Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

EdgeAI - Sistema Embebido para el reconocimiento y clasificación de expresiones faciales

Taller de Sistemas Embebidos

Integrantes:

Joham Gabriel Mora Castrillo
Emanuel Arturo Barrantes Rodriguéz
Oscar Fernández Zúñiga

Profesor:

Ing. Johan Carvajal Godínez

II Semestre de 2023

Contents

1 Tutorial del sistema

1.1 ¿Cómo operar el Sistema?

Para usar el sistema no es necesario conectarle periféricos al Raspberry Pi 4 además de la cámara ya que este se utiliza en una configuración headless lo que significa que la Raspberry Pi 4 se controla desde un computador externo a través de una conexión SSH. Paso previo: Para poder ejecutar el programa que establece la comunicación entre el computador y la Raspberry Pi 4 es necesario que ambos dispositivos estén conectados en la misma red y escribir el puerto de conexión en el programa que coincida con el de la Raspberry Pi 4.

1. Ejecutar el programa interfaz.exe:
   Para poder operar el sistema de detección facial es necesario que le computador tenga el ejecutable interfaz.exe. Una vez lo tenga solo tiene que ejecutar el programa. Una vez iniciado el programa aparecerá una interfaz 1 la cual consiste en 3 botones, Connect, Run App y Request Data, que sirven para realizar las funciones necesarias del sistema y 2 mensajes de en la parte inferior de la interfaz, Connection Status y Aplication Running, que sirven para informar el estado del sistema.

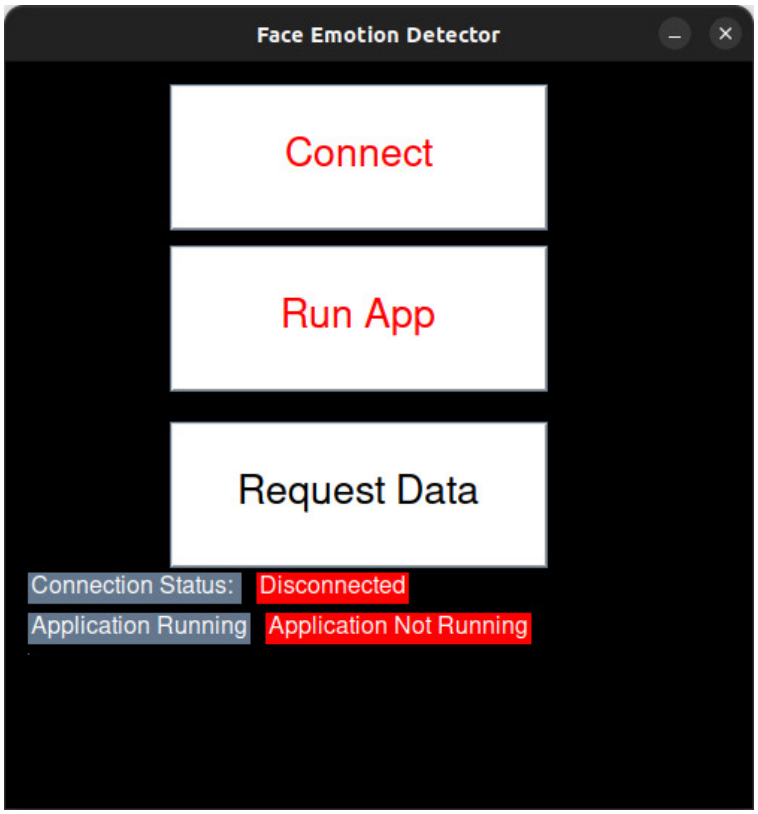

Figure 1: Interfaz gráfica implementada.

2. Se inicia la conexión con el raspberry pi 4:
   Para establecer la conexión entre el computador y la Raspberry Pi 4 simplemente se le debe de dar click al botón de Connect. Si la conexión es exitosa el botón Connect rojo se convertirá en un boton de Disconnect verde y el mensaje de Connection Status indicará el estado de Connected en verde. 2

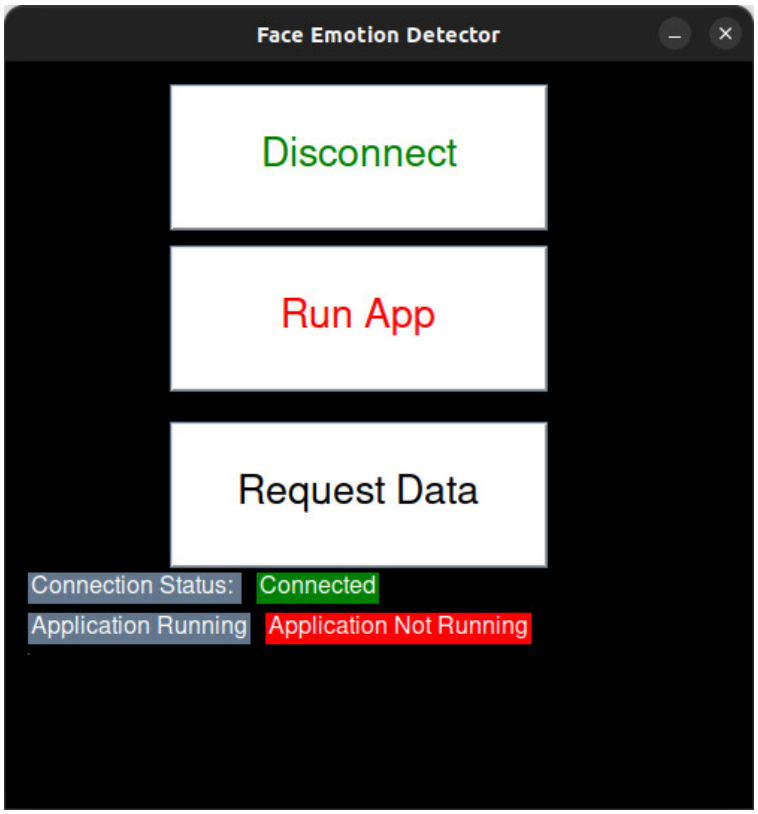

Figure 2: Interfaz gráfica implementada Conectada.

3. Se inicia la aplicación:
   Para iniciar la aplicación de reconocimiento facial hay que dar click en el botón Run App lo que provocara que el botón diga Stop App en color verde y el mensaje de Application Running mostrara un estado de Application Running en verde. A partir de este punto la cámara empieza a capturar imágenes las cuales son procesadas por el modelo de reconocimiento facial y los datos obtenidos se empiezan a escribir en un reporte. 3

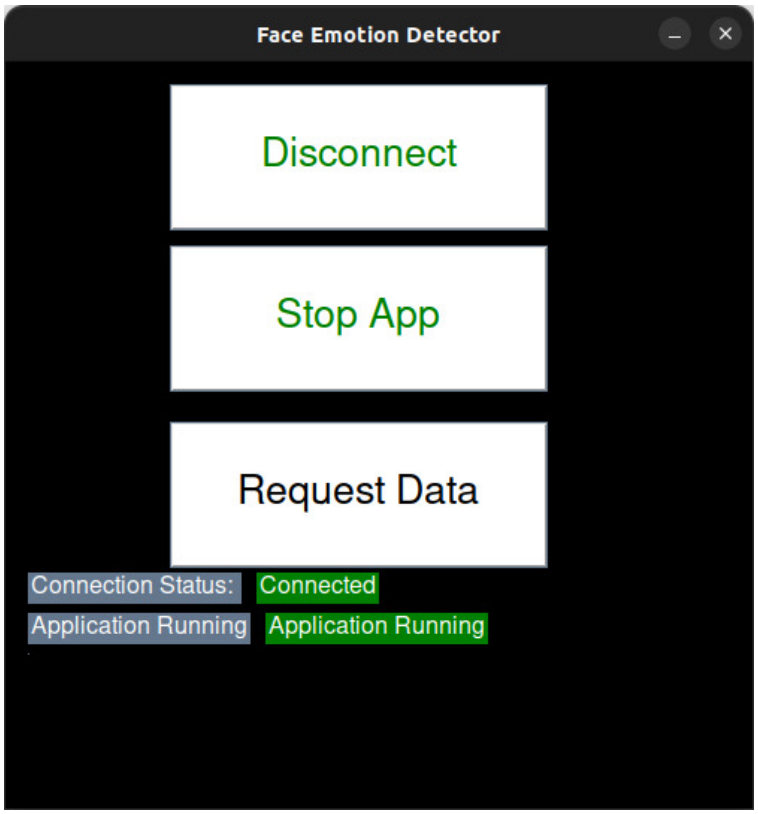

Figure 3: Interfaz gráfica implementada Aplicación Ejecutada.

4. Se detiene la aplicación:
   Para detener la aplicación de reconocimiento facial se tiene que dar click en el botón Stop App lo que provocara que el botón diga Run App y el mensaje de Application Running muestre el estado de Application Not Running en rojo. 2

5. Se solicita el reporte:
   Para descargar el reporte de la Raspberry Pi 4 hay que darle click al botón de Request Data lo que descargara el reporte emotions_detected.csv por defecto en la carpeta /home/usuario en el caso de una maquina Linux o en C:/Users/usuario en una maquina Windows. La carpeta en la que se guarda el reporte se puede cambiar modificando el programa de interfaz. 2

6. Analizar el reporte:
   Una vez el reporte descargado se pueden dirigir a la carpeta en la que se encuentra y abrirlo para realizar el análisis de las emociones capturadas durante el tiempo de ejecución de la aplicación. 14

1.2 ¿Cómo ver el funcionamiento del modelo?

Utilizando la Raspberry Pi 4 en configuración headless no se puede apreciar visualmente el funcionamiento del modelo por lo que si se deseara observar el funcionamiento del modelo de reconocimiento facial es necesario realizar los siguientes pasos.

1. Conectar Periféricos a la Raspberry Pi 4:
   Es necesario conectar un teclado, un mouse y un display a la Raspberry para poder operar directamente la Raspberry Pi 4.

2. Dirigirse al directorio donde se encuentra el modelo:
   Ejecutar el comando:

```
$ cd ../../usr/bin
```

3. Ejecutar el código camara.py:
   Ejecutar el comando:

```
$ python3 camara.py
```

4. Iniciar el programa:
   Para iniciar el funcionamiento del programa se debe de presionar la tecla i 4, una vez hecho eso se observarán las imágenes que están siendo capturadas y procesadas por el modelo.
   5

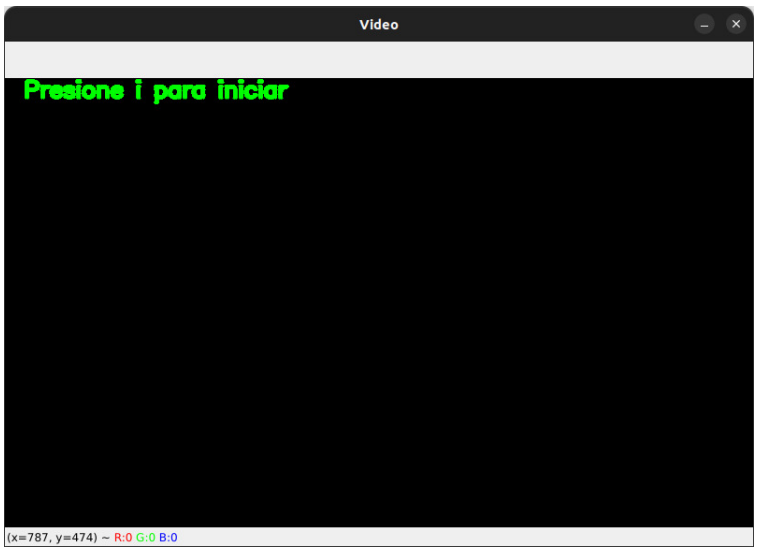

Figure 4: Interfaz gráfica implementada.

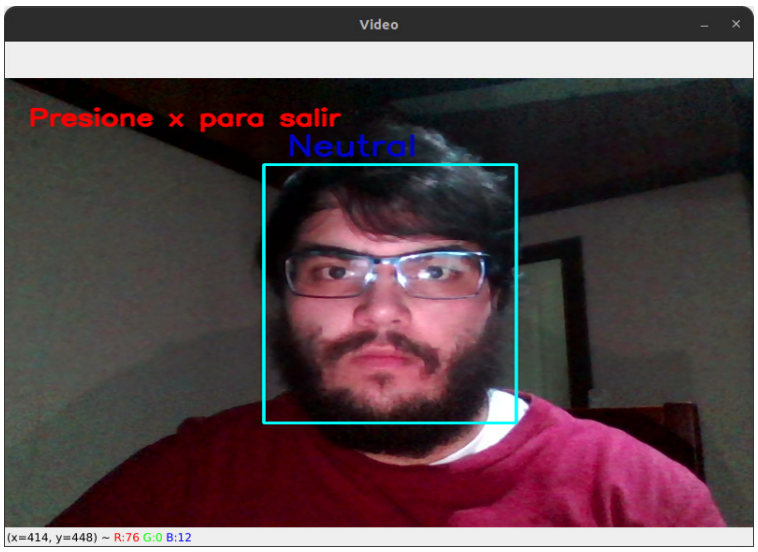

Figure 5: Interfaz gráfica implementada.

5. Detener el programa:
   Una vez se termina de detectar las emociones se debe de presionar la tecla x para cerrar la aplicación.

6. Ver reporte:
   El reporte se encuentra en el mismo directorio que el programa, se llama emotions_detected.csv y se puede visualizar con un editor de texto como vim ya que la Raspberry no tiene programas para leer archivos .csv
   Ejecutar el comando:

```
$ vim emotions_detected.csv
```

2 Flujo de síntesis del software

Para iniciar se selecciono un modelo de detección de emociones que se obtuvo de la siguientes paginas: [1], [2] el modelo esta en un formato h5 por lo tanto fue necesario crear un código de python coverter.py que agarra el modelo en formato h5 y lo cambia a un modelo de tflite [3]. Ya con el modelo de tensorflow lite se escribieron dos códigos de python para correr en modelo, el primero llamado camara.py que muestra una ventana con la cámara donde se hace un cuadro con la cara y se ve que emoción se esta mostrando, para esto se utilizo opencv[4]. El segundo es facedetection.py que ejecuta el la detección pero no muestra nada en pantalla.

Para poder realizar la imagen del proyecto fue necesario crear un meta-layer que los archivos de python, el modelo de tensorflow lite y el archivo ".xml". Para esto se creo meta-proyecto2, comenzando con la del archivo proyecto2.bb que tal como se muestra en la figura 6 dice en la ubicación que se encuentran los archivos y en que dirección quiero que se guarden cuando se cree la imagen.

```
LICENSE = "CLOSED"
LIC_FILES_CHKSUM = ""

SRC_URI += " \
    file://facedetection.py \
    file://camera.py \
    file://model.tflite \
    file://cascade_frontalface_default.xml \
"

S = "${WORKDIR}"

TARGET_CC_ARCH += "${LDFLAGS}"

do_install () {
        install -d ${D}${bindir}
        install -m 0755 facedetection.py ${D}${bindir}
        install -m 0755 camera.py ${D}${bindir}
        install -m 0755 model.tflite ${D}${bindir}
        install -m 0755 cascade_frontalface_default.xml ${D}${bindir}
}
```

Figure 6: Layers.bb

En el archivo layer.conf como vemos en la figura 7 le decimos al builder donde esta ubicado el archivo proyecto2.bb, la prioridad de la meta-layer, las dependencias que tiene y con que versión de poky es compatible.

```
# We have a conf and classes directory, add to BBPATH
BBPATH .= ":${LAYERDIR}"

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "meta-proyecto2"
BBFILE_PATTERN_meta-proyecto2 = "^${LAYERDIR}/"
BBFILE_PRIORITY_meta-proyecto2 = "6"

LAYERDEPENDS_meta-proyecto2 = "core"
LAYERSERIES_COMPAT_meta-proyecto2 = "kirkstone"
```

Figure 7: Layer Config

En la figura 8 podemos observar el árbol de contenidos de meta-proyecto2. Permitiendonos ver de forma mas clara el formato y contenido de esta meta-layer

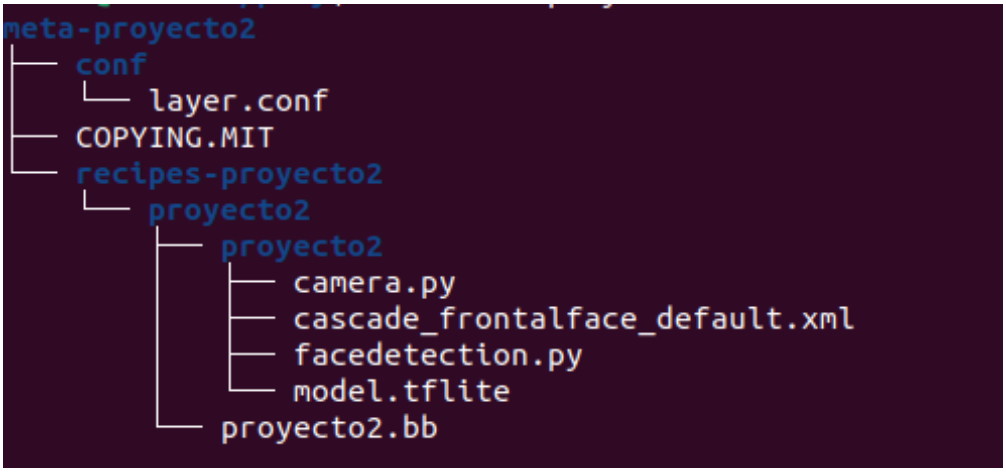

Figure 8: Arbol Meta-Proyecto2

Para realizar la imagen de proyecto fue necesario agregar algunas meta-layers vistas en la figura 9. Como el proyecto se inplementa en una raspberry pi 4 es necesario agregar meta-raspberrypi que nos permite crear imagenes para raspberry y modificar algunas configuraciones de raspberry.Para ello se clono el repositorio de meta-raspberrypi ubicado en la pagina de yocto [5] y se cambio el branch a kirkstone ya que es la versión de poky con la que se esta trabajando. Tambien fue necesario agregar meta-openembedded ya que requeríamos muchos de los metas incluidos en la misma como meta-oe, meta-python, meta-xfce, meta-gnome, meta-multimedia y meta-networking que también se obtuvo meta-openembedded desde la pagina de yocto. Por ultimo también se ocupo meta-tensorflow la cual se clono en el archivo de meta-openembeded porque tiene dependencias con meta-python. Estos metas son necesarios para poder abarcar todas las dependencias de los archivos de python que se van a ejecutar.

```
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/emabr/poky/meta \
  /home/emabr/poky/meta-poky \
  /home/emabr/poky/meta-yocto-bsp \
  /home/emabr/poky/meta-openembedded/meta-oe \
  /home/emabr/poky/meta-openembedded/meta-python \
  /home/emabr/poky/meta-openembedded/meta-xfce \
  /home/emabr/poky/meta-openembedded/meta-gnome \
  /home/emabr/poky/meta-openembedded/meta-multimedia \
  /home/emabr/poky/meta-openembedded/meta-networking \
  /home/emabr/poky/meta-openembedded/meta-tensorflow \
  /home/emabr/poky/meta-proyecto2 \
  /home/emabr/poky/meta-raspberrypi \
  "
```

Figure 9: bb layers proyecto

En el archivo local.conf fue necesario realizar algunas modificaciones para agregar lo necesario para la creacion de la imagen. Lo primero fue asignar la maquina que se tiene como objetivo al crear la imagen en este caso una rasberrypi4-64, despues se configuro el formato que va a tener la imagen creada que es .rpi-sdimg que es el formato que se ocupa para que la raspberry se inicialice con una tarjeta sd. Se agrego un VIDEO-CAMERA que le permite a la raspberry gestionar la web camera que se esta utilizando en el proyecto. Se agregaron los cambios para que la raspberry pueda gestionar los puertos usb. Se agregaron todos las dependencias necesarias para que la imagen tuviera un gestor de ventanas x11. Se pone un tamaño de imagen de

1,2 Gb. Se agregan las librerias de python necesarias para el proyecto y algunos otros pogramas utililes para el funcionamiento del programa. Por ultimo se añade proyecto2 que esta asociado a meta-proyecto.
En la siguiente figura se muestran todas la modificaciones que se hicieron al archivo local.conf

```
MACHINE ??= "raspberrypi4-64"


##IMAGEN PARA RASPBERRY PI##
INHERIT += "rm_work"
IMAGE_FSTYPES = "tar.xz ext3 rpi-sdimg"

##CAMARA WEB##
VIDEO_CAMERA = "1"

MACHINE_FEATURES:append = " usbhost"
DISTRO_FEATURES:append = " usbhost"
IMAGE_INSTALL:append = " usbutils"


##VENTANAS X11##
DISTRO_FEATURES:append = " x11"
IMAGE_INSTALL:append = " xinit"
IMAGE_INSTALL:append = " xserver-xorg"
IMAGE_INSTALL:append = " packagegroup-core-x11-base"
IMAGE_INSTALL:append = " packagegroup-core-x11"
IMAGE_INSTALL:append = " packagegroup-xfce-base"
IMAGE_INSTALL:append = " matchbox-wm"

DISTRO_FEATURES:append = " wifi"
IMAGE_INSTALL:append = " connman connman-client"
GPU_MEM = "16"

IMAGE_ROOTFS_SIZE= "1248000"

##DEPENDENCIAS##
IMAGE_INSTALL:append = "\
      python3-pip\
      python3-numpy\
      python3-matplotlib\
      python3-pybind11\
      tensorflow-lite\
      "

IMAGE_INSTALL:append = " glibc"
IMAGE_INSTALL:append = " nano"
IMAGE_INSTALL:append = " vim"
IMAGE_INSTALL:append = " openssh"
IMAGE_INSTALL:append = " git"
IMAGE_INSTALL:append = " libxcb"
IMAGE_INSTALL:append = " gstreamer1.0"

##Meta-proyecto2##
IMAGE_INSTALL:append = " proyecto2"
```

Figure 10: Local Config

Después de realizar todos los cambios de los puntos anteriores se procede a abrir el builder de poky, abriendo la terminal en la direcion de poky y luego escribiendo el comando source oe-init-build-env, despues vamos a proceder a hacer el bitbke de la imagen para esto vamos a realizar el comando bitbake core-image-base, esto puede tardar varios minutos. Cuando el proceso de bitbake finalize en el directorio poky/build/tmp/deploy/images vamos a encontar el archivo llamado core-image-base-rasberrypi4-64.rpi-sdimg que es la imagen que vamos a utilizar en la raspberry pi.
Para guardar la imagen en la sd en modo boot vamos a utilizar el programa llamado raspberry pi imager que nos proporciona un entorno gráfico donde es fácil incluir la imagen a la sd. Después de crear la sd de boot procedemos a introducirla en la rasberry pi 4 esto nos va a dar un sistema

como el que se muestra en la 9. Hay podemos ver en función el gestor de ventanas x11, el ambiente de escritorio de xfce y la cámara en funcionamiento.

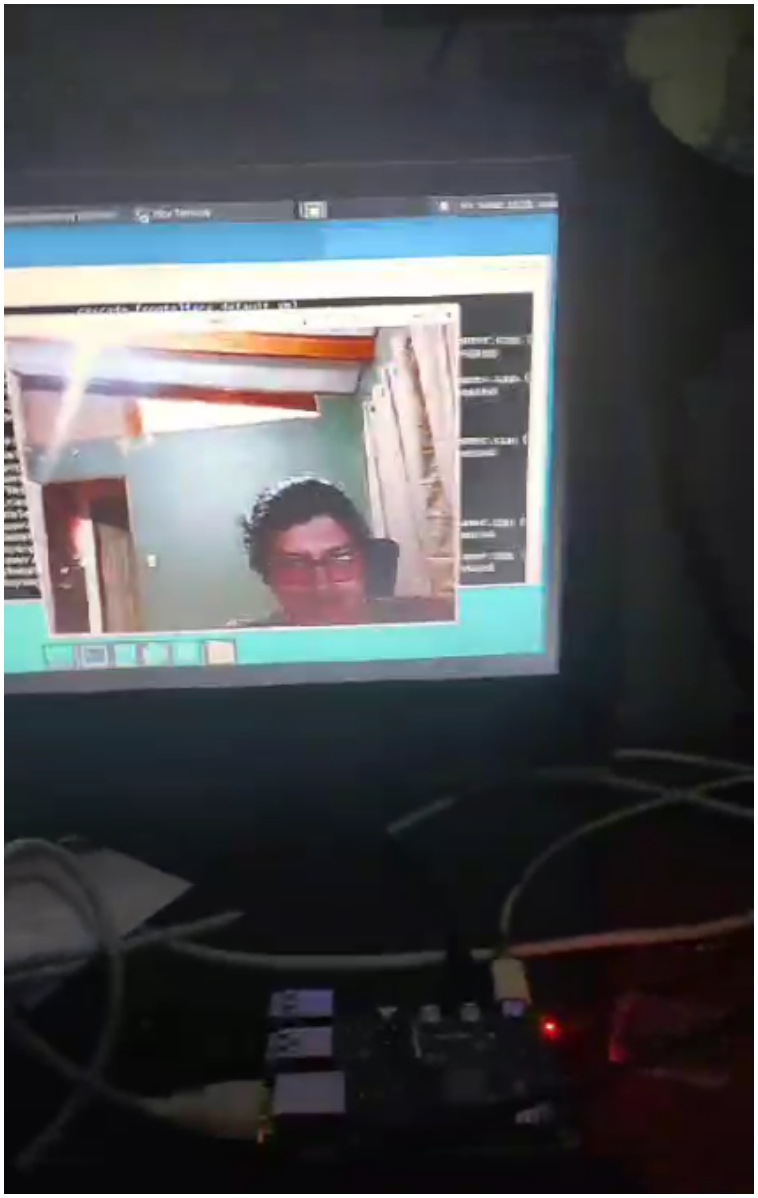

Figure 11: Cámara y Escritorio

Al ver que todo estaba funcionando como era deseado se procedió a probar la aplicación desde la raspberry como vemos en la figura 12.

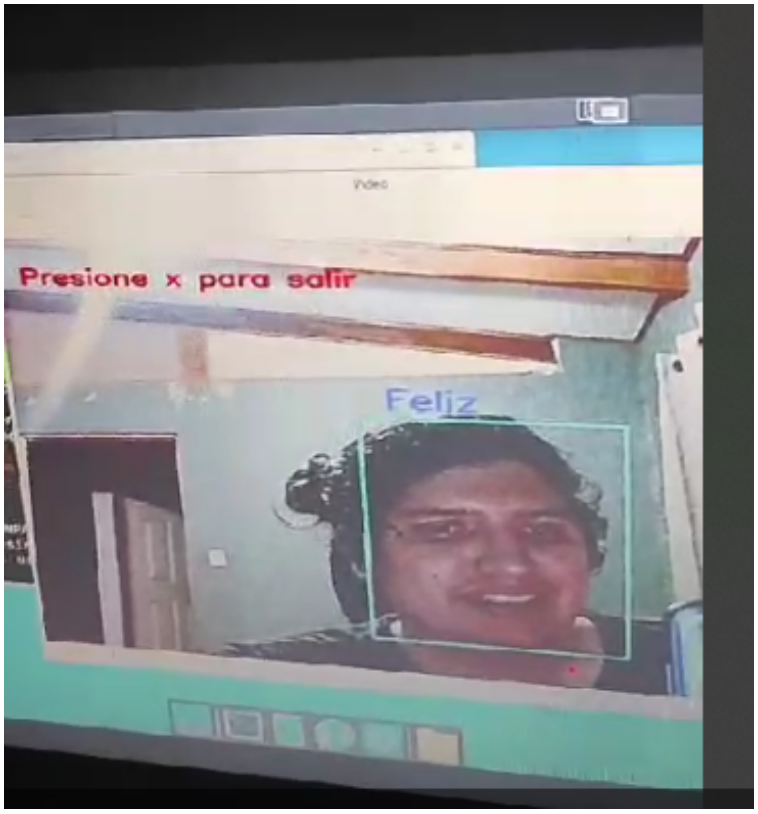

Figure 12: Modelo

Después de comprobar bien el sistema se procedió con la creación de la interfaz gráfica para controlar el sistema mediante el protocolo ssh tal como se hablo en el punto anterior.

3 Consistencia con la propuesta de diseño

Como se puede observar en la figura 13, en este gráfico se evidencia la arquitectura del sistema propuesto en el documento de la propuesta de diseño, es con base en este es que se hará contraste con el sistema que se logró implentar en físico.

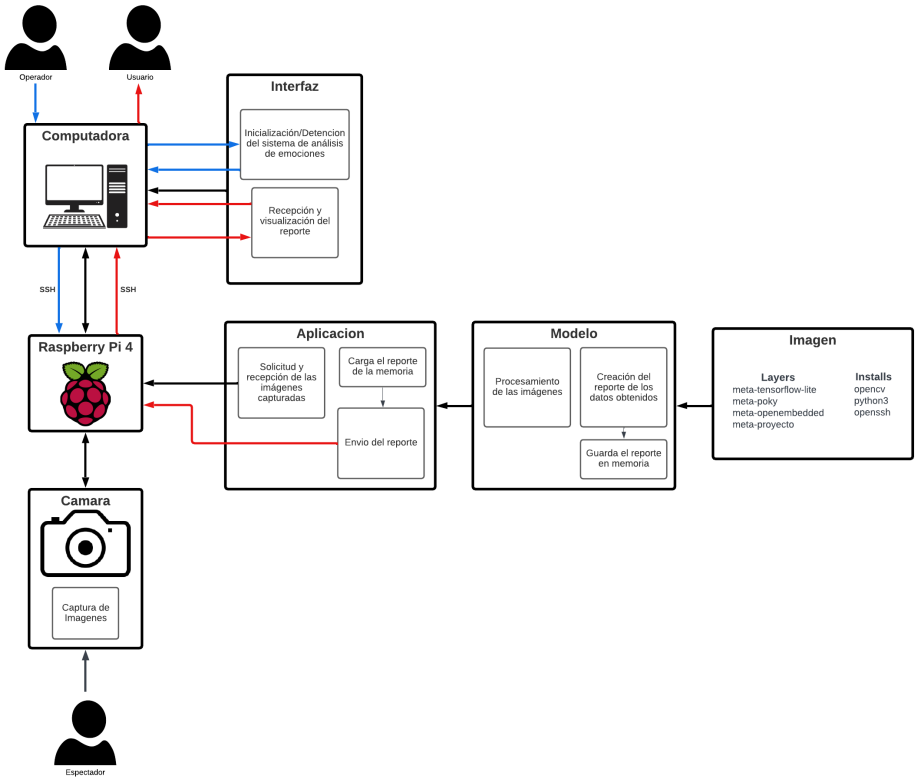

Figure 13: Arquitectura integrada de software y hardware

- Si comparamos el sistema propuesto a nivel de hardware, podemos asegurar que se cumple con lo propuesto, ya que sí utilizamos una cámara conectada por medio de USB a la Raspberry Pi4.
- Con respecto a la comunicación de la Raspberry Pi4 con la computadora en la cual se mostrará la interfaz gráfica con el menú de uso, se cumplió tal como se propuso, ya que efectivamente estamos utilizando comunicación vía SSH, cabe destacar que en esta parte utilizamos un Router propio ya que la red del Tecnológico bloquea las páginas y conexiones no seguras.
- En cuanto a la interfaz gráfica, es intuitiva y fácil de manejar, tal como se explicó en la sección 1 de este documento y se muestra en la figura 1.
- En cuanto al modelo implementado, se logra utilizar un modelo el cual es capaz de reconocer 7 emociones ”Enojado”, ”Disgustado”, ”Temeroso”, ”Feliz”, ”Neutral”, ”Triste”, ”Sorprendido”, es preciso decir que si bien el modelo corriendo el la computadora es capaz de reconocer estas emociones, las condiciones de luz deben ser las adecuadas y el uso de lentes oscuro o gorras puede alterar las medicines, además cuando se pasa dicho modelo a correr en la Raspberry Pi4, este no se comporta de manera ideal, debio principalmente a los recursos que esta posee, sin embargo se logra una detección buena de las emociones.
- En cuanto al reporte que se genera, este se crea en formato .CSV el cual es un formato de excel, en la figura 14 se observa el formato en el que los datos se guardan, estos datos se toman cada segundo y se guardan con una marca temporal.
- Con respecto a la imagen que se propone versus a la imagen implementada en Yocto, la funcional, sí hay bastantes cambios, ya que si contrastamos la figura 13 en la parte de la

imagen versus la figura 9, donde tenemos ya las layers implementadas, observamos que se agregaron más de las que se propusieron inicialmente, esto debido a la funcionalidad del modelo y del código implementado, así como la interfaz gráfica.

- De manera similar, en la figura 13 en la parte de los Installs que corresponde al archivo de local.conf, si comparamos con la figura 10 observamos que de igual manera, se agregan más installs de los que se proponen, también debido a la funcionalidad del sistema, ya que en la propuesta quizá no se tomaron algunas consideraciones.

1	Neutral;2023-11-08 11:53:04.121986
2	Feliz;2023-11-08 11:53:05.349886
3	Feliz;2023-11-08 11:53:06.598445
4	Feliz;2023-11-08 11:53:07.878609
5	Enojado;2023-11-08 11:53:09.152236
6	Enojado;2023-11-08 11:53:10.445506
7	Enojado;2023-11-08 11:53:11.753929
8	Sorprendido;2023-11-08 11:53:13.051867
9	Sorprendido;2023-11-08 11:53:14.317617
10	Sorprendido;2023-11-08 11:53:15.787932
11	Temeroso;2023-11-08 11:53:17.239016
12	Sorprendido;2023-11-08 11:53:18.507789
13	Sorprendido;2023-11-08 11:53:19.794179
14	Neutral;2023-11-08 11:53:21.049843
15	Neutral;2023-11-08 11:53:22.303324
16	Neutral;2023-11-08 11:53:23.587016
17	Sorprendido;2023-11-08 11:53:24.865953
18	Temeroso;2023-11-08 11:53:26.156904
19	Temeroso;2023-11-08 11:53:27.421524
20	Temeroso;2023-11-08 11:53:28.689864
21	Temeroso;2023-11-08 11:53:29.953778
22	Neutral;2023-11-08 11:53:31.219023
23	Neutral;2023-11-08 11:53:32.455318

Figure 14: Datos obtenidos.

En síntesis, se consigue una funcionalidad del sistema que, si bien no coincide de manera exacta con la propuesta inicial, presenta diferecnias más notables en la implementación del software. A lo largo de la ejecución de cualquier proyecto, resulta difícil anticipar situaciones imprevistas, como el malfuncionamiento de una biblioteca o la necesidad de actualizar alguna capa del software. Es precisamente en esta área donde se originan las diferencias más destacadas entre la propuesta original y la implementación final del proyecto.

Es fundamental reconocer que debido a la naturaleza dinámica del entorno tecnológico, los componentes de software están sujetos a cambios inesperados, ya sea por actualizaciones, modificaciones en las dependencias, o incluso por la obsolescencia de ciertos elementos. Estos

factores impredecibles pueden influir significativamente en la ejecución del proyecto, llevando a ajustes y adaptaciones durante su desarrollo.

Por consiguiente, aunque se haya concebido una propuesta inicial sólida, la realidad de la implementación puede diferir debido a las variables inherentes al ámbito del software. La gestión ágil y la capacidad de respuesta ante imprevistos se vuelven esenciales para garantizar que el sistema resultante sea funcional y eficiente, a pesar de las variaciones inesperadas en el entorno tecnológico.

El repositorio desarrollado para este proyecto está disponible en GitHub: `https://github.com/Oscar-FZ/EdgeAI`.

4 Conclusiones

Este proyecto de detección de emociones en los espectadores de cine representa una solución robusta y efectiva, cuya viabilidad se manifiesta tanto en su hardware como en su software, la capacidad del sistema para capturar rostros y detectar emociones con gran precisión cumple de manera sólida con su objetivo principal, destacando su potencial expansión a otros mercados, como obras de teatro y espectáculos en general, si bien la resolución del modelo no es ideal, sí cumple de manera satisfactoria con el objetivo establecido.

En última instancia, este proyecto no solo cumple con su función principal, sino que también se proyecta como una solución adaptable a diversos mercados, abriendo así nuevas oportunidades y posibilidades comerciales, para asegurar el éxito continuo y la adaptabilidad del sistema en diferentes contextos y mercados, se plantean consideraciones cruciales, como pruebas continuas para mantener la precisión, abordar cuestiones éticas y de privacidad, mantenerse actualizado tecnológicamente, diseñar una interfaz de usuario intuitiva, planificar la escalabilidad, ofrecer capacitación y soporte técnico efectivos, realizar análisis de mercado antes de la expansión y valorar la retroalimentación de los usuarios. Estas consideraciones son esenciales para garantizar la versatilidad, eficiencia y sostenibilidad del sistema en su aplicación práctica.

5 Referencias

[1] H. Fahrudin, “Fastemotrecognition,” https://github.com/hfahrudin/FastEmotRecognition, 2019, indonesia.

[2] G. Solano. (2020) Detección de rostros con haar cascades python – opencv. [Online]. Available: https://omes-va.com/deteccion-de-rostros-con-haar-cascades-python-opencv/

[3] T. Lite, “Tensorflow lite,” 2023. [Online]. Available: https://www.tensorflow.org/lite/

[4] OpenCV, “Opencv,” 2023. [Online]. Available: https://opencv.org/

[5] Y. Project, “Yocto project,” 2023. [Online]. Available: https://www.yoctoproject.org/