Detector de puntos faciales: estimación de la posición de los ojos

Pontificia Universidad Javeriana

Facultad de Ingeniería
Carrera de Ingeniería Electrónica

Inteligencia Artificial

Ing. Francisco Carlos Calderon Bocanegra

Karen Andrea Beltrán Silva
Laura Alejandra Estupiñan Martínez

2020

Índice

1. Introducción **3**

2. Objetivos **3**
2.1. Objetivo general 3
2.2. Objetivos específicos 3

3. Desarrollo **4**

4. Resultados **6**

5. Conclusiones **12**

1. Introducción

El ser humano tiene mayor capacidad de percibir el mundo a partir de las imágenes, y es por esto que la vista es uno de los principales enfoques de investigación para el desarrollo de tecnologías que permitan emular propiedades de percepción a partir de la detección y seguimiento de los ojos.

Los sistemas de estimación de la mirada se dividen en dos categorías principales, la primera se basa en la detección sobre puntos de referencia de una región específica del ojo, como la localización del iris, a partir de suposiciones geométricas que suelen ser sensibles a la fluctuaciones de variables como la iluminación. Por otro lado, existen técnicas de arquitecturas CNN profundas, que a pesar de requerir una alta capacidad de computo para el proceso de entrenamiento y generación de modelos, permite estimar de manera precisa las características de interés [1].

La detección de los puntos de referencia se consideran un componente fundamental en el análisis de inferencia, verificación y reconocimiento facial. El artículo *Learning Deep Representation for Face Alignment with Auxiliary Attributes* procura optimizar este proceso de estimación para inferir atributos adicionales como la postura de la cabeza, género y edad, desde la predicción de la ubicación de la nariz, comisura de los labios y ojos. Bajo el objetivo previamente mencionado, sugiere un modelo de red convulocional profunda que genere un aprendizaje efectivo, sin importar las dificultades que representa la amplia diversidad de tasas de convergencia [2].

El entrenamiento de esta tipología se generó con el dataset *Facial Landmark Detection by Deep Multi-task Learning* el cual realiza las evaluaciones siguiendo el estándar de extracción de información propuesta. Este método aumentó su soporte a 68 puntos de referencia en 5 zonas faciales diferentes, sin afectar el costo del tiempo de ejercición, asegurando un error medio de 9.15 % [3].

Por otro lado, Omri Goldstein contribuidor de Kaggle, propuso un conjunto de datos de detección de puntos faciales, conocido como *Face Images with Marked Landmark Points*, el cual permite implementar kernels sobre sus 7049 imágenes totales. A partir de esta fuente se obtuvo acceso al notebook *Landmark detection* el cual propone un otro modelo de entrenamiento y validación con redes neuronales [4] [5].

En el presente documento describe el procedimiento realizado para implementar las de redes neuronales convolucionales propuestas en *Learning Deep Representation for Face Alignment with Auxiliary Attributes* y *Landmark detection*, y posteriormente entrenarlas con la base de datos *Facial Landmark Detection by Deep Multi-task Learning* de 4151 fotos con sus respectivas marcas de referencia. Esto con la finalidad de evaluar su rendimiento a partir de la detección de atributos faciales, y así concluir el modelo de mayor capacidad en términos de precisión [2] [5] [3].

2. Objetivos

2.1. Objetivo general

Implementar una red neuronal convolucional eficiente que determine la posición ocular de un usuario con respecto a una cámara web, a partir de la detección de puntos faciales de referencia.

2.2. Objetivos específicos

- Investigar diferentes modelos de redes neuronales convolutivas.
- Determinar una base de datos adecuada para el entrenamiento y validación de las topologías.

- Modelar las configuraciones seleccionadas en python, a través de las bibliotecas Tensorflow y Keras.
- Realizar un análisis comparativo entre dos modelos entrenados con la misma base de datos para establecer el de mejor rendimiento.

3. Desarrollo

Para cumplir los objetivos del proyecto se llevaron a cabo cinco pasos. En primer lugar, se realizó una investigación sobre las diferentes tipologías desde diferentes fuentes como *Kaggle*, siendo una página web con bases de datos e implementaciones, o en publicaciones con distintas propuestas de arquitecturas que cumplen el propósito.

Las diferencias principales que se encontraron entre los modelos son el tamaño de las imágenes de entrada, la cantidad de capas y las combinaciones posibles entre los tipos de capas. Se escogieron dos tipologías, la primera extraída del artículo *"Learning Deep Representation for Face"* [2], el cuál describe las especificaciones de 6 capas. La segunda es un notebook de *Kaggle* [3], donde proponen un diseño de 3 capas a partir de una base de datos expuesta en la misma página.

El modelo de 6 capas se ilustra en la figura 1 que tiene como salida el punto en x y y del centro de los ojos, la punta de la nariz y las comisuras de los labios, además de atributos cómo si usa gafas, si tiene barba, el género, entre otros. La capa de entrada permite un tamaño de 60x60 píxeles, se realizan 20 filtros de 5x5 y un *max-pool* de 2x2. Las capas dos, tres y cuatro realizan el mismo procedimiento, pero con un filtro de 3x3 y la cantidad de neuronas es de 48, 64 y 80 respectivamente. La capa cinco, realiza un *flatten* de 256 con las posibles ubicaciones de las etiquetas. Por último, la capa con las 10 salidas de los 5 puntos faciales y los atributos especificados [2].

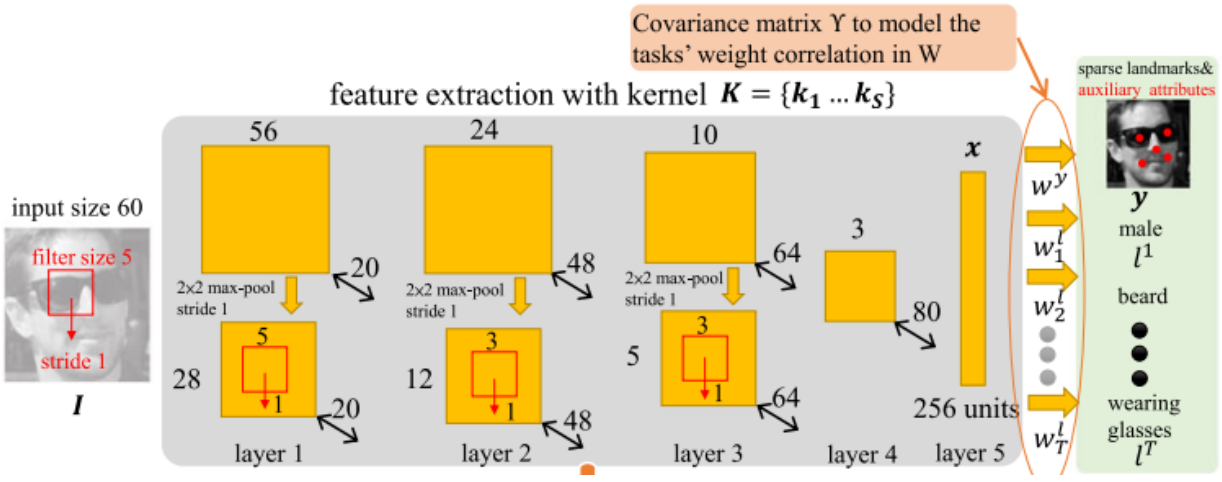


Figura 1: Diagrama de distribución de las capas implementadas por el primer modelo [2]

El modelo de 3 capas tiene cómo entrada una imagen de 96x96 con 32 filtros de 3x3 y una ventana de *max-pool* de 2x2. La capa del medio realiza un *dropout* para evitar un sobre entrenamiento y un *flatten* de los datos a 256 unidades. La última que es la de salida, saca los puntos fáciles que se pueden modificar dependiendo de la base de datos a utilizar [3].

El segundo paso que se realizó fue la búsqueda de bases de datos que cumplieran con los resultados esperados. La fuente del primer modelo expone una propuesta de 10.000 imágenes de entrenamiento y 3.000 de validación. Las etiquetas se encuentran organizadas por nombres en un archivo *.csv*, lo que permitió generar un archivo de texto con el orden de las imágenes y hacer el procesamiento correctamente. Los datos proporcionados son la posición en x y y del centro de los ojos, la punta de la nariz, la comisura de la boca y atributos como el género, si está sonriendo, si usa gafas y la posición de la cabeza [6].

En el tercer paso se diseñó el procesamiento de la base de datos. A partir de los archivos *traininglistaimagenes.txt* y *testinglistaimagenes.txt* se obtuvieron los nombres en el orden que están las etiquetas, luego se unió el *path* donde se encuentran en el computador. Por medio de la biblioteca *opencv* se leen y se cambia el tamaño al especificado por la red neuronal, luego se guardan las matrices en una variable que se convierte en la matriz X tanto de entrenamiento como de validación. Para las etiquetas, con la librería pandas se obtienen los datos a partir de los archivos *training.csv* y *testing.csv* que se convierten al vector Y. También se debe ajustar la dimensión de los datos al tamaño de las imágenes que se ajustaron en el paso anterior. Con esto, es posible entrenar la red desde los vectores X y Y obtenidos.

El cuarto paso es implementar las redes neuronales convolucionales para realizar el entrenamiento. A partir del lenguaje de programación *Python* se utilizaron las bibliotecas de *Keras* y *Tensorflow*. Las funciones que permitieron la elaboración del proyecto son *Conv2D*, *MaxPooling2D*, *Dropout*, *Flatten* y *Dense*. Los atributos como *sequential* posibilitaron crear la red, *add* agregar capas y *fit* realizar el entrenamiento.

La capa *Conv2D* [7]crea un núcleo de convolución con la entrada para producir un tensor de salidas. Los argumentos de entrada utilizados son dejando los demás por defecto:

- **Filters:** Cantidad de filtros de salida en la convolución, debe ser un valor entero.
- **Kernel_size:** Especificación de la altura y ancho de la ventana de convolución. Los valores admitidos son un valor entero tomando el mismo valor en las dimensiones espaciales o una tupla/lista que especifique cada tamaño.
- **Padding:** Las entradas posibles son "valid" o "same", que significa que no realiza relleno o que el relleno es uniforme hacia alguna dirección respectivamente. La salida tiene las mismas dimensiones de ancho y alto que la entrada.
- **Activation:** Especifica la función de activación a utilizar. Como la 'relu' que es la unidad lineal rectificada, la 'sigmoid' que implementa la función sigmoide y la 'tanh' que es la tangente hiperbólica.
- **Input_shape:** Solo se aplica si es la capa de entrada, que define el tamaño de la imagen que ingresa a la red neuronal.

La capa *MaxPooling2D* [8] reduce la entrada tomando el valor máximo sobre el tamaño de la entrada definida para cada dimensión en todas las características. Los argumentos establecidos son:

- **Pool_size:** Tamaño de la ventana que se tomará el valor máximo. Los valores admitidos pueden ser un entero, el cual tomaran el mismo valor para las dos dimensiones o una tupla que especifique cada una.
- **Strides:** Especifica el valor de cada paso en que se mueve la ventana.

La capa *Dropout* [9] establece aleatoriamente en 0 unidades de la entrada durante el tiempo de entrenamiento para evitar el sobreajuste. El único argumento que se especificó es *units* que define las dimensiones del espacio de salida.

La capa *Flatten* [10] pasa la entrada a una dimensión, no afecta el tamaño del *batch*. El argumento que tiene es *data_format* que es el orden de las dimensiones en las entradas, se deja por defecto el cuál es (*batch*,

tamaño, canales).

La capa *Dense* [11]en una capa de red neuronal conectada densamente. Implementa la operación salida = *activación*(punto(entrada, *kernel*)+*sesgo*) donde el *kernel* es una matriz de pesos y el sesgo es un vector inclinación por cada capa. Los argumentos especificados son:

- **Units:** Dimensionalidad de la salida, acepta números enteros.
- **Activation:** Determina la activación de la capa, como tanh, simoid o relu.

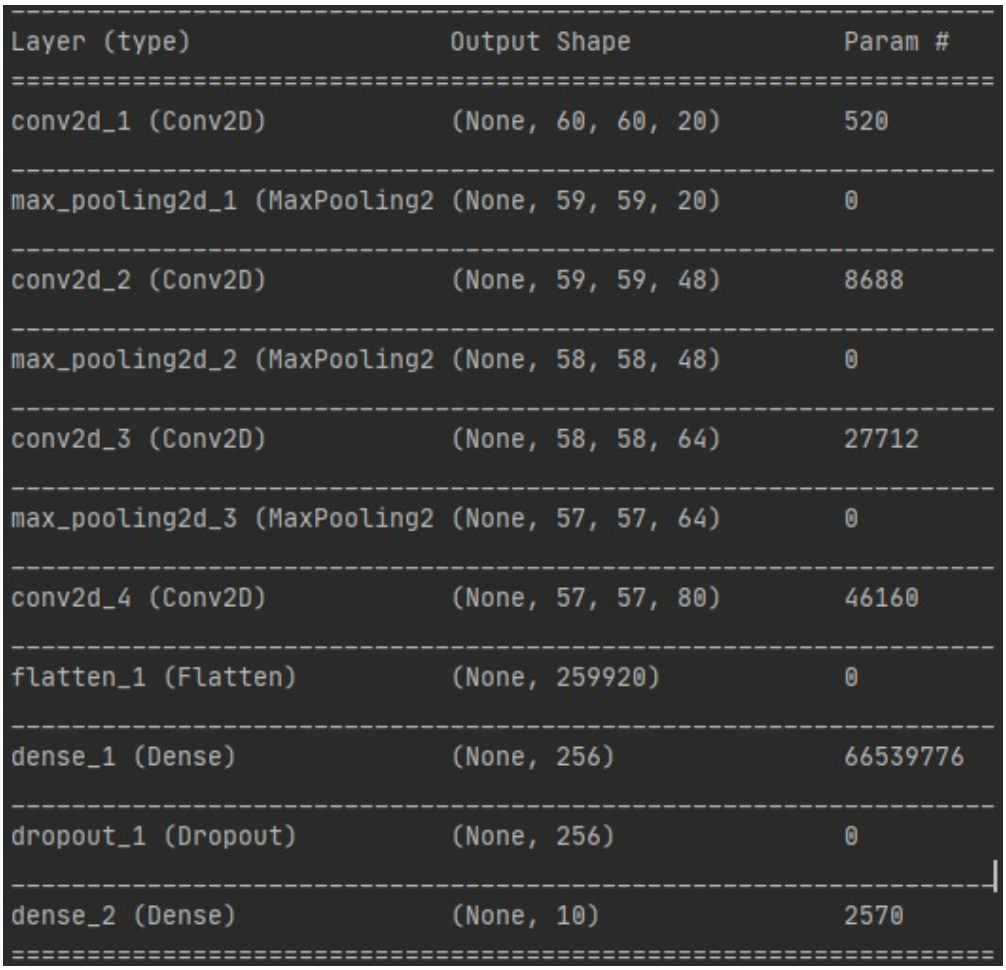
El proceso de entrenamiento se realizó igual para ambos modelos. El número de iteraciones de 1000 y el *batch size* de 128, además del mismo vector de entrenamiento y validación.

4. Resultados

A partir del procedimiento previamente descrito, se implementó cada respectivo modelo con las facilidades ofrecidas por las biliotecas *Keras* y *Tensorflow*. Es posible observar en la figura 3 el resultado que detalla las 6 capas elaboradas con las especificaciones del artículo *Learning Deep Representation for Face Alignment with Auxiliary Attributes*, mientras que en la figura 2 se describe la aplicación del diseño de 3 capas obtenido del notebook *Landmark detection* [2] [5].Las figuras 8 y 9 muestran un diagrama de abstracción de alto nivel de las capas elaboradas para el desarrollo de cada tipología propuesta.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 96, 96, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 48, 48, 32)	0
dropout_1 (Dropout)	(None, 48, 48, 32)	0
flatten_1 (Flatten)	(None, 73728)	0
dense_1 (Dense)	(None, 256)	18874624
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570

Figura 2: Resumen modelo *Landmark detection* [5]



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 60, 60, 20)	520
max_pooling2d_1 (MaxPooling2	(None, 59, 59, 20)	0
conv2d_2 (Conv2D)	(None, 59, 59, 48)	8688
max_pooling2d_2 (MaxPooling2	(None, 58, 58, 48)	0
conv2d_3 (Conv2D)	(None, 58, 58, 64)	27712
max_pooling2d_3 (MaxPooling2	(None, 57, 57, 64)	0
conv2d_4 (Conv2D)	(None, 57, 57, 80)	46160
flatten_1 (Flatten)	(None, 259920)	0
dense_1 (Dense)	(None, 256)	66539776
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570

Figura 3: Resumen modelo *Learning Deep Representation for Face Alignment with Auxiliary Attributes* [2]

Se empleó el dataset *Facial Landmark Detection by Deep Multi-task Learning* el cual contiene 4151 imágenes para realizar la distribución porcentual del conjunto de datos de entrenamiento y validación, que después sería aplicado sobre de los modelos de redes neuronales implementados.[3].

Para obtener un buen resultado se debe ingresar una imagen similar a las de la base, por esta razón que se utilizó el método de *Machine Learning* (ML), conocido como *Haar Cascade*, el cual es una técnica de detección de objetos por medio de una estructura de cascada que filtra espacios para concentrar el procesamiento sobre aquellas ventanas en donde la probabilidad de encontrar un objeto es más alta. Se creó un *bounding box* delimitado por el tamaño facial con 200 pixeles adicionales en cada eje de dimensionamiento, dentro del cual se centro el rostro, siendo este la entrada de la red neuronal.

La figura 4(a) se compara con respecto a la 4(b) en términos de eficiencia a la hora de determinar la ubicación de los puntos de interés faciales de un usuario a un distancia media. Se puede apreciar que ambas detectan con una óptima precisión los ojos, pero la segunda se aproxima ligeramente mejor a la localización de la nariz y comisura de los labios.

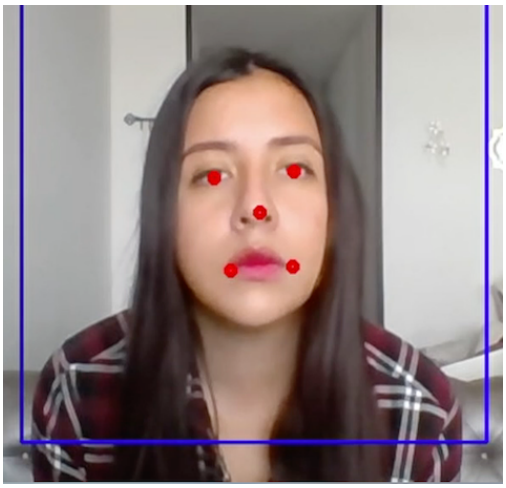
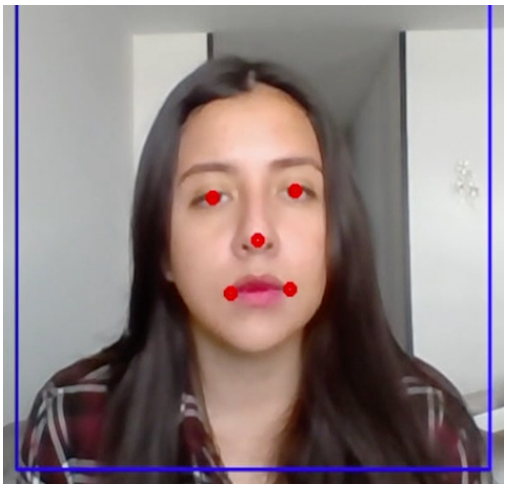
(a)  (b) 

Figura 4: (a) Modelo implementado a partir del artículo *Learning Deep Representation for Face Alignment with Auxiliary Attributes* [2] (b) Modelo extraído de Kaggle [5]

Al analizar la figura 5(a) con la 5(b) se evidencia la presencia del atributo sonrisa a una distancia más cercana, lo cual confirma que en efecto el segundo modelo detecta con mayor exactitud los extremos de la boca, pero el primero tiene un mejor desempeño con los ojos. Ambos conservan el buen pronóstico de la punta de la nariz según el ángulo del rostro.

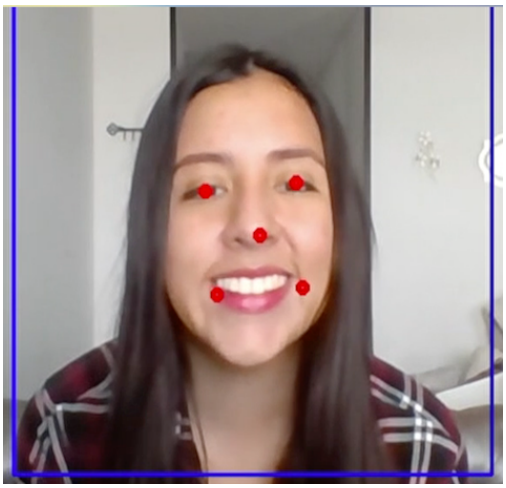
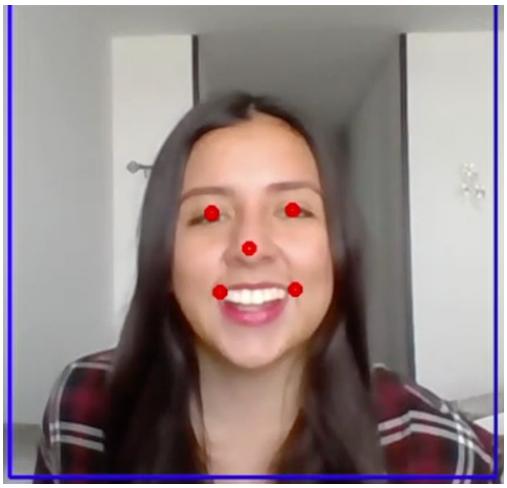
(a)  (b) 

Figura 5: (a) Modelo implementado a partir del artículo *Learning Deep Representation for Face Alignment with Auxiliary Attributes* [2] (b) Modelo extraído de Kaggle [5]

La figura 6(a) comparada con la 6(b) a larga distancia, demuestra que el primer modelo tiene una capacidad superior de hallar los ojos y con respecto a estos, la distribución de las demás marcas de interés. La segunda denota un mayor corrimiento en la detección de los puntos faciales, sin embargo es valido mencionar que ambas presentan notorias falencias.

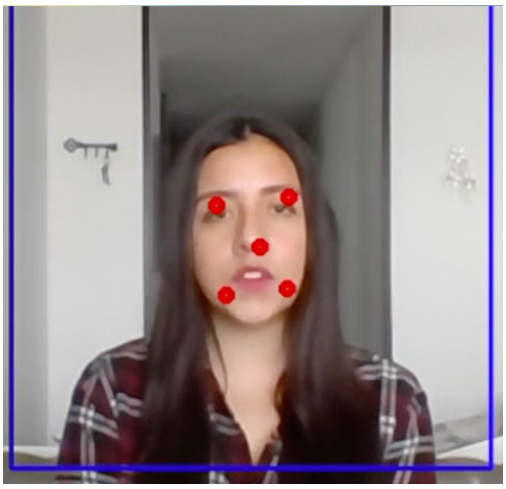
(a)  (b) 

Figura 6: (a) Modelo implementado a partir del artículo *Learning Deep Representation for Face Alignment with Auxiliary Attributes* [2] (b) Modelo extraído de Kaggle [5]

Finalmente evaluamos las fallas de ambos modelos ante las variaciones de distancia como se muestra la figura 7(a) y 7(b), de las cuales es posible concluir que el primer modelo presentó un comportamiento más estable que el segundo, ya que solamente mostró errores excesivos en la detección de los puntos de interés cuando el usuario se situaba demasiado cerca o lejos de la cámara, y además de esto parpadeaba. Por el contrario, el segundo mostró estos mismos inconvenientes, pero dependiendo únicamente del factor distancia, el cual se situó dentro de un rango reducido con respecto al primero.

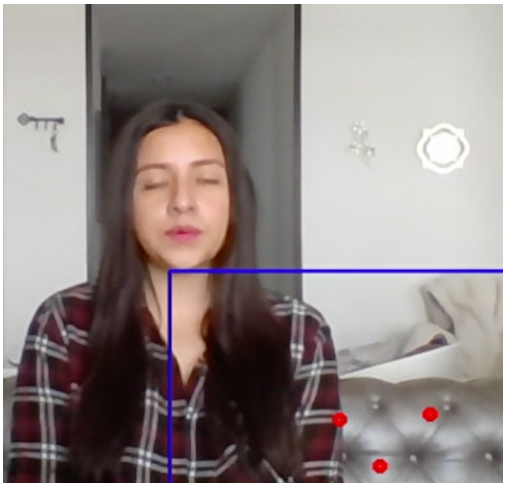
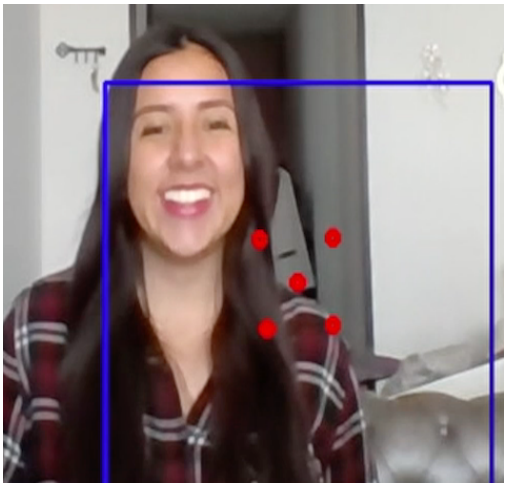
(a)  (b) 

Figura 7: (a) Modelo implementado a partir del artículo *Learning Deep Representation for Face Alignment with Auxiliary Attributes* [2] (b) Modelo extraído de Kaggle [5]

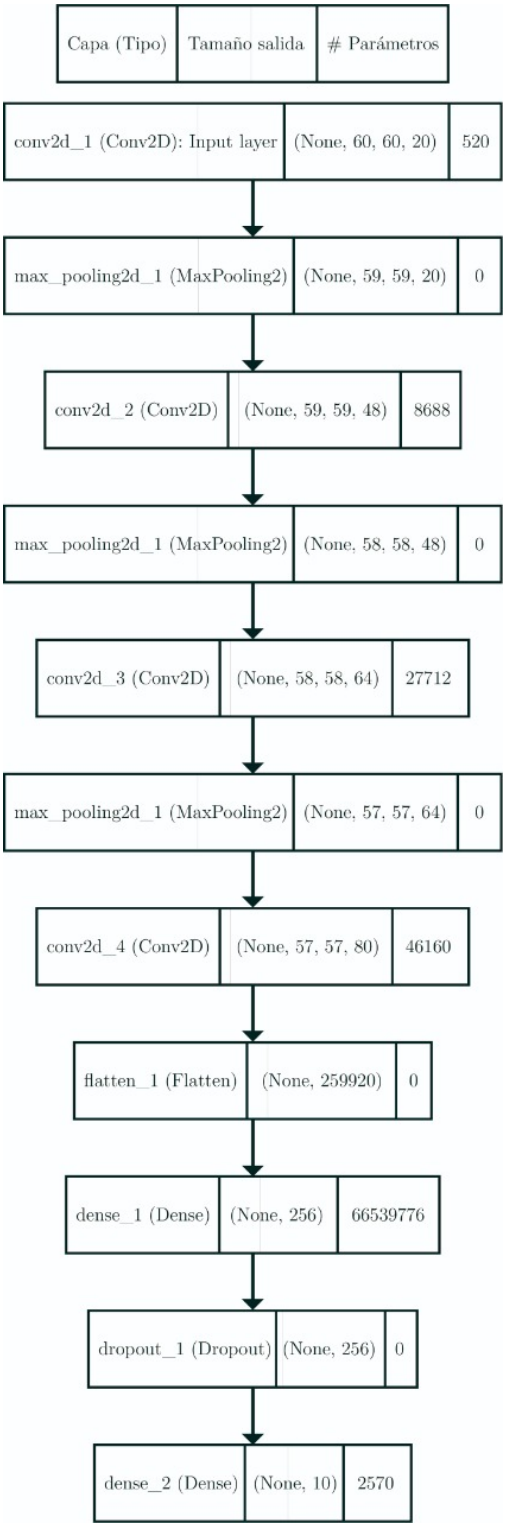


Figura 8: Arquitectura modelo *Learning Deep Representation for Face Alignment with Auxiliary Attributes* [2]

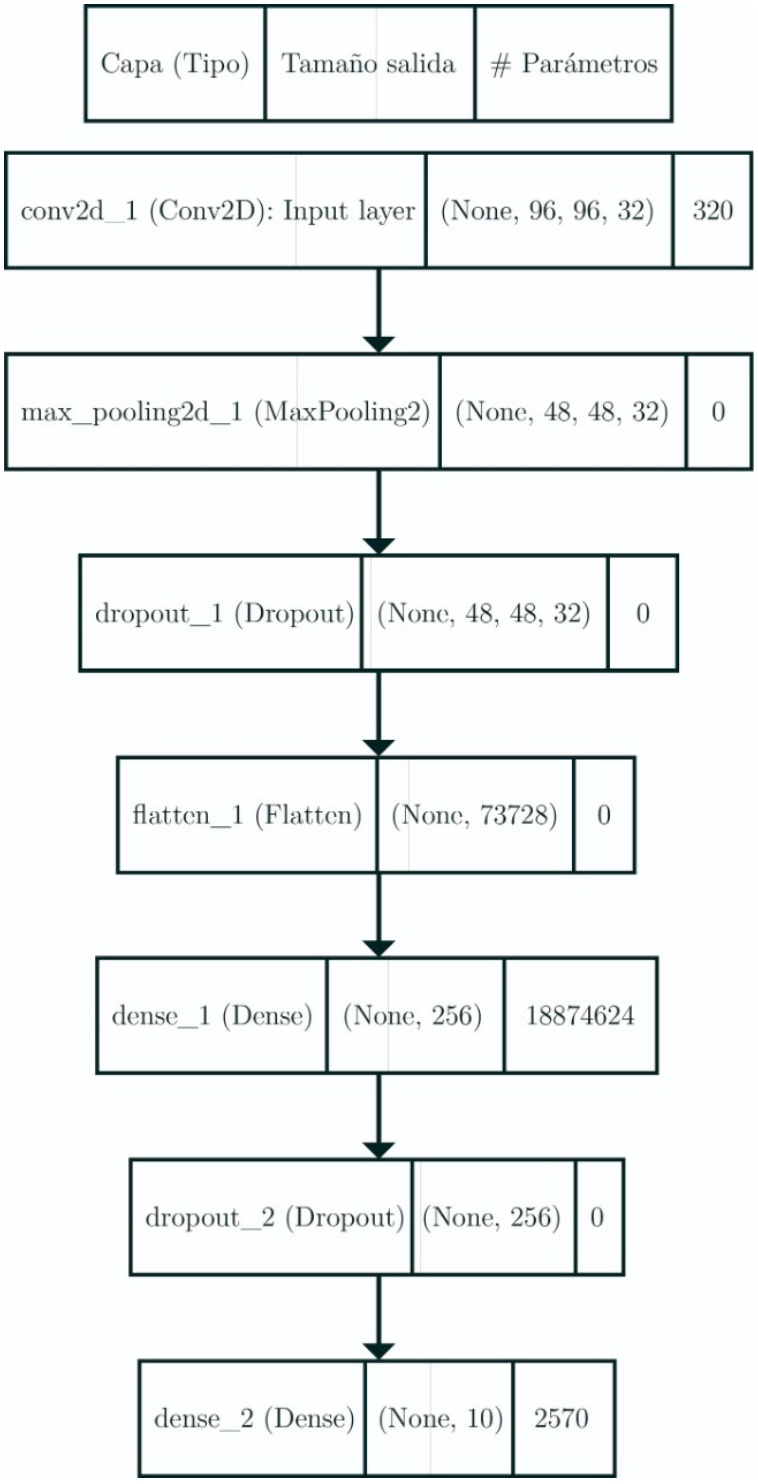


Figura 9: Arquitectura modelo *Landmark detection* [5]

5. Conclusiones

- En el momento de probar los modelos, se dedujo que para obtener un buen resultado las imágenes de entrada de la red neuronal deben ser similares a las de entrenamiento. Es decir, si las delimitamos a la cara del usuario, la red no va a realizar la detección esperada porque debe centrarse en la mitad de la imagen con una ventana lo suficientemente grande que los hombros se visualicen. Debido al método utilizado *Haar Cascade* se identifico el rostro como primer paso y a partir del tamaño, se amplió la ventana 200 píxeles de más.

- Los modelos fueron entrenados con un valor de *epoch* equivalente a 1.000, este término hace referencia a un ciclo a través del conjunto completo de datos de entrenamiento. Es bien sabido que en el proceso de aprendizaje de la red neuronal, a mayor cantidad de épocas en diferentes patrones, mejor será el pronóstico ante una nueva entrada con los datos de prueba. Por lo anterior se afirma que al incrementar este atríbuto, como se ejemplifica en el artículo Learning Deep Representation for Face Alignment with Auxiliary Attributes con 200.000 *epoch*, la calidad de los resultados de detección aumentaría destacablemente.

- La base de datos actualmente trabajada cuenta con 4151 imágenes faciales con sus correspondientes marcas, sin embargo el aumentar esta cantidad contribuiría en la mejora de la fase de entrenamiento y validación, para esto consideramos oportuno como trabajo futuro incluir una mayor densidad de imágenes que además contemplen diferentes variaciones de distancia, ángulos, condiciones de entorno y demás características que abarquen en una mayor proporción todas las posibles condiciones que podrían presentarse a la hora de detectar a un usuario.

- Al probar el seguimiento de los ojos en vídeo de los modelos, se observó que hay un límite máximo y mínimo de distancia del usuario con respecto a la cámara. En el primer modelo, el valor mínimo lo delimitaba el tamaño de la ventana de la imagen que se ingresa a la red, pero la detección seguía siendo válida y el valor máximo fue de 80cm, en este punto los puntos no eran exactos. Para el segundo modelo, el menor fue de 35cm, los puntos son erróneos antes de que la ventana lo delimite y el mayor a los 60cm sucede lo mismo que el anterior.

- Los modelos se entrenaron en las mismas condiciones de base de datos, de número de iteraciones, de tamaño de los *batch*. Como resultado el primero se desempeña mejor, es el que tiene mayor cantidad de capas y además el tamaño de entrada de las imágenes en píxeles es menor, esto permite que pueda realizar una búsqueda más profunda de los objetos.

Referencias

[1] S. Park, X. Zhang, A. Bulling, and O. Hilliges, "Learning to find eye region landmarks for remote gaze estimation in unconstrained settings," *Proceedings of the 2018 ACM Symposium on Eye Tracking Research Applications*, Jun 2018.

[2] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Learning deep representation for face alignment with auxiliary attributes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, p. 918–930, May 2016.

[3] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, *Facial Landmark Detection by Deep Multi-task Learning.* http://mmlab.ie.cuhk.edu.hk/projects/TCDCN.html, 2014.

[4] O. Goldstein, *Face Images with Marked Landmark Points.* https://www.kaggle.com/drgilermo/face-images-with-marked-landmark-points, 2017.

[5] A. Choudhary, *Landmark detection.* https://www.kaggle.com/apurvachoudhary/datasets, 2020.

[6] C. C. L. X. T. Zhanpeng Zhang, Ping Luo, *Facial Landmark Detection by Deep Multi-task Learning.* http://mmlab.ie.cuhk.edu.hk/projects/TCDCN.html, 2014.

[7] know, *Conv2D layer.* https://keras.io/api/layers/convolution$_layers/convolution2d/$.

[8] know, *MaxPooling2D layer.* https://keras.io/api/layers/pooling$_layers/max_pooling2d/$.

[9] know, *Layer activation functions.* https://keras.io/api/layers/activations/.

[10] know, *Flatten layer.* https://keras.io/api/layers/reshaping$_layers/flatten/$.

[11] know, *Dense layer.* https://keras.io/api/layers/core$_layers/dense/$.