



“SISTEMA PARA EL MONITOREO, DETECCIÓN Y ALERTA DE
SOMNOLENCIA DEL CONDUCTOR MEDIANTE VISIÓN ARTIFICIAL,
COMUNICACIÓN INALÁMBRICA Y GEOLOCALIZACIÓN”

Segundo Reporte Parcial

Lista de actividades

- Maquetación web
- Investigación de la documentación del módulo 3G/4G LTE-Base Hat
- Enlace de Amazon S3 con el sistema backend
- Creación de los servicios backend
- Familiarizarse con el entorno de desarrollo de la NVIDIA Jetson Nano
- Implementar algoritmo para la detección del rostro y ojos
- Implementar puntos faciales en el rostro y métrica MOR

Autores:

Alan Eduardo Gamboa Del
Ángel
Maite Paulette Díaz Martínez

Asesores:

M.en C. Niels Henrik Navarrete
Manzanilla
Dr. Rodolfo Vera Amaro

21 de Abril 2023

Índice

1. Maquetación web	5
1.1. Objetivo	5
1.2. Descripción	5
1.3. Resultados	5
2. Investigación de la documentación del módulo 3G/4G LTE-Base Hat	8
2.1. Objetivo	8
2.2. Descripción	8
2.3. Resultados	10
3. Enlace de Amazon S3 con el sistema backend	11
3.1. Objetivo	11
3.2. Descripción	11
3.3. Resultados	13
4. Creación de los servicios backend	14
4.1. Objetivo	14
4.2. Descripción	14
4.3. Resultados	20
5. Familiarizarse con el entorno de desarrollo de la NVIDIA Jetson Nano	21
5.1. Objetivo	21
5.2. Descripción	21
5.3. Resultados	23
6. Implementar algoritmo para la detección del rostro y ojos	26
6.1. Objetivo	26
6.2. Descripción	26
6.3. Resultados	29
7. Implementar puntos faciales en el rostro y la metrica MOR	31
7.1. Objetivo	31
7.2. Descripción	31
7.3. Resultados	34
8. Conclusiones	38
9. Bibliografía	39

Índice de figuras

1.	Página Principal - Layout.jsx	6
2.	Vista Reporte Incidencia Incidencia - Incidencia.jsx	6
3.	Vista Ubicacion	7
4.	Vista Conductores - Conductores.jsx	7
5.	Módulo Base-Hat SIM7600G-H	8
6.	Instalación de librerías	8
7.	Instalación de librerías	9
8.	Minicom	9
9.	Actualización de drivers	9
10.	Establecer dirección IP	10
11.	Configuración de servicio de almacenamiento S3	11
12.	Generación de Endpoint de GraphQL	11
13.	Generación de Endpoint de GraphQL	11
14.	Generación de Endpoint de GraphQL	12
15.	Generación de Endpoint de GraphQL	12
16.	Generación de Endpoint de GraphQL	12
17.	Generación de Endpoint de GraphQL	13
18.	Tablas generadas mediante los schemas definidos	13
19.	Función getConductor	14
20.	Función listConductors	15
21.	Función getIncidencia	15
22.	Función listIncidencias	16
23.	Función createConductor	16
24.	Función updateConductor	17
25.	Función deleteConductor	17
26.	Función createIncidencia	18
27.	Función oncreateIncidencia	18
28.	Consola de AWS	19
29.	Funcionamiento de Appsync	19
30.	Crear Conductor	20
31.	Resultado	20
32.	Ranura para tarjeta microSD para almacenamiento	23
33.	Periféricos de entrada para el Kit de desarrollo Jetson Nano.	24
34.	instalación y configuracion completada.	24
35.	Instalación de Python3 y librerías.	25
36.	Instalación del IDE PyCharm.	25
37.	Código de la detección de rostro y ojos de una imagen, implementando el modelo entrenado	26
38.	Código de la detección de rostro y ojos de una imagen implementando el modelo entrenado	27
39.	Código de la detección de rostro y ojos de una imagen implementando el modelo entrenado	28
40.	Código de la detección de rostro y ojos de una imagen implementando el modelo entrenado	29

41.	Implementación para la detección de rostro y ojos aplicando el modelo entrenado . . .	30
42.	Captura de pantalla del código para la detección de apertura de la boca	35
43.	Captura de pantalla del código para la detección de apertura de la boca	35
44.	Prueba de la detección de apertura de boca sin mostrar valores de apertura	36
45.	Prueba de la detección de apertura de boca con valores de apertura, boca cerrada .	36
46.	Prueba de la detección de apertura de boca con valores de apertura, boca abierta .	37

Índice de tablas

1. Maquetación web

1.1. Objetivo

Crear la aplicación web e implementar el sistema de diseño de manera local.

1.2. Descripción

ReactJs

Para el desarrollo del front-end del presente proyecto, se hará uso de la librería de diseño de ReactJs. ReactJs facilita la creación de componentes reutilizables e interactivos para las interfaces de usuario.

Los componentes que darán lugar a las vistas del presente proyecto son los siguientes:

- `Layout.jsx`: Este componente será la vista principal de la Aplicación Web. Se trata de un diseño tipo *dashboard* que contendrá una sección principal que contendrá etiquetas para poder ingresar a las diferentes vistas de la aplicación. Además estará compuesta también de una sección secundaria que mostrará el contenido de dichas vistas.
- `Incidencias.jsx`: Este componente se encargará de mostrar todas las incidencias registradas en la base de datos. Las incidencias serán desplegadas en forma de lista.
- `Incidencias.jsx`: Este componente de mostrar un reporte de incidencia a detalle. Contendrá una ventana que permitirá ver el video del momento de la incidencia registrada. Así como los datos de la fecha y hora. Además de botones para poder confirmar o rechazar la incidencia. Finalmente contendrá el nombre del conductor además de una opción para poder consultar la ubicación en tiempo real del conductor.
- `Ubicación.jsx`: Este componente mostrará la ubicación en tiempo real del conductor con ayuda del servicio de diseño de mapas Leaflet.
- `Conductores.jsx`: Este componente mostrará todos los conductores registrados en la base de datos en forma de lista.

1.3. Resultados

De acuerdo con los componentes explicados anteriormente, las vistas que contendrá la aplicación web son las siguientes:

- Página Principal

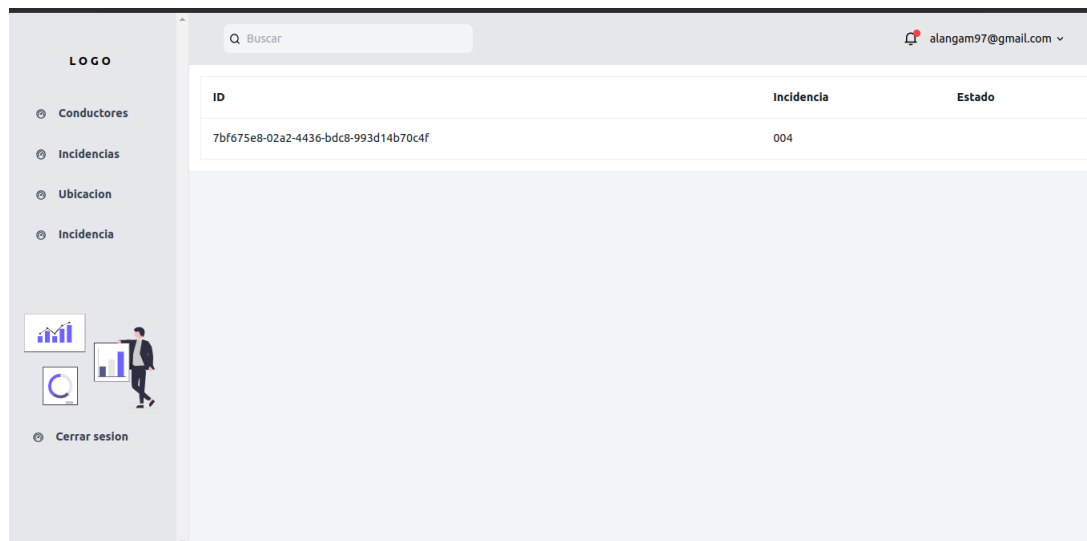


Figura 1: Página Principal - Layout.jsx

- Reporte de Incidencia



Figura 2: Vista Reporte Incidencia Incidencia - Incidencia.jsx

- Ubicación

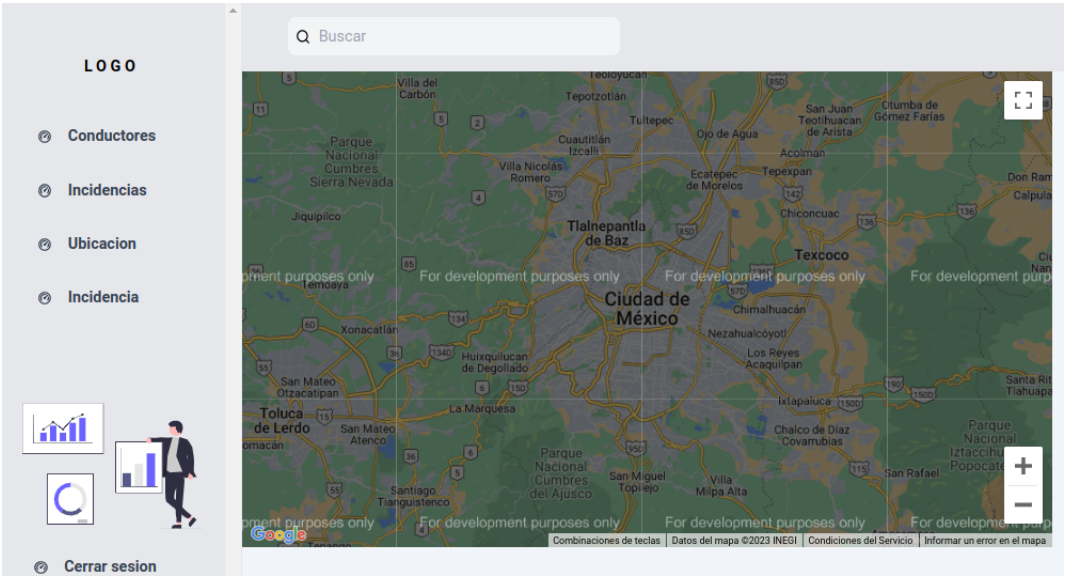


Figura 3: Vista Ubicacion

■ Conductores

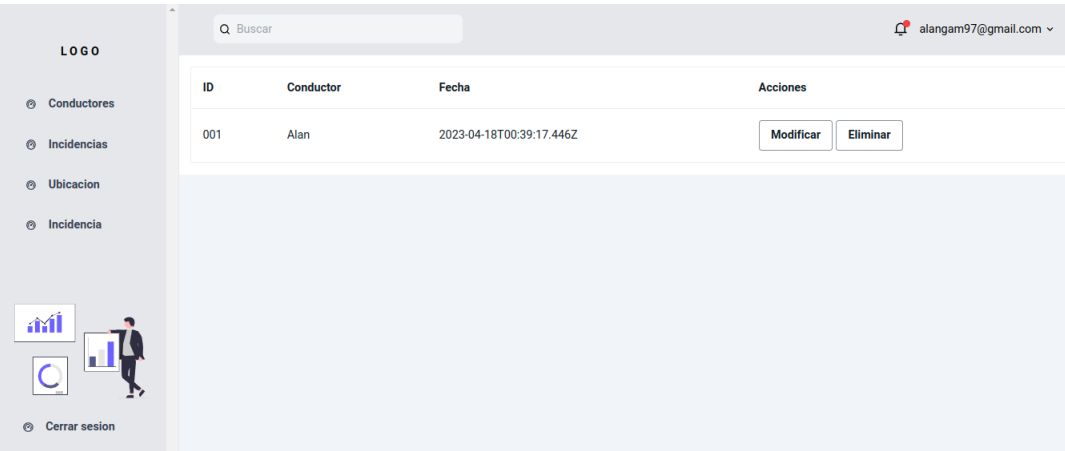


Figura 4: Vista Conductores - Conductores.jsx

2. Investigación de la documentación del módulo 3G/4G LTE-Base Hat

2.1. Objetivo

Familiarizarse con las distintas funciones y comandos, así como el entorno de desarrollo que ofrece el dispositivo Base-Hat

2.2. Descripción

Para el presente proyecto, se hará uso del Base-Hat SIM7600G-H 4G para Jetson Nano.



Figura 5: Módulo Base-Hat SIM7600G-H

En primer lugar, utilizando la terminal del sistema operativo Ubuntu, se ingresan los siguientes comandos:

```
sudo apt-get update
sudo apt-get install python3-pip
sudo pip3 install pyserial
mkdir -p ~/Documents/SIM7600X_4G_for_JETSON_NANO
wget -P ~/Documents/SIM7600X_4G_for_JETSON_NANO/ https://www.waveshare.com/w/upload/6/64/SIM7600X_4G_for_JETSON_NANO.tar.gz
cd ~/Documents/SIM7600X_4G_for_JETSON_NANO/
tar -xvf SIM7600X_4G_for_JETSON_NANO.tar.gz
sudo pip3 install Jetson.GPIO
sudo groupadd -f -r gpio
sudo usermod -a -G gpio your_user_name
sudo udevadm control --reload-rules && sudo udevadm trigger
sudo apt-get install minicom
```

Figura 6: Instalación de librerías

Los comandos ingresados en la figura 6 se encargan de instalar todas las librerías y el software necesario para poder comenzar a utilizar la red LTE mediante el módulo SIM7600G-H. Además también se crea un directorio que contendrá la configuración de usuario así como un cuenta enlazada al módulo.

Posteriormente, probamos que el puerto GPIO de nuestra Jetson Nano esté funcionando con los siguientes comandos:

```
echo 200 > /sys/class/gpio/export
echo out > /sys/class/gpio200/direction
echo 1 > /sys/class/gpio200/value
echo 0 > /sys/class/gpio200/value
```

Figura 7: Instalación de librerías

Después de haber realizado los pasos anteriores, el pin con el nombre NET deberá parpadear constantemente, lo que significa que el módulo está listo para ser utilizado.

Para realizar la comunicación LTE, primero se ingresa a la librería minicom utilizando los siguientes comandos:

```
sudo su
killall ModemManager
minicom -D /dev/ttyUSB2
```

Figura 8: Minicom

Posteriormente, se necesita actualizar los drivers:

```
-----
cd
wget https://www.waveshare.com/w/upload/4/46/Simcom_wwan.zip
unzip Simcom_wwan.zip
cd simcom_wwan
sudo su
make
```

Figura 9: Actualización de drivers

Finalmente se establece una dirección IP con el siguiente comando:

- Allocate IP

```
apt-get install udhcpc  
udhcpc -i wwan0
```

Figura 10: Establecer dirección IP

2.3. Resultados

Se realizó la investigación para conocer los pasos necesarios para establecer comunicación LTE utilizando el módulo SIM7600G-H para la NVIDIA Jetson Nano.

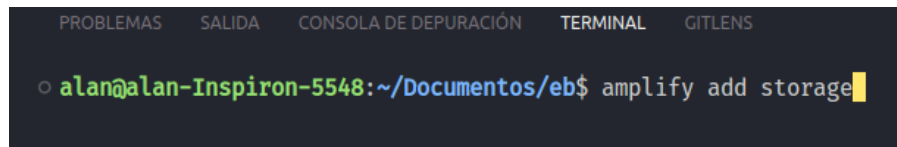
3. Enlace de Amazon S3 con el sistema backend

3.1. Objetivo

Configurar e implementar la comunicación entre el sistema de alojamiento Amazon S3 y el sistema backend

3.2. Descripción

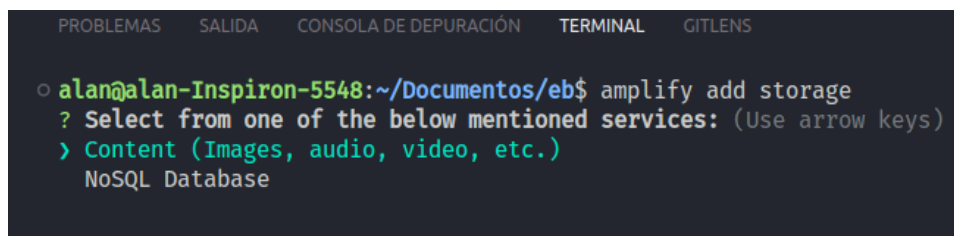
Utilizando un editor de código, y desde el directorio raíz de la aplicación, se deberá introducir el siguiente comando:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
```

Figura 11: Configuración de servicio de almacenamiento S3

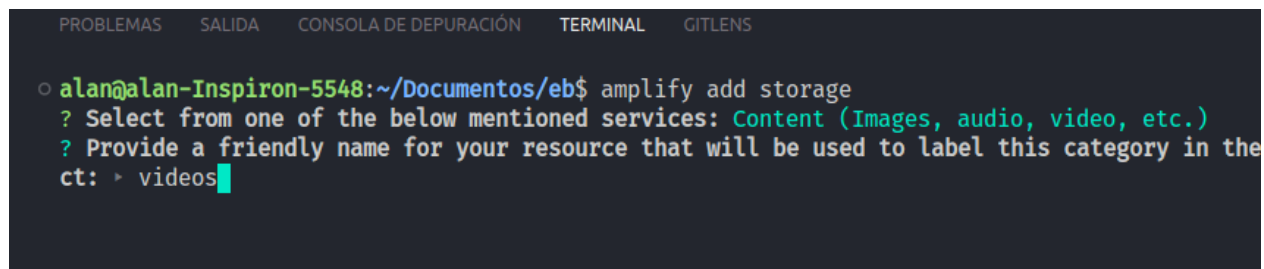
Posteriormente, se requiere especificar que tipo de servicio de almacenamiento se integrará a la aplicación (multimedia o base de datos NoSQL). Para el presente proyecto, se utilizará el almacenamiento de contenido multimedia, por lo tanto, se seleccionará dicha opción.



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: (Use arrow keys)
> Content (Images, audio, video, etc.)
  NoSQL Database
```

Figura 12: Generación de Endpoint de GraphQL

Ingresamos el nombre de nuestro espacio de almacenamiento:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
? Provide a friendly name for your resource that will be used to label this category in the ct: > videos
```

Figura 13: Generación de Endpoint de GraphQL

Después, se necesita establecer cuantos usuarios, así como cuales podrán acceder a dicho servicio:

```
o alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
✓ Provide a friendly name for your resource that will be used to label this category in
  ct: · videos

✓ Provide bucket name: · videos
? Who should have access: ... (Use arrow keys or type to filter)
> Auth users only
  Auth and guest users
```

Figura 14: Generación de Endpoint de GraphQL

```
o alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
✓ Provide a friendly name for your resource that will be used to label this category in
  ct: · videos

✓ Provide bucket name: · videos
✓ Who should have access: · Auth and guest users
? What kind of access do you want for Authenticated users? ... (Use arrow keys or type to
  ● create/update
  ● read
  >● delete
  (Use <space> to select, <ctrl + a> to toggle all)
```

Figura 15: Generación de Endpoint de GraphQL

```
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
✓ Who should have access: · Auth and guest users
✓ What kind of access do you want for Authenticated users? · create/update, read, delete
? What kind of access do you want for Guest users? ... (Use arrow keys or type to filter)
>● create/update
  ● read
  o delete
  (Use <space> to select, <ctrl + a> to toggle all)
```

Figura 16: Generación de Endpoint de GraphQL

```

Edit your schema at /home/alan/Documentos/eb/amplify/backend/api/ebase/schema.graphql or place
graphql files in a directory at /home/alan/Documentos/eb/amplify/backend/api/ebase/schema
✓ Successfully pulled backend environment dev from the cloud.

Current Environment: dev

```

Category	Resource name	Operation	Provider plugin
Storage	videos	Create	awscloudformation
Auth	ebase35f92a6b	Update	awscloudformation
Function	S3Trigger6956e03e	No Change	awscloudformation
Api	ebase	No Change	awscloudformation

```

? Are you sure you want to continue? (Y/n) >

```

Figura 17: Generación de Endpoint de GraphQL

3.3. Resultados

Al ingresar a la consola de servicios de AWS, en la sección de buckets de S3, se puede observar que se encuentra el bucket recién creado llamado *videos175126-dev*.

Buckets (3) Info				
Los buckets son contenedores de datos almacenados en S3. Más información				
	Copiar ARN	Vaciar	Eliminar	Crear bucket
<input type="text" value="Buscar buckets por nombre"/>				
Nombre		Región de AWS	Acceso	Fecha de creac
<input type="radio"/>	amplify-ebase-dev-175126-deployment	EE. UU. Este (Norte de Virginia) us-east-1	Los objetos pueden ser públicos	28 Feb 2023 5
<input type="radio"/>	ebase6c6dcb3b214e4c3fa82df5d47dce7c22175126-dev	EE. UU. Este (Norte de Virginia) us-east-1	Los objetos pueden ser públicos	25 Mar 2023 1
<input type="radio"/>	videos175126-dev	EE. UU. Este (Norte de Virginia) us-east-1	Los objetos pueden ser públicos	17 Apr 2023 1

Figura 18: Tablas generadas mediante los schemas definidos

4. Creación de los servicios backend

4.1. Objetivo

Desarrollar los servicios que constituyen las funciones de la API, tales como obtención, creación y eliminación. Además, crear de los servicios que llevarán a cabo las funciones internas en la plataforma.

4.2. Descripción

GraphQL trabaja con 3 tipos de archivos:

- *Queries*: Este archivo contiene las funciones que permitirán acceder a los datos.
- *Mutations*: En este archivo se encuentran todas las funciones que permitirán realizar el manejo de datos (actualizar, eliminar, agregar)
- *Subscriptions*: Las *subscriptions* en GraphQL son funciones de consulta especiales, que se envían a través de un punto de conexión websocket. Permiten realizar cierta operación cada vez que se ejecuta una acción en el backend.

Para comenzar con el archivo de *Queries* se tienen las siguientes funciones:

```
export const getConductor = /* GraphQL */ `
  query GetConductor($id: ID!) {
    getConductor(id: $id) {
      id
      nombre
      apellido
      incidencias {
        items {
          id
          estado
          url_video
          ubicacion
          fecha_hora
          createdAt
          updatedAt
          conductorIncidenciasId
        }
      }
      nextToken
    }
    num_incidencias
    createdAt
    updatedAt
  }
`;
```

Figura 19: Función getConductor

La función de la figura ??, obtiene los datos de un solo Conductor.

```
export const listConductors = /* GraphQL */ `
  query ListConductors(
    $filter: ModelConductorFilterInput
    $limit: Int
    $nextToken: String
  ) {
    listConductors(filter: $filter, limit: $limit, nextToken: $nextToken) {
      items {
        id
        nombre
        apellido
        incidencias {
          nextToken
        }
        num_incidencias
        createdAt
        updatedAt
      }
      nextToken
    }
  }
`;
```

Figura 20: Función listConductors

La función de la figura 20 obtiene todos los datos de todos los conductores almacenados en la base de datos.

```
export const getIncidencia = /* GraphQL */ `
  query GetIncidencia($id: ID!) {
    getIncidencia(id: $id) {
      id
      conductor {
        id
        nombre
        apellido
        incidencias {
          nextToken
        }
        num_incidencias
        createdAt
        updatedAt
      }
      estado
      url_video
      ubicacion
      fecha_hora
      createdAt
      updatedAt
      conductorIncidenciasId
    }
  }
`;
```

Figura 21: Función getIncidencia

La función de la figura 21 obtiene los datos de una sola Incidencia.


```
port const listIncidencias = /* GraphQL */ `
query ListIncidencias(
  $filter: ModelIncidenciaFilterInput
  $limit: Int
  $nextToken: String
) {
  listIncidencias(filter: $filter, limit: $limit, nextToken: $nextToken) {
    items {
      id
      conductor {
        id
        nombre
        apellido
        num_incidencias
        createdAt
        updatedAt
      }
      estado
      url_video
      ubicacion
      fecha_hora
      createdAt
      updatedAt
      conductorIncidenciasId
    }
  }
  nextToken
}
```

Figura 22: Función listIncidencias

La función de la figura 22 obtiene los datos de todas las incidencias de almacenadas en la base de datos.

En cuanto al archivo de *Mutations* se tienen las siguientes funciones:

```
export const createConductor = /* GraphQL */ `
mutation CreateConductor(
  $input: CreateConductorInput!
  $condition: ModelConductorConditionInput
) {
  createConductor(input: $input, condition: $condition) {
    id
    nombre
    apellido
    incidencias {
      items {
        id
        estado
        url_video
        ubicacion
        fecha_hora
        createdAt
        updatedAt
        conductorIncidenciasId
      }
    }
    nextToken
  }
  num_incidencias
  createdAt
  updatedAt
}
```

Figura 23: Función createConductor

La función de la figura 23 se encarga de crear el registro de un conductor en la base de datos.

```
export const updateConductor = /* GraphQL */ `
mutation UpdateConductor(
  $input: UpdateConductorInput!
  $condition: ModelConductorConditionInput
) {
  updateConductor(input: $input, condition: $condition) {
    id
    nombre
    apellido
    incidencias {
      items {
        id
        estado
        url_video
        ubicacion
        fecha_hora
        createdAt
        updatedAt
        conductorIncidenciasId
      }
    }
    nextToken
  }
  num_incidencias
  createdAt
  updatedAt
}
```

Figura 24: Función updateConductor

La función de la figura 24 se encarga de modificar datos del registro de un conductor.

```
export const deleteConductor = /* GraphQL */ `
mutation DeleteConductor(
  $input: DeleteConductorInput!
  $condition: ModelConductorConditionInput
) {
  deleteConductor(input: $input, condition: $condition) {
    id
    nombre
    apellido
    incidencias {
      items {
        id
        estado
        url_video
        ubicacion
        fecha_hora
        createdAt
        updatedAt
        conductorIncidenciasId
      }
    }
    nextToken
  }
  num_incidencias
  createdAt
  updatedAt
}
```

Figura 25: Función deleteConductor

La función de la figura 25 se encarga de eliminar un conductor de la base de datos.

```
;
export const createIncidencia = /* GraphQL */ `
mutation CreateIncidencia(
  $input: CreateIncidenciaInput!
  $condition: ModelIncidenciaConditionInput
) {
  createIncidencia(input: $input, condition: $condition) {
    id
    conductor {
      id
      nombre
      apellido
      incidencias {
        nextToken
      }
      num_incidencias
      createdAt
      updatedAt
    }
    estado
    url_video
    ubicacion
    fecha_hora
    createdAt
    updatedAt
    conductorIncidenciasId
  }
}
`;
```

Figura 26: Función createIncidencia

La función de la figura 26 se encarga de crear una Incidencia en la base de datos. Para el archivo de *subscriptions* se tienen la siguiente función:

```
export const onCreateIncidencia = /* GraphQL */ `
subscription OnCreateIncidencia(
  $filter: ModelSubscriptionIncidenciaFilterInput
) {
  onCreateIncidencia(filter: $filter) {
    id
    conductor {
      id
      nombre
      apellido
      incidencias {
        nextToken
      }
      num_incidencias
      createdAt
      updatedAt
    }
    estado
    url_video
    ubicacion
    fecha_hora
    createdAt
    updatedAt
    conductorIncidenciasId
  }
}
`;
```

Figura 27: Función onCreateIncidencia

La función de la figura 27 se encarga de realizar una consulta cada vez que una Incidencia nueva es dada de alta en la base de datos.

Al estar trabajando con GraphQL dentro del proyecto, la manera en que se podrán realizar las operaciones *CRUD* - (*Create, Read, Update, Delete*), será mediante funciones JSON.

Para comprobar que la API permite dichas operaciones, se debe ingresar a la consola de AWS y dirigirse a la sección de AWS AppSync.



Figura 28: Consola de AWS

Posteriormente se necesita ingresar a la sección de consultas.

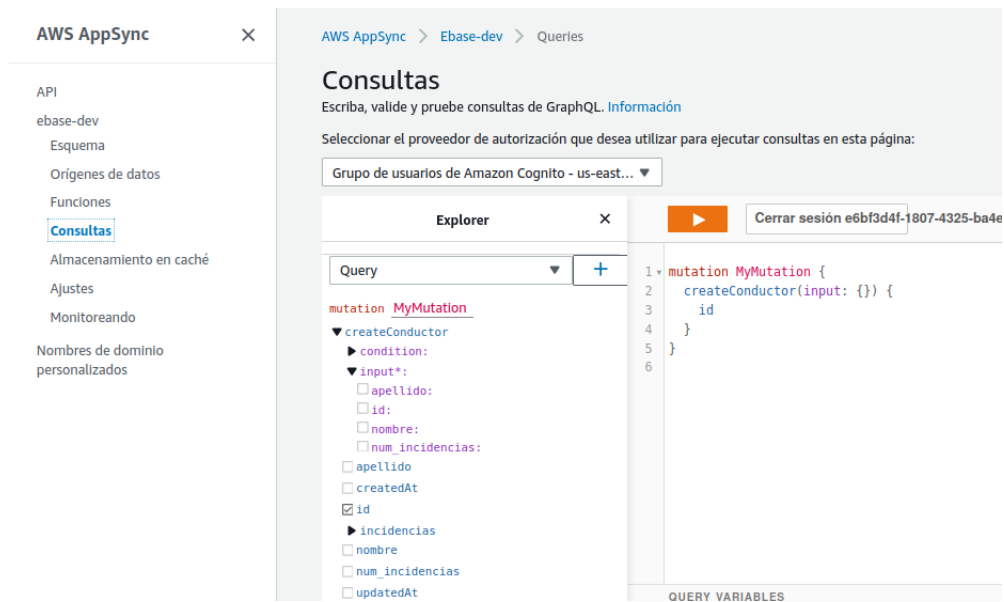


Figura 29: Funcionamiento de Appsync

AWS permite elegir si realizar un *query*, *mutation*, o *subscription* mediante código JSON

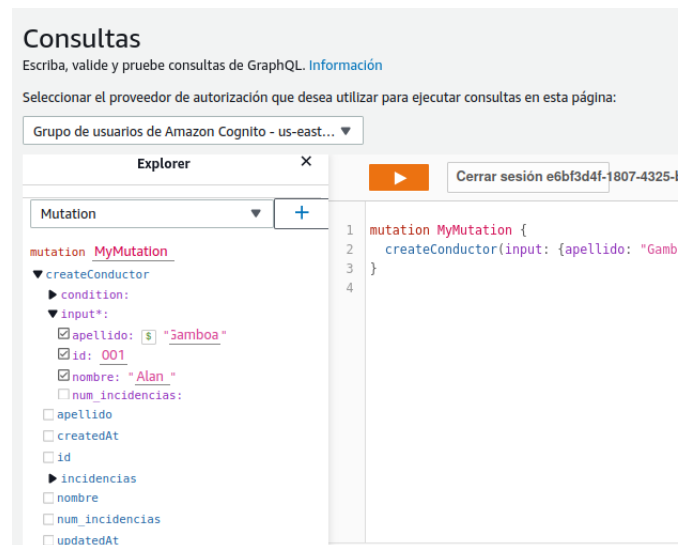


Figura 30: Crear Conductor

En la figura 31 se puede apreciar una sentencia JSON que permite utilizar las funciones previamente creadas para dar de alta un conductor.

4.3. Resultados

Como resultado tenemos un log que nos muestra la entrada creada



Figura 31: Resultado

5. Familiarizarse con el entorno de desarrollo de la NVIDIA Jetson Nano

5.1. Objetivo

Investigar la documentación ofrecida por NVIDIA sobre el uso y entorno de desarrollo de la jetson nano.

5.2. Descripción

Instalación en la tarjeta microSD

El Jetson Nano Developer Kit utiliza una tarjeta microSD como dispositivo de arranque y almacenamiento principal. Por tanto, fue necesario instalar una entorno de desarrollo en la propia placa, para lo cual se requirió una tarjeta microSD de un mínimo recomendado de 32 GB de acuerdo a la documentación de Nvidia [4].

Como primer paso, se descargó el *Jetson Nano Developer Kit SD Card Image* [5], la imagen para la Jetson Nano se refiere a un archivo de imagen del sistema operativo específicamente diseñado y optimizado para ser utilizado en la placa de desarrollo NVIDIA Jetson Nano, la imagen generalmente incluye un sistema operativo Linux, controladores de hardware específicos para la Jetson Nano, y una configuración predefinida que permite aprovechar las capacidades de procesamiento de la placa.

Posteriormente se instaló en la tarjeta microSD desde el sistema operativo Linux, utilizando los siguientes pasos:

1. Se abrió una terminal y se insertó la tarjeta microSD.
2. Se utilizó el siguiente comando para mostrar qué dispositivo de disco se le asignó:

```
dmesg | tail | awk '$3 == "sd" {print}'
```

3. Se escribió la imagen de la tarjeta SD comprimida (previamente descargada) en la tarjeta microSD con el siguiente comando:

```
/usr/bin/unzip -p ~/Downloads/jetson_nano_devkit_sd_card.zip |  
sudo /bin/dd of=/dev/sda bs=1M status=progress
```

4. Finalmente, se expulsó el dispositivo de disco desde la línea de comando utilizando:

```
sudo eject /dev/sda
```

Configuración y primer arranque

Jetson Nano Developer Kit permitió dos formas de interactuar, una a través de otra computadora y otra utilizando una pantalla, teclado y mouse conectados. Además, el kit de desarrollo no contaba

con una fuente de alimentación incluida, por lo que se utilizó una fuente de alimentación Micro-USB (5V-2A).

Para iniciar el kit de desarrollo, se conectó el mouse, la pantalla, el teclado y la fuente de alimentación. Posteriormente, se realizó la configuración inicial del sistema operativo, la cual incluyó los siguientes pasos:

- Se revisó y aceptó el EULA del software NVIDIA Jetson.
- Se seleccionó el idioma del sistema, la distribución del teclado y la zona horaria.
- Se creó un nombre de usuario, contraseña y nombre de la computadora.
- Se seleccionó el tamaño de partición de la aplicación, se utilizó el tamaño máximo sugerido.

Después de haber iniciado la Jetson Nano, se descargaron e instalaron Python3, pip, NumPy, imutils, dlib, TensorFlow, Keras y el IDE PyCharm en la placa. Sin embargo, como la Jetson Nano ya viene con una versión de OpenCV preinstalada por defecto, no fue necesario realizar la instalación de OpenCV.

Los pasos para la instalación fueron los siguientes:

- Instalación de Python3: Se utilizó el siguiente comando en la terminal para instalar Python3 en la Jetson Nano:

```
sudo apt-get update
sudo apt-get install python3
```

- Instalación de pip: Se instaló pip, el gestor de paquetes de Python, utilizando el siguiente comando:

```
sudo apt-get install python3-pip
```

- Instalación de NumPy: Se utilizó pip para instalar NumPy, una biblioteca de Python para cálculos numéricos, con el siguiente comando:

```
pip3 install numpy
```

- Instalación de imutils: Se utilizó pip para instalar imutils, una biblioteca de Python para manipulación de imágenes y video, con el siguiente comando:

```
pip3 install imutils
```

- Instalación de dlib: Se instaló dlib, una biblioteca de procesamiento de imágenes y detección de rostros, utilizando el siguiente comando:

```
pip3 install dlib
```

- Instalación de PyCharm: Se descargó e instaló PyCharm, un entorno de desarrollo integrado (IDE) para Python, desde el sitio web oficial de JetBrains. La instalación se realizó utilizando archivos tar (tarball).
 1. descargo el archivo tar para procesadores ARM64 para sistema operativo linux.
 2. Se descomprimió el archivo pycharm-.tar.gz en una carpeta diferente desde la terminal.

```
tar xzf pycharm-*.tar.gz -C <nueva_carpeta_archivo>
```

3. Para instalar PyCharm, se utilizó el siguiente comando con privilegios de superusuario:

```
sudo tar xzf pycharm-*.tar.gz -C /opt/
```

4. Se accedió al subdirectorio bin:

```
cd /opt/pycharm-2022.2.4/bin
```

5. Se ejecutó el archivo pycharm.sh desde el subdirectorio bin para iniciar PyCharm.

```
sh pycharm.sh
```

5.3. Resultados

Se instaló la imagen para la Jetson Nano en la placa de desarrollo, y se realizó la configuración y el armado de los periféricos de entrada necesarios. Se instaló Python y las librerías requeridas para el proyecto, así como el IDE PyCharm. Además, se dedicó tiempo para familiarizarse con el entorno de desarrollo de NVIDIA, lo cual implicó explorar las herramientas y algunas configuraciones de la Jetson Nano.

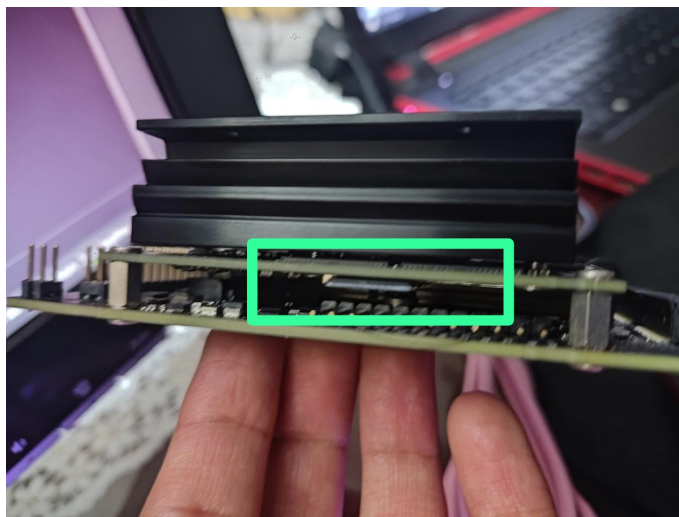


Figura 32: Ranura para tarjeta microSD para almacenamiento

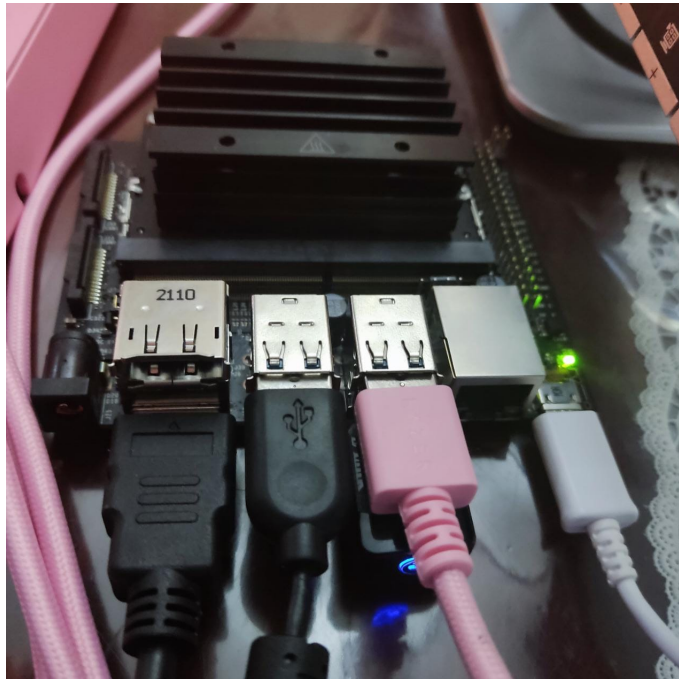


Figura 33: Periféricos de entrada para el Kit de desarrollo Jetson Nano.



Figura 34: instalación y configuracion completada.

```
nano@nano-desktop: ~/PycharmProjects/pythonProject
nano@nano-desktop:~/PycharmProjects/pythonProject$ python3
Python 3.6.9 (default, Mar 10 2023, 16:46:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> import imutils
import>>> import dlib
>>> import time
>>> import cv2
>>> exit()
nano@nano-desktop:~/PycharmProjects/pythonProject$
```

Figura 35: Instalación de Python3 y librerías.

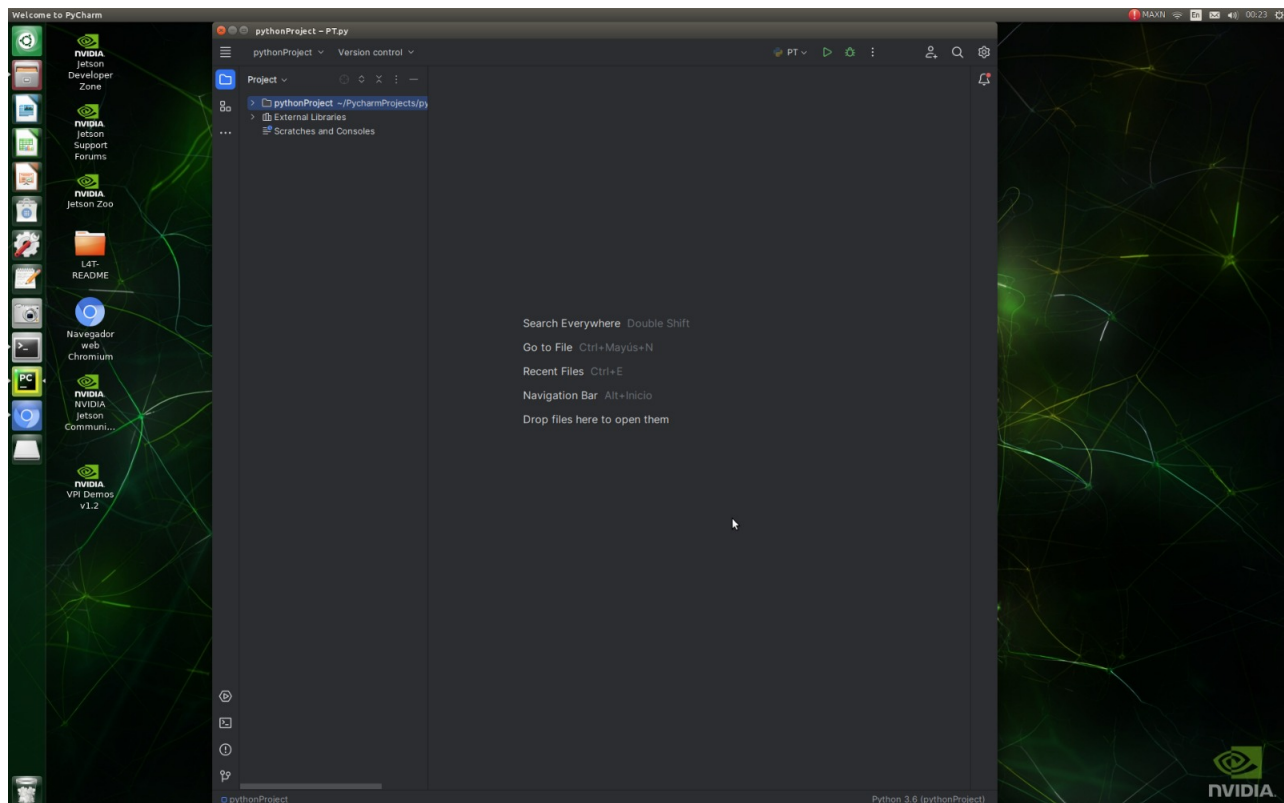


Figura 36: Instalación del IDE PyCharm.

6. Implementar algoritmo para la detección del rostro y ojos

6.1. Objetivo

realizar la detección de rostro y ojos en la Jetson Nano .

6.2. Descripción

Para el siguiente desarrollo se utilizó Google Colab desde el kit de desarrollo Jetson Nano, las pruebas y la implementación se realizaron con una imagen como entrada.

Desde Google Colab se importaron las bibliotecas TensorFlow, Keras, Matplotlib, OpenCV y NumPy en Python. Luego, se cargó el modelo de aprendizaje profundo previamente entrenado llamado MobileNetV2 utilizando la función load_model de Keras, y se asignó el modelo a la variable model. Después, se cargó una imagen en formato JPG desde la ruta de archivo content/drive/MyDrive/DATASET/pruebas/d utilizando la función imread de OpenCV, y asignó la imagen a la variable img para su posterior procesamiento. Finalmente, se utilizó la función imshow de Matplotlib para mostrar la imagen cargada en una ventana de visualización, después de realizar una conversión de espacio de color de BGR a RGB utilizando la función cvtColor de OpenCV para asegurar que la imagen se mostrara correctamente en colores en la salida.

```
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import cv2
import numpy as np

#cargar el modelo
model = tf.keras.models.load_model('/content/drive/MyDrive/DATASET/MobileNetV2.h5')

#cargar imagen y mostrar
img = cv2.imread('/content/drive/MyDrive/DATASET/pruebas/d.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

<matplotlib.image.AxesImage at 0x7fd110064b80>



Figura 37: Código de la detección de rostro y ojos de una imagen, implementando el modelo entrenado

Se cargaron dos clasificadores en cascada (Haar cascades) para la detección de rostros y ojos, utilizando las rutas de archivo predefinidas en la biblioteca OpenCV. Los clasificadores fueron asignados a las variables `face_cascade` y `eye_cascade`, respectivamente. La imagen fue convertida a escala de grises utilizando la función `cvtColor`. Posteriormente se detectaron los ojos en la imagen en escala de grises utilizando la función `detectMultiScale` de OpenCV con parámetros específicos de escala, y las coordenadas y dimensiones de los ojos detectados se guardaron en la variable `eyes`. Se dibujaron cuadros alrededor de los ojos detectados en la imagen original utilizando la función `rectangle`. Finalmente mostró la imagen con los cuadros delimitadores alrededor de los ojos utilizando la función `imshow` de Matplotlib.

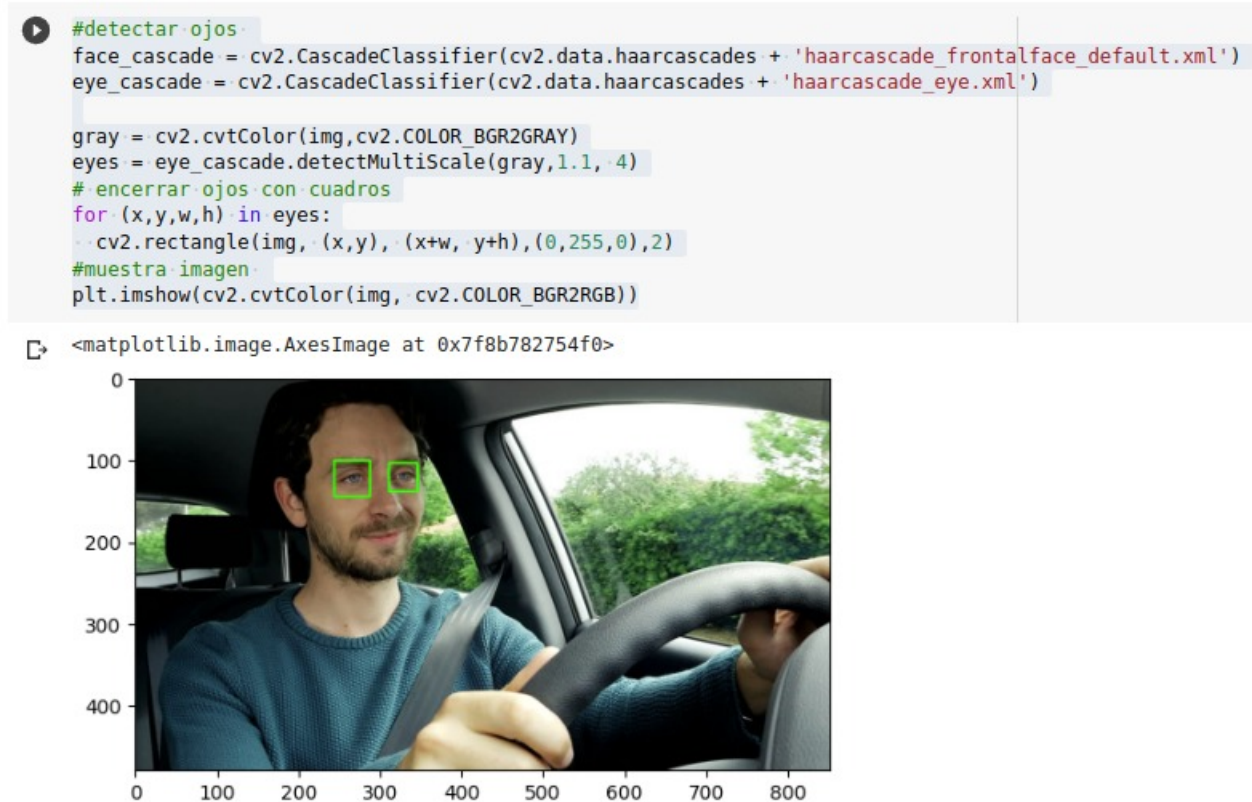


Figura 38: Código de la detección de rostro y ojos de una imagen implementando el modelo entrenado

Se realizó una detección adicional de los ojos dentro de las regiones de interés (ROI) definidas por los cuadros delimitadores, y se guardaron los datos de las regiones de interés en las variables `roi_gray` y `roi_color` para su posterior procesamiento.

Se verificó si se detectaron ojos en las regiones de interés, y se mostró la imagen recortada de los ojos utilizando la función `imshow` de Matplotlib, después de realizar una conversión de espacio de color de BGR a RGB utilizando la función `cvtColor` de OpenCV para asegurar que la imagen se muestre correctamente en colores en la salida.


```

✓ [4] #
0 s eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
#detectar ojos
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
eyes = eye_cascade.detectMultiScale(gray,1.1, 4)
#guardar datos de la delimitacion de los ojos
for x,y,w,h in eyes:
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyess = eye_cascade.detectMultiScale(roi_gray)

#guardar datos y verificar deteccion de ojos
if len(eyess)== 0:
    print('No se detectaron ojos')
else:
    for (ex,ey,ew,eh) in eyess:
        eyes_roi = roi_color[ey:ey+eh, ex:ex+ew]

plt.imshow(cv2.cvtColor(eyes_roi, cv2.COLOR_BGR2RGB))

No se detectaron ojos
<matplotlib.image.AxesImage at 0x7fe6c805e2b0>

```



Figura 39: Código de la detección de rostro y ojos de una imagen implementando el modelo entrenado

La imagen de los ojos previamente recortada como región de interés y almacenada en la variable `eyes_roi` fue redimensionada a un tamaño de 224 x 224. El resultado se guardó en la variable `final_image`. Posteriormente se agregó una dimensión adicional a la imagen redimensionada en el eje 0 utilizando la función `np.expand_dims()`. Esto se hizo para adecuar el formato de entrada requerido por muchos modelos de aprendizaje profundo que esperan un lote (batch) de imágenes como entrada. El resultado se guardó nuevamente en la variable `final_image`.

La imagen se normalizó dividiéndola por 255.0 para asegurar que los valores de los píxeles estén en el rango de 0 a 1. La imagen normalizada se guardó nuevamente en la variable `final_image`.

Finalmente, se utilizó la función `model.predict()` para realizar una predicción utilizando un modelo de aprendizaje profundo previamente entrenado. La imagen normalizada `final_image` se utilizó como entrada para el modelo, y se obtuvo una salida de predicción basada en los datos de la imagen

procesada.

El modelo previamente entrenado utiliza dos categorías. Si la predicción muestra un valor entre 0.5 y 1, se cataloga como un ojo abierto, mientras que un valor entre 0 y 0.49 se cataloga como un ojo cerrado, ya que la red entrenada utiliza en su última capa la función sigmoide para clasificación binaria, devolviendo valores entre 0 y 1.

1 = ojo abierto. 0 = ojo cerrado.

```
✓ [5] #Redimensiona y normaliza
0 s
final_image = cv2.resize(eyes_roi, (224,224))
final_image = np.expand_dims(final_image,axis=0)
final_image = final_image/255.0

final_image.shape

(1, 224, 224, 3)

✓ #prediccion
8 s
model.predict(final_image)

1/1 [=====] - 8s 8s/step
array([[0.9999297]], dtype=float32)
```

Figura 40: Código de la detección de rostro y ojos de una imagen implementando el modelo entrenado

Como se pudo observar en la figura anterior, el valor de la predicción fue de 0.999, lo cual se cataloga como un ojo abierto.

6.3. Resultados

Se cargó el modelo previamente entrenado y una imagen de un conductor en la cual se implementó un clasificador en cascada para detectar el rostro y otro para detectar los ojos en la imagen. Los ojos detectados se encerraron con cuadros verdes y se guardó la región de interés (ROI) de los ojos detectados, donde finalmente se realizó una predicción utilizando el modelo previamente entrenado.

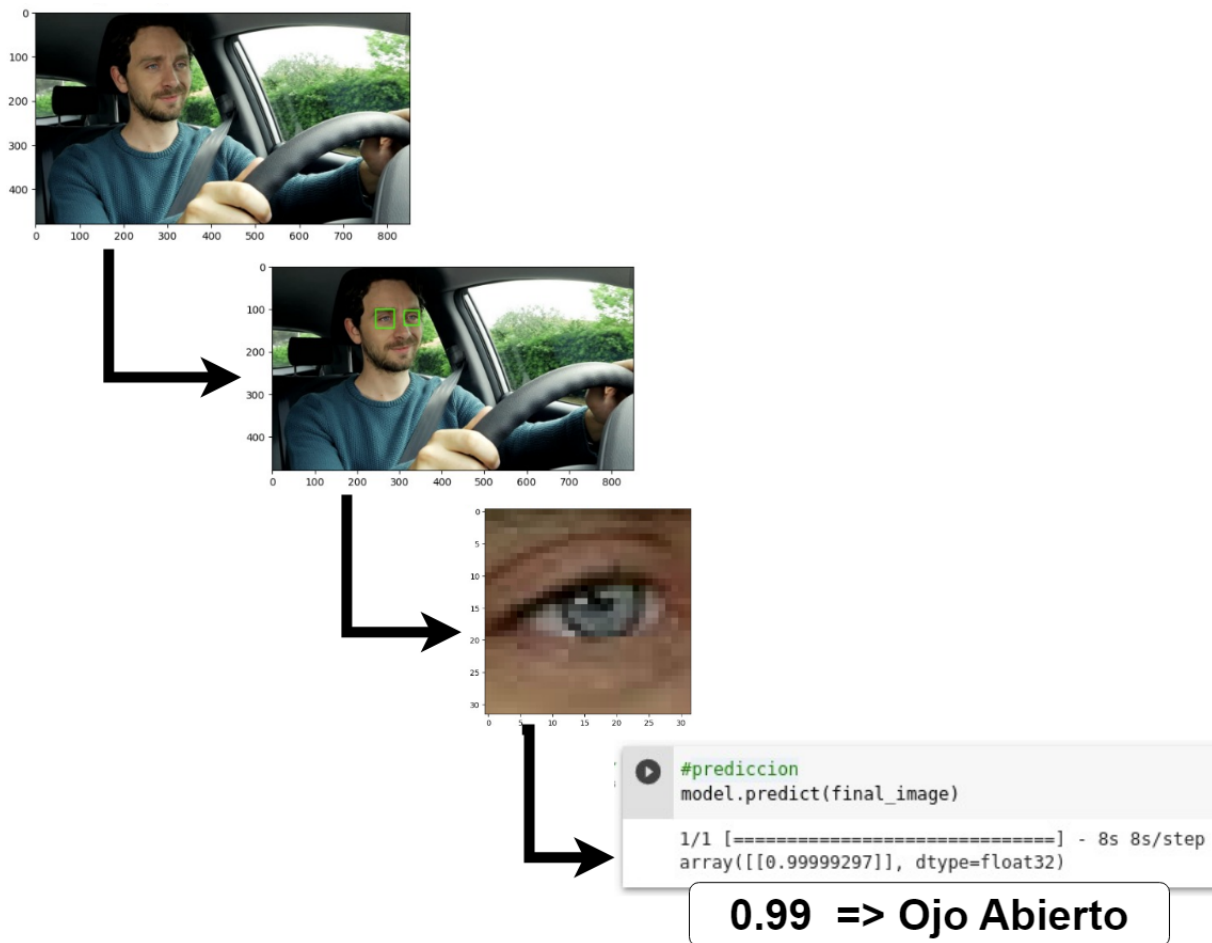


Figura 41: Implementación para la detección de rostro y ojos aplicando el modelo entrenado

7. Implementar puntos faciales en el rostro y la metrica MOR

7.1. Objetivo

Implementar la detección con puntos faciales en el rostro y la metrica mor para detectar si la boca se encuentra abierta o cerrada.

7.2. Descripción

Se realizó un programa en PyCharm en la Jetson Nano para implementar puntos faciales en el rostro. Para ello, se requirió un detector de rostro que se obtuvo del modelo dlib. `get_frontal_face_detector()`. Posteriormente, se descargó el archivo `hape_predictor_68_face_landmarks.dat` para los puntos de referencia.

El proceso de implemetaión fue el siguiente:

- Importación de bibliotecas: Se importan las siguientes bibliotecas necesarias para el programa: `numpy`: Para operaciones numéricas en Python. `cv2`: Para procesamiento de imágenes y video utilizando OpenCV. `dlib`: Para la detección de rostros y puntos faciales. `time`: Para medir el tiempo y calcular el FPS (cuadros por segundo) del video. `distance` de `scipy.spatial`: Para calcular la distancia euclidiana entre dos puntos en un plano. `face_utils` de `imutils`: Para funciones auxiliares relacionadas con el procesamiento de rostros.

```
import numpy as np
import cv2
import dlib
import time
from scipy.spatial import distance as dist
from imutils import face_utils
```

- Se creó la función `calcula` la distancia entre los puntos faciales asignados a la boca, tomando como entrada el arreglo de puntos faciales detectados por `dlib`. Los puntos faciales se dividen en los puntos correspondientes al labio superior e inferior, se calcula el promedio de coordenadas de cada conjunto de puntos, y finalmente se calcula la distancia euclidiana entre los dos promedios. Esta distancia se utiliza como medida de la apertura de la boca.

`top_lip = shape[50:53]`: Extrae las coordenadas de los puntos 50 a 52 del conjunto de coordenadas `shape`, que representan los puntos en el labio superior izquierdo de la cara.

`top_lip = np.concatenate((top_lip, shape[61 : 64]))`: Concatena las coordenadas de los puntos 50 a 52 con las coordenadas de los puntos 61 a 63 del conjunto de coordenadas `shape`, que representan los puntos en el labio superior derecho de la cara.

`low_lip = shape[56:59]`: Extrae las coordenadas de los puntos 56 a 58 del conjunto de coordenadas `shape`, que representan los puntos en el labio inferior izquierdo de la cara.

`low_lip = np.concatenate((low_lip, shape[65:68]))`: Concatena las coordenadas de los puntos 56 a 58 con las coordenadas de los puntos 65 a 67 del conjunto de coordenadas `shape`, que representan los puntos en el labio inferior derecho de la cara.

`top_mean = np. mean(top_lip, axis=0)`: Calcula el promedio de las coordenadas en el labio superior mientras que `low_mean = np. mean(low_lip, axis=0)`: Calcula el promedio de las coordenadas en el labio inferior

`distance = dist. euclidean(top_mean,low_mean)`: Calcula la distancia euclidiana entre los dos puntos obtenidos anteriormente, que representan el centro del labio superior e inferior. Esta distancia es la medida de la apertura de la boca o la separación entre los labios.

```
def cal_yawn(shape):
    top_lip = shape[50:53]
    top_lip = np.concatenate((top_lip, shape[61:64]))

    low_lip = shape[56:59]
    low_lip = np.concatenate((low_lip, shape[65:68]))

    top_mean = np.mean(top_lip, axis=0)
    low_mean = np.mean(low_lip, axis=0)

    distance = dist.euclidean(top_mean,low_mean)
    return distance
```

- Captura de video: Se inicia la captura de video desde la cámara web utilizando la función `cv2.VideoCapture(0)`, donde 0 representa el índice de la cámara web predeterminada en el sistema.

```
cam = cv2.VideoCapture(0)
```

- Carga de modelos: Se cargan dos modelos necesarios para el programa:

`face_model`: Utiliza la función `dlib.get_frontal_face_detector()` para obtener un modelo de detección de rostros frontal. `landmark_model`: Se carga el modelo pre-entrenado `shape_predictor_68_face_landmarks.dat` utilizando la función `dlib.shape_predictor()` para obtener un modelo de predicción de 68 puntos faciales.

```
face_model = dlib.get_frontal_face_detector()
landmark_model = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
```

5. Bucle principal del programa: Se inicia un bucle infinito para capturar y procesar los cuadros del video en tiempo real.

```
#Variables
yawn_thresh = 35
ptime = 0
while True :
```

```
suc,frame = cam.read()
```

```
if not suc :  
break
```

- Medición del FPS: Se calcula el tiempo transcurrido entre cuadros consecutivos para estimar el FPS del video.

ctime = time.time(): Obtiene el tiempo actual en segundos utilizando la

fps = int(1/(ctime-ptime)): Calcula la tasa de cuadros por segundo (FPS)

ptime = ctime: Actualiza el tiempo del cuadro anterior (ptime) con el tiempo actual (ctime) para su uso en el próximo cuadro.

cv2.putText(): Agrega un texto en la ventana de video

```
ctime = time.time()
```

```
fps= int(1/(ctime-ptime))
```

```
ptime = ctime
```

```
cv2.putText(frame,f'FPS:{fps}',(frame.shape[1]-120,frame.shape[0]-20),cv2.FONT_HERSHEY
```

- Detección del rostro: Se convierte cada cuadro a escala de grises utilizando la función cv2.cvtColor() y se utiliza el modelo face_model para detectar los rostros en la imagen. Luego, para cada rostro detectado en la imagen, se detectan los landmarks o puntos faciales utilizando el modelo de landmark_model previamente definido, finalmente los puntos faciales detectados se convierten en un arreglo NumPy.

```
#Deteccion del rostro
```

```
img_gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
```

```
faces = face_model(img_gray)
```

```
for face in faces:
```

```
#Detectar Landmarks
```

```
shapes = landmark_model(img_gray,face)
```

```
shape = face_utils.shape_to_np(shapes)
```

- Marcado del labio: Se dibujan contornos alrededor del labio superior e inferior utilizando la función cv2.drawContours() en cada cuadro.

Se utiliza la variable shape que contiene los puntos de referencia (landmarks), se extraen los puntos de referencia del labio inferior y superior y Se utiliza la función cv2.drawContours() de OpenCV para dibujar los contornos del labio inferior y superior en la imagen o marco (frame).

```
lip = shape[48:60]
```

```
cv2.drawContours(frame,[lip],-1,(0, 165, 255),thickness=3)
```

- Cálculo de la distancia del labio: Se llama a la función `cal_yawn()` para calcular la distancia del labio, que representa la apertura de la boca y se imprime.

```
#Calcular la distancia del labio
lip_dist = cal_yawn(shape)
# print(lip_dist)
```

- Detección de límite de apertura de la boca: Se compara la distancia del labio calculada con un umbral predefinido (`yawn_thresh`) para determinar si se ha superado el límite de apertura de la boca. Si es así, se muestra.

```
if lip_dist > yawn_thresh :
    cv2.putText(frame, f'Límite de apertura superado', (frame.shape[1]//2 - 170 , frame.shap
```

- `cv2.imshow('Webcam', frame)`: Muestra el marco de video capturado en una ventana hasta pulsar la tecla `q` para finalizar. `cam.release()`: Libera los recursos del objeto de la cámara y `cv2.destroyAllWindows()` cierra todas las ventanas de visualización creadas por OpenCV.

```
#mostramos los cuadros
cv2.imshow('Webcam' , frame)
if cv2.waitKey(1) & 0xFF == ord('q') :
    break
cam.release()
cv2.destroyAllWindows()
```

7.3. Resultados

Se detectó el rostro y se aplicaron los landmark points para medir la apertura de la boca en tiempo real utilizando una cámara web. Además, se mostró un mensaje si se superaba un umbral predefinido y se imprimió el valor de cada medición.

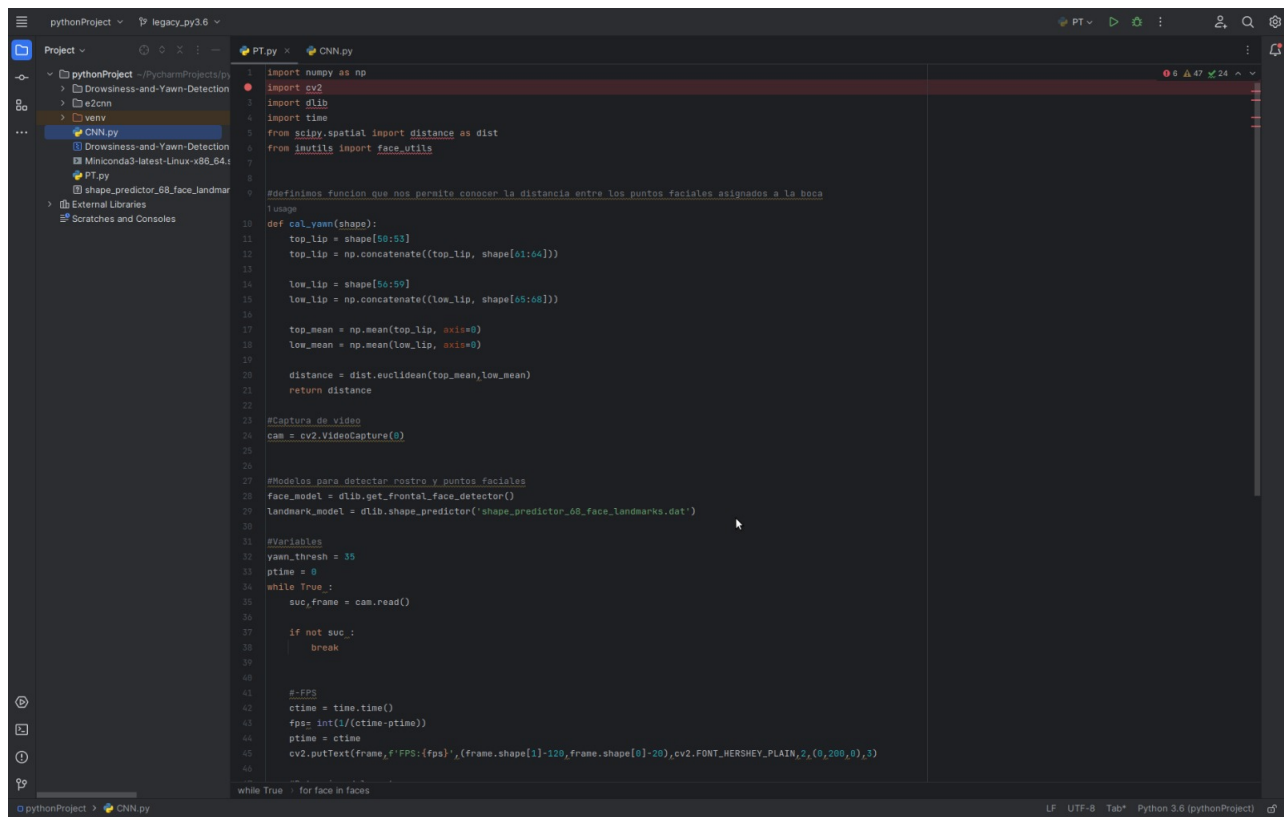


Figura 42: Captura de pantalla del código para la detección de apertura de la boca

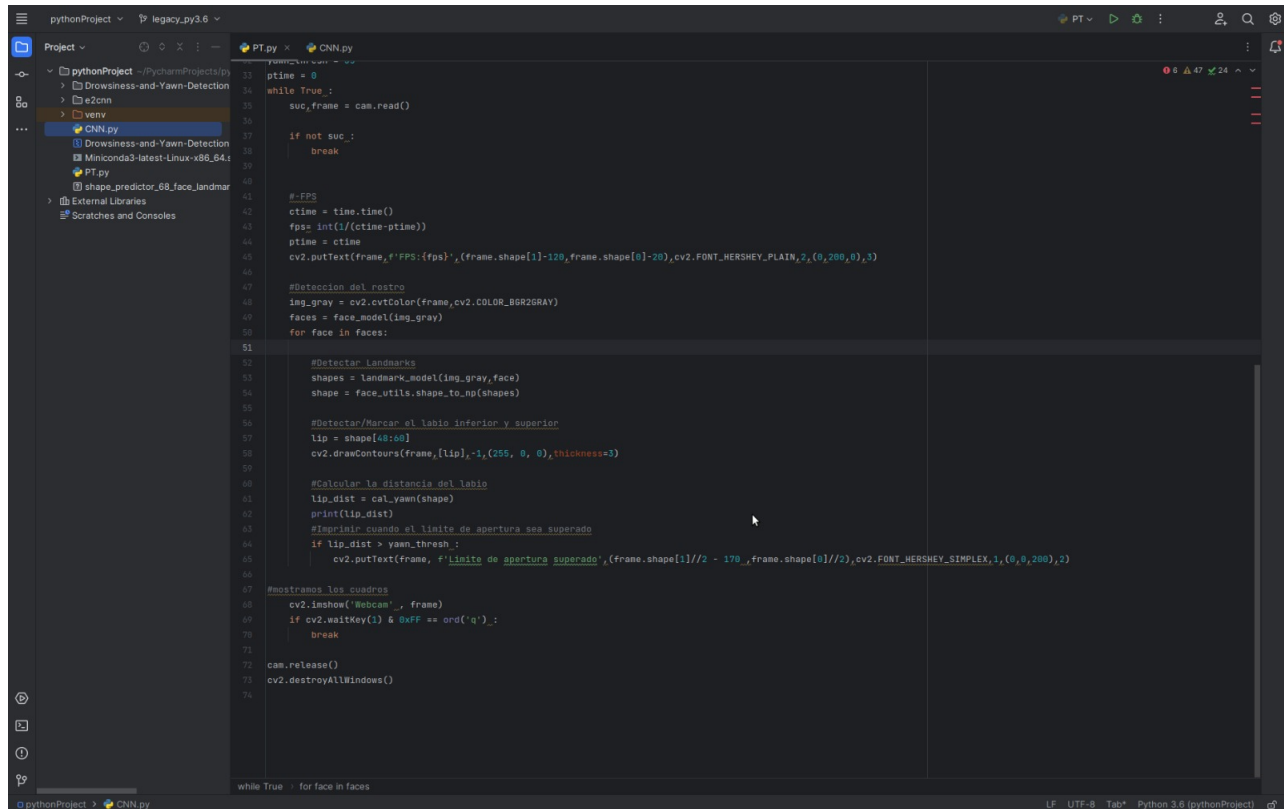


Figura 43: Captura de pantalla del código para la detección de apertura de la boca



Figura 44: Prueba de la detección de apertura de boca sin mostrar valores de apertura

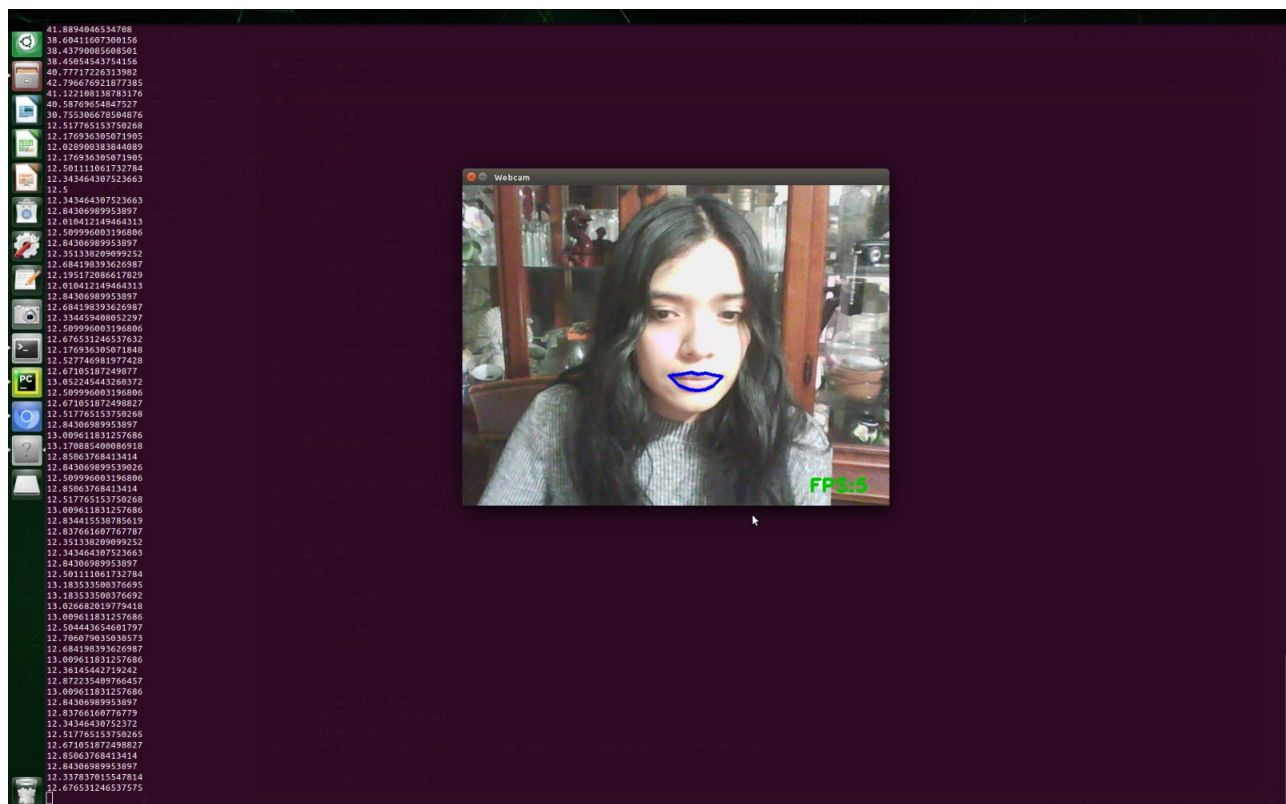


Figura 45: Prueba de la detección de apertura de boca con valores de apertura, boca cerrada

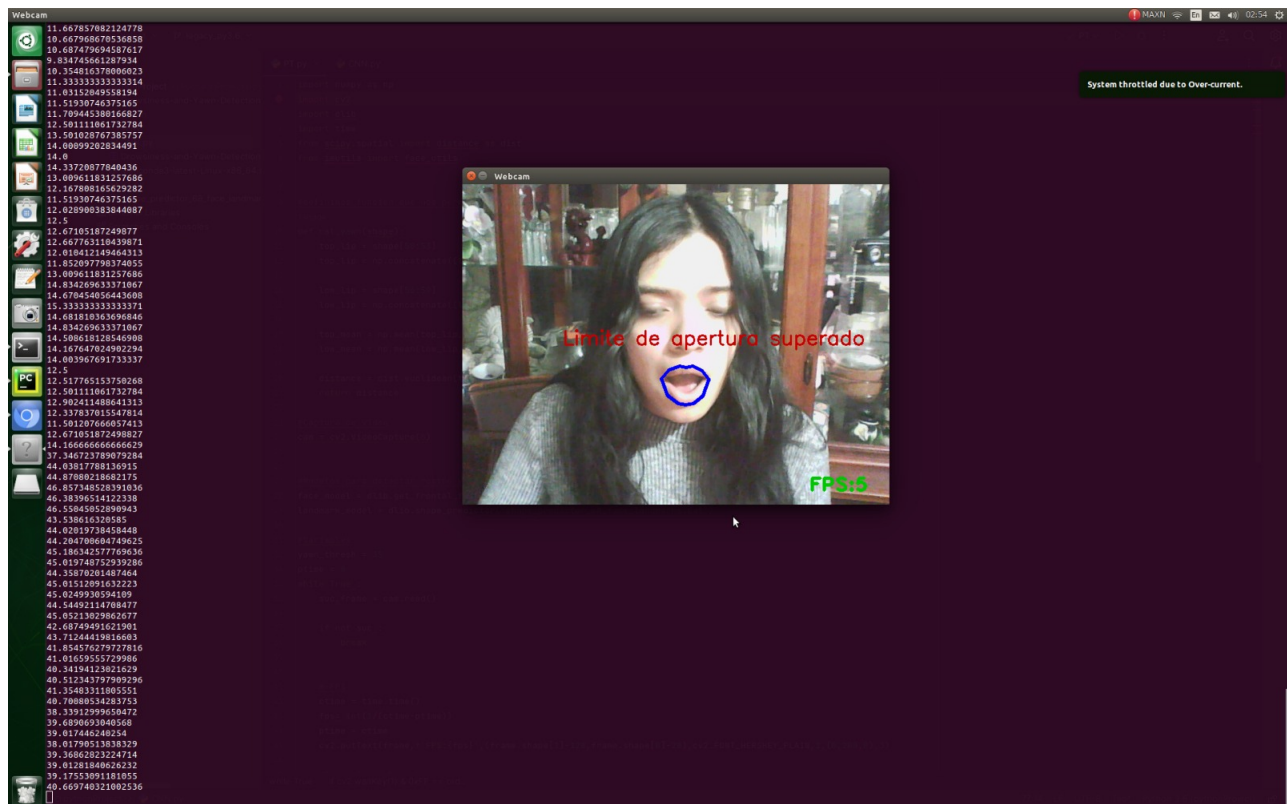


Figura 46: Prueba de la detección de apertura de boca con valores de apertura, boca abierta

8. Conclusiones

Se describió el uso de la librería de diseño ReactJs para el desarrollo del front-end de un proyecto, incluyendo la utilización de diferentes componentes y capturas de pantalla de las vistas creadas en la aplicación web. Se realizó una investigación de la documentación del módulo 3G/4G LTE-Base Hat SIM7600G-H para Jetson Nano, incluyendo los pasos de instalación de librerías y software, pruebas del puerto GPIO, acceso a la librería minicom y actualización de drivers, así como la configuración de la dirección IP en el módulo SIM7600G-H. Se realizó la configuración e implementación de la comunicación entre el sistema de almacenamiento Amazon S3 y el sistema backend de una aplicación, con capturas de pantalla de los comandos y pasos necesarios, y se describe el desarrollo de servicios backend utilizando GraphQL. Se detalla la implementación de la detección de rostro y la aplicación de landmark points para medir la apertura de la boca en tiempo real utilizando una cámara web, con la muestra de mensajes y valores de medición. Se describe la implementación de un clasificador en cascada para detectar rostros y ojos en una imagen de un conductor, con el almacenamiento de regiones de interés (ROI) de los ojos detectados y predicciones utilizando un modelo previamente entrenado. Se realizó una descripción de la instalación de la imagen para la Jetson Nano, la configuración y armado de periféricos de entrada, la instalación de Python y librerías requeridas, y la familiarización con el entorno de desarrollo de NVIDIA, explorando herramientas y configuraciones de la Jetson Nano.

En general, se realizó el desarrollo de aplicaciones utilizando diferentes tecnologías y plataformas, incluyendo el front-end con ReactJs, la configuración de módulos de comunicación LTE, la implementación de servicios backend con GraphQL, y la detección de rostros y ojos en imágenes en tiempo real y en imágenes fijas. También se describen los pasos para la instalación y configuración de la Jetson Nano, una plataforma de desarrollo de NVIDIA.

9. Bibliografía

Referencias

- [1] React Dev Team, *React*, React. <https://react.dev/> (accedido el 1 de abril de 2023).
- [2] Waveshare Electronics, *SIM7600G-H 4G for Jetson Nano - Waveshare Wiki*, Waveshare Electronics https://www.waveshare.com/wiki/SIM7600G-H_4G_for_Jetson_Nano_4G_connecting (accedido el 16 de abril de 2023).
- [3] Facebook Dev Team, *Introduction to GraphQL — GraphQL — A query language for your API* <https://graphql.org/learn/> (accedido el 4 de abril de 2023).
- [4] NVIDIA, "Get Started with the Jetson Nano Developer Kit", NVIDIA Developer, 2019. [Online]. Disponible: <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro>. [Accedido: Abril 02 2023].
- [5] Nvidia Developer, "Get Started with Jetson Nano Devkit," Nvidia Developer. [En línea]. Disponible: <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write>. [Accedido: 2 de abril de 2023].
- [6] Dusty, N. "Building the Repo - NVIDIA Jetson Inference," GitHub. [Online]. Disponible en: <https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo-2.md>. [Accedido en: 02-abr-2023].