el bot está hablando o no). Estos intervalos son muy sencillos de identificar, pues el bot habla en el momento en el que se llama la función "tts" y termina cuando esta regresa, por lo que debemos cambiar el valor de la variable antes y después de cada una de las lineas en de código en las que utilizamos dicha función. Finalmente, se declara una función que reciba como parámetro la variable donde tendremos el estado de habla del bot en nuestro archivo de módulo que se encarga del envío de datos por medio de comunicación serial. Se importa la librería "Serial"; se crea un objeto serial con un baud rate conocido, se especifica el puerto en el que se conectará el microcontrolador; se abre el puerto, se envía el mensaje y se cierra. Esta función la llamaremos al momento de actualizar el mensaje del bot en la interfaz gráfica, de modo que tendremos una actualización de este dato en el puerto serial cada medio segundo para el microcontrolador.

```
# Función para escribir en el puerto serial
def serialSend(state):
    ser = serial.Serial()
    ser.baudrate = 9600
    ser.port = 'COM5'
    ser.open()
    ser.write(state)
    ser.close()
    return
```

Figura 54: Función para enviar datos por medio de un puerto serial.

CAPÍTULO 13

Conclusiones

- Se lograron reconocer 24 temas de conversación distintos con al menos 4 patrones diferentes para cada uno de ellos.
- Según los resultados en el Cuadro 4, la cantidad de tiempo requerida para el entrenamiento del modelo aumenta según la cantidad de tópicos propuestos en la base de datos.
- Se logró crear una base de datos estructurada de fácil acceso y ampliación.
- Utilizar la herramienta de TensorFlow, Keras, fue una buena opción para el desarrollo del proyecto debido a la naturaleza del mismo.
- Se implementó la API de Google para el módulo de Speech Recognition gracias a su gran velocidad de respuesta y facilidad de uso.
- Se alcanzó la sincronización entre la reproducción de audio de la voz y los movimientos del rostro.
- Se implementó una interfaz gráfica atractiva y amigable para que el operador pueda ver el flujo de la conversación entre el chatbot y el usuario.
- Se logró mejorar la solidez de la estructura de la base de los ojos del rostro con el nuevo diseño de las piezas.
- Se logró implementar una librería para la sintetización de voz al momento de responder al usuario.
- Se logró desarrollar una herramienta de fácil uso para agregar nuevo conocimiento al modelo, por lo que la base de datos puede ser tan amplia como el operador lo decida.

CAPÍTULO 14

Recomendaciones

- Crear una encuesta o cuestionario para que alumnos, profesores o directores de distintas carreras puedan proponer información que consideren adecuada para nutrir la base de datos del animatrónico. De esta forma, este podría presentarse como un guía en las ferias para estudiantes de nuevo ingreso o aspirantes.
- Montar el equipo en un ambiente preferiblemente cerrado con un micrófono con cancelación de ruido con el objetivo de presentar la mayor claridad de audio para que el proceso de reconocimiento de voz resulte en un texto claro y limpio. Hablar con un tono de voz moderado y claro.
- Implementar un sistema de comunicación entre el sistema de reconocimiento facial y el sistema de reconocimiento de voz y generación de respuesta. De esta forma, poder conseguir una interacción no verbal y reactiva entre el animatrónico y el usuario para ampliar su funcionamiento.
- Es posible implementar reconocimiento de voz y lematización de forma offline. El sistema implementado en este proyecto necesita estar conectado a internet pues utiliza la nube de Google para realizar las consultas necesarias para el reconocimiento de voz, lematización y sintetizaron de voz. A excepción de la ultima consulta, es posible reemplazar las librerías implementadas por algunas que funcionan offline o directamente tener un modelo (no tan extenso como el de Google pero bastante robusto) de reconocimiento de voz y de lematización; sin embargo, hay que aclarar que estas dos opciones llevan un trabajo más extenuante y requieren un equipo más potente.
- Crear una versión movil para la interfaz y procesamiento de datos y conectar el controlador por medio de bluetooth al dispositivo. De esta manera, obtendríamos un proyecto más portátil.
- Modificar la interfaz de la herramienta de expansión, de modo que tenga un botón el cual primero modifique la base de datos agregando los nuevos elementos propuestos por el operador y otro botón el cual de comienzo al entrenamiento del modelo. De este modo, podrá agregar varios tópicos y entrenar el modelo una vez a diferencia de

entrenar el modelo cada vez que se agrega un nuevo tópico (que es el estado actual del proyecto).

- Crear una nueva interfaz o una nueva pestaña en la que el operador del sistema pueda ver, habilitar y deshabilitar, borrar y modificar los tópicos ya entrenados de la base de datos. La función de despliegue podría agregarse al programa principal y el resto de funciones a la herramienta de expansión.
- Para la ejecución del programa se recomienda que el equipo cuente de Windows 7 a 10, con 4GB de memoria RAM (8GB recomendados), gráficos integrados como Intel Integrated UHD 620 o similar y un procesador multinucleo y multihilo de al menos 7ma generación de Intel (o similar de AMD).
- Para el entrenamiento del modelo, además de las recomendaciones hechas para la ejecución del programa, se recomienda una tarjeta gráfica dedicada con al menos 2GB de VRAM. Cabe resaltar que esta que la tarjeta utilizada para el desarrollo y pruebas fué la GTX 960M. Esta es una recomendación de entrada y puede requerir un mejor hardware con una base de datos más amplia.

CAPÍTULO 15

Bibliografía

[1] G. Llorach y J. Blat, "Say Hi to Eliza," en *International Conference on Intelligent Virtual Agents*, Springer, 2017, págs. 255-258.

[2] M. Salecha, *Story of Eliza, the first chatbot developed in 1966*, jun. de 2020. dirección: https://analyticsindiamag.com/story-eliza-first-chatbot-developed-1966/.

[3] B. N. Mundo, "La sorprendente y poco conocida historia de Eliza, el primer bot conversacional de la historia," 2018. dirección: https://www.bbc.com/mundo/noticias-44290222.

[4] p. I. Ros, *Google Meena se convierte en el chatbot más avanzado que existe*, ene. de 2020. dirección: https://www.muycomputerpro.com/2020/01/30/google-meena-se-convierte-en-el-chatbot-mas-avanzado-que-existe.

[5] S. Sink, "Walt Disney, the 1964-65 World's Fair, and the emergence of audio-animatronics," 2015.

[6] FUAM, "Manual de comunicación para investigadores," 2015. dirección: http://fuam.es/wp-content/uploads/2012/10/INTRODUCCION.-La-Comunicacion.-Principios-y-procesos.pdf.

[7] C. Chen, *Elementos de la Comunicación*, ene. de 2021. dirección: https://www.significados.com/elementos-de-la-comunicacion/.

[8] P. Pandey, *Construyendo un Chatbot simple desde cero en Python (usando NLTK)*, mayo de 2021. dirección: https://planetachatbot.com/construyendo-un-chatbot-simple-desde-cero-en-python-usando-nltk-31b9ae4f71db.

[9] *El modelo de redes neuronales*. dirección: https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model.

[10] Á. Gonzalo, *Problemas comunes en Aprendizaje Automático*, dic. de 2018. dirección: https://machinelearningparatodos.com/problemas-comunes-en-aprendizaje-automatico/.

[11] *¿Qué es el clustering?: Detección de Comunidades*, sep. de 2020. dirección: https://www.grapheverywhere.com/que-es-el-clustering/.

[12] D. Calvo, *Clasificación de Redes neuronales artificiales*, mar. de 2019. dirección: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/.

[13] J. Cevallos Ampuero, "Redes Neuronales de Base Radial aplicadas a la mejora de la calidad," Español, *Industrial Data*, 2008, ISSN: 1560-9146. dirección: https://www.redalyc.org/articulo.oa?id=81619829009.

[14] D. Calvo, *Función de Activación - Redes neuronales*, dic. de 2018. dirección: https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/.

[15] P. I. G. Gavilán, *Catálogo de Componentes de Redes neuronales (iii): Funciones de pérdida*, mayo de 2020. dirección: https://ignaciogavilan.com/catalogo-de-componentes-de-redes-neuronales-iii-funciones-de-perdida/.

[16] J. I. Bagnato, *Qué es overfitting y underfitting y cómo solucionarlo*, dic. de 2017. dirección: https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/.

[17] S. Raschka, *Python machine learning*. Packt publishing ltd, 2015.

[18] *learn*. dirección: https://scikit-learn.org/stable/index.html.

[19] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu y Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. dirección: https://www.tensorflow.org/.

[20] *PYTORCH documentation*. dirección: https://pytorch.org/docs/stable/index.html.

[21] G. Samadrita, *Top 10 best machine learning tools for model training*, dic. de 2021. dirección: https://neptune.ai/blog/top-10-best-machine-learning-tools-for-model-training.

[22] J. Brownlee, *A Gentle Introduction to the Rectified Linear Unit (ReLU)*, ago. de 2020. dirección: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[23] K. Team, *Keras documentation: Dropout layer*. dirección: https://keras.io/api/layers/regularization_layers/dropout/.

[24] *Algoritmos de optimización en redes neuronales (BGD, SGD, MBGD, Momentum, NAG, Adagrad, AdaDelta, Adam, AMSGrad, método Newton), cómo elegir algoritmos de optimización para redes neuronales - programador clic*. dirección: https://programmerclick.com/article/682025059/.

CAPÍTULO 16

Anexos

Videos del proceso

https://youtube.com/playlist?list=PLT4XMdAQe-uLUDg8DBUvLjUOfUt5vMphP

Drive con los archivos del proyecto

https://drive.google.com/drive/folders/1iBHPVcqbHk2t5uZD6TR8P7Q_zsTe0vr2?usp=sharing