UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Manejo de brazo robótico en función del reconocimiento de género utilizando una red neuronal convolucional

Leonel David Miranda Guerrero

Daniel Alejandro Jiménez Bosmediano

Ingeniería Electrónica y Automatización

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniería en Electrónica

Quito, 15 de Mayo de 2022

Universidad San Francisco de Quito USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Manejo de brazo robótico en función del reconocimiento de género utilizando una red neuronal convolucional

Leonel David Miranda Guerrero

Daniel Alejandro Jiménez Bosmediano

Nombre del profesor, Título académico **Diego Benítez, Ph.D.**

Quito, 15 de Mayo de 2022

© DERECHOS DE AUTOR

Nombres y apellidos: Leonel David Miranda Guerrero

Código: 00203532

Cédula de identidad: 1719626150

Lugar y fecha: Quito, 15 de Mayo de 2022

Nombres y apellidos: Daniel Alejandro Jiménez Bosmediano

Código: 00202031

Cédula de identidad: 1752740413

Lugar y fecha: Quito, 15 de Mayo de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

La inteligencia artificial es una herramienta que permite a los seres humanos interactuar con las máquinas. Este proceso ha ido evolucionando constantemente y determinará el futuro de la tecnología. En el presente trabajo se desarrolla una red neuronal convolucional con el fin de entrenar un modelo capaz de reconocer género. Adicionalmente, con el resultado del reconocimiento se implementa el control de un brazo robótico de la marca Yahboom. El brazo robótico cuenta con el ordenador Jetson Nano y con 6 servos que controlan la posición de este. Se desarrollaron tres prototipos a lo largo del proyecto, con sus respectivas pruebas. En el primer modelo caffe, debido a su simpleza se obtuvo un margen de error del 29%. Posteriormente, tras el cambio del modelo a uno desarrollado con tensor Flow, el prototipo número 2 obtuvo un margen de error del 9%. Por último, tras mejorar el prototipo anterior, el prototipo final obtuvo un 1.8% de margen de error.

Palabras clave: Inteligencia artificial, Brazo Robótico, Reconocimiento Facial, Reconocimiento de Género, Redes Neuronales, Red Convolucional, Algoritmo, Entrenamiento, Base de Datos, Python.

ABSTRACT

Artificial intelligence is a tool that allows humans to interact with machines. This process has been constantly evolving and will determine the future of technology. In the present work, a convolutional neural network is developed to train a model capable of recognizing gender. Additionally, with the result of the recognition, the control of a robotic arm of the Yahboom brand is implemented. The robotic arm has the Jetson Nano computer and 6 servos that control its position. Three prototypes were developed throughout the project, with their respective tests. In the first caffe model, due to its simplicity, a margin of error of 29% was obtained. Subsequently, after changing the model to one developed with Tensor Flow, prototype number 2 obtained a margin of error of 9%. Finally, after improving the previous prototype, the final prototype obtained a 1.8% margin of error.

Keywords: Artificial intelligence, Robotic Arm, Facial Recognition, Gender Recognition, Neural Networks, Convolutional Network, Algorithm, Training, Database, Python.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Uno de los principales campos de desarrollo tecnológico actualmente es la interacción entre maquina y ser humano. Las maquinas son herramientas que facilitan la ejecución de procesos y lo hacen en medida de su complejidad. El desarrollo de las maquinas ha llegado a tal punto que ahora podemos interactuar con las mismas de formas más complejas. Sus aplicaciones van desde maquinas capaces de responder a comandos de voz, hasta robots autónomos. La tecnología que hace esto posible es la Inteligencia Artificial (IA), la cual permite que la maquina simule el comportamiento humano.

En el presente trabajo se hace uso de la IA para entrenar un algoritmo capaz de detectar el género de una persona. Para esto utilizamos una red neuronal convolucional. Esta función es implementada en una computadora Jetson nano integrada a un brazo robótico de la marca Yahboom. El objetivo del proyecto es controlar el brazo robótico para que realice la entrega de un objeto dependiendo de la predicción de género. La predicción es llevada a cabo mediante una cámara web y el modelo pre entrenado.

El documento presenta inicialmente un marco teórico, el cual contiene toda la investigación previa necesaria para el entendimiento y desarrollo del proyecto. A lo largo del proyecto se desarrollaron tres prototipos. Es por esto por lo que la sección de desarrollo se encuentra dividida en tres partes, las cuales contienen el desarrollo del código de entrenamiento y el código de aplicación para cada prototipo. Además, en cada uno se realizó las pruebas practicas respectivas. En la siguiente sección se presentan los resultados de cada prototipo comparando su eficacia. Posteriormente se presentan las conclusiones que resumen todos los hallazgos importantes del proyecto. Finalmente se encuentra la sección de anexos que incluyen todos los códigos al completo utilizados a lo largo del proyecto.

MARCO TEORICO

Inteligencia artificial

La Inteligencia Artificial (IA), es un concepto que ha ido tomando forma con cada avance tecnológico. Todo esto surge a partir de la necesidad de automatizar procesos por medio de las máquinas. Se buscaba que estas máquinas potencien las capacidades físicas e intelectuales del hombre. Todo esto fue progresando hasta llegar a las computadoras de hoy en día, las cuales son el mayor referente de inteligencia artificial. Si hacemos una analogía las máquinas serían el cuerpo, sus procesadores vendrían a ser el cerebro y por último el software la mente. Todo esto como conjunto conformaría un ser informático cuyo objetivo es realizar diversas tareas por nosotros. En base a esto tenemos que la Inteligencia Artificial es la capacidad que tiene una máquina de imitar las capacidades intelectuales del hombre (Rouhiainen, 2018). Pero para que una de estas máquinas pueda decirse que cuenta propiamente con inteligencia artificial debe cumplir con ciertos requisitos, los cuales de igual forma determinarán con qué tipo de IA nos encontramos.

En informática se describe un sistema inteligente a el equipo capaz de percibir el entorno y completar acciones que maximicen el porcentaje de éxito del objetivo. La meta general de la IA es imitar las funciones cognitivas del ser humano como lo son: Percibir, razonar, aprender y resolver problemas. Dicho esto, se pueden plantear distintos tipos de inteligencia artificial.

Según Arend Hintze, profesor de Biología Integrada y Ciencias de la Computación de la Universidad de Michigan, se pueden diferenciar 4 tipos de IA (Blanco & Cohen, 2018).

- Máquinas reactivas. - Este tipo de IA es el sistema más básico y se basa en la capacidad de respuesta. No cuentan con una memoria por lo que no retendrán recuerdos. Su toma de decisiones no está basada en experiencias pasadas. Al sistema se le presenta una situación y al analizarla brevemente reaccionará a esta con una solución.

- Máquinas de memoria limitada. - En este caso ya cuenta con una memoria en la cual podrá almacenar información y recordarla en cuanto sea necesario. Sin embargo, como su nombre lo indica esta memoria es limitada con lo que dependerá del tamaño a almacenar que se proporcione. Su funcionamiento se basa en el aprendizaje con algoritmos diseñados para formar modelos y respuestas de acuerdo con las necesidades del usuario. Toma en cuenta tareas comunes para anticipar sus tareas.

- Máquinas con teoría de la mente. - La teoría de la mente en psicología significa la capacidad de formar representaciones sobre el mundo y sobre otros seres. En otras palabras, tiene que ver con la forma como interactuamos con el resto de los seres vivos. Por lo tanto, lo que se busca en este tipo de máquinas es que comprendan como pensamos y como sentimos. Con esta información deberán tomar sus decisiones basado en lo que creen que esperamos de ellos y en como esperamos ser tratados. Tienen en cuenta la información objetiva y las emociones implicadas en la conducta.

- Máquinas con autoconciencia. - En este caso se trata de una tecnología utópica en la que la máquina sea consiente sobre sí misma y que reúna todas las habilidades mencionadas anteriormente. Este es la meta final de la IA en la que las máquinas se conviertan en un ser informático que sea un solucionador de problemas y aprenda de manera autónoma .

En la actualidad es común encontrarse con distintos softwares y máquinas que cuentan con IA. Uno de los ejemplos más claros son los asistentes de voz encontrados en los dispositivos móviles y en bocinas. También está presente en el servicio al cliente automático y en las recomendaciones al usuario. En este trabajo nos enfocamos en la programación de un algoritmo que aprenda en base al entrenamiento de una base de datos. Es decir que cuenta con una memoria e información objetiva. Además, lo que se espera es que sea una máquina reactiva, la cual procese información en tiempo real y la compare con sus datos aprendidos para así responder con una acción. La información que se proporciona está en formato de imagen.

Procesamiento de imagen

Para un mejor entendimiento del proyecto, es necesario explicar el proceso de abstracción de información. Como se mencionó, la información a utilizar serán imágenes con sus respectivas etiquetas. Sin embargo, estas imágenes antes de ser utilizadas en el entrenamiento del algoritmo pasan por un proceso de preprocesamiento. De manera general a continuación se presenta un detalle de los factores a tomar en cuenta.

Ejes

El ojo del ser humano para reconocer que existe un objeto en una imagen se guía a partir de los ejes. Estos son contornos que apuntan en distintas direcciones dándole forma al objeto. Nosotros los distinguimos cuando notamos un cambio drástico de valor en la tonalidad de pixeles contiguos. Es decir que los ejes se van formando por los colores que percibimos en las imágenes ya sean imágenes a color o en blanco y negro. Lo que hacen las máquinas es percibir la imagen como un conjunto de pixeles. Es decir que dividen la imagen en pequeñas cuadrículas a las cuales se les asigna un número que informa el tono de color. Por ejemplo, si tenemos una imagen de blanco y negro, el

negro tendrá un valor de 0 mientras que el blanco llega a un valor de 255. Todos los valores intermedios se conocen como la escala de grises. Con los valores de cada píxel ya contamos con información importante para que la máquina determine si existe o no un objeto. Este proceso lo hace por medio de una convolución e iteración. La convolución es un proceso en el que se compara cada valor numérico con cada valor próximo al mismo (Palomares et al., 2016). La iteración significa que se repite este proceso para cada píxel en la imagen. Para este proceso se utiliza un núcleo que es una matriz de 3x3 pixeles a los cuales se asigna un valor numérico siendo el central cada píxel de la imagen. Existen distintos núcleos que se pueden utilizar para distintos propósitos. Por ejemplo, un núcleo identidad que tras el proceso de convolución devuelve la misma imagen. La matriz de ese núcleo se vería de la siguiente manera:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

También podemos crear un filtro por medio del núcleo de desenfoque normalizado que tiene la forma:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

De esta forma podemos así mismo utilizar un núcleo que como resultado final resalte los cambios drásticos de colores, formando así una especie de contorno el cual determina la existencia de un objeto.

Color

Mientras más pixeles tenga una imagen mayor será su calidad puesto que contiene más información. Sin embargo, el color en una imagen no determina en si al objeto como tal. Si tenemos un gato en una imagen lo podremos reconocer fácilmente

sin importar si está en formato blanco/negro o a color. Esto es un punto que se puede aprovechar dentro del procesamiento de imágenes. Esto se debe a que las imágenes a blanco y negro contienen la información esencial para determinar un sujeto, animal o cosa. En el mundo computacional el usar imágenes B/N significa que ocupa menos memoria por lo que es más rápido trabajar con menos cantidad de datos. En las imágenes a color se dice que tienen 3 canales por lo que técnicamente la imagen a color es una combinación de 3 imágenes. Esta combinación se da con 3 colores primarios: Rojo, verde y azul (Báez et al., 2004). Por el contrario, una imagen a blanco y negro únicamente tendrá 1 canal. En el momento de entrenar una red con imágenes, se debe especificar con cuantos canales se trabaja, 1 o 3.

Tamaño y orientación

El tamaño de las imágenes dentro de nuestra base de datos es muy importante. El tamaño de las imágenes determina el espacio que van a ocupar y por lo tanto el tiempo que va a tardar en procesarla. Reducir el tamaño de las imágenes es beneficioso siempre y cuando no se pierda el detalle de lo que queremos identificar. Además, para efectos prácticos se utiliza el mismo formato en todas las imágenes. Esto hace que exista un sentido y un manejo de información organizado. Además, del tamaño es importante mantener un sentido por lo que se puede redimensionar la imagen ya sean verticales u horizontales para que todas estén arregladas uniformemente.

Etiquetado y normalización

Cuando se trabaja con algoritmos de reconocimiento parte de la información que se proporciona es la etiqueta. La imagen por si sola contiene información, pero si queremos entrenar una red para que reconozca objetos y nos dé como respuesta el

nombre de este, debemos de contar con etiquetas en todas las imágenes de la base de datos. Por otra parte, para determinar la respuesta regresamos a los pixeles. Cuando realizamos el entrenamiento el producto final es un arreglo que contiene todos los valores esenciales de la imagen. Por lo que es importante normalizar estos valores con el fin de obtener números entre 0 y 1. Así se puede configurar que los valores más cercanos a 0, den como respuesta que se trata del objeto A y los valores más cercanos a 1 se pueden asignar al objeto B.

Arquitectura redes neuronales

Las redes neuronales simulan la capacidad de los humanos para memorizar y asociar hechos . Estas redes fueron creadas debido a que existían problemas que no se podían resolver con algoritmos comunes, ya que les hacía falta algo muy común para los humanos como es la experiencia. Al igual que funciona el cerebro humano, estas redes se componen de unidades básicas de procesamiento de información llamadas neuronas. Las neuronas se encuentran interconectadas para poder interactuar de forma constante. La red neuronal tiene la capacidad de aprender, eso quiere decir que es capaz de resolver problemas que inicialmente no podía resolver (Díez et al., 2001). Son utilizadas principalmente para reconocer patrones en las imágenes por medio de la visión artificial y pueden mejorar su funcionamiento aumentando la base de datos o modificando la arquitectura. La arquitectura está ligada directamente a los requerimientos de la aplicación deseada.

El uso de redes neuronales tiene varias ventajas que ayudan a la resolución de problemas, empezando por el aprendizaje adaptativo que permite desarrollar tareas utilizando como base un entrenamiento previo. También posee la capacidad de crear su propia representación de la información que obtuvo de la etapa de aprendizaje. Por último, funcionan

en tiempo real es decir que adaptan los pesos de las neuronas incluso cuando se encuentran operativas.

Las redes neuronales funcionan de tal manera que la información inicial pasa por las neuronas en la capa de entrada, en donde por medio de distintas funciones se procesan los datos. Posteriormente la información pasa por las capas ocultas en donde las funciones de activación determinan el estado de la neurona. Entre sus estados puede encontrarse activa, inactiva o parcialmente activa. Por último, la información resultante pasa a la capa de salida y esta determina la respuesta al problema planteado. Un esquema del proceso simplificado se muestra a continuación.

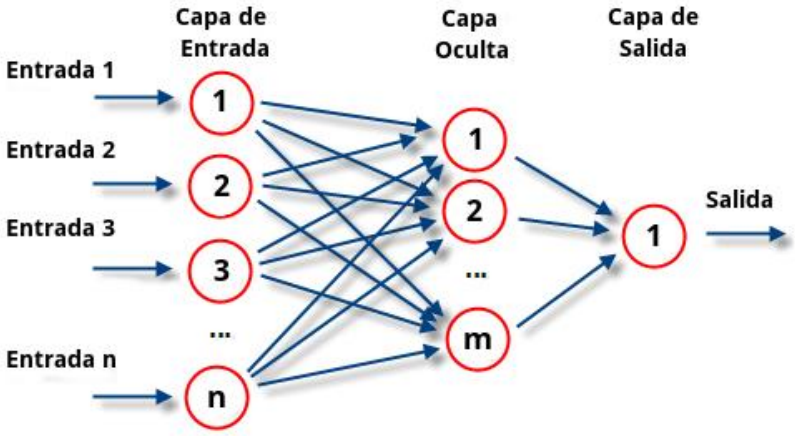


Figura 1: Estructura simple de red neuronal (Gengiskanhg, 2004)

Tipos de neuronas

La neurona artificial es una pequeña unidad de calculo que recibe un vector de entrada como estímulo para generar una respuesta. Los valores de entrada a la neurona artificial pueden provenir del exterior o ser la respuesta que genero otra neurona de la red. Existen diferentes tipos de clasificación de las neuronas artificiales, se pueden clasificar en función de la capa en la que se encuentran o por el tipo de entrada que reciben (Izaurieta & Saavedra, 2019).

Clasificación por capas

- Neuronas de entrada: Son las encargadas de recibir las señales del entorno, puede ser a través de sensores para comunicarse con el exterior o por las conexiones internas que las comunican con diversos segmentos del sistema.
- Neuronas ocultas: Son aquellas que solo reciben estímulos dentro del sistema y sus salidas se mantienen en el sistema, se encargan de procesar la mayor parte de la información de la red neuronal y no interactúan con el exterior.
- Neuronas de salida: Son aquellas que expulsan una salida al exterior del sistema después de que la información fue procesada.

Clasificación por los valores que puede adoptar la neurona

- Neurona binaria: Son aquellas que solamente pueden adquirir valores de 0 o 1 en una codificación y -1 o 1 en otra codificación.
- Neurona real: Son aquellas que adquieren valores dentro del rango [0,1] o [-1,1].

Capas

Las capas son agrupaciones de neuronas individuales que tienen características similares entre sí y que se encuentran conectadas a las neuronas de una capa anterior. Las redes neuronales pueden ser monocapa cuando solo existe una capa de neuronas que procesa la información y genera una salida correspondiente (Izaurieta & Saavedra, 2019). Este es el tipo de red neuronal más simple que existe. Las redes neuronales multicapa son aquellas que cuentan con una capa de entrada, una o varias capas ocultas y por último tienen una capa de salida.

La capa de entrada interactúa con la información que proviene de fuentes externas de la red neuronal. La capa de salida sirve para transferir la información

generada por la red neuronal hacia el exterior. Por último, se encuentran las capas ocultas que son internas de la red por lo tanto no tienen relación con el exterior. Las neuronas de estas capas pueden estar conectadas de distintas formas, por lo que dependiendo de la cantidad de capas ocultas y del tipo de capa que sean se determina la arquitectura o topología de la red neuronal.

Tipos de capas ocultas

Capa Densa: Es aquella donde todas las neuronas de la capa se encuentran conectadas con todas las neuronas de la capa anterior. Es una capa bastante efectiva para procesar información debido a que dispone de una mayor cantidad de datos para analizar. Su principal desventaja es la lentitud con la que procesa los datos por el volumen de información que maneja.

Capa Recurrente: Es aquella donde la salida en un cierto instante depende tanto de las entradas en ese instante como la salida que se generó con anterioridad. También tienen la capacidad de mantener un estado y su principal aplicación es el procesamiento de texto. Tiene dos variantes la primera se conoce como la Long Short-Term Memory (LSTM) en donde la información no solo es transmitida al paso previo sino también se transmite a varios pasos previos. La segunda se conoce como la Gated Recurrent Unit que funciona de igual forma que la LSTM, pero son más eficientes porque gastan menos recursos computacionales.

Capa convolucional: Es aquella en la cual se realizan operaciones de multiplicación y adición en los valores de la capa anterior, para lo cual ocupa un filtro que genera un plano de las características significativas de los datos que se están procesando. La aplicación más común de esta capa es en la visión artificial. Los beneficios que trae usar esta capa es reducir el sobreajuste y aumentar el sub-ajuste por

lo cual la precisión de la respuesta puede superar fácilmente el 90%. Como se observa en la imagen se tienen los datos de la capa de partida, estos pasan por el filtro convolucional para obtener las características más significativas. Una vez que se analiza toda la imagen se crea la capa convolucionada en donde están las características principales de los datos de la capa de partida.

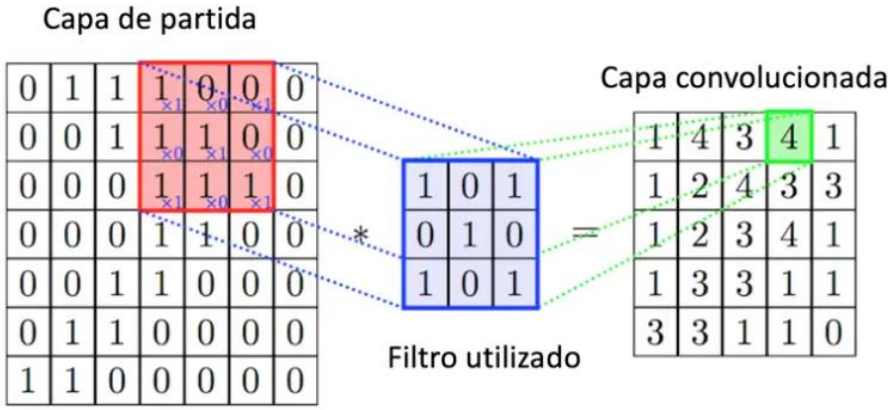


Figura 2: Funcionamiento de una capa convolucional (Calvo, 2022).

Capa de Reducción: Es aquella cuya principal función es reducir el tamaño de la capa anterior, de esa capa selecciona las características más importantes detectadas por los filtros aplicados previamente. Esta capa usa diversos tipos de filtros para reducir la muestra siendo el más común el max-pooling. Este filtro es una matriz que recorre los datos que entregó la capa previa, tomando el valor más alto que encuentre en ese recorte. Finalmente lo pasa a la salida haciendo que de esta forma la salida de la capa se reduzca en comparación con la capa anterior. Las matrices que se pueden usar son de 2x2 que reducen a la mitad el volumen de los datos que ingresaron a la capa, la de 3x3 reduce a una tercera parte el volumen de los datos de entrada y así sucesivamente. La información almacenada a la salida de la capa contiene las características más importantes del conjunto de datos de entrada. Como se observa en la imagen se tiene

una capa con 16 datos en una matriz 4x4, a esta capa se le aplica un filtro Max Pooled de 2x2 para reducir el tamaño de la matriz original, se divide a la primera matriz en 4 secciones y se toma el valor mayor y se pasa a la capa reducida que tiene un tamaño de 2x2.

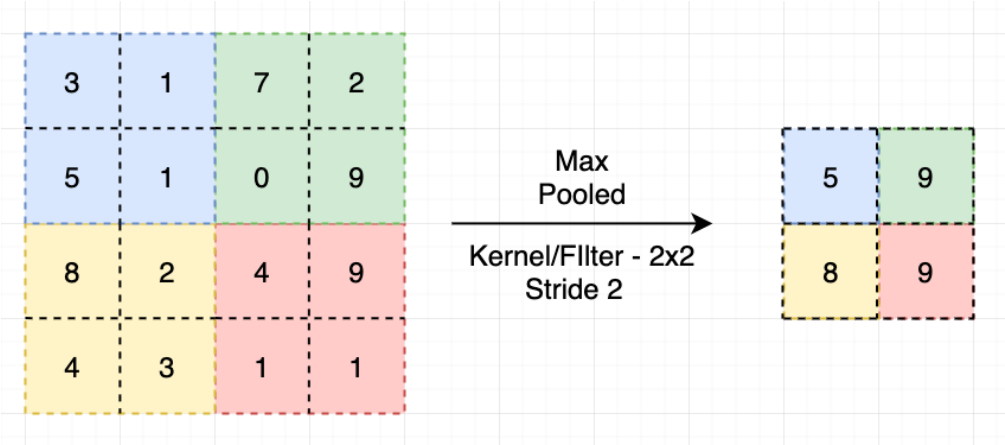


Figura 3: Funcionamiento del filtro Max Pooled (Rana, 2020).

Estructura de la neurona artificial

La neurona artificial se compone de tres elementos para poder procesar la información. A estos elementos se les llama funciones y se clasifican dependiendo de su uso. La primera es la función de entrada que agrupa los valores del vector de entrada en una entrada global. La segunda es la función de activación que determina si la neurona esta activada o desactivada. Por último, la función de salida determina el valor que va a ser compartido con las otras neuronas que se encuentran interconectadas.

En la función de entrada se agrupan los valores del vector de entrada ponderados con los pesos sinápticos. Esto quiere decir que cada valor del vector de entrada se multiplica con un coeficiente para determinar su importancia relativa en el tratamiento de la información. Para poder agrupar las entradas se utilizan varios métodos, uno de ellos es sumar todas las entradas ponderadas y de esta forma generar una entrada global. Otra forma es multiplicar todas las entradas ponderadas y de igual forma obtener una

entrada global. También se puede escoger la entrada que tenga el valor más alto y convertirla en la entrada global.

Las funciones de activación sirven para combinar las entradas con el estado actual de la neurona para de esta forma generar un nuevo estado de activación (Berzal, 2019). Esta función se aplica sobre la entrada global. Existen varios tipos de funciones de activación como se detallan a continuación:

- Función lineal: Se usa cuando se requiere que la salida sea una información analógica. Por lo general sus límites están entre 0 y 1 o entre -1 y 1. Se toma el valor mínimo cuando la entrada global no supera el valor umbral y toma el valor máximo cuando la entrada global está saturada.
- Función escalón: Determina el valor que toma la neurona en función del valor umbral. Si la entrada global sobrepasa el valor umbral esta tomará el valor de 1, y si la entrada global no alcanza el valor umbral esta tomará el valor de 0.
- Función sigmoide: Es usada para representar probabilidades, debido a su curvatura favorece a transformar los valores de entrada que se encuentran entre más y menos infinito en una salida que se encuentra en el rango de 0 a 1.
- Función tangente hiperbólica: Es bastante parecida a la función sigmoide con la única diferencia de que sus valores de salida se encuentran en el rango de -1 a 1.
- Función ReLU: Es la función más usada para las capas convolucionales y en las redes de clasificación. Su principal característica consiste en cambiar por 0 a los valores que están por debajo del umbral y mantener iguales a los valores que se encuentran por encima del umbral (Berzal, 2019).

En la función de salida se determina el valor que se entregará a las demás neuronas que se encuentran conectadas con esta neurona. Cuando la función de

activación se encuentra por debajo del umbral la salida es 0 y cuando sobrepasa el umbral la salida es 1. Cuando no existe función de salida, la entrada se mantiene después de pasar por la neurona a esta función se le llama identidad. Los valores de la salida suelen estar comprendidos en los rangos de 0 a 1 o de -1 a 1.

Entrenamiento de la red

Para el entrenamiento se utiliza el aprendizaje automático que consiste en que la computadora sea capaz de encontrar por su cuenta métodos para resolver el problema planteado, de cierta forma es un algoritmo que construye otro algoritmo. Se utiliza debido a que resulta más fácil que realizar el algoritmo que resuelva el problema directamente, ya que la programación tradicional tiene una dificultad elevada. Los tipos de aprendizaje automático que existen son:

Aprendizaje supervisado: Es un aprendizaje que utiliza ejemplos los cuales cuentan con la salida esperada del sistema. El objetivo del entrenamiento consiste en ajustar los pesos de tal manera que, ante una entrada diferente a los datos de ejemplo, se obtenga la salida deseada. Su principal objetivo es que el modelo generalice de forma correcta para siempre obtener la respuesta que se busca alcanzar.

Aprendizaje no supervisado: Este aprendizaje utiliza los datos de entrenamiento para construir hipótesis, descripciones y teorías. De esta forma se puede clasificar y agrupar los datos de entrenamiento, con el fin de encontrar patrones interesantes dentro de ellos que le ayuden a resolver el problema planteado (Matich, 2001).

Los elementos con los que cuenta el aprendizaje son: primero el número de lotes con el que trabaja en cada época, es decir el número de iteraciones que procesará para ajustar los pesos. Dependiendo de ese valor la red neuronal tardará más o menos tiempo en el entrenamiento. El algoritmo de backpropagation se encarga de observar como el cambio de los pesos afecta en la función de perdida y además busca mejorar las predicciones de la red

neuronal. El algoritmo de fowardpropagation es el que predice cual es la salida a la que pertenecen los datos de entrenamiento. La época determina cuantas veces se ejecutarán los algoritmos de backpropagation y de fowardpropagation para actualizar los pesos. La función de pérdida determina que tan buena es la red neuronal y mientras sea más bajo la red neuronal funcionará de mejor manera.

Computer visión

El computer visión (CV) es uno de los principales subcampos contenidos en machine learning. Esta disciplina también es conocida como visión artificial, su principal función es enseñar a las computadoras a ver y entender el contenido de las imágenes digitales. Actualmente en todo nuestro entorno nos encontramos con imágenes y videos. Desde teléfonos móviles hasta cámaras de seguridad, vemos toda clase de dispositivos que recopilan información digital todo el tiempo. Esta información puede llegar a ser muy útil si la almacenamos y analizamos detenidamente. Es por esto por lo que el propósito del computer visión es buscar métodos para la obtención, el procesamiento, análisis, y comprensión de las imágenes del mundo real. Todo esto con el fin de traducir la información visual a contenido numérico o simbólico que sea tratable por un ordenador. Las aplicaciones que encontramos son: reconocimiento de objetos, reconstrucción de una escena, restauración de imágenes, seguridad, robots con IA, carros autómatas, salud, ventas, etc. Para hacer uso de estas herramientas es importante utilizar un lenguaje de programación y librerías de libre uso que se encuentran en el internet. Los lenguajes que se utilizan con más frecuencia en sistemas de CV son Python y C++. La librería más amplia y utilizada para proyectos de Computer Vision es OpenCV.

Figura 4: Logo librería OpenCV (OpenCV, 2020).

OpenCV es una librería de software de visión, que contiene un conjunto de implementaciones funcionales codificadas en lenguaje de programación C++. Es decir que es un conjunto de funciones dedicadas a la visión artificial las cuales pueden ser invocadas en distintos lenguajes de programación como Python, Java, Octave, etc. La librería fue desarrollada originalmente por Intel, sin embargo, empresas como Google, Microsoft, IBM, Sony, etc. han hecho uso de esta para la creación de importantes proyectos tecnológicos. Al ser una librería de código abierto es accesible para el mundo entero y es compatible con todos los principales sistemas operativos. Actualmente cuenta con más de 2500 algoritmos enfocados a la visión del ordenador y aprendizaje automático (OpenCV, 2020). Es por estas razones que para este proyecto es esencial el uso de esta librería. A continuación, se presentan los equipos con los que se trabajó.

Jetson Nano

El equipo principal por utilizar es el Jetson Nano. Este equipo es un ordenador pequeño de la marca NVIDIA el cual permite desarrollar múltiples proyectos de IA. Su principal característica es que puede ejecutar múltiples redes neuronales en paralelo, lo que es útil para aplicaciones como: Clasificación de imágenes, detección de objetos y procesamiento de voz. El sistema trabaja con Linux por lo que es fácil de usar y su consumo es de 5 vatios. Cuenta con una RAM de 4GB y una ranura SD de almacenamiento. Para este trabajo se colocó una tarjeta SD de 64GB. El equipo se presenta a continuación.

Figura 5: Kit de desarrollo de Jetson Nano (NVIDIA, 2022).

Dofbot Yahboom

Adicionalmente en este proyecto se utiliza el brazo robótico DOFBOT de la marca china Yahboom. Esta empresa se enfoca en la educación por medio del desarrollo de distintos robots de inteligencia artificial programables. Además, la empresa provee de hardware de código abierto y kits educativos. Sus productos vienen con instrucciones para el ensamblaje del equipo y además con funciones precargadas para facilitar el entendimiento del equipo. A partir de aquí el equipo se puede programar según las aplicaciones que el usuario le quiera dar. En nuestro caso utilizaremos el modelo DOFBOT, el cual es un brazo robótico con cámara integrada. A continuación, se muestra una imagen del equipo.

Figura 6: Brazo robótico DOFTBOT (Yahboom, 2022).

El control principal del brazo es llevado a cabo por medio del ordenador Jetson Nano. La principal librería de procesamiento de imagen es Computer Vision la cual es de código abierto. El lenguaje principal de programación es Python3 y Jupyter Lab Mainstream como herramienta de desarrollo. El robot cuenta con 6 servos los cuales son motores que pueden ubicarse en cualquier ubicación dentro del rango. Este tipo de motores son controlables en velocidad y posición colocándose en dicha posición de manera estable. El resumen de algunas de sus características se expone en la siguiente imagen.

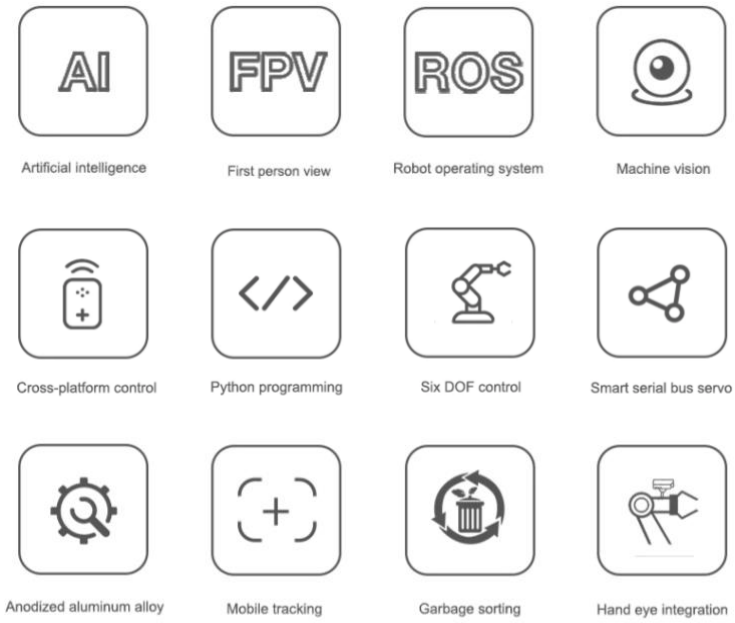


Figura 7*: Características de* DOFTBOT *(Yahboom, 2022).*

En base a todas estas características, el brazo robótico nos permite desarrollar el proyecto de reconocimiento de género y fusionarlo con el movimiento del brazo. Con toda la información previamente detallada, a continuación, se expone todo el proceso de desarrollo que nos llevó a obtener el producto final.

DESARROLLO PROTOTIPO N°1

Código de Entrenamiento

Modelo Caffe

Durante el proceso de investigación acerca de algoritmos de IA aplicados al reconocimiento facial, se encontraron varios tipos de modelos. El primero de ellos fue un conjunto de archivos provenientes del entorno Caffe. Caffe es uno de los primeros marcos de aprendizaje profundo de código abierto en 2013. Sus siglas significan 'Convolutional Architecture for Fast Feature Embedding' y fue desarrollado en la Universidad de Berkeley, California, por Yangqing Jia (Caffe, 2022). En este trabajo, inicialmente se buscó entender como funcionaban los modelos de aprendizaje, por lo que decidimos poner a prueba el modelo de Caffe. Para esto accedimos a la plataforma de Github que contenía un modelo pre-entrenado en Caffe titulado 'Age and Gender Classification using Convolutional Neural Networks' (Levi & Hassner, 2015). El modelo proviene de un artículo con el mismo título, sus autores son Gil Levi y Tal Hassner. Lo que podemos encontrar en el proyecto son los siguientes archivos:

- age_net.caffemodel
- gender_net.caffemodel
- gender_deploy.prototxt
- age_deploy.prototxt
- opencv_face_detector_uint8.pb
- opencv_face_detector.pbtx

En el caso de los archivos con la extensión '.caffemodel' son aquellos que contienen los pesos finales de las neuronas. Este archivo es el producto final luego del entrenamiento. En el caso de los archivos con la extensión '.prototxt' son aquellos que contienen: una lista de las capas de red neuronal que contiene el modelo, los parámetros de cada capa, su nombre, tipo, dimensiones de entrada, dimensiones de salida y especificaciones para las conexiones entre capas. Podemos notar que en este caso contamos con tres conjuntos el primero que se dedica al reconocimiento de edad, el segundo para el reconocimiento de género y por último el encargado de detectar un rostro. Con este modelo listo para ser utilizado, el siguiente paso es desarrollar el código que permite la comunicación con el robot y la Cámara.

Código de Aplicación

El código completo de aplicación del prototipo N°1, junto con los comandos que se describen a continuación se encuentran detallados en el Anexo A.

Computer Visión

Lo primero que se hace para trabajar con la librería de computer visión es importar la librería cv2 con el siguiente comando 'import cv2'. Una vez hecho esto podemos utilizar las funciones que nos permiten controlar la cámara. El primer paso es abrir la cámara y para esto se usa la función 'cv2.VideoCapture'. Posteriormente se necesita que la cámara detecte los rostros en tiempo real y coloque un marco en los mismos. Para lo cual utilizamos la función 'cv2.dnn.blobFromImage' que nos devuelve la imagen de entrada después de una resta media, normalización y el intercambio de canales. Esta función es típicamente usada para combatir los cambios de iluminación. Luego se utiliza la función 'cv2.rectangle' para crear el marco de color verde alrededor

del rostro. Adicionalmente, se utiliza la función ‘cv2.putText’ para colocar arriba del rostro el género y la edad de la persona. Por último, se utiliza la función ‘cv2.imshow’ para mostrar la imagen de la cámara en una ventana. Al momento de finalizar el programa es importante utilizar el comando ‘cv2.VideoCapture.release’ para liberar el recurso de la cámara y el comando ‘cv2.destroyAllWindows’ para cerrar todas las ventanas.

Predicción

Lo primero que hacemos es cargar el modelo con ayuda de la función ‘cv2.dnn.readNet’, esta función recibe como argumentos el ‘caffemodel’ y el ‘prototxt’. De esta forma ya estamos listos para comenzar a realizar las predicciones. Con el procesamiento de imagen realizado anteriormente, la imagen esta lista para entrar en nuestra red convolucional para lo cual usamos la función ‘net.setInput’ en donde especificamos que la entrada va a ser la imagen luego del procesamiento. Posteriormente usamos la función ‘net.forward’ para que ejecute el modelo y calcule una salida neta de tipo Numpy que nos entrega un vector con 2 números.

Cuando detecta un hombre el número mayor se posiciona en el índice 0, mientras que cuando detecta a una mujer el número mayor se posiciona en el índice 1. Esta particularidad nos sirve para seleccionar la etiqueta correspondiente usando la función ‘numpy.argmax’ que nos devuelve el índice donde se encuentra el número mayor del vector. Cuando el índice es 0 la etiqueta que despliega es la de hombre, mientras que cuando el índice es 1 la etiqueta que despliega es la de mujer. Lo mismo se hace con la edad, la diferencia es que la predicción entrega 8 números y de igual manera se extrae el índice del número mayor. Ese índice de igual forma corresponde a los 8 rangos de edad que determinamos con anterioridad.

Para hacer el movimiento, el robot debe realizar 8 predicciones y con el último valor de la predicción realizará el movimiento correspondiente para entregar el objeto como se detalla a continuación.

Dofbot Yahboom (servos y movimiento)

El control del brazo róbotico es empleado mediante el archivo 'Arm_Lib.py' y su función principal 'Arm_Device', proporcionados por el Dofbot. Este archivo se encuentra precargado en el jetson nano y contiene una serie de configuraciones que permiten su movimiento. Por lo que inicialmente se importa la función con el comando 'from Arm_Lib import Arm_Device'. El brazo se moverá en función de la respuesta de detección por lo que se debe definir una serie de posiciones en los 6 servos. Para hacer esto se necesitan de 3 comandos esenciales. El primer comando es 'self.Arm.Arm_serial_servo_write6' este nos permite cambiar los ángulos de los 6 servos y también controlar su velocidad. El rango de los servos va desde de 0 a 255 grados. El segundo comando es 'self.Arm.Arm_serial_servo_write' este nos permite elegir un servo, cambiar su ángulo y controlar su velocidad. El tercer comando es 'time.sleep' este sirve para otorgarle tiempo al brazo de llegar a una posición determinada antes de que se continue con la siguiente posición.

Se configuraron varios sets de posiciones y estos fueron almacenados en 3 grupos. El primero es el que enfoca a la persona para realizar la detección. El segundo se utiliza para entregar el cubo amarillo que se encuentra a la derecha del robot. El tercero se utiliza para entregar el cubo rojo que se encuentra a la izquierda del robot. Con la función 'action.start_action' se selecciona el grupo. Inmediatamente después del while se coloca al brazo en el grupo 1, cuando se realiza la detección de un hombre el

brazo realiza la acción del grupo 2 y cuando se detecta a una mujer el brazo realiza la acción del grupo 3.

Fase de Pruebas N°1

Con el modelo y el brazo robótico funcionando se procedió con las pruebas prácticas del prototipo N°1. Para esto se hizo la presentación del prototipo en la Casa abierta de la Universidad San Francisco de Quito.

Figura 8: Stand de pruebas, Casa abierta USFQ.

Como se puede observar en la figura 8 la organización del stand consistía en colocar el robot sobre una mesa de 80 cm de alto. Además, se ubicó una pantalla para que el usuario pueda observar la predicción en tiempo real. El experimento consistió en una breve explicación del funcionamiento y posteriormente se solicitó que los participantes se ubiquen de frente al robot a una distancia entre 50cm y un metro. En ese momento el robot realiza la predicción y entrega el cubo de color amarillo a los hombres y el rojo a las mujeres. Al final de la jornada se obtuvieron 100 muestras de personas. El gráfico de los resultados se muestra a continuación:

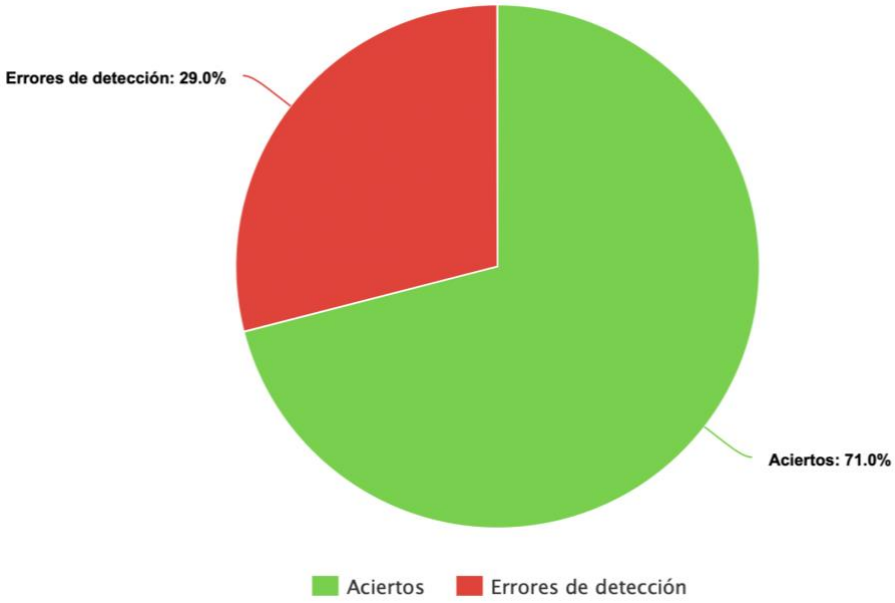


Figura 9: Pruebas Prototipo 1 – Detección de Genero.

Como se puede notar el prototipo 1 tiene un porcentaje de error del 29%, el cual es un valor considerablemente alto. Sin embargo, también hay factores que pudieron afectar en su rendimiento. Uno de ellos fue que durante las pruebas existía mucha carga visual en la cámara, por lo que en ocasiones habían más de dos personas en el rango visual. Esto resultaba ocasionalmente en el fallo de la predicción. Otro factor importante que se noto es la cantidad de iluminación. Es decir que cuando existía una correcta iluminación el modelo acertaba en sus predicciones. Por último, las personas que asistieron llevaban una mascarilla en sus rostros por lo que esto pudo afectar en las predicciones. Luego de experimentar con el modelo caffe e investigar la documentación pudimos notar lo siguiente:

- La documentación de caffe es muy escasa por lo que no se encontró una guía para el entrenamiento de modelos.
- Al ser uno de los primeros marcos de aprendizaje, actualmente su tecnología es obsoleta.

- Caffe utiliza un lenguaje de programación en C de alta complejidad.
- Para definir el modelo se requiere de un editor de texto lo que impide que exista modularidad.

Por estas razones se decidió utilizar tensorflow como la siguiente alternativa. Lo que llevo a la elaboración del segundo prototipo.

DESARROLLO PROTOTIPO N°2

Código de Entrenamiento

Tensorflow es una librería de código abierto cuyo propósito es la construcción y entrenamiento de redes neuronales. Esta librería fue desarrollada por el equipo de Google Brain y fue lanzada en 2015 (tensorflow, 2022). Tensorflow utiliza C++ y python como sus lenguajes de programación. Se puede utilizar en windows, macOS, linux y plataformas móviles Actualmente es el marco de aprendizaje profundo más utilizado debido a su amplia documentación. Al trabajar con Python se puede utilizar abstracciones de alto nivel, por lo tanto, el código es simple y limpio. Una ventaja importante es que sus aplicaciones se pueden ejecutar casi cualquier entorno ya sea la nube, en máquina local, en móviles, CPU y GPU. Adicionalmente cuenta con su propia unidad de procesamiento llamada TPU, la cual se encuentra optimizada para Tensorflow y se puede acceder a esta mediante la nube.

El programa que se utilizó para escribir el código fue Google Colaboratory. Este es un entorno el cual permite editar un cuaderno de jupyter. Jupyter es la aplicación que permite escribir y ejecutar código en Python (Google Colaboratory, 2022). Google Colab combina esta función con la de un editor de texto dando la posibilidad de escribir anotaciones y correr código en python a la vez. Todo esto en la nube por medio de la página web. Esto quiere decir que no requiere de ningún tipo de instalación. Los archivos de este programa tienen la extensión

‘.ipynb’ el cual significa cuaderno de python. Al finalizar el entrenamiento exportaremos el modelo a un archivo de extensión ‘.h5’ que contendrá todos los datos y pesos del aprendizaje.

A partir de Google Colab y Tensorflow se desarrolló el código de entrenamiento de reconocimiento de genero el cual se encuentra completo en el anexo B. A continuación, se describen los comandos y líneas de código que hicieron posible la obtención del modelo.

Librerías y Base de Datos

Al iniciar el código de entrenamiento es necesario importar las siguientes librerías y funciones:

- from tensorflow.keras.preprocessing.image import ImageDataGenerator
- from tensorflow.keras.optimizers import Adam
- from tensorflow.keras.preprocessing.image import img_to_array
- from tensorflow.keras.utils import to_categorical, plot_model
- from tensorflow.keras.models import Sequential
- from tensorflow.keras.layers import BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense
- from tensorflow.keras import backend as K
- from sklearn.model_selection import train_test_split
- import matplotlib.pyplot as plt
- import numpy as np
- import random
- import cv2
- import os
- import glob

Estas librerías nos permitirán utilizar todas las funciones de tensorflow, keras, procesamiento de imagen, manejo de arreglos y variables, etc. Las funciones de cada librería se encuentran descritas en los siguientes apartados. En cuanto a la base de datos se utilizó la base de 2909 imágenes proporcionadas en kaggle por el usuario SteveGrant38 (Grant, 2022). Esta base contiene imágenes de retratos tanto de hombres y mujeres en dos carpetas separadas.

Procesamiento de Imagen

Lo primero que se hace es determinar las dimensiones y los canales de las imágenes del entrenamiento. Para este modelo las dimensiones fueron de 96x96 y a color por lo que se usaron los 3 canales RGB. Después se lee la dirección de los archivos del set de datos que se encuentran en la carpeta de la siguiente dirección '/content/drive/MyDrive/GD/gender_dataset_face'. Dentro de esa carpeta se encuentran las carpetas de man-woman con 1529 y 1380 imágenes respectivamente. A continuación, se mezclan las direcciones con la función 'random.shuffle(image_files)', esto se hace con el objetivo de que las imágenes de hombres y mujeres se presenten de forma intercalada. Para poder leer las imágenes utilizamos un for el cual hace un barrido por cada dirección. Con el comando 'cv2.imread' se lee la imagen de cada dirección y se guarda en una variable temporal. Luego se utiliza la función 'cv2.ctvColor' para cambiar el orden de los canales de BGR a RGB, esto nos ayuda a imprimir la imagen en el orden de colores correcto. Con el comando 'cv2.resize' se ajustan todas las imágenes a la dimensión de 96x96. Para el procesamiento de imágenes es necesario utilizar un arreglo tipo Numpy por lo que por utilizamos de la función 'img_to_array' que crea un arreglo 3D de tipo Numpy. Todos los arreglos se guardan en el vector data, con ayuda de la función 'append' que inserta el contenido al final del vector. Para las etiquetas se creó el vector labels y con ayuda de la dirección se verifica de que carpeta pertenece la imagen. Si esta era de la carpeta woman, la variable temporal label tomaría el valor de 1 caso contrario tomaría el valor de 0. Por medio de la función 'append' se guardarían todas las etiquetas generadas en el vector labels.

Se normaliza a los valores del arreglo dividiendo para 255 a cada valor. El número resultante es de tipo flotante y estará en el rango de 0 a 1. Se divide a los datos

en un set para el entrenamiento y un set para validación mediante la función 'train_test_split' que divide a las imágenes en 'trainX' y 'testX'. Por otro lado, divide a las etiquetas en 'trainY' y 'testY'. Dentro de esta función se puede especificar el porcentaje de imágenes que pasan al set de validación en nuestro caso sería del 20%.

Para aumentar la eficiencia del algoritmo se realiza el aumento de datos utilizando la función 'ImageDataGenerator' la cual crea aleatoriamente transformaciones a las imágenes del set de entrenamiento. Las transformaciones que se hicieron fueron las siguientes: rotaciones a un ángulo de 25°, desplazamientos de la imagen a lo ancho en un rango de 0.1, desplazamientos de la imagen a lo alto en un rango de 0.1, recortar porciones de la imagen en un rango de 0.2, zoom a un rango de 0.2, volteos horizontales, rellenar los nuevos pixeles usando el pixel más cercano, recorte o desplazamiento.

Todas estas modificaciones aleatorias permiten tener una variedad más amplia de imágenes. El aumento de datos evita que el algoritmo se aprenda de memoria las imágenes del set de entrenamiento. En otras palabras, existe un sobreajuste y en las pruebas de validación un mínimo cambio en la imagen presentada, podría resultar en una predicción errónea. El aumento de datos aporta flexibilidad en la predicción, mejora la eficacia y reduce las pérdidas.

Arquitectura redes neuronales

Luego de organizar y procesar los datos, el siguiente paso es definir la arquitectura que se utilizara para la red. Lo primero que hacemos es crear una función para construir el modelo por medio del comando 'def' siendo los argumentos de entrada: el ancho, el largo, los canales de la imagen y también se especifica cual es el número de clases que tiene la salida. Se especifica que el modelo será del tipo secuencial

con el comando ‘sequential()’ esto quiere decir que la capa 1 será la entrada de la capa 2 que a su vez será la entrada de la capa 3 y así sucesivamente. Luego se crea una variable que tiene las especificaciones de los datos de la forma de las imágenes que provienen de los argumentos de entrada de la función. Dependiendo del formato que tenga la imagen ubicando primero los canales o al final se guarda en las variables correspondientes.

Se empiezan a definir las capas empezando por la convolucional la cual usa 32 filtros con un tamaño de 3x3 y mantiene el mismo volumen de entrada en la salida especificando que el parámetro ‘padding’ tendrá el valor de ‘same’, se utilizará la función de activación ‘relu’ que elimina los números negativos y los reemplaza por 0 después de aplicar la convolución. Luego utilizamos el comando ‘BatchNormalization’ que hace una normalización a los datos de las capas ocultas para que sean valores de entre 0 y 1, de tal manera que a la red neuronal pueda entrenar más fácilmente al tener un rango más pequeño en los valores que puede predecir y hace que su línea de decisión sea más compacta. Posteriormente, realizamos la reducción con la capa ‘MaxPooling2D’ la cual utilizará una matriz 3x3 para escoger el valor más alto dentro de esta ventana y pasarla a una nueva imagen de tamaño reducido. Para finalizar esta etapa se utiliza el comando ‘Dropout’ para desactivar una porción de las neuronas para evitar el sobreajuste ya que las neuronas dependerán menos de las neuronas aledañas y serán más eficientes por sí mismas, el valor de dropout que se usó en este modelo fue de 0.25 lo cual quiere decir que el 25% de las neuronas totales de esa capa podrían ser desactivadas.

En la siguiente capa de convolución se incrementa el número de filtros al doble debido a que la imagen de entrada es más pequeña, el resto de los parámetros que son el tamaño del filtro, el padding y la función de activación se mantienen, también se

vuelve a realizar un batchnormalization para mantener los valores en un rango de 0 a 1. Se repite nuevamente la capa de convolución con 64 filtros con normalización incluida y luego de eso se aplica la reducción, pero esta vez la matriz es de 2x2, lo que de igual manera hace que la imagen tenga un tamaño aún más reducido. Por último, se mantiene el dropout del 25% para conseguir una mejor precisión luego del entrenamiento. Este proceso se repite una vez más aumentando el número de filtros esta vez a 128. Estas capas sirven para agrupar características poco a poco, primero detectando bordes, luego detectando figuras y así sucesivamente va detectando patrones más complejos.

Después de pasar por todas las capas de convolución y reducción, se utiliza el comando ‘flatten’ para poder reducir las dimensiones del arreglo numpy a una sola dimensión con el objetivo de procesar los datos de manera más ágil. El siguiente paso es definir una capa densa con 1024 neuronas con el comando ‘Dense’, esta capa tiene a todas sus neuronas interconectadas y su función de activación es la ‘relu’, volvemos a hacer la normalización del batch y por último definimos el dropout, pero esta vez lo aumentamos al 50%. Definimos la última capa densa la cual nos entrega nuestra salida de 2 clases, siendo 0 para hombres y 1 para mujeres, la función de activación que usa esta capa es la de sigmoid que tiene como respuesta valores entre 0 y 1. De esta forma el algoritmo predecirá si la imagen de entrada es de un hombre o de una mujer.

Creamos el modelo usando la función build que creamos con anterioridad y como entrada ponemos los valores de la imagen: el ancho de 96 pixeles, el alto de 96 pixeles y es de 3 canales RGB, y a la salida tiene 2 clases que son Hombre y Mujer.

Entrenamiento de la red

Para comenzar con el entrenamiento del modelo primero creamos un optimizador para el modelo utilizando el comando ‘Adam’ el cual sirve para actualizar los pesos de la red de forma iterativa en función de los datos de entrenamiento, en esta función se define el learning rate el cual nos indica que tanto se actualizarán los pesos en cada iteración, el valor típicamente usado en tensorflow es de 0.001 y es el mismo que estamos usando. La tasa de decaimiento es igual al learning rate dividido para el número de épocas, esto sirve para reducir la tasa de aprendizaje a medida que avance en el número de épocas. Compilamos nuestro modelo, lo cual básicamente es configurar el proceso de aprendizaje para esto usamos el comando ‘compile’ y los parámetros que configuramos son la función de pérdida (loss) que nos sirve para observar cuanto es el porcentaje de fallas en las predicciones del algoritmo, el optimizador que en nuestro caso es el Adam y por último la precisión (accurracy) que nos indica cuál es el porcentaje de aciertos en las predicciones del algoritmo.

Por último, realizamos el entrenamiento como tal utilizando la función ‘fit’ en la cual especificamos la matriz de datos de entrenamiento utilizando el aumento de datos, también especificamos la matriz de los datos de validación, configuramos el batch_size que es del tamaño del vector trainX en este caso es de 28 y por último definimos el número de épocas que es igual a 100 en nuestro modelo. Se ejecuta y se debe esperar alrededor de 2 horas a que se entrene, a medida que termina cada época utiliza los datos de validación para calcular la pérdida y la precisión y se puede ver como estos valores van disminuyendo y aumentando respectivamente, además de que los valores de pérdida y precisión del entrenamiento son bastante similares a los de validación por lo que nuestro modelo no se encuentra sobre ajustado y debería tener un

buen funcionamiento en el mundo real. Al final del entrenamiento obtuvimos una pérdida del 10% y una precisión del 90%, lo cual muestra que el modelo funciona bastante bien.

Código de Aplicación

El código completo de aplicación del prototipo N°2, junto con los comandos y funciones descritas a continuación, se encuentran en el anexo C.

Requisitos Jetson Nano

Es importante mencionar que para que la implementación de este modelo de tensorflow es necesario un conjunto de requisitos en el Jetson Nano. Uno de los principales requerimientos es la versión del software, la cual debe ser la 'Jetpack 4.6.1'. En conjunto con esto debe contar con las versiones de 'CUDA 10.2' y 'Cudnn 8.2.1'. El sistema operativo debe estar en Ubuntu 18.04 LTS. Puesto que obtener estas versiones conlleva formatear el disco es necesario reinstalar las siguientes instancias:

- Tensor Flow en su versión 2.5.0
- Cvlib y Cv2 en sus últimas versiones.
- ROS para el control del robot
- Smbus que permite controlar los servos.
- Adafruit que permite la utilización de la pantalla led.

Un factor esencial para tener en cuenta es la memoria RAM con la que cuenta el Jetson nano. Esto determina su capacidad para correr modelos complejos. Mientras más robusto y complejo sea el modelo requerirá de más recursos, principalmente RAM. Sin embargo, existe una modificación que nos permite crear memoria virtual y así poder correr los modelos que requieran de mayor RAM. Esto se hace por medio de la instalación de un Swap file. Esto crea memoria virtual en la que se almacenan páginas de la memoria que se encuentran inactivas, liberando así la memoria física requerida

para procesos activos (Ubuntu, 2022). Una vez contemos con todos estos requisitos, estaremos listos para la implementación de un modelo de tensorflow con las características descritas previamente.

Computer Visión

Se hizo en base al prototipo anterior, pero se realizaron los siguientes cambios:

- Se añadió la librería cvlib para poder detectar los rostros con la función 'cvlib.detect_face' de esta forma ya no es necesario cargar un modelo específico para detectar la presencia de un rostro.
- Usando la detección se encuentran las esquinas para crear el rectángulo alrededor del rostro con la función 'cv2.rectangle'. Se recorta el rostro y se lo almacena en una nueva variable 'face_crop'.
- Si el cuadro del rostro es muy pequeño ya que la persona se encuentra muy alejada, este dato es ignorado y no pasa a la predicción hasta que el cuadro tenga un tamaño más grande.
- Se realiza un preprocesamiento al cuadro del rostro antes de hacer la predicción, primero se ajusta el tamaño a 96x96, después se normaliza a la imagen dividiendo los 3 canales para 255 y por último se pasa la imagen a un arreglo Numpy.
- Se despliega el género de la predicción realizada acompañado del porcentaje de confiabilidad de la predicción.

Predicción

En este caso al ser un modelo del tipo keras se necesita una función diferente para cargar el modelo esta se llama 'load_model' y recibe un archivo con extensión

‘.h5’. En este se encuentran los pesos de las neuronas luego del entrenamiento y la arquitectura desarrollada en el apartado anterior, a diferencia del modelo en caffe en donde los pesos y la arquitectura se encontraban en 2 archivos separados. Para predecir utilizamos la función ‘model.predict’ y el argumento de entrada es el rostro recortado. De la misma manera esta nos devuelve una salida neta de tipo Numpy. Esta salida está compuesta por dos números que son la confiabilidad de la predicción. Para esto se selecciona el índice del número mayor con la función ‘np.argmax’, con el que determina el género al que pertenece la persona que está frente a la cámara. El 0 pertenece a un hombre y el 1 para la mujer. Para obtener el porcentaje de confiabilidad se multiplica al número mayor de la predicción por 100.

Dofbot Yahboom (servos y movimiento)

Se mantuvieron los mismos movimientos que en el prototipo anterior siendo el 1 para ponerse en posición para detectar un rostro, el 2 para entregar el objeto cuando el algoritmo predice que es un hombre y el 3 para entregar el objeto cuando el algoritmo predice que es mujer. De igual manera se deben realizar 8 predicciones antes de realizar el movimiento y utilizará la última predicción para entregar el objeto escogido por el algoritmo. La única diferencia es que se dividieron los archivos que tenían el control de los movimientos con el de aplicación. Esto se hizo con el objetivo de tener una mejor organización y evitar un código de aplicación extenso. Para poder usar los dos archivos es necesario que se encuentren en la misma carpeta y se llama al archivo ‘action_group.py’ con la función ‘import’.

Fase de Pruebas N°2

Con el modelo y el brazo robótico funcionando se iniciaron las pruebas prácticas del prototipo N°2. Para esto se realizaron pruebas a personas de manera aleatoria en Universidad San Francisco de Quito.

Figura 10: Stand de pruebas prototipo 2, USFQ.

Como se puede observar en la figura 10 la organización del stand consistía en colocar el robot sobre una mesa de 80 cm de alto. Además, se ubicaron dos pantallas para que el usuario pueda observar la predicción en tiempo real y las estadísticas al momento. El experimento consistió en una breve explicación del funcionamiento, posteriormente se solicitó a los participantes que se ubiquen en la silla frente al robot. Esta silla se ubicó a una distancia entre 50cm y un metro. En ese momento el robot realiza la predicción y entrega el cubo de color amarillo a los hombres y el rojo a las mujeres. Al final de la jornada se obtuvieron 100 muestras de personas. El grafico de los resultados se muestra a continuación:

Se puso a prueba el funcionamiento del modelo, obteniendo al final de la jornada 100 muestras de personas. El gráfico de los resultados se muestra a continuación:

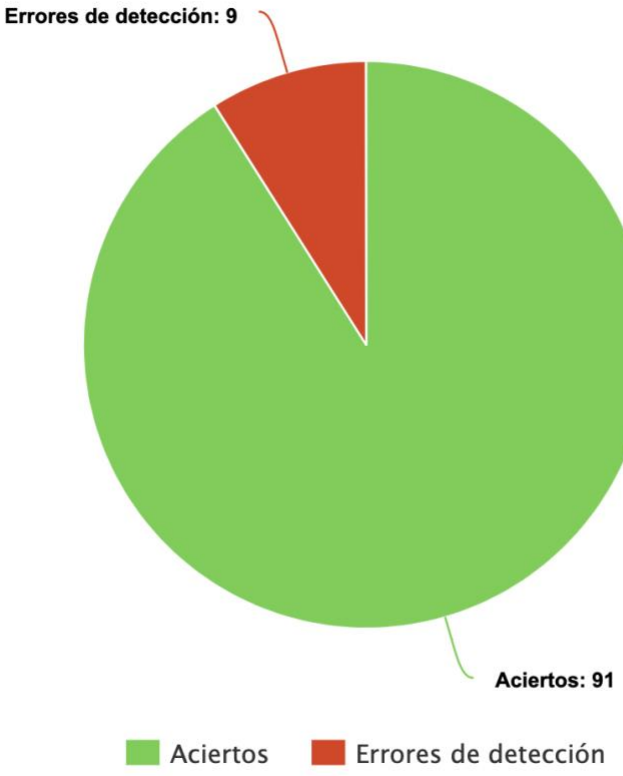


Figura 11:Pruebas Prototipo 2 – Detección de Genero.

Como se observa en el gráfico el robot logro detectar el género satisfactoriamente en 91 personas, siendo así que tuvo un 9% de errores de detección. Tras estos resultados se determinó que el algoritmo aún puede mejorar y reducir el margen de error. Por lo que esto llevo a el desarrollo del modelo final.

DESARROLLO MODELO FINAL

Código de Entrenamiento

El código completo de entrenamiento del prototipo final, junto con los comandos que se describen a continuación se encuentran detallados en el Anexo D.

Librerías y Base de Datos

Las librerías que se utilizaron son las mismas que en el prototipo previo. Sin embargo, aumentamos los datos de imágenes por medio de una base de datos en kaggle. Para utilizar las bases de datos ubicadas en kaggle directamente es necesaria la instalación de su librería. Esto se hace mediante el comando 'pip install kaggle'. Adicionalmente es necesario crearse una cuenta en kaggle la cual te proporciona un archivo de nombre 'kaggle.json'. Este archivo contiene las credenciales y permisos para ocupar los datasets de la comunidad. La base de datos que se utilizo fue proporcionada por Jessica Li y es titulada "Gender Classification 200K Images | CelebA" (Ashish, 2020). La base de datos contiene alrededor de 200 mil imágenes. Sin embargo, se utilizaron únicamente 50000 debido a los recursos limitados de RAM.

Procesamiento de Imagen

Se realiza igual que en el prototipo anterior con algunas modificaciones:

- El tamaño de la imagen cambio de 96x96 a 100x100 esto con el objetivo de tener más datos para tener un mejor rendimiento del algoritmo.
- Se utilizó un set de datos de kaggle con 50K imágenes para tener más variedad en los datos de entrenamiento y disminuir las fallas de predicción del algoritmo.
- Se realizó un cambio en el orden de los canales de BGR a RGB con la función 'cv2.COLOR_BGR2RGB' para poder imprimir adecuadamente las imágenes.

Arquitectura redes neuronales

La arquitectura se mantuvo exactamente igual que en el prototipo anterior ya que demostró ser bastante efectiva en el reconocimiento de género. No fue necesario modificar el número de neuronas por capa o la cantidad de capas ocultas que ya definimos con anterioridad.

Entrenamiento de la red

Se realizó el entrenamiento con los mismos parámetros del prototipo anterior manteniendo la tasa de aprendizaje, el decaimiento, el optimizador y las demás configuraciones. Lo que si cambio fue el batch, previamente mantenía un valor de 28 pero debido a que hay más datos el vector trainX creció. Es por esto que ahora el tamaño del batch es de 625. Se observo que la precisión en el entrenamiento y la validación aumentaba a la par. De igual manera la pérdida en el entrenamiento y la validación disminuía conjuntamente. Al final del entrenamiento se obtuvo una precisión del 96.5% y una pérdida del 3.5% lo cual demuestra que nuestro algoritmo trabaja de forma excepcional y que está listo para ser probado.

Código de Aplicación

El código completo de aplicación del prototipo final, junto con los comandos y funciones descritas a continuación, se encuentran en el anexo E.

Computer Visión

No hubo modificaciones significativas con respecto a la aplicación del prototipo anterior, lo único que se modificó fue el tamaño del recorte del rostro que paso de 96x96 a 100x100 para ajustarse a las características del modelo desarrollado en el apartado anterior.

Predicción

Se realiza de la misma forma que en el prototipo anterior la diferencia es el archivo del modelo que se va a usar en este caso el nombre del archivo es ‘genderD.h5’ este se carga en lugar del anterior archivo. Se efectuarán los mismos pasos para determinar si la persona al frente de la cámara es un hombre o una mujer.

Dofbot Yahboom (servos y movimiento)

Durante las pruebas anteriores se pudo notar que un ligero movimiento de la persona durante la predicción podía resultar en un fallo. Esto debido a que solo tomaba el último valor de la predicción. Por este motivo se implementa un promedio donde por mayoría simple se realiza el movimiento de acuerdo con el género que tiene 5 o más predicciones. Se necesitan 9 predicciones en lugar de 8 para realizar el movimiento, esto para eliminar la posibilidad de empate. Se utilizó un contador que aumenta su valor cada vez que se detecta a un hombre y por medio de un comparador se determina el género correspondiente. Si el contador es igual o mayor a 5 se le considera hombre, y si el contador no llega a 5 se le considera mujer. Luego de realizar el movimiento el contador se reinicia. Tras estos cambios se presenció un error en donde el contador de predicciones y de genero se disparaba hacia el infinito. Por lo tanto, no llegaba a un resultado y el robot no presentaba movimiento. Este error se solucionó con un reinicio de esos contadores en el momento que excedían el número 9. De igual forma se mantuvieron los 3 movimientos con los mismos propósitos que en el prototipo anterior, la diferencia fue el objeto que entregaba paso de ser un cubo de distintos colores a dos paletas de sabores distintos. La paleta de color amarillo se entrega cuando se reconoce a un hombre y la paleta roja se entrega cuando se reconoce a una mujer.

Fase de Pruebas Final

Con el modelo y el brazo robótico funcionando se iniciaron las pruebas prácticas del prototipo Final. Para esto se realizaron pruebas a personas de manera aleatoria en Universidad San Francisco de Quito.

Figura 12: Stand de pruebas prototipo Final, USFQ.

Como se puede observar en la figura 12 la organización del stand consistía en colocar el robot sobre una mesa de 80 cm de alto. Además, se ubicó una pantalla para que el usuario pueda observar la predicción en tiempo real. Se ubicó una silla a una distancia entre 50cm y un metro. En esta ocasión la silla fue ubicada estratégicamente para evitar problemas de iluminación. El experimento consistió en una breve explicación del funcionamiento, posteriormente se solicitó a los participantes que se ubiquen en la silla con la vista dirigida hacia al robot. Además, los participantes debían presentarse sin ningún accesorio como: lentes, gorras, gafas, cubrebocas, etc. Esto con el fin de eliminar posibles distractores. A continuación, el robot realiza la predicción y entrega una paleta de color amarillo a los hombres y una paleta roja a las mujeres. Al final de la jornada se obtuvieron 110 muestras de personas. En esta

ocasión se procuró tomar el mismo número de medidas tanto en mujeres como en hombres, obteniendo 55 muestras para cada género. El gráfico de los resultados se muestra a continuación:

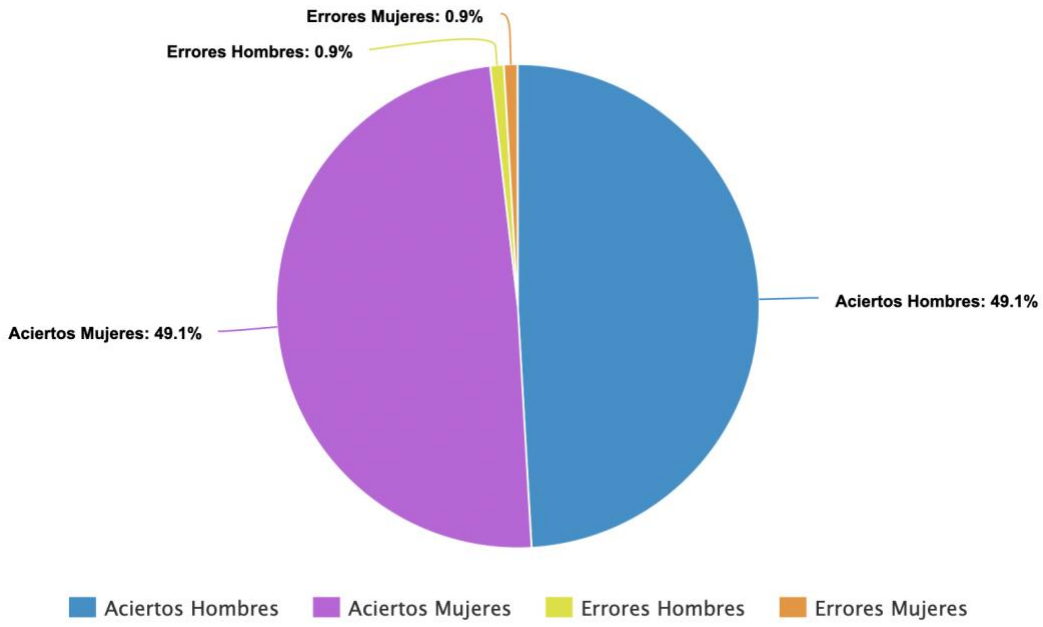


Figura 13: Pruebas Prototipo Final – Detección de Genero.

Como se puede notar en este caso dividimos las muestras de manera equitativa para observar si existía una dependencia del género en los aciertos de detección. Sin embargo, los resultados fueron homogéneos y se presentó 1 error de detección en cada género, dando un total de 1.8% de error.

RESULTADOS Y DISCUSIÓN

En esta sección se evalúa y compara la efectividad que tuvo cada uno de los modelos propuestos. En el caso del primer modelo se extrajeron los datos teóricos de precisión y pérdida expuestos en artículo ‘Age and Gender Classification using Convolutional Neural Networks’ (Levi & Hassner, 2015). Estos datos son comparados con los resultantes del experimento 1 y se exponen en la tabla 1.

Los valores teóricos, tanto del prototipo 2 como del prototipo final, se obtuvieron de la última época de entrenamiento respectivamente. Esto gracias a que, durante el entrenamiento, el programa imprime los valores de precisión y pérdida para cada una de las épocas. En la siguiente tabla se comparan los porcentajes de precisión y perdidas de los tres prototipos. Además, se presenta el resultado del cálculo del error relativo.

Tabla 1:Error relativo de cada prototipo.

	Prototipo 1 gender_net.caffemodel		Prototipo 2 gender_detection.h5		Modelo Final genderD.h5	
	Precisión	Pérdida	Precisión	Pérdida	Precisión	Pérdida
Teórico	77.8%	22.2%	90%	10%	96.5%	3.5%
Práctico	71%	29%	91%	9%	98.2%	1.8%
Error	8%	30.63%	1.11%	10%	1.76%	48.57%

Como se puede observar la precisión del primer modelo es bastante baja desde el entrenamiento y en el experimento esta se redujo aún más. Por este motivo este modelo quedo totalmente descartado. Sin embargo, en esta etapa el desarrollo del movimiento del brazo robótico resulto de manera exitosa. Es importante mencionar que existieron varias fallas durante el experimento que pudieron ser ocasionadas por la excesiva contaminación visual en el margen de la cámara. Otro factor para tener en cuenta son las mascarillas, puesto que el robot tendía a fallar en personas que utilizaban una. A pesar de esto se procuró utilizar únicamente

las mediciones consideradas validas. Es decir, casos en los cuales la persona se situaba de frente y que se presentaba la mínima existencia de contaminación visual en la cámara.

En el modelo N°2 se observan datos interesantes como la precisión, la cual es mayor en la práctica que en la teoría. Eso nos indica que el modelo trabajó mejor con los datos del experimento que con los datos de entrenamiento. En este caso observamos que la luz juega un papel importante ya que cuando pusimos la cámara a contraluz fue cuando se generó la mayor cantidad de errores de detección. Por el contrario, cuando hubo un buen enfoque e iluminación la detección mejoro notablemente. Se observa de igual forma que los valores prácticos y teóricos son similares y por eso se obtienen errores relativos bajos.

En el modelo final, de igual forma que en el anterior, funciono mejor de lo que se esperaba ya que el experimento presento un porcentaje de precisión mayor al del teórico. En esta ocasión se trabajó con la luz adecuada y se buscó que la cantidad de hombres y mujeres este balanceada. Esto en función de verificar si existía una tendencia en el género al momento de fallar en la predicción. No obstante, esto no sucedió ya que los errores se presentaron de manera equitativa en cada género. A pesar de aumentar el número de muestras de prueba el error disminuyó, razón por la cual este algoritmo es el que mejores resultados obtuvo.

De manera general podemos observar el progreso en la precisión de un modelo a otro en el siguiente grafico de barras:

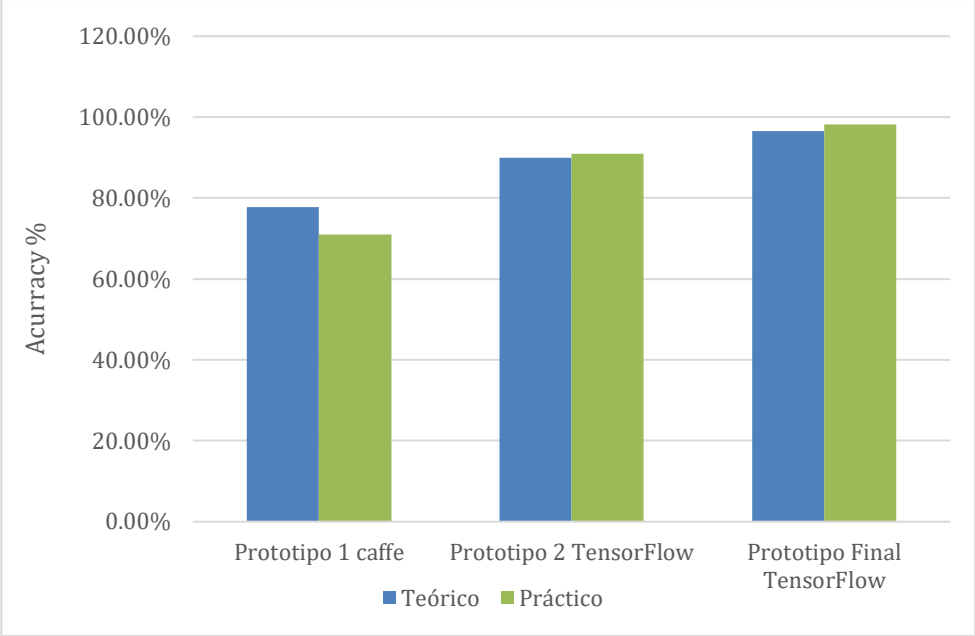


Figura 14: Comparación de prototipos.

Por último, gracias a la herramienta Matplot se graficaron las curvas de precisión y pérdida a lo largo de cada una de las épocas. La gráfica del prototipo 2 se muestra a continuación:

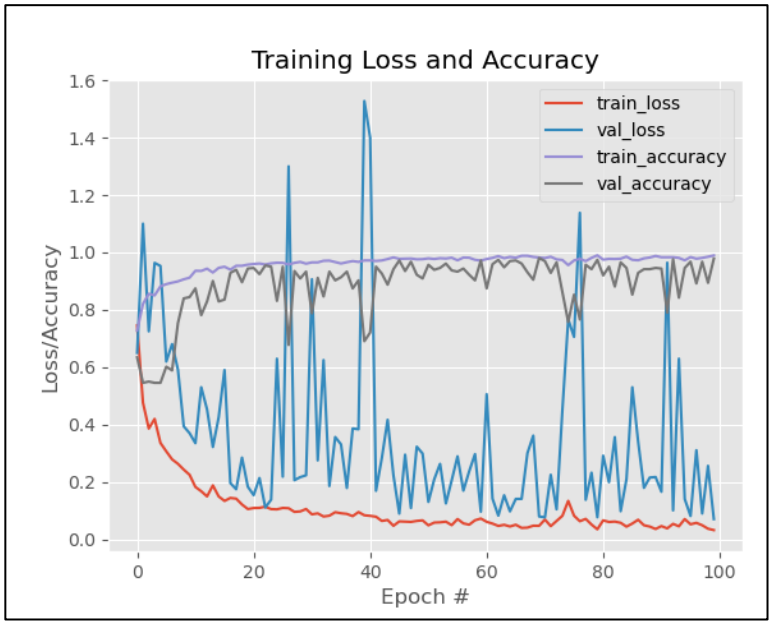


Figura 15: Precisión/Pérdidas VS Época Prototipo 2.

Para el último modelo de igual forma se extrajeron los datos de precisión y perdida. La gráfica del entrenamiento para el prototipo final se observa a continuación:

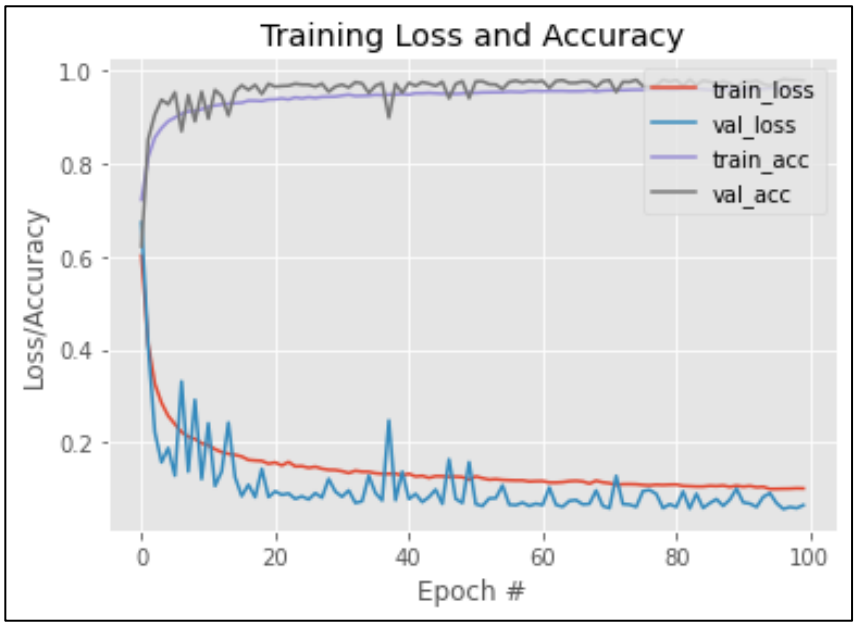


Figura 16:Precisión/Pérdidas VS Época Prototipo Final.

En las figuras 12 y 13 podemos notar que existen cuatro curvas. Las curvas superiores están relacionadas con la precisión en el entrenamiento y la precisión en la validación. La curva de validación es una curva en la cual el algoritmo pone a prueba su funcionamiento simulando una fase de pruebas. Mientras la curva tienda a uno y se mantenga estable significa que el modelo funciona de manera correcta. Este es el caso de la figura correspondiente a el prototipo final. Por otro lado, tenemos las curvas inferiores que de igual forma representan las pérdidas durante el entrenamiento y en una de validación. Así mismo se busca que las pérdidas tiendan a cero y se mantengan estables. En la figura 12 podemos notar que curva de pérdidas en la validación está muy inestable. Esto puede resultar en la pérdida de precisión al momento de realizar el experimento. Por lo contrario, en la figura 13 la curva de pérdidas se encuentra variable, pero en un rango reducido y conforme avanzan las épocas se va estabilizando.

APLICACIONES

Actualmente podemos encontrar el uso de la inteligencia artificial en los softwares de las computadoras. Principalmente en charlas virtuales debido al procesamiento de lenguaje natural. También en asistentes de voz con los que podemos controlar entornos manejados por el internet de las cosas. Sin embargo, el uso de robots físicos sigue en pleno desarrollo y no existe una variedad amplia aún. Es por esto por lo que este es un tema de investigación que puede escalar con el tiempo. El uso de la inteligencia artificial debe ir acompañado con del desarrollo del hardware. De esta manera los humanos podremos vernos beneficiados por sistemas que no requieran de supervisión.

El prototipo de este proyecto puede ser utilizado como punto de partida para diseñar varios productos. Uno de ellos puede ser la utilización del robot en una tienda la cual pretenda brindar un servicio al cliente personalizado. Para esto el reconocimiento de género y de edad puede ser un factor importante. Además, el entrenamiento de redes neuronales no sirve únicamente para el reconocimiento de género, sino que también puede servir para distinción de objetos, distinción de especies, etc. Esto nos lleva a la idea de un segundo ejemplo de producto que utiliza la IA. Se podría diseñar un dispensador de comida automático para personas que cuenten con un perro y un gato como mascotas. El dispensador puede contar con una cámara la cual reconoce el animal que se acerque y podrá dispensar la comida correspondiente.

El desarrollo de estos prototipos puede derivar en productos innovadores que ayuden a automatizar la vida de las personas. Además de realizar aplicaciones para la vida cotidiana también se pueden desarrollar aplicaciones industriales como control de calidad, donde el brazo usando su cámara detecte si algún producto tiene alguna falla. Entonces el brazo tomará el producto defectuoso y lo sacará de la línea de producción y lo colocará en un sitio designado.

De este modo se reducirían costos ya que no sería necesario que un operador deba chequear fallos, tomar manualmente el producto y parar la producción.

CONCLUSIONES

Tras haber finalizado el proyecto se pueden abstraer las siguientes conclusiones:

- Es importante saber con qué versiones de los softwares se está trabajando debido a que, si el entrenamiento fue hecho con una versión más reciente que la que se tiene en el Jetson Nano el modelo no va a cargar y no se realizará ninguna predicción. Por lo cual es importante tener las versiones más nuevas que soporte el Jetson Nano y tomar en cuenta las limitantes de hardware como la memoria RAM o la memoria de almacenamiento.
- Es importante tener documentación para poder desarrollar nuevos modelos en base a lo que se ha estudiado con anterioridad. Es recomendable utilizar Python y recursos de uso libre ya que permiten a todos los usuarios colaborar para poder desarrollar aplicaciones más complejas. Como se vio a lo largo del trabajo caffe es un modelo obsoleto que no cuenta con suficiente información. Esto impide el entendimiento de cómo desarrollar nuevos modelos y utiliza una programación compleja como es C++. No obstante, Tensor Flow tiene una amplia documentación y varios ejemplos de su utilización. Actualmente es uno de los más utilizados por lo que se cuenta con un soporte actualizado.
- Las redes neuronales convolucionales siguen siendo las más utilizadas para clasificar objetos, ya que por medio de sus filtros y el uso de ventanas pueden determinar similitudes y diferencias en los datos de entrenamiento. De esta forma realizan una predicción acertada. Utilizar capas de reducción permite no sobrecargar el sistema de información y solo utilizar la más relevante. Además, es necesario ir aumentado el

número de filtros a medida que aumentamos las capas convolucionales, este número debe ser siempre un múltiplo de 2.

- Aumentar el número de datos mejora la precisión ya que al tener más variedad de imágenes puede determinar las características más importantes que distinguen a los géneros. Todas las personas contamos con rasgos únicos con lo que mientras más datos sean utilizados, mejor será la predicción. El limitante que tiene utilizar más datos es dado por la memoria RAM y la memoria de almacenamiento.
- Es interesante ver cómo la validación tiene más precisión que el entrenamiento ya que por lo general es al revés. Esto se vio evidenciado en los experimentos ya que hubo una precisión más alta de lo que se esperaba por los valores teóricos del entrenamiento. Aquí radica la importancia de siempre realizar pruebas que contrasten a los datos teóricos.
- El movimiento que se desarrollo fue el de control por posición, eso quiere decir que si el objeto a entregar no se encuentra en la ubicación seleccionada con anterioridad el brazo no entregará nada. Si se quisiera mejorar esto y que el brazo buscara el objeto en su rango de movimiento, sería necesario desarrollar un nuevo algoritmo que le permita al robot hacer un barrido de todo su entorno en busca del objeto. Y una vez que lo localice proceda a sujetarlo para entregárselo a la persona que tiene al frente.

REFERENCIAS BIBLIOGRÁFICAS

About - OpenCV. OpenCV. (2022). Recuperado el 15 Mayo 2022, de https://opencv.org/about/.

Ashish, J. (2020). *Gender Classification 200K Images | CelebA. Kaggle.com.* Recuperado el 16 Mayo 2022, de https://www.kaggle.com/datasets/ashishjangra27/gender-recognition-200k-images-celeba.

Báez Rojas, J. J., Guerrero, M. L., Conde Acevedo, J., Padilla Vivanco, A., & Urcid Serrano, G. (2004). *Segmentación de imágenes de color.* Revista mexicana de física, 50(6), 579-587.

Berzal, F. (2019). *Redes neuronales & deep learning*. [Independently published].

Blanco, J. M., & Cohen, J. (2018*). Inteligencia artificial y poder.* Real Instituto Elcano.

Caffe, (2022). *Deep Learning Framework. Caffe.berkeleyvision.org*. Recuperado el 16 Mayo 2022, de https://caffe.berkeleyvision.org.

Calvo, D. (2022). *Funcionamiento de una capa convolucional* [Imagen]. Recuperado el 16 Mayo 2022, de https://www.diegocalvo.es/red-neuronal-convolucional/.

Díez, R. P., Gómez, A. G., & de Abajo Martínez, N. (2001). *Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva.* Universidad de Oviedo.

Gengiskanhg. (2004). *Red Neuronal Artificial de tipo es:Perceptrón simple con n neuronas de entrada, m neuronas en su capa oculta y una neurona de salida.* [Image]. Recuperado el 16 Mayo 2022, de https://commons.wikimedia.org/wiki/File:RedNeuronalArtificial.png.

Grant, S. (2022). *Gender_dataset_face. Kaggle.com*. Recuperado el 16 Mayo 2022, de https://www.kaggle.com/datasets/stevegrant38/gender-dataset-face.

Google Colaboratory. (2022). *Colab.research.google.com.* Recuperado el 16 Mayo 2022, de https://colab.research.google.com/notebooks/welcome.ipynb?hl=es.

Izaurieta, F., & Saavedra, C. (2019). *Redes Neuronales Artificiales. Universidad De Concepción Chile.* Recuperado el 16 Mayo 2022, de https://disi.unal.edu.co/~lctorress/RedNeu/LiRna003.pdf.

Kaggle. (2022). *Kaggle: Your Machine Learning and Data Science Community.* Recuperado de 16 Mayo 2022, de https://www.kaggle.com.

Levi, G. (2018*). GitHub - GilLevi/AgeGenderDeepLearning*. GitHub. Recuperado el 16 Mayo 2022, de https://github.com/GilLevi/AgeGenderDeepLearning.

Levi, G., & Hassner, T. (2015). Age and Gender Classification using Convolutional Neural Networks. University Of Israel. Recuperado el 16 Mayo 2022, de

https://talhassner.github.io/home/projects/cnn_agegender/CVPR2015_CNN_AgeGenderEstimation.pdf.

Matich, D. (2001). *Redes neuronales: Conceptos básicos y aplicaciones.* Rosario: Universidad Tecnológica Nacional Rosario. Recuperado el 16 Mayo 2022, de https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monografias/matich-redesneuronales.pdf

Nvidia. (2022). *Jetson Nano Developer Kit*. NVIDIA Developer. Recuperado el 16 Mayo 2022, de https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

Palomares, F. G., Monsoriu, J. A., & Alemany, E. (2016). *Aplicación de la convolución de matrices al filtrado de imágenes.* Modelling in Science Education and Learning, 9(1), 97-108.

Rana, Kartikeya. *Funcionamiento de filtro maxpooled.* [Imagen]. Recuperado el 16 de mayo de 2022, de https://ai.plainenglish.io/pooling-layer- beginner-to-intermediate-fa0dbdce80eb.

Rouhiainen, L. (2018). *Inteligencia artificial.* Madrid: Alienta Editorial.

TensorFlow, (2022). *Why Tensorflow.* Recuperado el 16 Mayo 2022, de https://www.tensorflow.org/about.

Ubuntu. (2022*). SwapFaq - Community Help Wiki. Help.ubuntu.com*. Recuperado el 16 Mayo 2022, de https://help.ubuntu.com/community/SwapFaq.

Yahboom. Yahboom.net. (2022). Recuperado el 16 Mayo 2022, de http://www.yahboom.net/study/Dofbot-Jetson_nano.

ANEXO A: CÓDIGO DE APLICACIÓN PROTOTIPO N°1

```python
#Gender Action Code
#Leonel Miranda & Daniel Jimenez
#Version 1

import cv2
import time
from Arm_Lib import Arm_Device

class action_group:
    def __init__(self):

        self.Arm = Arm_Device()
        self.started = 0


    def set_state(self, state):
        self.started = state


    def read_state(self):
        return self.started


    def arm_move(self, p, s_time = 500):
        for i in range(5):
            id = i + 1
            if id == 5:
                time.sleep(.1)
                self.Arm.Arm_serial_servo_write(id, p[i], int(s_time*1.2))
            else:
                self.Arm.Arm_serial_servo_write(id, p[i], int(s_time))
            time.sleep(.01)
        time.sleep(s_time/1000)


    def arm_clamp_block(self, enable):
        if enable == 0:
            self.Arm.Arm_serial_servo_write(6, 60, 400)
        else:
            self.Arm.Arm_serial_servo_write(6, 135, 400)
        time.sleep(.5)


    def start_action(self, index):
        if self.started == 1:
            self.custom_action_group(index)
            self.started = 0
```

```
    def custom_action_group(self, number):
        if number == 1:

            self.Arm.Arm_serial_servo_write6(90, 180, 0, 0, 90, 0, 1000)

        elif number == 2:

            if self.started == 1:
                self.Arm.Arm_serial_servo_write6(63, 22, 64, 54, 270, 0, 1000)
            if self.started == 1:
                time.sleep(1)

            if self.started == 1:
                self.Arm.Arm_serial_servo_write(6, 155, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write6(90, 90, 64, 54, 270, 155, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write(2, 22, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write(6, 0, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write6(90, 180, 0, 0, 90, 0, 1000)
                time.sleep(1)

        elif number == 3:

            if self.started == 1:
                self.Arm.Arm_serial_servo_write6(117, 19, 66, 56, 270, 0, 1000)
            if self.started == 1:
                time.sleep(1)

            if self.started == 1:
                self.Arm.Arm_serial_servo_write(6, 155, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write6(90, 90, 64, 54, 270, 155, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write(2, 22, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write(6, 0, 1000)
                time.sleep(1)
            if self.started == 1:
                self.Arm.Arm_serial_servo_write6(90, 180, 0, 0, 90, 0, 1000)
                time.sleep(1)

        elif number == 4:

            self.Arm.Arm_serial_servo_write6(65, 22, 64, 56, 270, 155, 1000)
```

```
104
105         elif number == 5:
106
107             self.Arm.Arm_serial_servo_write6(65, 22, 64, 56, 270, 155, 1000)
108
109         elif number == 6:
110             self.Arm.Arm_serial_servo_write6(90, 180, 0, 0, 90, 155, 1000)
111         elif number == 7:
112             self.Arm.Arm_serial_servo_write6(90, 0, 90, 90, 90, 180, 1000)
113         elif number == 8:
114             self.Arm.Arm_serial_servo_write6(90, 134, 43, 13, 90, 180, 1000)
115
116         self.started = 0
117
118     def arm_move_h(self, p, s_time = 500):
119         for i in range(5):
120             id = i + 1
121             if id == 5:
122                 time.sleep(.1)
123                 self.Arm.Arm_serial_servo_write(id, p[i], int(s_time*1.2))
124             elif id == 1 :
125                 self.Arm.Arm_serial_servo_write(id, p[i], int(3*s_time/4))
126
127             else:
128                 self.Arm.Arm_serial_servo_write(id, p[i], int(s_time))
129             time.sleep(.01)
130         time.sleep(s_time/1000)
131
132     def arm_move_up(self):
133         self.Arm.Arm_serial_servo_write(2, 90, 1500)
134         self.Arm.Arm_serial_servo_write(3, 90, 1500)
135         self.Arm.Arm_serial_servo_write(4, 90, 1500)
136         time.sleep(.1)
137
138
139 def faceBox(faceNet,frame):
140     frameHeight=frame.shape[0]
141     frameWidth=frame.shape[1]
142     blob=cv2.dnn.blobFromImage(frame, 1.0, (300,300), [104,117,123], swapRB=False)
143     faceNet.setInput(blob)
144     detection=faceNet.forward()
145     bboxs=[]
146     for i in range(detection.shape[2]):
147         confidence=detection[0,0,i,2]
148         if confidence>0.7:
149             x1=int(detection[0,0,i,3]*frameWidth)
150             y1=int(detection[0,0,i,4]*frameHeight)
151             x2=int(detection[0,0,i,5]*frameWidth)
152             y2=int(detection[0,0,i,6]*frameHeight)
153             bboxs.append([x1,y1,x2,y2])
154             cv2.rectangle(frame, (x1,y1),(x2,y2),(0,255,0), 1)
155     return frame, bboxs
156
```

```
faceProto = "opencv_face_detector.pbtxt"
faceModel = "opencv_face_detector_uint8.pb"

ageProto = "age_deploy.prototxt"
ageModel = "age_net.caffemodel"

genderProto = "gender_deploy.prototxt"
genderModel = "gender_net.caffemodel"



faceNet=cv2.dnn.readNet(faceModel, faceProto)
ageNet=cv2.dnn.readNet(ageModel,ageProto)
genderNet=cv2.dnn.readNet(genderModel,genderProto)

MODEL_MEAN_VALUES = (78.4263377603, 87.7689143744, 114.895847746)
ageList = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)', '(48-53)', '(60-100)']
genderList = ['Male', 'Female']


video=cv2.VideoCapture(0)
waitT=0
padding=20

while True:
    ret,frame=video.read()
    frame,bboxs=faceBox(faceNet,frame)
    action = action_group()
    action.set_state(True)
    action.start_action(1)
    for bbox in bboxs:
        # face=frame[bbox[1]:bbox[3], bbox[0]:bbox[2]]

        waitT=waitT+1


    face = frame[max(0,bbox[1]-padding):min(bbox[3]+padding,frame.shape[0]-1),max(0,bbox[0]-padding):min(bbox[2]+padding, frame.shape[1]-1)]
    blob=cv2.dnn.blobFromImage(face, 1.0, (227,227), MODEL_MEAN_VALUES, swapRB=False)
    genderNet.setInput(blob)
    genderPred=genderNet.forward()
    gender=genderList[genderPred[0].argmax()]

    ageNet.setInput(blob)
    agePred=ageNet.forward()
    age=ageList[agePred[0].argmax()]

    label="{},{}".format(gender,age)
    cv2.rectangle(frame,(bbox[0], bbox[1]-30), (bbox[2], bbox[1]), (0,255,0),-1)
    cv2.putText(frame, label, (bbox[0], bbox[1]-10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255,255,255), 2,cv2.LINE_AA)

    #print(waitT)

    if waitT == 8:

      if gender == 'Male':
        action = action_group()
        action.set_state(True)
        action.start_action(2)
        waitT=0
      elif gender ==  'Female':
        action = action_group()
        action.set_state(True)
        action.start_action(3)
        waitT=0


    cv2.imshow("Gender_Action",frame)
    k=cv2.waitKey(1)

    if k==ord('q'):
        break
video.release()
cv2.destroyAllWindows()
```

ANEXO B: CÓDIGO DE ENTRENAMIENTO PROTOTIPO N°2

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] %cd /content/drive/MyDrive/GD
    ! ls
```

```
/content/drive/MyDrive/GD
GenderD  gender_dataset_face
```

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.preprocessing.image import img_to_array
    from tensorflow.keras.utils import to_categorical, plot_model
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense
    from tensorflow.keras import backend as K
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
    import numpy as np
    import random
    import cv2
    import os
    import glob
```

```
[ ] # load image files from the dataset
    image_files = [f for f in glob.glob(r'/content/drive/MyDrive/GD/gender_dataset_face' + "/**/*", recursive=True) if not os.path.isdir(f)]
    random.shuffle(image_files)
```

```
[ ] # initial parameters
    epochs = 100
    lr = 1e-3
    batch_size = 64
    img_dims = (96,96,3)

    data = []
    labels = []
```

```
[ ] # converting images to arrays and labelling the categories
    for img in image_files:

        image = cv2.imread(img)

        image = cv2.resize(image, (img_dims[0],img_dims[1]))
        image = img_to_array(image)
        data.append(image)

        label = img.split(os.path.sep)[-2] # /content/drive/MyDrive/GD/gender_dataset_face/face_1162.jpg
        if label == "Female":
            label = 1
        else:
            label = 0
```

```
[ ]     labels.append([label]) # [[1], [0], [0], ...]
```

```
[ ] # pre-processing
    data = np.array(data, dtype="float") / 255.0
    labels = np.array(labels)
```

```
[ ] # split dataset for training and validation
    (trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2,
                                                      random_state=42)

    trainY = to_categorical(trainY, num_classes=2) # [[1, 0], [0, 1], [0, 1], ...]
    testY = to_categorical(testY, num_classes=2)
```

```
[ ] # augmenting datset
    aug = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
                             height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
                             horizontal_flip=True, fill_mode="nearest")
```

```
[ ] # define model
    def build(width, height, depth, classes):
        model = Sequential()
        inputShape = (height, width, depth)
        chanDim = -1

        if K.image_data_format() == "channels_first": #Returns a string, either 'channels_first' or 'channels_last'
            inputShape = (depth, height, width)
            chanDim = 1

        # The axis that should be normalized, after a Conv2D layer with data_format="channels_first",
        # set axis=1 in BatchNormalization.

        model.add(Conv2D(32, (3,3), padding="same", input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(3,3)))
        model.add(Dropout(0.25))

        model.add(Conv2D(64, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))

        model.add(Conv2D(64, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))
```

```
[ ]     model.add(Conv2D(128, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))

        model.add(Conv2D(128, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))

        model.add(Flatten())
        model.add(Dense(1024))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        model.add(Dense(classes))
        model.add(Activation("sigmoid"))

        return model
```

```
[ ]  # build model
     model = build(width=img_dims[0], height=img_dims[1], depth=img_dims[2],
                                 classes=2)
```

```
[ ]  # compile the model
     opt = Adam(learning_rate=lr, decay=lr/epochs)
     model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
[ ]  # train the model
     H = model.fit(aug.flow(trainX, trainY, batch_size=batch_size),
                           validation_data=(testX,testY),
                           steps_per_epoch=len(trainX) // batch_size,
                           epochs=epochs, verbose=1)
```

```
[ ]  # save the model to disk
     model.save('gender_detection.h5')
```

```
[ ]  # plot training/validation loss/accuracy
     plt.style.use("ggplot")
     plt.figure()
     N = epochs
     plt.plot(np.arange(0,N), H.history["loss"], label="train_loss")
     plt.plot(np.arange(0,N), H.history["val_loss"], label="val_loss")
     plt.plot(np.arange(0,N), H.history["accuracy"], label="train_accuracy")
     plt.plot(np.arange(0,N), H.history["val_accuracy"], label="val_accuracy")

     plt.title("Training Loss and Accuracy")
     plt.xlabel("Epoch #")
     plt.ylabel("Loss/Accuracy")
     plt.legend(loc="upper right")
```

```
[ ]  # save plot to disk
     plt.savefig('plot.png')
```

ANEXO C: CÓDIGO DE APLICACIÓN PROTOTIPO N°2

```
#Gender Detection Code
#Leonel Miranda & Daniel Jimenez
#Version 2

from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from action_group import action_group
import numpy as np
import cv2
import os
import cvlib as cv

# load model
model = load_model('gender_detection.h5')

# open webcam
webcam = cv2.VideoCapture(0)
waitT=0

classes = ['man','woman']

# loop through frames
while webcam.isOpened():

    # read frame from webcam
    status, frame = webcam.read()

    # apply face detection
    face, confidence = cv.detect_face(frame)

    #set initial position
    action = action_group()
    action.set_state(True)
    action.start_action(1)

    # loop through detected faces
    for idx, f in enumerate(face):

        # delay time
        waitT=waitT+1


        # get corner points of face rectangle
        (startX, startY) = f[0], f[1]
        (endX, endY) = f[2], f[3]

        # draw rectangle over face
        cv2.rectangle(frame, (startX,startY), (endX,endY), (0,255,0), 2)

        # crop the detected face region
        face_crop = np.copy(frame[startY:endY,startX:endX])

        if (face_crop.shape[0]) < 10 or (face_crop.shape[1]) < 10:
            continue

```

```
55          # preprocessing for gender detection model
56          face_crop = cv2.resize(face_crop, (96,96))
57          face_crop = face_crop.astype("float") / 255.0
58          face_crop = img_to_array(face_crop)
59          face_crop = np.expand_dims(face_crop, axis=0)
60
61          # apply gender detection on face
62          conf = model.predict(face_crop)[0] # model.predict return a 2D matrix,
63
64          # get label with max accuracy
65          idx = np.argmax(conf)
66          label = classes[idx]
67          gender = label
68
69          label = "{}: {:.2f}%".format(label, conf[idx] * 100)
70
71          Y = startY - 10 if startY - 10 > 10 else startY + 10
72
73          # write label and confidence above face rectangle
74          cv2.putText(frame, label, (startX, Y),  cv2.FONT_HERSHEY_SIMPLEX,
75                      0.7, (0, 255, 0), 2)
76
77          # Move the arm by the classification
78          if waitT == 8:
79
80              if label == 'man':
81                  action = action_group()
82                  action.set_state(True)
83                  action.start_action(2)
84                  waitT=0
85              elif label == 'woman':
86                  action = action_group()
87                  action.set_state(True)
88                  action.start_action(3)
89                  waitT=0
90
91      # display output
92      cv2.imshow("gender detection", frame)

93
94      # press "Q" to stop
95      if cv2.waitKey(1) & 0xFF == ord('q'):
96          break
97
98  # release resources
99  webcam.release()
100 cv2.destroyAllWindows()
101
```

```python
#Action Group code
# !/usr/bin/env python
# coding: utf-8

import time
from Arm_Lib import Arm_Device


class action_group:
    def __init__(self):

        self.Arm = Arm_Device()
        self.started = 0


    def set_state(self, state):
        self.started = state


    def read_state(self):
        return self.started


    def arm_move(self, p, s_time = 500):
        for i in range(5):
            id = i + 1
            if id == 5:
                time.sleep(.1)
                self.Arm.Arm_serial_servo_write(id, p[i], int(s_time*1.2))
            else:
                self.Arm.Arm_serial_servo_write(id, p[i], int(s_time))
            time.sleep(.01)
        time.sleep(s_time/1000)


    def arm_clamp_block(self, enable):
        if enable == 0:
            self.Arm.Arm_serial_servo_write(6, 60, 400)
        else:
            self.Arm.Arm serial servo write(6, 135, 400)

        time.sleep(.5)


    def start_action(self, index):
        if self.started == 1:
            self.custom_action_group(index)
            self.started = 0

    def custom_action_group(self, number):
        if number == 1:

            self.Arm.Arm_serial_servo_write6(90, 180, 0, 0, 90, 0, 100)

```

```
63                  time.sleep(1)
64              if self.started == 1:
65                  self.Arm.Arm_serial_servo_write6(90, 90, 64, 54, 270, 155, 1000)
66                  time.sleep(1)
67              if self.started == 1:
68                  self.Arm.Arm_serial_servo_write(2, 22, 1000)
69                  time.sleep(1)
70              if self.started == 1:
71                  self.Arm.Arm_serial_servo_write(6, 0, 1000)
72                  time.sleep(1)
73              if self.started == 1:
74                  self.Arm.Arm_serial_servo_write6(90, 180, 0, 0, 90, 0, 1000)
75                  time.sleep(1)
76
77              # time.sleep(1)
78          elif number == 3:
79
80              if self.started == 1:
81                  self.Arm.Arm_serial_servo_write6(117, 19, 66, 56, 270, 0, 1000)
82              if self.started == 1:
83                  time.sleep(1)
84
85              if self.started == 1:
86                  self.Arm.Arm_serial_servo_write(6, 155, 1000)
87                  time.sleep(1)
88              if self.started == 1:
89                  self.Arm.Arm_serial_servo_write6(90, 90, 64, 54, 270, 155, 1000)
90                  time.sleep(1)
91              if self.started == 1:
92                  self.Arm.Arm_serial_servo_write(2, 22, 1000)

93                  time.sleep(1)
94              if self.started == 1:
95                  self.Arm.Arm_serial_servo_write(6, 0, 1000)
96                  time.sleep(1)
97              if self.started == 1:
98                  self.Arm.Arm_serial_servo_write6(90, 180, 0, 0, 90, 0, 1000)
99                  time.sleep(1)
100
101             # time.sleep(1)
102
103         self.started = 0
104
105     def arm_move_h(self, p, s_time = 500):
106         for i in range(5):
107             id = i + 1
108             if id == 5:
109                 time.sleep(.1)
110                 self.Arm.Arm_serial_servo_write(id, p[i], int(s_time*1.2))
111             elif id == 1 :
112                 self.Arm.Arm_serial_servo_write(id, p[i], int(3*s_time/4))
```

```
113
114             else:
115                 self.Arm.Arm_serial_servo_write(id, p[i], int(s_time))
116             time.sleep(.01)
117         time.sleep(s_time/1000)
118
119     def arm_move_up(self):
120         self.Arm.Arm_serial_servo_write(2, 90, 1500)
121         self.Arm.Arm_serial_servo_write(3, 90, 1500)
122         self.Arm.Arm_serial_servo_write(4, 90, 1500)
123         time.sleep(.1)
124
```

ANEXO D: CÓDIGO DE ENTRENAMIENTO MODELO FINAL

Algoritmo de reconocimiento de Genero

```
[ ] ! pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle) (2021.10.8)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4.64.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle) (6.1.2)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (2.10)
```

```
[ ] import os
    os.environ['KAGGLE_CONFIG_DIR'] ='/content'
```

```
[ ] !chmod 600 /content/kaggle.json
```

```
[ ] !kaggle datasets download -d ashishjangra27/gender-recognition-200k-images-celeba --force
```

```
Downloading gender-recognition-200k-images-celeba.zip to /content
 98% 1.29G/1.32G [00:10<00:00, 196MB/s]
100% 1.32G/1.32G [00:10<00:00, 140MB/s]
```

```
[ ] !unzip /content/gender-recognition-200k-images-celeba.zip
```

```
  inflating: Dataset/Validation/Male/189708.jpg
  inflating: Dataset/Validation/Male/189710.jpg
  inflating: Dataset/Validation/Male/189717.jpg
  inflating: Dataset/Validation/Male/189718.jpg
  inflating: Dataset/Validation/Male/189719.jpg
  inflating: Dataset/Validation/Male/189720.jpg
  inflating: Dataset/Validation/Male/189722.jpg
  inflating: Dataset/Validation/Male/189726.jpg
  inflating: Dataset/Validation/Male/189727.jpg
  inflating: Dataset/Validation/Male/189728.jpg
  inflating: Dataset/Validation/Male/189729.jpg
  inflating: Dataset/Validation/Male/189734.jpg
  inflating: Dataset/Validation/Male/189736.jpg
  inflating: Dataset/Validation/Male/189742.jpg
  inflating: Dataset/Validation/Male/189744.jpg
  inflating: Dataset/Validation/Male/189745.jpg
  inflating: Dataset/Validation/Male/189746.jpg
  inflating: Dataset/Validation/Male/189747.jpg
  inflating: Dataset/Validation/Male/189752.jpg
  inflating: Dataset/Validation/Male/189753.jpg
  inflating: Dataset/Validation/Male/189754.jpg
  inflating: Dataset/Validation/Male/189755.jpg
  inflating: Dataset/Validation/Male/189756.jpg
  inflating: Dataset/Validation/Male/189758.jpg
  inflating: Dataset/Validation/Male/189762.jpg
  inflating: Dataset/Validation/Male/189763.jpg
  inflating: Dataset/Validation/Male/189764.jpg
  inflating: Dataset/Validation/Male/189765.jpg
  inflating: Dataset/Validation/Male/189767.jpg
  inflating: Dataset/Validation/Male/189768.jpg
  inflating: Dataset/Validation/Male/189770.jpg
```

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.preprocessing.image import img_to_array
    from tensorflow.keras.utils import to_categorical, plot_model
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import BatchNormalization, Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense
    from tensorflow.keras import backend as K
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
    import numpy as np
    import random
    import cv2
    import os
    import glob
```

```
[ ] # initial parameters
    epochs = 100
    lr = 1e-3
    batch_size = 64
    img_dims = (100,100,3)

    data = []
    labels = []
```

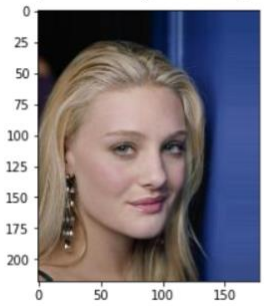
```
[ ] # load image files from the dataset
    image_files = [f for f in glob.glob(r'/content/Dataset/Train' + "/**/*", recursive=True) if not os.path.isdir(f)]
    random.shuffle(image_files)
```

```
[ ] image_files1=image_files[0:50000]
    print(image_files1[25000])
```

```
/content/Dataset/Train/Male/132339.jpg
```

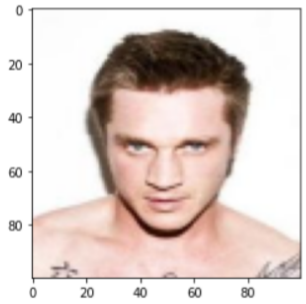
```
[ ] u=cv2.imread(image_files1[0])
    u=cv2.cvtColor(u, cv2.COLOR_BGR2RGB)
    plt.imshow(u)
```

```
<matplotlib.image.AxesImage at 0x7f7e6bd5d790>
```



```
[ ] # pre-processing
    data = np.array(data, dtype="float") / 255.0
    labels = np.array(labels)
    #print((data[96]*255).astype(int))
    plt.imshow((data[96]*255).astype(int))
```

```
<matplotlib.image.AxesImage at 0x7f7e6e3ede90>
```

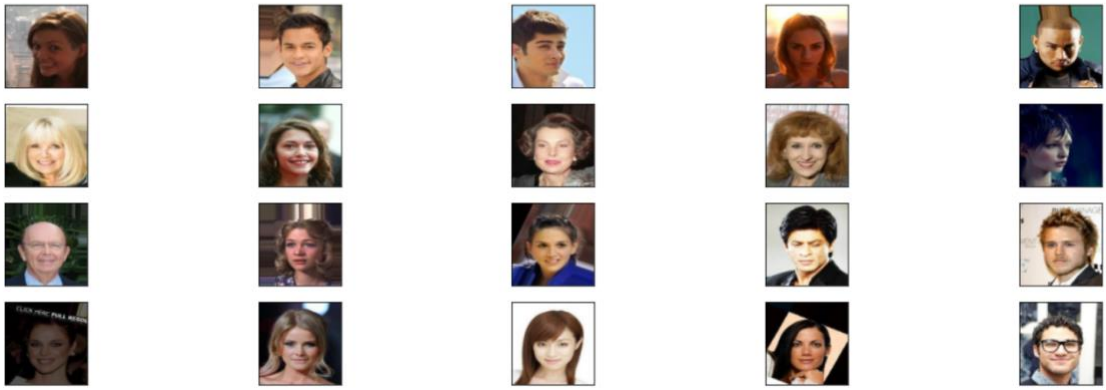



```
# split dataset for training and validation
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2,
                                                  random_state=42)

trainY = to_categorical(trainY, num_classes=2) # [[1, 0], [0, 1], [0, 1], ...]
testY = to_categorical(testY, num_classes=2)
```

```
[ ] # augmenting datset
    aug = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
                             height_shift_range=0.1,
                             shear_range=0.2, zoom_range=[0.2,1.5],
                             horizontal_flip=True, fill_mode="nearest"
                             )
```

```
[ ] #aug.fit(trainX)
    plt.figure(figsize=(20,8))
    aug.flow(trainX, trainY, batch_size=batch_size,shuffle=False)
    for i in range(25):
        plt.subplot(5, 5, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow((trainX[i*10]*255).astype(int))
        #print((trainX[i]*255).astype(int))
```



```
[ ]  # define model
     def build(width, height, depth, classes):
         model = Sequential()
         inputShape = (height, width, depth)
         chanDim = -1

         if K.image_data_format() == "channels_first": #Returns a string, either 'channels_first' or 'channels_last'
             inputShape = (depth, height, width)
             chanDim = 1

         # The axis that should be normalized, after a Conv2D layer with data_format="channels_first",
         # set axis=1 in BatchNormalization.

         model.add(Conv2D(32, (3,3), padding="same", input_shape=inputShape))
         model.add(Activation("relu"))
         model.add(BatchNormalization(axis=chanDim))
         model.add(MaxPooling2D(pool_size=(3,3)))
         model.add(Dropout(0.25))

         model.add(Conv2D(64, (3,3), padding="same"))
         model.add(Activation("relu"))
         model.add(BatchNormalization(axis=chanDim))

         model.add(Conv2D(64, (3,3), padding="same"))
         model.add(Activation("relu"))
         model.add(BatchNormalization(axis=chanDim))
         model.add(MaxPooling2D(pool_size=(2,2)))
         model.add(Dropout(0.25))
```

```
[ ]      model.add(Conv2D(128, (3,3), padding="same"))
         model.add(Activation("relu"))
         model.add(BatchNormalization(axis=chanDim))

         model.add(Conv2D(128, (3,3), padding="same"))
         model.add(Activation("relu"))
         model.add(BatchNormalization(axis=chanDim))
         model.add(MaxPooling2D(pool_size=(2,2)))
         model.add(Dropout(0.25))

         model.add(Flatten())
         model.add(Dense(1024))
         model.add(Activation("relu"))
         model.add(BatchNormalization())
         model.add(Dropout(0.5))

         model.add(Dense(classes))
         model.add(Activation("sigmoid"))

         return model
```

```
[ ]  # build model
     model = build(width=img_dims[0], height=img_dims[1], depth=img_dims[2],
                                 classes=2)
```

```
[ ]  # compile the model
     opt = Adam(learning_rate=lr, decay=lr/epochs)
     model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
[ ] # train the model
    H = model.fit(aug.flow(trainX, trainY, batch_size=batch_size),
                            validation_data=(testX,testY),
                            steps_per_epoch=len(trainX) // batch_size,
                            epochs=epochs, verbose=1)
```

```
Epoch 72/100
625/625 [==============================] - 158s 253ms/step - loss: 0.1100 - accuracy: 0.9566 - val_loss: 0.1289 - val_accuracy: 0.9521
Epoch 73/100
625/625 [==============================] - 159s 254ms/step - loss: 0.1112 - accuracy: 0.9563 - val_loss: 0.0685 - val_accuracy: 0.9743
Epoch 74/100
625/625 [==============================] - 157s 250ms/step - loss: 0.1113 - accuracy: 0.9568 - val_loss: 0.0675 - val_accuracy: 0.9739
Epoch 75/100
625/625 [==============================] - 157s 250ms/step - loss: 0.1110 - accuracy: 0.9572 - val_loss: 0.0626 - val_accuracy: 0.9785
Epoch 76/100
625/625 [==============================] - 156s 250ms/step - loss: 0.1095 - accuracy: 0.9572 - val_loss: 0.0964 - val_accuracy: 0.9651
Epoch 77/100
625/625 [==============================] - 157s 251ms/step - loss: 0.1088 - accuracy: 0.9584 - val_loss: 0.0988 - val_accuracy: 0.9638
```

```
[ ]
```

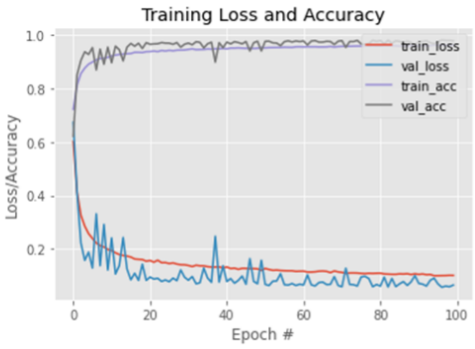
```
Epoch 99/100
625/625 [==============================] - 159s 255ms/step - loss: 0.1024 - accuracy: 0.9608 - val_loss: 0.0600 - val_accuracy: 0.9781
Epoch 100/100
625/625 [==============================] - 161s 257ms/step - loss: 0.1021 - accuracy: 0.9607 - val_loss: 0.0662 - val_accuracy: 0.9770
```

```
[ ] # save the model to disk
    model.save('genderD.h5')
```

```
[ ] # plot training/validation loss/accuracy
    plt.style.use("ggplot")
    plt.figure()
    N = epochs
    plt.plot(np.arange(0,N), H.history["loss"], label="train_loss")
    plt.plot(np.arange(0,N), H.history["val_loss"], label="val_loss")
    plt.plot(np.arange(0,N), H.history["accuracy"], label="train_accuracy")
    plt.plot(np.arange(0,N), H.history["val_accuracy"], label="val_accuracy")

    plt.title("Training Loss and Accuracy")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss/Accuracy")
    plt.legend(loc="upper right")
```

```
<matplotlib.legend.Legend at 0x7f3166254310>
```




```
[ ] # save plot to disk
    plt.savefig('plot.png')
```

```
<Figure size 432x288 with 0 Axes>
```

ANEXO E: CÓDIGO DE APLICACION MODELO FINAL

```
#Gender Detection Code
#Leonel Miranda & Daniel Jimenez
#Version 3

from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from action_group import action_group
import numpy as np
import cv2
import os
import cvlib as cv

# load model
model = load_model('genderD.h5')

# open webcam
webcam = cv2.VideoCapture(0)
waitT=0
pred=0

classes = ['man','woman']

# loop through frames
while webcam.isOpened():

    # read frame from webcam
    status, frame = webcam.read()

    # apply face detection
    face, confidence = cv.detect_face(frame)

    #set initial position
    action = action_group()
    action.set_state(True)
    action.start_action(1)

    # loop through detected faces
    for idx, f in enumerate(face):

        # delay time


        waitT=waitT+1

        # get corner points of face rectangle
        (startX, startY) = f[0], f[1]
        (endX, endY) = f[2], f[3]

        # draw rectangle over face
        cv2.rectangle(frame, (startX,startY), (endX,endY), (0,255,0), 2)

        # crop the detected face region
        face_crop = np.copy(frame[startY:endY,startX:endX])

        if (face_crop.shape[0]) < 10 or (face_crop.shape[1]) < 10:
            continue

        # preprocessing for gender detection model
        face_crop = cv2.resize(face_crop, (96,96))
        face_crop = face_crop.astype("float") / 255.0
```

```
59          face_crop = img_to_array(face_crop)
60          face_crop = np.expand_dims(face_crop, axis=0)
61
62          # apply gender detection on face
63          conf = model.predict(face_crop)[0] # model.predict return a 2D matrix, ex: [|
64
65          # get label with max accuracy
66          idx = np.argmax(conf)
67          label = classes[idx]
68          gender = label
69
70          label = "{}: {:.2f}%".format(label, conf[idx] * 100)
71
72          Y = startY - 10 if startY - 10 > 10 else startY + 10
73
74          # write label and confidence above face rectangle
75          cv2.putText(frame, label, (startX, Y),  cv2.FONT_HERSHEY_SIMPLEX,
76                      0.7, (0, 255, 0), 2)
77
78          # mean
79          if gender == 'man':
80             pred = pred+1
81
82          #Fix arm don't move
83          if pred > 10 or waitT >10:
84             pred=0
85             waitT=0
86
87          print(waitT, ' ', pred)
88          # Move the arm by the classification
89          if waitT == 9:
90
91             if pred >= 5: #Male
92                print(pred, 'Male')
93                action = action_group()
94                action.set_state(True)
95                action.start_action(2)
96                waitT=0
97                pred=0
98             elif pred < 5: #Female
99                print(9-pred, 'Female')
100               action = action_group()
101               action.set_state(True)
102               action.start_action(3)
103               waitT=0
104               pred=0
105
106      # display output
107      cv2.imshow("gender detection", frame)
108
109      # press "Q" to stop
110      if cv2.waitKey(1) & 0xFF == ord('q'):
111          break
112
113  # release resources
114  webcam.release()
115  cv2.destroyAllWindows()
116
```