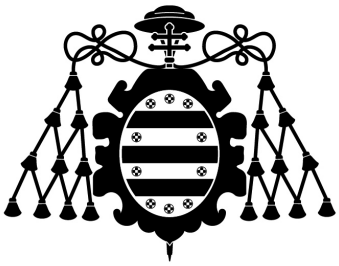


Universidad de Oviedo

Escuela de Ingeniería Informática

TFG

Aplicación para la traducción del alfabeto dactilológico en LSE a partir de imágenes usando deep-learning

Marcos Tobías Muñiz

Oviedo
8 de julio de 2022

Resumen

LSETranscriber es un servicio y aplicación web cuyo objetivo es facilitar la comunicación por internet para las personas que se comunican con el Lenguaje de Signos Español. Esto se hace a través del reconocimiento de signos del alfabeto dactilológico, que son obtenidos a través de la webcam del usuario e interpretados por un algoritmo de aprendizaje automático.
La aplicación web está enfocada a probar el funcionamiento de LSETranscriber, mientras que el servicio web proporciona un acceso al modelo entrenado para poder expandir el alcance de la aplicación y poder predecir signos para cualquier ámbito. Originalmente la idea era para facilitar la comunicación a través de internet para personas mudas, por ejemplo en Microsoft Teams o Google Meet, medios por los cuales se realizó el curso académico 2020-2021.
Estadísticamente en una conversación hay mayor entendimiento si esta es hablada, por lo que un intercambio entre una persona que usa lenguaje de signos a una persona que habla no solo es difícil por la barrera del lenguaje, si no por la pérdida de información que se produce en el caso de que la persona muda tuviera que comunicarse a través de escritura, siendo este el único medio de entendimiento común.

Dejado a un lado las posibilidades, como se comenta LSETranscriber es una prueba de concepto, por lo que se limita a reconocer el lenguaje de signos español (LSE), y dentro de este solo los signos estáticos del alfabeto dactilológico, un equivalente a nuestro abecedario pero con signos. Esto se debe a una reducción del problema para ser factible en el espacio de tiempo y carga de trabajo disponible, especialmente dado que los signos que implican movimiento aumentan la dificultad en gran medida.

La restAPI que compone el servicio web, permite la evaluación de dichos signos a través de imágenes, mientras que la aplicación web es capaz de evaluar los signos en tiempo real. Esto significa que el usuario puede verse realizar los signos a través de su webcam, donde habrá un temporizador que enviará una captura actual a intervalos específicos al servicio web, el cual evaluará la imagen y devolverá la predicción. También permite una opción para eliminar posibles predicciones incorrectas y para leer en voz alta.

La predicción se hace a través de un modelo de Redes Neuronales Profundas que ha sido previamente entrenado con muchas imágenes. Ya que el entrenamiento es solo de la región de la mano, hay un paso intermedio en el que se detecta la mano en la imagen y se recorta la zona.

Keywords

Red Neuronal, Python, WCAG2.0, Python, JavaScript, restAPI, React, Redux, Petición POST, ML, ASL, LSE, MNIST

Índice

Índice de figuras

1 Memoria del proyecto

En este apartado se describirá el sistema a construir en base a su origen, su alcance y los diferentes objetivos que se plantean conseguir al final de su desarrollo.

1.1 Resumen de la motivación, Objetivos y Alcance del proyecto

Originalmente este proyecto fue una idea para un proyecto de verano, debido a que me interesaba el ámbito de la inteligencia artificial y el Machine Learning. Ya que para empezar desde cero era un proyecto un poco desmesurado, decidí aprovechar el verano para aprender las bases de ML y dejar el proyecto para este Trabajo Fin de Grado.

La idea de hacer un Transcriptor de lenguaje de signos a voz originó un poco por dos motivos. Uno fue la pregunta de cómo se comunicaría las personas con discapacidad auditiva a través de internet, especialmente durante el curso académico 2020-2021 que fue completamente online. Por otra parte, debido al trabajo de mi madre como profesora de Educación Especial, era consciente de muchos problemas que puede tener las personas con algún tipo de discapacidad, problemas que muchas veces para las personas sin discapacidad pasan desapercibidos.

Pensé que una forma de poder reconocer los signos y leerlos facilitaría mucho la comunicación, no solo a través de internet si no en el día a día, y después de investigar un poco más vi que existían aplicaciones similares para el ASL (American Sign Language) pero que no había nada para el LSE (Lenguaje de Signos Español).

Hacer una aplicación comercial o realmente útil era un objetivo demasiado ambicioso para un TFG, decidí dejarlo en una prueba de concepto, intentando alcanzar el tiempo real, lo que dejaría muchas puertas abiertas a una posible futura extensión de la aplicación.

Por ello LSETranscriber se basa en un servicio web para el reconocimiento de signos del alfabeto dactilológico en imágenes, y la aplicación web es una interfaz de este servicio, mejorando su usabilidad y facilitando entender el objetivo y las increíbles posibilidades que se podrían conseguir a partir de esta base.

2 Introducción

En esta sección se realizará un estudio de las diferentes alternativas viables actualmente en el mercado, así como tecnologías. También se desarrollarán los objetivos del proyecto y se justificará el lugar que plantea ocupar y las necesidades que prevé cubrir.

2.1 Justificación del proyecto

LSETranscriber surge con la intención de permitir a las personas con discapacidad auditiva comunicarse por internet de una forma fluida, sin necesidad de tener que usar el teclado y usando signos como se comunicarían de forma normal. Al no existir ninguna aplicación para este propósito adaptado al LSE, sería la primera aplicación para este uso. Existen aplicaciones para este fin usando el ASL, pero al igual que en los lenguajes orales ambos son sustancialmente diferentes.

Debido a los limitados recursos y tiempo para el desarrollo de esta idea, LSETranscriber consistirá en una prueba de concepto, que obtendrá imágenes de la webcam de los usuarios y devolverá una predicción de la letra realizada en tiempo real. Solo identificará signos estáticos del alfabeto dactilológico para reducir la complejidad del problema.

Aún así, con LSETranscriber se pretende dar a conocer la posibilidad de realizar reconocimiento de signos de una forma fiable y rápida, estableciendo una línea base para futuros proyectos, incluso con la posibilidad de extender el propio LSETranscriber sin necesidad de tener que realizar todo el trabajo desde cero.

El código de la generación y entrenamiento del modelo será abierto para cualquier persona que quiera usarlo en sus proyectos, código que es fácilmente adaptable incluso a otros lenguajes de signos que no sea el LSE, mientras que la entrada consista en signos estáticos a través de imágenes. En el caso de querer usar el modelo implementado, este también estará disponible para su uso.

De esta forma, se plantea la generación de un proyecto abierto y colaborativo, para intentar reducir la barrera del lenguaje en internet para las personas con discapacidad auditiva, sin establecer una barrera de pago en la comunicación, lo cuál iría en contra del Derecho a la comunicación, uno de los principales derechos humanos.

2.2 Objetivos del proyecto

- Reconocimiento de signos estáticos del alfabeto dactilológico del Lenguaje de Signos Español
- Proveer un servicio web que permita a cualquier persona utilizar el modelo entrenado para predecir signos en sus aplicaciones.
- Realizar una aplicación web que ayude a entender cómo funciona el modelo
- Capacidad de realizar suficientes predicciones por segundo para soportar a varios usuarios concurrentes
- Obtener un porcentaje de exactitud suficiente a la hora de predecir signos que asegure un entendimiento de la información que se intenta transmitir

2.3 Estudio de la situación actual

El problema de reconocer signos de imágenes es un problema actual y no resuelto de la rama de redes neuronales. Es un problema complicado, especialmente cuando se incluyen signos en movimiento o una gran cantidad de signos reconocibles.

Muchos investigadores han desarrollado aplicaciones para reconocer números, o las letras del abecedario dactilológico del ASL (American Sign Language), pero nadie lo había intentado para el LSE, al menos públicamente hasta la fecha de escritura de esta memoria.

Esto supone que LSETranscriber no tiene ninguna alternativa real en el mercado, especialmente debido a la gran diversidad en el lenguaje de signos, no solo en la multitud de variantes dependiendo del país de residencia, sino ya refiriéndose dentro del propio lenguaje. Por lo tanto, una persona que utilice el LSE lo más normal es que no sea capaz de entender otros lenguajes de signos como el ASL, convirtiendo los productos antes mencionados inservibles para él.

Aun así, en términos de desarrollo estas aplicaciones son muy similares a LSETranscriber, siendo la única diferencia las imágenes que se usan para el entrenamiento. Por ello y por incluir el apartado de evaluación de alternativas, se tendrán en cuenta las siguientes aplicaciones para reconocimiento de signos del lenguaje dactilológico del ASL.

2.3.1 Evaluación de Alternativas

Debido a la similitud entre las diferentes soluciones para problemas similares, se contemplarán solo dos alternativas que utilizan acercamientos diferentes a la hora de la resolución. En primer lugar, observaremos el de Amrita Thakur et al. [1], el cuál usa un método de resolución muy similar al elegido en

LSETranscriber, y por último evaluaremos el de Paulo Trigueiros et al. [2], que propone una solución muy diferente.

Amrita Thakur et al.
Este modelo fue desarrollado por un equipo de Nepal [1], con una solución al problema muy similar a la mía. Se centra en el ASL (American Sign Language), y contiene un dataset de 40.000 imágenes obtenidas a través de la webcam y eligiendo frames correctos, posteriormente realizando dataset expansion. Ya que en el ASL también hay signos dinámicos, eligieron frames estáticos que fueran representativos para el signo, y también eligieron VGG-16 como modelo base para realizar Transfer Learning. Su modelo es diferente al mío, tanto en composición como en arquitectura, y al final obtienen una exactitud de 99.62 %, aunque en la versión final solo predijeron signos de la A a la H, por lo tanto mi acercamiento es más completo.

Una diferencia significativa respecto a mi solución es que ellos reescalaron las imágenes a 50x50 píxeles, mientras que LSETranscriber usa 150x150 píxeles. También mencionar que utilizan imágenes para predecir en los que la región de la mano ya ha sido extraída de la imagen original.

Paulo Trigueiros et al.
Este proyecto [2] está destinado al reconocimiento de signos del Lenguaje de Signos Portugués. La ventaja de este lenguaje es que todos los signos pueden ser correctamente predichos de manera estática, por lo tanto pueden reconocer el alfabeto entero. A pesar de ello, el sistema asume que el usuario está a una distancia predefinida de la cámara, en un interior y en un perímetro definido. Utilizan OpenCV para obtener la entrada de la cámara, y realizan las perdiciones con un SVM (support vector machine) para clasificación múltiple.

Reescalan las imágenes a 16x16, lo cual pueden permitirse gracias a las asunciones de la entrada, y solo realizan el entrenamiento en las cinco vocales, obteniendo una exactitud del 99.6 %. Mencionar que este sistema no es en tiempo real.

A pesar de obtener buenos resultados, LSETranscriber pretende predecir todos los signos posibles con un acercamiento en el que el usuario no tenga restricciones de posicionamiento ni localización, para ser lo más usable posible.

Ahora observaremos diferentes tecnologías que podrían haber sido usadas para la realización de LSETranscriber, y las evaluaremos con las que fueron elegidas definitivamente, comparando sus ventajas e inconvenientes, explicando el por qué de no haberlas elegido.

Framework para deep-learning
Hay varios frameworks que podía haber utilizado para el desarrollo de LSETranscriber. Explicaré por qué elegí Tensorflow-Keras por encima de PyTorch, siendo estos tres las mejores opciones a elegir.

PyTorch [10] es un marco de trabajo para el aprendizaje automático. Es utilizado moderadamente, y es de código libre. Tiene toda la funcionalidad requerida para realizar LSETranscriber, pero no fue elegido por varias razones. Tensorflow es desarrollado y mantenido por Google, lo cual significa que tiene una documentación muy buena. Hay muchos tutoriales y siendo el framework de aprendizaje automático más usado, tiene una gran comunidad detrás para resolver dudas. Además, tiene muchos modelos disponibles para usar en Transfer Learning, lo cual es vital para la solución que se propone con LSETranscriber. También tener en cuenta que Tensorflow provee con visualizaciones mucho mejores, lo cual tiene mucha importancia, especialmente ya que no tengo gran experiencia en el campo del aprendizaje automático.

Respecto al uso de Keras sobre PyTorch tiene una explicación muy sencilla. Tensorflow viene por defecto instalado con Keras, por lo tanto está optimizado para usarse en conjunto.

Framework para la aplicación web
En este apartado explicaré por qué elegí React-Redux por encima de Django, siendo estas dos tecnologías las que consideré más apropiadas para LSETranscriber.

Django [3] es un marco de trabajo para realizar desarrollo web con Python. Era una alternativa muy tentativa, ya que todo el código del modelo está escrito en Python. No fue elegido principalmente por dos motivos. Uno de ellos siendo que no tengo conocimiento de Django, por lo tanto realizar una aplicación web con React era la alternativa más fácil, sabiendo que no me iba a llevar mucho tiempo y que el resultado final iba a ser óptimo. La otra es que al tener que crear una API REST con el código del modelo, la facilidad que hubiera generado el tener el modelo cargado en la propia aplicación web siendo todo en Python desaparece, por lo tanto React era una alternativa mucho más viable.

Podría seguir listando marcos de trabajo para desarrollo web, pero todos caen el el mismo saco que Django. Al final del día React hace un trabajo más que suficiente para el objetivo de este desarrollo, y es un entorno que conozco.

3 Aspectos teóricos

Aquí se encontrarán los conceptos teóricos y las diferentes herramientas usadas para el desarrollo de LSETranscriber.

3.1 Conceptos

En esta sección describiré los diferentes conceptos teóricos necesarios para entender como fue el desarrollo de LSETranscriber, entre los que se incluyen principalmente conceptos de aprendizaje automático y redes neuronales, siendo este una parte troncal del proyecto y la más complicada de este.

3.1.1 Aprendizaje automático y Redes neuronales

El aprendizaje automático es una rama de la inteligencia artificial que persigue la resolución de problemas utilizando una máquina que "aprende" de los datos. Es usado en una gran amplitud de problemas hoy en día, desde recomendaciones de Netflix hasta reconocimiento facial.

Hay diferentes algoritmos para conseguir aprendizaje automático, pero para este proyecto nos interesan especialmente las redes neuronales.
Por lo tanto, las redes neuronales son un subset del aprendizaje automático que simula el funcionamiento de las neuronas del cerebro humano para resolver problemas.

La unidad básica en una red neuronal es la neurona. Una neurona es un nodo que dispone de varias entradas (ya sean variables del problema o salidas de neuronas anteriores), una entrada de sesgo, un vector de pesos y una salida. La neurona realiza cálculos con los diferentes pesos y entradas de los que obtiene un valor. Posteriormente, en base a una función de activación la salida será decidida.
Una función de activación decide como se transforma la suma ponderada de las entradas a una salida.

El concepto de red neuronal viene debido al uso de una gran cantidad de neuronas. Esto se debe a que con un perceptrón solo se podrían solucionar problemas linealmente separables, lo que significa que si representamos todos los elementos en una gráfica, una sola línea recta sería capaz de dividir los elementos en dos grupos correspondientes.

Por ejemplo, un solo perceptrón podría resolver el siguiente problema:

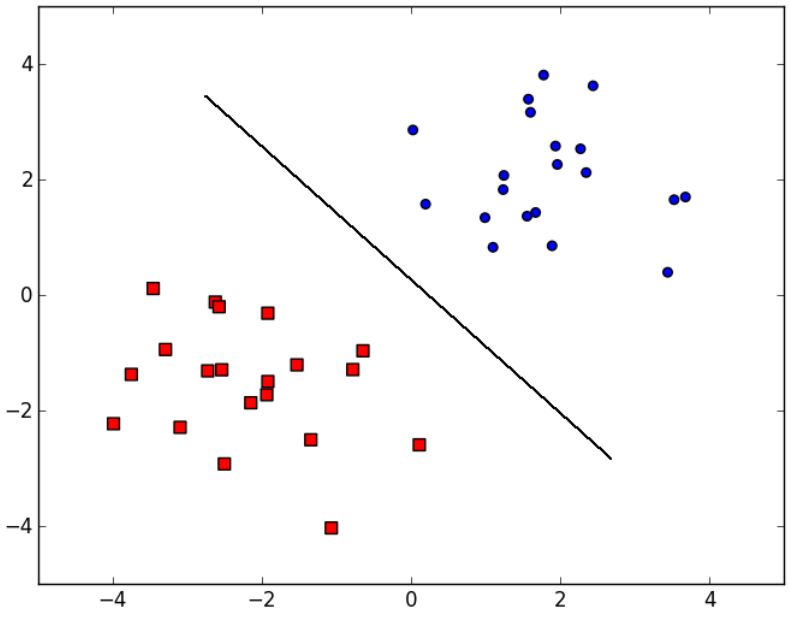


Figura 1: Problema linearmente separable

Debido a que en la realidad los elementos no suelen estar tan perfectamente distribuidos, o debido a que como en el caso de LSETranscriber, no hay solo dos elementos que queramos diferenciar, se necesita el uso de una gran cantidad de neuronas.

Para usar varias neuronas, la práctica más común es dividirlas en capas. Tenemos una capa de entrada, la cual se corresponde a las variables del problema, varias capaz ocultas y una capa de salida. Originalmente se tendía al uso de pocas capas ocultas con muchas neuronas en cada capa, pero la práctica demostró que era más eficiente el uso de muchas capas con menores neuronas, lo que se conoce como Redes Neuronales Profundas.

Diferentes capas realizan diferentes operaciones, las cuales se van juntando para realizar la predicción final. Por ejemplo en el caso de clasificación de imágenes, la finalidad de las primeras capas sería lo que se conoce como "feature extraction"(esto es, extracción de características). En una imagen hay mucho ruido alrededor del elemento que queremos clasificar. Es importante que el modelo sea capaz de ignorar este ruido y centrarse en los píxeles que realmente contienen información relevante para realizar la predicción.

El proceso por el cual se le enseña a un modelo a clasificar las imágenes se conoce como entrenamiento del modelo. En esta fase, se construye un dataset, consistente del mayor número de imágenes con características diferentes de los objetos que queremos clasificar. Es importante que cada imagen tenga una etiqueta que represente la clase correcta, y evitar tener ruido en las imágenes, o imágenes mal clasificadas.

Una vez que tenemos un dataset, este es proporcionado al modelo, que empezará a realizar iteraciones sobre él. En estas iteraciones, el modelo cogerá cada imagen una a una y realizará predicciones sobre ellas. Después, consultará la etiqueta y observará si su predicción ha sido correcta. Solo en el caso de que no lo fuera, los pesos son actualizados.

Como comenté antes, el perceptrón tiene un vector de pesos, que básicamente deciden la importancia de determinadas entradas. Al ser la predicción incorrecta, a través de un algoritmo de "Back propagation"(propagación hacia atrás en español), cada perceptrón corrige ligeramente sus pesos y comunica el error a las neuronas anteriores, que realizan lo mismo.

Después de pasar por todas las imágenes, estos pesos han sido actualizados muchas veces, avanzando hacia un estado en el que todas o la gran mayoría de las imágenes son clasificadas correctamente. Después de realizar muchas iteraciones, si el modelo está construido correctamente, se podrá observar un incremento en la exactitud de las predicciones.

Un factor a tener en cuenta a la hora de entrenar un modelo es el sobreajuste. Un modelo sobreajustado implica que los pesos han sido actualizados demasiado acorde con las imágenes que se han usado durante el entrenamiento. A priori no parece ser un problema, pero errores en predicción aparecerán en imágenes que el modelo no ha visto nunca. Es importante evitar el sobreajuste. Esto significa que la exactitud que hemos estado observando puede ser engañosa, ya que esta se corresponde al entrenamiento.

Para ello se realiza la validación. Normalmente, cuando se crea un dataset, las imágenes se particionan en una proporción de 80-20, siendo 80 para entrenamiento y 20 para validación. De esta forma, entrenamos el modelo en los datos de entrenamiento, y luego calculamos su efectividad con los datos de validación. Es una forma eficaz de evitar el sobreajuste, pero hay que tener cuidado para no introducir sesgo en la separación de los datos.

3.1.2 Conceptos sobre el entrenamiento

Debido a la complejidad que supone el entrenamiento de un modelo, voy a explicar algunos conceptos con el objetivo de ahorrar explicaciones cuando se mencionen posteriormente.

Los elementos que se van a contemplar son:

- Tasa de aprendizaje
- Transfer learning
- Tipos de capas de neuronas
- Función de activación
- Función de pérdida
- Optimizadores

Tasa de aprendizaje
La tasa de aprendizaje controla el cambio de los pesos de las neuronas cuando estas aprenden. Como se comentó en el apartado anterior, el aprendizaje de las neuronas se realiza a través del algoritmo de propagación hacia atrás, el cual actualiza el peso de las neuronas en función a la dimensión del error en la predicción.

Si dejáramos que cambiara el valor de los pesos a su gusto, esto produciría que los pesos se actualizaran a un valor que permitiera clasificar a la perfección la imagen actual, pero la neurona no estaría aprendiendo en base al dataset entero. Por ello se introduce el concepto de tasa de aprendizaje.

La tasa de aprendizaje pondera el valor que se quiere utilizar para modificar los pesos, reduciendo esta modificación para que se realice aprendizaje real y útil para todas las posibles entradas. Por lo tanto, es un valor entre 0 y 1, que se ajustará en base a prueba y error observando el sobreajuste de las diferentes combinaciones.

Como se explicará en el siguiente apartado, al utilizar un modelo base y la técnica de transfer learning, tenemos una cantidad de neuronas muy grande, por lo tanto nos interesa una tasa de aprendizaje muy pequeña.

Transfer learning
Crear un modelo desde cero, especialmente si el problema es relativamente complejo, es una tarea muy exhaustiva, tanto en tiempo de diseño, prueba y error, como en la potencia computacional requerida para entrenar modelos grandes. Es por ello que surge la técnica de transfer learning. Esto nos permite utilizar modelos entrenados previamente, construir nuestro pequeño modelo encima de ellos y que el rendimiento sea bueno, manteniendo un tiempo de entrenamiento bajo. Estos modelos contienen multitud de capas con millones de neuronas, previamente entrenados en ordenadores con mucha potencia. Los más conocidos son Inception, creados por Google, VGG, MobileNet...

Todos estos modelos fueron creados para el concurso de ImageNet de clasificación de imágenes. Este concurso consiste en entrenar modelos para predecir imágenes que pueden ser de una gran cantidad de clases. Es un problema muy complicado, y a pesar de ello estos modelos obtienen una exactitud de $>90\,\%$ en poco tiempo.

Estos modelos son públicos, por lo tanto utilizando los pesos ya calculados, seguimos una serie de pasos para utilizar nuestro modelo encima de ellos.

El primer paso es eliminar la última capa del modelo, la que decidía a cuál de las clases del concurso de ImageNet pertenecía la imagen.

A continuación, colocaremos nuestro modelo, que usará de entrada la salida de la penúltima capa (ahora la última). Necesitaremos entrenar nuestro modelo, por lo que congelaremos los pesos del modelo base para predecir sobreajuste, y entrenaremos en pocos epochs con una tasa de aprendizaje muy baja. Una vez veamos que nuestro modelo comienza a sobreajustar, pararemos el entrenamiento y pasaremos a la fase que se conoce como fine tunning.

En esta fase, descongelaremos los pesos del modelo base y realizaremos muy pocos epochs con una tasa de aprendizaje todavía inferior a la de entrenamiento.
Estos pasos permitirán que nuestro simple modelo, con poco tiempo de entrenamiento obtenga unos resultados muy prometedores.

Tipos de capas de neuronas
Para el entrenamiento del modelo, Tensorflow 3.2.2 proporciona muchas capas de neuronas ya configuradas, cada una de las cuales tiene un propósito. Ya que hay infinidad de ellas, se comentarán las que tuvieron más relevancia en el desarrollo de LSETranscriber.

1. **Dropout layer**: Esta capa nos permite eliminar parte de las salidas de la capa anterior. Por ejemplo, si tenemos una capa con 12 salidas, y la capa siguiente tiene 2 neuronas, implicaría 6 pesos de entrada para cada neurona. Ya que sospechamos que hay sobreajuste, no nos interesa que todas las salidas se utilicen como entrada. Por ello, podemos elegir un dropout de 0.5 lo que implicaría que la mitad de ellas serán eliminadas. Por lo tanto, tendríamos 2 neuronas con 3 entradas cada una. El factor de dropout es entre 0 y 1 como si fuera un porcentaje, y los pesos que se eliminan son aleatorios.

2. **Dense layer**: Esta capa consiste en neuronas formando un grafo denso, lo cual implica que el número de aristas es cercano al máximo. Esta interconexión entre neuronas genera sobreajuste,

razón por la cual es usual encontrar una capa de Dropout justo después de ella.

3. **Flatten layer**: Esta capa nos permite reducir la matriz de entrada de dimensiones NxN a un array de longitud NxN. Simplifica las posteriores operaciones con los valores de entrada iniciales para no generar matrices de dimensiones exageradas.

Función de activación
La función de activación decide el valor que la neurona va a generar como salida. Esta función toma como entrada el valor de los cálculos de la neurona respecto a sus entradas y sus pesos, y decide la salida.

Funciones de activación comunes son la Función Signo, que devuelve 1 si el valor es positivo, y 0 en cualquier otro caso, la función Sigmoide, que devuelve un número en el rango (0,1) donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0, función ReLU, función softmax...

Función de pérdida
La función de pérdida determina cómo de lejos yace la predicción respecto al valor actual de la entrada. Es el valor que se utilizará posteriormente en el optimizador, que nos ayudará a calcular el valor que se usará en la propagación hacia atrás para determinar cuánto deberán cambiar las neuronas sus pesos para predecir la imagen actual correctamente.

No voy a entrar en detalle en ninguna de ellas, pero las más típicas son sparse categorical crossentropy, maximum likelihood, mean squared error loss... Por la naturaleza del problema de clasificación de LSETranscriber, se utilizará sparse categorical crossentropy (Entropía Cruzada Categórica Dispersa en español).

Optimizadores
Un optimizador es una función o un algoritmo que determina cuánto deberán cambiar los pesos de las neuronas para mejorar la exactitud. Utiliza el valor obtenido en la función de pérdida, y su objetivo es evitar mínimos relativos en el error y continuar hacia el mínimo absoluto. Hay varios optimizadores, en LSETranscriber se utiliza RMSprop y Adam.

3.2 Tecnologías

En esta sección explicaré las tecnologías más importantes usadas en el desarrollo de LSETranscriber.

3.2.1 Tensorflow

"Tensorflow es una plataforma de código abierto de extremo a extremo para el aprendizaje automático" [13]. Es creada y mantenida por Google, y ha sido y es usada en multitud de proyectos de Machine Learning. Un ejemplo es Inception, un modelo de Google creado con Tensorflow para el reto de ImageNet de clasificación de imágenes, en el que obtuvo una gran puntuación.

En LSETranscriber se usa la versión 2.7 para crear y entrenar el modelo que se encarga de predecir los signos en las imágenes.

3.2.2 Keras

Keras es una API escrita en Python para TensorFlow, lo que hace que la combinación de ambas tecnologías sea increíblemente potente y muy usado. Antes no formaban parte del mismo paquete pero ahora Keras viene integrado en la instalación de TensorFlow.

3.2.3 React-Redux

"React es una biblioteca de JavaScript para construir interfaces de usuario" [11], desarrollada por Facebook y es una de las principales bibliotecas de lado de cliente usadas hoy en día. Se basa en el concepto de "single-page", lo que significa que la mayoría de la carga de elementos se hace al entrar en la aplicación, pero a partir de ahí la navegación es automática. Además, la mayoría de computaciones se hacen en el lado del cliente, relajando el número de peticiones y cargas del servidor.

Se basa en la creación de componentes, que tienen un estado, su propio método para renderizarse y una jerarquía. Gracias a la comunidad, hay multitud de bibliotecas para React que facilitan mucho el desarrollo de una aplicación web, entre ellas está Redux.

"Redux es un contenedor predecible del estado de aplicaciones JavaScript." [12]
Es una forma de centralizar el estado de la aplicación y quitarlo de los componentes particulares. A través de una "store" centralizada, todos los componentes son capaces de acceder al estado actual de la aplicación y modificarlo, evitando tener que compartir el estado entre componentes que en jerarquías grandes es una ardua tarea. También facilita cambiar el estado a través de llamadas no asíncronas, como en el caso de LSETranscriber son las llamadas que se realizan a la RestAPI para predecir un nuevo signo. De esta forma, al obtener la predicción esta es añadida automáticamente en pantalla.

Ambas tecnologías se usan para crear la página web de LSETranscriber. Las versiones utilizadas son React 17.0.2 y Redux Toolkit 1.6.2.

3.2.4 Jest

"Jest es un Framework de Testeo para JavaScript, compatible con proyectos de React"[7]. Permite el uso de mocks para elementos externos al test, lo que facilita enormemente la tarea de testear las interacciones con las predicciones devueltas por la restAPI sin necesidad de perder tiempo y evitando saturación haciendo la llamada.

También provee Code Coverage, lo que permite saber cuantas líneas de código están siendo ejecutadas al realizar los tests, además de si se están evaluando las diferentes ramas de las condiciones.

Se usa para realizar los tests unitarios de la aplicación web, en su versión 26.6.0.

3.2.5 Gatling

Gatling es un programa de código abierto para realizar tests de carga. [6] A través de realizar una tarea y grabándola, permite simular cualquier número de usuarios concurrentes realizándola. Es usada en LSETranscriber para comprobar que capacidad de predicciones por segundo es capaz de realizar la restAPI ante una situación de estrés.

3.2.6 Flask-RESTX

"Flask RESTX es una extensión para Flask que añade soporte para montar REST APIs rápidamente". [5]
Está escrita en Python y permite crear endpoints, a los que se puede acceder a través de diferentes peticiones HTTP. Es un fork de Flask, y es la biblioteca usada para realizar la restAPI de LSETranscriber. Básicamente, consta de una dirección a la que realizar peticiones y contiene toda la lógica para enviar la imagen al modelo y obtener una predicción.

La versión utilizada es Flask 2.0.2 y flask-restx 0.5.1.

4 Planificación del proyecto y resumen de presupuestos

En esta sección se introducirá la planificación inicial del proyecto, siendo esta una estimación del tiempo que duraría. También se hará un resumen del presupuesto confeccionado para este proyecto, pudiendo encontrar el presupuesto detallado al final de la memoria.

4.1 Planificación

Este proyecto debe ser completado antes del 11 de Julio. Al haber elegido tema, pude empezar a trabajar el 1/10/2021, lo que significa que el desarrollo puede durar como máximo 9 meses.

El trabajo será dividido en varios módulos, para facilitar su estimación a la hora de la planificación, y poder establecer metas razonables y no encontrarse con tareas inesperadas que alarguen el desarrollo.

La primera separación de las tareas serán las que se considerarán hitos posteriormente en el desarrollo:

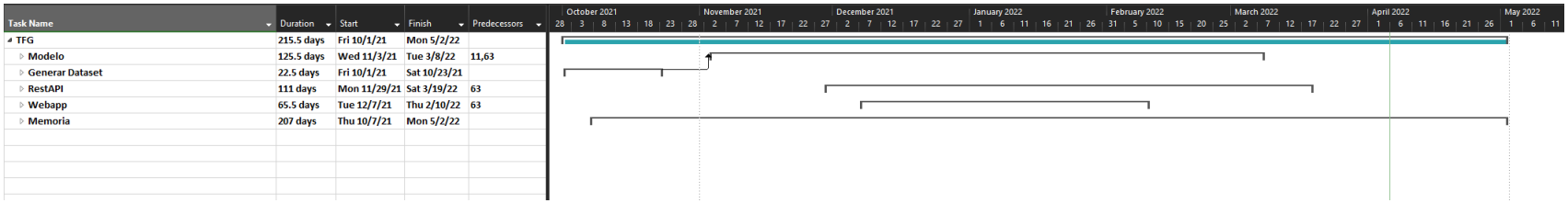
- Aplicación web
- Servicio web
- Creación del dataset
- Creación del modelo
- Redacción de la memoria

Estas tareas serán refinadas y divididas hasta que no encuentre más tareas en un primer acercamiento. Aún así, cuento con que la estimación no sea perfecta, de la misma forma que muy probablemente aparezcan nuevas tareas durante el desarrollo del proyecto, por lo que se intentará limitar el alcance para dejar aproximadamente 1 mes de margen.

Después, se marcarán las precedencias de todas las tareas y se empezarán a planificar con un orden lógico, en un calendario de 2h de trabajo al día.

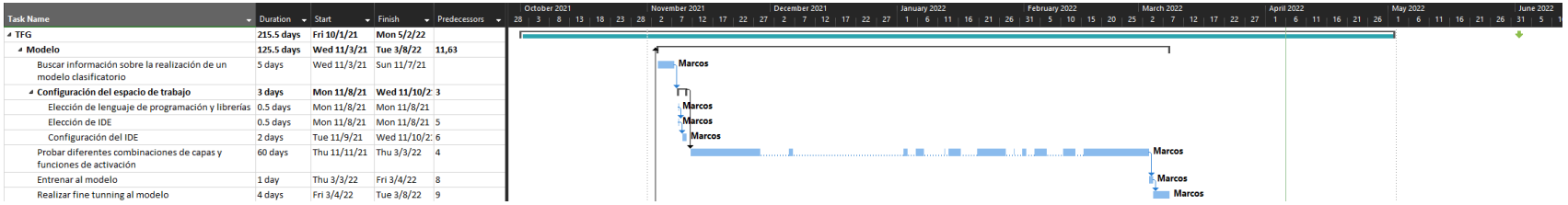
Para realizar la planificación y obtener problemas a la hora de estimar, asignación de las tareas o tiempo disponible, se usará Microsoft Project. Este programa también me permitirá obtener un diagrama de Gantt con la planificación, así como establecer una línea base y añadir el trabajo que vaya realizando a lo largo del tiempo, hasta la generación de la planificación final.

El resultado de esta planificación inicial es un diagrama de Gantt como se muestra a continuación:



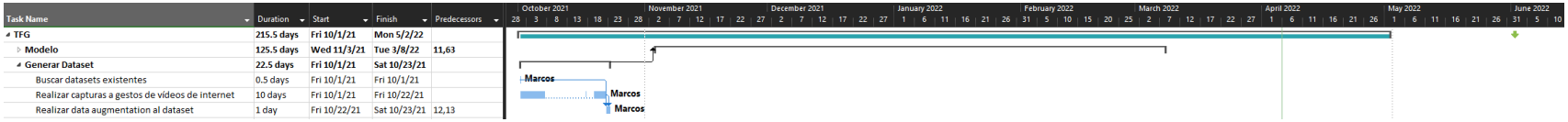
Task Name	Duration	Start	Finish	Predecessors
TFG	215.5 days	Fri 10/1/21	Mon 5/2/22	
Modelo	125.5 days	Wed 11/3/21	Tue 3/8/22	11,63
Generar Dataset	22.5 days	Fri 10/1/21	Sat 10/23/21	
RestAPI	111 days	Mon 11/29/21	Sat 3/19/22	63
Webapp	65.5 days	Tue 12/7/21	Thu 2/10/22	63
Memoria	207 days	Thu 10/7/21	Mon 5/2/22	

Figura 2: Diagrama Gantt de los módulos principales



Task Name	Duration	Start	Finish	Predecessors
TFG	215.5 days	Fri 10/1/21	Mon 5/2/22	
Modelo	125.5 days	Wed 11/3/21	Tue 3/8/22	11,63
Buscar información sobre la realización de un modelo clasificatorio	5 days	Wed 11/3/21	Sun 11/7/21	
Configuración del espacio de trabajo	3 days	Mon 11/8/21	Wed 11/10/2	3
Elección de lenguaje de programación y librerías	0.5 days	Mon 11/8/21	Mon 11/8/21	
Elección de IDE	0.5 days	Mon 11/8/21	Mon 11/8/21	5
Configuración del IDE	2 days	Tue 11/9/21	Wed 11/10/2	6
Probar diferentes combinaciones de capas y funciones de activación	60 days	Thu 11/11/21	Thu 3/3/22	4
Entrenar al modelo	1 day	Thu 3/3/22	Fri 3/4/22	8
Realizar fine tunning al modelo	4 days	Fri 3/4/22	Tue 3/8/22	9

Figura 3: Diagrama Gantt de las tareas del modelo



Task Name	Duration	Start	Finish	Predecessors
TFG	215.5 days	Fri 10/1/21	Mon 5/2/22	
Modelo	125.5 days	Wed 11/3/21	Tue 3/8/22	11,63
Generar Dataset	22.5 days	Fri 10/1/21	Sat 10/23/21	
Buscar datasets existentes	0.5 days	Fri 10/1/21	Fri 10/1/21	
Realizar capturas a gestos de vídeos de internet	10 days	Fri 10/1/21	Fri 10/22/21	
Realizar data augmentation al dataset	1 day	Fri 10/22/21	Sat 10/23/21	12,13

Figura 4: Diagrama Gantt de las tareas del dataset

Task Name	Duration	Start	Finish	Predecessors
TFG	**215.5 days**	**Fri 10/1/21**	**Mon 5/2/22**	
Modelo	**125.5 days**	**Wed 11/3/21**	**Tue 3/8/22**	**11,63**
Generar Dataset	**22.5 days**	**Fri 10/1/21**	**Sat 10/23/21**	
RestAPI	**111 days**	**Mon 11/29/21**	**Sat 3/19/22**	**63**
Buscar información sobre tecnologías para hacer la RestAPI	0.5 days	Mon 11/29/21	Mon 11/29/21	
Configuración del espacio de trabajo	**2 days**	**Mon 11/29/21**	**Wed 12/1/21**	**16**
Elección de lenguaje de programación y librerías	0.5 days	Mon 11/29/21	Mon 11/29/21	
Elección de IDE	0.5 days	Tue 11/30/21	Tue 11/30/21	18
Configuración del IDE	1 day	Tue 11/30/21	Wed 12/1/21	19
Crear un endpoint que soporte peticiones POST con imágenes	2 days	Wed 12/1/21	Sat 12/4/21	17
Cargar el modelo	1 day	Thu 3/10/22	Sun 3/13/22	2
Identificar la mano en la imagen	2 days	Sat 12/4/21	Fri 1/28/22	21
Mandar la imagen al modelo para su predicción	3 days	Sun 3/13/22	Fri 3/18/22	22
Devolver la predicción al usuario	1 day	Sat 3/19/22	Sat 3/19/22	24

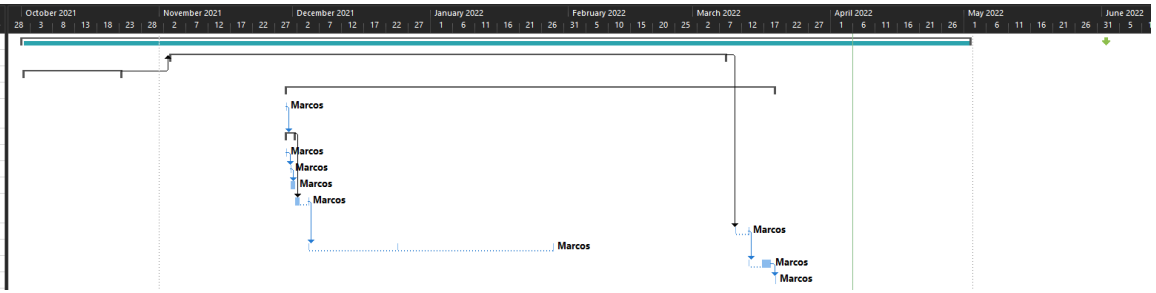


Figura 5: Diagrama Gantt de las tareas de la restAPI

Task Name	Duration	Start	Finish	Predecessors
TFG	**215.5 days**	**Fri 10/1/21**	**Mon 5/2/22**	
Modelo	**125.5 days**	**Wed 11/3/21**	**Tue 3/8/22**	**11,63**
Generar Dataset	**22.5 days**	**Fri 10/1/21**	**Sat 10/23/21**	
RestAPI	**111 days**	**Mon 11/29/21**	**Sat 3/19/22**	**63**
Webapp	**65.5 days**	**Tue 12/7/21**	**Thu 2/10/22**	**63**
Configuración del espacio de trabajo	**2.5 days**	**Tue 12/7/21**	**Thu 12/9/21**	
Decidir tecnologías para realizar la aplicación web	0.5 days	Tue 12/7/21	Tue 12/7/21	
Instalar dependencias y programas correspondientes	0.5 days	Tue 12/7/21	Tue 12/7/21	28
Elección de IDE	0.5 days	Wed 12/8/21	Wed 12/8/21	29
Configuración del IDE	1 day	Wed 12/8/21	Thu 12/9/21	30
Realizar un mockup de las vistas	1 day	Fri 12/10/21	Fri 12/10/21	
Implementar vista principal	4 days	Sat 12/11/21	Tue 12/14/21	32
Permitir al usuario cargar imágenes	1 day	Wed 12/15/21	Wed 12/15/2	33
Permitir al usuario obtener una predicción de las imágenes	1 day	Thu 12/16/21	Thu 12/16/21	34
Implementar TTS para las imágenes predichas	1 day	Fri 12/17/21	Fri 12/17/21	35
Conectar con la restAPI y mandarle las imágenes del usuario	4 days	Sat 12/18/21	Thu 2/3/22	34
Realizar barra de navegación	1 day	Sun 1/30/22	Mon 1/31/22	33
Diseñar logo	1 day	Sun 12/19/21	Sun 12/19/21	
Diseñar look & feel de la aplicación	3 days	Mon 12/20/21	Wed 2/9/22	32
Generar vista de ayuda	2 days	Tue 12/21/21	Thu 2/10/22	32
Generar vista con información sobre la aplicación	1 day	Wed 12/22/21	Wed 12/22/2	32

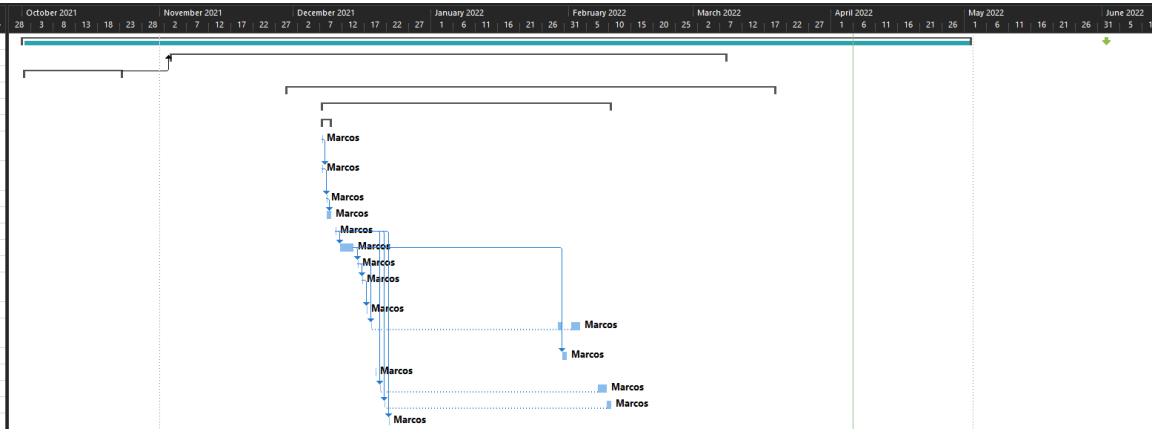


Figura 6: Diagrama Gantt de las tareas de la webapp

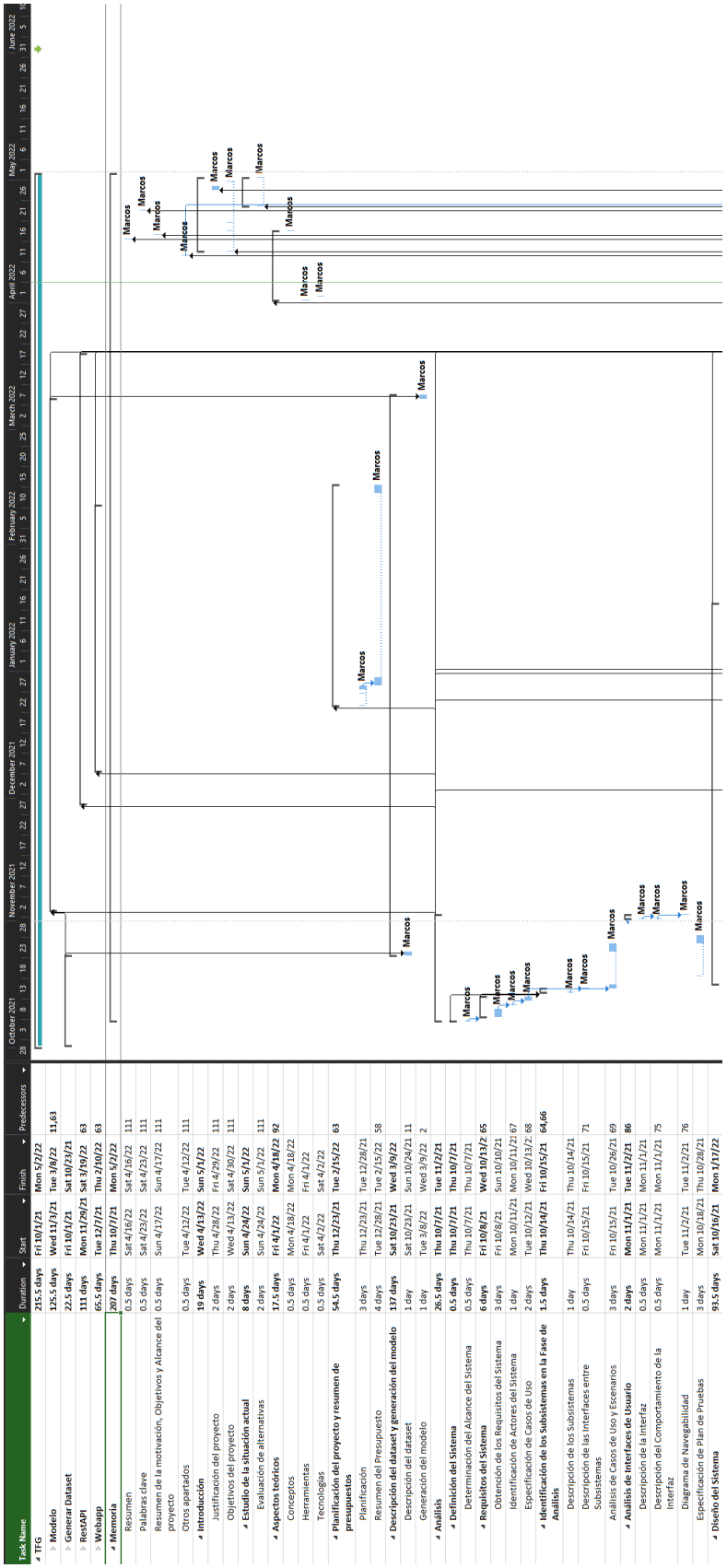


Figura 7: Diagrama Gantt de las tareas de la memoria 1

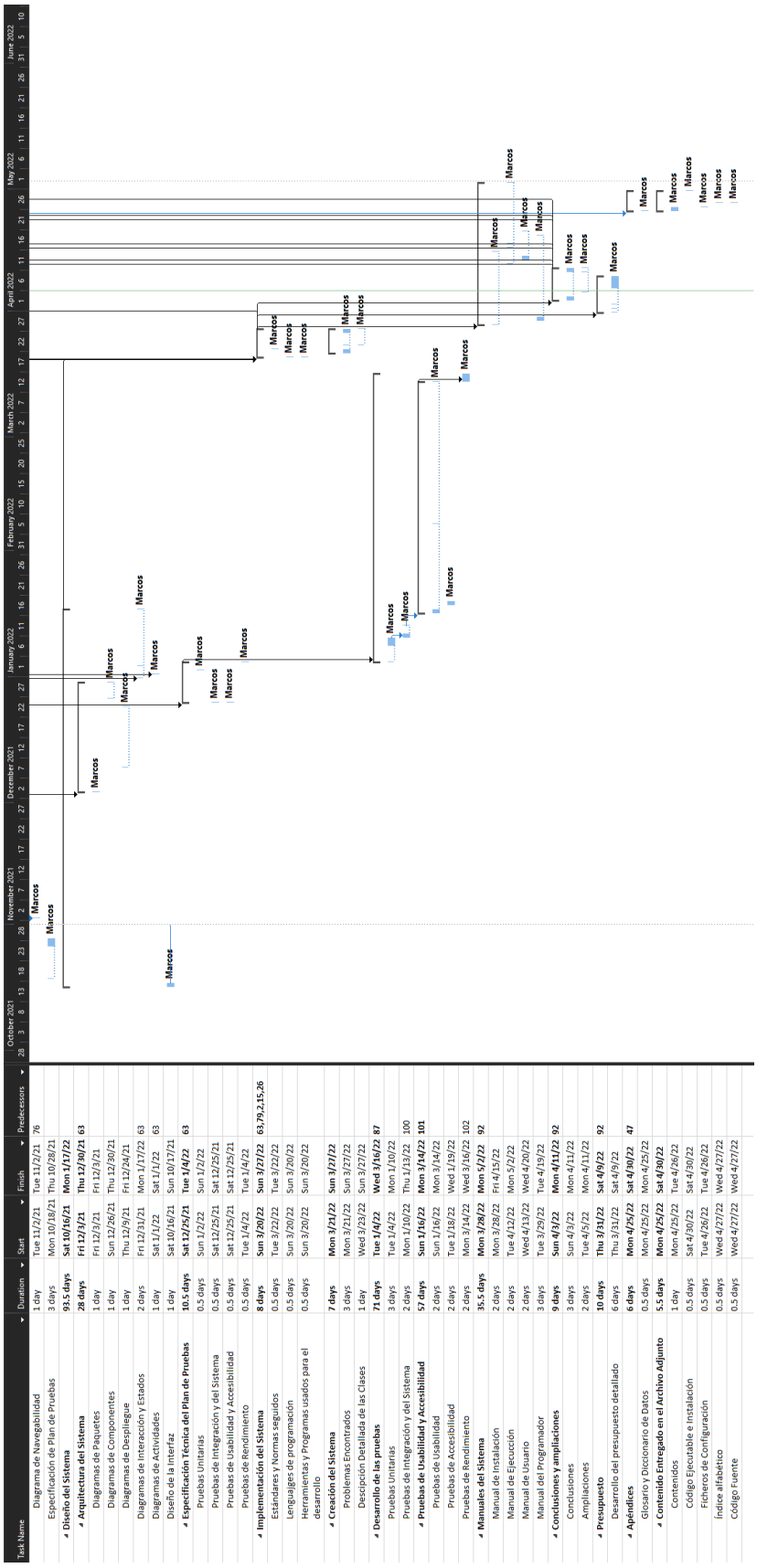


Figura 8: Diagrama Gantt de las tareas de la memoria 2

4.2 Resumen del Presupuesto

El coste total del proyecto es de 17.208,00€. Está desglosado en 3 partidas, que se describirán a continuación:

- Estudios preliminares: Aquí se encuentran todas las tareas relacionadas con el estudio y diseño del sistema, siendo estas preliminares al desarrollo.
- Documentación: En esta partida se encuentran todas las tareas de la redacción de la memoria
- Implementación: Partida con los costes de las tareas relacionadas con el desarrollo y las pruebas

Teniendo en cuenta estas partidas, el presupuesto final es:

Presupuesto de costes		
Cod.	***Partida***	***Total***
1	Diseño preliminar	576,00 €
2	Documentación	5.832,00 €
3	Implementación	10.800,00 €
	TOTAL	**17.208,00 €**

Figura 9: Presupuesto de costes

Se explicarán las diferentes partidas y la definición de la empresa en profundidad en el apartado del presupuesto detallado 12.

5 Descripción del dataset y generación del modelo

En este apartado se explicará como se obtuvo un modelo capaz de predecir signos del Lenguaje de Signos Español con una exactitud suficientemente elevada como para hacer el proyecto viable.

5.1 Descripción del dataset

Debido a la inexistencia de un dataset previo de signos del alfabeto dactilológico para LSE, fue creado manualmente.
Tomando como referencia el dataset del MNIST para el ASL [9] fue desarrollado uno específico para LSE.
A través de diversos vídeos de YouTube donde gente realizaba los gestos, y diversos vídeos que solicité a conocidos, realicé 3 capturas por signo. Generalmente se siguió la pauta de una captura al comenzar el signo, una en la parte "estable", y finalmente una cuando se iba a transicionar hacia el siguiente signo.

Siendo 20 vídeos en total y 22 signos, esto resulta en 1320 imágenes. Dado que son pocas imágenes para entrenar un modelo, se utilizaron técnicas de Data Augmentation y Dataset Expansion para aumentarlas.

Primero, todas las imágenes fueron reescaladas a 150x150 píxeles, el cual será el tamaño del entrenamiento, centrado sobre la zona de la mano.

Después se aplicaron diferentes filtros a la imagen, como robidoux, Mitchell, catrom, spline y hermite, generando cada filtro una imagen nueva.

Sobre todas estas imágenes, se generaron nuevas aplicando diferentes aumentos y decrementos de brillo y contraste, la función Poisson de ruido y rotación de 3° en ambas direcciones.

Después de estas transformaciones el número de imágenes es más de 30000.

Aún así, no podemos comparar este dataset con uno que contenga 30000 imágenes diferentes reales. Claramente realizar esta técnica iba a provocar que tuviera que estar más pendiente del sobreajuste durante el desarrollo del modelo, pero era necesario aplicarla ya que no existe otra forma viable de obtener más datos en un marco de tiempo razonable.

De cara al entrenamiento, estas imágenes fueron organizadas en carpetas con el nombre de la clase que son. Por ejemplo, todas las imágenes correspondientes al signo 'A' fueron colocadas dentro de una carpeta llamada 'A', y así con el resto.

Para realizar un entrenamiento evitando el sobreajuste, los datos fueron particionados en tres grupos. Un primer grupo de train, compuesto por el 70 % de las imágenes, que fue usado para entrenar el modelo. Este realizaría una predicción sobre cada una de estas imágenes, y en el caso de ser incorrecta cambiaría los pesos para que la próxima vez sea más probable que acierte. El 20 % de las imágenes fue destinada a test. Estas son imágenes que el modelo predice cada iteración después de haber pasado por las de train, pero esta vez no corrige sus pesos en el caso de que se equivoque. Son usadas para realizar comprobaciones de que el modelo no está sobreajustando.

Por último, el 10 % restante fue separado para validación. Estas son imágenes que el modelo predecirá una vez acabado el proceso de entrenamiento al completo. Son como la verdadera prueba de que el modelo no sobreajusta y que se adapta bien a imágenes que no ha visto nunca. Son las imágenes sobre las que basaremos las conclusiones de si el proceso de aprendizaje ha sido realizado correctamente.

5.2 Generación del modelo

Hubo muchas fases de entrenamiento hasta dar con una que produjera resultados convincentes.

El primer acercamiento fue a través de un modelo original compuesto por una capa Flatten 3, seguida de una capa Densa 2 de 1024 neuronas completamente conectada. Después una capa de Dropout 1 al 0.2 y finalmente una capa densa de 19 neuronas con activación softmax que es la que predice la clase.

Rápidamente me di cuenta de que entrenar un modelo desde cero no era viable, a parte de por la falta de poder computacional y tiempo que llevaría, debido a el tamaño muy reducido del dataset. Es pequeño incluso después de aumentarlo, y aún así las imágenes son muy similares, ya que se generan muchas a partir de la misma, por lo que siguiendo este acercamiento el modelo sobreajusta mucho y generaliza muy mal. De hecho ni siquiera obtiene un buen rendimiento en entrenamiento.

Por ello decidí usar Transfer Learning 3.1.2. Esta técnica consiste en usar un modelo previamente entrenado durante mucho tiempo sobre un dataset muy grande.
De esa forma, los pesos de los enlaces entre las neuronas ya están entrenados, y aunque el uso final del modelo no tenga que ver con la intención original, por lo general se comportan de una forma mucho más fiable.

Inicialmente usé el modelo de InceptionV3 de Google, entrenado para el concurso de ImageNet de clasificación de imágenes. Para poder usar el modelo para predecir signos, hay que eliminar la última capa, que es la que predecía para el problema original, y congelar los pesos del modelo base. Esto es debido a que son modelos extremadamente grandes, y modificar los pesos que ya han sido entrenados produciría sobreajuste muy rápido. Estos pesos pueden ser descongelados en un proceso posterior llamado Fine Tunning, que explicaré más adelante.

De esa forma, después de la penúltima capa estaría mi modelo, que consistiría en algo simple ya que la carga la hace Inception. Ya que el modelo que diseñé anteriormente era suficientemente simple, es

el que usé. Esto generaba mejores resultados, pero aun así seguía sin ser suficiente para pasar del 60 % de exactitud. Después de realizar diferentes cambios en mi modelo sin frutos, decidí cambiar el modelo sobre el que haría Tranfer Learning.

Probé con MobileNet, un modelo especialmente rápido para móviles, pero que a cambio sacrifica exactitud y no funcionó nada bien.

Y finalmente llegué al modelo VGG16, otro modelo entrenado para el mismo problema que InceptionV3, usado especialmente para clasificación de imágenes.

La exactitud usando este modelo incrementó automáticamente hasta el 90 %.

Para evitar el sobreajuste que era fácil de observar, al ver como la exactitud del modelo saltaba 20 % cada iteración, reduje la tasa de aprendizaje 3.1.2 a 0.00001. Aún así el entrenamiento es muy rápido, y en 10-15 iteraciones ya había obtenido una exactitud elevada.

El único cambio respecto a mi primer modelo es que la primera capa densa es de 512 neuronas en vez de 1024, ya que no necesito una capa tan grande al usar Transfer Learning, ya que el modelo base tiene muchas neuronas. Usar más neuronas no es una buena práctica especialmente en un dataset tan pequeño. Y otro cambio es que la capa de Dropout es de 0.5, ya que se presentaba bastante sobreajuste.

Con estas configuraciones, todavía se podía observar mucho sobreajuste a partir del 80 % de exactitud, ya que se veía un incremento en la diferencia de exactitud entre los datos de train y los de test.

Para ello, utilicé una herramienta de Keras llamada EarlyStopping, que permite parar el entrenamiento del modelo, aunque las iteraciones provistas no se hayan alcanzado. Puse que cuando la pérdida en validación fuera mayor de 0.001 parara automáticamente. Esto hacía que el modelo parara el entrenamiento sobre el 80 % de la exactitud alcanzada. Para abarcar el 20 % restante, o la mayor parte de él, use la técnica de Fine Tunning.

Es una técnica usada para realizar una mejora final. En el caso de Transfer Learning esto consiste en un segundo entrenamiento durante muy pocos epochs, pero con los pesos del modelo base descongelados. Pocos epochs ya que como se comentó antes al descongelar los pesos se produce un sobreajuste muy rápido.

Al usar esta técnica el modelo obtuvo un 96 % de exactitud, siendo las letras más conflictivas las que más se parecen, lo que tenía sentido. Como esto satisfacía mi intención con el modelo, el entrenamiento paró ahí.

A través de las diferentes pruebas, concluí que la mejor combinación de optimizadores 3.1.2 era RMSprop para el primer entrenamiento, y Adam para el Fine Tunning. De la misma forma, en la capa Densa de 512 neuronas usé ReLU como función de activación.
La función de pérdida 3.1.2 utilizada fue entropía cruzada categórica dispersa.

Finalmente, los resultados obtenidos son:

```
Accuracy: 0.9440267335004177
Each accuracy: [0.80952381 1.         1.         1.         0.99206349 1.
 1.         0.98412698 1.         0.89285714 0.99603175 1.
 0.6031746  0.98809524 1.         0.99603175 0.76984127 0.9047619
 1.        ]
```

Figura 10: Resultados del entrenamiento

La lista contiene el porcentaje de acierto del modelo sobre los datos de validación para cada letra en particular. Es relativamente complicado entender que letra es cuál, por lo que lo observaremos mejor en un gráfico de barras:

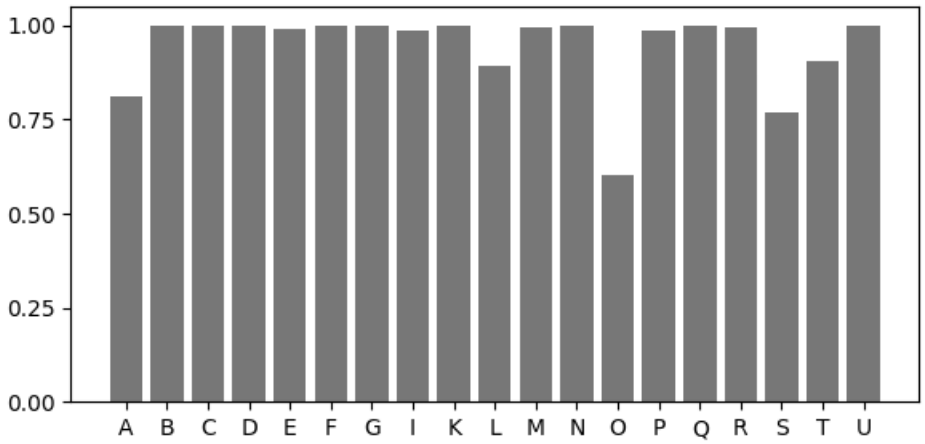


Figura 11: Gráfico de barras con los resultados del entrenamiento

Como se puede observar en la figura, para la mayoría de las letras el modelo tiene una exactitud muy buena. Son ciertas letras las que más fallan: la 'A', la 'O', la 'S' y la 'T'.

Por lo tanto, vamos a ver cuál fue la predicción dada por el modelo para cada una de las imágenes que correspondían a las clases anteriores.

Para la 'A':

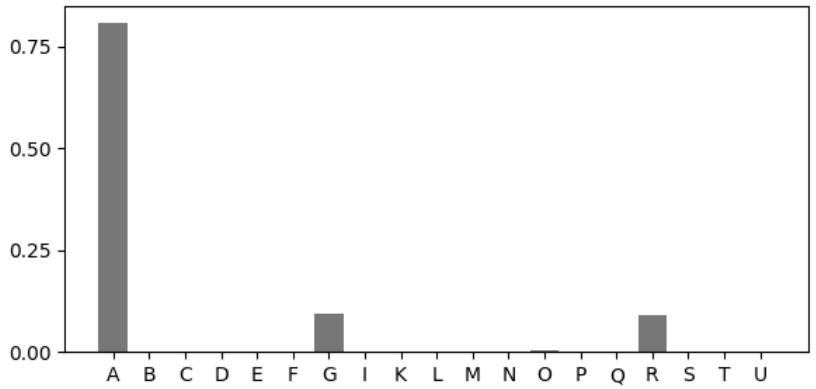


Figura 12: Resultados para imágenes del signo de A

La 'A' es mayoritariamente confundida con la 'G' y con la 'R'.
Para la 'O':

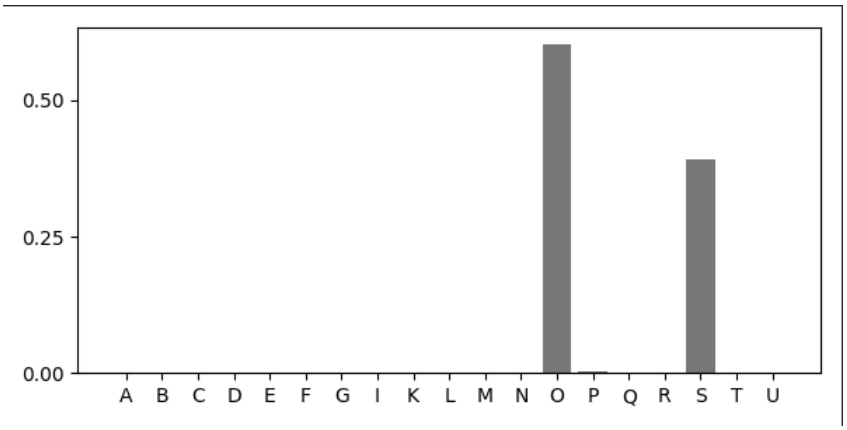


Figura 13: Resultados para imágenes del signo de O

La 'O' se confunde muy a menudo con la 'S'.

Para la 'T':

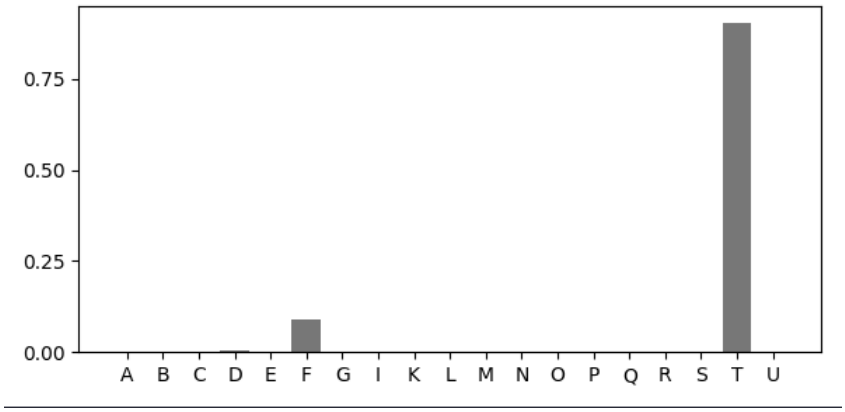


Figura 14: Resultados para imágenes del signo de T

La 'T' solo se confunde ocasionalmente con la 'F'.

Para entender mejor a que se debe este problema, vamos a analizar los signos de las letras anteriores con las alternativas que encuentra el modelo.

Figura 15: Conflictos para el signo A

Se puede ver que el conflicto no tiene mucho sentido mirando a los signos. Analizando un poco más en profundidad, se puede observar que hay dos variantes para el signo de 'A'. Una es la que representa la figura, y la otra es como 'G' pero con el índice recogido. La combinación de la similitud de la segunda variante con el signo de 'G', además del hecho de que en el conjunto de datos para el entrenamiento se encuentran signos de 'A' con las dos variantes, hace que 'A' tenga más variedad que el resto de signos, por lo tanto el modelo no aprende ambas maneras de representarlo y se equivoca más frecuentemente.

Respecto a la 'O':

Figura 16: Conflictos para el signo O

Para el signo 'O' es fácil entender por que el modelo lo confunde con 'S'. Ambos signos son increíblemente similares. La única solución que se me ocurre para solventar este error es tener un dataset más grande para que el modelo pueda entender las diferencias más sutiles entre ambos signos.

Finalmente, respecto a la 'T':

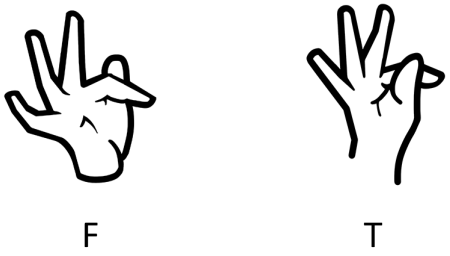


Figura 17: Conflictos para el signo T

Al igual que el caso de la 'O' y la 'S', los signos para 'T' y 'F' son muy similares. Por lo tanto, la solución sería aumentar el dataset al igual que en el caso anterior. A pesar de la similitud, se puede observar que en este caso el modelo confunde mucho menos un signo con otro, por lo tanto podemos deducir que la diferencia entre los signos es mayor, o el modelo tiene un sobreajuste hacia la 'S'. Posiblemente la explicación sea una mezcla de las dos.

Para terminar esta parte, analizaremos unos gráficos sobre las diferentes métricas durante el proceso de entrenamiento.

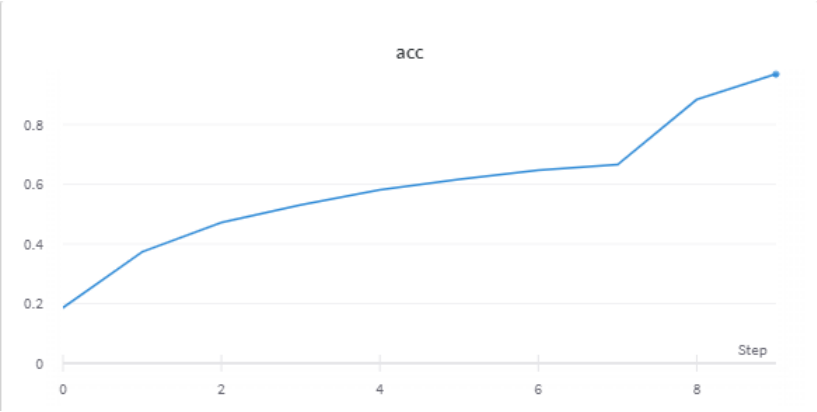


Figura 18: Evolución de la exactitud durante el entrenamiento

En la evolución de la exactitud durante el entrenamiento se pueden observar varias cosas. En el gráfico se representa tanto el entrenamiento inicial como el Fine Tunning. Se puede observar como el entrenamiento inicial alcanza un mínimo local entorno al 60 % de exactitud, pero gracias al Fine tunning hace que salga de ahí y llegue a ese 96 % de exactitud.

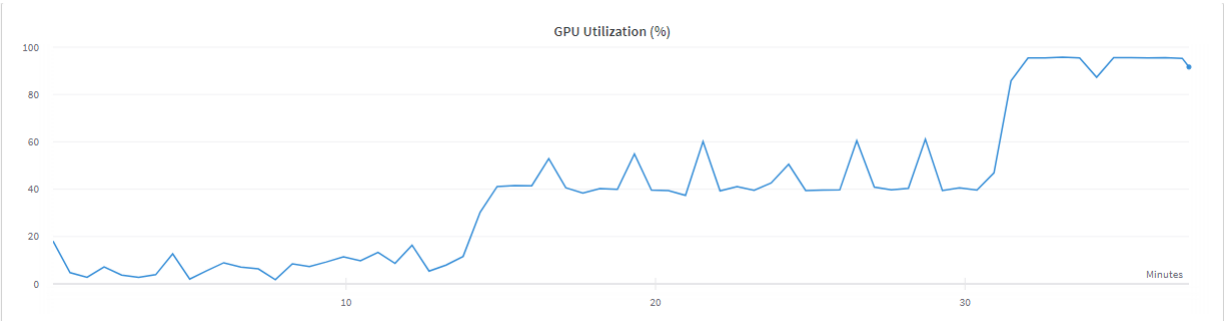


Figura 19: Uso de la GPU durante el entrenamiento

Por último, tenemos este gráfico que representa el uso de la tarjeta gráfica durante el entrenamiento. Debido a que mi sistema tiene una, decidí usarla para entrenar el modelo, ya que acelera mucho el proceso. Se puede observar que el entrenamiento en su completo llevó unos 36 minutos, y que a medida que nos acercamos al final se usa más. Se hace especialmente evidente que el número de neuronas, por tanto el tamaño del modelo, afecta a la computación requerida en el salto del final. Este salto se corresponde con Fine Tunning, ya que esta técnica descongela los pesos del modelo base permitiendo que estos también se actualicen. El tener que actualizar estos pesos cada vez que se predice una imagen de entrenamiento hace que el uso de la GPU incremente de esa manera. Esto explica por qué no era factible realizar un modelo muy grande desde el principio para mi problema, una de las razones por las que decidí usar Transfer Learning y Fine Tunning.

6 Análisis

Este apartado contendrá toda la especificación de requisitos y toda la documentación del análisis de la aplicación, a partir de la cual se elaborará posteriormente el diseño.

6.1 Definición del Sistema

En este apartado se contemplará el alcance del sistema, y todos los submódulos necesarios de desarrollar para realizarlo. También se evaluarán los diferentes requisitos, y se explicará a través casos de uso como funciona y debería funcionar el sistema, facilitando su entendimiento y también ayudando en la fase de desarrollo.

6.1.1 Determinación del Alcance del Sistema

El desarrollo de LSETranscriber se va a dividir en cuatro módulos principales:

- Creación del dataset
- Generación y entrenamiento del modelo
- Creación de la restAPI
- Confección de la página web

Creación del dataset
Debido a la inexistencia de un dataset con signos del LSE, se creará uno a mano. Se realizará a partir de capturas de pantalla de vídeos de internet y de vídeos proporcionados por voluntarios. Se aplicarán diferentes filtros a estas imágenes con el objetivo de aumentar el dataset, y se dividirán por clases en una estructura de ficheros adecuada para su posterior uso. De la misma forma, se reservarán imágenes para verificar que el entrenamiento se ha realizado con éxito y que los resultados son fiables.

Generación y entrenamiento del modelo
Se creará un modelo con diferentes capas, utilizando Keras. El modelo será continuamente refinado y modificado hasta alcanzar un porcentaje de exactitud mayor del 90 %, el cual es el mínimo satisfactorio. Para ello se utilizarán diferentes iteraciones sobre el mismo modelo, cambiando el número y tipo de las capas hasta obtener los resultados esperados. También se realizará Fine Tunning para refinar el modelo definitivo.

Este modelo posteriormente será almacenado de forma que sea reutilizable para la restAPI.

Creación de la restAPI
Consistirá en un servicio web que tendrá almacenado el modelo previamente entrenado. Este servicio consistirá en un endpoint al cual se podrán realizar peticiones HTTP de tipo POST con una imagen. Internamente, el servicio realizará las operaciones necesarias a la imagen para que sea compatible con el modelo, y usará el modelo para obtener una predicción. Finalmente, devolverá la predicción al usuario que realizó la petición.

Confección de la página web
Creación de una página web que permita consumir el servicio web. Consistirá en una vista sencilla en la que se mostrará la webcam del usuario, botones para realizar predicciones y una lista de las predicciones realizadas hasta el momento. De esta forma, el usuario podrá realizar una predicción, lo que consistirá en tomar una captura de ese momento de la webcam y enviarla a la restAPI, la cuál retornará una predicción que será mostrada en la página web.

De esta forma, varias predicciones serán mostradas como palabras, y se dispondrá de un botón para leer la predicción actual en voz alta. Ya que el modelo puede equivocarse, se permitirá al usuario borrar alguna predicción que fuera incorrecta. Para incluir el espacio como signo reconocido, este sucederá cuando no exista ninguna mano en la imagen de la predicción.

6.2 Requisitos del Sistema

En esta sección se expondrán los requisitos del sistema, tanto los funcionales como los no funcionales, todos ellos adquiridos en la fase de análisis del sistema.

6.2.1 Requisitos funcionales

RF_WEB_PRED.1. El sistema permitirá a los usuarios obtener predicciones a través de gestos.

RF_WEB_PRED.1.1. La predicción se realizará a través de capturas periódicas de la webcam

RF_WEB_PRED.1.2. El sistema enviará las capturas a la RESTApi que devolverá una predicción

RF_WEB_PRED.1.3. EL sistema mostrará las predicciones al usuario en orden

RF_WEB_PRED.2. EL sistema permitirá al usuario borrar las predicciones

RF_WEB_PRED.2.1. El borrado podrá ser:

RF_WEB_PRED.2.1.1. Individual

RF_WEB_PRED.2.1.1. De todas las predicciones

RF_WEB_PRED.3. El sistema permitirá leer todas las predicciones actuales

RF_WEB_PRED.3.1. Esto lo conseguiría a través de una tecnología de Text to Speech

RF_WEB_PRED.4. El sistema permitirá al usuario consultar tutoriales para su uso

RF_WEB_PRED.5. El sistema permitirá al usuario consultar información sobre la aplicación y su desarrollo

RF_REST_PRED.1. El Sistema devolverá una predicción al recibir una imagen

RF_REST_PRED.1.1. El puerto 5000 soportará peticiones POST para este cometido

6.2.2 Identificación de Actores del Sistema

Los diferentes actores que interactuarán con el sistema son:

Usuario web:
Estos usuarios usarán la aplicación web para realizar predicciones sobre sus gestos que serán obtenidos a través de la webcam. También consultarán los manuales de usuario de la aplicación.

Usuario experto servicio web:
Estos usuarios se comunicarán directamente con el servicio web para obtener predicciones de diferentes imágenes, sin usar la aplicación web. Se prevé una tasa de predicciones por segundo más alta para estos usuarios.

6.2.3 Diagrama de contexto del sistema

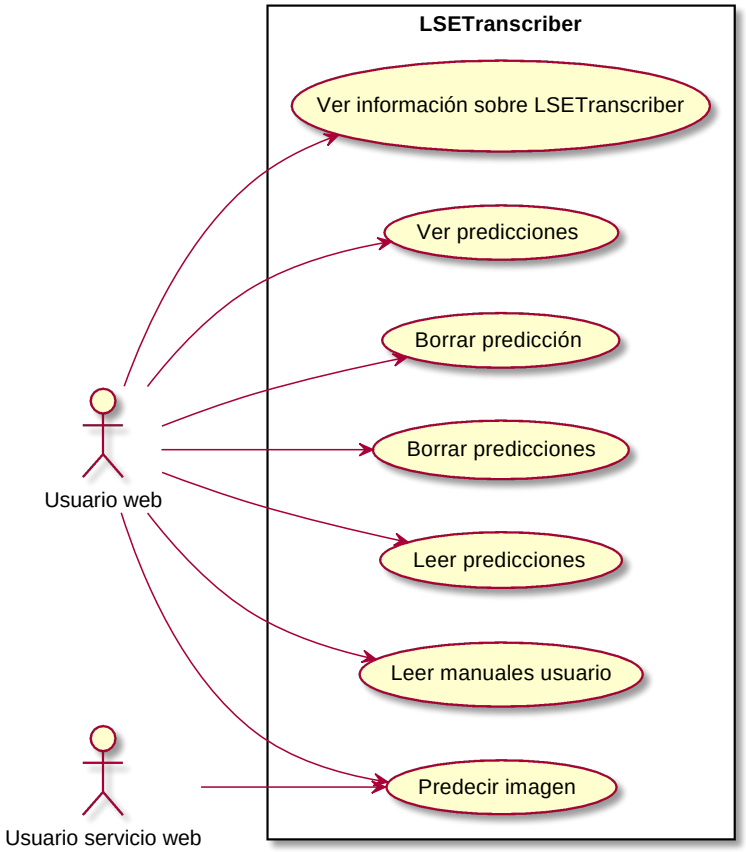


Figura 20: Diagrama de contexto

6.2.4 Especificación de Casos de Uso

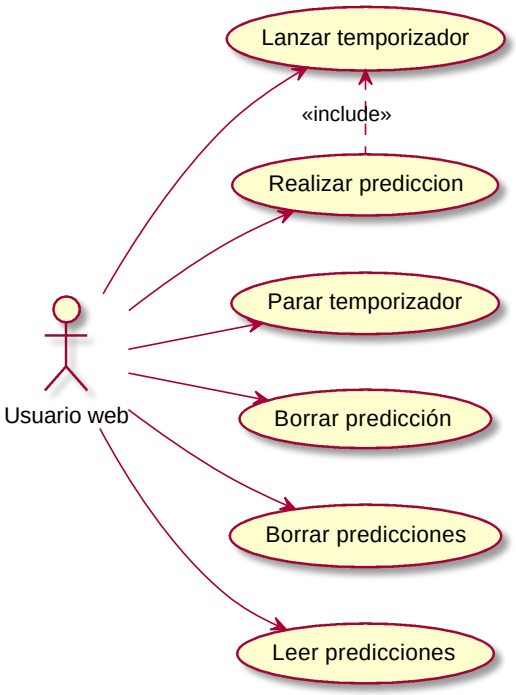


Figura 21: Caso de uso usuario web

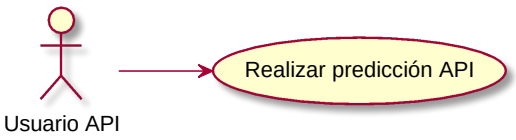


Figura 22: Caso de uso usuario servicio web

Nombre del Caso de Uso
Lanzar temporizador

Descripción
El usuario lanzará el temporizador para empezar a realizar predicciones.
El programa mostrará un botón para ello.

Nombre del Caso de Uso
Realizar predicción
Descripción
El usuario realizará signos que son capturados por la webcam. El programa mostrará la webcam en tiempo real y obtendrá predicciones de esta cada intervalo de 2 segundos mientras que el temporizador esté activado

Nombre del Caso de Uso
Parar temporizador
Descripción
El usuario parará el temporizador para terminar de realizar predicciones. El programa mostrará un botón para ello.

Nombre del Caso de Uso
Borrar predicción
Descripción
El usuario podrá borrar una letra de la predicción en el caso de que el o el programa se hubieran equivocado, haciendo click en la letra que desea eliminar.

Nombre del Caso de Uso
Borrar predicciones
Descripción
El usuario podrá borrar todas las predicciones, en el caso de que quiera realizar una nueva. El sistema mostrará un botón para ello.

Nombre del Caso de Uso
Leer predicciones
Descripción
El sistema mostrará un botón para leer la predicción actual completa.

Nombre del Caso de Uso
Realizar predicción API

Descripción
El usuario API podrá realizar una petición POST al puerto 5000/predict de la aplicación con una imagen para obtener una predicción. El sistema proporcionará un endpoint en esta dirección que se encargará de manejar la petición del usuario y devolver la predicción correspondiente.

6.3 Identificación de los Subsistemas en la Fase de Análisis

El sistema contiene dos subsistemas, la aplicación web y el servicio web.

6.3.1 Descripción de los Subsistemas

La aplicación web consistirá en todos los componentes que se encarguen de la aplicación web, tanto la parte visual como su comunicación con el servicio web. Consistirá en una vista en la que los usuarios podrán realizar predicciones a través de capturas en intervalos de su webcam. También consultarán manuales de usuario e información sobre la aplicación.

El servicio web, que contendrá el modelo entrenado, y recibirá imágenes y responderá con la predicción.

6.3.2 Descripción de las Interfaces entre Subsistemas

La comunicación entre los sistemas será a través de llamadas HTTP de tipo POST a los Endpoints correspondientes de la restAPI. La comunicación se hará localmente, ya que ambos subsistemas serán desplegados en la misma máquina.

6.4 Análisis de Casos de Uso y Escenarios

En este apartado explicaré los diferentes casos de uso identificados en 6.2.4. Se detallarán los escenarios más importantes, tanto de éxito como de error. También se tendrán en cuenta las precondiciones, postcondiciones, los actores que intervienen y las excepciones.

6.4.1 Lanzar temporizador

Precondiciones
El usuario está en la página de predicción. La grabación está detenida.

Postcondiciones
La grabación habrá comenzado. Cada intervalo de 2 segundos se tomarán capturas de pantalla de la webcam del usuario que serán predichas. Dicha predicción será añadida a la lista.

Actores
La acción es iniciada y terminada por el usuario.

Descripción
Para realizar la acción, el usuario simplemente tendrá que pulsar el botón de comenzar la grabación.

Variaciones
Si el usuario no tiene la webcam activada, las capturas realizadas de la webcam estarán en negro, por lo tanto no se detectará ninguna mano. Esto significa que la primera vez se detectará un espacio, y la segunda se parará la grabación.

En el caso de que la grabación estuviera en estado comenzado, volver a pulsar el botón haría que esta se detuviera.

6.4.2 Realizar predicción

Precondiciones
El usuario está en la página principal y la grabación ha sido comenzada.

Postcondiciones
Las imágenes capturadas de la webcam habrán sido predichas, añadiendo la predicción a la lista.

Actores
El usuario realiza y termina la acción.

Descripción
Una vez comenzada la grabación, cada intervalo de 2 segundos se realizará una captura de la webcam del usuario que será enviada para predecir. Esta predicción será añadida a la lista. En el caso de que no se encuentre una mano en la captura de la webcam, la primera vez se reconocerá como un espacio en blanco. La segunda vez consecutiva hará que la grabación se detenga.

Variaciones
Si el usuario realiza un signo incorrecto este será predicho de forma incorrecta, por lo tanto el usuario tendrá que eliminarla. Si el usuario no tiene la webcam encendida se tomarán capturas en negro, con resultado equivalente al de no encontrar una mano en la captura.

6.4.3 Parar temporizador

Precondiciones
El usuario está en la página principal y el sistema está grabando.

Postcondiciones
El sistema parará de grabar.

Actores
El usuario comienza y termina la acción.

Descripción
Hay dos vías para parar la grabación. La primera sería pulsar de nuevo el botón de grabar, la segunda sería a través de las predicciones. Como se ha comentado anteriormente, si el modelo no detecta ninguna mano en la imagen a predecir devuelve una respuesta especial. La primera vez que ocurre esto se considerará un espacio en blanco, mientras que la segunda vez consecutiva resultará en la parada de la grabación.

Variaciones
No hay variaciones para este caso, ya que incluso si el usuario no tiene la webcam activada esto se considerará como si no hubiera mano y se dejaría de grabar en la segunda predicción.

6.4.4 Borrar predicción

Precondiciones
El usuario está en la página principal. Hay una predicción actual con al menos 1 letra.

Postcondiciones
La predicción es igual que antes quitando la letra eliminada.

Actores
El usuario comienza la acción y el sistema la termina.

Descripción
Una vez que la predicción tiene al menos un elemento, el usuario puede borrar cualquier predicción que considere oportuna, una a una. Esto se realizará haciendo click en la predicción que se quiere borrar, ideal para cuando se realiza una predicción incorrecta.

Variaciones
El usuario podría hacer click en una predicción que no le interesaba borrar, en ese caso se perderá esa predicción. Ahora mismo LSETranscriber no permite insertar predicciones en un lugar específico, por lo tanto si la predicción borrada era una central, se tendrá que borrar al menos hasta ahí y repetir para enmendar el error.

En el caso de que no haya ninguna predicción no se podrá borrar, ya que es el propio botón de la predicción el que se debe pulsar.

6.4.5 Borrar predicciones

Precondiciones
El usuario está en la página principal y hay al menos una predicción.

Postcondiciones
No hay ninguna predicción y el botón de borrar predicciones está deshabilitado.

Actores
El usuario comienza la acción, el sistema la termina.

Descripción
El usuario puede borrar todas las predicciones actuales pulsando el botón de borrar predicción. No hay vuelta atrás en caso de que lo pulse por error.

Variaciones
No hay variaciones, ya que si no hay ninguna predicción el botón está deshabilitado.

6.4.6 Leer predicciones

Precondiciones
El usuario está en la página principal y hay al menos una predicción.

Postcondiciones
El estado posterior del sistema es el mismo que el anterior a realizar el caso de uso.

Actores
El usuario comienza la acción pulsando el botón de leer predicciones, el sistema la termina leyéndola.

Descripción
Si hay al menos una predicción, el usuario puede pulsar el botón de leer predicción para que el sistema la lea en voz alta.

Variaciones
No hay variaciones, ya que en el caso de que no haya ninguna predicción el botón estará deshabilitado.

6.4.7 Realizar predicción API

Precondiciones
El servicio web está desplegado en una red a la que el usuario puede acceder. El usuario conoce como realizar la petición, y tiene una imagen que quiere predecir

Postcondiciones
El usuario obtiene la letra de la predicción así como la confianza de la predicción.

Actores
El usuario comienza la acción y el sistema la termina

Descripción
Si el usuario no quiere utilizar la aplicación web de LSETranscriber, o quiere usar el servicio web para su propia aplicación, puede realizar peticiones al servicio web directamente con una imagen que será predicha. Simplemente tendrá que mandar una petición POST a la URL del servicio con la imagen en un form-data.

Variaciones
En el caso de que el usuario no construya la petición HTTP correctamente, el servicio enviará un mensaje correspondiente al error realizado.

6.5 Análisis de Interfaces de Usuario

En este apartado se explicará la interfaz de usuario, todos los componentes que lo forman y como será la navegabilidad entre ellos.

6.5.1 Descripción de la Interfaz

La interfaz con el usuario será muy sencilla. Constará de tres vistas:

- Vista de predicción
- Vista de ayuda
- Vista de información sobre LSETranscriber

Vista de predicción
Esta vista será la principal de la aplicación. Contendrá la barra de navegación en la parte superior, con acceso a las diferentes partes de la aplicación.

En la parte central dispondrá de un cuadro de vídeo donde se podrá observar la webcam del usuario en tiempo real. Debajo de esta, se encuentran unos botones que permitirán al usuario comenzar la grabación para la toma de imágenes, parar dicha grabación, borrar todas las predicciones y leerlas en voz alta.
Debajo de estos botones de utilidad, se encontrarán letras con las diferentes predicciones realizadas hasta la fecha. Estas irán surgiendo a medida que se realizan nuevas. Al pulsar cada botón individual, se eliminará esa predicción. Esto se hace con el objetivo de eliminar errores en la predicción, ya sea por culpa del usuario o por el modelo.

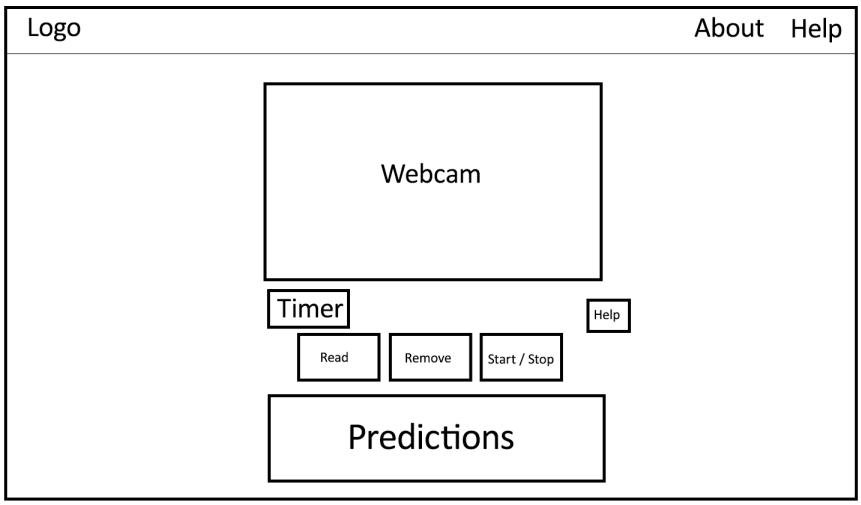


Figura 23: Mockup de la vista de predicción

Vista de ayuda
Esta vista consistirá en diferentes botones con ayuda de los diferentes apartados de la aplicación. Al pulsar sobre cualquiera de estos botones, se desplegará un cuadro más grande en el que se detallará a través de texto e imágenes ayuda sobre el aspecto elegido.

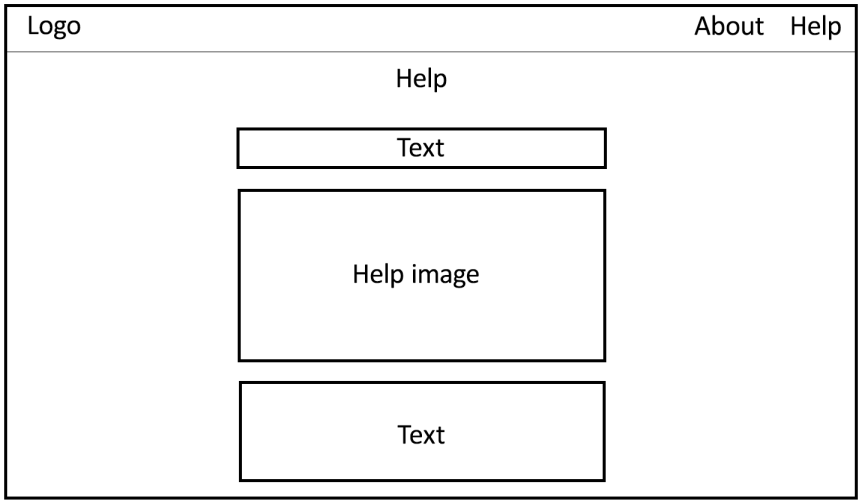


Figura 24: Mockup de la vista de ayuda

Vista de información sobre LSETranscriber
Aquí se podrá encontrar información sobre el creador, el propósito de LSETranscriber y como surge como TFG para la Universidad de Oviedo. También habrá información sobre la libertad de poder extenderla ya que es de código libre, y la dirección URL del repositorio de GitHub.

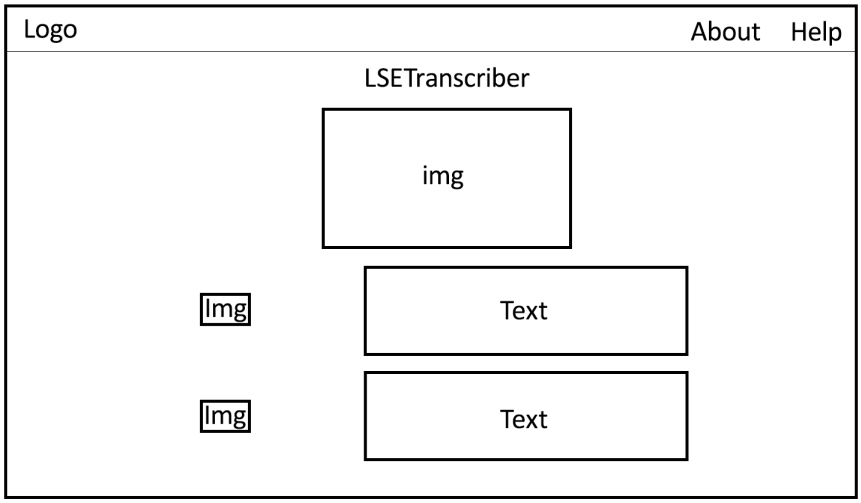


Figura 25: Mockup de la vista de información

6.5.2 Descripción del Comportamiento de la Interfaz

En este apartado especificaremos los convenios que vamos a crear para validar la entrada de datos de la aplicación, los mensajes de error que mostraremos y el tipo de ayuda que vamos a proporcionar al usuario.

Ya que la única entrada de LSETranscriber son las imágenes que queremos predecir, solo habrá una validación inicial, que será el comprobar si hay una mano en la imagen. Aún así, para facilitar el uso de la aplicación decidí que el hecho de que no se encontrara ninguna mano en la imagen generaría un espacio en blanco en la predicción la primera vez, y la segunda vez pararía la grabación. Por lo tanto, no se generaría ningún tipo de mensaje de error.

Respecto a la ayuda que se mostrará al usuario. Debido a que es posible que gente que no sea familiar con el Lenguaje de Signos Español use la aplicación, en la propia vista de predicción se encontrará una cheatsheet con los diferentes gestos reconocibles por LSETranscriber.

En cualquier caso, en la vista de ayuda habrá mas información de como usar la aplicación en su totalidad.

6.5.3 Diagrama de Navegabilidad

Ya que se puede ir desde cualquier vista a cualquier vista, el diagrama de navegabilidad es muy sencillo.

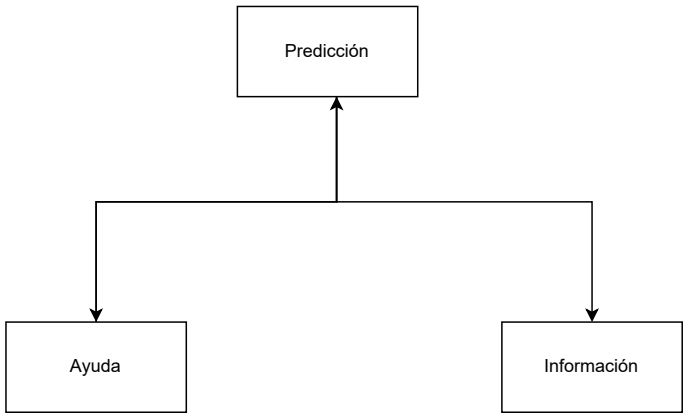


Figura 26: Diagrama de navegabilidad

6.6 Especificación del Plan de Pruebas

En esta sección se creará y diseñará el plan de pruebas de la aplicación y sus funciones, así como todos los mecanismos que utilizaremos para detectar errores y corregirlos en la fase de implementación. Las pruebas en LSETranscriber serán divididas en los siguientes tipos:

1. Pruebas unitarias
2. Pruebas de integración y sistema
3. Pruebas de accesibilidad y usabilidad
4. Pruebas de rendimiento y carga

6.6.1 Pruebas unitarias

Las pruebas unitarias se realizarán con Jest, y serán sobre la aplicación web. Jest 3.2.4 nos proporciona muchas funciones de mock. Mock significa que podemos enmascarar ciertos métodos, especialmente aquellos que realizan consultas externas a la aplicación o al componente, y establecer una respuesta específica dependiendo de la funcionalidad que queramos probar. De esta forma podemos probar como se comportaría el componente ante determinados valores de respuesta, y probar el funcionamiento con

las interfaces externas en las pruebas de integración.

Jest también nos permite que las pruebas se ejecuten de forma automática, por lo tanto sirven para garantizar que el programa funciona correctamente después de haber añadido nueva funcionalidad.

Las pruebas unitarias tienen dos objetivos. El primero es garantizar que la funcionalidad se ejecuta correctamente y el resultado de todas las operaciones dada una salida es el esperado. Por otro lado, nos interesa que el mayor número de líneas de código sea ejecutado, así podremos detectar otros fallos que podrían pasar desapercibidos, ya que no son visibles en el resultado final o porque no afectan a la funcionalidad principal.

Por lo tanto, especificaré que funcionalidad en particular voy a probar, pero tened en cuenta que el resto también será probado.

Los diferentes aspectos que se probarán son:

- Botón de eliminar predicción
- Botón de comenzar a grabar
- Botón de leer predicción
- Se añade un espacio al no detectar mano
- Se para la grabación al no detectar mano dos veces seguidas
- Se añade la letra a la predicción

6.6.2 Pruebas de Integración y Sistema

Las pruebas de integración y sistema se realizarán con Jest, Cucumber y Puppeteer. Jest al igual que comentamos antes proporciona herramientas para acceder fácilmente a los diferentes componentes y elementos de la página web, facilitando el proceso de testeo. Cucumber es una herramienta que permite describir los tests con un lenguaje (Gherkin) similar al lenguaje natural, de forma que sea fácil de entender por los clientes. Puppeteer es una herramienta que permite lanzar una instancia de Chrome headless, lo que significa sin interfaz. A partir de esta instancia, podemos desplegar nuestra aplicación, y navegar hasta ella. Con los tests escritos podemos observar si hay elementos, clicar botones, rellenar formularios... Básicamente casi cualquier acción que podría realizar un usuario corriente.
De la misma forma que las pruebas unitarias, estas pruebas están automatizadas para facilitar su ejecución.

Un aspecto que quisiera comentar en la realización de las pruebas de integración y sistema es que LSETranscriber, al necesitar input de una webcam del usuario, hay ciertos aspectos que no fui capaz de recrear para testear.
Al ser el objetivo de estas pruebas el probar la aplicación desplegada como tal, como lo vería un usuario real, utilizar Mocks como en las pruebas unitarias no es una buena práctica.

Esto conlleva que no puedo probar el comportamiento del sistema si el signo enviado es correcto, debido a que no puedo simular una webcam para las pruebas.

Por lo tanto, la funcionalidad que si se puede probar, y que se probará es:

1. Al no detectar mano se añade un espacio a la predicción
2. Al no detectar mano dos veces seguidas se para la grabación
3. El usuario puede navegar entre las diferentes vistas de la aplicación correctamente

6.6.3 Pruebas de Accesibilidad y Usabilidad

Las pruebas de accesibilidad y usabilidad se comprobarán manualmente. Serán conducidas por separado, y consistirán:

Pruebas de accesibilidad
Se supervisará a diferentes usuarios realizar unas tareas guiadas predefinidas. Tanto los usuarios como el responsable rellenarán unos cuestionarios sobre la experiencia usando LSETranscriber. El objetivo es tener una recepción general satisfactoria de la aplicación, especialmente entre usuarios no muy acostumbrados al uso de herramientas de este tipo.

Pruebas de usabilidad
La usabilidad se comprobará a través del cumplimiento de la aplicación del estándar WCAG AA. El objetivo será cumplir en su totalidad con el nivel AA.

6.6.4 Pruebas de Rendimiento

Las pruebas de rendimiento se realizarán con Gatling 3.2.5. Escribiremos tests en Java que simulen varios usuarios concurrentes realizando predicciones. De esta forma podremos ver que cantidad de usuarios soporta LSETranscriber. Para probar la carga de la aplicación web como tal, se simulará que los usuarios navegan entre las diferentes vistas también. Todo estas acciones se definirán en un proceso que seguirán todos los usuarios que simulemos.

7 Diseño del sistema

En este apartado mostraré múltiples diagramas y capturas que permitirán entender el diseño final de la aplicación, tanto desde un punto de vista físico como lógico.

7.1 Arquitectura del sistema

Esta primera sección se verá los diferentes apartados de la aplicación que realizan las diferentes tareas, y como la aplicación funciona y se comunica en su conjunto.

7.1.1 Diagramas de Paquetes

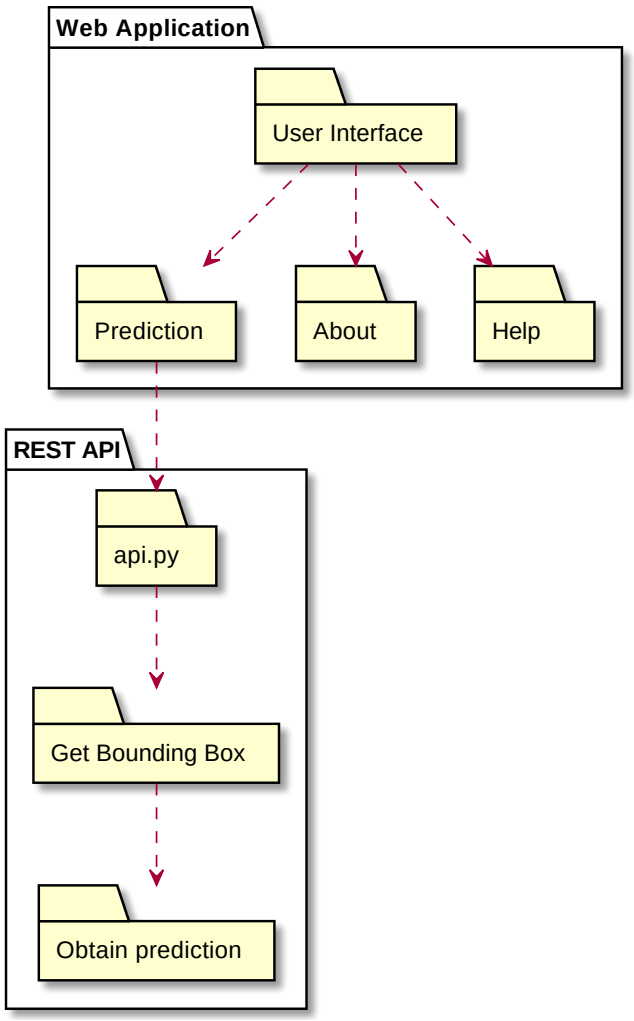


Figura 27: Diagrama de paquetes

7.1.2 Diagramas de Despliegue

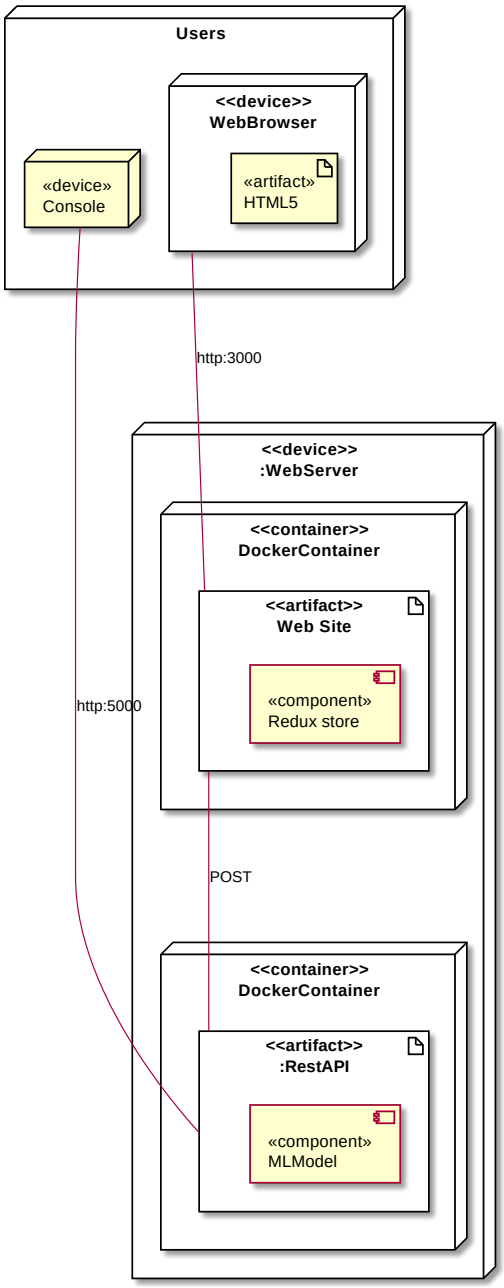


Figura 28: Diagrama de despliegue

7.2 Diagramas de Interacción y Estados

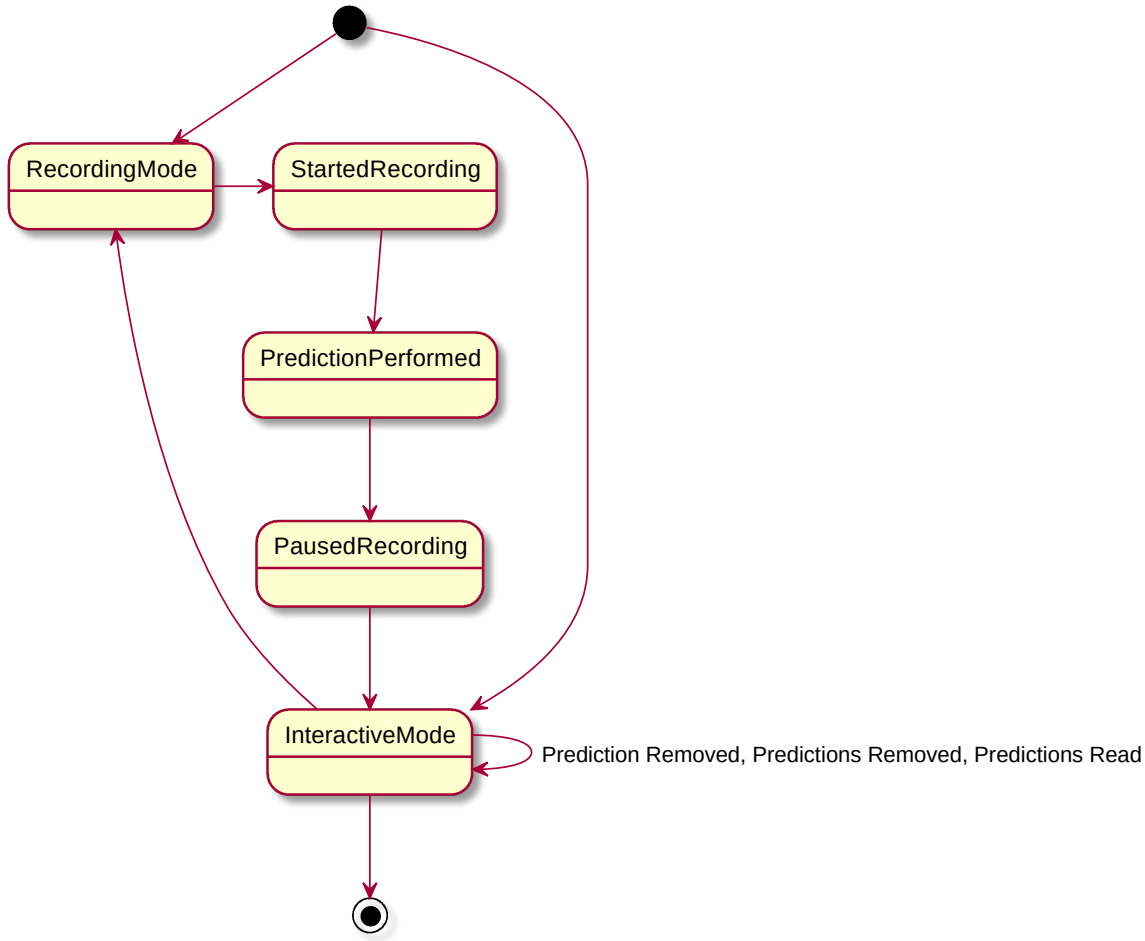


Figura 29: Diagrama de estado para la vista de predicción

7.3 Diagrama de Secuencia

La lógica que se representa en este diagrama se puede encontrar en el archivo predictionSlice.js.

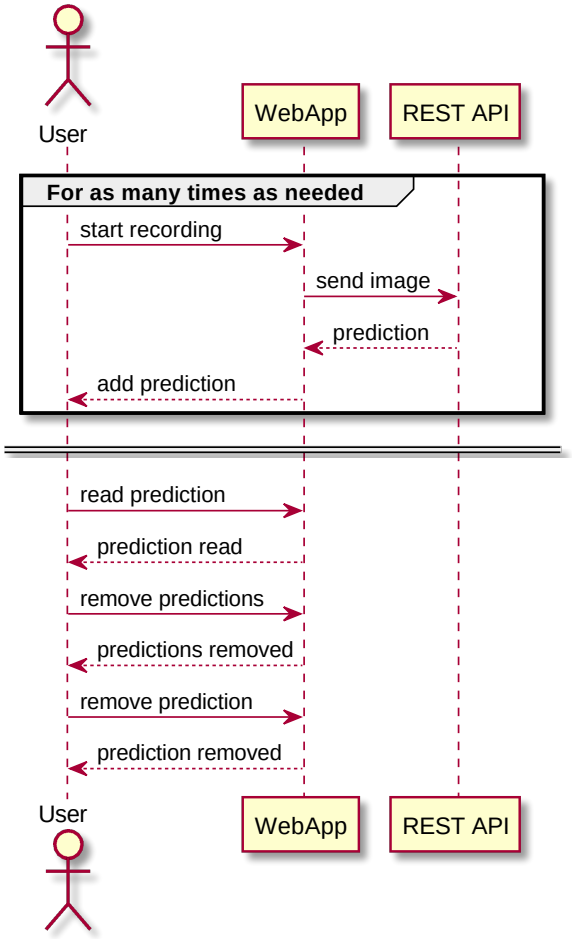


Figura 30: Diagrama de secuencia para la aplicación web

7.4 Diseño de la Interfaz

En este apartado enseñaré la versión final de las vistas de la interfaz, basadas en lo comentado en la parte de análisis 6.5. Tal y como se explicó entonces, el programa cuenta con tres vistas.

Vista principal/de predicción:

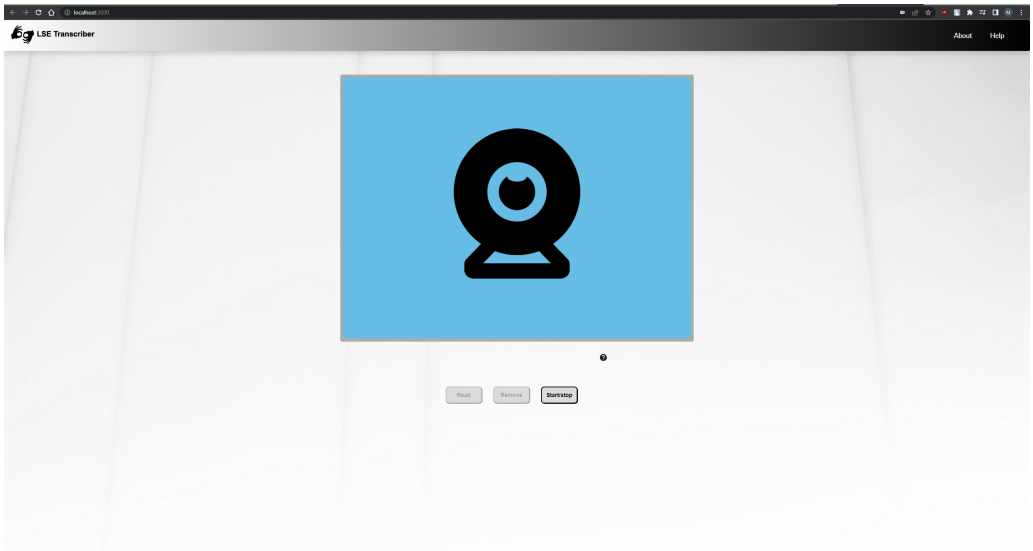


Figura 31: Página principal de LSETranscriber

Como se puede observar, la interfaz cuenta con una barra de navegación que será común a todas las vistas. Esta nos permite libertad y velocidad a la hora de ir a la vista que queramos. Posicionando el logo a la izquierda y el resto de botones a la derecha le damos más importancia a la vista de predicción.

Respecto a los elementos de la vista de predicción como tal, podemos ver que el objetivo es hacer una vista compacta evitando el desplazamiento vertical, para que el usuario pueda ver y usar toda la funcionalidad sin tener que usar el ratón. Esto se debe principalmente a que una vez que esté grabando signos sus manos estarán ocupadas. Por ello tanto los botones para manejar la predicción como las letras que contienen la predicción son relativamente pequeños y se encuentran hacia la mitad de la pantalla.

Ya que saber cada cuánto realizaba la aplicación una captura de pantalla era muy difícil, se colocó un temporizador que se activa cuando se comienza la grabación. De esta forma el usuario puede saber que cada vez que da una vuelta completa se realiza una captura.

Imaginando que puede haber usuarios que entren a la aplicación por curiosidad, pero sin conocer los signos del LSE, incorporé un signo de interrogación que te enseña los diferentes signos que puedes representar.

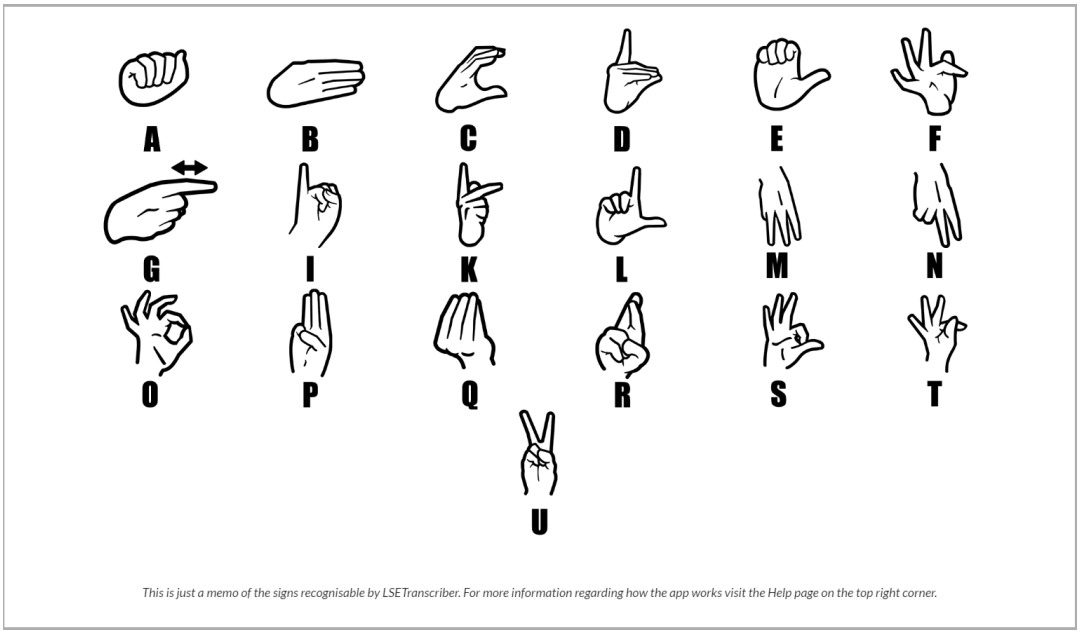


Figura 32: Signos reconocibles por LSETranscriber

Vista de información:

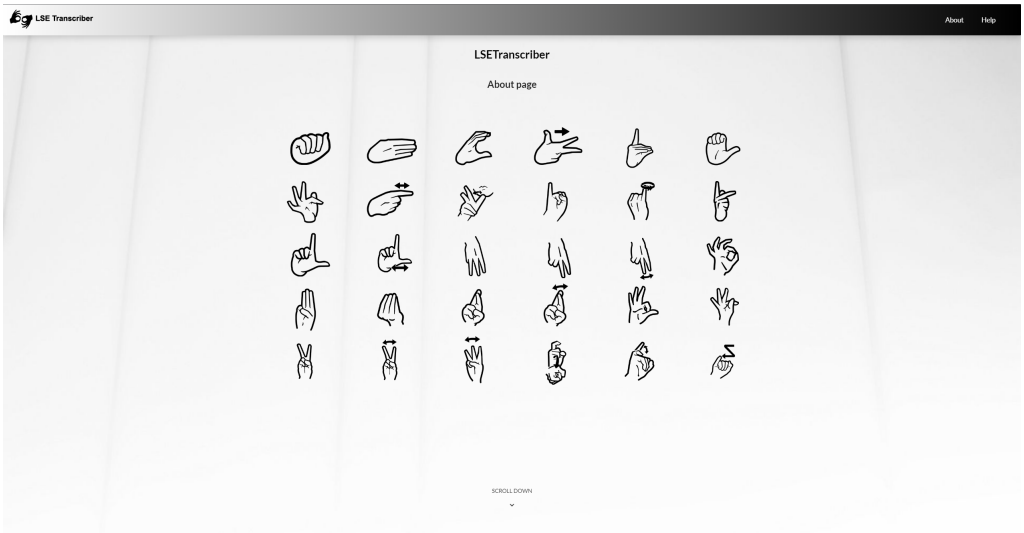


Figura 33: Página con información de LSETranscriber 1

En esta vista podemos encontrar información sobre la aplicación y sobre mi como su creador. Es una página simple, con animaciones para hacerla más entretenida. Si hacemos scroll o click en el elemento con la flecha hacia abajo, podemos ver la segunda mitad de la vista.

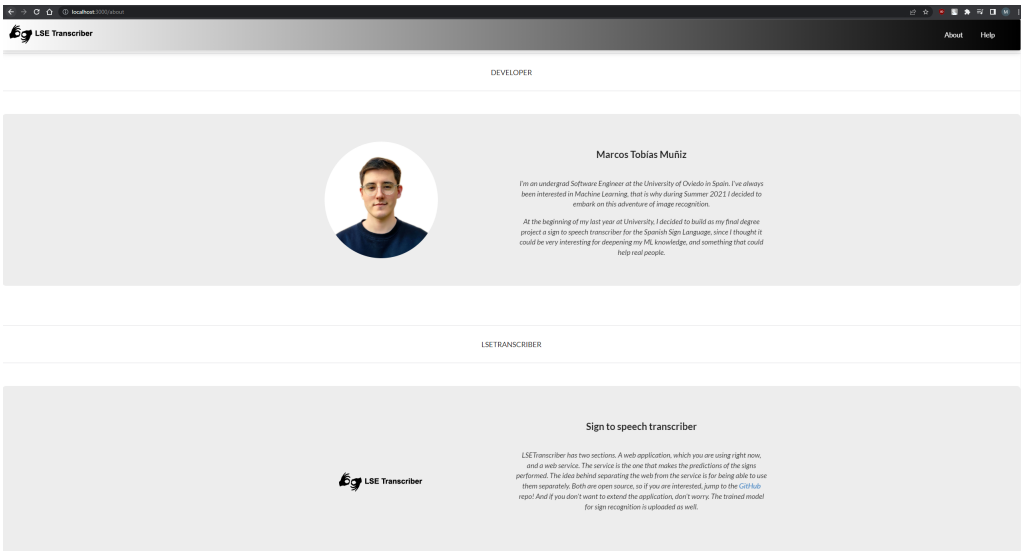


Figura 34: Página con información de LSETranscriber 2

Aquí podemos ver información sobre mí y sobre el proyecto.

7.5 Especificación Técnica del Plan de Pruebas

En este apartado se describirán las diferentes pruebas que se van a realizar durante todo el proceso de desarrollo, siendo estas de carácter unitario, de integración, del sistema, de usabilidad y de accesibilidad. El ordenador en el que se han llevado a cabo todas estas pruebas tiene las siguientes características:

- Windows 10 Pro
- Intel i7 8700
- Nvidia GeForce 1080
- VisualStudio Code
- Gatling
- Google Chrome

7.5.1 Pruebas Unitarias

En esta sección profundizaremos sobre los casos de prueba especificados anteriormente, incluyendo los valores específicos que usaremos y las salidas que esperamos encontrar.

Botón de eliminar predicción
Con una predicción inicial, se pulsará el botón y se comprobará que la predicción pasa a estar vacía

Botón de comenzar a grabar
Se pulsará el botón y se comprobará que la variable de grabación pasa a activado, y que al esperar dos segundos se realiza una llamada a la función externa con la imagen.

Botón de leer predicción
Con una predicción inicial, se pulsará el botón y se comprobará que la lógica asociada a la lectura de la predicción se ejecuta.

Con una predicción inicial vacía, se buscará el botón en el documento y no se encontrará.

Se añade un espacio al no detectar mano
Se mockeará la llamada a la API REST, devolviendo un valor por defecto de que no se encontró la mano. Se pulsará el botón de empezar a grabar, se esperarán dos segundos y se volverá a pulsar. Se comprobará que se realizó una petición a la función mockeada, y que el nuevo valor de la predicción es el original más un espacio en blanco al final.

Se para la grabación al no detectar mano dos veces seguidas
Se mockeará la llamada a la API REST, devolviendo un valor por defecto de que no se encontró la mano. Se pulsará el botón de empezar a grabar, se esperarán 5 segundos y se volverá a pulsar, comprobando que se realizan dos peticiones a la función mockeada. También se comprobará que el nuevo valor de la predicción es el mismo al original, y que la grabación está desactivada.

Se añade la letra a la predicción
Se partirá de una predicción vacía. Se mockeará la llamada a la API REST, devolviendo un valor por defecto de que se encontró la mano con valor 'K'. Se pulsará el botón de empezar a grabar, se esperarán dos segundos y se volverá a pulsar. Se comprobará que se realizó una petición a la función mockeada, y que el nuevo valor de la predicción es 'K'. También se comprobará que la grabación está desactivada.

7.5.2 Pruebas de Integración y del Sistema

Aquí explicaré los valores específicos que se utilizarán en los casos de prueba de integración y del sistema. Para ello, primero explicaré la definición de los pasos de la prueba en Gherkin, y posteriormente especificaré los valores explícitos utilizados en su ejecución.

Se han diseñado dos escenarios para las pruebas de integración, el primero es comprobar que se añade un espacio cuando no se detecta una mano en la captura, y la otra es observar si el usuario puede navegar por las diferentes vistas al clicar en los diferentes elementos de la barra de navegación.

Al no detectar mano se añade un espacio a la predicción

Funcionalidad: Realizar una predicción sin que haya una mano en la imagen

Escenario: El usuario está en la página principal

Dada una predicción
Cuando el usuario hace click en el botón de empezar a grabar
Y espera dos segundos sin que haya una mano presente en la webcam
Entonces la predicción añade un espacio en blanco

Al no poder modificar la webcam del usuario en los tests, la captura que se enviará en cualquier caso es una imagen completamente negra, por lo que nunca encontrará una mano presente.

Navegación

Funcionalidad: El usuario puede navegar entre las diferentes vistas

Escenario: El usuario puede ir a la página de información desde la página principal

Dado: El usuario está en la página principal

Cuando: El usuario hace click en About

Entonces: El usuario es redirigido a la página de información

Escenario: El usuario puede ir a la página de ayuda desde la de información

Dado: El usuario está en la página de información

Cuando: El usuario hace click en Help

Entonces: El usuario es redirigido a la página de ayuda

Escenario: El usuario puede ir a la página principal desde la de ayuda

Dado: El usuario está en la página de ayuda

Cuando: El usuario hace click en LSETranscriber

Entonces: El usuario es redirigido a la página principal

No se han podido diseñar más casos de prueba de integración y sistema debido al proceso de las imágenes. Ya que la aplicación al completo necesita de un usuario que de un input con sus signos, toda esa parte de la aplicación no puede ser simulada. También hay que tener en cuenta que no es objetivo de los tests de integración simular apartados, para la simulación de algún componente ya están los tests unitarios. Es por estos motivos que he decidido centrar las pruebas de integración y sistema en los elementos que sí puedo probar, dejando el resto para otro tipo de pruebas.

7.5.3 Pruebas de Usabilidad y Accesibilidad

Las pruebas de usabilidad y accesibilidad serán realizadas por separado. Primero, crearemos unos cuestionarios de evaluación, para que los usuarios que prueban la aplicación puedan rellenarlos. Estos cuestionarios tendrán preguntas generales para determinar el conocimiento técnico del usuario, y también preguntas sobre su tiempo usando la aplicación. También tendrá un apartado para que los usuarios dejen sus observaciones y sugerencias.

El cuestionario general será como sigue:

¿Usa un ordenador frecuentemente?

1. Todos los días
2. Varias veces a la semana
3. Ocasionalmente
4. Nunca o casi nunca

¿Qué tipo de actividades realiza con el ordenador?

1. Es parte de mi trabajo o profesión
2. Lo uso básicamente para ocio
3. Solo empleo aplicaciones estilo Office
4. Únicamente leo el correo y navego ocasionalmente

¿Ha usado alguna vez software como el de esta prueba?

1. Sí, he empleado software similar
2. No, aunque si empelo otros programas que me ayudan a realizar tareas similares
3. No, nunca

¿Qué busca usted principalmente en un programa?

1. Que sea fácil de usar

2. Que sea intuitivo
3. Que sea rápido
4. Que tenga todas las funciones necesarias

El cuestionario sobre las actividades guiadas es:

Actividad solicitada	Problemas y dificultades
Comience la grabación y realice los signos de A y C	
Pare la grabación y haz que se lea	
Borre la letra A de la predicción	
Borre toda la predicción	
Comience a grabar y añada un espacio	
Navegue hasta la página de ayuda	
Navegue hasta la página de información	

Por último, el cuestionario sobre la aplicación y observaciones:

Facilidad de uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?				
¿Existe ayuda para las funciones en caso de que tenga dudas?				
¿Le resulta sencillo el uso de la aplicación?				
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como usted espera?				
¿El tiempo de respuesta de la aplicación es muy grande?				
Calidad de la Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es				
Los iconos e imágenes usados son				
Los colores empleados son				
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?				
¿El diseño de las pantallas es claro y atractivo?				
¿Cree que el programa está bien estructurado?				
Observaciones				

Asimismo, el responsable de supervisar estas pruebas también tendrá un cuestionario, donde irá apuntando la solvencia y experiencia que observa del usuario, así como comentarios y observaciones adicionales que quiera hacer.

Este será:

Aspecto observado	Notas
El usuario comienza a trabajar de forma rápida por las tareas	
Tiempo en realizar cada tarea	
Errores leves cometidos	
Errores graves cometidos	

Ahora, las pruebas de accesibilidad. Para comprobar la accesibilidad, se comprobará que la aplicación cumple con el estándar WCAG 2 AA. Para ello, se seguirán las reglas establecidas por el W3C, que se explicarán en una sección posterior.

7.5.4 Pruebas de Rendimiento

Las pruebas de rendimiento serán realizadas con Gatling 3.2.5. Con esta herramienta podremos grabar la actividad de la red durante un proceso que haría un usuario normal, y luego reproducirlo bajo una carga de usuarios diferente. Para ello, vamos a considerar el siguiente escenario:

Usuario entra en la página web, espera 13 segundos para entender que está pasando, y realiza dos predicciones. Luego, en un afán investigador, visita la página de ayuda y la de información de la aplicación, haciendo pausas en cada una de estas para poder leer su contenido. Posteriormente, vuelve a la página principal y observa el pop-up con los signos reconocibles por LSETranscriber. Después de 6 segundos, realiza dos predicciones más.

Basándonos en ese recorrido, realizaremos dos tests. Un test en el que realizan estas acciones 2 usuarios nuevos cada segundo, durante 30 segundos. Y otro test donde lo realizan 4 usuarios nuevos por segundo, durante 60 segundos.

8 Implementación del sistema

En esta sección se explicarán los diferentes estándares y normas seguidos, así como los lenguajes de programación utilizados durante el desarrollo. También se comentarán todos los problemas encontrados y como se solucionaron o eludieron.

8.1 Estándares y Normas seguidos

Durante la creación de LSETranscriber se han seguido los siguientes estándares y Normas:

- WCAG2.0 - Grupo de normas sobre la accesibilidad web
- EsLint - se ha utilizado para asegurar que la complejidad del código es la adecuada, así como evitar malas prácticas como no acabar los archivos con línea nueva, utilizar comillas simples para los comentarios o no terminar las líneas con punto y coma.

8.2 Lenguajes de programación

LSETranscriber se ha creado usando Python y Javascript.
El servicio web y la creación y entrenamiento del modelo está escrito completamente en Python v.3.9. Para el modelo se han utilizado las siguientes librerías:

- Mediapipe v.0.8.9.1 - Utilizada para obtener las coordenadas de las manos en las imágenes

- OpenCV v.4.5.4.60 - Utilizada para realizar operaciones con las imágenes.
- Tensorflow y Keras v.2.7 - Usada para crear y entrenar el modelo
- Numpy v.1.19.5 - Utilizada para realizar operaciones matemáticas y de matrices

Para el servicio web se ha utilizado:

- Flask v.2.0.2 y flask-restx v.0.5.1 Para crear los endpoints y manejar las peticiones web

Para la aplicación web se ha utilizado:

- Node v.14.15.4 - Para gestionar los paquetes
- React v.17.0.2 y Redux Toolkit 1.6.2 - Utilizada para crear la aplicación web, utilizando JavaScript y el lenguaje JSX, que es una mezcla entre JavaScript y HTML
- CSS para realizar los estilos de la aplicación
- Otra multitud de paquetes de node para facilitar la creación de la aplicación web, omitidos por su abundancia y pequeña importancia

8.3 Herramientas y Programas usados para el desarrollo

En la realización de LSETranscriber se han usado multitud de herramientas y programas. En esta sección describiré cuáles son, su versión y el motivo de su utilización.

- VSCode v.1.65 - Visual Studio Code es un IDE para el desarrollo de aplicaciones en diversos lenguajes. Ha sido usado para realizar la aplicación web y la RestAPI, así como las pruebas unitarias y las de integración.
- IntelliJ PyCharm v.2020.1.2 - PyCharm es otro IDE, utilizado para crear y entrenar el modelo que predice los gestos.
- Gatling - 3.7.6 - Gatling es una aplicación para realizar tests de carga. Permite grabar y simular usuarios realizando acciones. Ha sido usada para realizar los tests de carga de la aplicación.
- Postman v.9.13 - Postman es una aplicación para realizar peticiones HTTP. Ha sido usada para probar manualmente el correcto funcionamiento de la RestAPI durante su desarrollo, así como el correcto despliegue de la aplicación en el servidor.
- MobaXTerm v.20.6.0.4532 - MobaXTerm es un programa para conexiones por SSH y FTP, usado para entrar al servidor donde LSETranscriber reside, subir los archivos correspondientes y realizar la configuración necesaria.
- React Developer Tools v.4.23.0 - Es una extensión de Chrome, utilizada para ver el estado de los componentes de la aplicación web en tiempo real, como herramienta de depuración.

8.4 Creación del Sistema

En esta sección describiré todos los aspectos con los que me he encontrado durante la implementación de LSETranscriber.

8.4.1 Problemas Encontrados

- Creación de un modelo suficientemente exacto
- Realizar validación cruzada al modelo
- Hacer fine tunning al modelo
- Comprobar y reducir sobreajuste del modelo
- Comunicar la aplicación web con la restAPI
- Tratar las imágenes en la restAPI para enviarlas al modelo
- Comprobar si la webcam del usuario está invertida
- Realizar un sistema de predicciones en tiempo real
- Facilitar el corregir predicciones erróneas
- Incorporar la predicción del espacio en blanco sin botón

Creación de un modelo suficientemente exacto
Debido a la falta de experiencia en entrenamiento de modelos, especialmente para problemas de clasificación de imágenes, fue muy complicado dar con un modelo que funcionara. Como se comenta en 5.2, después de diferentes intentos la solución fue utilizar transfer learning con el modelo VGG16.

Realizar validación cruzada al modelo
Al tener malos resultados durante la creación del modelo, se contempló realizar validación cruzada para encontrar un modelo con un rendimiento mejor. Por desgracia, esto probó ser muy complicado de implementar y se decidió finalmente no hacerlo. Al usar un dataset personalizado, este no se podía cargar igual que aquellos que vienen con las herramientas. Por ello, los métodos que había que usar dificultaban el tratamiento de los datos y cambiaban la implementación hasta tal punto que no pude realizar validación cruzada.

Hacer fine tunning al modelo
Una vez conseguido un modelo con buen rendimiento, era el momento de mejorarlo hasta el límite. Para ello había que implementar la técnica de fine tunning. Debido a que se estaba usando transfer learning para el entrenamiento, el modelo base era increíblemente grande. Por lo que descongelar todas sus capas era peligroso, ya que producía mucho sobreajuste muy rápidamente. Encontrar un balance entre la mejora de la técnica sin caer en el sobreajuste es complicado.

La solución a este problema fue iterar muy pocas veces, con una tasa de aprendizaje muy reducida.

Comprobar y reducir sobreajuste del modelo
Una vez que el modelo ya estaba considerado como acabado de entrenar, había que ver cuánto sobreajuste tenía y si era posible eliminarlo. Los datos del entrenamiento mostraban que el sobreajuste era mínimo, pero no parecían datos fiables. Ya que había guardado algunas imágenes que no fueron utilizadas durante el entrenamiento, fueron usadas para verificar el correcto funcionamiento del modelo. Estas devolvieron que existía un sobreajuste más significativo del que mostraba en la fase de entrenamiento, especialmente entre los signos que eran muy similares entre sí. Al ser la exactitud en el resto de signos muy elevada, daba una falsa sensación de exactitud, mientras que en los signos conflicto esta exactitud no excedía en muchos casos el 60 %.

Una vez descubierto que efectivamente existe sobreajuste, se redujeron las iteraciones de entrenamiento a partir del 80 % de exactitud, ya que es a partir de esta iteración que se empezaba a hacer evidente

un aumento de exactitud mayor en entrenamiento que en validación, lo cual es una clara muestra de sobreajuste. El porcentaje de exactitud restante se obtiene de hacer fine tunning.

Después de este proceso, la reducción de sobreajuste fue evidente, pero aun así sigue habiendo confusión entre los gestos similares.

Comunicar la aplicación web con la restAPI
Al decidir realizar los módulos de la aplicación como servicios independientes, esto generó un nuevo problema. Hay que conectar la aplicación web con la restAPI a la hora de mandar las imágenes y obtener la predicción. Hay varias formas de realizar esta conexión. Alguna especialmente interesante era utilizar webSockets, ya que la intención de la aplicación era ser en tiempo real. Otra opción era utilizar un flujo de datos de la webcam del usuario a la restAPI, similar a lo que sería un servicio de vídeo por demanda.

Al final, debido a la limitación computacional y la complejidad de implementar estas opciones, me decanté por enviar las capturas a intervalos periódicos razonables, a través de peticiones HTTP POST, donde la imagen estaría en el cuerpo de la petición.

Tratar las imágenes en la restAPI para enviarlas al modelo
Una vez obtenidas las imágenes en la restAPI, la exactitud del modelo prediciéndolas no era muy elevado. Posteriormente me di cuenta de que esto se debía a que el modelo esperaba que las imágenes tuvieran un formato preestablecido, igual al de las imágenes que fueron usadas en el entrenamiento.

Esta transformación implicaría reescalar las imágenes a 150x150, lo cual siendo una imagen de todo el cuerpo donde la mano ya era una pequeña parte, provocaría que en la imagen reescalada esta apenas fuera perceptible.

Por ello, usé la librería de mediapipe, la cual con su solución 'hands' me permitió obtener el área de la imagen que contenía la mano. Después de extraer este área, fue esa imagen la que se reescaló al tamaño requerido por el modelo.

Descubrí un problema adicional, ya que las imágenes de prueba no devolvían las mismas probabilidades que las mismas imágenes enviadas desde la aplicación web. En algún punto del proceso la imagen era invertida en los canales rojo y azul, por lo que tuve que realizar una operación extra para desinvertirlos.

Comprobar si la webcam del usuario está invertida
Este problema no debería ser un gran factor a la exactitud del modelo, ya que las imágenes son invertidas durante el entrenamiento. No encontré ninguna forma de comprobar esto, y después de varias pruebas vi que lo normal es que la cámara estuviera invertida. Ya que el problema podría surgir cuando la cámara está en espejo y no invertida, decidí no seguir investigando como detectarlo.

Realizar un sistema de predicciones en tiempo real
El objetivo inicial era que LSETranscriber pudiera usarse en tiempo real para facilitar su uso. Contemplé que realizar una aplicación en la que el usuario tuviera que subir las imágenes que quería predecir no era suficientemente usable.

Había varias alternativas, podía realizar un sistema que enviara todos los frames de la webcam a la restAPI para su evaluación, pero esto era una opción muy costosa computacionalmente, y tendría el problema añadido a la hora de escribir la predicción actual, ya que si el usuario realizara un gesto durante 1 segundo, esto implicaría 30 predicciones iguales, e incluso diferentes en el caso de que el modelo se equivocara, especialmente durante el comienzo y final de la realización del gesto.

Por ello, me decanté por un sistema que realizara una captura cada 2 segundos. Esto permitiría al usuario tener suficiente tiempo para cambiar entre gestos, y que el gesto capturado tuviera la máxima fiabilidad posible para que fuera más fácil para el modelo predecirlo. De esta forma, el sistema sigue siendo en tiempo real y evito muchas complejidades. Además me permitió añadir el reconocimiento de los espacios sin necesidad de utilizar procesamiento de lenguaje natural para dividir las palabras.

Facilitar el corregir predicciones erróneas
A pesar de mejorar el modelo todo lo posible, eso no garantiza que no se equivoque, o que el usuario no mande un signo incorrecto. Por ello, era importante tener algún tipo de sistema para eliminar predicciones incorrectas.

Con el objetivo de facilitar la eliminación, cada letra predicha es implementada como un botón. A través de una animación cuando pasas el ratón por encima, se puede ver una 'x' que deja claro que al pulsar el botón esa predicción se eliminará.
En el caso de que la predicción actual ya se hubiera leído y se quisiera pasar a la siguiente, hay un botón para borrar todas las predicciones.

Incorporar la predicción del espacio en blanco sin botón
Al convertir las predicciones en un sistema de tiempo real, deja un problema a la hora de implementar el espacio en blanco. Inicialmente la idea era disponer de un botón que simplemente añadiera una predicción en blanco.

Al estar realizando los signos en intervalos de dos segundos, un botón es inviable. Por ello, la solución implementada fue considerar una predicción en la que no se detectara ninguna mano como espacio en blanco. Esto se debe gracias a que la librería de mediapipe detecta si hay una mano o no en pantalla.

9 Desarrollo de las pruebas

En esta sección se describirá el resultado de las pruebas que se han diseñado 7.5, observando los posibles errores y problemas encontrados.

9.1 Pruebas Unitarias

Siendo una aplicación tan sencilla, la ejecución de las pruebas unitarias no encontró ningún error. Se realizaron 15 tests, los cuales pasaron todos. Se pueden observar los resultados en la siguiente imagen:

```
-----------------------------------|---------|----------|---------|---------|-------------------
File                               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------------|---------|----------|---------|---------|-------------------
All files                          |     100 |      100 |     100 |     100 |
 src                               |     100 |      100 |     100 |     100 |
  App.js                           |     100 |      100 |     100 |     100 |
  index.js                         |     100 |      100 |     100 |     100 |
  reportWebVitals.js               |     100 |      100 |     100 |     100 |
 src/api                           |     100 |      100 |     100 |     100 |
  API.js                           |     100 |      100 |     100 |     100 |
  apiFetchPrediction.js            |     100 |      100 |     100 |     100 |
 src/components                    |     100 |      100 |     100 |     100 |
  AboutView.js                     |     100 |      100 |     100 |     100 |
  HelpView.js                      |     100 |      100 |     100 |     100 |
  LetterButton.js                  |     100 |      100 |     100 |     100 |
  NavBar.js                        |     100 |      100 |     100 |     100 |
  PredictionView.js                |     100 |      100 |     100 |     100 |
  Predictions.js                   |     100 |      100 |     100 |     100 |
 src/components/aboutComponents    |     100 |      100 |     100 |     100 |
  Developer.js                     |     100 |      100 |     100 |     100 |
  Project.js                       |     100 |      100 |     100 |     100 |
 src/redux                         |       0 |        0 |       0 |       0 |
  store.js                         |       0 |        0 |       0 |       0 |
 src/redux/slices/predictionSlice  |     100 |      100 |     100 |     100 |
  predictionSlice.js               |     100 |      100 |     100 |     100 |
-----------------------------------|---------|----------|---------|---------|-------------------

Test Suites: 12 passed, 12 total
Tests:       15 passed, 15 total
Snapshots:   0 total
Time:        9.856 s
Ran all test suites.
```

Figura 35: Ejecución de los tests unitarios

Las clases en gris son aquellas que unitariamente no se puede probar su funcionalidad, pero son utilizadas por otros componentes. En nuestro caso la única clase que no se puede probar es store.js, que contiene la definición de la lógica de Redux. Pero aún así, hemos probado la funcionalidad de predictionsSlice.js, que contiene la implementación de dicha lógica, por lo tanto podemos concluir que store.js también es correcta.

9.2 Pruebas de Integración y del Sistema

Después de implementar las pruebas de integración y sistema definidas en Gherkin, pudimos ejecutarlas automatizadas como si fueran tests normales. Esto nos permite poder observar su resultado y en base a el deducir si nuestro sistema funciona como debería. El resultado de estos tests ha sido:

```
 PASS  steps/no-hand-prediction.js (37.019 s)
192.168.1.110 - - [26/Jun/2022 20:20:36] "POST /predict HTTP/1.1" 500 -
 PASS  steps/navigation.js (42.727 s)

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        50.979 s, estimated 63 s
Ran all test suites.
```

Figura 36: Ejecución de los tests de integración y sistema

Se han demorado tanto ya que el entorno de pruebas levanta tanto la aplicación web como el servicio web y lo contabiliza como tiempo de ejecución de test.

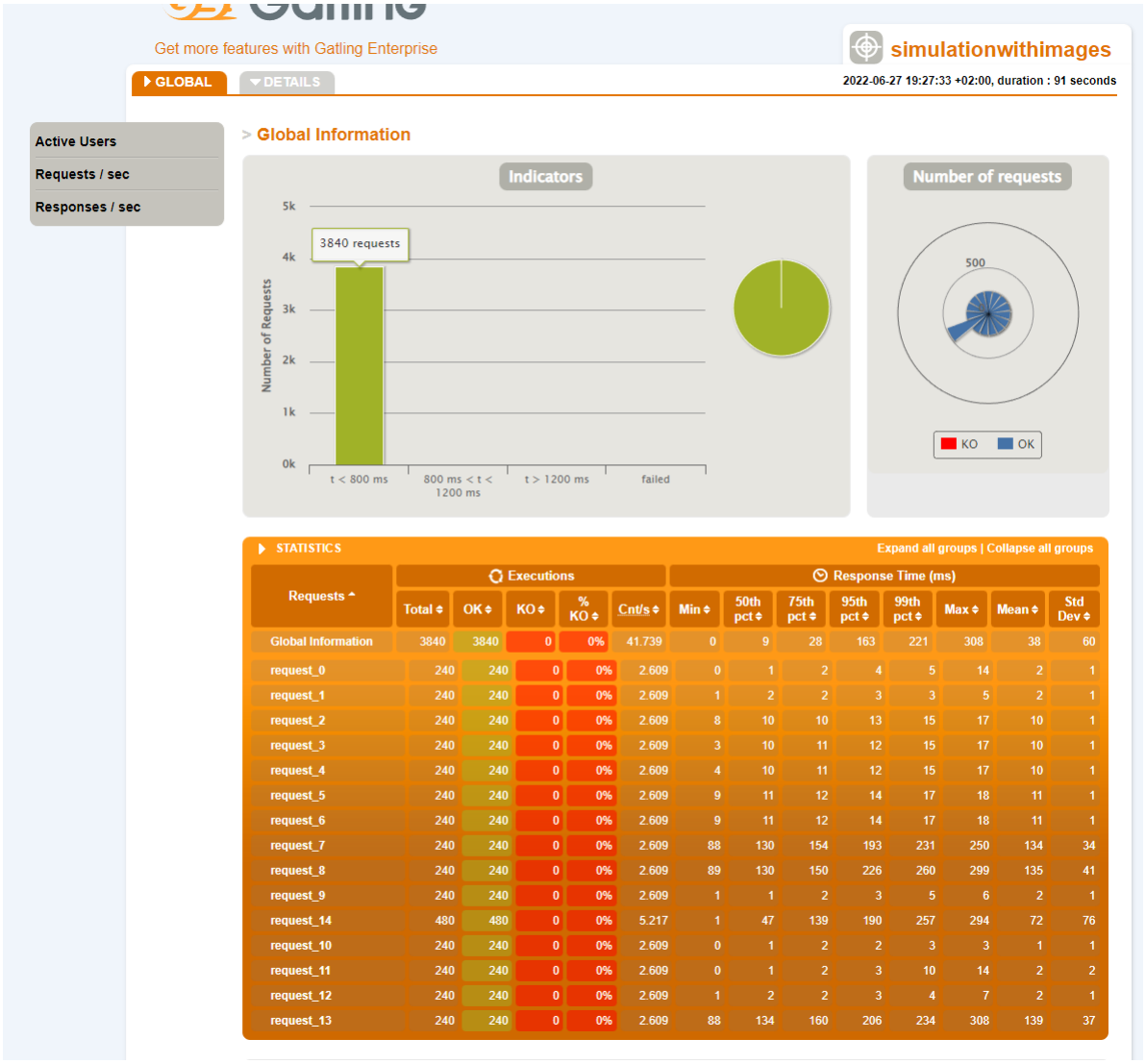
9.3 Pruebas de Usabilidad y Accesibilidad

Estas pruebas no fueron implementadas al final. No hubo tiempo para encontrar candidatos para realizar las diferentes pruebas de usabilidad. Las pruebas de accesibilidad no hubo ni tiempo ni recursos. La aplicación no está hosteada públicamente, por lo tanto no podía usar ninguna de las herramientas que se pueden encontrar online para la accesibilidad. La única opción restante era realizar la comprobación del cumplimiento de cada regla establecida en las WCAG 2.0 a mano, y no hubo tiempo para ello.

9.4 Pruebas de Rendimiento

La primera parte del desarrollo fue grabar la situación para reproducirla posteriormente. Para ello, se utilizó Google Chrome con la grabación de red activada. Una vez completado el recorrido, se descargó en un archivo HAR. Posteriormente utilicé Gatling para convertirlo en un archivo de java, y lo modifiqué para cumplir con los casos de prueba establecidos. Los resultados son los siguientes:

2 Usuarios por segundo durante 30 segundos
Como se puede ver en 37, con esta carga la aplicación fue capaz de manejar todas las peticiones sin problemas, con un tiempo de respuesta menor a 800ms que se considera óptimo.



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	3840	3840	0	0%	41.739	0	9	28	163	221	308	38	60
request_0	240	240	0	0%	2.609	0	1	2	4	5	14	2	1
request_1	240	240	0	0%	2.609	1	2	2	3	3	5	2	1
request_2	240	240	0	0%	2.609	8	10	10	13	15	17	10	1
request_3	240	240	0	0%	2.609	3	10	11	12	15	17	10	1
request_4	240	240	0	0%	2.609	4	10	11	12	15	17	10	1
request_5	240	240	0	0%	2.609	9	11	12	14	17	18	11	1
request_6	240	240	0	0%	2.609	9	11	12	14	17	18	11	1
request_7	240	240	0	0%	2.609	88	130	154	193	231	250	134	34
request_8	240	240	0	0%	2.609	89	130	150	226	260	299	135	41
request_9	240	240	0	0%	2.609	1	1	2	3	5	6	2	1
request_14	480	480	0	0%	5.217	1	47	139	190	257	294	72	76
request_10	240	240	0	0%	2.609	0	1	2	2	3	3	1	1
request_11	240	240	0	0%	2.609	0	1	2	3	10	14	2	2
request_12	240	240	0	0%	2.609	1	2	2	3	4	7	2	1
request_13	240	240	0	0%	2.609	88	134	160	206	234	308	139	37

Figura 37: Resultados del primer test de carga 1

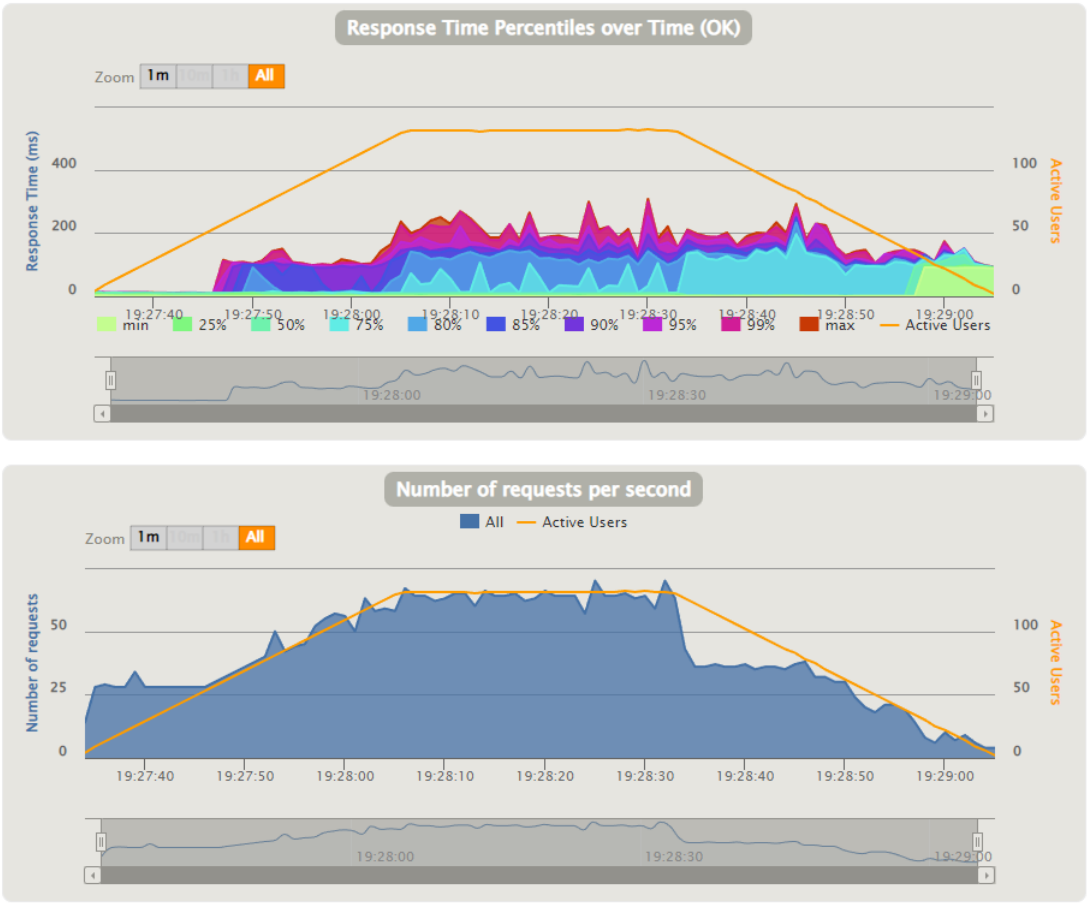


Figura 38: Resultados del primer test de carga 2

4 Usuarios por segundo durante 60 segundos

En esta segunda prueba el sistema tuvo muchos más problemas. El retraso en las predicciones por parte del servicio web provocó una diferencia en el timestamp de la imagen que se envía a mediapipe para que extraiga la bounding box de la mano, y esta diferencia hizo que con el tiempo pasara de ser un warning a un error. Por lo tanto, el servicio web colapsó hacia el final de la prueba.

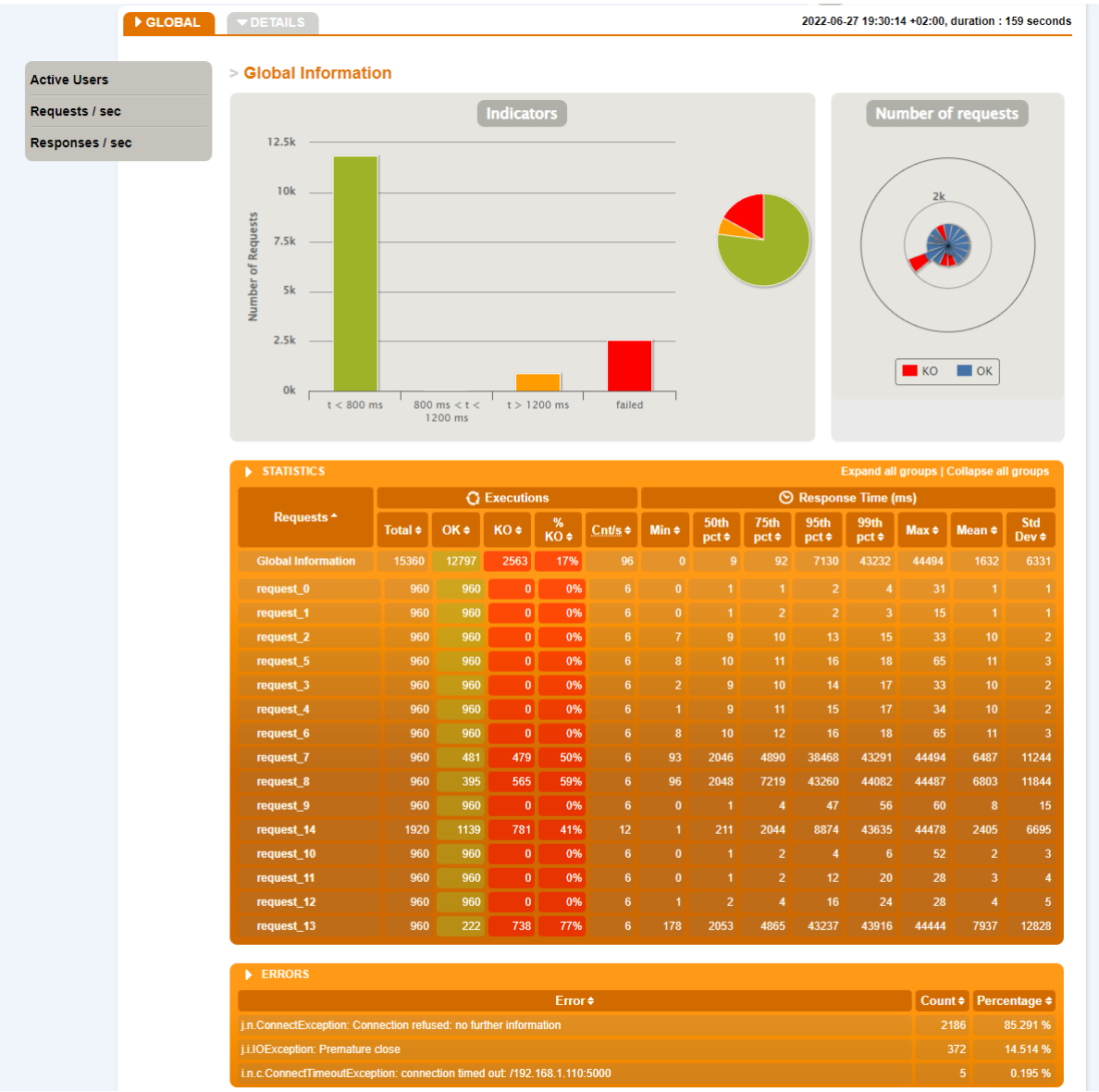


Figura 39: Resultados del segundo test de carga 1

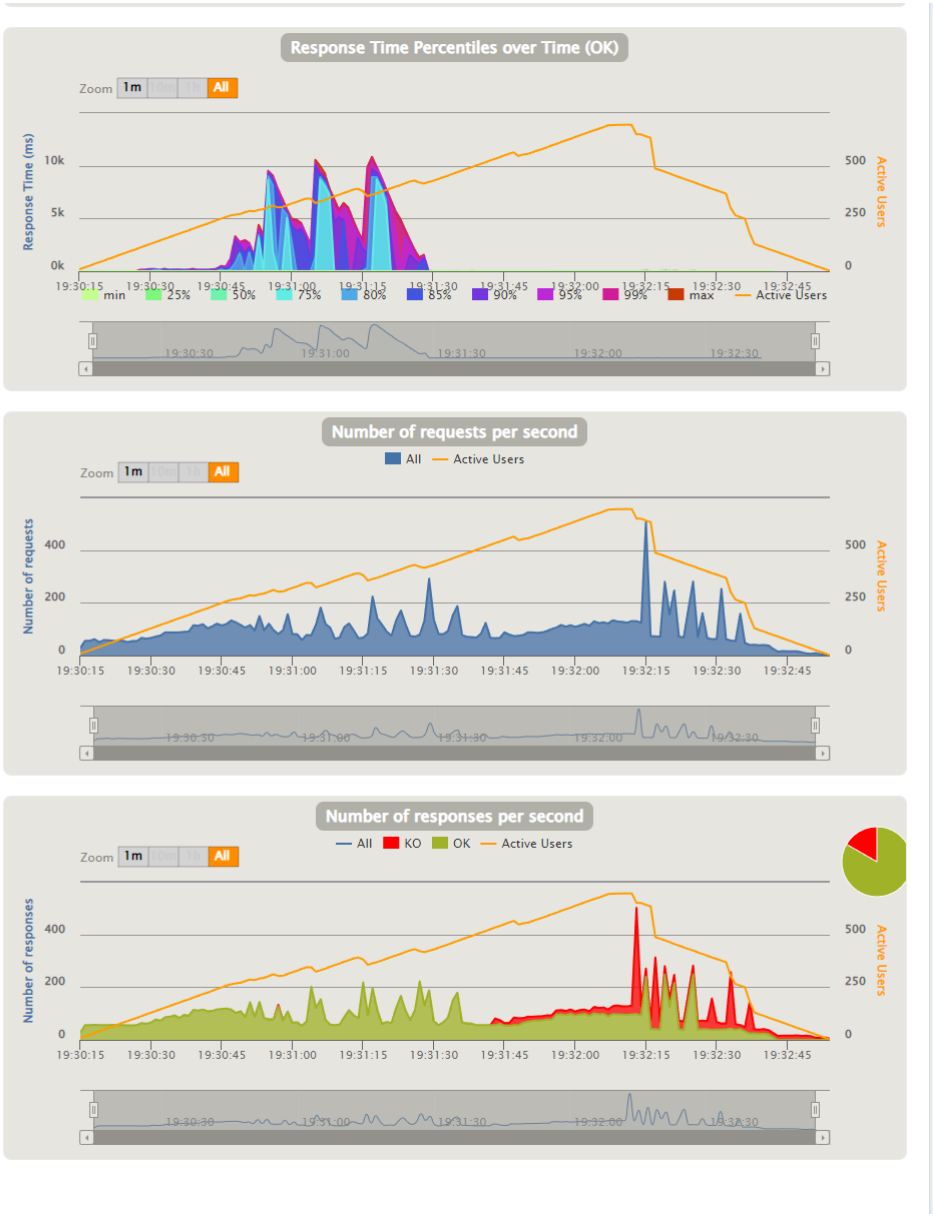


Figura 40: Resultados del segundo test de carga 2

Al ser un test del mismo test que el anterior, deberíamos poder observar una misma evolución de peticiones, siendo una especie de ascenso inicial, una meseta y un descenso al final. Por culpa del retraso en la respuesta de las peticiones, se puede ver como los usuarios se ven obligados a permanecer más tiempo en la página web para realizar la misma función, por lo tanto las peticiones pendientes van en aumento, hasta que se empieza a observar picos en los fallos, donde el servicio web empieza a dejar de responder hasta que finalmente acaba colapsando.

10 Manuales del sistema

En esta sección se explicarán los diferentes manuales del sistema, para los diferentes perfiles de usuarios que pueden querer usar nuestra aplicación.

10.1 Manual de Instalación

Para realizar la instalación de LSETranscriber lo primero es descargar el código del repositorio de GitHub [8]. Para ello, clicamos en 'Code' y luego en 'Descargar ZIP'.

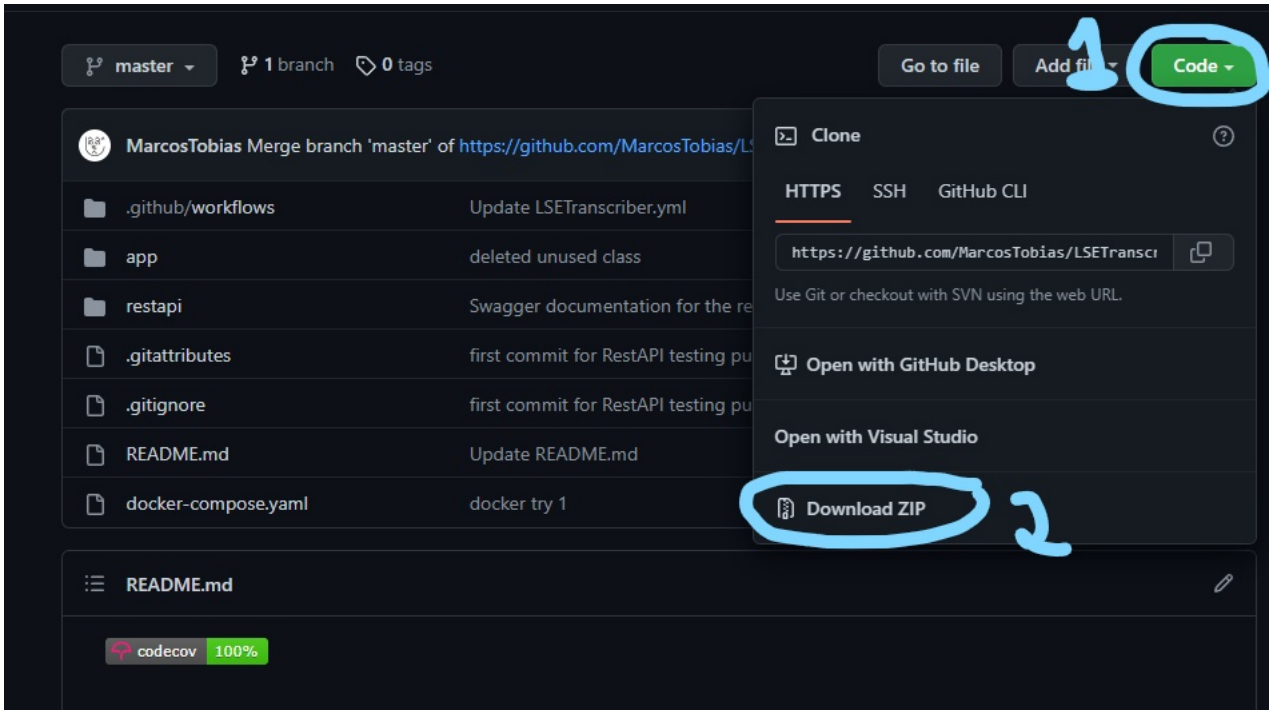


Figura 41: Descargar el código del proyecto

Posteriormente, descomprimimos el ZIP.

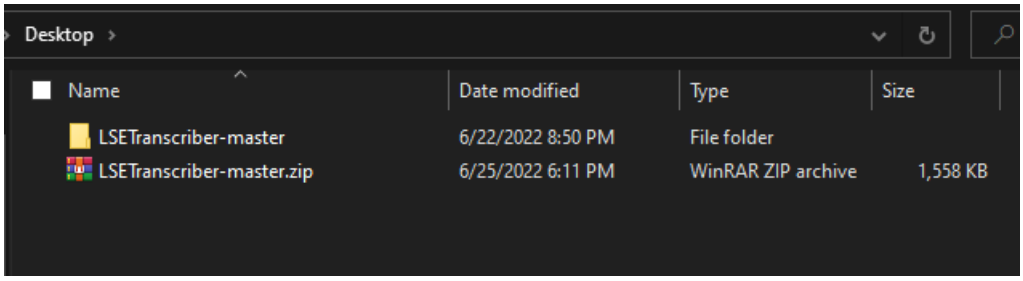


Figura 42: Descomprimimos la descarga

Para ejecutar la aplicación se necesita Docker [4], por lo cual habría que instalarlo también.

10.2 Manual de Ejecución

Una vez que tenemos el repositorio en local y Docker instalado, solo haría falta abrir una consola. Si estamos en Windows, lo más sencillo es situarse en el directorio raíz del proyecto desde el explorador de archivos, hacer click en la dirección, escribir 'cmd' y darle al Enter.

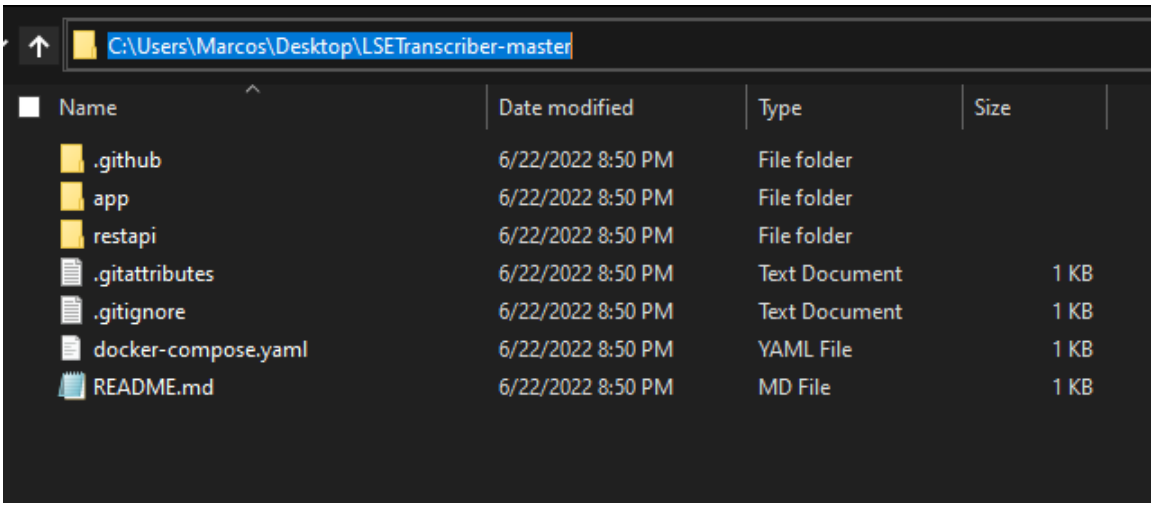


Figura 43: Crear una consola en el proyecto: paso 1

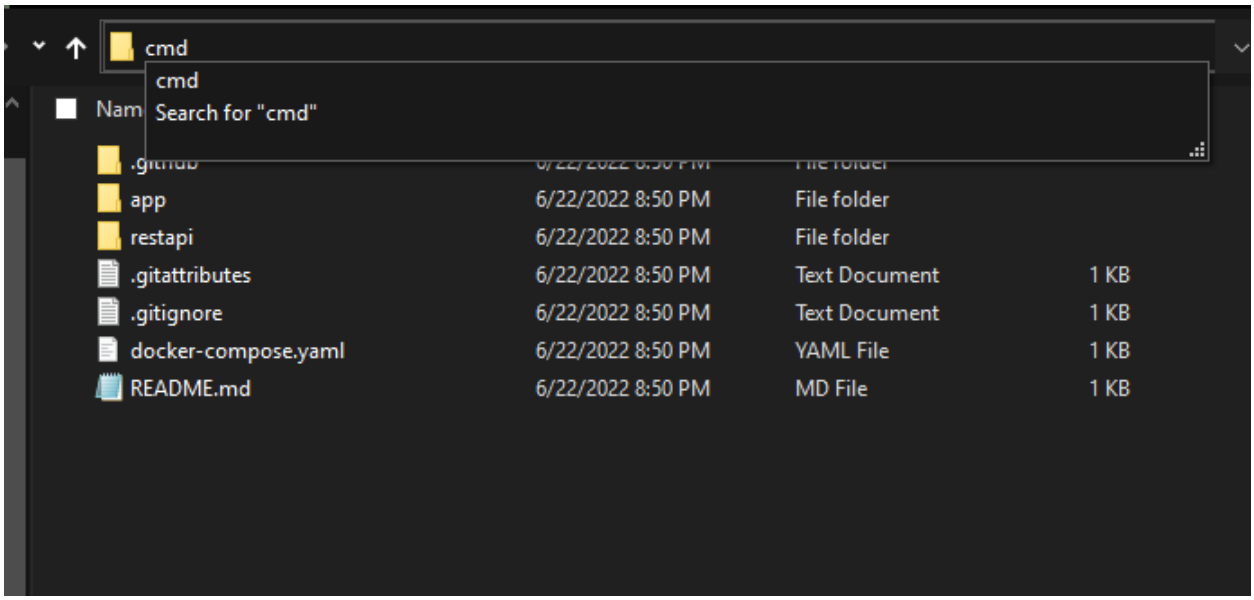


Figura 44: Crear una consola en el proyecto: paso 2

Si estamos en otro sistema operativo, habría que abrir una consola y escribir cd «Ruta al directorio raíz». Una vez situados, solo haría falta escribir 'docker-compose up' y esperar a que se despliegue.

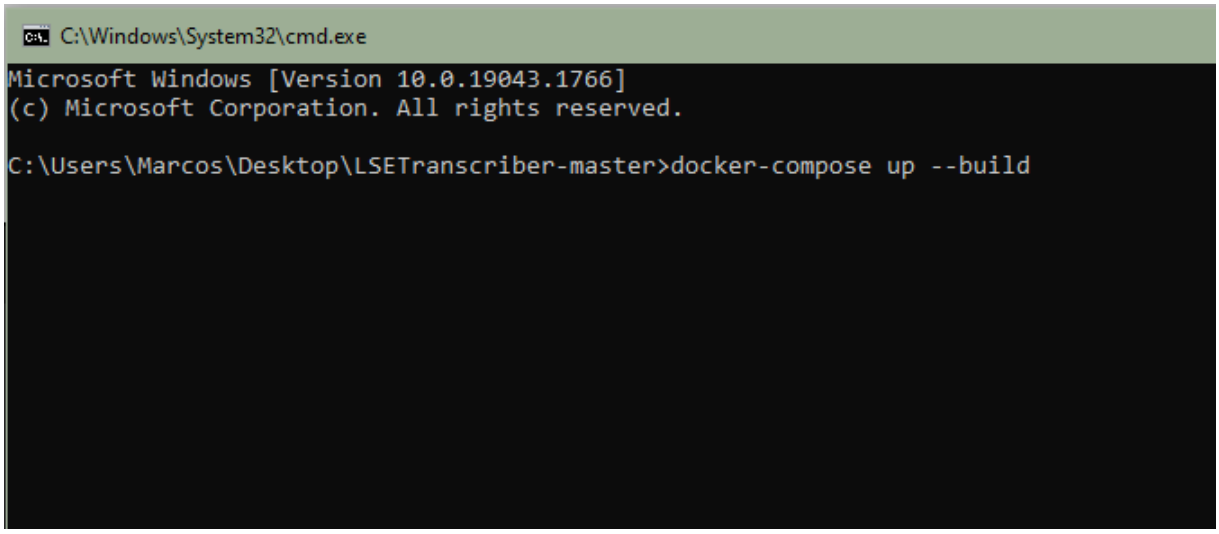


Figura 45: Arrancar el proyecto

La aplicación web estará en localhost:3000 y el servicio web en localhost:5000.

Es importante no cerrar la consola durante el tiempo que se quiera usar LSETranscriber. Para parar la aplicación, bastaría con hacer 'Ctrl + C' en la consola.

Estos pasos son obligatorios la primera vez, pero a partir de ahí es más fácil arrancar y parar la aplicación con el uso de Docker Desktop. Simplemente habría que ir a la vista de contenedores y poner el ratón encima del volumen 'lsetranscriber', y darle al Play o al Stop dependiendo de la acción que se quiera realizar.

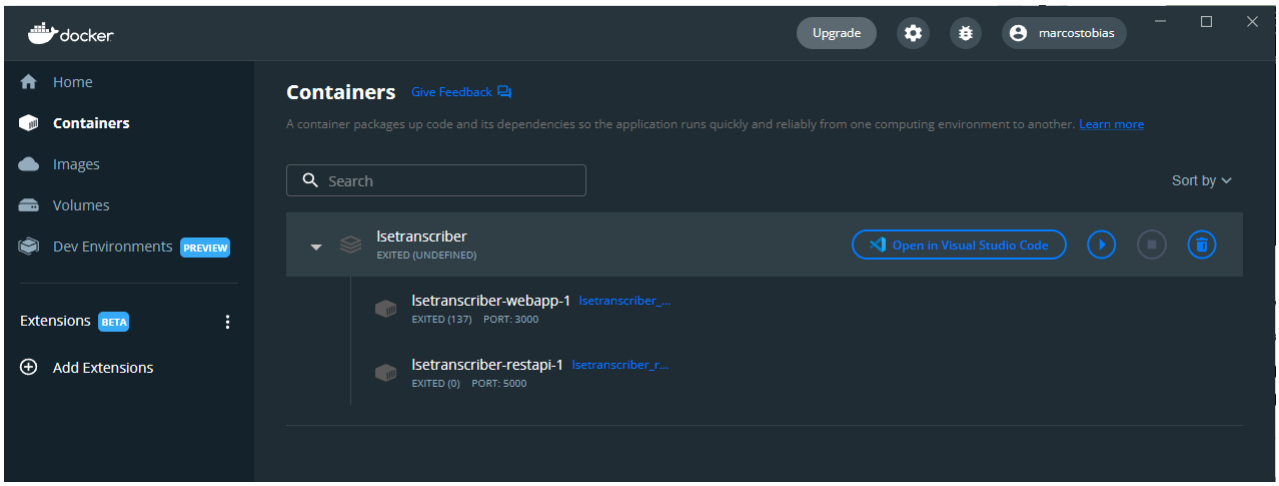


Figura 46: Alternativa para arrancar y parar el proyecto con Docker Desktop

10.3 Manual de Usuario

Para utilizar la página web, se tendrá que navegar a http://localhost:3000, donde se llegará a la página principal.

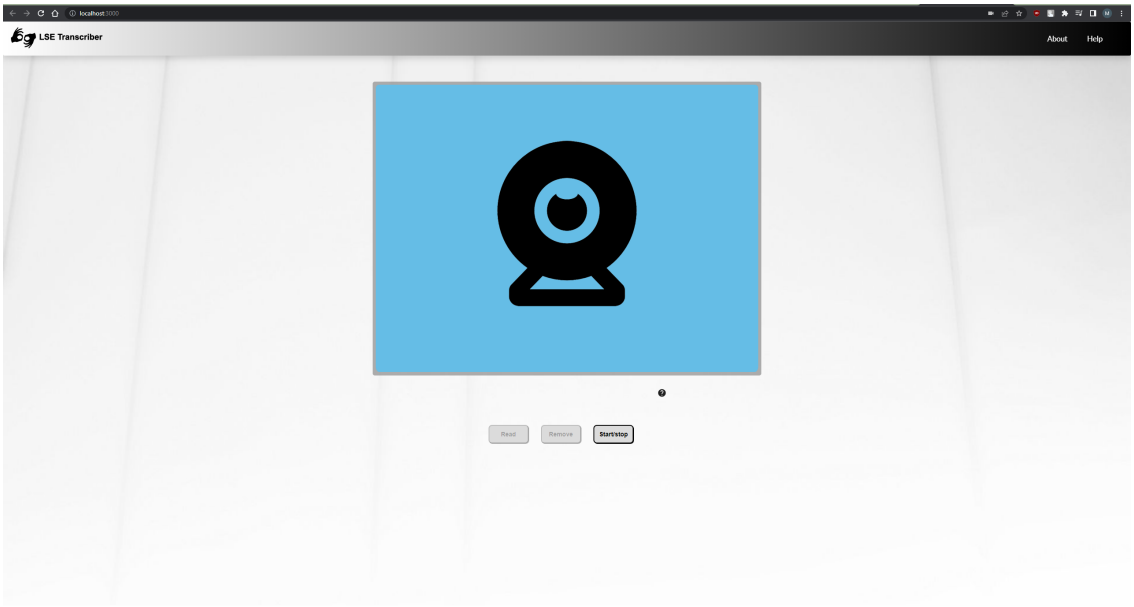


Figura 47: Vista principal de LSETranscriber

Para utilizar la aplicación, deberá asegurarse de que le ha dado los permisos de cámara. Esta opción es diferente en base al navegador utilizado, pero usualmente saldrá un pop-up para permitirlo.

A grandes rasgos, para usar LSETranscriber debe realizar signos en el modo de grabación. Los signos deberán estar en el cuadro de la cámara y con la mano apuntando a esta. En el caso de que al momento de realizar una predicción no se encuentre ninguna mano en la cámara, se predecirá como un espacio en blanco. En el caso de que ocurra dos veces seguidas se parará la grabación.

Una vez LSETranscriber tenga permisos de cámara, se podrá ver en el cuadrado del centro. Para realizar predicciones hay que usar los botones de la parte inferior.

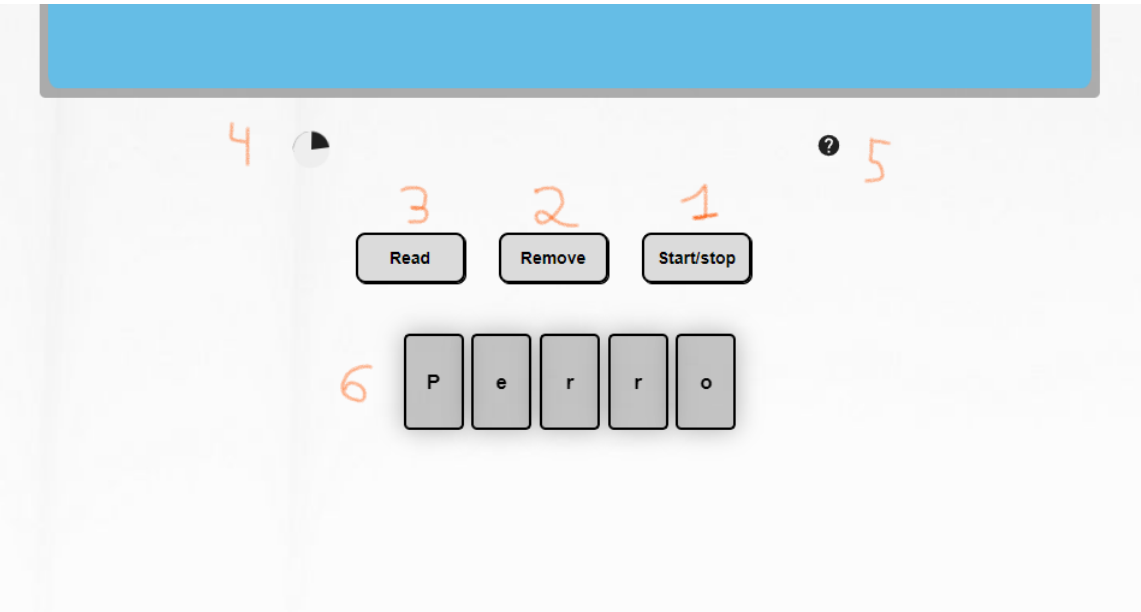


Figura 48: Botones de predicción

Se explicará las diferentes funcionalidades en base a la numeración de la figura:

1. Con este botón podremos comenzar el proceso de grabación. También podremos terminarlo, aunque es más fácil terminarlo con dos predicciones sin que se encuentre una mano en la imagen.
2. Con este botón borraremos la predicción en su totalidad, empezando una nueva.
3. Clicando en 'Read' haremos que se lea la predicción en voz alta.
4. Este temporizador sale cuando la aplicación está grabando signos para predecir, de tal forma que cada vez que completa una vuelta toma una captura.
5. Si pasa el ratón por encima de este signo de interrogación se abrirá un desplegable como este, en el que encuentra información sobre los signos reconocidos por LSETranscriber, como se puede ver en la figura 49. También útiles en caso de olvido.
6. Los botones con diferentes letras de la parte inferior es la predicción. Cada vez que se esté en el modo de grabación, y pasados los dos segundos que toma cada captura, se añadirá un botón con la letra predicha. En el caso de que tanto el usuario como la aplicación se confundan en alguna letra, haciéndole click se puede eliminar esa únicamente.

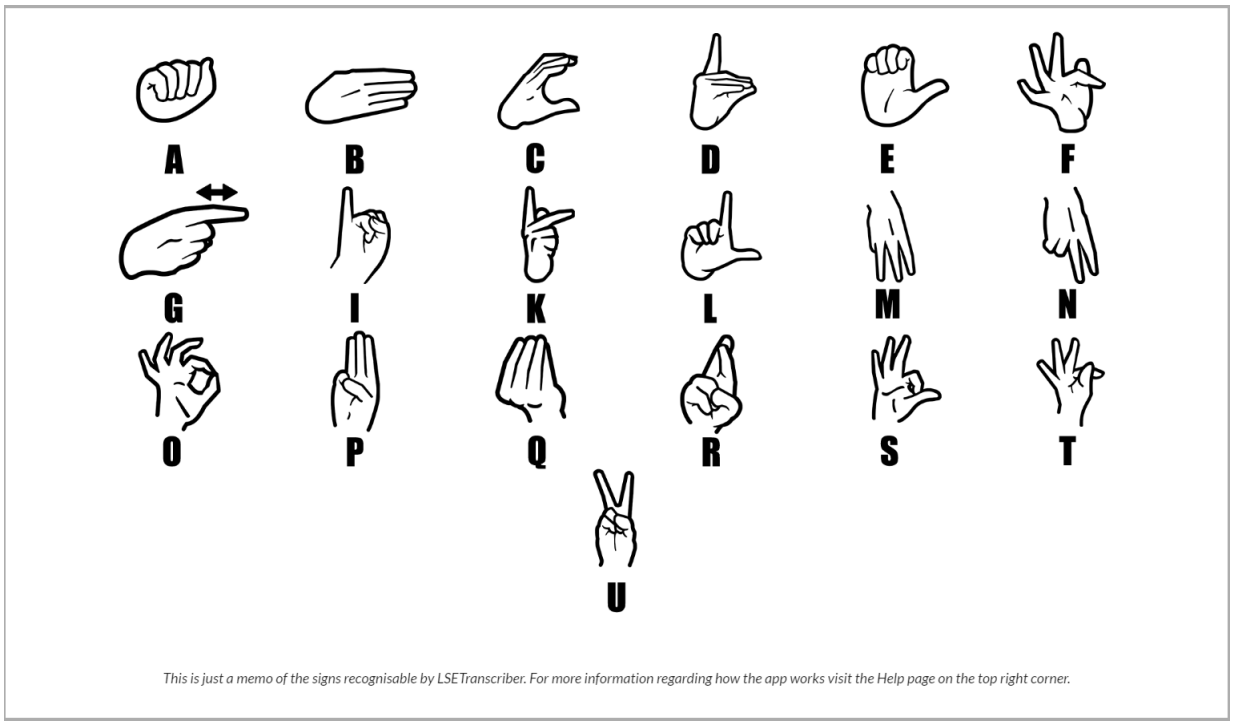


Figura 49: Tabla con signos reconocibles por LSETranscriber

Para navegar a otras vistas, utilizaremos los botones de la parte superior de la aplicación.

Clicando en la parte izquierda superior, en el logo de la aplicación, volveremos a esta vista de predicción.

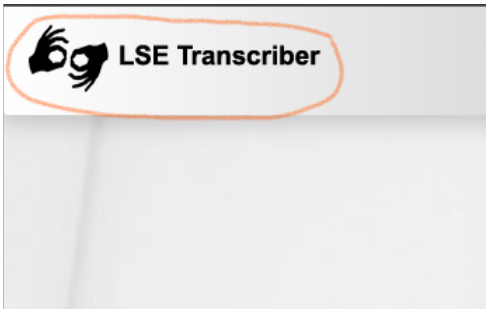


Figura 50: Botón para ir a la página principal

Si queremos ir a la vista de información sobre la aplicación o de ayuda, podemos usar los botones correspondientes en la parte superior derecha.

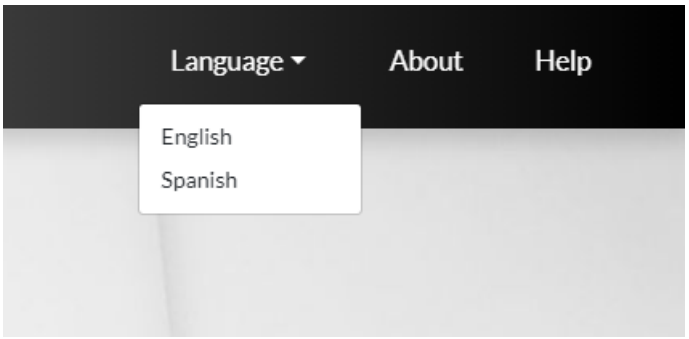


Figura 51: Botones para ir a páginas de información y ayuda

La aplicación está disponible en inglés y español. Para cambiar el idioma, basta con hacer click en el desplegable y elegir el idioma que más convenga. Se puede cambiar en cualquier momento.

10.4 Manual del Programador

Más adelante se contemplarán las ampliaciones que se quedaron en el tintero 11.2. Aquí haré una pequeña descripción del sistema como tal, y algunos puntos que habría que tener en cuenta en el caso de realizar las ampliaciones mencionadas.

Para volver a entrenar el proyecto, se necesitará partir de los scripts de entrenamiento proporcionados. Por desgracia, no serán de mucha utilidad en el caso de querer utilizar los puntos claves proporcionados por la librería de mediapipe. Esto implicaría tener que realizar un script para transformar el dataset en listas de puntos. Cambiando un poco la forma de proceso de las imágenes inicial se podría reutilizar gran parte del código del entrenamiento.

En caso de querer ampliar la funcionalidad de la aplicación web, sería muy simple. Todo lo relacionado con la parte de frontend se encuentra en la carpeta app, mientras que lo relacionado con la API REST se encuentra en la carpeta restapi.

El componente más complicado a la hora de ampliar sobre la aplicación web es el uso de React-Redux. Redux permite centralizar la información de la aplicación, de tal forma que es accesible desde cualquier componente sin necesidad de tener que compartir esa información por toda la cadena jerárquica de componentes. La implementación de Redux utilizada es Redux-toolkit, ya que me pareció la más asequible de las tres formas en las que se puede implementar.

A grandes rasgos, redux consiste en: Un estado inicial, definido en predictionSlice. Si quieres acceder a ese estado, se realiza a través de useSelector, desde cualquier clase.
Para modificar el valor del estado, hay que hacerlo completamente, no se puede añadir al estado. Esta modificación se realiza a través de reducers, que también se ubican en el predictionSlice.
En el caso de que la actualización de algún campo del estado sea asíncrona, se deberá usar asyncThunks. Hay un ejemplo también en el predictionSlice.

En el caso de querer añadir funcionalidad a la API REST, se deberán crear nuevos endpoints en el archivo api.py. Para consumir este servicio desde la aplicación web, se encuentra la clase API.js, que actúa de middleware entre el front y el back.

Si no se es familiar con las tecnologías mencionadas, lo mejor es mirar la documentación oficial:

- Redux [12]
- Flask [5]

11 Conclusiones y Ampliaciones

Aquí explicaré el resultado final del desarrollo, y si este ha cumplido con las expectativas iniciales. También contemplaré si las tecnologías utilizadas y las soluciones que he encontrado a los diferentes problemas han sido óptimas. También hablaré de funcionalidades que me hubiera gustado implementar, y el futuro de la aplicación.

11.1 Conclusiones

Recordando los objetivos que tenía el proyecto en su comienzo 2.2, podemos ver que LSETranscriber cumple con todos en cierta medida. Estos objetivos fueron adaptados para que el proyecto fuera posible en primer lugar. Sin duda el proyecto cumple con las expectativas esperadas, especialmente teniendo en cuenta las dificultades encontradas en su desarrollo con las que no se contaba inicialmente, como por ejemplo la necesidad de tener que crear un dataset manualmente.

Todas las tecnologías elegidas han resultado ser las correctas, han sido más que suficiente para el desarrollo completo de la aplicación, no suponiendo en ningún momento un impedimento para este. Por lo tanto, podemos decir que el proyecto ha sido un éxito.

11.2 Ampliaciones

A pesar de haber cumplido con las expectativas que se propusieron para el proyecto, hay varios aspectos de este que sería muy interesante mejorar. Empezando por el reconocimiento de todos los signos del alfabeto dactilológico. Ciertamente esto supone un incremento muy amplio tanto en las imágenes necesarias para entrenar el modelo, además de un entendimiento superior de como funciona el aprendizaje automático, ya que los signos restantes son signos dinámicos, esto significando que no pueden ser reconocidos estáticamente a través de imágenes, por lo tanto habría que usar varias imágenes para poder decidir que signo es.

Otro aspecto que hubiera aportado mucha usabilidad a la aplicación es el que estuviera alojada en un servidor mucho más potente, a poder ser con una tarjeta gráfica dedicada para las predicciones. Esto permitiría que muchos más usuarios pudieran usar LSETranscriber simultáneamente, lo cual es una cualidad muy importante. Esto no es viable en el proyecto desarrollado al ser un TFG y no una aplicación cuyo objetivo es ponerse en producción, especialmente al estar alojada en un servidor proporcionado por la universidad.

Otro aspecto que hubiera mejorado la aplicación sería modificar las imágenes antes de usarlas para entrenar el modelo. Como descubrí durante la implementación de la API REST, la librería de mediapipe me proporciona puntos donde se encuentran los diferentes huesos de la mano. Estos datos hubieran sido mucho más eficientes para realizar un modelo con un alto grado de exactitud, ya que no solo los datos de entrada pesarían mucho menos, haciendo que el entrenamiento fuera más rápido y haciendo que el dataset ocupara menos espacio, si no que también reduciría el ruido que se encuentra en las imágenes de signos. Esto se debe a que dentro de una imagen, solo el signo es relevante para la predicción. Esto implica que el modelo tenga que aprender a extraerlo antes de poder continuar con el entrenamiento, eliminando el fondo y otros elementos que no deberían influir en la predicción. Utilizando los puntos que nos proporciona mediapipe sabemos que todos ellos son fundamentales para la predicción, por lo que nos ahorraríamos ese paso.

Un problema asociado a esto sería que el proceso de dataset expansion seguido para obtener más imágenes de las mismas no funcionaría en su totalidad, por lo tanto necesitaría más información de entrada.

Otro aspecto que me hubiera gustado refinar es la exactitud del modelo. Esta es bastante buena, pero hay algunas desventajas de haber realizado el modelo yo mismo. Lo primero es que no se usaron personas acostumbradas a utilizar el lenguaje de signos, por lo tanto estos pueden ser poco precisos o en algunos casos casi incorrectos. Otro aspecto es que al usar personas de mi entorno falta diversidad racial en los datos. Esto seguramente produzca que el modelo tenga un rendimiento considerablemente inferior en personas de otras razas, lo cual no es admisible en una aplicación profesional.

12 Presupuesto

En este apartado se comentará el presupuesto desde el punto de vista de la empresa para este proyecto. Comenzaremos con la definición de nuestra empresa y luego veremos los costes de las diferentes tareas que hay que llevar a cabo en la realización del sistema solicitado.

12.1 Definición de la empresa

En este apartado se explicará la definición de la empresa, tomando en cuenta los salarios de los empleados, sus horas productivas y demás costes indirectos, para observar si la empresa puede ser rentable.

12.1.1 Salarios de los empleados

Empezaremos con los salarios de nuestros empleados. Para ello, mostraremos su sueldo anual bruto, al que le sumaremos el 30 % que es el coste de la seguridad social. De esta forma obtendremos el coste a la empresa por cada empleado. Para calcular el coste total de los diferentes tipos de empleados multiplicaremos este valor por el número de empleados de ese tipo. Se obtiene la siguiente tabla:

Recurso	Cantidad	Sueldo Bruto Año	Coste seguridad social	Coste salarial Año	TOTAL
FullStackEngineer	1	35.000,00 €	10.500,00 €	45.500,00 €	45.500,00 €
				TOTAL	45.500,00 €

Figura 52: Salarios de los empleados

12.1.2 Costes directos e indirectos de los empleados

Ahora necesitamos saber el coste indirecto y directo que supone cada empleado. Suponemos que todos están empleados a tiempo completo con contratos indefinidos, lo cual supondrían 1720 horas al año. Llamamos productividad al porcentaje de su trabajo que influye directamente en el producto que se le entrega al cliente. En base a los precios hora y la productividad, podemos saber qué parte del sueldo de cada empleado se considera un coste directo y cuál indirecto.

Personal	Total	Prod. (%)	Coste Directo	CI (%)	Coste Indirecto	Horas / año	Horas prod. (total empresa)	Precio / hora (sin beneficio)	Facturación	Precio / hora	N Trabajadores
FullStackEngineer	45.500,00 €	100%	45.500,00 €	0%	0,00 €	1720	1720	36,00 €	77.400,00 €	45,00 €	1
TOTAL	45.500,00 €		45.500,00 €		0,00 €		479		77.400,00 €		

Figura 53: Productividad del personal

12.1.3 Costes de los medios de producción

Estos costes se corresponden con software y hardware que los empleados necesitan obligatoriamente para realizar su trabajo.

Costes de los medios de producción						
Equipo / Licencia	***Unid.***	***Precio***	***Coste Total***	***Coste año***	***Tipo***	***Plazo***
Microsoft Windows 10	1	180,00 €	180,00 €	180,00 €	Amortización	12
Microsoft Office Professional 2020	1	19,80 €	19,80 €	178,20 €	Alquiler	9
Microsoft Project	1	1.017,00 €	1.017,00 €	1.017,00 €	Amortización	24
IntelliJ PyCharm	1	19,90 €	19,90 €	238,80 €	Alquiler	9
Ordenador	1	2.000,00 €	2.000,00 €	2.000,00 €	Amortización	48
Monitor	2	300,00 €	600,00 €	600,00 €	Amortización	48
			TOTAL	**4.214,00 €**		

Figura 54: Costes de los medios de producción

Se pueden observar dos tipos de costes, las amortizaciones y los alquileres. Las amortizaciones tendrán un plazo en el que se espera amortizar los diferentes productos, que influye en el tipo de producto que se elija. Los alquileres son costes mensuales, por lo tanto, tenemos que calcular el gasto anual.

12.1.4 Costes indirectos

Los costes indirectos se corresponden con asuntos relacionados con el alquiler y mantenimiento de la oficina y todo lo relacionado con ella. Se evalúa en coste mensual y es multiplicado por 12 para calcular el coste anual.

Costes indirectos		
Nombre	***Coste mensual***	***Coste anual***
Alquiler	184,00 €	2.208,00 €
Utilidades	20,00 €	240,00 €
Alimentación	240,00 €	2.880,00 €
	TOTAL	**5.328,00 €**

Figura 55: Costes indirectos

12.1.5 Definición de la empresa

Después de tener los costes anteriores, podemos calcular la definición de la empresa. Para ello, tomaremos los costes directos de la tabla de productividad del personal, y los costes indirectos. Los costes indirectos son la suma de los costes indirectos de la tabla de productividad del personal, los costes de servicios y los costes de los medios de producción. El beneficio es 20 % ya que es el beneficio esperado por la empresa. Sumando los costes indirectos y los directos obtenemos el coste total, y comparándolo con la facturación podemos ver si la empresa es rentable.

Definición de la empresa		
Nº	**CONCEPTO**	**IMPORTE**
1	Total de los costes directos	45.500,00 €
2	Total de los costes indirectos	9.542,00 €
3	Suma de los costes directos e indirectos	55.042,00 €
4	Beneficio deseado (20%)	11.008,40 €
5	Coste total (suma de los costes directos, indirectos y beneficios)	66.050,40 €
6	Facturación posible en función de las horas de producción y de los precios por hora calculados)	77.400,00 €
7	Margen entre el coste total y la facturación (relación entre 5 y 6)	17%

Figura 56: Definición de la empresa

12.2 Partidas de costes del proyecto

Aquí se detallarán todos los costes de las actividades relacionadas con el proyecto a desarrollar, separadas por partidas para facilitar su lectura. Finalmente, se realizará el presupuesto de la empresa para este proyecto. Se podrán observar con mas detenimiento en el archivo adjunto a esta memoria.

12.2.1 Partida de Estudios preliminares

Esta partida consistirá en las diferentes tareas de estudio y diseño preliminar. Serán llevadas a cabo antes del desarrollo del sistema y se centrarán en obtener los requisitos para realizar un diseño del sistema entero, facilitando la estimación del desarrollo.

Estudio preliminar									
I1	*I2*	*I3*	*Descripción*	*Cantidad*	*Unidades*	*Precio*	*Subtotal (3)*	*Subtotal (2)*	*Total*
1			Modelo						432,00 €
	1		Buscar información sobre la realización de un modelo clasificatorio					360,00 €	
		1	FullStackEngineer	10	horas	36,00 €	360,00 €		
	2		Elección de lenguaje de programación y librerías					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	3		Elección de IDE					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		36,00 €
2			Generar Dataset						
	1		Buscar datasets existentes					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
3			RESTAPI						108,00 €
	1		Buscar información sobre tecnologías para hacer la RESTAPI					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	2		Elección de lenguaje de programación y librerías					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	3		Elección de IDE					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
								TOTAL	**576,00 €**

Figura 57: Partida de Estudio

12.2.2 Partida de Documentación

Aquí se incluirán todas las tareas relacionadas con la confección de los diferentes apartados de la memoria del proyecto.

Documentación									
I1	*I2*	*I3*	*Descripción*	*Cantidad*	*Unidades*	*Precio*	*Subtotal (3)*	*Subtotal (2)*	*Total*
1			Memoria						5.832,00 €
	1		Resumen					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	2		Palabras clave					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	3		Resumen de la motivación, Objetivos y Alcance del proyecto					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	4		Otros apartados					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	5		Justificación del proyecto					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	6		Objetivos del proyecto					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	7		Evaluación de alternativas					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	8		Conceptos					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	9		Herramientas					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	10		Tecnologías					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	11		Planificación					180,00 €	
		1	FullStackEngineer	5	horas	36,00 €	180,00 €		
	12		Resumen del presupuesto					288,00 €	
		1	FullStackEngineer	8	horas	36,00 €	288,00 €		
	13		Descripción del dataset					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	14		Generación del modelo					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	15		Determinación del Alcance del Sistema					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	16		Obtención de los Requisitos del Sistema					216,00 €	
		1	FullStackEngineer	6	horas	36,00 €	216,00 €		
	17		Identificaicón de Actores del Sistema					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	19		Especificación de Casos de uso					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	20		Descrición de los Subsistemas					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	21		Descripción de las Interfaces entre Subsistemas					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	22		Análisis de Casos de Uso y Escenarios					216,00 €	
		1	FullStackEngineer	6	horas	36,00 €	216,00 €		
	23		Descripción de la Interfaz					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	24		Descripción del Comportamiento de la Interfaz					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	25		Diagrama de Navegabilidad					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	26		Especificación del Plan de Pruebas					216,00 €	
		1	FullStackEngineer	6	horas	36,00 €	216,00 €		
	27		Diagrama de contexto					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	28		Diagramas de Despliegue					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	29		Diagramas de Interacción y Estados					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	30		Diagramas de Actividades					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	31		Diseño de la Interfaz					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	32		Pruebas Unitarias					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	33		Pruebas de Integración y del Sistema					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	34		Pruebas de Usabilidad y Accesibilidad					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	35		Pruebas de Rendimiento					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	36		Estándares y Normas seguidos					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	37		Lenguajes de programación					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	38		Herramientas y Programas usados para el desarrollo					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	39		Problemas Encontrados					216,00 €	
		1	FullStackEngineer	6	horas	36,00 €	216,00 €		
	40		Descripción Detallada de las Clases					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	41		Pruebas Unitarias					216,00 €	
		1	FullStackEngineer	6	horas	36,00 €	216,00 €		
	42		Pruebas de Integración y del Sistema					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	43		Pruebas de Usabilidad y Accesibilidad					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	44		Pruebas de Accesibilidad					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	45		Pruebas de Rendimiento					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	46		Manual de Instalación					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	47		Manual de Ejecucción					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	48		Manual de Usuario					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	49		Manual del Programador					216,00 €	
		1	FullStackEngineer	6	horas	36,00 €	216,00 €		
	50		Conclusiones					216,00 €	
		1	FullStackEngineer	6	horas	36,00 €	216,00 €		
	51		Ampliaciones					144,00 €	
		1	FullStackEngineer	4	horas	36,00 €	144,00 €		
	52		Desarrollo del presupuesto detallado					432,00 €	
		1	FullStackEngineer	12	horas	36,00 €	432,00 €		
	53		Glosario y Diccionario de Datos					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	54		Contenidos					72,00 €	
		1	FullStackEngineer	2	horas	36,00 €	72,00 €		
	55		Código Ejecutable e Instalación					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	56		Ficheros de Configuración					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	57		Índice alfabético					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
	58		Código Fuente					36,00 €	
		1	FullStackEngineer	1	horas	36,00 €	36,00 €		
								TOTAL	5.832,00 €

Figura 58: Partida de Documentación

12.2.3 Partida de Implementación

En esta partida se mostrarán las tareas de creación del sistema, tanto la construcción como las pruebas y el despliegue.

Implementación									
l1	*l1*	*l3*	*Descripción*	*Cantidad*	*Unidades*	*Precio*	*Subtotal(3)*	*Subtotal (2)*	*Total*
1			Modelo						8.568,00 €
	1		Configuración del IDE					144,00 €	
		1	FullStackEngineer	4	horas	36	144,00 €		
	2		Probar diferentes combinaciones de capas y funciones de activación					432,00 €	
		1	FullStackEngineer	12	horas	36	432,00 €		
	3		Realizar el mejor modelo con transfer learning					4.968,00 €	
		1	FullStackEngineer	138	horas	36	4.968,00 €		
	4		Entrenar al modelo					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	5		Realizar fine tunning al modelo					252,00 €	
		1	FullStackEngineer	7	horas	36	252,00 €		
	6		Buscar datasets existentes					36,00 €	
		1	FullStackEngineer	1	horas	36	36,00 €		
	7		Realizar capturas a gestos de vídeos de internet					2.592,00 €	
		1	FullStackEngineer	72	horas	36	2.592,00 €		
	8		Realizar data augmentation al dataset					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
2			RESTAPI						756,00 €
	1		Configuración del IDE					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	2		Crear un endpoint que soporte peticiones POST con imágenes					216,00 €	
		1	FullStackEngineer	6	horas	36	216,00 €		
	3		Cargar el modelo					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	4		Identificar la mano en la imagen					108,00 €	
		1	FullStackEngineer	3	horas	36	108,00 €		
	5		Modificar imagen y reescalarla					36,00 €	
		1	FullStackEngineer	1	horas	36	36,00 €		
	6		Mandar la imagen al modelo para su predicción					180,00 €	
		1	FullStackEngineer	5	horas	36	180,00 €		
	7		Devolver la predicción al usuario					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
3			Webapp						1.476,00 €
	1		Configuración del IDE					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	2		Realizar un mockup de las vistas					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	3		Implementar vista principal					288,00 €	
		1	FullStackEngineer	8	horas	36	288,00 €		
	4		Permitir al usuario cargar imágenes					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	5		Permitir al usuario obtener una predicción de las imágenes					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	6		Implementar TTS para las imágenes predichas					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	7		Conectar con la RESTAPI y mandarle las imágenes del usuario					252,00 €	
		1	FullStackEngineer	7	horas	36	252,00 €		
	8		Realizar barra de navegación					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	9		Diseñar logo					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
	10		Diseñar look & feel de la aplicación					216,00 €	
		1	FullStackEngineer	6	horas	36	216,00 €		
	11		Generar vista de ayuda					144,00 €	
		1	FullStackEngineer	4	horas	36	144,00 €		
	12		Generar vista con información sobre la aplicación					72,00 €	
		1	FullStackEngineer	2	horas	36	72,00 €		
								TOTAL	**10.800,00 €**

Figura 59: Partida de Implementación

12.3 Presupuesto de costes

Este presupuesto de la empresa. Contiene todos los costes reales de las diferentes actividades en la realización del proyecto. No tiene en cuenta el beneficio que se espera obtener.

Presupuesto de costes		
Cod.	***Partida***	***Total***
1	Diseño preliminar	576,00 €
2	Documentación	5.832,00 €
3	Implementación	10.800,00 €
	TOTAL	**17.208,00 €**

Figura 60: Presupuesto de costes

13 Apéndices

En este apartado se comentará el contenido de los archivos entregados junto a esta documentación, así como los pasos para instalar los diferentes proyectos y el código fuente de todos ellos.

13.1 Contenido Entregado en el Archivo Adjunto

Aquí detallaré cada directorio del archivo entregado, así como el contenido, función e instalación si se requiriese de los archivos. Importante tener el cuenta que todas las carpetas mostradas incluyen dentro un archivo README.txt con más información del contenido y como ejecutarlo en caso de que así se desee.

13.1.1 Contenidos

codigo
Esta carpeta contiene a su vez tres carpetas:

- aplicacion: Contiene la aplicación final de LSETranscriber, que contiene la aplicación web y la API REST
- aprendizaje: Esta carpeta contiene dos proyectos.
  - El primero, llamado "bounding box y real timeçontiene una aplicación nativa con OpenCV para predicción en tiempo real, realizando una captura por cada frame obtenido de la webcam del usuario. Fue una prueba para el funcionamiento del modelo y la obtención de la zona de la mano de las diferentes capturas.
  - El segundo proyecto es "scripts generacion modelos". Como su nombre indica, este contiene tres de los scripts que fueron utilizados para la creación del modelo y su posterior validación. Hubo muchos más durante el proceso, pero se han incluido los más significativos en el avance.
- dataset: Aquí se encuentran los dos datasets finales usados para el entrenamiento del modelo. Ya que incluir el dataset expandido ocupaba mucho espacio, son los datasets iniciales con un script de batch que permite generar las imágenes aumentadas, lo que compondría el dataset utilizado realmente. Son dos carpetas:
  - dataset: Esta carpeta tiene el dataset de entrenamiento utilizado.
  - validation: Esta carpeta contiene el dataset utilizado para validación.

diagramas
Esta carpeta contiene todos los diagramas utilizados y mostrados en esta documentación. Se incluyen tres carpetas:

- drawio: Esta carpeta contiene el diagrama realizado en drawio, con esa extensión.
- pdf: Aquí se pueden encontrar todos los diagramas en formato pdf.
- plantuml: Aquí se pueden encontrar los diagramas que fueron realizados con plantuml en esa misma extensión, en caso de que se quiera ver como fueron generados.

documentación
Aquí se encuentran las diferentes imágenes usadas en esta documentación, así como este pdf en sí.
La carpeta imagenes tiene la siguiente estructura:

- entrenamiento: Imágenes relacionadas con el entrenamiento del modelo, exactitud, consumo y rendimiento.
- mockups: Imágenes de los mockups realizados para cada vista de la aplicación web.
- planificacion: Capturas del archivo Project de la planificación inicial, incluidos en este documento.
- presupuesto: Capturas del archivo Excel en el que se encuentra el presupuesto que en esta documentación se muestra.
- signos: Imágenes de los signos sin fondo, que son usadas en la documentación y también en la aplicación web.
- tests: Imágenes con los resultados de los diferentes tests realizados.
- tutoriales: Capturas de pantalla utilizadas en este documento para realizar los manuales.

planificacion
En esta carpeta se puede encontrar los archivos Project con la planificación inicial y final.

prespuesto
Aquí se encuentra el Excel con el presupuesto detallado.

pruebas de carga
En este directorio se encuentra el proyecto con el que se realizaron las pruebas de carga con Gatling. Dentro de este proyecto se encuentra Gatling como tal, la prueba generada y modificada, la imagen de prueba usada, y el archivo HAR que contiene los diferentes pasos que realicé y grabé, para posteriormente generar la clase java con esas peticiones.

13.1.2 Código Ejecutable e Instalación

Para la instalación se tendrá en cuenta solo la aplicación principal, LSETranscriber. Este se encuentra en: codigo -> aplicacion -> LSETranscriber

Para ello, solo se necesitará tener instalado Docker en el sistema. Para lanzar la aplicación, habrá que abrir una terminal en el directorio raíz del proyecto, a la misma altura que el docker-compose.yml.

Una vez que se disponga de una consola en esa dirección, se ejecutara: "docker-compose up –build -d", lo cuál al cabo de un rato desplegará la aplicación web en http://localhost:3000 y la API REST en http://localhost:5000. Para parar la aplicación hay dos alternativas, hacerlo desde Docker Desktop, o en la misma consola de antes ejecutar: "docker-compose down".

Referencias

[1] Amrita Thakur et al. «Real Time Sign Language Recognition and Speech Generation». En: *Journal of Innovative Image Processing (JIIP)* 02.02 (2020), págs. 65-76. DOI: https://doi.org/10.36548/jiip.2020.2.001.

[2] Paulo Trigueiros et al. «Vision-based Portuguese Sign Language Recognition System». En: *Springer Science and Business Media LLC* (2014). DOI: https://core.ac.uk/download/pdf/55638597.pdf.

[3] *Django*. URL: https://www.djangoproject.com/. (accessed: 06.17.2022).

[4] *Docker*. URL: https://docs.docker.com/get-docker/. (accessed: 05.03.2022).

[5] *Flask-RESTX*. URL: https://flask-restx.readthedocs.io/en/latest/. (accessed: 03.03.2022).

[6] *Gatling*. URL: https://gatling.io/. (accessed: 03.03.2022).

[7] *Jest*. URL: https://jestjs.io/es-ES/. (accessed: 03.03.2022).

[8] *LSETranscriber*. URL: https://github.com/MarcosTobias/LSETranscriber. (accessed: 05.03.2022).

[9] *MNIST sign-language dataset*. URL: https://www.kaggle.com/datamunge/sign-language-mnist/data. (accessed: 05.04.2022).

[10] *PyTorch*. URL: https://pytorch.org/. (accessed: 06.17.2022).

[11] *React*. URL: https://es.reactjs.org/. (accessed: 03.03.2022).

[12] *Redux*. URL: https://es.redux.js.org/. (accessed: 03.03.2022).

[13] *Tensorflow*. URL: https://www.tensorflow.org/overview?hl=es-419. (accessed: 03.03.2022).