ESTIMACIÓN DE POSE DE LA CARA PARA APLICACIONES DE RE-IDENTIFICACIÓN Y BIOMETRÍA BLANDA

Nelson González Machín
Supervisado por: Pedro A. Marín Reyes y José Javier Lorenzo Navarro
Universidad de Las Palmas de Gran Canaria
Junio de 2019

Resumen

En este trabajo se plantea un estudio sobre el rendimiento de las técnicas de biometría blanda, afectadas por las diferentes poses de las personas. En particular, se comprueba el rendimiento de la identificación de sexo y sonrisa para diferentes configuraciones de pose de la cara. Para ello se ha utilizado una base de datos de caras, etiquetadas con la pose, sexo y sonrisa, y con ello se ha entrenado múltiples redes profundas para clasificar caras por su pose como también una red neuronal profunda que estima el grado de la pose respecto a la posición frontal de la cara. El resultado de este trabajo consiste en estudiar cómo el rendimiento de los métodos de biometría blanda de personas se ven influenciados por la pose de la cara.

Abstract

This work presents a study of the importance of the pose of the face for the performance of soft biometrics methods. The performance of the identification of gender and smiling for different poses of the face is going to be considered. To do this, a face database has been used, and a deep neural network has been trained to classify faces by their pose and also a deep neural network that estimates the degree of the pose with respect to the frontal position of the face. The result of this work is to demonstrate that the performance of the methods of soft biometrics of people are influenced by the pose of the face.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Descripción

El rendimiento de la mayoría de los métodos que basan su funcionamiento en la imagen de la cara para obtener características de biometría blanda como puede ser el sexo o la edad, o para la identificación/re-identificación/verificación de personas, se ven bastante influenciados por la pose de la cara [Ding and Tao2016]. En este sentido, cuando la cara está totalmente girada, el resultado de estos métodos puede ser muy deficiente. Por tanto, el conocer la pose de la cara va a permitir aplicar diferentes estrategias, como puede ser disponer de varios métodos que se aplicarían en función del ángulo de rotación de la cara o asignar un valor de certeza o calidad al resultado del método que diferenciará la metodología de entrenamiento en función del ángulo de giro de la cara. Por este motivo, se plantea este Trabajo de Fin de Grado (TFG), partiendo de bases de datos de caras con diferentes ángulos de giro, permitirá obtener un clasificador/estimador del ángulo de la cara. De esta forma, se incorporaría una etapa previa a los métodos de estimación de biometría blanda o de re-identificación de personas basadas en la imagen facial.

1.2. Objetivos

El objetivo de este proyecto es el de obtener un clasificador de la pose de la cara tanto cualitativo como cuantitativo. Para ello se deben cumplir los siguientes objetivos específicos:

- Recopilar y estudiar la bibliografía relativa a la temática.
- Seleccionar el conjunto de datos para las tareas de entrenamiento y pruebas.
- Estudiar el entorno de programación seleccionado.
- Desarrollar métodos de estimación de la pose.
- Evaluar el rendimiento de los métodos desarrollados.

1.3. Competencias

En este trabajo ha sido cubierta la siguiente competencia de la mención de Computación del Grado de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria:

CP04: Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

Esta competencia queda justicada a través de la realización de las tareas de investigación y estudio de las redes neuronales convolucionales (CNN) aplicadas a la estimación de pose, perteneciendo estas tareas a los objetivos generales y específicos de este TFG.

1.4. Planificación inicial

La planificación inicial para el desarrollo de este proyecto está dividida en cuatro fases bien diferenciadas que tienen sus respectivas tareas:

- **Estudio previo/Análisis**:
  1. Recopilación y estudio de bibliografía.
  2. Búsqueda y selección de conjuntos de datos para entrenamiento y pruebas
  3. Estudio y familiarización con el entorno de desarrollo.
- **Diseño/Desarrolo/Implementación**:
  1. Implementación de un estimador de poses considerado como un problema de clasificación (cualitativo)
  2. Implementación de un estimador de poses considerando como un problema de regresión (numérico).
- **Evaluación/Validación/Prueba**:
  1. Realización de pruebas considerada la pose en un número finito de orientaciones.
  2. Realización de pruebas utilizando la estimación en grados de la orientación de la cara.
  3. Estudio del rendimiento de estimaciones de características de biometría blanca en función de la pose.
- **Documentación**:
  1. Redacción de la memoria del TFG.
  2. Realización y ensayos de la presentación del TFG.

1.5. Metodología

La metodología a utilizar para el desarrollo de este proyecto se basada en prototipos, ya que se incorporan funcionalidades a los clasificadores desarrollados en función del rendimiento que se obtiene de los mismos. Esto supone que en algún caso habrá que desechar soluciones para explorar otras diferentes. Este TFG esta centrado en la estimación de pose, es por ello que se adoptarán modelos pre-entrenados; y no incidiremos en su implementación ya que no está entre los objetivos del proyecto.
El desarrollo se desarrolla utilizando el lenguaje de programación Python, para lo que se utilizarán además de los módulos de tratamiento de imágenes del mismo, como puede ser scikit-image, otras librerías Visión por Computador de código abierto como puede ser OpenCV o Dlib. Para la programación de las redes neuronales profundas se emplea la librería Keras, que es una capa que facilita el desarrollo de este tipo de métodos usando como base Tensorflow o Theano, empleando para este TFG el primero de estos.

1.6. Estructura de la memoria

El objetivo de la presente memoria es documentar el proceso que se ha llevado a cabo en el TFG. Este documento esta estructurado en diferentes capítulos y secciones, de tal forma que sigan un orden lógico y coherente de los contenidos que alcanza este este trabajo, en aras de satisfacer al lector. A continuación se detalla cada una de los capítulos del TFG.

En el primer capítulo se da una introducción del proyecto y al problema a abordar. También incluye los objetivos, competencias, planificación metodología y estructura del documento.

En el segundo capítulo se detalla el marco teórico, fundamental para comprender los conceptos y técnicas que se utilizan en este proyecto.

En el tercer capítulo se centra en el estudio previo y el análisis, incluyendo el planteamiento del problema, el análisis y la evaluación de conjuntos de datos que sirvan como entrada para el sistema clasicador/estimador de poses.

En el cuarto capítulo, muestra las diferentes tecnologías que se han utilizado en el proyecto. Luego, se describe cómo se a diseñado el conjunto de datos, y también, cómo se ha implementado los estimadores/clasificadores.

En el quito capítulo, se muestra los resultados del entrenamiento de los estimadores/clasificadores y se evalúan y se prueban su robustez. Para concluir, se evalúa la eficiencia de los métodos de identificación de sexo y sonrisa según la pose determinada por los modelos creados anteriormente.

Y el último capítulo, la síntesis del compendio de ideas extraídas durante la realización del proyecto. Además, se incorpora la valoración personal del trabajo desarrollado.

Capítulo 2

Marco Teórico

2.1. Haar Cascades

Este método se emplea para detectar entidades, es desarrollado por Viola y Jones [Viola et al.2001]. Esta técnica de aprendizaje automático se basa en la utilización de una función cascada, es como un filtro, que es entrenada de forma supervisadas, por ejemplo si se entrenara para la detección de caras, sería necesario dispones de un conjunto de imágenes de caras y otro conjunto donde no figuren imágenes de caras. Este método se basa en la búsqueda de una serie de patrones simples en la imagen, véase **Figura 2.1** .

Imagen extraída de:
`https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html`

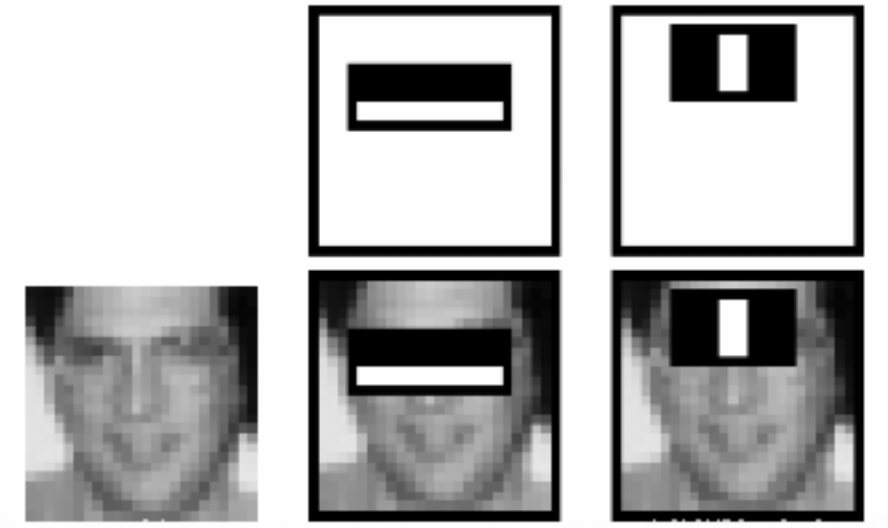


Figura 2.1: Ejemplo haar cascade.

2.2. Redes neuronales

Son un modelo computacional bioinspirados que están conformados por unidades llamadas neuronas artificiales, estas se encuentran conectadas entre sí para transmitirse señales. Para entender el funcionamiento de las redes neuronales, es importante conocer primero el funcionamiento de un perceptrón simple [Rosenblatt1958]. Este modelo está compuesto por una capa de neuronas de entrada y otra de salida.

Esta arquitectura recibe una serie de entradas, que son ponderadas utilizando los pesos asociados a cada una de las conexiones. Si el resultado es superior al valor umbral o bias, la neurona se activa. La principal desventaja que representa este modelo es que solo pueden representar funciones linealmente separables.

Para solucionar el problema de los perceptrones simples aparecieron los perceptrones multicapa, que se diferencian de los anteriores en que por lo menos tienen una capa intermedia entre la capa de entrada y la capa de salida. Estas capas intermedias se denomina "capas ocultas". La ventaja de añadir estas capas a nuestro modelo es que aumenta el espacio de hipótesis que puede representar la red [Russell and Norvig2016], por lo tanto, puede tratar problemas no lineales.

Los perceptrones multicapa, al igual que otros modelos neuronales, utilizan una técnica de aprendizaje supervisado denominada backpropagation [Hecht-Nielsen1992]. Esta técnica permite modificar los pesos de todas las capas tanto para las capas de entrada y salida como para las capas ocultas. Esta técnica se utiliza junto a métodos de optimización, como puede ser el Descenso por Gradiente Estocástico (SGD). Una vez obtenida la salida de la red, se compara con el resultado de salida real haciendo uso de una función de coste, y así, obteniendo el error de la red. Una vez obtenido el error se propaga hacia atrás (backpropagation) hacia la capa de neuronas anterior, para ajustar los pesos de cada una de ellas, que generalmente son inicializadas de forma aleatoria. El error se sigue propagando hacia atrás hasta llegar a la capa de entrada. Este proceso se realiza con todas las instancia del conjunto de entrenamiento. El aprendizaje de los pesos de las capas ocultas permiten a las redes aprender nuevas características del conjunto de datos.

La redes neuronales, actualmente son utilizadas debido a que tienen la peculiaridad de poder aprender, en lugar de ser programados de forma explícita. Se utilizan bastante en el campo de la medicina para apoyar al médico en el diagnóstico y tratamiento del paciente.

2.2.1. Red completamente conectada

Son un tipo de red neuronal, formado por varias capas cuyas capas están formados por unidades de procesamiento denominadas neuronas donde cada neurona de cada capa está conectada a todas la neuronas de la siguiente capa.

2.2.2. Red Neuronal Convolucional

Las redes neuronales convolucionales son un tipo red neuronal profunda que se utiliza normalmente para procesar imágenes o vídeos debido a su alta dimensionalidad. Es por ello que se utiliza para problemas de detección/clasificación de objetos, categorización de escenas, clasificación de imágenes en general, reconocimiento facial etc.

Unas de las primeras redes convolucionales fue LeNet desarrollada por Yann LeCun [LeCun et al.1998], que eran simples pilas de convoluciones para la extracción de características y operaciones de max-pooling para submuestreo espacial. Esta red se usó principalmente para el reconocimiento de caracteres, como la lectura de códigos postales, dígitos, etc. En 2012, con el avance del poder de cómputo de los actuales ordenadores, estas ideas se refinaron en la arquitectura AlexNet [Sutskever et al.2012], donde las operaciones de convolución se repetían varias veces entre las operaciones de max-pooling, lo que permite a la red aprender características más complejas a diferente escala espacial. Dado los buenos resultados obtenidos, esta arquitectura precipitó una tendencia a hacer este estilo de red cada vez más profundo, principalmente impulsado por la competencia anual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky et al.2015] que es una competición que evalúa algoritmos para detección y clasificación de imágenes a gran escala.

El nombre de las redes convolutivas es debido a la operación convolución que utilizan estas redes para extraer las características de una imagen de entrada. Las redes convolutivas están formadas por una secuencia de capas. Estas capas tienen neuronas organizadas en tres dimensiones ancho, alto y profundidad, a diferencia de las redes completamente conectadas.

Las capas más destacables que se utilizan en estas redes son las siguientes:

1. **Capa de convolución**: Contiene un conjunto de filtros denominados kernels, que realizan la operación de convolución a una entrada dada para generar una salida, a lo que se le llama un mapa de características.

2. **Capa de grupación**: Esta capa trabaja por bloques sobre el mapa de características, con el objetivo de combinar características y reducir sus dimensiones para que cuando actúe como entrada en la siguiente capa reducir el número de operaciones. Esta operación de combinación se lleva a cabo con una función de media o de valor máximo. La media obtiene todos los valores del bloque y obtiene la media y la de máximo valor coge el mayor valor del bloque.

3. **Capa completamente conectada**: Cada neurona de cada capa está conectada a todas la neuronas de la siguiente capa.

2.2.3. Descripción de Redes Neuronales Convolucionales

En está sección se va a revisar diferentes arquitecturas de redes convolucionales.

2.2.3.1. MobileNet

El modelo MobileNet [Howard et al.2017] está basado en convoluciones profundamente separables que es una forma de convoluciones factorizadas, donde se factoriza una convolución estándar entre una convolución profunda y una convolución puntual 1×1. Para MobileNet, la convolución profunda aplica un único filtro a cada uno de los canales de entrada. La convolución puntual después aplica una convolución 1×1 para combinar las salidas de la convolución profunda. Una convolución estándar filtra y combina las entradas en un conjunto de salidas en un solo paso. La convolución profunda separable divide esto en dos capas, una para filtrar y otra para combinar. Esta factorización tiene el efecto de reducir drásticamente la computación y el tamaño del modelo, en la **Figura 2.2** se detalla la estructura de MobileNet.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figura 2.2: Arquitectura MobileNet [Howard et al.2017]

2.2.3.2. VGG16

Este modelo fue desarrollado en el trabajo de investigación de [Simonyan and Zisserman2014] sobre el efecto de la profundidad de las redes convolucionales sobre su precisión en la configuración de reconocimiento de imagen a gran escala. Dicho trabajo fue realizado por el equipo Visual Geometry Group (VGG). La VGG16 está estructurada por capas convolucionales que usan filtros con campo receptivo 3×3, usando un desplazamiento de un píxel y un padding de uno. La red tiene 5 capas de agrupación que utiliza un kernel de 2×2. Su arquitectura la podemos ver en la **Figura 2.3**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 2.3: Arquitectura VGG16 [Simonyan and Zisserman2014]

2.2.3.3. Xception

Es una red neuronal convolucional [Chollet2017] profunda inspirada por Inception [Szegedy et al.2015], donde cada módulo Inception ha sido reemplazado por convoluciones profundamente separables. Tiene una arquitectura de 36 capas convolucionales que forman la base de extracción de características de la red. Las 36 capas convolucionales están estructuradas en 14 módulos. La arquitectura de Xception es una pila lineal de capas convolucionales profundamente separables. Su arquitectura la podemos ver en la **Figura 2.4**

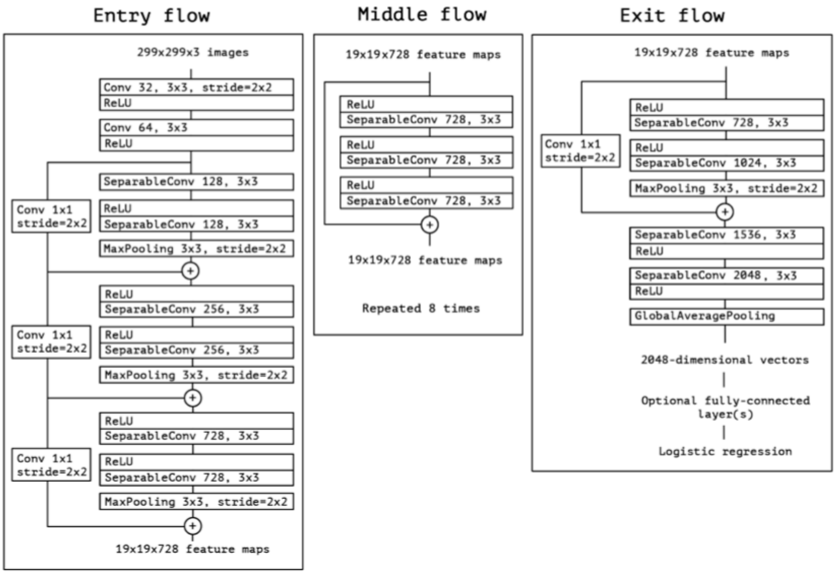


Figura 2.4: Arquitectura Xception [Chollet2017]

2.2.3.4. NASNetLarge y NASNetMobile

NASNetLarge y NASNetMobile [Zoph et al.2018] son arquitecturas creadas por una inteligencia artificial llamada AutoML desarrollada por Google. Esta inteligencia artificial es capaz de encontrar una arquitectura óptima para un problema concreto. Este sistema prueba varias arquitecturas y se queda con la mejor. Esta inteligencia artificial aprende con la experiencia, es decir, si se encuentra con un problema que ya ha resuelto con una determinada arquitectura es muy probable que pruebe primero con dicha arquitectura antes que otras.

2.3. Validación por Holdout

Este método consiste en dividir el conjunto de datos en dos conjuntos de forma aleatoria, el conjunto de entrenamiento y el conjunto de test. El conjunto de entrenamiento será usado para el entrenamiento de la red neuronal y por consiguiente se le pedirá a la red entrenada que prediga los valores de salida del conjunto de test. El error que cometa será anotado para reducir la varianza y mejorar la estimación. Este proceso se repite varias veces y luego se saca la media del error. Este método es utilizado para determinar el rendimiento del modelo. Podemos ver un ejemplo en la **Figura 2.5**

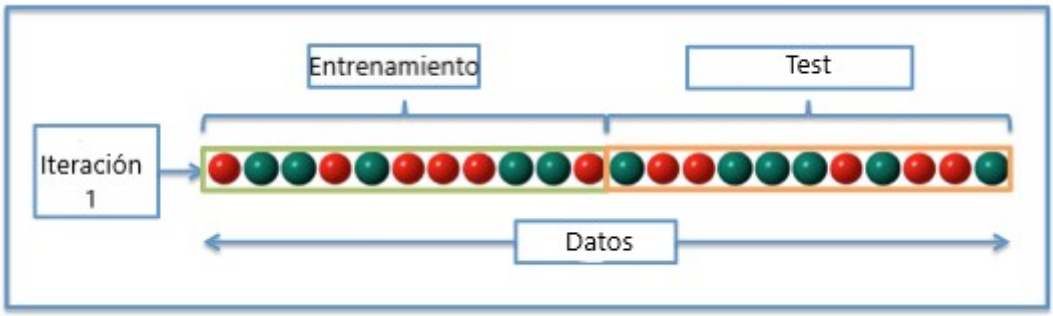


Figura 2.5: Una iteración de holdout

2.4. Trabajos similares

Antes de comenzar a desarrollar el trabajo, se buscó trabajos con alguna similitud con el proyecto. A continuación se nombrará y se explicará los puntos destacados de los trabajos análogos a este.

1. **Face recognition in video streams for mobile assistive devices dedicated to visually impaired** [Tapu et al.2018]: Este artículo de investigación propone un sistema de detección y reconocimiento de cara utilizando redes convolutivas, diseñado para mejorar la interacción y comunicación de los usuarios con discapacidad visual en encuentros sociales.

El módulo del sistema que se encarga del reconocimiento facial, utiliza una red VGG16 que produce una salida de dimensión de 4,096 que representa las características de la cara extraída de una imagen. Después, para cada característica facial se le asigna un peso que se corresponde a su relevancia para la representación global de la cara. Usando un esquema de adaptación de peso, el sistema puede tener en cuenta diversas variaciones de postura, desenfoque, compresión o movimiento existente en el flujo del vídeo.

Para determinar tales pesos, se ha entrenado una red VGG que asigna a una instancia de una cara si es relevante o irrelevante. La salida de la red es una probabilidad de que la instancia de la cara pertenezca a la clase relevante.

El símil de este trabajo con el nuestro es que utiliza características de biometría blanda para una mejora del rendimiento del reconocimiento facial.

2. **Real-Time Face Pose Estimation with Deep Learning** [Gualbertob]: Es un trabajo,echo por Arnaldo Gualberto [Gualbertoa], donde explica cómo desarrolló un estimador de la rotación de la cara sobre tres ejes llamados roll, pitch, and yaw, **Figura 2.6**, obteniendo un estimador que en la fase de test tuvo un error de 39.92 usando la función del error cuadrático medio.

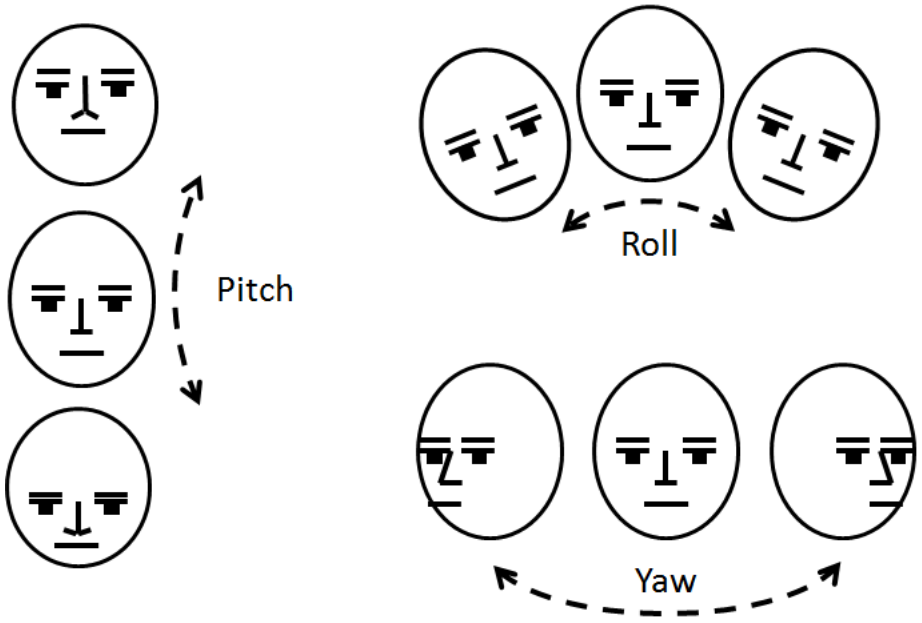


Figura 2.6: Roll, Pitch, and Yaw
Imagen extraída de: `https://medium.com/analytics-vidhya/face-pose-estimation-with-deep-learning-eebd0e62dbaf`

3. **Shot Classication and Keyframe Detection for Vision based Speakers Diarization in Parliamentary Debates [Marín-Reyes et al.2016]:** Este artículo propone el uso de imágenes claves para la re-identificación de personas. Se propone una medida basada en las relaciones antropométricas de los elementos faciales con el fin de seleccionar las caras claves. De esta forma se procesarán exclusivamente caras que tengan una pose frontal y que estén mirando al frente. De esta forma se consigue una mejora significativa frente a los métodos que incluyen múltiples poses.

Capítulo 3

Análisis

Para cumplir con los objetivos descritos en la Sección **1.2** se ha investigado diferentes conjuntos de datos y se ha elegido el más adecuado para el problema.

3.1. Bases de datos interesantes

Se ha analizado multiples conjuntos de datos relacionados con estimación de pose, como son:

- **The BioID Face Database** [BioID]: Es una base de datos que contiene 1,521 imágenes de personas en escala de grises con resolución 384×286 con formato ".pgm", distribuidas entre 23 personas diferentes. Contiene imágenes en escenarios no controlados con diferentes iluminaciones. La mayoría de las fotos son tomadas en lugares muy semejantes y carece de variedad de personas, es decir, personas de diferentes etnias. BioID tiene manualmente marcado la posición de los ojos de cada imagen.

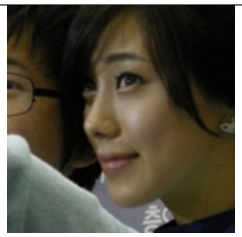
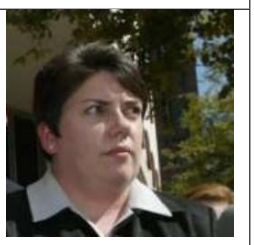
- **Caltech 10,000 Web Faces** [Cal]: es una base de datos que contiene 10,000 imágenes, aproximadamente 10,000 personas diferentes. Tiene una amplia variabilidad de imágenes. Proporciona la posición de cuatro rasgos faciales de cada imagen, pero carece de la identidad de los individuos. Por tanto, no es particularmente adecuado para experimentos de reconocimiento facial.

- **Multi-task facial landmark (MTFL) [MTF]**: es un conjunto de datos con 12,995 imágenes de poses de cabezas. Algunas de estás imágenes poseen la identidad del individuo,por tanto, es adecuado para experimentos de reconocimiento facial. Además, contiene imágenes en escenarios no controlados con diferentes iluminaciones. Las fotos son tomadas en diferentes lugares y posee una gran variedad de personas, es decir, personas de diferentes etnias, proporcionando al conjunto de datos una gran complejidad.

3.2. Análisis y Evaluación del Conjunto de Datos

Para la realización de este proyecto se ha optado por el conjunto de datos denominado MTFL por su alta complejidad, su gran variedad de poses y su gran variedad de personas de diferentes etnias.

El conjunto de datos MTFL contiene 12,995 imágenes de caras, cada una viene anotada con sus cinco puntos de referencias faciales, sexo, si está sonriendo o no, si lleva puesto gafas y la pose de la cabeza.

P.Izquierdo			
Izquierda			
Frontal			
Derecha			
P. Derecho			

Cuadro 3.1: Imágenes de la base de datos MTFL

La estructura del conjunto de datos viene pre-definida, entrenamiento y testeo. Las anota-

ciones están guardadas en archivos de texto (training.txt and testing.txt). Cada registro en el fichero de datos corresponde a una imagen de cara. El formato es el siguiente:

1. $x_1 \cdots x_5, y_1 \cdots y_5$: las localizaciones para el ojo izquierdo, ojo derecho, nariz, esquina izquierda de la boca, esquina derecha de la boca.
2. Género: masculino (1), femenino (2).
3. Sonrisa: sonriendo (1), sonriendo (2).
4. Pose de la cabeza: perfil izquierdo (1), izquierda (2), frontal (3), derecha (4), perfil derecho (5).

El conjunto de datos está formado por 9,270 imágenes con poses frontales y 3,725 imágenes con otras poses. Entre las otras poses, 1,563 tienen una pose hacia la izquierda, 329 muestran una pose extremadamente hacia la izquierda, 1,512 tienen una pose hacia la derecha y 321 muestran una pose extremadamente hacia la derecha. Podemos ver la cantidad de imágenes por clase en el histograma de la **Figura 3.1**.

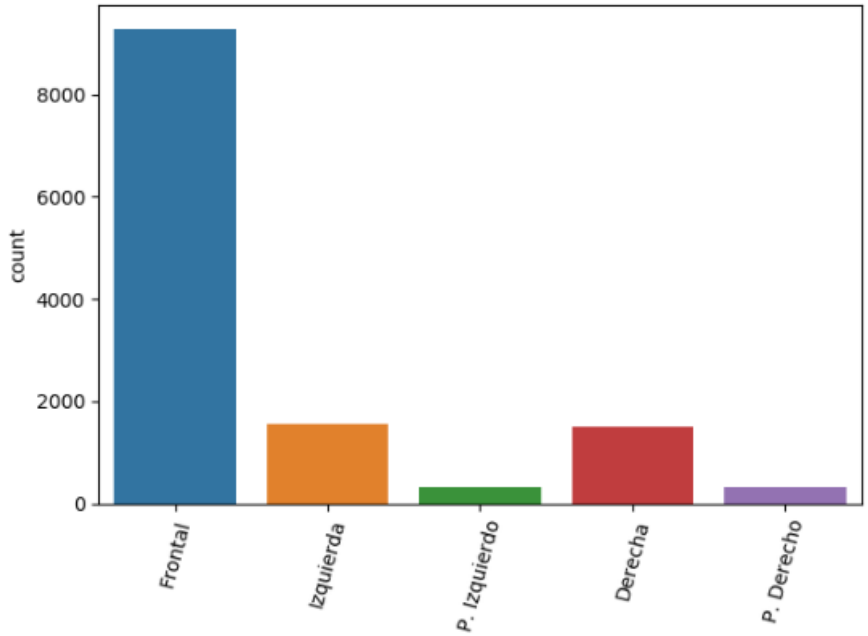


Figura 3.1: Número de imágenes por clase

3.2.1. Características del conjunto de datos MTFL

Las características principales que radican en este conjunto de datos son:

- El conjunto de datos incluye variaciones en la pose, iluminación, expresión, fondo, etnia, edad, sexo, ropa, peinados, calidad de cámara, color de la saturación, foco, y otros paráme-

tros con el objetivo de tener variedad de caras en condiciones diferentes con el propósito de que el modelo que se emplee sea capaz de generalizar.

- Algunas imágenes contienen más de una cara, pero la cara a identificar es la que ocupa la región central de la imagen.
- El nombre de la persona está en el título de la imagen para las imágenes que están en la carpeta "flw_5590", las restantes no los tiene.
- Hay algunas personas que tienen dos o más imágenes en el conjunto de datos. Las imágenes con el mismo individuo suelen estar tomadas en diferentes lugares y diferente momento. Por ejemplo, una persona quizás está fotografiado durante un evento deportivo y en una entrevista.
- Todas las imágenes están en formato JPEG. Las imágenes que están en la carpeta "flw_5590" son de 250×250 píxeles, las que están en "AFLW" son de 150×150 y las que están en "net_7876" son de 160×216.
- La mayoría de las imágenes están en el espacio de color RGB.

Capítulo 4

Diseño e implementación

En este capítulo se describirá las tecnologías utilizadas, el diseño del conjunto de datos y cómo se ha implementado los clasificadores/estimadores haciendo uso de redes neuronales convolutivas.

4.1. Tecnologías

A continuación se describirá las herramientas utilizadas durante este proyecto.

- **Python:** Es un lenguaje de programación de código abierto, interpretado. Soporta programación orientada a objetos, operación imperativa y, en menor medida, programación funcional, por lo tanto, se trata de un lenguaje de programación multiparadigma. Actualmente es uno de los lenguajes más populares en el ámbito de la investigación ya que dispone de múltiples librerías estadísticas y numéricas con las que es compatible; y por ello se ha decidido utilizar Python para realizar este proyecto.
- **Keras:** es una API de redes neuronales de alto nivel escrito en Python y capaz de ejecutarse sobre Tensorflow, CNTK o Theano. Fue desarrollado con el objetivo de permitir una rápida experimentación con redes neuronales. También permite ejecutar programas que usen GPUs.
- **NumPy:** Consiste en una librería de código abierto disponible en Python que permite definir tensores multidimensionales, y además proporciona funciones matemáticas para operar entre ellos.
- **Pycharm** Es un entorno de desarrollo integrado (IDE) para desarrollar programas en Python desarrollado por la compañía JetBrains. Todos los programas que se ha desarrollado en este proyecto han sido desarrollado en este entorno.
- **Overleaf:** Es un servicio web que permite la edición colaborativa de documentos utilizando LaTex. Se pueden iniciar documentos usando plantillas ya definidas y modificarlas para

el beneficio propio. Ha sido utilizada para la realización de esta memoria ya que permite compartir el estado de desarrollo del documento.

- **OpenCV:** es una librería con una gran cantidad de algoritmos relacionados con la Visión por Computador y el Aprendizaje Automático. Actualmente , OpenCV admite una amplia variedad de lenguajes de programación como C++, Python, Java, etc. Y están en desarrollo activo para mejorar las operaciones haciendo uso de GPU.

- **MTCNN:** es un detector de cara desarrollado en Python. Fue escrito desde cero, utilizando como referencia la implementación de MTCNN de David Sandberg (MTCNN de FaceNet) en Facenet. Se basa en el artículo de [Zhang et al.2016].

- **Pandas:** es una librería de Python desarrollada para el análisis de datos y estructuras de sencilla utilización.

- **Seaborn:** es una librería de visualización de python basada en Matplotlib. Proporciona una interfaz de alto nivel para mostrar o renderizar gráficas atractivas e informativas.

- **Amazon Rekognition:** es un servicio que ofrece amazon para el análisis de vídeo e imágenes.

- **Scikit-learn:** es una librería de código abierto para el lenguaje de programación Python. Incluye varios algoritmos de machine learning entre los cuales están las máquinas de vectores de soporte, bosques aleatorios, Gradient boosting, y K-means.

4.2. Diseño del conjunto de datos

Antes de la implementación de los clasificadores/estimadores se diseñó el conjunto de datos de forma adecuada para que el entrenamiento de los clasificadores/estimadores sea la más eficiente y adecuado para los recursos que se tenía.

4.2.1. Redimensionando las imágenes

Como vimos en la Sección **3.2.1**, las imágenes del conjunto de datos MTFL tienen diferentes tamaños. Entonces se decidió adaptar todas las imágenes a un tamaño 100×100 para reducir la computación de las imágenes. Para ello, se utilizó la librería OpenCV haciendo uso de una función llamada *resize()* en la cual le debes pasar como parámetro la imagen y el tamaño que deseas de la imagen, lo podemos ver en la **Figura 4.1**

```
cv2.resize(img, dsize=(100, 100))
```

Figura 4.1: Método para cambiar tamaño de imagen.

4.2.2. Aumento de datos

Antes de entrenar los modelos para obtener resultados, es importante balancear las clases. Como se muestra en la **Figura 3.1** las clases ”frontal” respecto a las otras poseen un gran número de imágenes. También, tanto la clase ”perfil izquierdo” y la clase ”perfil derecho” poseen muy pocas imágenes. Tras ver esto se decidió reducir la clase ”frontal” a 4,500 imágenes y aplicar aumento de datos en las otras clases. La técnica utilizada para aumentar el número de imágenes de las clases, que no son frontales, ha sido la de reflexión horizontal ya que al aplicarle esta técnica a una imagen de un perfil izquierdo de una cara obtendré el perfil derecho de una cara. Podemos ver esto en la **Figura 4.2**.

Figura 4.2: Aplicando reflexión horizontal

Después de haber aplicado la técnica reflexión horizontal y la reducción de imágenes para la clase ”frontal” obtenemos la siguiente distribución, **Figura 4.3.**

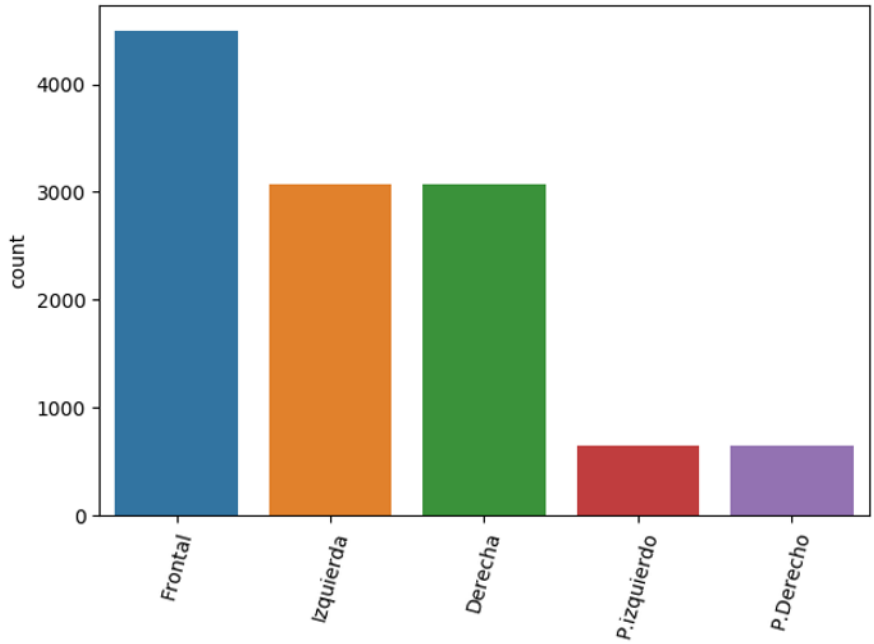


Figura 4.3: Datos aumentados y reducidos.

4.2.3. Etiquetado para el problema de clasificación

En este proyecto, se implementó 3 tipos de clasificadores. Cada clasificador se iba a encargar de clasificar un número diferente de clases. Los clasificadores a obtener son los siguientes:

1. Clasificador de caras frontales y no frontales.

2. Clasificador de caras frontales, derecha e izquierda.

3. Clasificador de caras frontales, perfil derecho, perfil izquierdo, izquierda y derecha.

Por tanto, se ajustaron las etiquetas del conjunto de datos según el número de clases a clasificar. Se desarrolló un programa Python para que hiciera los cambios. Para el clasificador de dos clases, se decidió identificar la clase no frontal con un cero y la clase frontal con un uno, para el clasificador de tres clases, se decidió identificar las caras frontales con un uno, las caras derechas con un dos y las caras izquierdas con un cero y para terminar, para el clasificador de cinco clases, se decidió identificar las caras frontales con un dos, el perfil izquierdo con un cero, izquierda con un uno, perfil derecho con un cuatro y derecha con un tres.

4.2.4. Etiquetado para el problema de regresión

Para este problema de regresión se decidió crear un estimador que devuelva el grado de la pose de la cara respecto a la vertical de su cuello, a este grado lo vamos a nombrar como 'Yaw'. Sin embargo, el conjunto de datos MTFL no están etiquetados con el grado de su pose, por eso se utilizó el servicio que proporciona Amazon Web Services (AWS) llamado "Amazon Rekognition". Este servicio analiza las caras de una imagen dada, tras el análisis, devuelve varios datos y entre ellos se encuentra el valor de Yaw que cuyo valor puede ser entre [-90, 90] grados.

Para analizar todas las imágenes del conjunto de datos, se tuvo que crear un programa Python que enviara las fotos al servidor de Amazon Rekognition una a una. En la **Figura 4.4** se muestra un código para el análisis de una imagen usando Amazon Rekognition. Hay que destacar que el análisis de imágenes que ofrece AWS no es gratis. Si eres nuevo en la plataforma te dan una capa gratis que te permite analizar 5,000 imágenes gratis por mes. En el caso de que no tengas tal oferta, cada 1,000 imágenes te cobrarán 1$.

```python
import boto3
import csv

with open('nelsonkey.csv', 'r') as input:
    next(input)
    reader = csv.reader(input)
    for line in reader:
        access_key_id = line[2]
        secret_access_key = line[3]

photo = '4.jpg'
client = boto3.client('rekognition',
                      aws_access_key_id = access_key_id,
                      aws_secret_access_key = secret_access_key)

with open(photo, 'rb') as source_image:
    source_bytes = source_image.read()

response = client.detect_faces(Image={'Bytes':source_bytes},
                               Attributes=['ALL'])
print(response)
print(len(response['FaceDetails']))
```

Figura 4.4: Programa de análisis de cara

Después de etiquetar cada imagen, se comenzó con el análisis de los datos obtenidos. Como podemos observar en la **Figura 4.5** hay muchas imágenes entre [-20 y 20] grados. Entonces, se decidió hacer un programa en Pyhton que redujera la cantidad de imágenes a 300 para aquellos intervalo de grados que tuvieran más de 300 imágenes. El resultado lo podemos observar en la **Figura 4.6**

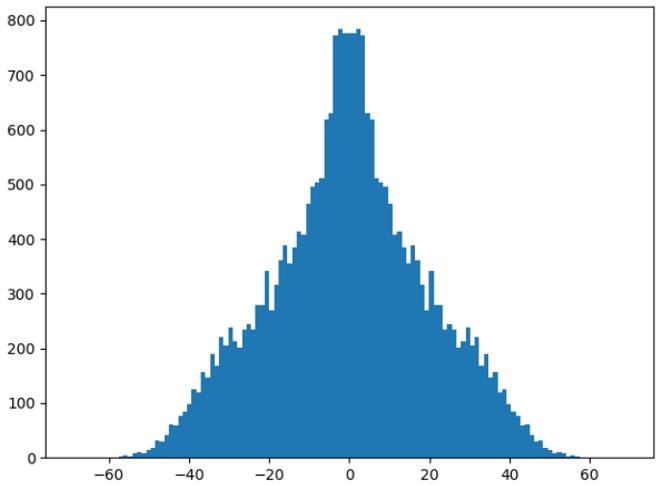


Figura 4.5: Cantidad de imágenes por intervalos de un grado.

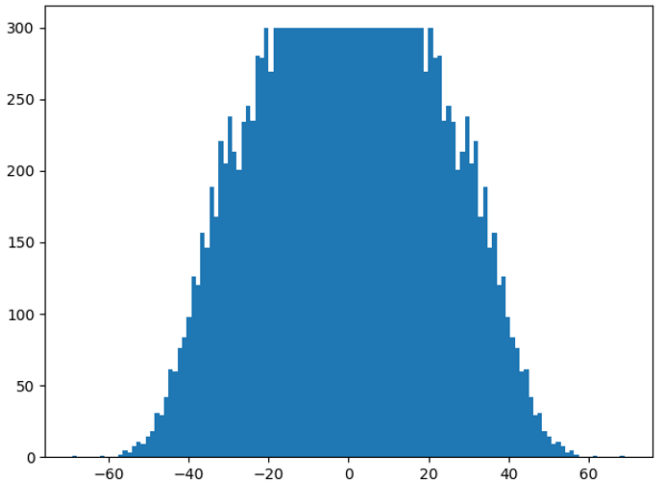


Figura 4.6: Cantidad de imágenes por intervalos de un grado reducido.

Por consiguiente, como se iba a probar el modelo de regresión con diferente función de activación se tuvo que ajustar los yaw al rango de valores de cada función de activación. Como podemos ver en la **Figura 4.11**, cada función de activación devuelve un rango de valores diferentes. Entonces, para cada modelo se tuvo que ajustar los Yaw de las imágenes para diferentes rangos, por ejemplo, para una función de activación sigmoide, se tuvo que ajustar los valores de Yaw entre cero y uno, y en el caso de fuera una función de activación tangente hiperbólica, se tuvo que ajustar los valores de Yaw entre menos uno y uno.

4.3. Implementación de los modelos

En esta sección se describirán las diferentes implementaciones realizadas en este TFG. Esta fase de implementación está dividida en las siguientes tareas:

1. Implementar clasificadores de poses para dos, tres y cinco clases.
2. Implementar estimadores de poses según el grado de rotación de la cabeza con diferentes funciones de activación.

4.3.1. Implementación de modelos de clasificación

Para la implementación de los clasificadores se ha optado por el uso de redes neuronales convolucinales debido a que son bastantes eficaces cuando se trabaja con imágenes. Keras proporciona varias redes convolucionales, por tanto, se miró la documentación de Keras para seleccionar y testear varias redes neuronales convolucionales, para comprobar cual de ellas funciona mejor para este problema. Las siguientes redes neuronales convolucionales fueron elegidas, ya que en la competición ILSVRC dieron buenos resultados:

- VGG16
- Xception
- MobileNet
- NASNetLarge
- NASNetMobile

La implementación se hizo en cuatro pasos:

- Cargar y configurar la red neuronal convolucional.
- Personalizar la capa completamente conectada.
- Seleccionar función de error.
- Seleccionar optimizador.

4.3.1.1. Carga y configuración de la red neuronal

En la **Figura 4.7** podemos ver los parámetros utilizados para cargar el modelo VGG16. El parámetro $include\_top$ se ha puesto a falso para que no se añada una capa completamente conectada al final de la red ya que se va a personalizar, $weights = None$ para que inicialice los pesos de la red de forma aleatoria, $input\_tensor = None$ ya que no se le va a pasar como entrada un tensor Keras, $input\_shape = (100, 100, 3)$ que son las dimensiones de las imágenes que le vamos a introducir como entrada de la red, *pooling=None* para que no añada capas de agrupación entre las capas convolucionales y *classes=2* ya que se iba a probar como funciona el modelo con dos clases, "Frontal" y "No frona". Para los otros modelos convolucionales se cargaron de la misma manera pero utilizando sus funciones correspondientes.

```
model = vgg16.VGG16(include_top=False, weights=None, input_tensor=None,
                    input_shape=(100, 100, 3), pooling=None, classes=2)
```

Figura 4.7: Creando VGG16

4.3.1.2. Capa completamente conectada personalizada

El modelo carece de capa completamente conectada, por tanto, se le añadió una capa completamente conectada. El número de neuronas de dicha capa tendrán tantas neuronas como de clases que se vaya a clasificar. Y después de dicha capa, se le ha añadido una capa de activación softmax. Lo que hace esta capa es que la salida sea un vector de probabilidades, cuya suma es 1, en el cual indica la probabilidad de que la entrada pertenezca a cada clase. Asimismo, se colocó

una capa de normalización por lotes antes de la última capa de activación, **Figura 4.8.** Lo que hace la normalización por lotes es normalizar la salida de la capa anterior restando la media y dividiendo por la desviación estándar.

```
dense_layer = Dense(2, name='dense_layer')(reshape_layer)
dense_layer = BN()(dense_layer)
dense_layer = Activation('softmax')(dense_layer)
```

Figura 4.8: Capa normalización por lotes.

4.3.1.3. Función de error

La función de error que se eligió para los modelos de clasificación fue la función de error *categorical cross-entropy,* cuya formula es:

$$CE = -\frac{1}{M}\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ji} * log(\hat{y}_{ji}))$$

Comparará la distribución de las predicciones $\hat{y}$ (las activaciones en la capa de salida, una para cada clase) con la distribución real y, donde la probabilidad de la clase verdadera se establece en 1 y 0 para las otras clases

4.3.1.4. Optimizador

El optimizador elegido fue el *stochastic gradient descent* con un learning rate adaptativo personalizado. El learning rate es un parámetro muy importante en las redes neuronales ya que determina la velocidad en la que descendemos por la pendiente de la función de error. Si el learning rate es muy bajo nos estaremos acercando a un mínimo de la función de error lentamente y es probable que nos quedemos en un mínimo local alto. Por el otro lado si tomamos un learning rate muy alto es posible que nos saltemos mínimos locales buenos e incluso puede suceder que no lleguemos a un mínimo absoluto, en la **Figura 4.9** podemos ver una ilustración.

Imagen extraída de: `https://www.jeremyjordan.me/nn-learning-rate/`

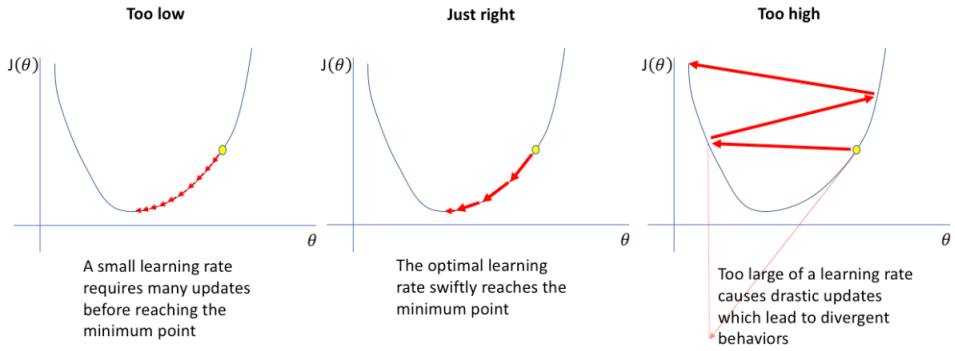


Figura 4.9: Efectos del learning rate.

Por eso, se decidió aplicar un learning rate adaptativo, de tal forma que al principio del entrenamiento el learning rate sea alto, para que salte los mínimos locales altos, y a medida que vaya avanzando el entrenamiento ir reduciendo el learning rate para encontrar un mínimo. Para aplicar esto, Keras permite personalizar el learning rate y aplicarlo al entrenamiento. El learning rate adaptativo que se utilizó es la que se muestra en la **Figura 4.10**, donde aplica un learning rate de 0.1 en las primeras 20 épocas, entre las 20 y 30 épocas aplica un learning rate de 0.01 y en las últimas épocas un learning rate de 0.001.

```
def scheduler(epoch):
    if epoch < 20:
        return .1
    elif epoch < 30:
        return 0.01
    else:
        return 0.001
```

Figura 4.10: Learning rate personalizado.

4.3.2. Implementación de modelos de regresión

Para el modelo de regresión se decidió utilizar la VGG16 ya que dio buenos resultados en el problema de clasificación, Sección **5.2**, pero haciendo uso de una capa completamente conectada diferente. La nueva capa completamente conectada es la que se muestra en la **Figura 4.12**. Además, se implementaron varios modelos con diferentes funciones de activación de la última capa del modelo para ver cual es mejor para este problema. Las funciones de activación que se probó fueron la relu, sigmoide, leaky relu, tangente hiperbólica y elu. También, se utilizó una función de error diferente, en esta ocación se utilizo la función del error cuadrático medio.

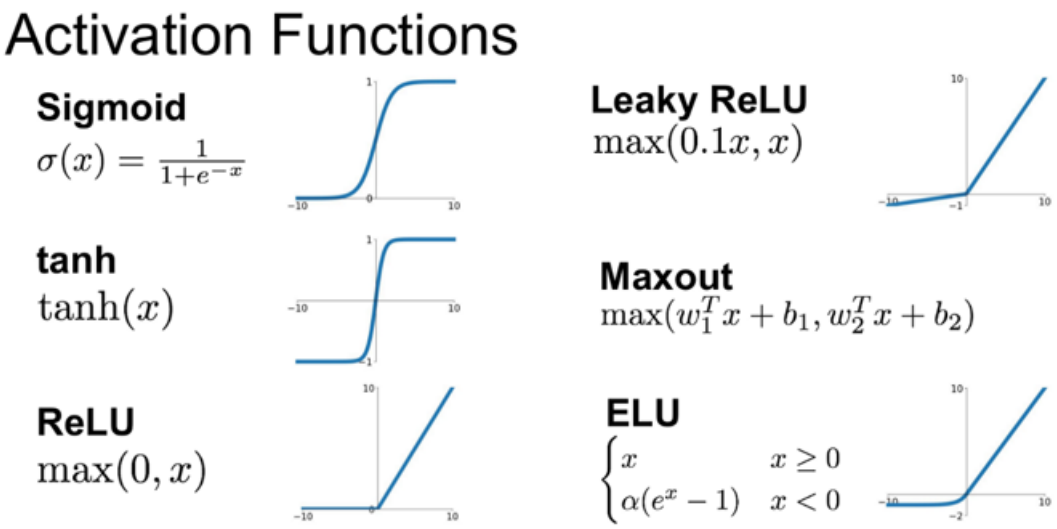


Figura 4.11: Funciones de activación.

```
dense_layer = Dense(2048, name='dense_laayer')(reshape_layer)
dense_layer = BN()(dense_layer)
dense_layer = Activation('relu')(dense_layer)
dense_layer = Dense(2048, name='dense_layerd')(dense_layer)
dense_layer = BN()(dense_layer)
dense_layer = Activation('relu')(dense_layer)
dense_layer = Dense(1, name='dense_layer')(dense_layer)
dense_layer = BN()(dense_layer)
dense_layer = Activation('relu')(dense_layer)
```

Figura 4.12: Capa completamente conectada del modelo de regresión.

Capítulo 5

Experimento y resultados

En esta sección se mostrará como fueron entrenados y evaluados los modelos. También, se comprobará el correcto funcionamiento de estos modelos usando la webcam de un ordenado. Y finalmente, se comprobará la eficiencia de algunos métodos de obtención de características de biometría blanda según la pose de la cara determinada por los estimadores/clasicadores anteriores.

5.1. Comprobación de la arquitectura con la VGG16

Después de haber elegido la arquitectura para el problema de clasificación, se tuvo que comprobar que dicha arquitectura era adecuada para el problema de clasificación. Para dicha comprobación, se entrenó el modelo VGG16 con su capa completamente conectada personalizada para dos clases.

Para mejorar la variedad de imágenes, Keras proporciona un método que durante el entrenamiento hace aumentación de datos según cómo se haya configurado. En este caso se ha configurado para que nos generara imágenes rotadas entre ±2 grados y que haga translaciones en el ancho y alto de la imagen no más de un 5% de la imagen. En la **Figura 5.1.** podemos ver la configuración.

```
datagen = ImageDataGenerator(
    rotation_range=2,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=False)
```

Figura 5.1: Datagenerator de keras.

Debido al tamaño del conjunto de datos las imágenes no podían tenerse todas en memoria ya que producía un error. Para solucionar esto se tuvo que crear un generador de imágenes que cargara imágenes del disco cada vez que Keras solicitara imágenes.

Antes de entrenar, es adecuado normalizar las imágenes ya que reduce el tiempo de entrenamiento. En la **Figura 5.2** podemos ver que la función de error cuyas entradas no están normalizadas y podemos observar que para llegar al mínimo del error el trayecto es más largo respecto a la función de error que vemos en la **Figura 5.3** cuyas entradas están normalizadas.

Imagen extraída de:
`https://medium.com/@pnkshir/concept-of-normalization-427108d1ccfa`

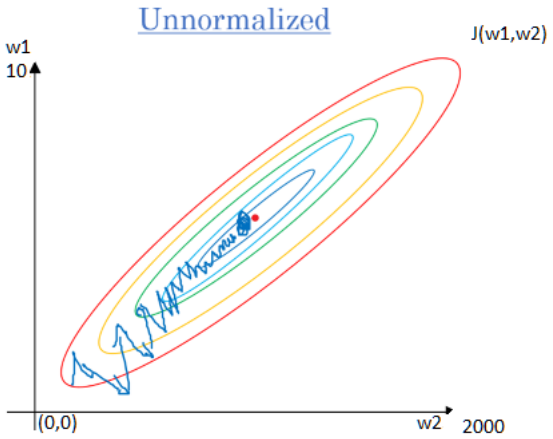


Figura 5.2: Función de error con datos no normalizados.

Imagen extraída de:
`https://medium.com/@pnkshir/concept-of-normalization-427108d1ccfa`

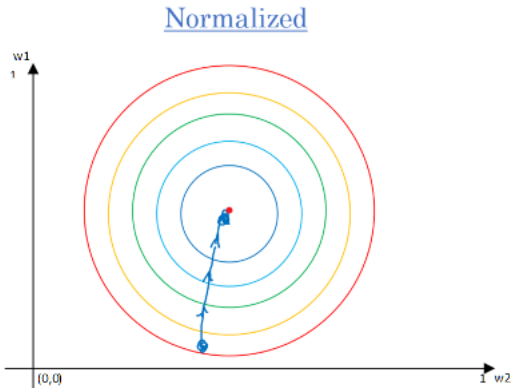


Figura 5.3: Función de error con datos normalizados.

En ese momento el modelo ya estuvo preparado para ser entrenado. Para el entrenamiento se aplicó el método llamada hold-out, explicado en la Sección **2.3**, de 10 repeticiones y el número de épocas fueron 40 usando un tamaño de lote de 16 imágenes.

El resultado obtenido obtenido en el entrenamiento fue un 90.7% de tasa de acierto en el conjunto de test. Y finalmente, podemos decir que la arquitectura es buena.

5.2. Resultados de entrenamiento de los modelos de clasificación

Tras la comprobación anterior, hay 5 modelos a probar para tres problemas de clasificación, por tanto, se prepararon 15 programas de entrenamientos como el de la sección anterior. La duración del entrenamiento tuvo una duración de una semana. Obteniendo los resultados que se muestra en la **Tabla 5.1**. El modelo VGG16 fue la que mejor resultado dió para la clasificación de caras frontales y no frontales y también para la clasificación de caras frontales, derecha e izquierda. Y para la clasificación de cinco clases la Xception.

Resultados			
Modelos	Dos clases (%)	Tres clases (%)	Cinco clases (%)
MobileNet	88.5	88.2	76.7
NASNetLarge	64.4	68.1	62.5
NASNetMobile	71.4	69.3	62.1
XCeption	90.4	90.6	79.9
VGG16	90.7	91.3	79.6

Cuadro 5.1: Tasa de acierto de los modelos de clasificación.

Para comprobar que no es un error los resultados de los mejores modelos, se sacó la matriz de confusión.

Matriz de confusión Xception					
X	P.izquierdo	Izquierda	Frontal	Derecha	P. derecho
P.izquierdo	453	193	2	1	1
Izquierda	78	2853	138	6	0
Frontal	0	115	4277	108	0
Derecha	1	8	128	2840	98
P. derecho	1	0	2	163	484

Cuadro 5.2: Matriz de confusión para el modelo Xception con 5 clases.

Matriz de confusión VGG16			
X	Izquierda	Frontal	Derecha
Izquierda	3589	131	5
Frontal	132	4225	143
Derecha	8	102	3615

Cuadro 5.3: Matriz de confusión para el modelo VGG16 con 3 clases.

Matriz de confusión VGG16		
X	No frontal	Frontal
No frontal	7216	234
Frontal	254	4246

Cuadro 5.4: Matriz de confusión para el modelo VGG16 con 2 clases.

Se observa en las **Tablas 5.2, 5.3, 5.4**, que la mayoría de las predicciones son acertadas, también que se clasifica de forma erronea las clases vecinas, pero no clasifica mal las clases de los extremos.

5.3. Resultados de entrenamiento de los modelos de regresión

Los resultados obtenidos tras el entrenamiento, también aplicado el hold-out con 10 repeticiones y 40 épocas, para los modelos de regresión con diferentes funciones de activación, se ven en la **Tabla 5.5**. Como se observa, el modelo VGG16 con la función de activación leaky relu fue el que menos error produjo en el conjunto de test.

Resultados				
Elu	Leaky Relu	Relu	Sigmoide	Tanh
2.11	0.117	0.124	0.31	0.2

Cuadro 5.5: Error de los modelos de regresión.

5.4. Realización de pruebas de los estimadores

A continuación, se probará la eficiencia de los estimadores con un programa Python haciendo uso de la librería OpenCV para usar las imágenes captadas por una webcam de un portátil y también haciendo uso del detector de caras MTCNN para recortar la imagen obteniendo la cara detectada y luego será enviada al modelo para que nos de un resultado.

El primer modelo que se probó fue el modelo de clasificación de dos clase, Frontal o No frontal. En la **Figura 5.4** podemos observar que cuando la cara esta en posición frontal el bounding box de la cara es coloreado en azul y cuando no está en posición no frontal en el bounding box es pintado en verde.

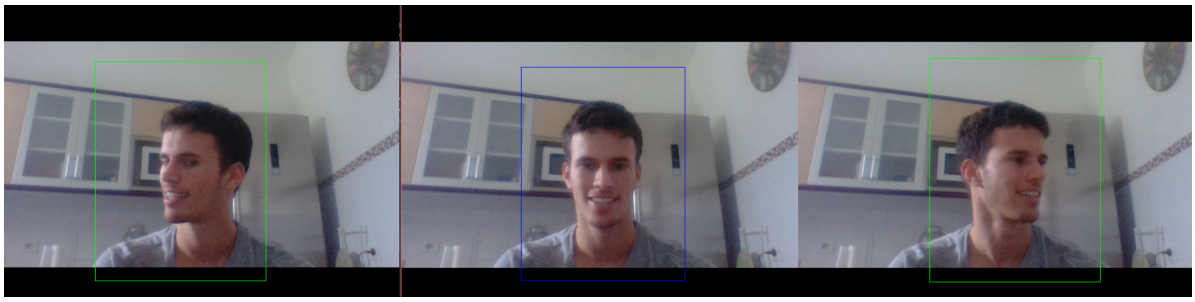


Figura 5.4: Detector de poses Frontal y no frontal

Por consiguiente, se probó el modelo con tres clases, pose girada a la derecha, pose girada a la izquierda y posición frontal. En la **Figura 5.5** podemos observar que cada tipo de pose está identificado con un color distinto.

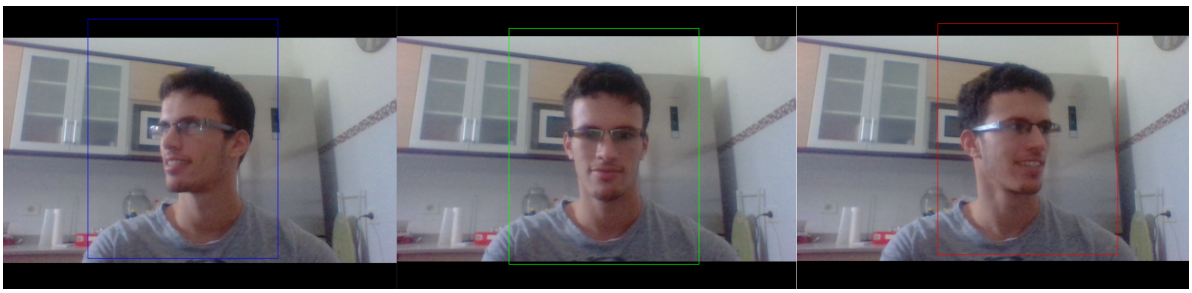


Figura 5.5: Detector de poses de tres clases.

De los modelos de clasificación, quedaba por probar el modelo con cinco clases, perfil izquierdo, izquierda, frontal, derecha y perfil derecho. En la **Figura 5.6** podemos observar que cada tipo de pose está identificado con un color distinto.

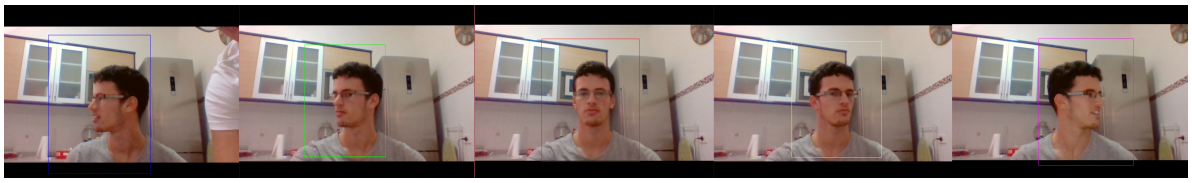


Figura 5.6: Detector de poses de cinco clases.

Y para finalizar, quedaba el modelo de regresión que a diferencia de los anteriores devuelve el grado de rotación de la cabeza. En la **Figura 5.7** se muestran varias capturas de la aplicación en funcionamiento.

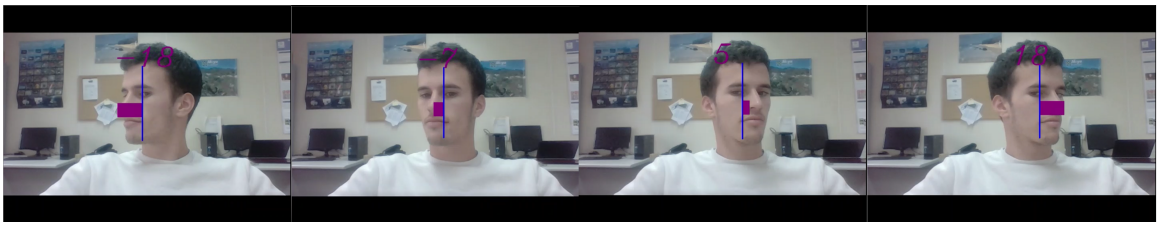


Figura 5.7: Modelo de regresión en funcionamiento.

5.5. Comprobación de la eficiencia de algunos métodos de obtención de características de biometría blanda según la pose

En esta sección se va a utilizar dos métodos de obtención de características de biometría blanda para comprobar su eficiencia para diferentes poses de la cara. Estos son los siguientes:

- Clasificador de sexo.
- Detector de sonrisa.

Estos modelos han sido elegidos porque el conjunto de datos MTFL tienen las imágenes etiquetadas con su sexo y también si en la imagen está sonriendo la persona o no.

El clasificador de sexo fue tomado de [gen] que consiste en un modelo Keras y el detector de sonrisa lo obtuve de la librería OpenCV que consiste en modelo haar cascade desarrollado en [Déniz et al.2008].

En primer lugar, se etiquetó las imágenes según la salida de los modelos mencionado anteriormente en un archivo .csv donde almacenaba la etiqueta real y la predicha por el modelo, tanto para el sexo como para la sonrisa, y también la ruta de la imagen.

Por consiguiente, haciendo uso de los estimadores/clasificadores de poses se clasificó las imágenes según cada modelo y se obtuvo la precisión de los modelos de obtención de características biometría blanda, para cada una de las poses clasificadas por cada modelo, haciendo uso de la función que nos proporciona Sklearn llamado *accuracy_score* en el cual se le pasa como primer parámetro un vector de etiquetas verdaderas y como segundo un vector de etiquetas predichas.

Los resultados obtenidos son los que se muestran en las **Tablas 5.6 5.7, 5.8 5.9**. Como se observa, los métodos de obtención de características de biometría blanda dan mejor rendimiento cuando la pose de la cara es frontal, a medida que se va alejando de dicha pose la eficiencia se va reduciendo. Con estos resultados queda claro que la pose de la cara afecta a la eficiencia de los métodos de obtención de características de biometría blanda.

Resultados		
X	Estimador de sexo (%)	Detector de sonrisa (%)
No frontal	59.0	40.7
Frontal	91.6	59.7

Cuadro 5.6: Tasa de acierto de los modelos de obtención de características de biometría blanda, dos clases.

Resultados		
X	Estimador de sexo (%)	Detector de sonrisa (%)
Izquierda	58.8	40.2
Frontal	91.4	59.6
Derecha	57.0	39.8

Cuadro 5.7: Tasa de acierto de los modelos de obtención de características de biometría blanda, tres clases.

Resultados		
X	Estimador de sexo (%)	Detector de sonrisa (%)
P.izquierdo	47.6	30.3
Izquierda	60.2	41.1
Frontal	91.2	60.1
Derecha	58.3	40.4
P. derecho	45.5	28.9

Cuadro 5.8: Tasa de acierto de los modelos de obtención de características de biometría blanda, cinco clases.

Resultados		
Intervalo de grados	Estimador de sexo (%)	Detector de sonrisa (%)
$[-\infty:-30]$	52.1	30.5
$(-30:-20]$	57.1	33.2
$(-20:-10]$	71.0	47.0
$(-10:0]$	87.3	61.8
$(0:10]$	86.1	59.7
$(10:20]$	74.5	50.2
$(20:30]$	53.7	35.3
$(30:\infty]$	46.6	29.2

Cuadro 5.9: Tasa de acierto de los modelos de obtención de características de biometría blanda, 8 intervalos de grados.

Capítulo 6

Conclusion

El rendimiento de la mayoría de los métodos que basan su funcionamiento en la imagen de la cara para obtener características de biometría blanda como puede ser el sexo o la edad, o para la identificación/re-identificación/verificación de personas, se ven bastante influenciados por la pose de la cara. En este proyecto he recopilado y estudiado el estado del arte. También, he seleccionado la base de datos MTFL para el entrenamiento de los modelos de clasificación de poses y el modelo de estimación del grado de la pose probando varios modelos que nos proporciona keras, como la VGG16, Xception, MobileNet, NASNetLarge y NASNetMobile, y luego evaluando los resultados sacando la matriz de confusión y creando un programa para obtener, a tiempo real, los resultados devuelto por los modelos según las imágenes captadas por una webcam y corroborar su correcto funcionamiento. Por otro lado, se ha hecho un estudio de varias herramientas como Keras, Numpy, Pandas, Seaboen, Overleaf, etc. Y finalmente, he evaluado el rendimiento de dos estimadores de sonrisa y sexo según la pose de la cara y he obtenido como resultado que las poses frontales favorecen al rendimiento de estos estimadores y que a medida que el ángulo respecto a la posición frontal va siendo mayor el rendimiento de estos estimadores se van reduciendo.

6.1. Valoración personal.

En mi opinión, este proyecto me ha aportado más conocimiento sobre el mundo de la inteligencia artificial, he conocido varios tipos de redes neuronales convolutivas y además las he puesto a prueba. También, he leído bastantes artículos relacionado con el entorno en el que iba a trabajar y obtenido conocimiento que me han ayudado a afrontar este proyecto.

Además, quiero decir que estoy satisfecho con el trabajo que he realizado. Nunca había experimentado la sensación de trabajar en un proyecto de investigación y de aprovechar todo mi conocimiento tanto académico y personal, para decidir cada uno de los pasos que he realizado.

Y para concluir, esta investigación puede servir para futuras investigaciones que vaya a trabajar en el mismo entorno. Utilizar la información de la poses para mejorar o desarrollar

nuevos modelos.

Bibliografía

[Cal] Caltech 10000 web faces.

[gen] Gender estimation.

[MTF] Mtfl: Multi-task facial landmark.

[BioID] BioID. The bioid face database.

[Chollet2017] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258.

[Déniz et al.2008] Déniz, O., Castrillon, M., Lorenzo, J., Anton, L., and Bueno, G. (2008). Smile detection for user interfaces. In *International Symposium on Visual Computing*, pages 602–611. Springer.

[Ding and Tao2016] Ding, C. and Tao, D. (2016). A comprehensive survey on pose-invariant face recognition. *ACM Transactions on intelligent systems and technology (TIST)*, 7(3):37.

[Gualbertoa] Gualberto, A. Linkedin.

[Gualbertob] Gualberto, A. Real-time face pose estimation with deep learning.

[Hecht-Nielsen1992] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

[Howard et al.2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[LeCun et al.1998] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Marín-Reyes et al.2016] Marín-Reyes, P. A., Lorenzo-Navarro, J., Castrillón-Santana, M., and Sánchez-Nielsen, E. (2016). Shot classification and keyframe detection for vision based speakers diarization in parliamentary debates. In Luaces, O., Gámez, J. A., Barrenechea, E.,

Troncoso, A., Galar, M., Quintián, H., and Corchado, E., editors, *Advances in Artificial Intelligence*, pages 48–57, Cham. Springer International Publishing.

[Rosenblatt1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Russakovsky et al.2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Russell and Norvig2016] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[Simonyan and Zisserman2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Sutskever et al.2012] Sutskever, I., Hinton, G. E., and Krizhevsky, A. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105.

[Szegedy et al.2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Tapu et al.2018] Tapu, R., Mocanu, B., and Zaharia, T. (2018). Face recognition in video streams for mobile assistive devices dedicated to visually impaired. In *2018 14th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 137–142. IEEE.

[Viola et al.2001] Viola, P., Jones, M., et al. (2001). Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1:511–518.

[Zhang et al.2016] Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503.

[Zoph et al.2018] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.