

# Proyecto Inteligencia Artificial

## IA en Vigilancia y Seguridad

2023



### Integrantes :

- Guarnier Facundo
- Reyes Martín
- Robbio Matias
- Sarmiento Luis María

## Índice

<b>Introducción.....</b>	<b>3</b>
<b>Entorno.....</b>	<b>3</b>
<b>Dataset.....</b>	<b>4</b>
Preparación del Dataset.....	5
<b>Modelos Pre-entrenados.....</b>	<b>7</b>
<b>Nuestro Modelo.....</b>	<b>8</b>
Compilación.....	8
Entrenamiento.....	8
Métricas.....	8
<b>Predicción.....</b>	<b>10</b>
Predicciones de InceptionV3.....	10
Predicciones de MobileNetV2.....	10
<b>Mapa de Calor.....</b>	<b>11</b>
<b>Reconocimiento en Vivo.....</b>	<b>12</b>

# Introducción

El presente informe se enfoca en un proyecto que se sitúa en el ámbito de la inteligencia artificial y la visión por computadora, concretamente en la clasificación de imágenes.

El objetivo fundamental de este proyecto radica en el desarrollo de un sistema capaz de, a través de una imagen, reconocer y diferenciar a los cuatro miembros del equipo. Esto implica la construcción, compilación y entrenamiento de modelos destinados a predecir a qué categoría pertenece una imagen de un conjunto de etiquetas.

A lo largo de este informe, se abordarán las diversas etapas y técnicas involucradas en este proceso, proporcionando una comprensión completa de la metodología empleada.

## Entorno

Para llevar a cabo este proyecto de visión artificial, se utilizaron diversas herramientas y entornos que desempeñaron un papel fundamental en su desarrollo. Estos recursos incluyen:

**Google Drive:** Esta plataforma se empleó para el almacenamiento de imágenes utilizadas en el entrenamiento y evaluación del modelo, facilitando la gestión y compartición de datos.

**Google Colab:** Esta plataforma basada en la nube permitió ejecutar código Python y acelerar el proceso de entrenamiento de modelos de inteligencia artificial utilizando GPU. Además, facilitó la colaboración en tiempo real entre los miembros del equipo.

**Python:** Se eligió Python como lenguaje de programación principal para el desarrollo del proyecto debido a su versatilidad y abundante oferta de bibliotecas y herramientas relacionadas con la visión artificial.

**TensorFlow:** Se utilizó TensorFlow, una biblioteca de código abierto para aprendizaje automático desarrollada por Google, para construir, entrenar y desplegar modelos de redes neuronales.

**Keras:** Es una interfaz de alto nivel que se ejecuta sobre TensorFlow, simplificó la construcción y el entrenamiento de modelos de redes neuronales, lo que permitió un enfoque más eficiente en el desarrollo de soluciones para la visión artificial.

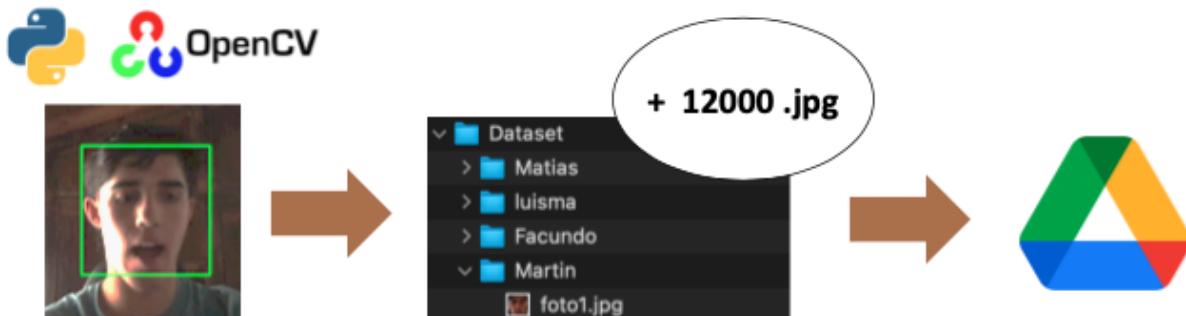
## Dataset

La obtención de un conjunto de datos adecuado es un paso crucial en el desarrollo de modelos de visión artificial. Para facilitar esta tarea en este proyecto, se utilizó un script de Python que recopila imágenes de los rostros de los miembros del grupo. Este script aprovecha la biblioteca OpenCV (cv2) para llevar a cabo la detección de rostros en imágenes en tiempo real.

Para lograrlo, el script utiliza un objeto llamado 'CascadeClassifier' y carga el archivo XML 'haarcascade\_frontalface\_default.xml'. Este archivo XML contiene patrones predefinidos que el algoritmo de detección utiliza para identificar y delimitar los rostros en las imágenes capturadas por la cámara web.

Una vez que se detecta un rostro en una imagen, procedemos a marcarlo y almacenarlo en la carpeta destinada para el respectivo miembro del grupo. Estas imágenes de los integrantes se organizan en sus carpetas individuales correspondientes, lo que facilita su utilización posterior en el proyecto.

Finalmente, subimos la carpeta que contiene las carpetas con las fotos de cada uno de los miembros del grupo a Google Drive para su posterior uso en Google Colab.

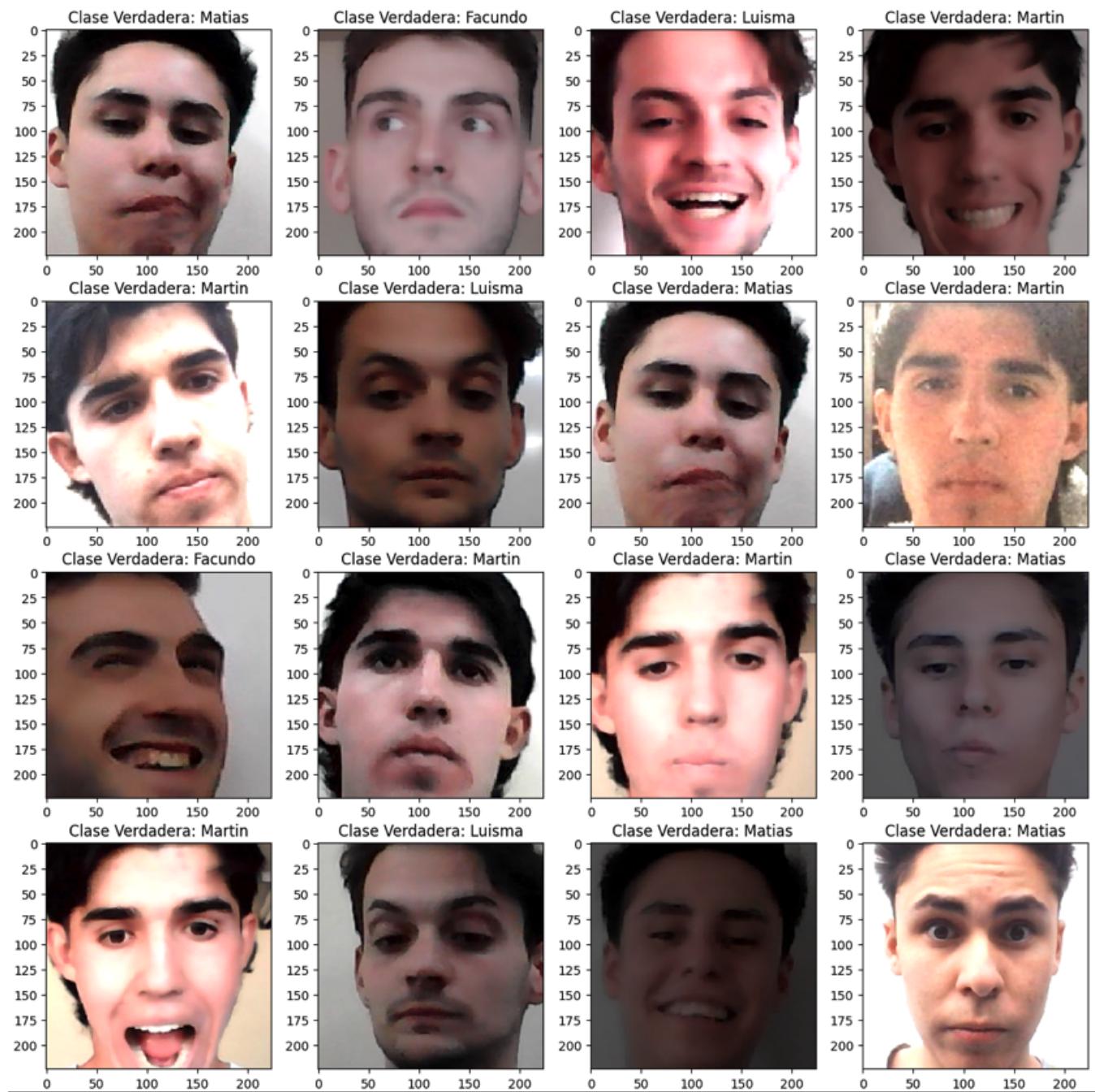


## Preparación del Dataset

Para preparar nuestro conjunto de datos, que consiste en carpetas con imágenes de cada miembro del grupo, nos aseguramos de que cada carpeta tenga la misma cantidad de imágenes, lo que resultó en 2987 imágenes por persona (11948 en total). Luego utilizamos una poderosa herramienta llamada ImageDataGenerator. Este proceso involucra varios pasos esenciales:

1. **Separación de Datos:** Utilizamos ImageDataGenerator para dividir automáticamente nuestras imágenes en datos de validación y entrenamiento. El 20% de nuestras imágenes (2388) se destinan como datos de validación, mientras que el 80% restante (9560) se reserva para el entrenamiento. Esta separación se realiza de manera aleatoria.
2. **Transformación de Imágenes:** Para que las imágenes sean adecuadas como entrada para nuestro modelo, aplicamos transformaciones clave. Re-escalamos los valores de los píxeles. Originalmente, estos valores varían de 0 a 255, pero los convertimos a un rango de 0 a 1. Este paso es fundamental para facilitar el proceso de entrenamiento.
3. **Redimensión Estándar:** Todas las imágenes se redimensionan a un tamaño objetivo de 224x224 píxeles. Este tamaño estándar es común en el procesamiento de imágenes y asegura que todas las imágenes tengan la misma dimensión, lo que simplifica el procesamiento.
4. **Batch para Eficiencia (hiper parámetro):** Para acelerar el proceso de entrenamiento, agrupamos las imágenes en lotes de 32. Esto significa que el modelo procesará 32 imágenes a la vez en cada iteración de entrenamiento.
5. **Etiquetas:** Las etiquetas de cada imagen corresponden al nombre de la carpeta de la cual provienen. Por ejemplo, todas las imágenes dentro de la carpeta "Matias" tienen la etiqueta "Matias". Dado que hay 4 miembros en el grupo, tenemos 4 clases o tipos de etiquetas: Matias, Martín, Facundo y Luisma.
6. **Aumento de Datos:** Además, el ImageDataGenerator nos brinda la capacidad de aumentar nuestros datos. Por ejemplo, aplicamos aleatoriamente el volteo horizontal a las imágenes y ajustamos el brillo de manera aleatoria. Este enfoque diversifica nuestros datos de entrenamiento y puede mejorar la capacidad del modelo para generalizar.

A continuación se puede observar algunas de las fotos que se utilizarán para entrenar y evaluar el modelo:



## Modelos Pre-entrenados

Cuando nos enfrentamos a nuestro desafío en el campo de la visión artificial, adoptamos una estrategia al aplicar el concepto de Transfer Learning. Esta táctica se basa en aprovechar modelos previamente entrenados por expertos, que presentan dos ventajas fundamentales: una mayor capacidad de procesamiento y acceso a conjuntos de datos masivos. Este enfoque nos proporcionó una base sólida sobre la cual construir nuestro modelo de visión artificial, acelerando significativamente el proceso y mejorando la calidad de los resultados.

Para obtener estos modelos pre entrenados, hicimos uso de la biblioteca Keras. En este proceso, omitimos la parte superior del modelo, que generalmente consta de capas densas (totalmente conectadas) utilizadas para la clasificación en las clases originales para las que fueron entrenados. Esto se debe a que necesitábamos agregar nuestras propias capas de clasificación para que el modelo pueda reconocer a los 4 miembros del grupo.

Además, congelamos las capas del modelo pre entrenado, lo que significa que las hicimos no entrenables. Como resultado, los pesos pre entrenados del modelo no se modificarían durante el proceso de entrenamiento con nuestros propios datos, lo que nos permitió retener el conocimiento previo del modelo.

Después de numerosas pruebas con modelos pre entrenados, llegamos a la conclusión de que InceptionV3 y MobileNetV2 son los que se adaptaron mejor a nuestro problema. Ambos son arquitecturas de redes neuronales convolucionales desarrolladas por Google y utilizadas en tareas de visión por computadora. La diferencia radica en que InceptionV3 es una arquitectura de red neuronal más grande y profunda en comparación con MobileNetV2. MobileNetV2, por otro lado, está diseñada para ser una arquitectura "ligera" y eficiente en términos de recursos, optimizada para dispositivos móviles y aplicaciones en tiempo real.

Es importante destacar que, en la teoría, los resultados de InceptionV3 parecían superiores a los de MobileNetV2 según los gráficos de precisión y pérdida. Sin embargo, en la práctica, durante la implementación y las pruebas en tiempo real, MobileNetV2 demostró un rendimiento superior. Esto subraya la importancia de no depender exclusivamente de métricas teóricas, ya que la efectividad real de un modelo puede variar según el escenario de implementación y las condiciones del mundo real.

## Nuestro Modelo

A ambos modelos pre-entrenados, InceptionV3 y MobileNetV2, se les ha ampliado su arquitectura mediante la incorporación de una capa de salida. Esta adición se ha realizado con el fin de permitir la clasificación en cuatro categorías específicas, representando a los miembros del equipo.

El proceso de ampliación del modelo incluye la introducción de una capa de convolución, seguida de una capa de dropout diseñada para regularizar el modelo. Posteriormente, la salida se aplanó, y se han añadido dos capas densas (fully connected), siendo la última capa compuesta por cuatro unidades y empleando una función de activación softmax. Esto habilita la clasificación en las cuatro clases correspondientes a los integrantes del equipo.

## Compilación

Una vez que hemos definido los modelos, avanzamos a la fase de configuración. En nuestro caso, hemos optado por utilizar el optimizador conocido como "Adam", ajustando su tasa de aprendizaje a 0.0001. La función principal de este optimizador es optimizar los pesos de la red neuronal con el propósito de minimizar la función de pérdida durante el proceso de entrenamiento.

En cuanto a la función de pérdida, hemos seleccionado "CategoricalCrossentropy" para medir la discrepancia entre las predicciones del modelo y las etiquetas reales.

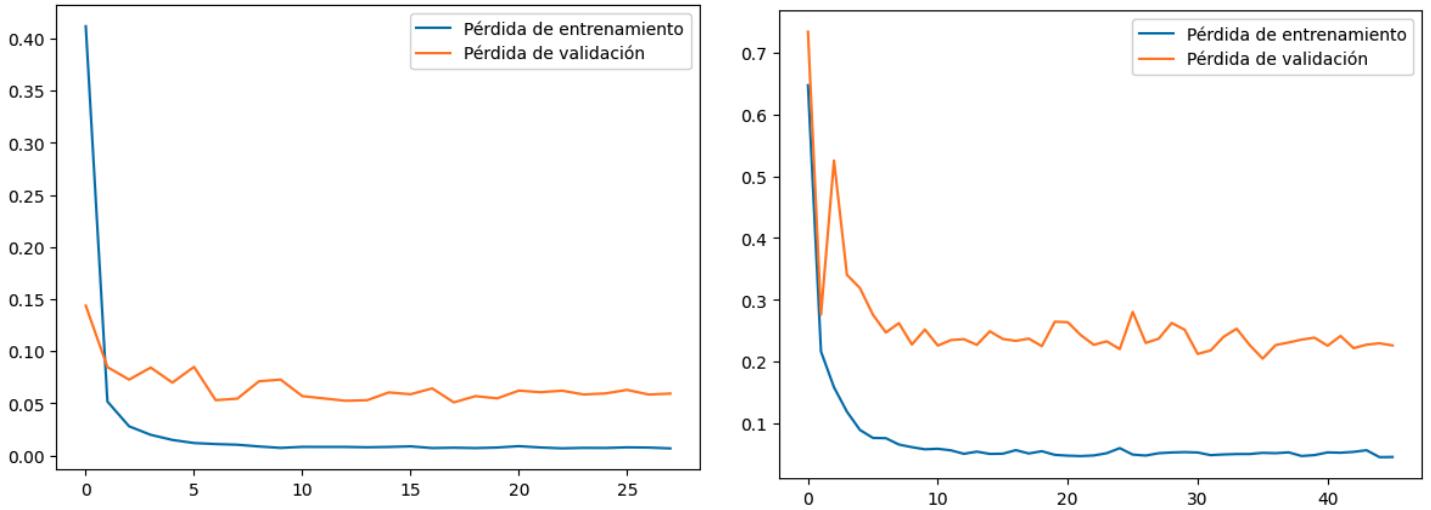
## Entrenamiento

Las dos redes fueron entrenadas de la misma manera. Utilizamos una función de parada temprana (EarlyStopping) que finaliza el entrenamiento si no se observan mejoras significativas después de un cierto número de épocas. También se aplica un ajuste dinámico de la tasa de aprendizaje mediante la función "LearningRateScheduler", lo que permite reducir gradualmente la tasa de aprendizaje a medida que el modelo converge. Además, se utiliza "ModelCheckpoint" para guardar el modelo en un punto donde alcanza el mejor rendimiento durante el entrenamiento.

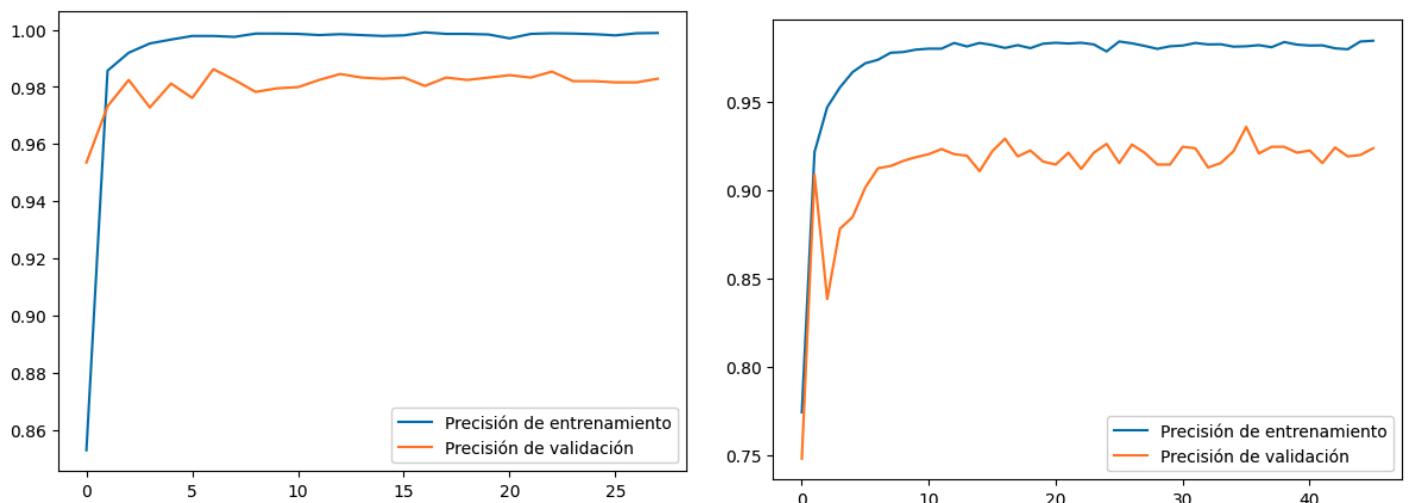
## Métricas

El gráfico de pérdida es una métrica fundamental que nos proporciona información sobre cómo el modelo está mejorando a lo largo del tiempo durante el entrenamiento. Idealmente, la pérdida de entrenamiento y de validación deben disminuir en cada época, lo que indica que el modelo está aprendiendo y ajustándose a los datos de entrenamiento. La métrica de precisión, por otro lado, indica la proporción de predicciones correctas en relación con el número total de ejemplos. En términos más simples, cuantifica la capacidad del modelo para realizar predicciones certeras. Además, hemos llevado a cabo un análisis adicional mediante la creación de una matriz de confusión. Esta herramienta nos permite evaluar cómo los modelos realizan sus predicciones, proporcionando una visión detallada de los posibles "falsos positivos".

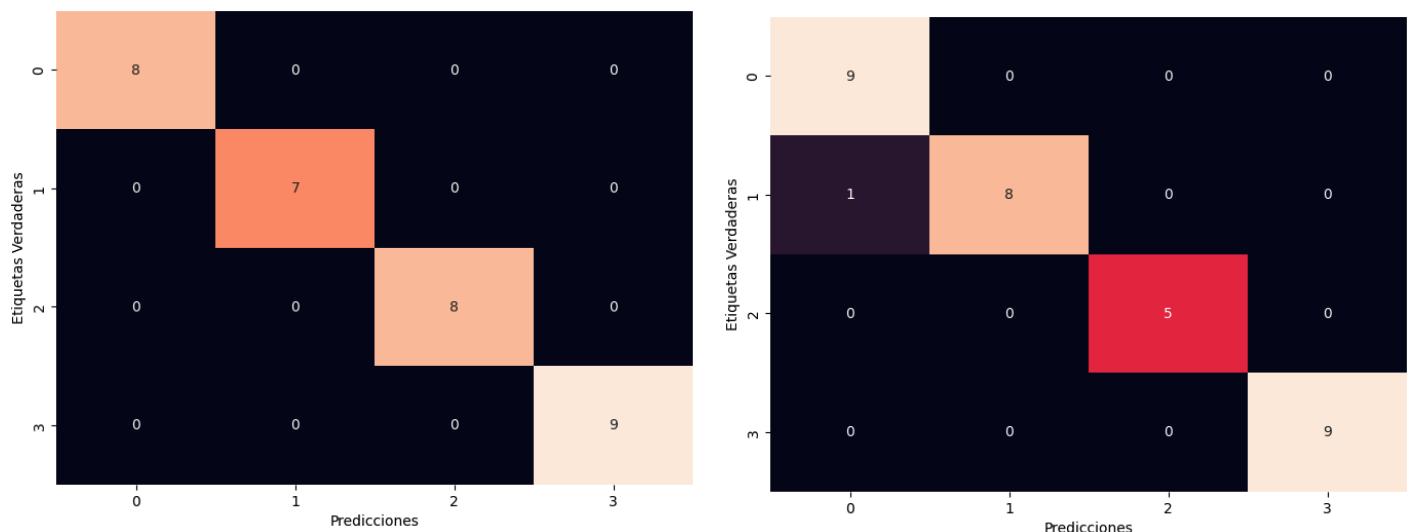
### Comparación gráficos de pérdida de InceptionV3 y MobileNetV2 respectivamente



### Comparación de gráficos de precisión de InceptionV3 y MobileNetV2



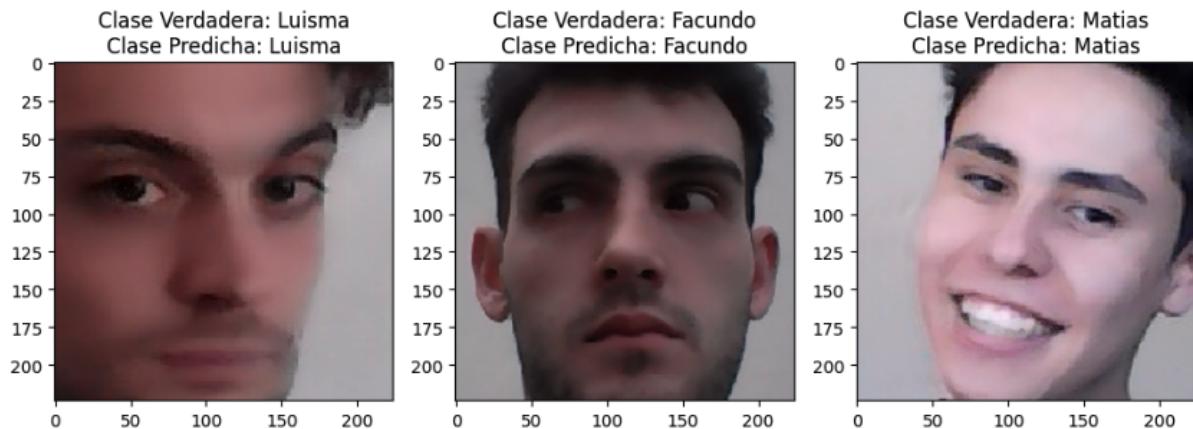
### Comparación de matrices de confusión de InceptionV3 y MobileNetV2



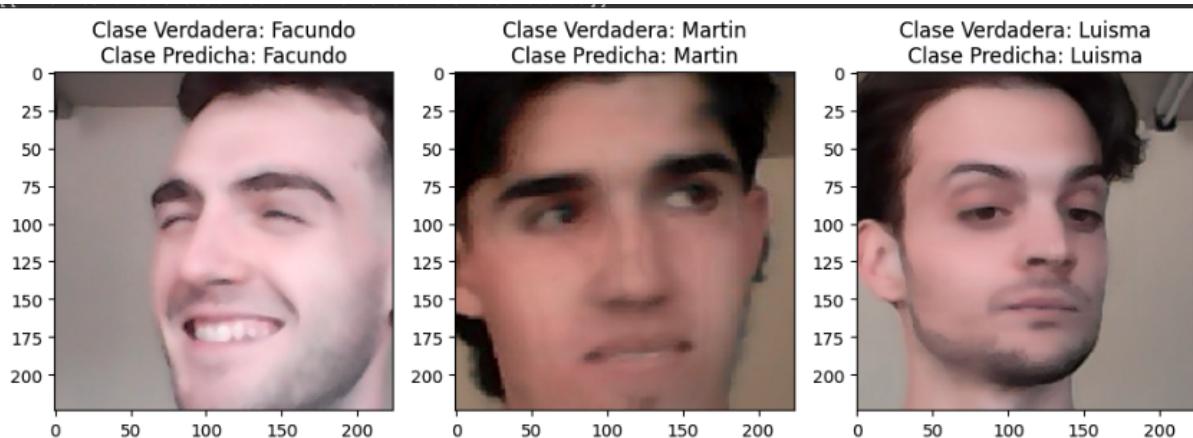
## Predicción

A continuación, presentaremos ejemplos concretos que ilustran las predicciones efectuadas por nuestros modelos

### Predicciones de InceptionV3

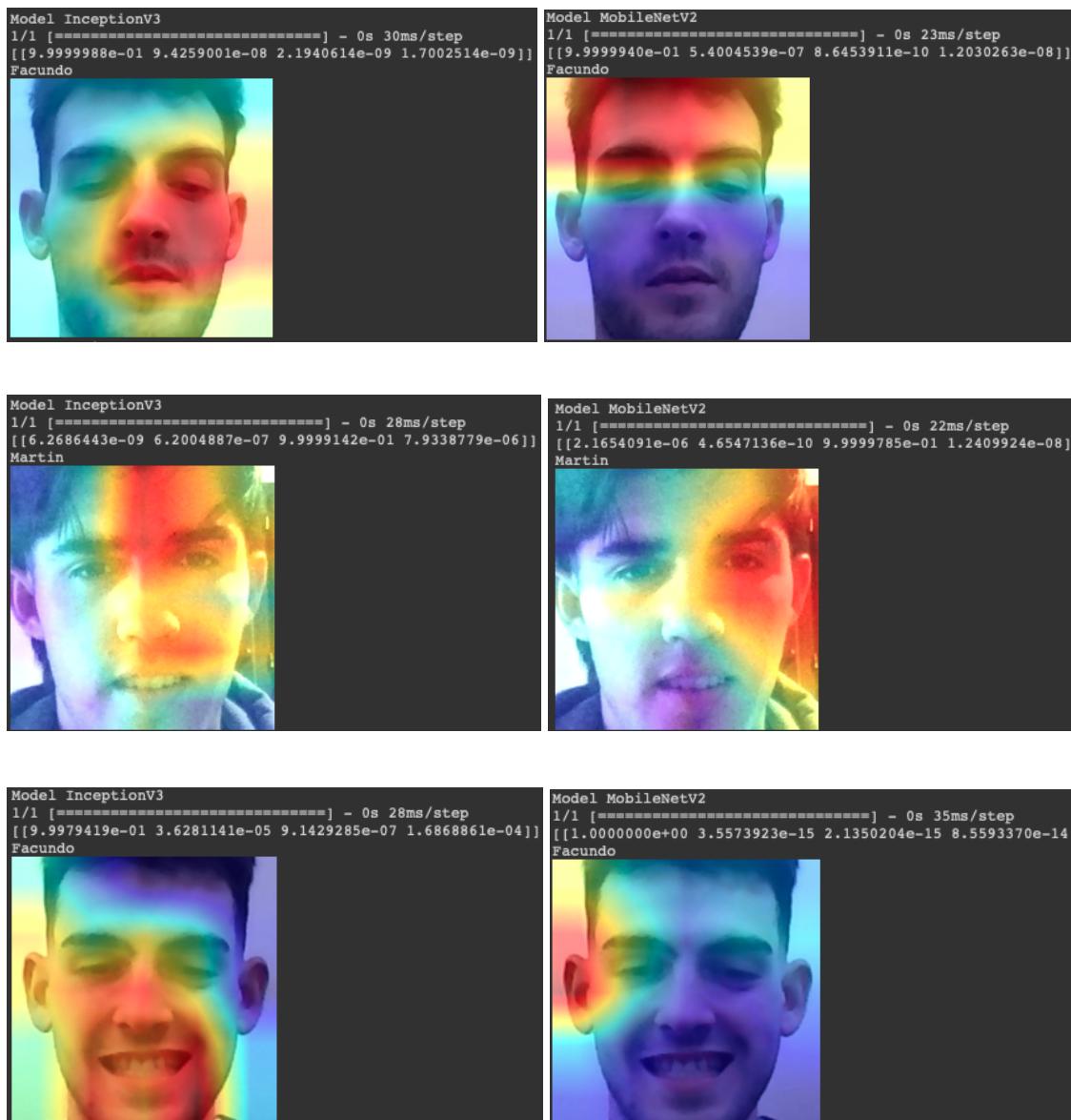


### Predicciones de MobileNetV2



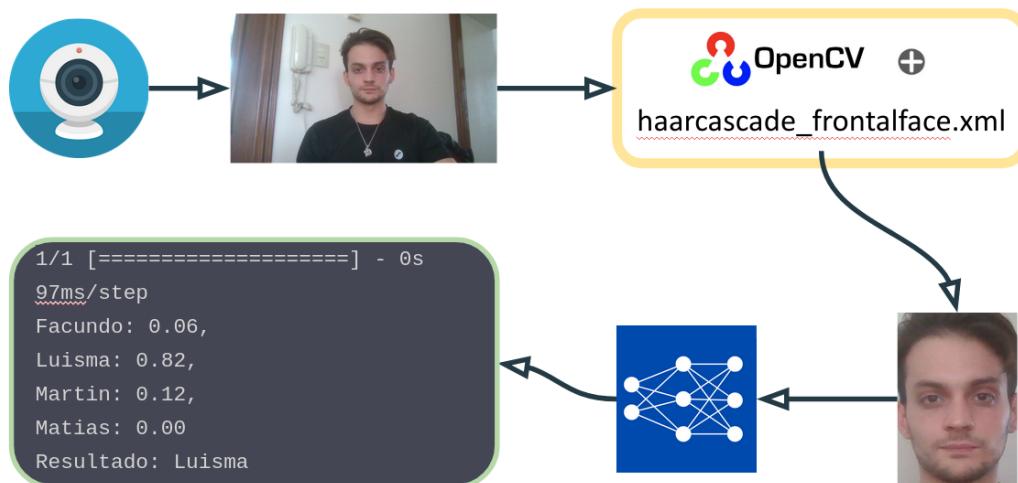
## Mapa de Calor

Para comprender cómo ambos modelos toman sus decisiones, hemos empleado la función GradCAM, que genera y visualiza mapas de activación basados en Grad-CAM (Gradient-weighted Class Activation Mapping). Grad-CAM es una técnica que permite identificar las áreas fundamentales en una imagen que influyen en las decisiones de un modelo de aprendizaje profundo. Esta función se utiliza para destacar las regiones cruciales en una imagen que afectan la decisión del modelo en relación con una clase específica. Como resultado, obtenemos una imagen que resalta las áreas de mayor activación en la imagen original.



## Reconocimiento en Vivo

Una vez entrenados ambos modelos los exportamos como checkpoints para poder utilizarlos en el script "reconocer\_en\_vivo.py". En este se utiliza la misma biblioteca OpenCV usada para la obtención del dataset. Una vez que detecta un rostro usando el CascadeClassifier este se lo envía al modelo entrenado y este devuelve las probabilidades de que pertenezca a cada clase, mostrando así sobre el recuadro la clase con mayor probabilidad y su porcentaje de certeza.



Ejemplo de ejecución del sistema de reconocimiento facial en vivo

