escolauniversitaria
POLITÉCNICA

Universidade da Coruña

TÍTULO	IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO DE GESTOS MEDIANTE TÉCNICAS INTELIGENTES
MÁSTER	INFORMÁTICA INDUSTRIAL Y ROBÓTICA
ASIGNATURA	TRABAJO DE FIN DE MÁSTER
Nº TFM	4538M01A002
ALUMNO	MANUEL PALACIOS FRAGOSO
TUTOR	ESTEBAN JOVE PÉREZ
COTUTOR	FRANCISCO ZAYAS GATO
FECHA	JUNIO 2021

Índice

Índice de figuras

Índice de tablas

1. Objeto

En este Trabajo Final de Máster se realizará el estudio y la implementación de un sistema de reconocimiento de gestos mediante técnicas de machine learning y de visión artificial. Los gestos identificados serán realizados con una mano.

Inicialmente, se hará un estudio del tipo de software apropiado para aplicar las librerías de tratamiento de imágenes y las técnicas de machine learning. Posteriormente, tratará de implementarse el reconocimiento de los diferentes gestos realizados.

Se estudiará la posibilidad de realizar una aplicación capaz de analizar los gestos detectados.

2. Alcance

En este apartado se presentan los objetivos del trabajo.

- Análisis del software adecuado para realizar el reconocimiento de los diferentes gestos realizados con la mano.
- Aplicación de librerías de tratamiento de imágenes para facilitar la detección de los gestos.
- Aplicación de las técnicas apropiadas de machine learning para reconocer los gestos.
- Estudio de la posibilidad de realización de una aplicación con el sistema de reconocimiento de gestos.

3. Antecedentes

En esta sección se expone el punto de partida y los condicionantes de este trabajo.

Al tratarse de un problema de clasificación de imágenes en una aplicación de tiempo real, se introduce en este apartado la visión artificial y el aprendizaje automático.

3.1. Inteligencia artificial

La inteligencia artificial (IA) trata de dotar de inteligencia a las máquinas. Esta, puede ser dividida en dos tipos:

- General: Busca desarrollar sistemas flexibles y versátiles como la inteligencia humana para solucionar un amplio abanico de problemas complejos. Como ejemplo más claro de esto, están las arquitecturas cognitivas como puede ser Icarus, que trata de imitar el comportamiento humano.
- Especializada: Busca desarrollar sistemas que solo se puedan aplicar a las tareas para las que fueron diseñados. Un ejemplo de este tipo de IA podría ser una algoritmo de traducción de texto o uno de reconocimiento de imágenes.

Este problema se busca resolverlo de la forma más eficiente posible empleando una IA especializada, pues la realización de un programa general conlleva a priori muchas más dificultades y unos resultados peores.

El campo de la inteligencia artificial se divide en muchas ramas, como pueden ser:

- Aprendizaje automático.
- Procesamiento del lenguaje natural.
- Procesamiento del habla.
- Sistemas expertos.
- Planificación y optimización.
- Robótica.
- Visión artificial.

De entre estos campos, se combinarán aprendizaje automático y visión para buscar encontrar la mejor solución posible.

3.2. Visión artificial

La visión artificial toma, procesa y analiza imágenes para lograr que una máquina disponga de información del mundo real. Intenta lograr que el sistema sea capaz de analizar los datos del mismo modo que lo haría la vista humana.

Las imágenes se descomponen en pequeñas unidades llamadas píxeles, que son la unidad básica de una imagen digitalizada en pantalla, ya sea a color o en blanco y negro.

Trabajando en escala de grises solo hay un único valor por cada píxel, mientras que, trabajando a color, se tienen tres valores por cada píxel. En el caso de que la escala de color sea RGB, los valores representan cada uno un color: el primero sería rojo, el segundo verde y el tercero azul.

3.2.1. Ruido en imágenes

Trabajar con imágenes hace inevitable tener que lidiar con interferencias, las cuáles aparecen en la toma, en la transferencia y en el procesado de la imagen.

Cuando se ejecuta un programa, la imagen ha sido tomada por una cámara y transmitida y procesada por un ordenador. Por ello, las imágenes tendrán diferentes interferencias afectándoles en mayor o menor medida. Para las interferencias que parezca que pueden causar problemas se aplican soluciones.

Existen muchos tipos de ruido pero los principales son los tres siguientes:

- **Sal y pimienta:** Esta interferencia se caracteriza por la presencia de píxeles con valor máximo o mínimo (blancos o negros). Es una interferencia de tipo impulso y, para eliminarla, se emplean filtros que tienen en cuenta el valor de los píxeles cercanos

Figura 3.1: Ruido sal y pimienta

- **Gaussiano:** Este tipo de ruido se debe a que los píxeles de una imagen varían siguiendo una distribución normal, tal y como dice el teorema central del límite. Según este teorema, siempre y cuando el número de variables sea muy elevado, la presencia de los distintos tipos de ruido hará que este se reparta siguiendo una distribución Gaussiana.

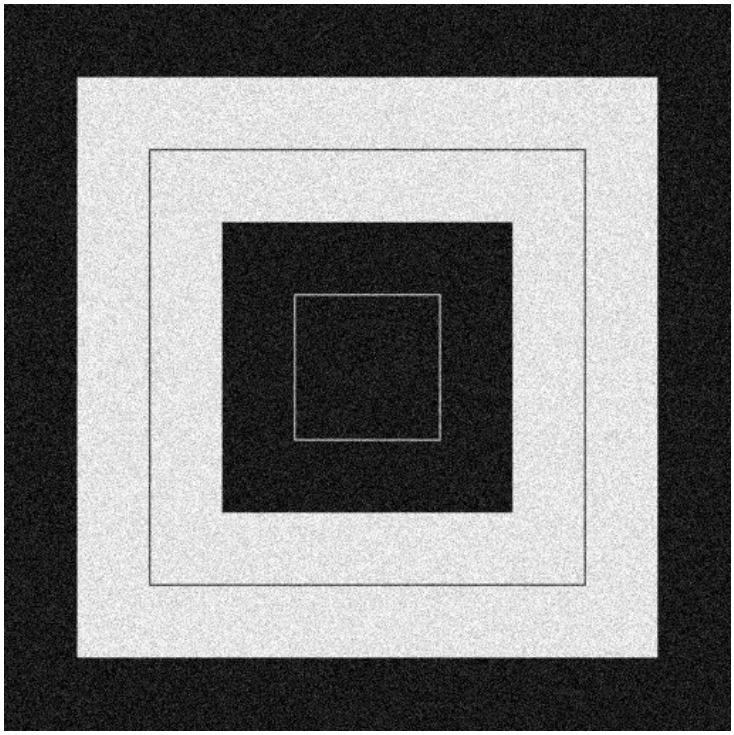


Figura 3.2: Ruido gaussiano

- **Uniforme:** Con este tipo de ruido,los píxeles cambian su valor siguiendo una distribución uniforme, lo que provoca que la imagen se vea como si estuviese codificada. Para arreglarlo, una de las soluciones posibles sería aplicar un filtro de promedio espacial pero, esto conlleva una disminución de la calidad de la imagen. En la Figura 3.3, se muestra este tipo de ruido en gran cantidad para poder observar claramente la diferencia con los dos filtros anteriores. Se puede ver como en el caso de este ruido, ni sigue una distribución concreta ni los píxeles son en blanco y negro.

Figura 3.3: Ruido uniforme

3.3. Aprendizaje automático

El aprendizaje automático es una rama de la inteligencia artificial basado en desarrollar algoritmos que permitan a las máquinas aprender. Los modelos de aprendizaje automático se basan en técnicas matemáticas como son la estadística, la optimización o la probabilidad. A partir de estas técnicas y de unos datos disponibles, realizan inferencia.

La mayor desventaja que tiene el aprendizaje automático es la dificultad de interpretas algunos de los modelos, pues trabajan como cajas negras. Por lo demás, mejoran a los sistemas de reglas tanto en velocidad, como la tasa de aciertos globales y en su mantenimiento.

Los sistemas de reglas pueden funcionar muy bien para determinados problemas pero, cuando no se tienen suficientes conocimientos explícitos para resolver la tarea pero sí muchos ejemplos, es una de las situaciones en las que se elige el aprendizaje automático.

El aprendizaje automático puede dividirse en aprendizaje supervisado, no supervisado y reforzado.

3.3.1. Aprendizaje supervisado

En el aprendizaje supervisado se disponen de datos etiquetas con el resultado correcto de su clasificación. A partir de estos datos se realiza un entrenamiento buscando que el modelo se ajuste lo más posible a la clasificación real o a un valor numérico

(problemas de regresión).

3.3.2. Aprendizaje no supervisado

Es este tipo de aprendizaje, los datos de entrenamientos no están etiquetados, por lo que hay que buscar otra manera para clasificarlos. Uno de los métodos más comunes de aprendizaje no supervisado es el *clustering*, pues clasifica el conjunto de datos en función de características similares.

3.3.3. Aprendizaje reforzado

Aunque estos algoritmos no parten de una serie de datos etiquetados, no se pueden considerar aprendizaje no supervisado, pues no trata de realizar una clasificación en base a una medida de distancia. Mientras que en el aprendizaje supervisado y no supervisado se intenta minimizar una función de error, en el aprendizaje por refuerzo se trata de maximizar la recompensa.

En el aprendizaje por refuerzo, un agente aprenderá a elegir las acciones que le devuelvan una mayor recompensa al actuar sobre el entorno.

4. Análisis de soluciones

En esta sección se plantean diferentes alternativas estudiadas para la realización del trabajo, desde el lenguaje de programación, pasando por el preprocesado de las imágenes hasta el tipo de aprendizaje (supervisado o no supervisado) que se emplea.

4.1. Lenguaje de programación

Entre las principales alternativas para la realización de este trabajo, hay varios lenguajes de programación que resultan apropiados para realizarlo. Los principales son:

- **Python:** Sencillo de usar y cuenta con una gran variedad de librerías tanto para visión artificial como para *machine learning*. Es el más usado en este tipo de aplicaciones y el lenguaje de programación favorito según la encuesta de PYPL en 2020.
- **C/C++:** Más complejo que el anterior pero mucho más rápido. Su uso suele estar más centrado en el desarrollo de software de sistemas y la creación de aplicaciones, aún así, eso no implica que no pueda ser empleado en aplicaciones de *machine learning*.
- **Matlab:** Muy empleado tanto en aplicaciones de visión artificial como en aplicaciones de *machine learning*. El mayor inconveniente de este lenguaje de programación es que las licencias de Matlab son de pago y el propio programa ocupa mucho espacio en el disco duro. Este lenguaje tiene mucha menos documentación ya que no es de código abierto y libre.
- **Julia:** Este lenguaje, aunque mucho más desconocido que los anteriores, también resultaría una gran opción para este trabajo. Se trata de un lenguaje de programación de alto nivel muy sencillo y rápido. Además, puede emplear tanto paquetes de C como de Python.

El lenguaje de programación empleado en este trabajo es finalmente Python, ya que es el más empleado en problemas de clasificación y de visión artificial (siempre que no se requiera mucha más velocidad, si no se emplearía C#). Julia es una gran alternativa pero a día de hoy, la comunidad científica está más centrada en python.

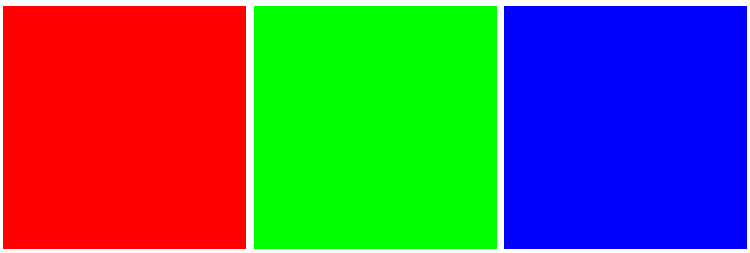
4.2. Preprocesado

Cuando se busca realizar el preprocesado de los datos se intentan cumplir dos objetivos:

- Eliminar toda la información de escasa relevancia para disminuir el peso de los datos. Una mayor cantidad de información conlleva un mayor coste computacional y, al trabajar con imágenes (matrices), cualquier pequeña disminución de la información empleada en el entrenamiento puede suponer una gran diferencia en el tiempo de entrenamiento.
- Resaltar la información relevante. Al igual que se busca eliminar toda la información que sea innecesaria por el coste computacional, resaltar la información más importante mejorará los resultados del entrenamiento.

Para realizar un buen preprocesado del conjunto de datos será importante tener en cuenta ambos puntos.

El primer problema aparece al estar trabajando con imágenes a color. Estas imágenes están en formato RGB, por lo que sus datos se encuentran en una matriz con tres valores por cada píxel (rojo, verde y azul). Es importante tener cuidado con el formato de color al trabajar también con la librería de opencv, pues trabaja en BGR. En la siguiente imagen se muestra como vendrían representados los datos para un pixel de cada uno de los tres colores primarios en RGB:



$$(255 \quad 0 \quad 0)\,(0 \quad 255 \quad 0)\,(0 \quad 0 \quad 255)$$

Figura 4.1: Colores primarios en RGB.

Si se intenta entrenar los clasificadores con las imágenes a color y sin redimensionarlas, el entrenamiento tardaría mucho más tiempo en llevarse a cabo. Al aplicar filtros que resalten las características relevantes, se eliminan muchos datos que no son necesarios para el entrenamiento. Para empezar, si los filtros transforman la ima-

gen a escala de grises, ya se estaría trabajando con un volumen de datos tres veces menor.

Se plantea la opción de pasar a blanco y negro las imágenes a partir del color rojo. El motivo por el que se escoge el color rojo es porque, de los tres colores primarios, es el más presente en una persona del color de piel del autor. Tras realizar la conversión de la imagen a blanco y negro, se plantean de nuevo varias alternativas:

- Aplicar un filtro de ruido y realizar una ecualización del histograma de la imagen.
- Aplicar un filtro de ruido y realizar una detección de bordes.
- Aplicar el descriptor HOG.

En una primera prueba inicial, se aplican las diferentes opciones de preprocesado a una imagen de cada uno de los diferentes tipos de iluminación. El resultado de la ecualización del histograma no es el deseado pues, aunque resalta bien la mano en las imágenes con luz intermedia y elevada, en las que tienen poca luz se vuelve muy difícil determinar el gesto realizado.

Figura 4.2: Gesto 5 tras realizar una ecualización adaptativa del histograma.

Como se puede ver en la Figura 4.2, en la primera imagen (luz intermedia) la mano se ve ligeramente resaltada pero, a pesar de haber realizado un filtro de ruido gaussiano previo a la ecualización, la imagen sigue teniendo mucho ruido y además no cumple bien el objetivo de resaltar la información importante y eliminar la poco importante.

La segunda imagen es la única para la cual la técnica de ecualización del histograma resultaría útil pues, resalta la mano y elimina información poco importante debido a la luz. En la tercera imagen se ve como empeora mucho la calidad de la imagen y resulta mucho más complicado distinguir el gesto que está siendo realizado.

Se podría plantear realizar una detección de bordes para las imágenes con el histograma modificado, pero solo sería útil en los casos en los que hay mucha luz, puesto que en los otros dos (sobre todo cuando hay poca luz) sería prácticamente imposible detectar los bordes de la mano.

En la Figura 4.3 se muestra el detector de bordes aplicado a las imágenes de partida. Se puede ver como este filtro funciona mucho mejor para esta aplicación y, por lo tanto, se emplea para el trabajo. Este filtro es explicado más en detalle en el apartado 7.2.1.

Figura 4.3: Gesto 5 tras la detección de bordes *canny*

En la siguiente imagen, Figura 4.4, se muestra el resultado de aplicar el descriptor HOG a tres imágenes del gesto 5. Al igual que el detector de bordes, parece funcionar bien para los tres tipos de iluminación del dataset, por lo que se prueba también en este trabajo. En el apartado 7.2.2 es explicado en detalle.

Figura 4.4: Gesto 5 tras aplicar el descriptor HOG.

Se entrenan todos los modelos, exceptuando las redes CNN (*Convolutional Neural Networks*), tanto con un preprocesado con el descriptor HOG como con un preprocesado con detección de bordes, para así disponer de más resultados entre los que comparar.

Para las redes CNN no se realiza preprocesado ya que las primeras capas de la red se encargan de realizar los filtros necesarios para el análisis de las imágenes. Además, al cargarlas desde el repositorio de Tensorflow, indican unos tamaños de imagen determinados y que las imagen sean a color.

4.3. Tipo de aprendizaje

Al iniciar este trabajo, se plantea si emplear aprendizaje supervisado, no supervisado o ambos. Disponer de etiquetas en el dataset permite escoger el tipo de aprendizaje.

El aprendizaje de tipo supervisado es muy empleado para este tipo de problemas, ya que al disponer de las etiquetas del conjunto de entrenamiento, se garantiza que la clasificación se haga conforme a los grupos que se desean.

Tal como se explica en el apartado 3.3.2, para este tipo de aprendizaje se emplean los datos sin etiquetar. Para comprobar si este tipo de técnicas serían útiles para este trabajo, se realizan varias pruebas.

4.3.1. Pruebas de aprendizaje no supervisado

Para la realización de estas pruebas, se emplea el conjunto de 1154 imágenes en el que está aplicado el filtro detector de bordes *Canny* (el dataset inicial se detalla en el apartado 7.1 y, el preprocesado aplicado, en el apartado 7.2). Estas imágenes se introducen aplanadas como vectores en cada modelo. Para estas pruebas, se realiza KMeans y fcluster.

Para poder comprobar cómo se realizan los agrupamientos, se guardan en figuras todas las imágenes de cada clúster para poder observar cuál es la relación existente entre las imágenes de un mismo grupo.

En Kmeans se prueba a introducir como número de clústers en los que clasificar las imágenes el número de gestos de los que se dispone. El resultado de realizar KMeans para 7 clústers es que los grupos se realizan en función de la iluminación de las imágenes y de la camiseta que se lleva puesta.

Se realiza la lista de inercia para comprobar en cuántos clústers sería necesario dividir el conjunto de datos según este método. El codo de la lista de inercia se encuentra aproximadamente en un valor de K=12, pero la clasificación que realiza sigue sin estar relacionada con el gesto realizado en la imagen.

5. Software y librerías empleadas

En esta sección se especifica el software y las librerías que se han empleado para llevar a cabo este proyecto.

5.1. Software

El trabajo es realizado en el sistema operativo Ubuntu 20.04. Resulta importante mencionar el sistema operativo en el que se realiza pues al trabajar con las rutas de las imágenes, en Linux las barras diagonales son las normales, mientras que en Windows se entregan barras diagonales invertidas. Al estar trabajando con ese caracter, para poder leer la carpeta en donde se encuentra la imagen, habría que modificar el código para su ejecución en otros sistemas operativos.

El entorno de desarrollo seleccionado para el trabajo es Visual Studio Code.

El diseño de la aplicación final para la prueba del código se lleva a cabo empleando Tkinter, ya que resulta sencillo y práctico para la creación de una aplicación de este estilo.

5.2. Librerías empleadas

A continuación se nombran todas las librerías empleadas en el trabajo:

- **OpenCV:** Es la librería más popular de visión artificial. En este trabajo se emplea para el filtro detector de bordes *Canny* en la etapa de preprocesado.
- **Skimage:** Librería de visión artificial que se emplea en este proyecto para aplicar el preprocesado HOG en las imágenes.
- **Pandas:** Librería empleada para el manejo y análisis de datos. Se emplea para guardar los datos relevantes de cada modelo de aprendizaje en una hoja de cálculo.
- **Matplotlib:** Librería empleada para realizar gráficos.
- **Numpy:** Librería de cálculo numérico que permite trabajar con *arrays*.
- **Tensorflow:** Librería necesaria para el entrenamiento de las redes. En este trabajo se emplea para poder emplear redes convolucionales.

- **Tensorflow_hub:** Librería que permite emplear modelos preentrenados disponibles en el repositorio de Tensorflow.
- **Sklearn:** Librería más importante de *machine learning* en python. Proporciona tanto los modelos de clasificación supervisada y no supervisada, como las métricas de evaluación, las funciones de validación cruzada y la función de búsqueda de hiperpaáremtros.
- **Scipy:** Esta librería se emplea en este trabajo para algunos métodos de clasificación no lineales.
- **Random:** Librería empleada para poder generar números “aleatorios”.

6. Técnicas de aprendizaje automático empleadas

Para este proyecto se emplean un total de siete técnicas de aprendizaje supervisado, dos de tipo lineal y cinco no lineales. Las técnicas de aprendizaje se consideran lineales o no lineales en función de la región de decisión que generen.

Para poder explicar los diferentes modelos, se explica a continuación el significado de exactitud como métrica de evaluación.

- Exactitud (Accuracy): Tasa de acierto global del sistema.

$$Exactitud = \frac{Aciertos\,totales}{Casos\,totales} \tag{6.1}$$

El resto de métricas se encuentran explicadas en el apartado 9.

Las técnicas lineales de aprendizaje supervisado empleadas son:

- **Regresión logística.**
- **Discriminante lineal.**

Las técnicas no lineales de aprendizaje supervisado utilizadas son:

- **K vecinos más cercanos (KNN)** (En este caso es supervisado porque se dispone de los datos etiquetados)
- **Árboles de decisión.**
- **Bosque aleatorio.**
- **Máquina de vectores de soporte.**
- **Red convolucional (CNN).**

6.1. Modelos lineales

6.1.1. Regresión Logística

La regresión logística emplea una función sigmoide que fuerza a que la salida se encuentre limitada en el intervalo (0,1), lo que permite que pueda ser intrepretada como una probabilidad.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{6.2}$$

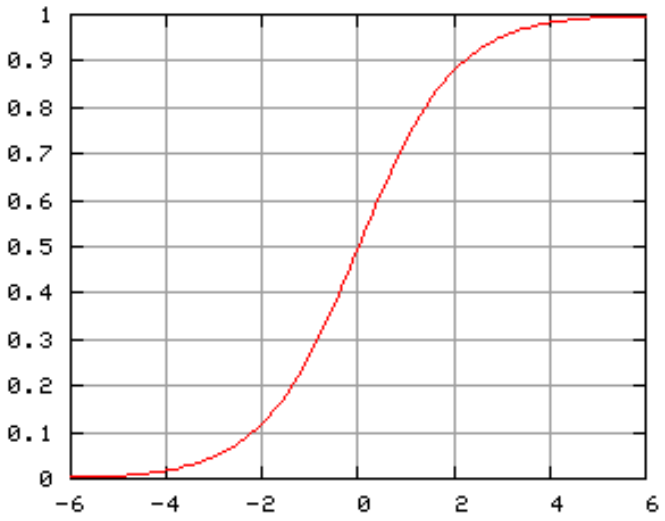


Figura 6.1: Función sigmoide.

El motivo por el que se emplea una función sigmoide y no una escalón, es que la sigmoide es derivable en todos los puntos. Aunque se trata de una función no lineal, la región de decisión que general es lineal, por eso está clasificado como un modelo lineal.

Como la función de error es convexa, se realizan métodos iterativos de descenso de gradiente para buscar minimizarlo.

Para este modelo, se emplea la siguiente función obtenida de la librería de Sklearn.

$$LogisticRegression()$$

6.1.2. Discriminante Lineal

Este modelo se basa en encontrar el hiperplano ideal que permita proyectar sobre él los datos, separando todo lo posible las diferentes clases.

Se busca una proyección en la que la distancia entre los datos de una misma clase sea la menor posible, lo que se correspondería con una varianza pequeña para cada clase. Además, se quiere que las medias proyectadas de los datos de cada clase, queden separadas lo máximo posible entre ellas.

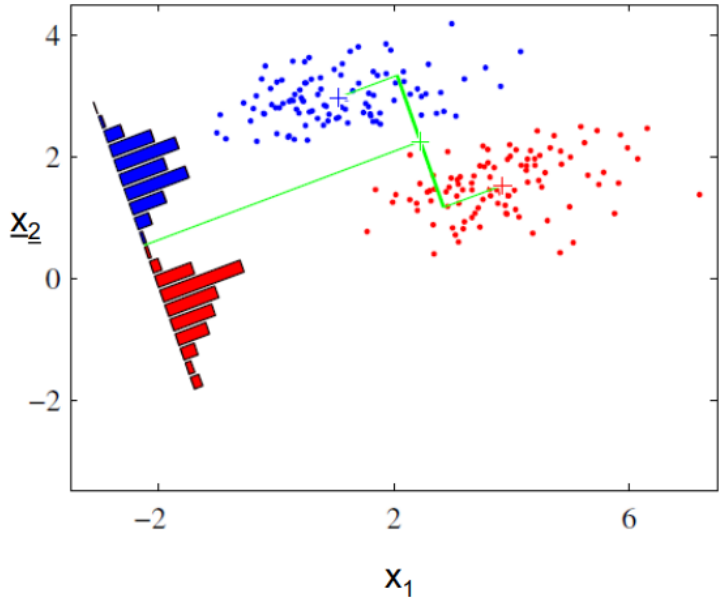


Figura 6.2: Proyección de los datos empleando el discriminante lineal de Fisher (Imagen extraída de [3]).

La función empleada para el discriminante lineal es:

$$LinearDiscriminantAnalysis()$$

6.2. Modelos no lineales

Lo normal es que estos modelos sean capaces de obtener mejores resultados que los modelos lineales ya que pueden separar los datos con regiones de decisión más complejas. Por norma general, suelen tener un mejor rendimiento que los lineales pero, como se verá en los resultados, no siempre es así.

6.2.1. KNN

Este método se basa en la idea de que los datos pertenecientes a una misma clase estarán más próximos en el espacio de representación.

Hay que tener en cuenta que cuanto mayor sea el valor del número de vecinos, más suave será la región de decisión y, en caso de un valor muy elevado, escogerá siempre la clase mayoritaria.

Este algoritmo genera regiones de decisión muy flexibles pero tiene como inconveniente una elevada sensibilidad al ruido.

Para explicar el funcionamiento de esta técnica, se dispone de la Figura 6.3. En esta, se muestra un dato en verde que es clasificado en función de los datos cercanos que tenga. Si se selecciona un valor de K=3, el dato sería clasificado como un triángulo rojo, ya que es la clase mayoritaria de entre los tres vecinos más cercanos. Seleccionando un k=5, la clase mayoritaria serían los cuadrados azules y, por ello, el dato sería clasificado como un cuadrado azul.

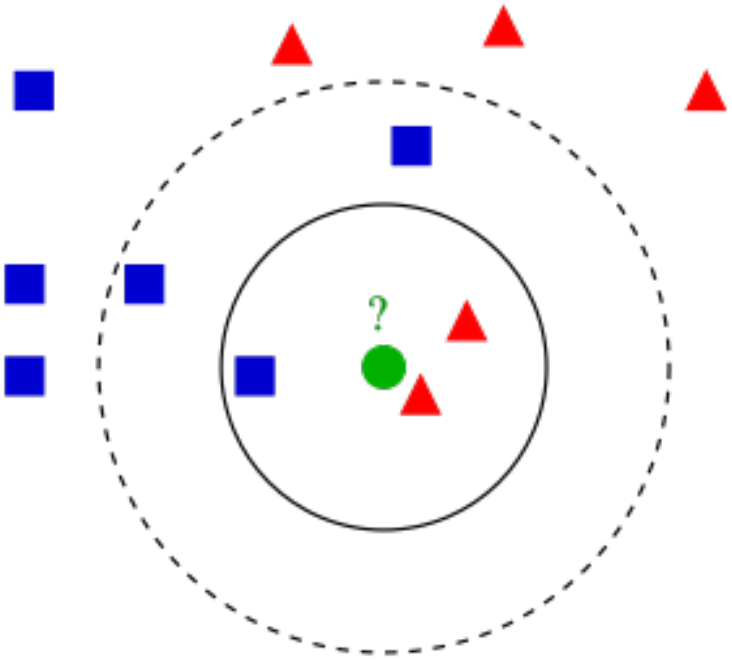


Figura 6.3: Clasificación en función del número de vecinos cercanos.

La función empleada para los K vecinos más cercanos es:

$$KNeighborsClassifier()$$

6.2.2. Árboles de decisión

Los árboles de decisión realizan una clasificación en forma de árbol. Disponen de dos tipos de nodos:

- Nodos de decisión: Asociados a una variable y con dos o más ramas que salen de él.
- Nodos de respuesta (hojas): Asociados a la clasificación final y devuelven la decisión final.

Para realizar una clasificación, se recorre el árbol desde el inicio hasta llegar a un

nodo de respuesta. Cada nodo de decisión representa un test sobre una variable del dato e indica la rama que se debe seguir.

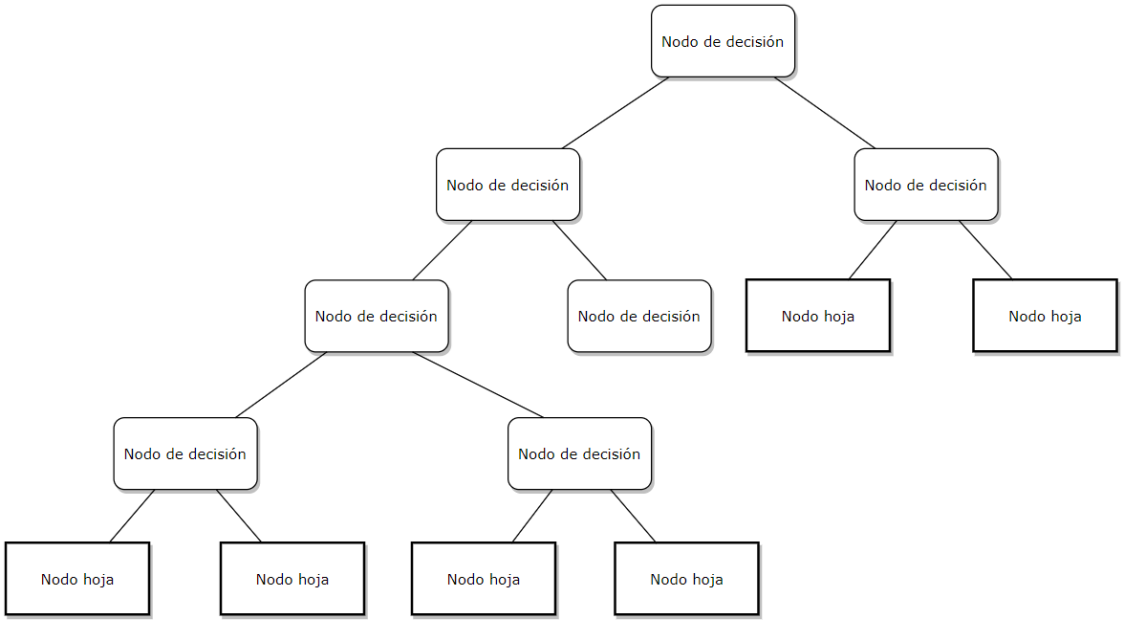


Figura 6.4: Ejemplo de estructura de un árbol de decisión.

La ventaja de esta técnica es que resulta sencillo interpretar las decisiones tomadas y permite manejar tanto variables numéricas como categóricas.

Como desventaja, tiende a sobreajustar, ser inestable y, en caso de una clase dominante, genera árboles sesgados.

La función empleada para el árbol de decisión es:

$$DecisionTreeClassifier()$$

6.2.3. Bosques aleatorios

A partir de muestras aleatorias del conjunto de entrenamiento, crea árboles de decisión. Para elegir a qué clase pertenece una muestra, se realiza una votación entre los árboles y la elección mayoritaria resulta seleccionada.

Este modelo es más difícil de interpretar que un árbol de decisión único. Normalmente, cuantos más estimadores se emplean, mejor resulta el bosque aleatorio pero también incrementa el coste computacional.

En la Figura 6.5 se puede ver como varía la región de decisión realizada en fun-

ción del número de árboles de decisión empleados. Disponer de más árboles permite realizar regiones de decisión más complejas.

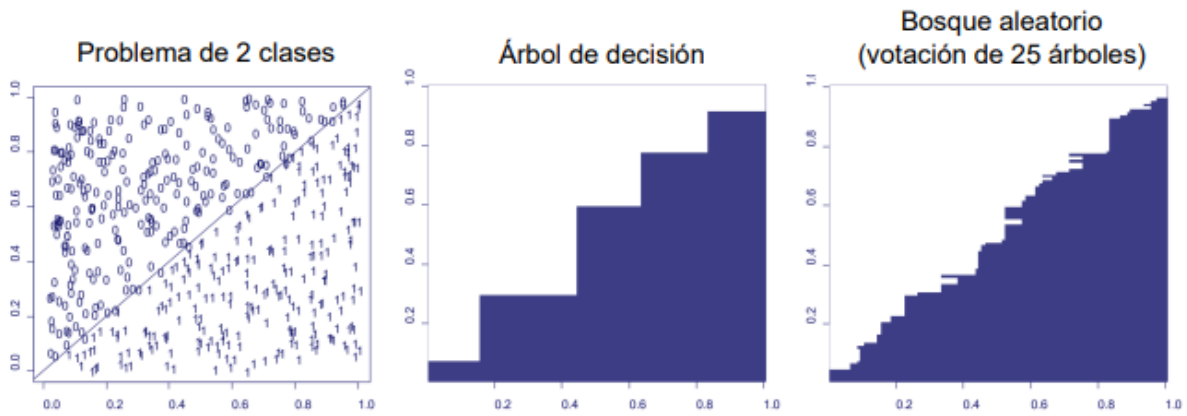


Figura 6.5: Diferencia según el número de árboles de decisión empleados (Imagen extraída de [4]).

La función empleada para los bosques aleatorios es:

$$RandomForestClassifier()$$

6.2.4. Máquinas de vectores de soporte

Mientras la mayoría de métodos se centran en minimizar los errores cometidos, las máquinas de vectores de soporte se basan en minimizar el riesgo estructural, disminuyendo así la posibilidad de cometer errores en el futuro. Para lograrlo, este modelo trata de separar seleccionando el hiperplano que tenga un mayor margen con los datos que toma como referencia, siendo estos conocidos como vectores de soporte.

Uno de los hiperparámetros que se modificarán en la búsqueda del mejor modelo posible, será el kernel. El truco del kernel consiste en convertir un espacio de una dimensión inferior no lineal en uno lineal de una dimensión superior. Según el hiperparámetro escogido, variará el resultado del espacio transformado.

En la parte izquierda de la Figura 6.6 se tiene un espacio en el que los datos están distribuidos de tal forma que no pueden ser separados con una región de decisión lineal. Aplicando una función kernel, se convierte el espacio de dos dimensiones en un espacio de tres (parte derecha de la Figura), pudiéndose en este último separar los datos empleando únicamente un hiperplano.

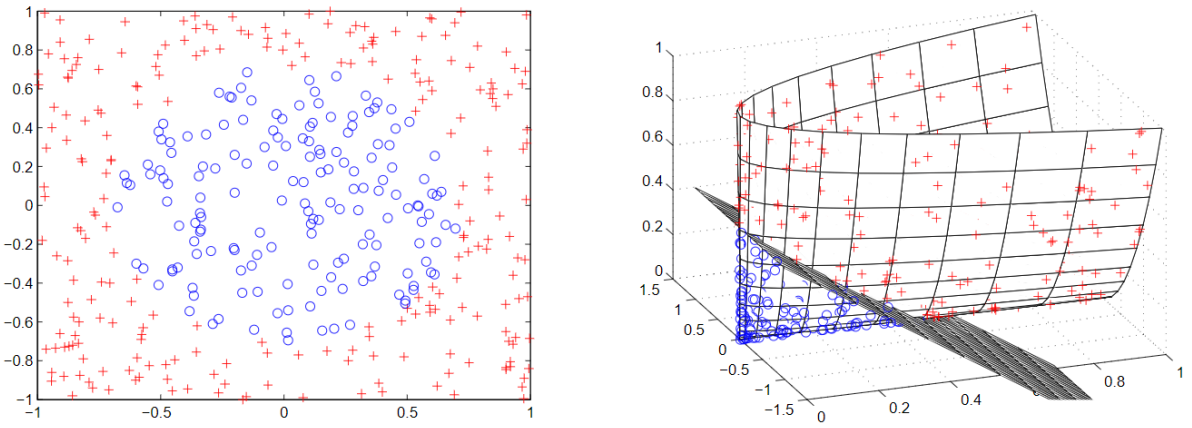


Figura 6.6: Truco del kernel.

La función empleada para las máquinas de vectores de soporte es:

$$SVC()$$

6.2.5. CNN

Este es el modelo principal de este proyecto ya que, al estar trabajando con imágenes, será el que proporcione un resultado mucho mejor que los demás.

Algunas de las capas más usadas son:

- Capa convolucional: Esta capa consiste en un conjunto de filtros que se aplican a la imagen. Estos, son activados al detectar alguna determinada característica en la imagen ya que su objetivo es extraer características relevantes de la misma. Las primeras capa convolucionales suelen ser responsables de detectar características de bajo nivel (como los bordes) y, las siguientes capas de convolución, se encargan de características de alto nivel. En la Figura 6.7 se muestra un ejemplo del funcionamiento de este tipo de capas. A partir de una imagen de una cara sonriente, se aplican diferentes filtros y se va logrando así extraer diferentes característica de ella, como por ejemplo, en la segunda de las tres imágenes azules en la que se detectan los ojos.

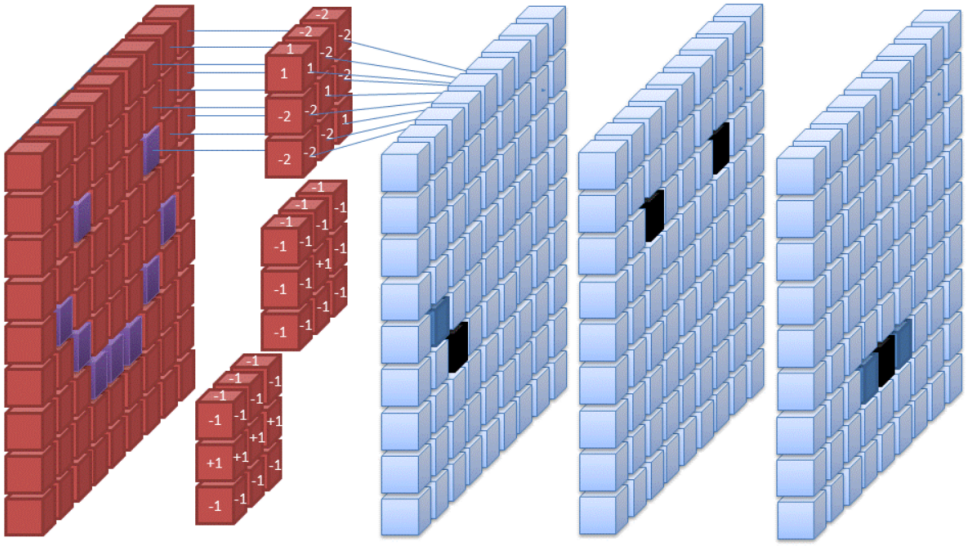


Figura 6.7: Funcionamiento de la capa convolucional.

- Capa RELU: Esta capa suele ir inmediatamente después de una capa convolucional pues transforma la operación lineal realizada por ella. Es una función de activación definida como el máx(0,x).

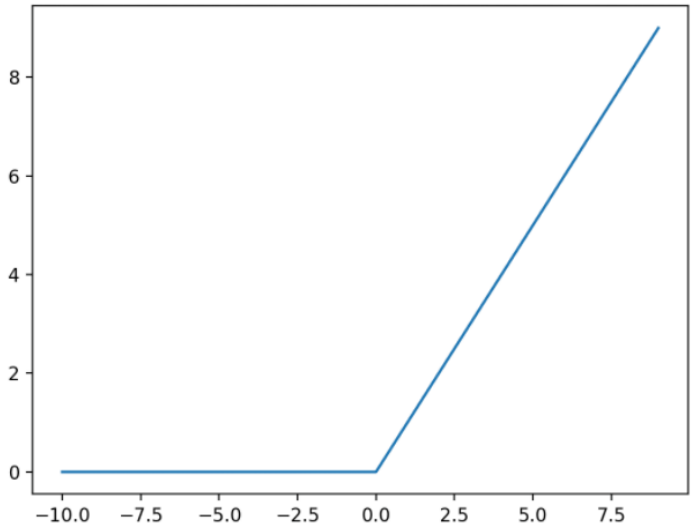


Figura 6.8: Función de activación de la capa RELU.

- Capa de agrupamiento: Reduce el tamaño a la salida de la capa convolucional. Esta capa sirve para controlar el sobreajuste y para conseguir transformaciones invariantes a rotación y traslación. En la siguiente imagen se muestra un ejemplo de agrupamiento por máximos.

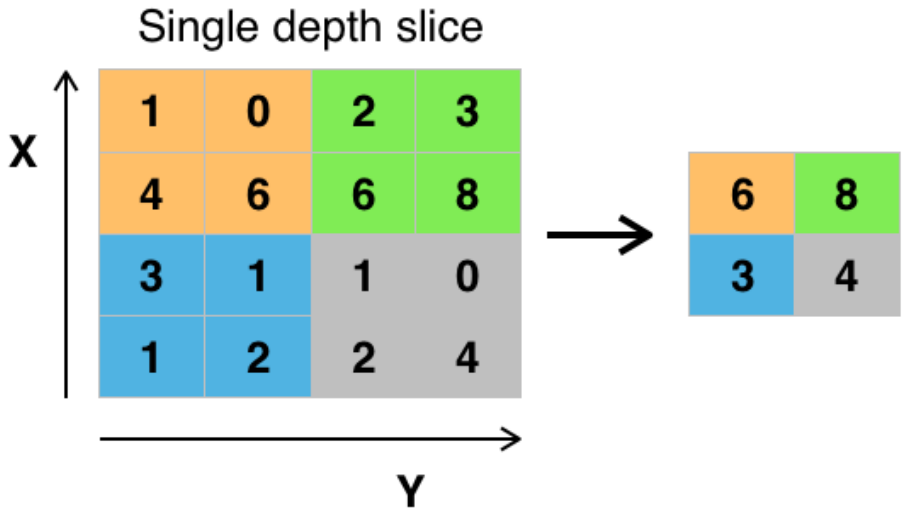


Figura 6.9: Capa de agrupamiento por máximos.

- Capa completamente conectada: Permite aprender combinaciones no lineales de las características de alto nivel entregadas por la capa convolucional.

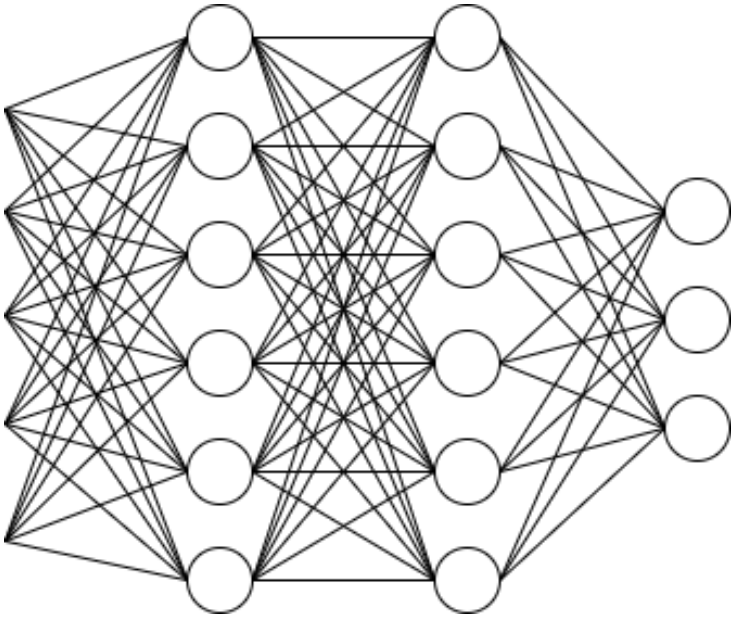


Figura 6.10: Capa completamente conectada.

Entrenar de cero una CNN llevaría un tiempo muy elevado y, al disponer de un dataset pequeño, sería muy difícil lograr un buen entrenamiento de todas las capas encargadas del preprocesado y de la extracción de características. Para lidiar con este problema, se decide realizar *transfer learning*.

Al realizar *transfer learning*, se carga un modelo ya preentrenado sin la última capa. Utilizar un modelo preentrenado permite que, todas las capas que se corresponden

con filtros y permiten extraer características de las imágenes, ya hayan sido entrenadas con un dataset mucho más grande. En el caso de las redes empleadas en este trabajo, han sido todas entrenadas previamente con el conjunto de imágenes de ImageNet. Para emplear estas redes ya preentrenadas, basta con descargar del repositorio de tensorflow la red que interese del apartado “feature_vector” y, añadirle al final una capa de tipo densa con el número de clases a clasificar.

Los modelos preentrenados usados se corresponden con cuatro arquitecturas diferentes:

- MobileNet V2: Esta arquitectura es introducida como una nueva arquitectura diseñada para entornos móviles o con recursos limitados. Para poder funcionar con recursos limitados, trata de mantener la precisión a la vez que reduce el número de operaciones y la memoria necesaria. La principal novedad de esta arquitectura es el módulo de capa residual invertido con cuello de botella lineal.
- ResNet-50: Las redes residuales saltan capas intermedias imitando el hecho de que las neuronas reales no siempre conectan con aquellas que son inmediatamente contiguas a ellas. En la Figura 6.11 se muestra un ejemplo de conexión entre sus capas.

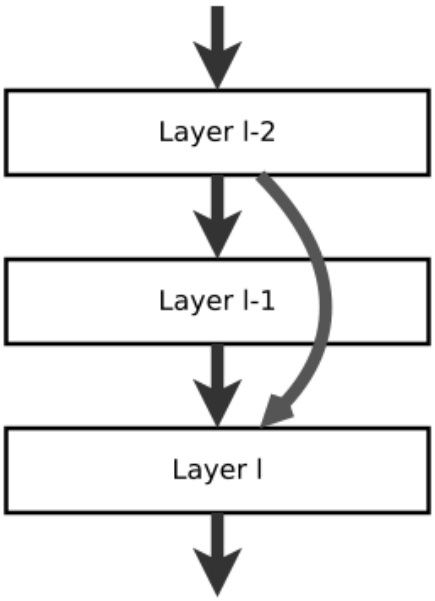


Figura 6.11: Conexión entre capas en ResNet(Imagen extraída de [2]).

- Inception V3: Modelo formado por bloques de construcción simétricos y asimétricos, con convoluciones, reducción promedio, reducción máxima, concatenaciones, retirados y capas completamente conectadas. En el modelo se emplea con frecuencia normalización por lotes y se aplica a las entradas de activación.

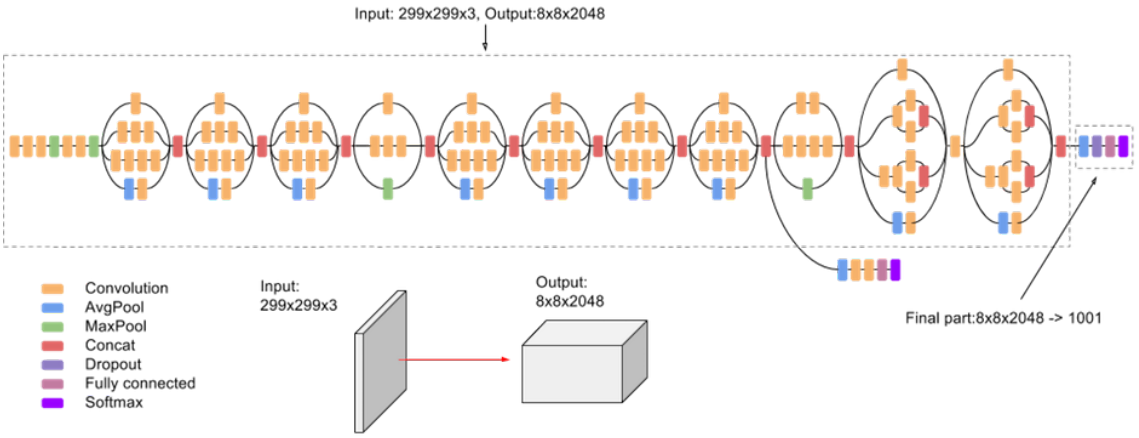


Figura 6.12: Estructura de Inception V3 extraída de la web de Google Cloud.

- EfficientNet: Esta arquitectura se basa en un nuevo método de escalado que que escala uniformemente la profundidad, el ancho y la resolución, empleando un coeficiente simple. Este modelo tiene 8 variaciones de la arquitectura, que van desde la B0 a la B7. En la Figura 6.13 se puede ver el resultado de diferentes redes frente al repositorio de ImageNet y como este modelo resulta más rápido y con mayor acierto. En este trabajo se emplea la versión B4, ya que es la que está en la curva de los resultados y, por tanto, la que tiene mejor resultado en proporción acierto-número de parámetros. Se probó también con el modelo B7 pero el ordenador no fue capaz de hacerlo funcionar.

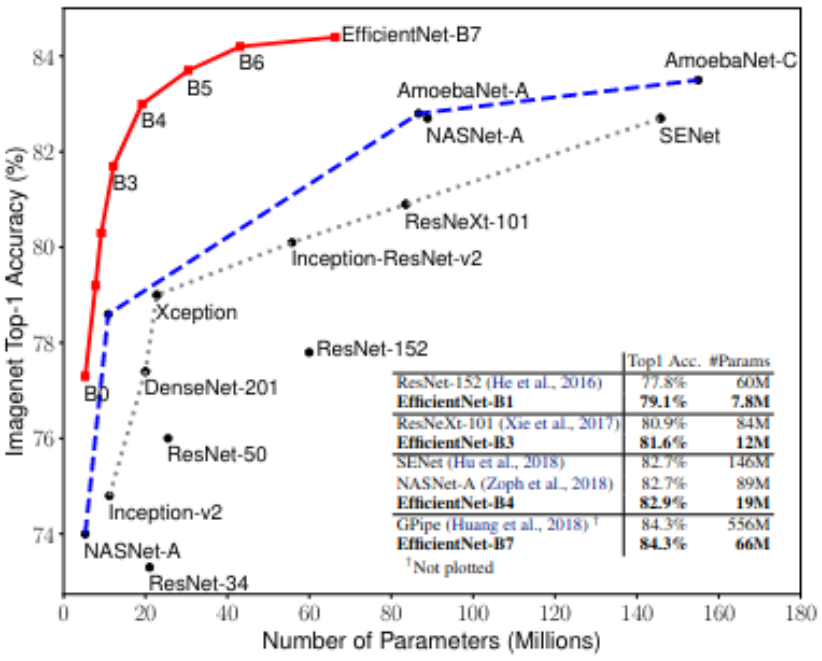


Figura 6.13: Comparación del acierto de diferentes modelos sobre el repositorio de ImageNet(Imagen extraída de [7]).

Al modelo preentrenado, se le añade con la función *tf.keras.Sequential()* una última capa que será la encargada de clasificar las imágenes en cada una de las diferentes clases disponibles.

El resultado de emplear *transfer learning*, es un modelo con un mayor acierto y un tiempo de entrenamiento mucho menor.

7. Experimentos

En este apartado, se describen los experimentos realizados para este trabajo. Primero se realiza una descripción del conjunto de datos creado, después se explican los diferentes preprocesados empleados para cada técnica. Por último, se explica la búsqueda de hiperparámetros para cada uno de los modelos.

7.1. Conjunto de datos

El conjunto de datos empleado en este trabajo constará de 577 fotografías realizadas por el autor de este proyecto. Estas, estarán divididas en siete clases.

En cada una de las fotografías, se ve al autor realizando un gesto con su mano derecha y, este gesto, será el que deberá ser identificado gracias a las técnicas de aprendizaje automático.

Las características del dataset son:

- 7 gestos diferentes:
    - Número 1 realizado con el dedo índice.
    - Número 2 realizado con los dedos índice y corazón.
    - Número 3 realizado con los dedos índice, corazón y anular.
    - Número 4 realizado con todos los dedos excepto el pulgar.
    - Número 5 con todos los dedos de la mano extendidos.
    - Pulgar hacia arriba.
    - Pulgar hacia abajo.
- 577 fotografías distribuidas de la siguiente forma:
    - Número 1: 88 fotografías.
    - Número 2: 70 fotografías.
    - Número 3: 76 fotografías.
    - Número 4: 78 fotografías.
    - Número 5: 90 fotografías.
    - Pulgar hacia arriba: 99 fotografías.
    - Pulgar hacia abajo: 76 fotografías.

- Las fotografías se encuentran en formato jpg.
- Han sido tomadas en diferentes condiciones de iluminación.
- La resolución de las fotografías es de 1280x720.

A continuación se muestran tres fotografías de cada uno de los diferentes gestos. Para cada gesto, se muestran tres imágenes con los tres diferentes tipos de iluminación probados.

Con las siguientes fotografías se puede tener una idea general de todo el conjunto de imágenes disponible:

Figura 7.1: Gesto del número 1.

Figura 7.2: Gesto del número 2.

Figura 7.3: Gesto del número 3.

Figura 7.4: Gesto del número 4.

Figura 7.5: Gesto del número 5.

Figura 7.6: Gesto del pulgar hacia arriba.

Figura 7.7: Gesto del pulgar hacia abajo.

Como se puede ver, en todas las fotografías aparece el autor además del gesto, ya que la finalidad de esta clasificación será obtener un modelo entrenado que, en caso de recibir una foto realizada por la webcam, interprete el gesto. Por ello resulta importante que las fotografías coincidan con la imagen que se obtendrían la cámara de un usuario que estuviese realizando el gesto.

7.2. Preprocesado de las imágenes

Para este apartado, es muy importante tener en cuenta que las imágenes no serán preprocesadas de la misma forma para todos los modelos.

Para todos los modelos CNN empleados, se introducen las imágenes a color y redimensionadas. En la Figura 7.8, se muestran las dimensiones empleadas en las imágenes para cada modelo de CNN. Estas dimensiones aparecen como las recomendadas en el repositorio de Tensorflow para cada uno de los modelos.

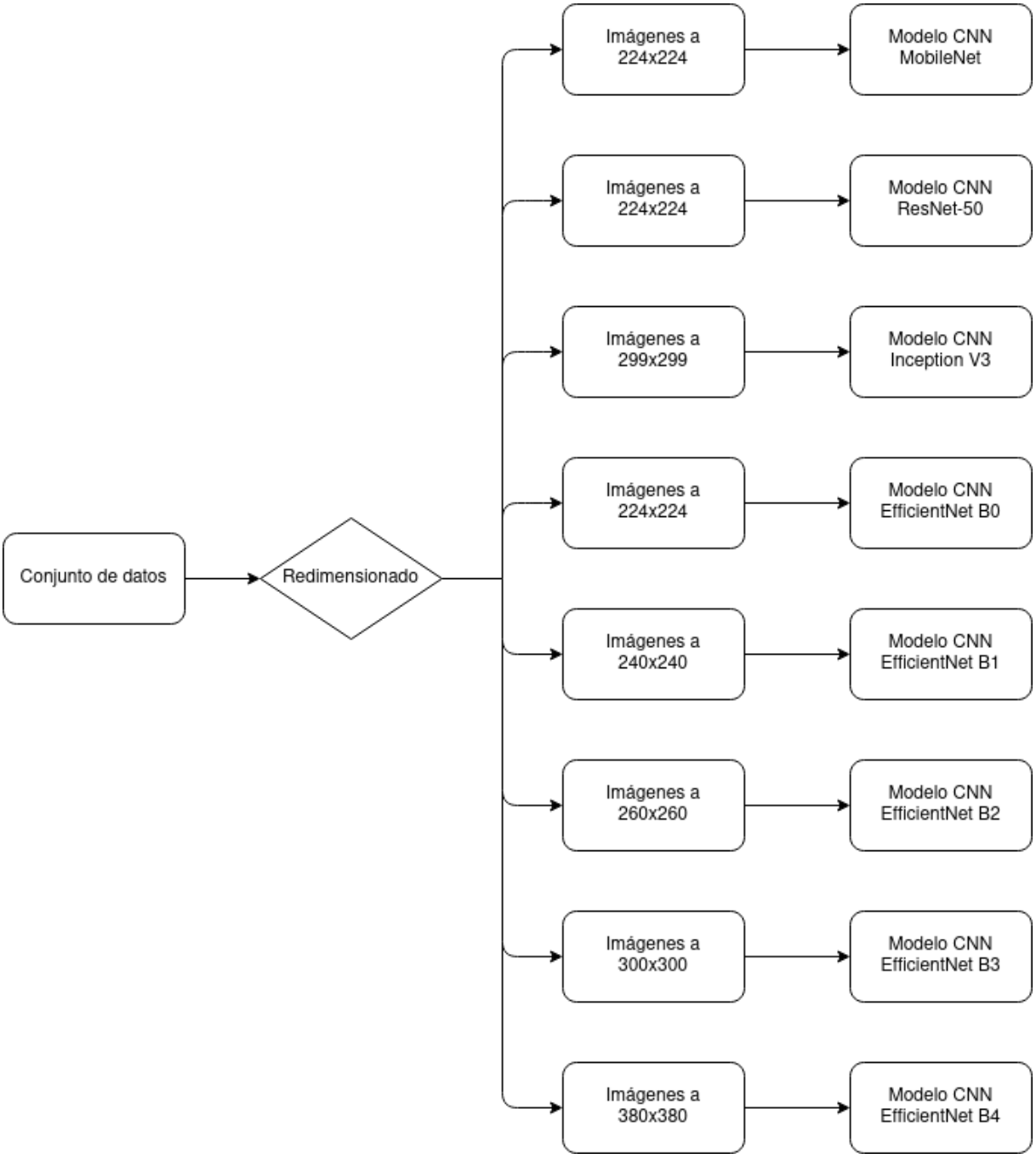


Figura 7.8: Escalado para los modelos CNN

Para el resto de modelos, las dimensiones de las imágenes empleadas son 64x64. Se probaron otras dimensiones y no se logró mejorar los resultados. Para cada modelo, exceptuando los modelos CNN, se realizan los experimentos con dos diferentes preprocesados:

- Imágenes a 64x64 con el detector de bordes Canny.
- Imágenes 64x64 con el descriptor HOG.

A las 577 imágenes de partida se les realiza un efecto espejo empleando la función *cv2.flip*. Así, se logra duplicar el número de imágenes disponibles hasta tener 1154 y, será a estas últimas, a las que se les apliquen el filtro detector de bordes y el descriptor HOG.

Las imágenes con los filtros aplicados son guardadas en carpetas separadas para que, en el momento del entrenamiento, únicamente haya que leer el contenido de las carpetas.

7.2.1. Detección de bordes

Para poder realizar la detección de bordes tratando de encontrar unos valores para el filtro a los que no afecten demasiado las variaciones lumínicas del dataset, se emplea una función de la librería de visión artificial OpenCV.

En OpenCV existe un filtro de detección de bordes llamado *Canny*. Este filtro realiza una reducción de ruido gaussiano y un filtro de detección de bordes Sobel. Aunque se podrían aplicar estos filtros por separado mediante la librería *Sklearn*, se elige usar esta función ya que permitirá fácilmente iterar en función de dos de sus parámetros principales para buscar unos valores que ofrezcan un resultado aceptable en todas las condiciones lumínicas.

Lo único que será necesario es pasar las imágenes a escala de grises antes de la aplicación del filtro. En otras condiciones podría ser recomendable realizar un filtrado gaussiano adicional del que va incluido en la propia función pero, al realizar la prueba sin añadirlo se obtuvieron resultados satisfactorios y se decidió no añadir ningún filtrado de ruido a mayores. El principal problema al realizar filtrados de ruido es que las imágenes se vuelven borrosas y, en el caso de las imágenes con poca luz, se puede perder completamente el gesto realizado.

La función *Canny* es la siguiente:

$$cv2.Canny(imagen, hist1, hist2, apertureSize = 3)$$

Los parámetros que se varían son *hist1* e *hist2*, ya que variando estos parámetros se elige el rango en el que debe estar el valor de un píxel para ser seleccionado como perteneciente a un borde.

Para asignar los valores al umbral de histéresis hay que saber cómo funciona. Se asigna un valor máximo y uno mínimo. Si el valor de un píxel es mayor que el valor máximo, este pertenecerá al límite, mientras que, si es inferior al valor mínimo, no lo

hará. Por último, encaso de que un píxel tenga un valor situado entre el máximo y el mínimo, formará parte del límite siempre que esté en contacto con otro píxel perteneciente a él.

En la Figura 7.9 se muestran los cuatro pasos que ejecuta la función del detector de bordes *Canny*.

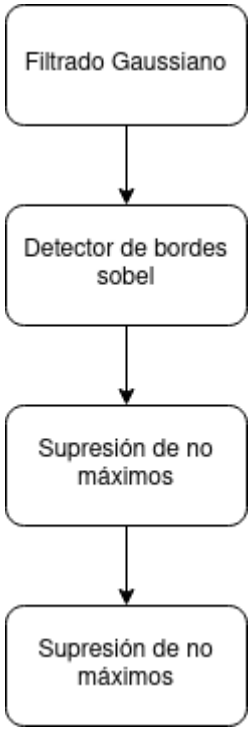


Figura 7.9: Pasos de la detección de bordes *Canny*

Los valores seleccionados finalmente serán 40 y 60 para *hist1* e *hist2* respectivamente. Aunque estos valores son un poco agresivos y detectan muchos bordes en exceso, son los más apropiados teniendo en cuenta la existencia de imágenes con poca luz. En las Figuras 7.10 a 7.12 se muestra el resultado obtenido (las imágenes se muestran más grandes para permitir observar mejor la detección de bordes realizada y cómo los bordes se vuelven mucho menos nítidos en la última imagen):

Figura 7.10: Detección de bordes *Canny* para el gesto 5 con luz intermedia.

Figura 7.11: Detección de bordes *Canny* para el gesto 5 con mucha luz.

Figura 7.12: Detección de bordes *Canny* para el gesto 5 con poca luz.

La gran ventaja de utilizar un filtro que deje solo los bordes detectados en blanco y ponga el resto de la imagen en negro es que mucha de la información que no resultaría relevante para el entrenamiento se está eliminado, facilitando así la detección.

Además del filtro detector de bordes, se prueba otro preprocesado distinto para disponer de más resultados con los que comparar. Para este otro preprocesado se emplea el descriptor HOG.

7.2.2. Descriptor HOG

El algoritmo del descriptor HOG realiza los siguientes pasos:

1. Preprocesado: Normaliza el color y realiza un escalado. Se recomienda un tamaño de imagen de 64x128 o 64x64.
2. Cálculo de los gradientes de intensidad, horizontales y verticales, en celdas de tamaño 8x8.
3. Cálculo de la magnitud y la orientación de los gradientes por cada celda.
4. Compresión reduciendo el número de vectores de gradientes por celda.
5. Obtención de un vector final de características.

Este descriptor resulta rápido y preciso en muchos casos y suele ser empleado para reconocimiento y clasificación de objetos, por eso se decide probar a emplearlo para clasificar gestos.

A continuación se muestran como se ven las fotografías una vez se ha aplicado el filtro HOG. En este trabajo se emplearán redimensionadas a 64x64.

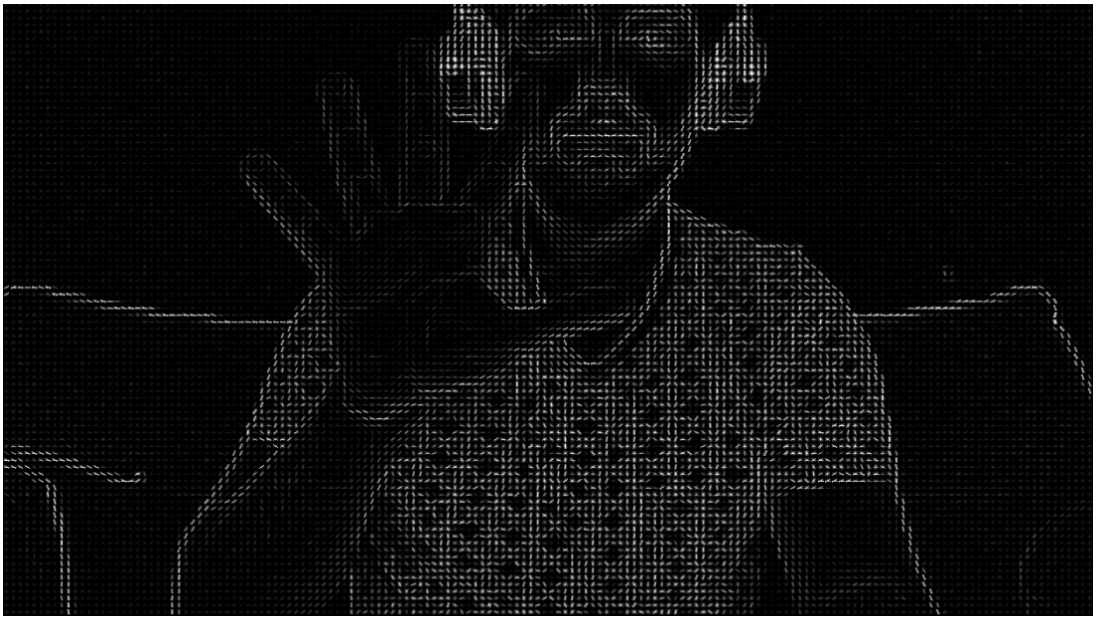


Figura 7.13: Descriptor HOG para el gesto 5 con luz intermedia.

Figura 7.14: Descriptor HOG para el gesto 5 con mucha luz.

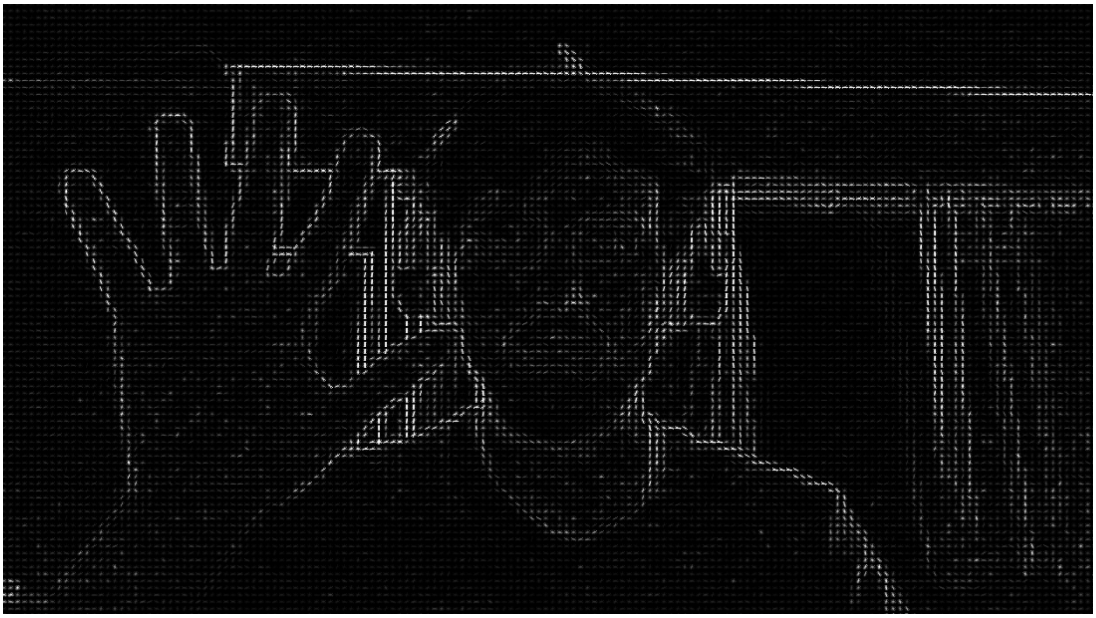


Figura 7.15: Descriptor HOG para el gesto 5 con poca luz.

Una vez aplicado el descriptor, las imágenes quedan en escala de grises, por lo que también proporciona la ventaja de trabajar con un volumen inferior de datos.

7.3. Búsqueda de hiperparámetros

El proceso de búsqueda de hiperparámetros se ha automatizado para todos los métodos menos para CNN, ya que al hacer *transfer learning* no hay hiperparámetros relevantes de los que hacer un barrido.

Para realizar el barrido en busca de hiperparámetros, se emplea la función *RandomizedSearchCV()* y se le asigna el número de combinaciones máximo que deben ser probadas.

$$RandomizedSearchCV()$$

La función *RandomizedSearchCV*, realiza validación cruzada en el modelo variando los hiperpárametros de forma aleatoria entre los valores de un diccionario que se le asigna. Esta función devuelve la mejor combinación de hiperparámetros para un modelo.

Aunque en la red CNN no se use esta función, también se realiza validación cruzada.

Las combinaciones probadas para cada método se guardan en un excel junto con los resultados de su prueba, teniéndose así una hoja de cáculo con los hiperparámtros

probados en cada modelo.

Como hay algunos hiperparámetros que son incompatibles con otros, si se realiza la función *RandomizedSearchCV()* pueden producirse fallos. En función de esto, se seleccionan para probar los hiperparámetros que son compatibles y que proporcionan un mayor número posible de combinaciones. Algunas combinaciones con los hiperparámetros que no fueron seleccionados se probaron de forma manual por separado, pero en ninguno de los casos comprobados se obtuvo un resultado que mejorara significativamente el modelo.

A continuación se muestran los hiperparámetros que se han variado en cada modelo y los valores que han podido ser seleccionados. Siempre que la opción está disponible, se pone el valor *n_jobs* a "-1". Así se acelera en gran medida el proceso, dedicándole más procesadores.

7.3.1. Regresión Logística

El diccionario que se pasa a la función es:

$$solver : [newton - cg, lbfgs]$$
$$penalty : [l2, none]$$

- *Solver:* Algoritmo que se usa en el problema de optimización.
  - *Newton-cg:* Algoritmo de optimización de gradientes conjugados de Newton.
  - *Lbfgs:* Algoritmo de optimización similar al Broyden-Fletcher-Goldfarb-Shanno (BFGS) pero empleando una cantidad limitada de memoria.
- *Penalty:* Se emplea para especificar la norma empleada en la penalización.
  - *L2.*
  - *None.*
- *Max_iter:* Indica el valor de iteraciones máximo hasta converger. Lo único importante al seleccionar este valor es seleccionar uno lo bastante grande para que el modelo no entregue un error y llegue a converger. En este caso se seleccionan mil iteraciones.
- *Multi_class:* En este hiperparámetro se hace la selección de su valor en función del tipo de problema que se esté tratando.
  - *Multinomial:* Se selecciona este ya que es el más apropiado al trabajar un problema de varias clases.

7.3.2. Discriminante Lineal

El diccionario que se pasa a la función es:

$$solver : [lsqr, svd]$$

- *Solver:* Algoritmo que se emplea en el problema de optimización.
  - *Lsqr:* Algoritmo de optimización basado en mínimos cuadrados.
  - *Svd:* Algoritmo de optimización de descomposición en valores singulares.

En las primeras pruebas se modificaba también el parámetro *shrinkage* y se probaban los algoritmos de optimización “sag” y “saga”, pero ralentizaban mucho el entrenamiento y no mejoraban los resultados.

7.3.3. K vecinos más cercanos

El diccionario que se pasa a la función es:

$$\begin{aligned} &n\_neighbors : list(range(1, 50)) \\ &weights : [uniform, distance] \\ &algorithm : [ball\_tree, kd\_tree, brute] \\ &leaf\_size : [5, 7, 11, 15, 20, 30, 40, 50, 70, 80, 90] \\ &p : [1, 2] \end{aligned}$$

- *N_neighbors:* Número de vecinos empleados en el modelo.
- *Weights:* Función de peso usada en las predicciones.
  - *Uniform:* Todos los puntos en cada vecindario tienen el mismo peso.
  - *Distance:* Los vecinos más cercanos al punto tienen mayor influencia que los más alejados.
- *Algorithm:* Algoritmo empleado para el modelo.
  - *Ball_tree:* Las particiones de los datos se hace en un conjunto de hiperesferas anidadas.
  - *Kd_tree:* Este tipo de árbol solo emplea planos perpendiculares a uno de los ejes de coordenadas.
  - *Brute:* Realiza la búsqueda por fuerza bruta.

- *Leaf_size:* Tamaño de hoja pasado para el árbol del algoritmo.
- *P:* Parámetro para la medida *Minkowski*.
- *1:* Se emplea la distancia Manhattan.
- *2:* Se emplea la distancia euclídea.

En este método se establece un máximo de 30 combinaciones a probar.

7.3.4. Árboles de decisión

El diccionario que se pasa a la función es:

$$
\begin{aligned}
&splitter : [best, random] \\
&max\_features : [sqrt, log2] \\
&min\_samples\_split : [5, 7, 11, 15, 20, 30, 40, 50, 70, 80, 90] \\
&min\_samples\_leaf : [1, 5, 7, 11, 15, 20, 30, 40, 50, 70, 80, 90] \\
&criterion : [gini, entropy]
\end{aligned}
$$

- *Splitter:* Estrategia empleada para separar cada nodo.
  - *Best:* Escoge la mejor separación.
  - *Random:* Escoge una sepración aleatoria.
- *Max_features:* Número de características a considerar al buscar por la mejor separación.
  - *Sqrt:* La raíz cuadrada del número de características.
  - *Log2:* Logaritmo en base 2 del número de características.
- *min_samples_split:* Número mínimo de muestras para separar un nodo interno.
- *min_samples_leaf:* Número mínimo de muestras necesarias para estar en un nodo hoja.
- *Criterion:* Función para medir la calidad de la separación.
  - *Gini:* Para emplear la impureza con el índice de Gini.
  - *Entropy:* Para emplear la ganancia de información.
- *Max_depth:* Máxima profundidad del árbol.

7.3.5. Bosques aleatorios

El diccionario que se pasa a la función es:

$$splitter : [best, random]$$
$$max\_features : [sqrt, log2]$$
$$min\_samples\_split : [5, 7, 11, 15, 20, 30, 40, 50, 70, 80, 90]$$
$$min\_samples\_leaf" : [1, 5, 7, 11, 15, 20, 30, 40, 50, 70, 80, 90]$$
$$bootstrap : [True, False]$$
$$criterion : [gini, entropy]$$

- *Splitter:* Estrategia empleada para separar cada nodo.
  - *Best:* Escoge la mejor separación.
  - *Random:* Escoge una separación aleatoria.
- *Max_features:* Número de características a considerar al buscar por la mejor separación.
  - *Sqrt:* La raíz cuadrada del número de características.
  - *Log2:* Logaritmo en base 2 del número de características.
- *min_samples_split:* Número mínimo de muestras para separar un nodo interno.
- *min_samples_leaf:* Número mínimo de muestras necesarias para estar en un nodo hoja.
- *Bootstrap:* Si se pone a falso se emplea todo el dataset para construir cada árbol y si se pone a verdadero no.
  - *True.*
  - *False.*
- *Criterion:* Función para medir la calidad de la separación.
  - *Gini:* Para emplear la impureza con el índice de Gini.
  - *Entropy:* Para emplear la ganancia de información.

7.3.6. Máquinas de vectores de soporte

$$kernel : [linear, poly, rbf, sigmoid]$$
$$gamma : [0{,}01, auto]$$
$$decision\_function\_shape : [ovo, ovr]$$

- *Kernel:* Tipo de kernel empleado en el algoritmo.
  - *Linear:* Función lineal.
  - *Poly:* Función polinómica.
  - *Rbf:* Función de base radial.
  - *Sigmoid:* Función sigmoide.
- *Gamma:* Coeficiente para el kernel.
  - *0.01:* Se probaron a mano muchos valores y este era de los que mejores resultados obtenía, por eso se añadió a la búsqueda.
  - *Auto*
- *Decision_function_shape:* Función de decisión.
  - *Ovo: One vs One*.
  - *Ovr: One vs Rest*.

8. Entrenamientos realizados

En este trabajo se realizan 20 modelos y se comparan entre ellos para hallar el mejor modelo posible para el problema planteado.

En la Figura 8.1 se muestran todos los modelos de clasificación supervisada que se han realizado junto con su preprocesado.

MODELOS ENTRENADOS		
MODELOS CON PREPROCESADO DETECTOR DE BORDES	**MODELOS CON PREPROCESADO DESCRIPTOR HOG**	**MODELOS CON SOLO ESCALADO EN EL PREPROCESADO**
• Regresión Logística • Discriminante lineal • KNN • Árboles de decisión • Bosques aleatorios • Máquina de vectores de soporte(SVM)	• Regresión Logística • Discriminante lineal • KNN • Árboles de decisión • Bosques aleatorios • Máquina de vectores de soporte(SVM)	• CNN-MobileNet • CNN-ResNet50 • CNN-Inception V3 • CNN-EfficientNet B0 • CNN-EfficientNet B1 • CNN-EfficientNet B2 • CNN-EfficientNet B3 • CNN-EfficientNet B4

Figura 8.1: Todos los modelos realizados de clasificación supervisada.

Una vez realizada la búsqueda de hiperparámetros para los modelos con el preprocesado con el detector de bordes y para los modelos con el preprocesado con el descriptor HOG, se emplea la mejor combinación en los hiperparámetros para realizar la validación cruzada.

Como el entrenamiento de estos modelos lleva poco tiempo, se realiza la validación cruzada dividiendo el conjunto de datos en 10 partes iguales. De esta manera, cada modelo se entrena diez veces usando nueve de los diez conjuntos para el entrenamiento y el conjunto restante para el test.

En el caso de los modelos con la red CNN, se prueban manualmente variaciones

del optimizador en algunos de ellos. El que mejor resultado entrega suele ser el optimizador “adam” por lo que se selecciona como algoritmo de optimización para estos modelos. Estos modelos tardan mucho más en su entrenamiento, por lo que se decide realizar la validación cruzada dividiendo el conjunto de datos en 5 partes.

Una vez realizada la validación cruzada de cada modelo, se extraen sus métricas de evaluación y quedan guardadas en la hoja de cálculo correspondiente para cada modelo. También se guardan los valores medios obtenidos de la validación cruzada de cada modelo para facilitar la comparación entre ellos.

Terminado el proceso de validación cruzada, se realiza un entrenamiento de cada modelo para poder obtener así otros métodos de evaluación como son la matriz de confusión y la curva ROC.

Se extrae la matriz de confusión de todos los modelos menos de los de la red CNN y se extraen todas las curvas ROC menos las de la máquina de vectores de soporte. En estos casos el programa falla.

En la Figura 8.2 se muestra todo el proceso para facilitar la comprensión de los pasos realizados.

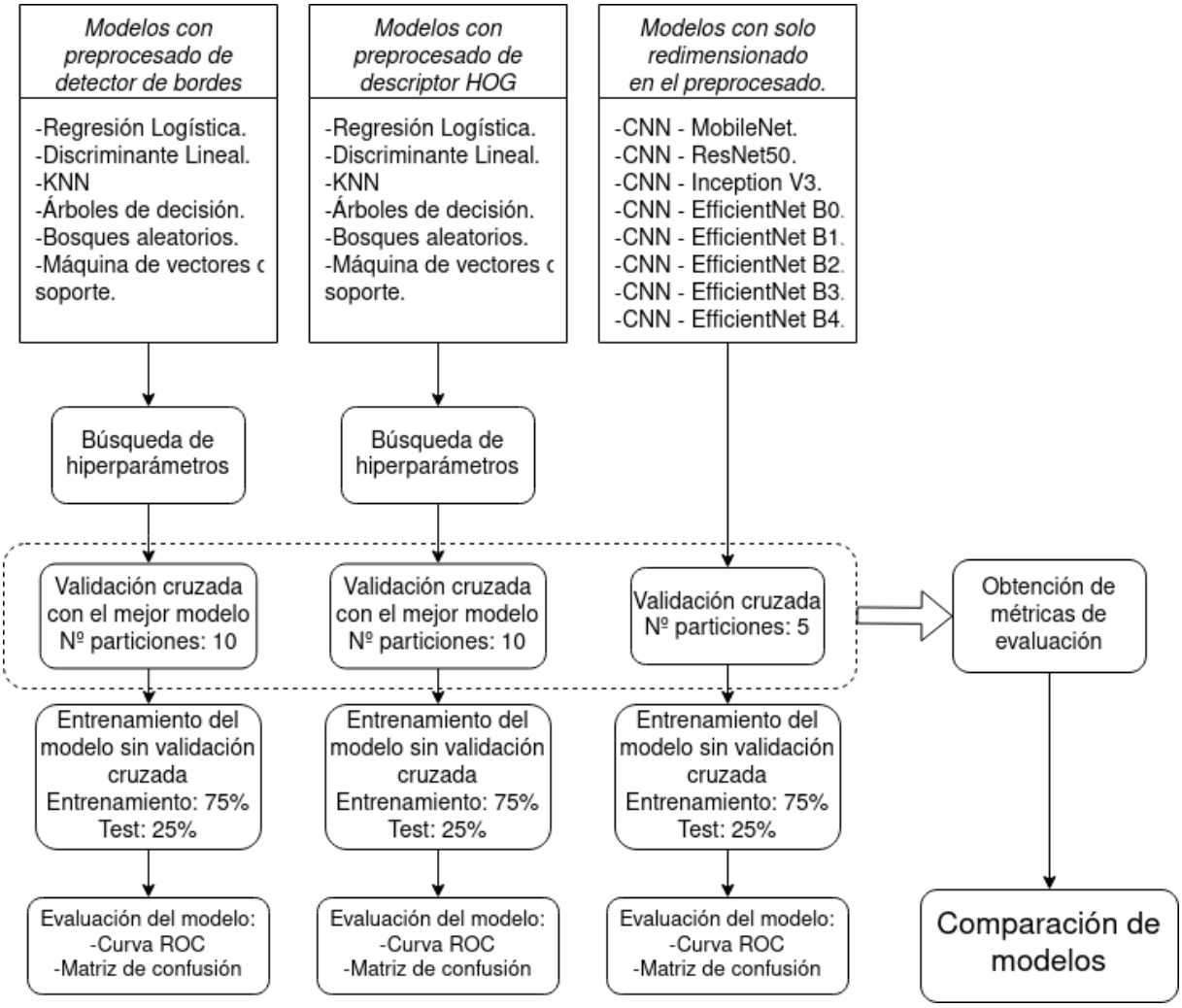


Figura 8.2: Proceso realizado.

Los modelos de las redes convolucionales se entrenan cada un con 80 epochs. Como el proceso de entrenamiento se hace muy largo, se implementa aceleración por GPU empleando la librería de *tensorflow-gpu* y CUDA. Con esto, se reduce por mucho el tiempo de entrenamiento.

Para el entrenamiento final de la red CNN (el que se realiza una vez se ha realizado validación cruzada y se han obtenido las métricas de evaluación), un 75 % de las 1154 imágenes disponibles se emplean para el entrenamiento, y un 25 % para la validación. Para dividir el conjunto de entrenamiento, se hace uso de la función:

$$train\_test\_split(datos, test\_size, train\_size)$$

9. Medidas

Para poder explicar los diferentes modelos y su resultado, se explican brevemente las principales métricas de evaluación de modelos:

- Exactitud (Accuracy): Tasa de acierto global del sistema.

$$Exactitud = \frac{Aciertos\ totales}{Casos\ totales} \tag{9.1}$$

- Sensibilidad (*Recall*): De todos los elementos de una clase, cuanto son detectados como pertenecientes a ella.

$$Sensibilidad = \frac{Casos\ detectados\ como\ una\ clase}{Casos\ totales\ de\ una\ clase} \tag{9.2}$$

- Especifidad: Número de casos clasificados como negativos de todos los casos negativos.

$$Especifidad = \frac{Aciertos\ de\ casos\ negativos}{Casos\ negativos\ totales} \tag{9.3}$$

- Precisión: De todos los elementos que son identificados como pertenecientes a una clase, cuantos pertenecen realmente a esa clase.

$$Precisión = \frac{Aciertos\ de\ los\ casos\ clasificados\ como\ una\ clase}{Casos\ clasificados\ como\ una\ clase} \tag{9.4}$$

- F1-score: Es la media armónica entre la precisión y la sensibilidad.

$$F1 - score = 2 * \frac{Precisión\ \cdot\ Sensibilidad}{Precisión\ +\ Sensibilidad} \tag{9.5}$$

Para problemas multiclase, la exactitud global suele calcularse directamente pero, el valor global de las demás métricas, puede ser calculado usando micropromedio, promedio ponderado o macropromedio.

- Macropromedio: Calcula el promedio de la métrica entre todas las clases. Este método no tiene en cuenta el desbalanceo de clases.
- Promedio ponderado: Calcula el promedio de la métrica pero realizando una ponderación por el número de casos de cada clase.

- Micropromedio: En cada métrica se suma de forma conjunta los verdaderos positivos, falsos positivos y falsos negativos para cada clase.

Como en este problema se trabaja con clases balanceadas, la diferencia entre las métricas usar la métrica micro o macro no resulta significativa.

Otros valores que se extraeran y que son determinantes a la hora de escoger un modelo u otro son el *fit time* y el *score time*.

El *fit time* es el tiempo que tarda en completarse el entrenamiento. Algunos modelos pueden llegar a tener tiempos de entrenamiento tan elevados que resulta muy difícil trabajar con ellos sin el equipo apropiado.

El *score time* es el tiempo que le lleva evaluar el conjunto de test para cada partición de la validación cruzada. Como en los modelos que no son redes CNN se divide el conjunto en 10 particiones al realizar la validación cruzada, el valor del *score time* se referirá a 114 imágenes en esos casos. En el caso de las redes CNN, como se trabaja con una validación cruzada dividida en 5 partes, el *score time* será el tiempo que tarda en etiquetar 228 imágenes.

9.1. Curva ROC

Para evaluar los modelos obtenidos se realiza la curva ROC en todos ellos menos en la máquina de vectores de soporte ya que mostraba un error.

La curva ROC muestra la sensibilidad frente a la tasa de falsos positivos. Cuanto más cercana sea la línea representada a una recta, más aleatoria (y por tanto peor) será la clasificación. Se emplea este método de evaluación porque tiene la ventaja de ser insensible a distribuciones no balanceadas entre las clases.

9.2. Matriz de confusión

Otro método para representar una evaluación del modelo consiste en enfrentar la clase real con la clase predicha, formándose así una matriz que, en caso de concentrar más valores en las posiciones de su diagonal principal, el modelo habrá tenido más acierto.

10. Resultados

10.1. Evaluación de los modelos resultantes

En una hoja de excel, se recogen los resultados la evaluación de los modelos seleccionados por la búsqueda de hiperparámetros. Como el proceso está automatizado, no es seguro que siempre se seleccione el mismo modelo, por ello las comparaciones pueden diferir de una ejecución del código a otra, pero nunca en gran medida.

Recordatorio para la interpretación de los datos:

- Clase 0: Pulgar hacia arriba.
- Clase 1: Número 1.
- Clase 2: Número 2.
- Clase 3: Número 3.
- Clase 4: Número 4.
- Clase 5: Número 5.
- Clase 6: Pulgar hacia abajo.

10.1.1. Evaluación de la regresión logística

En la Figura 10.1, se muestra la curva ROC del modelo de regresión logística con preprocesado *canny*. De los modelos que usaron este tipo de preprocesado, es de los que obtiene mejores resultados.

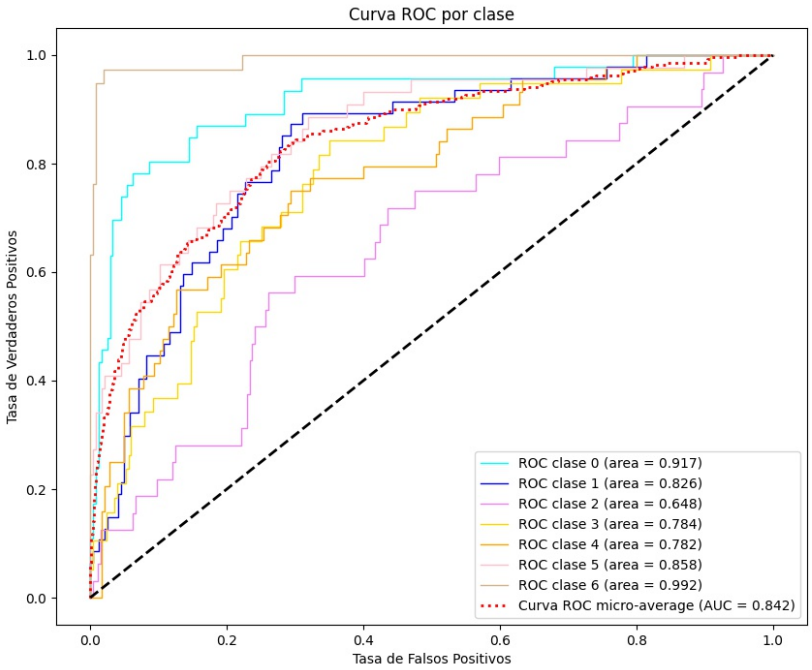


Figura 10.1: Curva ROC del modelo de regresión logística con preprocesado canny.

Mirando su matriz de confusión de la Figura 10.2 se puede ver cómo la clase 2 no es capaz de clasificarla correctamente, coincidiendo con la curva ROC que indica que es la que peor clasifica. Sin embargo, la clase 6, que es la que muestra casi perfecta en la curva ROC, se puede ver como es clasificada muy bien, fallando solo 4 de 40 muestras que son clasificadas como pertenecientes a la clase seis.

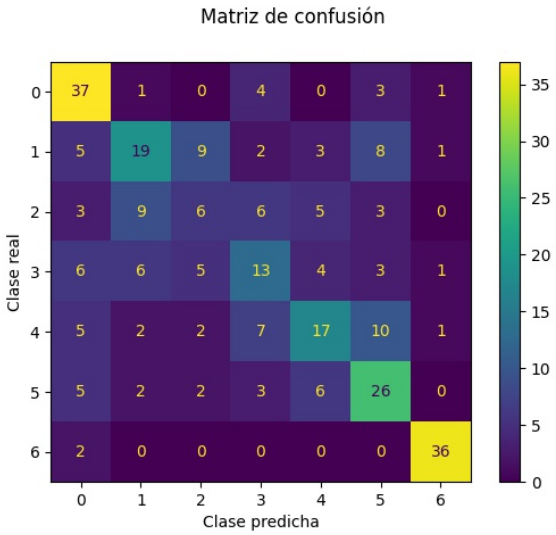


Figura 10.2: Matriz de confusión del modelo de regresión logística con preprocesado canny.

Al emplear el preprocesado HOG para este mismo modelo, se mejoran los resultados obtenidos. Fijándose en la curva de la clase 2, se puede ver cómo se aleja más de la diagonal, obteniéndose por tanto un mejor resultado.

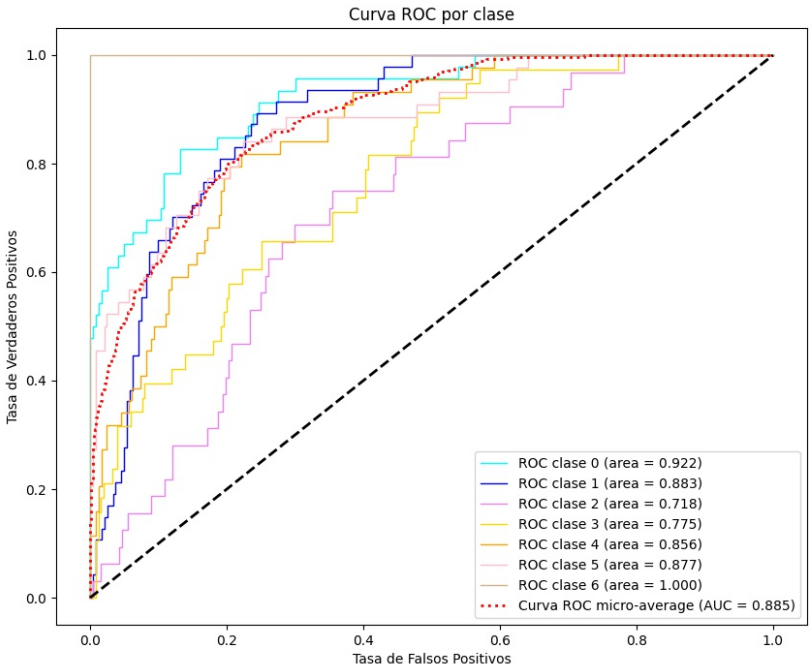


Figura 10.3: Curva ROC del modelo de regresión logística con preprocesado HOG.

Además, tanto en la matriz como en la curva se puede observar cómo todos los valores que son clasificados como pertenecientes a la clase 6, lo son. Solo queda un caso perteneciente a la clase seis que es clasificado de forma errónea como perteneciente a la clase 0.

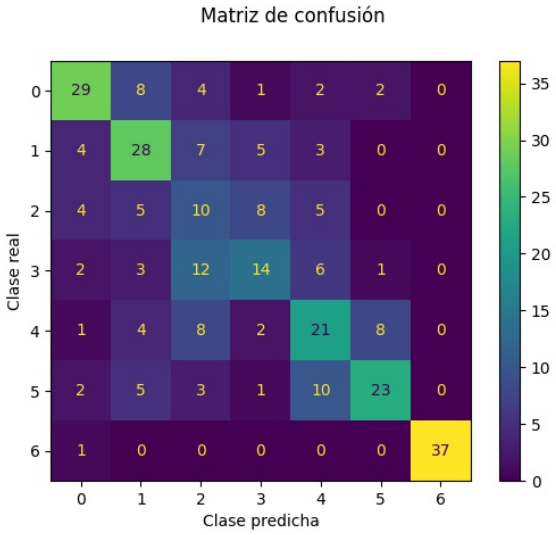


Figura 10.4: Matriz de confusión del modelo de regresión logística con preprocesado HOG.

10.1.2. Evaluación del discriminante lineal

Sabiendo que cuanto más cerca estén las curvas a la diagonal del gráfico, peor resultará el clasificador, se puede ver cómo el discriminante lineal con preprocesdado *canny* empeora por mucho los resultados obtenidos en la regresión logística.

El gesto seis, el del pulgar hacia abajo, sigue siendo el mejor clasificado. Esto se debe a que es el gesto más diferente de los seis, ya que es el único que en vez indicar un dedo hacia arriba en la imagen, se encuentra hacia abajo.

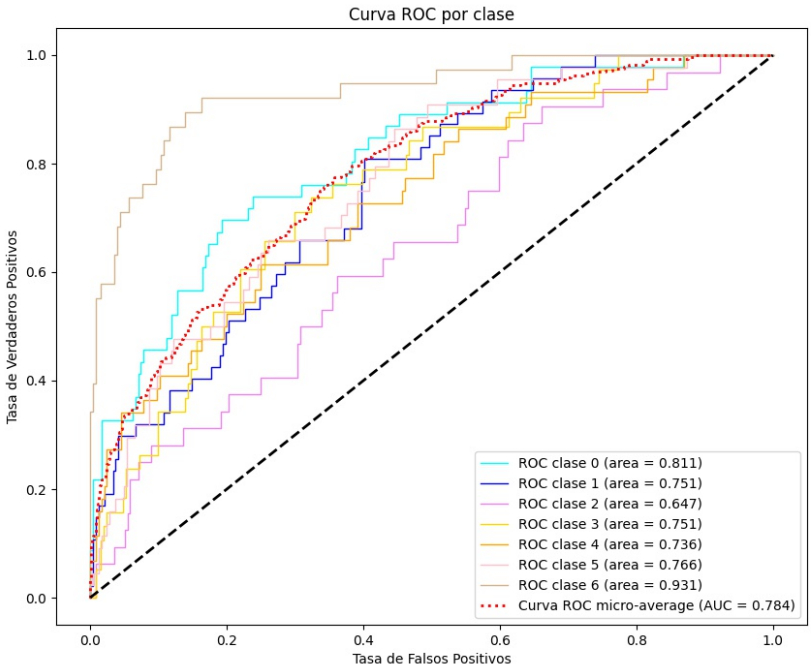


Figura 10.5: Curva ROC del modelo de discriminante lineal con preprocesado canny.

En la matriz de confusión 10.6, se puede ver que es incapaz de reconocer las clases más cercanas a la línea discontinua en la curva ROC.

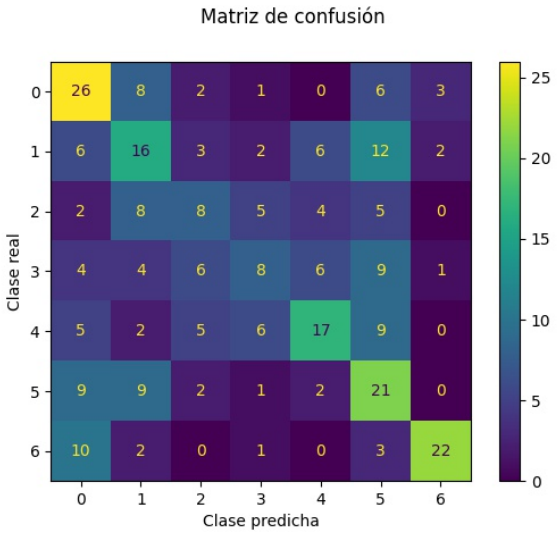


Figura 10.6: Matriz de confusión del modelo de discriminante lineal con preprocesado canny.

Con el preprocesado HOG, se puede ver como las clases que tienen menos aciertos son la 2 y la 3.

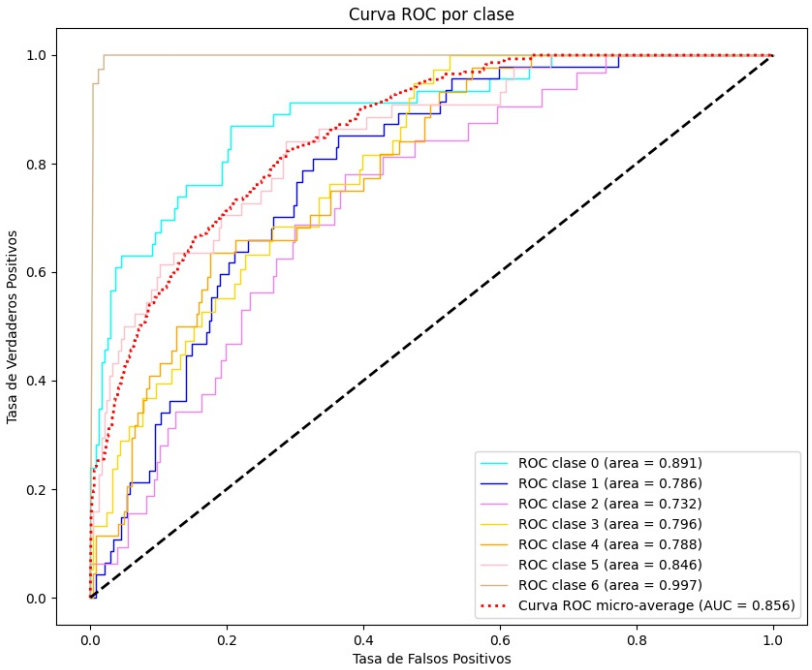


Figura 10.7: Curva ROC del modelo de discriminante lineal con preprocesado HOG.

En la matriz de confusión de la Figura 10.8 se puede ver que existe una tendencia a confundirse entre clase 1 y la 2. Esto puede deberse a que al ser el gesto del 1 y el 2, son consecutivos y, basta con que en una imagen no se detecte bien uno de los dedos para que el gesto 2 sea clasificado como el 1. También puede darse el caso de que, con el ruido de las imágenes, en casos en los que se está haciendo el gesto de un 1 con la mano, se detecte como que hay otro dedo levantado a mayores, dando lugar a que la clase 1 sea clasificada como la 2 por error.

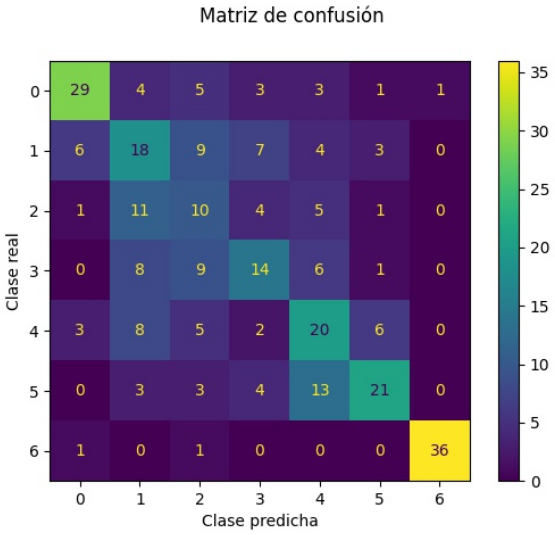


Figura 10.8: Matriz de confusión del modelo de discriminante lineal con preprocesado HOG.

10.1.3. Evaluación de K vecinos más cercanos

Los resultados de KNN son ligeramente mejores que los del discriminante lineal en el caso del preprocesado con el detector de bordes, sin embargo, sigue siendo incapaz de clasificar correctamente la clase 2.

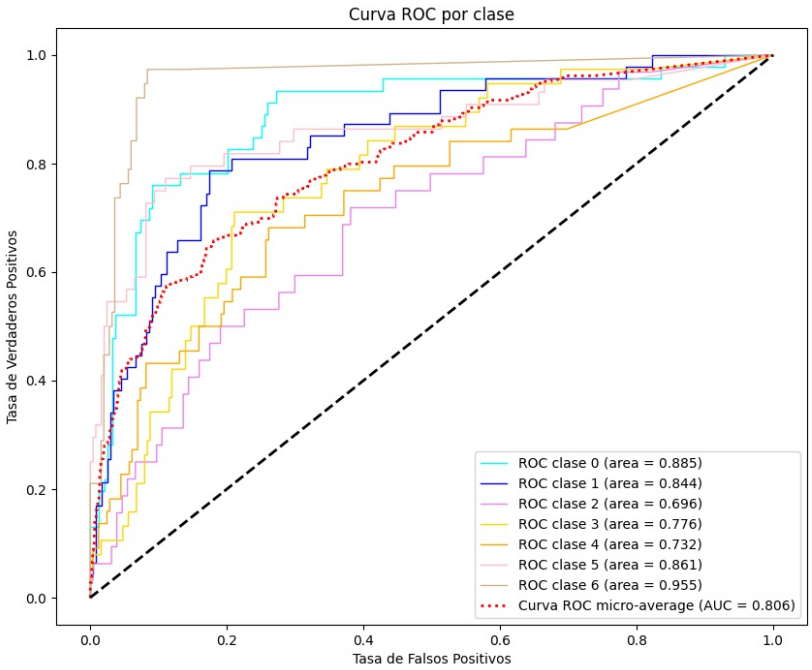


Figura 10.9: Curva ROC del modelo de K vecinos más cercanos con preprocesado canny.

Puede parecer que es capaz de identificar mejor la clase 1 que el resto de modelos, pero prestando atención a la matriz de confusión se puede ver que lo que sucede es que este modelo tiene tendencia a clasificar lo que recibe como perteneciente a la clase 1.

Esto suele producirse por escoger un K muy elevado, pues esto suaviza la región de decisión y tiende a clasificar como la clase mayoritaria.

Este modelo concentra la mayor parte de sus predicciones entre la clase 0 y la 1.

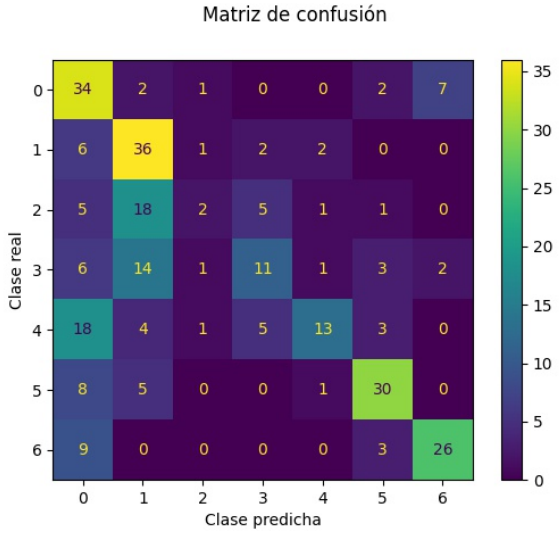


Figura 10.10: Matriz de confusión del modelo de K vecinos más cercanos con preprocesado canny.

En el modelo KNN con preprocesado HOG, se puede observar como se obtiene un resultado mejor que la mayoría de modelos.

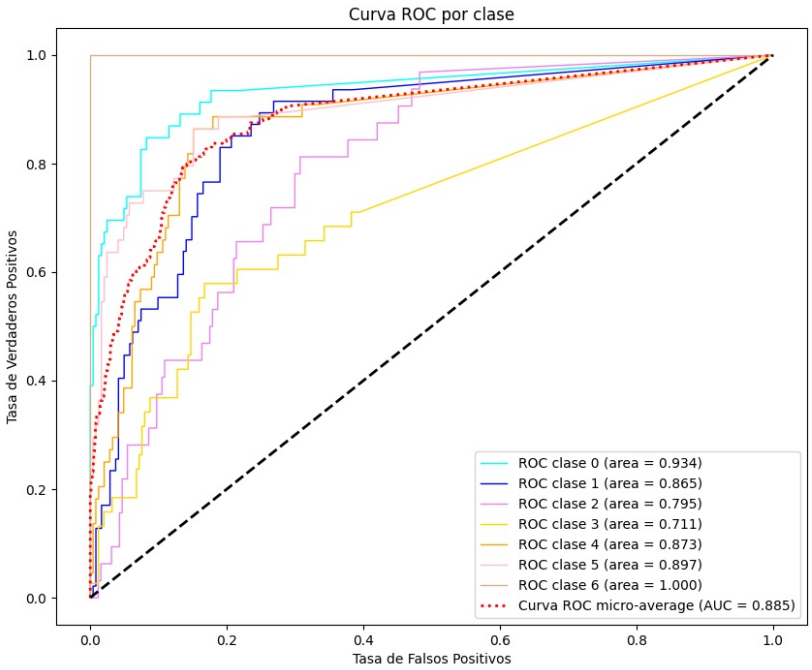


Figura 10.11: Curva ROC del modelo de K vecinos más cercanos con preprocesado HOG.

En la matriz de confusión de la Figura 10.12 se ve como el problema a la hora de clasificar suele estar entre la clase 2 y la 3. Aún así, en este modelo se clasifican muchos más casos de forma correcta que en los vistos hasta ahora.

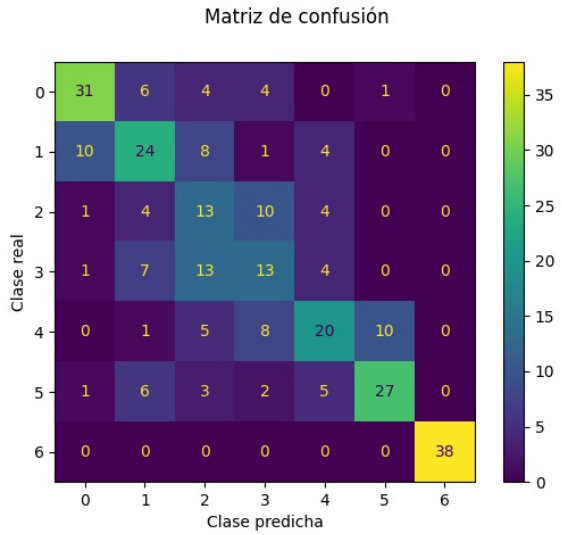


Figura 10.12: Matriz de confusión del modelo de K vecinos más cercanos con preprocesado HOG.

10.1.4. Evaluación de los árboles de decisión

En el caso de este modelo con preprocesado *canny* se obtiene un muy mal clasificador que es incapaz de predecir ninguna clases. Observando la curva ROC de la Figura 10.13, se puede ver como todas las clases se encuentran cercanas a la diagonal central, por lo que el modelo no es capaz de clasificar ninguna de ellas.

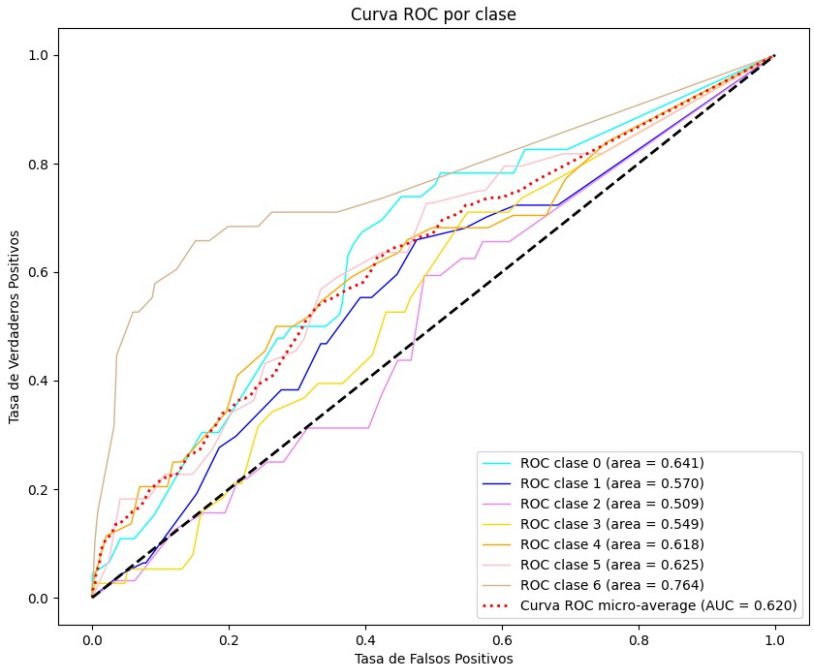


Figura 10.13: Curva ROC del modelo de árboles de decisión con preprocesado canny.

La clase que mejor predice es la clase 6 pero y aún así, los resultados obtenidos en esa clase no resultan extremadamente buenos.

Al realizar árboles de decisión con el preprocesado HOG, el resultado mejora, pero sigue siendo un mal clasificador que tan solo es capaz de clasificar bastante bien la clase seis.

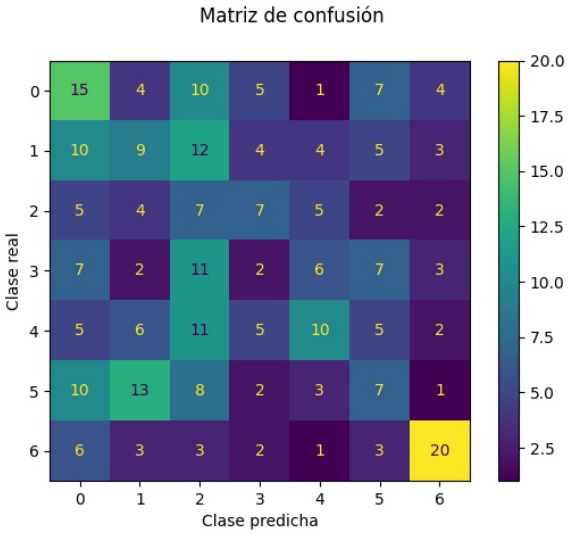


Figura 10.14: Matriz de confusión del modelo de árboles de decisión con preprocesado canny.

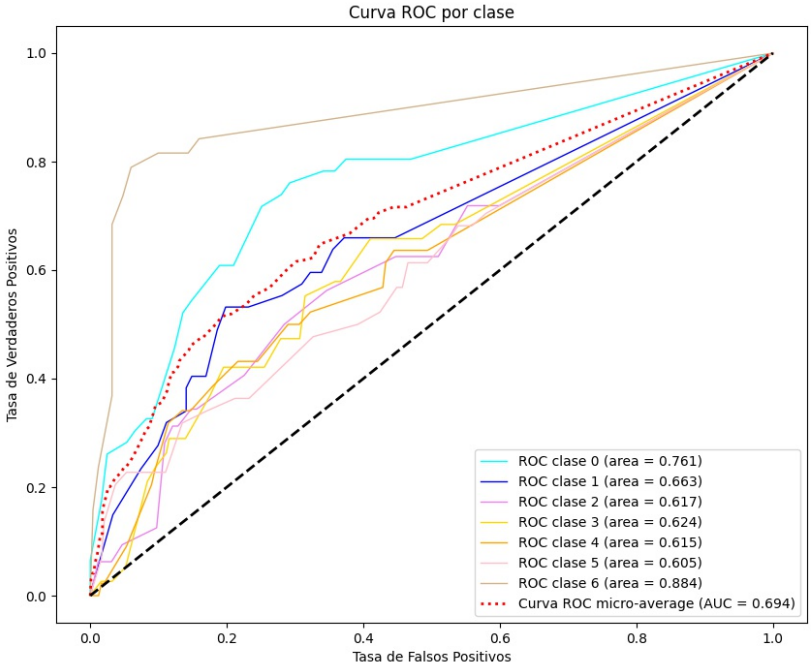


Figura 10.15: Curva ROC del modelo de árboles de decisión con preprocesado HOG.

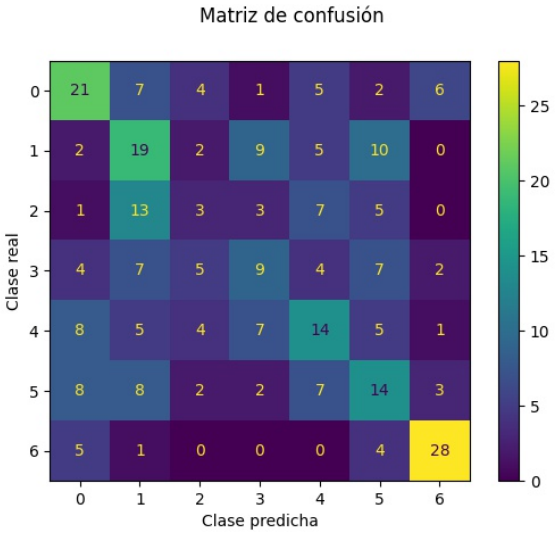


Figura 10.16: Matriz de confusión del modelo de árboles de decisión con preprocesado HOG.

10.1.5. Evaluación de los bosques aleatorios

En la curva de este clasificador cuando su preprocesado se ha realizado con el filtro *canny*, se produce cierta disparidad.

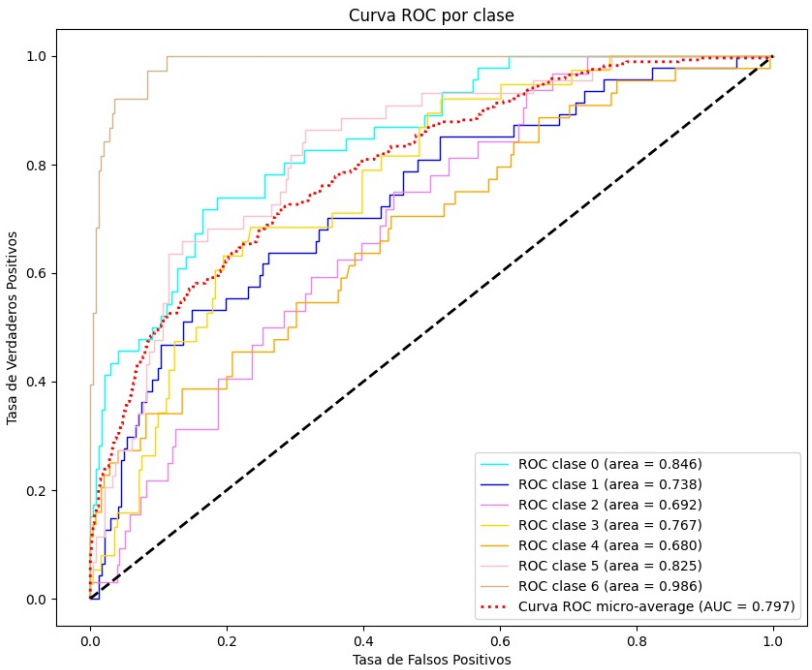


Figura 10.17: Curva ROC del modelo de bosques aleatorios con preprocesado canny.

En la matriz de confusión de la Figura 10.18 se puede ver como el modelo es completamente incapaz de predecir la clase 2 y como tiende a clasificar la mayoría de

imágenes como pertenecientes a la clase 0.

Las clases que mejor logra predecir son la 0, la 5 y la 6.

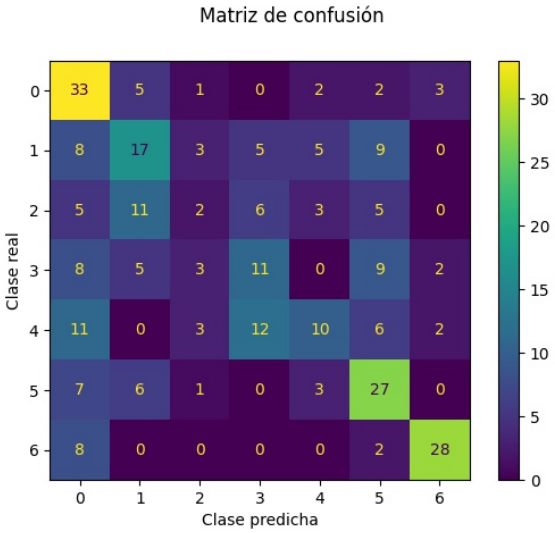


Figura 10.18: Matriz de confusión del modelo de bosques aleatorios con preprocesado canny.

En el caso del modelo de bosques aleatorios con el descriptor HOG, se mejora el resultado previo y se puede ver cómo aumenta el área bajo la curva.

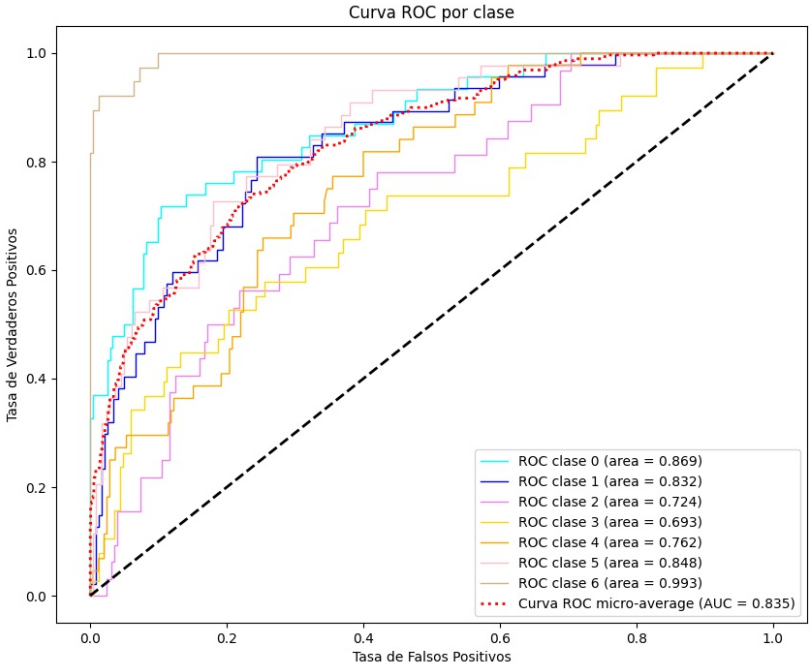


Figura 10.19: Curva ROC del modelo de bosques aleatorios con preprocesado HOG.

En este modelo se puede ver como las clases 0, 1, 5 y 6 obtienen muy buenos resultados en la clasificación y como la clase dos se mantiene como la peor clasificada.

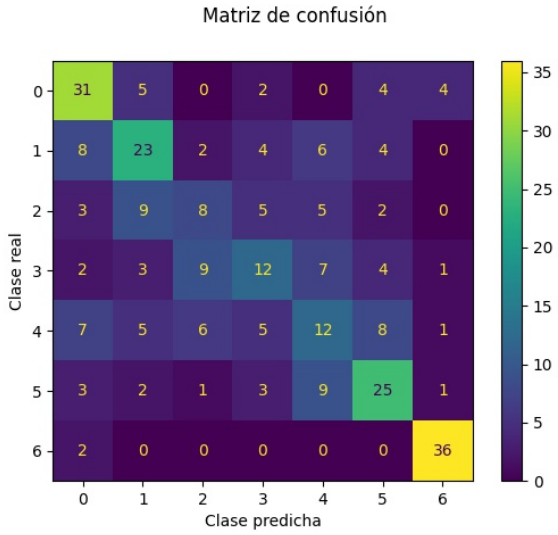


Figura 10.20: Matriz de confusión del modelo de bosques aleatorios con preprocesado HOG.

10.1.6. Evaluación de la máquina de vectores de soporte

Como resulta habitual en la mayoría de modelos, al modelo de SVM con preprocesado *canny*, le resulta imposible detectar la clase 2. Sin embargo, este modelo muestra un nivel de acierto aceptable en todas las otras clases, siendo las peores la clase 2 y la 3.

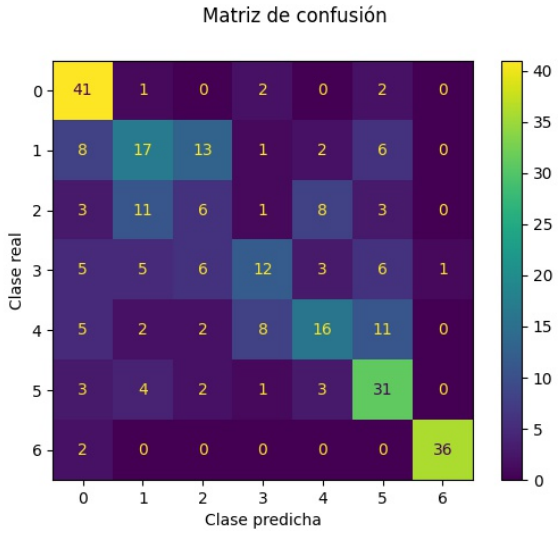


Figura 10.21: Matriz de confusión del modelo de máquina de vectores de soporte con preprocesado canny.

Al realizar el preprocesado HOG para este modelo, se observa en la matriz de

confusión 10.22 como se confunde entre las clases 2 y 3, pero el resto de predicciones son bastante buenas comparadas con el resto de modelos

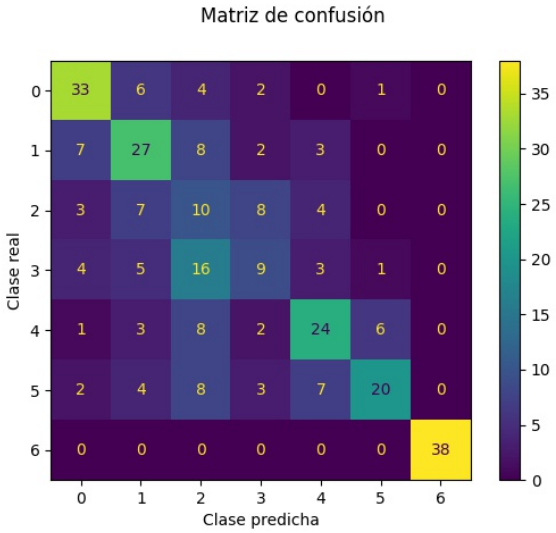


Figura 10.22: Matriz de confusión del modelo de máquina de vectores de soporte con preprocesado HOG.

10.1.7. Evaluación de las CNN

Al disponer de un dataset pequeño, se realiza *transfer learning* probando cuatro arquitecturas diferentes para la red CNN. Las cuatro arquitecturas probadas son:

- MobileNet
- ResNet50
- Inception V3
- EfficientNet

Las cuatro redes que se cargan han sido preentrenadas con el repositorio de ImageNet. Al haber sido entrenadas con una gran cantidad de imágenes, mejorará tanto en velocidad como en resultados lo que se podría lograr realizando una red convolucional con un dataset pequeño.

Para realizar *transfer learning*, se cargan las redes preentrendas sin la última capa y se conecta al final una capa de tipo densa con el número de clases que se pretenden clasificar (en este caso 7).

De las curvas ROC obtenidas, aunque son muy similares puesto que logran valores en torno a 0,90 de área bajo la curva para cada clase, cabe destacar que, en el caso

de EfficientNet, la clase que clasifica peor es la clase 3, a diferencia la mayoría de modelos en los que suele fallar más la clase 2.

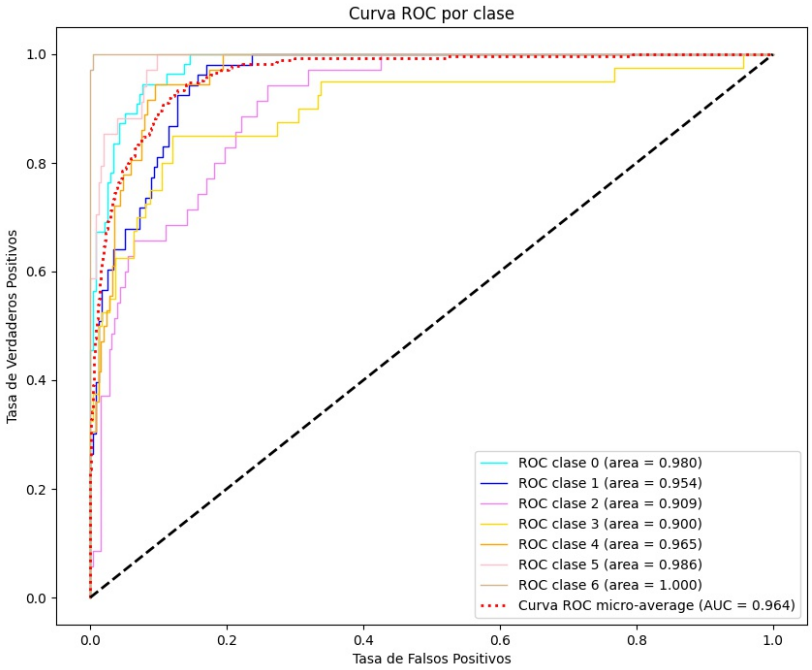


Figura 10.23: Curva ROC del modelo CNN con arquitectura MobileNet.

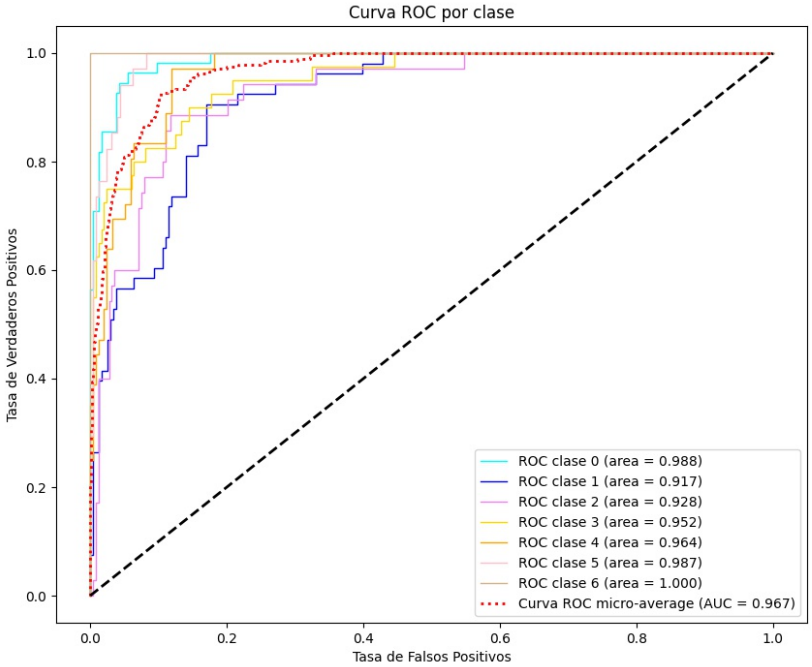


Figura 10.24: Curva ROC del modelo CNN con arquitectura ResNet.

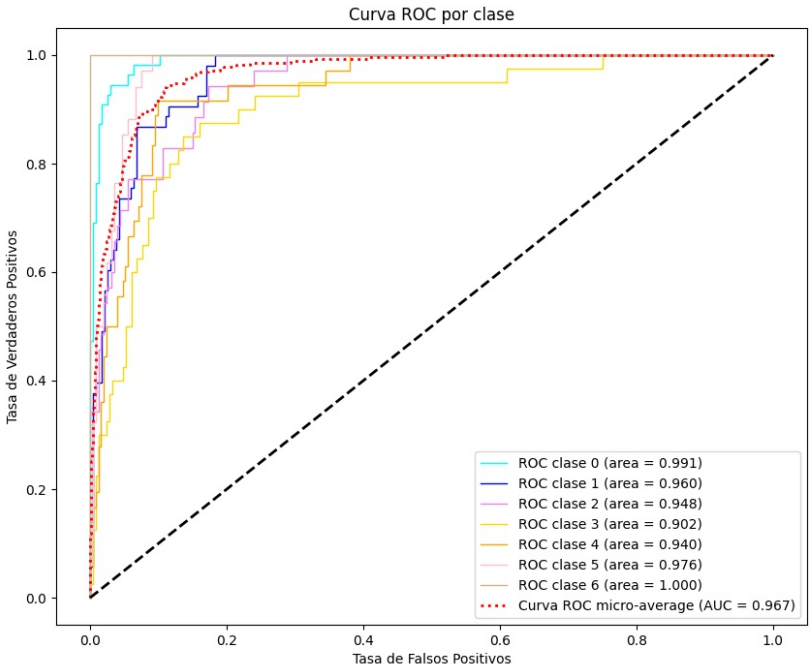


Figura 10.25: Curva ROC del modelo CNN con arquitectura Inception.

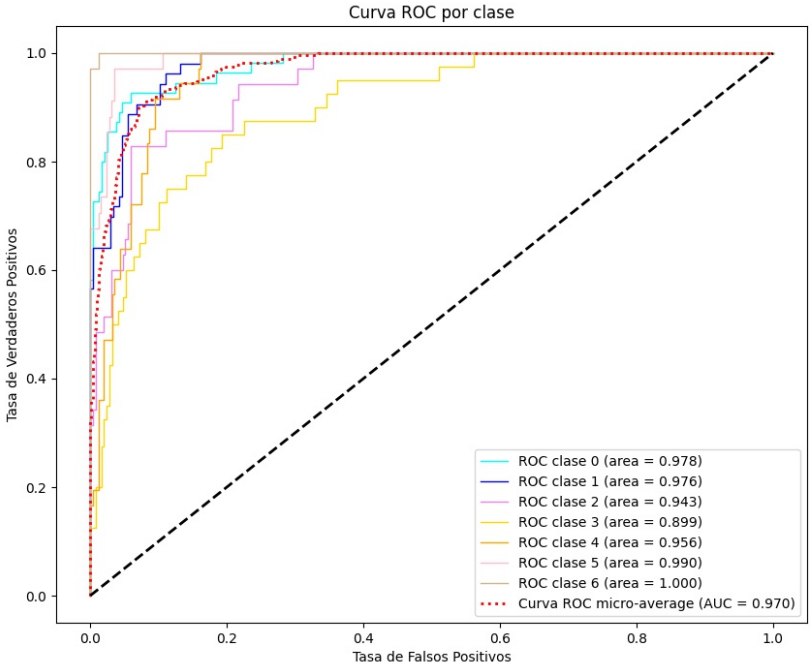


Figura 10.26: Curva ROC del modelo CNN con arquitectura EfficientNet.

En la Figura 10.27 se muestra el resultado de una predicción realizada en varias

imágenes con la red CNN con arquitectura de EfficientNet. Si hubiese alguna imagen mal clasificada, la etiqueta se mostraría en rojo, pero la red es suficientemente buena como para que rara vez clasifique mal una imagen del conjunto de validación..

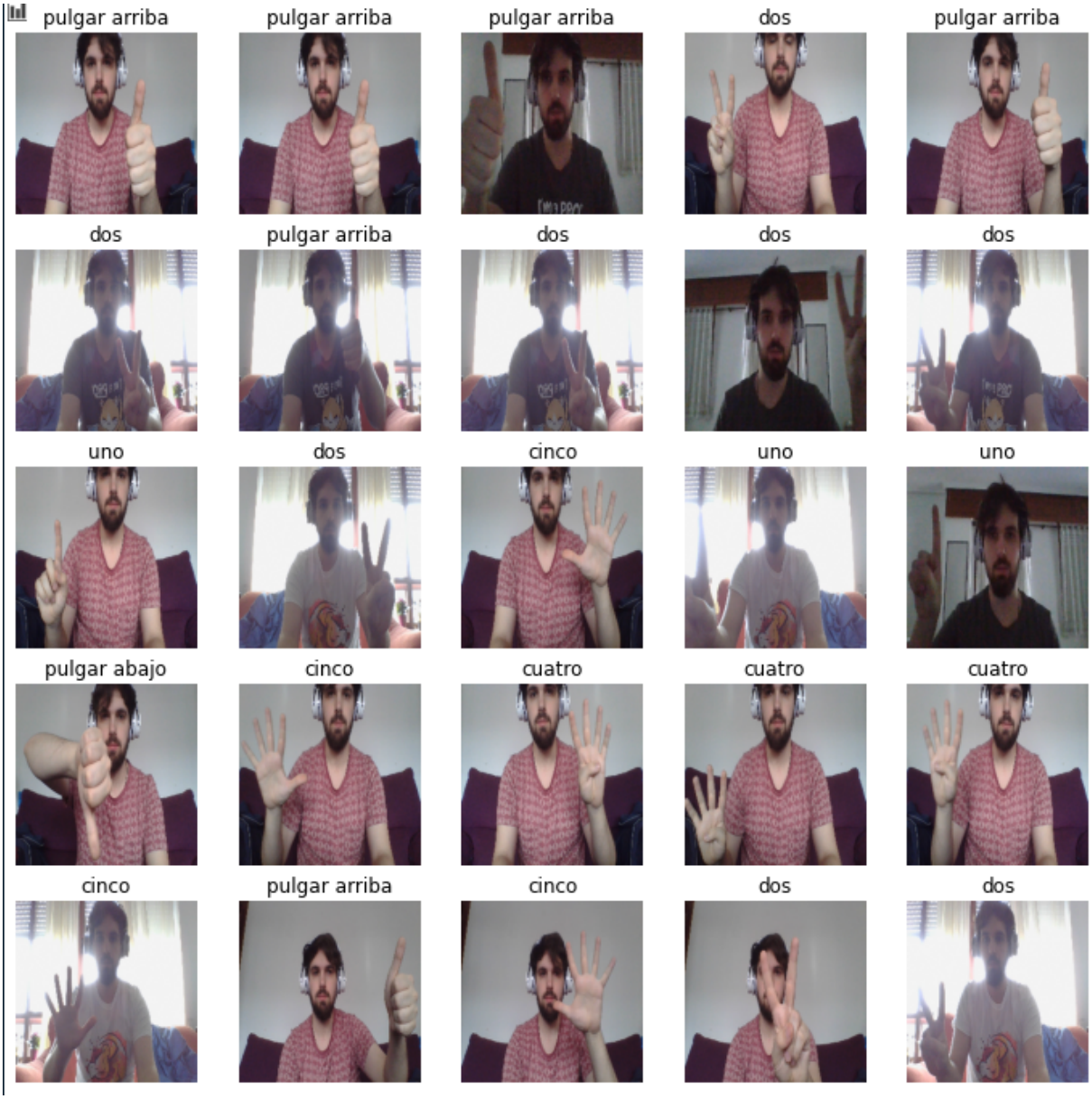


Figura 10.27: Prueba de predicción con imágenes.

De los modelos de las redes convolucionales, se puede mostrar el curva de exactitud (tasa de acierto global) y del error a lo largo del proceso de entrenamiento. En las Figuras 10.28-10.31, se muestran una gráfica de cada arquitectura empleada. Estas gráficas se sacan gracias al entrenamiento final que se hace de cada uno de estos modelos empleando un 75 % del conjunto de datos para el entrenamiento y un 25 % para la validación.

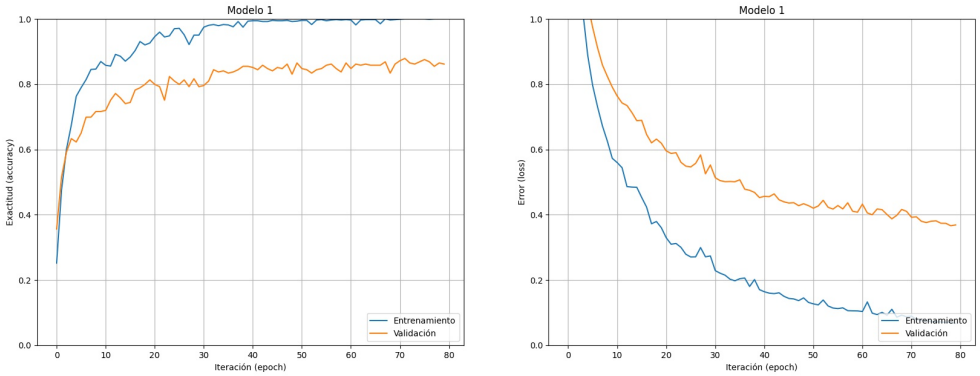


Figura 10.28: Curva de exactitud y error del modelo CNN con arquitectura MobileNet.

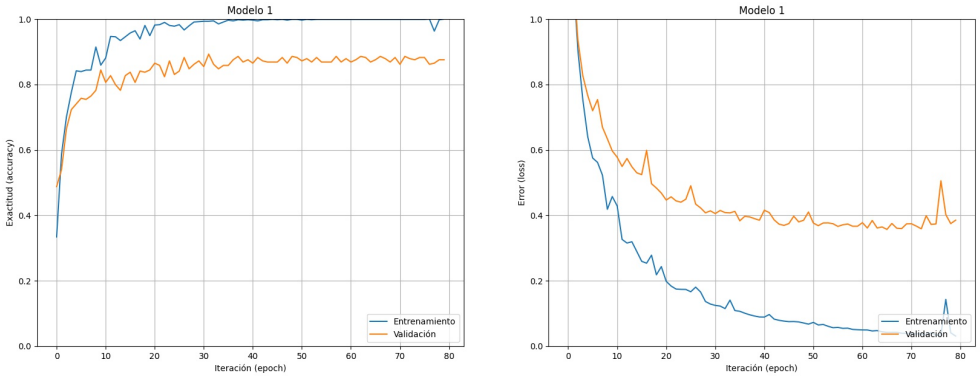


Figura 10.29: Curva de exactitud y error del modelo CNN con arquitectura ResNet.

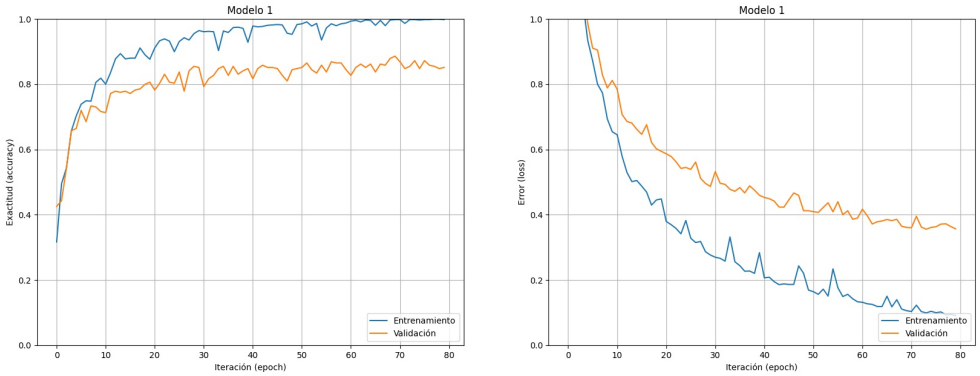


Figura 10.30: Curva de exactitud y error del modelo CNN con arquitectura Inception.

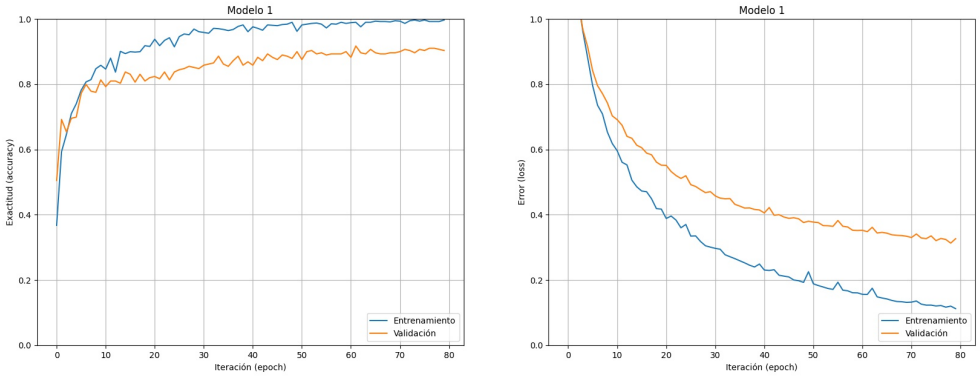


Figura 10.31: Curva de exactitud y error del modelo CNN con arquitectura EfficientNet.

Al estar cargando los modelos con *transfer learning*, existe un parámetro llamado *trainable* que puede ser puesto a verdadero o a falso. Todos los entrenamientos vistos hasta este momento se realizan con el parámetros a falso.

Si *Trainable* se pone a verdadero, se reentrenan todos los parámetros de la red. Esto hace que cada iteración del entrenamiento sea mucho más lenta al estar trabajando con muchos más parámetros, pero también se comprueba que permite obtener unos valores de exactitud mayores. El problema es que, la curva que muestra el proceso de la exactitud y el error, se vuelve completamente impredecible. En la Figura 10.32 se muestra la curva en la que, cuando parece que la exactitud está bajando y el sistema ha comenzado a sobreajustar, vuelve a subir repentinamente hasta alcanzar valores de exactitud de un 95 %.

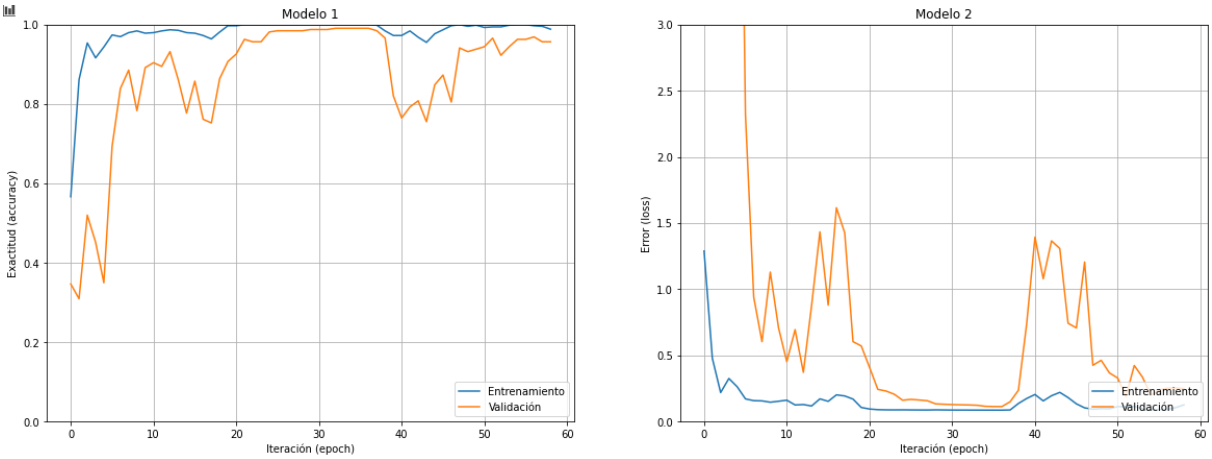


Figura 10.32: Curva de exactitud y error con todos los parámetros.

Para lidiar con este problema, se prueba a detener el entrenamiento una vez se alcance un valor de exactitud en el conjunto de evaluación superior al 95 %.

Para comprobar que el modelo resultante funciona bien, se prueba a dividir el dataset en tres grupos: entrenamiento, validación y test.

El conjunto de test se deja fuera del proceso de entrenamiento y, una vez la exactitud en el conjunto de validación sea superior al 95 %, se detiene el entrenamiento. Luego, se prueba el modelo en el conjunto de datos restante y se observa que se mantiene el porcentaje de acierto.

Este método finalmente no se selecciona porque el proceso de entrenamiento es demasiado impredecible y se logran encontrar otras redes convolucionales que alcanzan valores de acierto en la clasificación similares.

10.1.8. Comparativa de todos los modelos

En la hoja de cálculo 'métricas', se encuentran los valores de las métricas de evaluación obtenidas de cada modelo. Para facilitar la lectura del documento, cada métrica se ha redondeado a seis decimales. En el caso de querer consultar las cifras reales sin aproximar, se deberá acudir a la hoja de cálculo llamada 'métricas_medias'.

Para valorar que modelos funcionan mejor, primero hay que recordar que si las clases estuviesen muy desbalanceadas, sería importante si la métrica es micro o macro pero, al tratarse de clases balanceadas, la diferencia no resulta significativa. La medida que muestra más claramente que modelo funciona mejor es simplemente la exactitud en el conjunto de test.

En la Tabla 10.1 y en la Tabla 10.2, se muestran las medias de las métricas de evaluación obtenidas de todos los modelos en los que se aplicó el detector de bordes *canny* en el preprocesado. Como se puede ver, en este caso, los modelos que mejor funcionan son la regresión logística y la máquina de vectores de soporte, quedando ambos con una exactitud en el conjunto de validación de un 54 %.

En caso de tener que seleccionar un modelo de estos dos, se escogería el de regresión logística, ya que se entrena en un tiempo casi cinco veces menor y es capaz de clasificar nuevas imágenes mucho más rápido.

	ModelLR_canny	ModelLDA_canny	ModelKNN_canny
fit_time	3,728	5,474	0,985
score_time	0,008	0,009	0,670
test_precision_macro	0,528	0,426	0,401
test_recall_macro	0,536	0,418	0,319
test_precision_micro	0,546	0,425	0,341
test_recall_micro	0,546	0,425	0,341
test_f1_macro	0,527	0,415	0,286
test_accuracy	0,546	0,425	0,341

Tabla 10.1: Métricas de evaluación de modelos con preprocesado canny parte I.

	ModelDT_canny	ModelRF_canny	ModelSVM_canny
fit_time	0,382	1,104	15,011
score_time	0,011	0,013	0,599
test_precision_macro	0,254	0,480	0,535
test_recall_macro	0,251	0,481	0,534
test_precision_micro	0,253	0,493	0,548
test_recall_micro	0,254	0,493	0,548
test_f1_macro	0,245	0,467	0,526
test_accuracy	0,254	0,493	0,548

Tabla 10.2: Métricas de evaluación de modelos con preprocesado canny parte II.

En las siguientes dos tablas, Tabla 10.3 y Tabla 14.1, queda claro que el modelo que logra una mejor clasificación del conjunto de validación es el modelo KNN, logrando un 65,59 %.

	ModelLR_HOG	ModelLDA_HOG	ModelKNN_HOG
fit_time	13,630	4,327	0,213
score_time	0,008	0,007	0,695
test_precision_macro	0,582	0,560	0,660
test_recall_macro	0,563	0,543	0,646
test_precision_micro	0,571	0,547	0,656
test_recall_micro	0,571	0,547	0,656
test_f1_macro	0,567	0,545	0,646
test_accuracy	0,571	0,547	0,656

Tabla 10.3: Métricas de evaluación de modelos con preprocesado HOG parte I.

	ModelDT_HOG	ModelRF_HOG	ModelSVM_HOG
fit_time	0,351	4,177	7,444
score_time	0,006	0,009	0,380
test_precision_macro	0,322	0,560	0,603
test_recall_macro	0,330	0,555	0,582
test_precision_micro	0,335	0,567	0,591
test_recall_micro	0,335	0,567	0,591
test_f1_macro	0,320	0,546	0,585
test_accuracy	0,335	0,567	0,591

Tabla 10.4: Métricas de evaluación de modelos con preprocesado HOG parte II.

Fijándose en los valores de exactitud del conjunto de validación tanto en los modelos con el detector de bordes como en los modelos con el descriptor HOG, queda claro que el preprocesado con el descriptor HOG logra mejores resultados.

De las cuatro arquitecturas para las redes convolucionales probadas, fijándose en las métricas que se muestran en las tablas 10.5 y 10.6, destacan tanto ResNet como EfficientNet, quedando EfficientNet ligeramente por encima en sus métricas de evaluación. Lo que hace que definitivamente se seleccione EfficientNet por encima de las otras tres arquitecturas es que, además de ser la que mejores valores alcanza, es mucho más rápida tanto de entrenar como para clasificar nuevas imágenes si se compara con la red más cercana en cuanto a resultados.

	ModelCNN_mobilenet	ModelCNN_resnet
fit_time	41,042	113,079
score_time	0,605	1,160
test_precision_macro	0,859	0,914
test_recall_macro	0,861	0,906
test_precision_micro	0,864	0,912
test_recall_micro	0,864	0,912
test_f1_macro	0,857	0,906
test_accuracy	0,864	0,912

Tabla 10.5: Métricas de evaluación de modelos de redes CNN parte I.

	ModelCNN_inception	ModelCNN_efficientnet
fit_time	134,785	67,562
score_time	1,541	0,995
test_precision_macro	0,874	0,931
test_recall_macro	0,876	0,928
test_precision_micro	0,875	0,930
test_recall_micro	0,875	0,930
test_f1_macro	0,872	0,927
test_accuracy	0,875	0,930

Tabla 10.6: Métricas de evaluación de modelos de redes CNN parte II.

A continuación se realiza un diagrama de cajas con los valores de exactitud en el conjunto de test obtenidos de la validación cruzada de cada modelo.

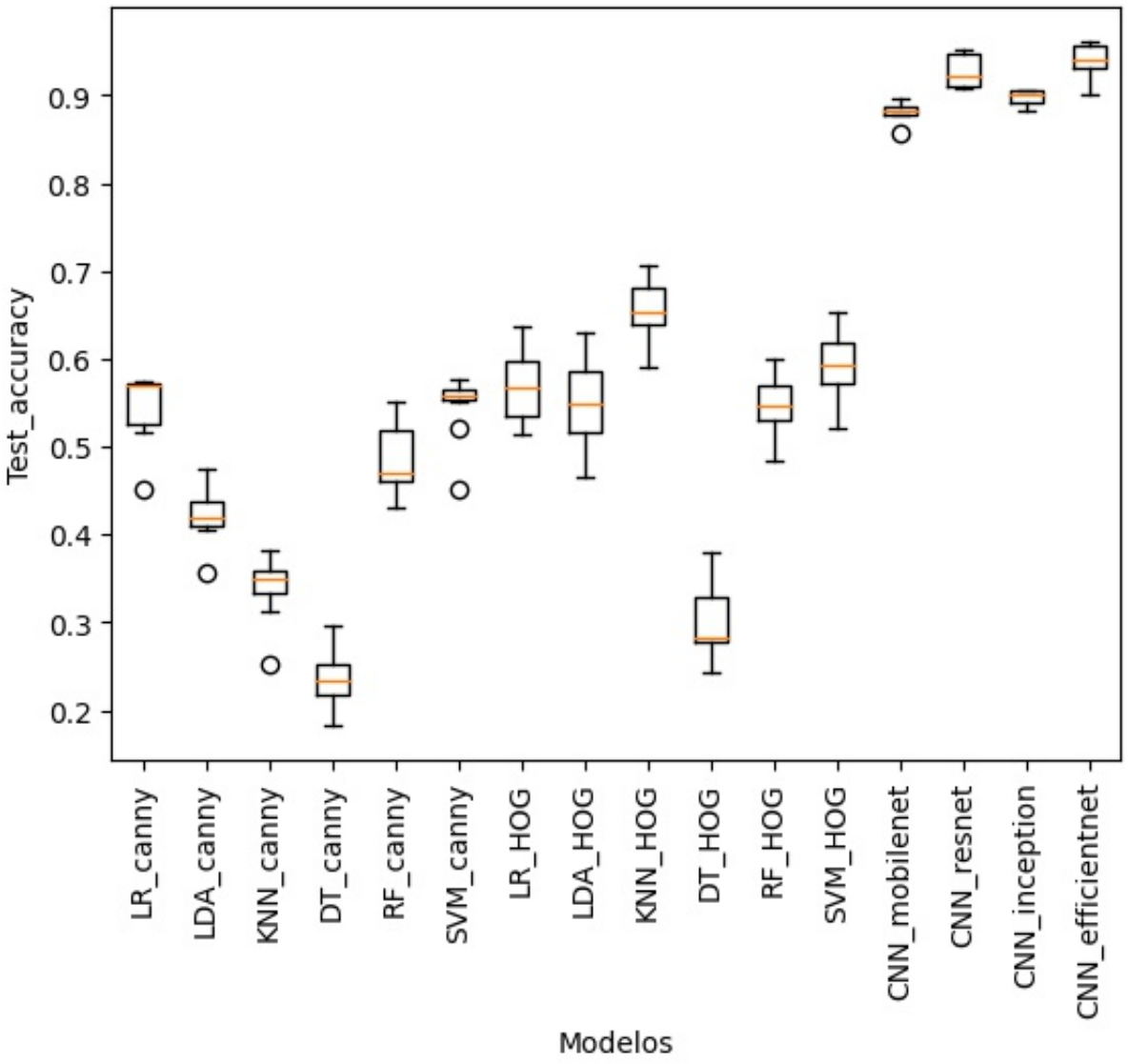


Figura 10.33: Diagrama de caja de los modelos con preprocesado con detector de bordes y de la red CNN.

Lo primero que se ve al realizar el diagrama de cajas con todos los modelos es que las redes CNN destacan por mucho respecto a los demás modelos.

Se puede ver cómo los modelos en los que se aplicó el descriptor HOG en su preprocesado alcanzan valores superiores a sus modelos equivalentes pero con preprocesado de detección de bordes.

De todos los modelos, los dos que destacan por una mayor exactitud en el conjunto de test son los modelos de EfficientNet y el de ResNet50. La diferencia entre ambos modelos no es significativa pero, al disponer de 8 variaciones del modelo EfficientNet, se decide probar las variaciones desde la B0 a la B4.

En las dos siguientes tablas muestran las métricas de evaluación de cinco variaciones de la arquitectura EfficientNet.

	Model_B0	**Model_B1**	**Model_B2**
fit_time	101,867	157,103	206,280
score_time	1,036	1,427	1,787
test_precision_macro	0,926	0,949	0,930
test_recall_macro	0,927	0,948	0,929
test_precision_micro	0,928	0,950	0,933
test_recall_micro	0,928	0,950	0,933
test_f1_macro	0,925	0,947	0,928
test_accuracy	0,928	0,950	0,933

Tabla 10.7: Métricas de evaluación de modelos de redes CNN con arquitectura EfficientNet parte I.

	Model_B3	**Model_B4**
fit_time	329,329	632,667
score_time	2,185	3,499
test_precision_macro	0,952	0,943
test_recall_macro	0,949	0,943
test_precision_micro	0,953	0,944
test_recall_micro	0,953	0,944
test_f1_macro	0,949	0,942
test_accuracy	0,953	0,944

Tabla 10.8: Métricas de evaluación de modelos de redes CNN con arquitectura EfficientNet parte II.

Haciendo contraste de hipótesis, se comprueba si los cuatro modelos de EfficientNet son iguales en función de su valor de *test_accuracy*. Para la realización de este análisis, se realizan los siguientes pasos:

1. Se define una hipótesis nula (H0).
2. Se selecciona un test estadístico para evaluar su validez.
3. Se elige el nivel de significación (α) del test.
4. Se calcula el valor p (la probabilidad de que la diferencia sea mayor de la observada siendo H0 verdad).
5. Comparar el valor p con α.
    - Si $p =< \alpha$ - Se rechaza H0.
    - Si $p > \alpha$ - Se acepta H0.

Se emplea el análisis de varianza Kruskal-Wallis y se propone como hipótesis que todos los modelos son iguales. Al probar este método con las cinco redes de EfficientNet, la hipótesis es aceptada y todos los modelos son considerados iguales.

En la siguiente Figura se muestran el diagrama de cajas de las cinco redes de EfficientNet en función de su valor de exactitud sobre el conjunto de validación.

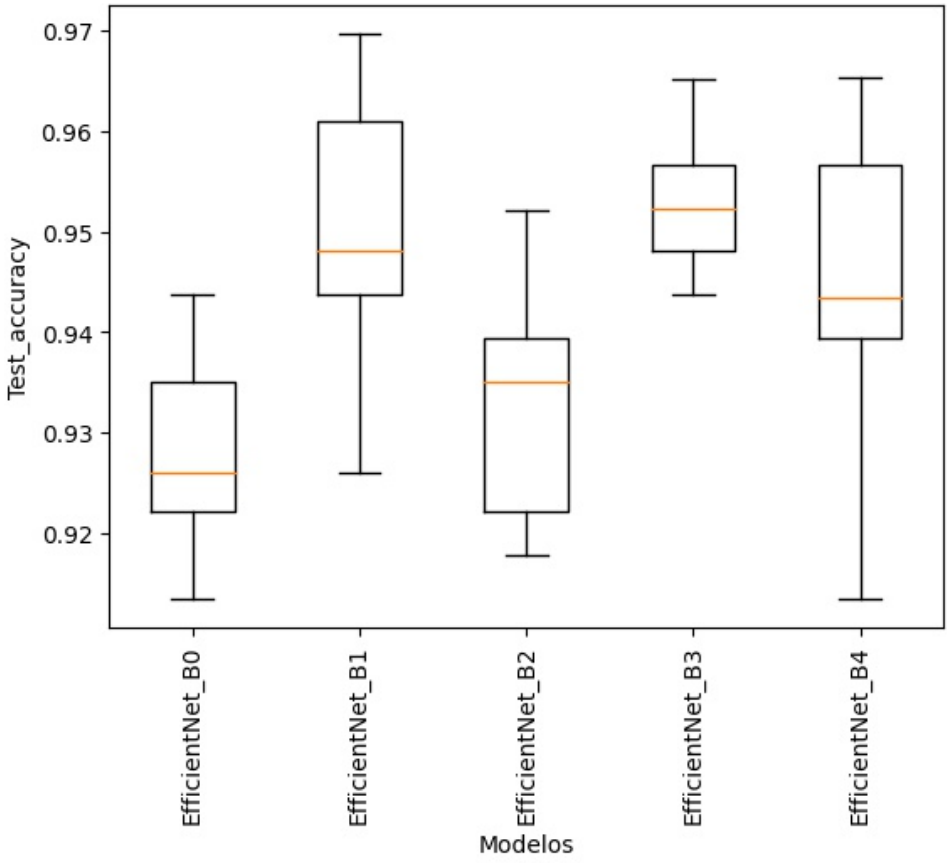


Figura 10.34: Diagrama de caja de los modelos de EfficientNet.

Para escoger entonces que modelo emplear, se miran las Tablas 10.7 y 10.8. Dos de los modelos tienen una exactitud en el conjunto de test del 95 %. Los modelos que logran esta cifra son el EfficientNet B1 y el EfficientNet B3. Al ser el modelo B1 mucho más rápido de entrenar y más rápido al etiquetar los datos, se selecciona el modelo EfficientNet B1 como el modelo que se empleará en la aplicación de prueba.

11. Aplicación

Para este trabajo, se diseña una aplicación que tendrá como objetivo clasificar los gestos de las fotografías que se capturen mediante la webcam de un ordenador. Se podrá probar así rápidamente el modelo en diferentes entornos.

Una vez se ha encontrado el modelo que mejor funciona para la clasificación de los gestos, vuelve a ser entrenado pero esta vez, al terminar el entrenamiento, se guardan sus pesos. Una vez guardados los pesos de la red en la carpeta modelo_elegido, ya se podrá volver a emplear esa red siempre que se desee.

La aplicación se diseña con el programa Tkinter. Consiste en un botón que una vez presionado, realiza una fotografía con la webcam del ordenador. Una vez realizada la foto, se mostrará junto a una imagen con el gesto correspondiente.

- Para cada uno de los números indicados con la mano se debe mostrar el número que se está indicando.
- Para el gesto del pulgar hacia arriba se debe mostrar un *ok*.
- Para el gesto del pulgar hacia abajo se debe mostrar un *not ok*.

En la Figura 11.1 se puede ver el estado inicial de la aplicación antes de realizar la primera fotografía.

Figura 11.1: Aplicación antes de realizar la primera fotografía.

En las Figuras 11.2, 11.3 y 11.4, se muestra la aplicación una vez ha realizado una fotografía. A la izquierda, se puede ver la fotografía que ha realizado mientras que, a la derecha, se muestra la predicción.

Figura 11.2: Aplicación prediciendo el gesto 5.

Figura 11.3: Aplicación prediciendo el gesto del pulgar hacia abajo.

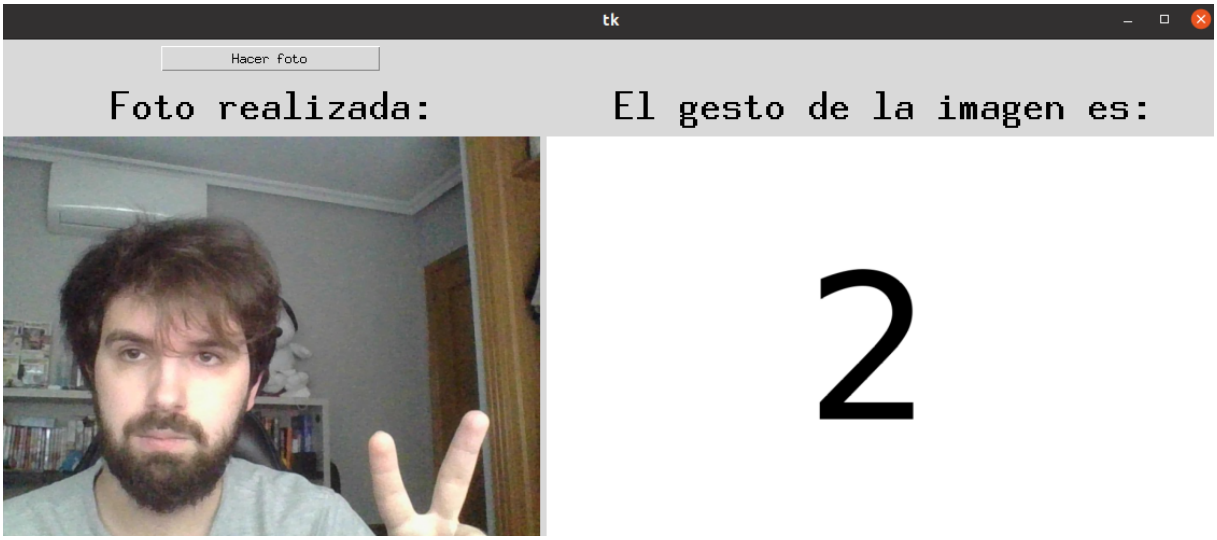


Figura 11.4: Aplicación prediciendo el gesto 2.

Al realizar las predicciones tiene un acierto mucho menor de lo que se muestra al realizar validación cruzada en el modelo. La causa de esto es que el dataset no es lo suficientemente amplio como para adaptarse a imágenes en las que se está variando el entorno, la vestimenta e incluso el ángulo de la cámara (ya que la mesa desde donde se utiliza el ordendor varía).

La solución sería ampliar el dataset y añadir más variedad, tanto de ángulos, como de iluminación, de entorno e incluso de usuarios.

12. Conclusiones y trabajos futuros

De entre todos los modelos empleados en este trabajo, las redes convolucionales obtienen resultados mucho mejores al resto de modelos.

Realizar *transfer learning*, mejora por mucho el rendimiento de la red si el dataset empleado es pequeño y, además, acelera en gran medida el proceso de entrenamiento.

Al variarse el entorno y diferir de los casos del dataset, la exactitud disminuye notablemente. Para realizar un caso general de clasificación de gestos, sería necesario un dataset mucho más grande y muy variado.

Como trabajo futuro que podría ampliar este proyecto, estaría la realización de un dataset que satisfaga las condiciones previamente mencionadas. Además, en caso de disponer de un dataset lo suficientemente grande, se podrían ampliar el número de gestos y tratar de identificar algunos gestos de lengua de signos. Evidentemente, no sería tan sencillo como clasificar los gestos el poder reproducir el lenguaje de signos pero, yendo más allá y combinando la clasificación de todos los gestos en tiempo real con la programación del lenguaje natural, se podría intentar crear un traductor válido.

13. Referencias web

[1] *Tema 1.- Introducción a la Visión Artificial*, Máster en Ingeniería Informática de la Universidad de Córdoba. [Fecha de consulta: 14 de marzo]. Disponible en: http://www.uco.es/users/ma1fegan/2015-2016/vision/Temas/ruido.pdf

[2] *Image Filtering*. OpenCV documentation. [Fecha de consulta: 12 de abril]. Disponible en: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#image-filtering

[3] *Imagen 3.1 .* de Marko Meza, Wikipedia, the free encyclopedia. [Fecha de consulta: 10 de mayo]. Disponible en: https://es.m.wikipedia.org/wiki/Archivo:Noise_salt_and_pepper.png

[4] *OpenCV: Canny edge detector*, OpenCV documentation. [Fecha de consulta: 5 de abril]. Disponible en: https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html

[5] *Imagen 3.2 del usuario DrWalencia de Wikipedia*, Wikipedia, the free encyclopedia. [Fecha de consulta: 10 de mayo]. Disponible en: https://es.wikipedia.org/wiki/Archivo:512x512-Gaussian-Noise.jpg

[6] *QUÉ ES LA VISIÓN ARTIFICIAL*, COGNEX. [Fecha de consulta: 18 de marzo]. Disponible en: https://www.cognex.com/es-es/what-is/machine-vision/what-is-machine-vision

[7] *2.6. Manipulación y procesamiento de imágenes usando Numpy y Scipy*, GITHUB. [Fecha de consulta: 25 de marzo]. Disponible en: https://claudiovz.github.io/scipy-lecture-notes-ES/advanced/image_processing/index.html

[8] *Sklearn.linear_model.LogisticRegression*, Sklearn [Fecha de consulta: 18 de marzo]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[9] *Sklearn.discriminant_analysis.LinearDiscriminantAnalysis*, Scikit learn [Fecha de consulta: 18 de marzo]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

[10] *Sklearn.neighbors.KNeighborsClassifiers*, Scikit learn [Fecha de consulta: 18 de marzo]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[11] *Sklearn.tree.DecisionTreeClassifier*, Scikit learn [Fecha de consulta: 18 de marzo]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[12] *Sklearn.ensemble.RandomForestClassifier*, Scikit learn [Fecha de consulta: 18 de marzo]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[13] *Sklearn.svm.SVC*, Scikit learn [Fecha de consulta: 18 de marzo]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[14] *Imagen 6.12*, Google Cloud [Fecha de consulta: 18 de marzo]. Disponible en: https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=es-419

[15] *Imagen 6.7 del usuario Cecbur de Wikipedia*, Wikipedia, the free encyclopedia. [Fecha de consulta: 24 de mayo]. Disponible en: https://commons.wikimedia.org/wiki/File:3_filters_in_a_Convolutional_Neural_Network.gif

[16] *Imagen 6.9 del usuario Aphex34 de Wikipedia*, Wikipedia, the free encyclopedia. [Fecha de consulta: 24 de mayo]. Disponible en: https://commons.wikimedia.org/wiki/File:Max_pooling.png

[17] *Imagen 6.6 del usuario Machine Learner de Wikipedia*, Wikipedia, the free encyclopedia. [Fecha de consulta: 24 de mayo]. Disponible en: https://commons.wikimedia.org/wiki/File:Nonlinear_SVM_example_illustration.svg

Bibliografía

[1] Alejandro Paz López y Alma María Casdelo. “Apuntes de la asignatura de Visión Artificial”. En: *Máster en Informática Industrial y Robótica en la* Univesidad de Coruña (2021).

[2] Kaiming He y col. “Deep Residual Learning for Image Recognition”. En: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[3] Cheng Li. “Fisher Linear Discriminant Analysis”. En: 2014.

[4] Óscar Fontenla Romero. “Apuntes de la asignatura de Aprendizaje Automático”. En: *Máster en Informática Industrial y Robótica en la* Univesidad de Coruña (2021).

[5] Mark Sandler y col. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. En: *arXiv e-prints*, arXiv:1801.04381 (ene. de 2018), arXiv:1801.04381. arXiv: 1801.04381 [cs.CV].

[6] Christian Szegedy y col. “Rethinking the Inception Architecture for Computer Vision”. En: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[7] Mingxing Tan y Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. En: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

14. Presupuesto

14.1. Mano de obra

Mano de obra			
Tipo de mano de obra	Número de horas	€/hora	Precio total
Graduado en Ingeniería Electrónica Industrial y Automática	350	40 €	14.000 €

Tabla 14.1: Coste la mano de obra empleada en el proyecto

15. Documentación de partida

15.1. Propuesta inicial de asignación del TFM

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE MÁSTER

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *Palacios Fragoso,Manuel*
APELIDOS E NOME:

Fecha de Solicitud: FEB2021
Fecha de Solicitude:

Alumno de esta escuela en la titulación de Máster en Informática Industrial y Robótica, se le comunica que la Comisión de Seguimiento del Máster ha decidido asignarle el siguiente Trabajo Fin de Máster:

O alumno de esta escola na titulación de Mestrado en Informática Industrial e Robótica, comunícaselle que a Comisión de Seguemento do Mestrado decideu asignarlle o seguinte Traballo Fin de Mestrado:

Título T.F.M:Implementación de un sistema de reconocimiento de gestos mediante técnicas inteligentes

Número TFM: *4538M01A002*

TUTOR:(Titor) *Jove Pérez,Esteban*

COTUTOR/CODIRECTOR: *Francisco Zayas Gato*

La descripción y objetivos del TFM son los que figuran en el reverso de este documento*:*

A descrición e obxectivos do TFM son os que figuran no reverso deste documento.

Ferrol a Jueves, 1 de Julio del 2021

Retirei o meu Traballo Fin de Máster o dia _____ de _____ do ano _____

Fdo: Palacios Fragoso,Manuel

DESCRIPCIÓN Y OBJETIVO:OBJETO:
En este Trabajo Final de Máster se realizará el estudio y la implementación de un sistema de reconocimiento de gestos mediante técnicas de machine learning y de visión artificial. Los gestos identificados serán realizados con una mano. Inicialmente, se hará un estudio del tipo de software apropiado para aplicar las librerías de tratamiento de imágenes y las técnicas de machine learning. Posteriormente, tratará de implementarse el reconocimiento de los diferentes gestos realizados. Se estudiará la posibilidad de realizar una aplicación capaz de analizar empleando los gestos detectados.

ALCANCE:

Análisis del software adecuado para realizar el reconocimiento de los diferentes gestos realizados con la mano.
Aplicación de librerías de tratamiento de imágenes para facilitar la detección de los gestos.
Aplicación de las técnicas apropiadas de machine learning para reconocer los gestos.
Estudio de la posibilidad de realización de una pequeña aplicación con el sistema de reconocimiento de gestos.

16. Código del programa

El código se sube en un archivo ZIP a GitHub para facilitar su prueba.

Link: https://github.com/nunuel/TFM_Manuel_palacios_fragoso
Contraseña archivo ZIP: tfm_2021_palacios_fragoso_manuel

16.1. Script main

```
'''Importación de librerías'''
import os
import re
import cv2
import matplotlib.pyplot as plt
import numpy as np
from numpy.lib.function_base import rot90

import Metodos_evaluacion
import modelos_aprendizaje
import efficientnet_completo
import Preprocesado


import pandas as pd
from pandas import ExcelWriter

def main():

    '''Supervisado'''

    scoresCNN_mobilenet, scoresCNN_resnet, scoresCNN_inception,
    scoresCNN_efficientnet = modelos_aprendizaje.CNN_FINAL(224)

    '''Supervisado con imágenes HOG'''

    scoresLR_HOG = modelos_aprendizaje.RL_FINAL(64,'Escritorio/
    trabajo_aprendizaje/HOG',canny_hog=1)

    scoresLDA_HOG = modelos_aprendizaje.LDA_FINAL(64,'Escritorio/
    trabajo_aprendizaje/HOG',canny_hog=1)

    scoresKNN_HOG = modelos_aprendizaje.KNN_FINAL(64,'Escritorio/
    trabajo_aprendizaje/HOG',canny_hog=1)
```

```
31
    scoresDT_HOG = modelos_aprendizaje.DT_FINAL(64,'Escritorio/
    trabajo_aprendizaje/HOG',canny_hog=1)
33
    scoresRF_HOG = modelos_aprendizaje.RF_FINAL(64,'Escritorio/
    trabajo_aprendizaje/HOG',canny_hog=1)
35
    scoresSVM_HOG = modelos_aprendizaje.SVM_FINAL(64,'Escritorio/
    trabajo_aprendizaje/HOG',canny_hog=1)
37

39   '''Supervisado con imagenes Canny'''

41   scoresLR_canny = modelos_aprendizaje.RL_FINAL(64,'Escritorio/
    trabajo_aprendizaje/canny',canny_hog=0)

43   scoresLDA_canny = modelos_aprendizaje.LDA_FINAL(64,'Escritorio/
    trabajo_aprendizaje/canny',canny_hog=0)

45   scoresKNN_canny = modelos_aprendizaje.KNN_FINAL(64,'Escritorio/
    trabajo_aprendizaje/canny',canny_hog=0)

47   scoresDT_canny = modelos_aprendizaje.DT_FINAL(64,'Escritorio/
    trabajo_aprendizaje/canny',canny_hog=0)

49   scoresRF_canny = modelos_aprendizaje.RF_FINAL(64,'Escritorio/
    trabajo_aprendizaje/canny',canny_hog=0)

51   scoresSVM_canny = modelos_aprendizaje.SVM_FINAL(64,'Escritorio/
    trabajo_aprendizaje/canny',canny_hog=0)

53

55   test_sc=[scoresLR_canny['test_accuracy'],scoresLDA_canny['test_accuracy
    '],scoresKNN_canny['test_accuracy'],scoresDT_canny['test_accuracy'],
    scoresRF_canny['test_accuracy'],scoresSVM_canny['test_accuracy'],
    scoresLR_HOG['test_accuracy'],scoresLDA_HOG['test_accuracy'],
    scoresKNN_HOG['test_accuracy'],scoresDT_HOG['test_accuracy'],
    scoresRF_HOG['test_accuracy'],scoresSVM_HOG['test_accuracy'],
    scoresCNN_mobilenet['test_accuracy'],scoresCNN_resnet['test_accuracy'],
    scoresCNN_inception['test_accuracy'],scoresCNN_efficientnet['
    test_accuracy']]
     #test_sc=[scoresLR_canny['test_accuracy'],scoresLDA_canny['
    test_accuracy'],scoresKNN_canny['test_accuracy'],scoresDT_canny['
    test_accuracy'],scoresRF_canny['test_accuracy'],scoresSVM_canny['
    test_accuracy'],scoresLR_HOG['test_accuracy'],scoresLDA_HOG['
```

```
    test_accuracy'],scoresKNN_HOG['test_accuracy'],scoresDT_HOG['
    test_accuracy'],scoresRF_HOG['test_accuracy'],scoresSVM_HOG['
    test_accuracy'],scoresCNN['test_accuracy']]
57  lab = ['LR_canny','LDA_canny','KNN_canny','DT_canny','RF_canny','
    SVM_canny','LR_HOG','LDA_HOG','KNN_HOG','DT_HOG','RF_HOG','SVM_HOG','
    CNN_mobilenet','CNN_resnet','CNN_inception','CNN_efficientnet']
    fig, ax = plt.subplots()
59
    ax.boxplot(test_sc,labels=lab)
61  ax.set_ylabel('Test_accuracy')
    ax.set_xlabel('Modelos')
63  plt.xticks(rotation = 90)

65  plt.savefig('/home/nunuel/Escritorio/trabajo_aprendizaje/boxplot/
    boxplot.jpg', bbox_inches='tight')


67
    df = pd.DataFrame({'Modelo': ['fit_time', 'score_time', '
    test_precision_macro', 'test_recall_macro', 'test_precision_micro','
    test_recall_micro','test_f1_macro','test_accuracy'],
69                  'ModelLR_canny': [scoresLR_canny['fit_time'],
    scoresLR_canny['score_time'], scoresLR_canny['test_precision_macro'],
    scoresLR_canny['test_recall_macro'],scoresLR_canny['test_precision_micro
    '],scoresLR_canny['test_recall_micro'],scoresLR_canny['test_f1_macro'],
    scoresLR_canny['test_accuracy']],
                    'ModelLDA_canny': [scoresLDA_canny['fit_time'],
    scoresLDA_canny['score_time'], scoresLDA_canny['test_precision_macro'],
    scoresLDA_canny['test_recall_macro'],scoresLDA_canny['
    test_precision_micro'],scoresLDA_canny['test_recall_micro'],
    scoresLDA_canny['test_f1_macro'],scoresLDA_canny['test_accuracy']],
71                  'ModelKNN_canny': [scoresKNN_canny['fit_time'],
    scoresKNN_canny['score_time'], scoresKNN_canny['test_precision_macro'],
    scoresKNN_canny['test_recall_macro'],scoresKNN_canny['
    test_precision_micro'],scoresKNN_canny['test_recall_micro'],
    scoresKNN_canny['test_f1_macro'],scoresKNN_canny['test_accuracy']],
                    'ModelDT_canny': [scoresDT_canny['fit_time'],
    scoresDT_canny['score_time'], scoresDT_canny['test_precision_macro'],
    scoresDT_canny['test_recall_macro'],scoresDT_canny['test_precision_micro
    '],scoresDT_canny['test_recall_micro'],scoresDT_canny['test_f1_macro'],
    scoresDT_canny['test_accuracy']],
73                  'ModelRF_canny': [scoresRF_canny['fit_time'],
    scoresRF_canny['score_time'], scoresRF_canny['test_precision_macro'],
    scoresRF_canny['test_recall_macro'],scoresRF_canny['test_precision_micro
    '],scoresRF_canny['test_recall_micro'],scoresRF_canny['test_f1_macro'],
    scoresRF_canny['test_accuracy']],
```

```
                    'ModelSVM_canny': [scoresSVM_canny['fit_time'],
    scoresSVM_canny['score_time'], scoresSVM_canny['test_precision_macro'],
    scoresSVM_canny['test_recall_macro'],scoresSVM_canny['
    test_precision_micro'],scoresSVM_canny['test_recall_micro'],
    scoresSVM_canny['test_f1_macro'],scoresSVM_canny['test_accuracy']],
75                  'ModelLR_HOG': [scoresLR_HOG['fit_time'], scoresLR_HOG[
    'score_time'], scoresLR_HOG['test_precision_macro'], scoresLR_HOG['
    test_recall_macro'],scoresLR_HOG['test_precision_micro'],scoresLR_HOG['
    test_recall_micro'],scoresLR_HOG['test_f1_macro'],scoresLR_HOG['
    test_accuracy']],
                    'ModelLDA_HOG': [scoresLDA_HOG['fit_time'],
    scoresLDA_HOG['score_time'], scoresLDA_HOG['test_precision_macro'],
    scoresLDA_HOG['test_recall_macro'],scoresLDA_HOG['test_precision_micro'
    ],scoresLDA_HOG['test_recall_micro'],scoresLDA_HOG['test_f1_macro'],
    scoresLDA_HOG['test_accuracy']],
77                  'ModelKNN_HOG': [scoresKNN_HOG['fit_time'],
    scoresKNN_HOG['score_time'], scoresKNN_HOG['test_precision_macro'],
    scoresKNN_HOG['test_recall_macro'],scoresKNN_HOG['test_precision_micro'
    ],scoresKNN_HOG['test_recall_micro'],scoresKNN_HOG['test_f1_macro'],
    scoresKNN_HOG['test_accuracy']],
                    'ModelDT_HOG': [scoresDT_HOG['fit_time'], scoresDT_HOG[
    'score_time'], scoresDT_HOG['test_precision_macro'], scoresDT_HOG['
    test_recall_macro'],scoresDT_HOG['test_precision_micro'],scoresDT_HOG['
    test_recall_micro'],scoresDT_HOG['test_f1_macro'],scoresDT_HOG['
    test_accuracy']],
79                  'ModelRF_HOG': [scoresRF_HOG['fit_time'], scoresRF_HOG[
    'score_time'], scoresRF_HOG['test_precision_macro'], scoresRF_HOG['
    test_recall_macro'],scoresRF_HOG['test_precision_micro'],scoresRF_HOG['
    test_recall_micro'],scoresRF_HOG['test_f1_macro'],scoresRF_HOG['
    test_accuracy']],
                    'ModelSVM_HOG': [scoresSVM_HOG['fit_time'],
    scoresSVM_HOG['score_time'], scoresSVM_HOG['test_precision_macro'],
    scoresSVM_HOG['test_recall_macro'],scoresSVM_HOG['test_precision_micro'
    ],scoresSVM_HOG['test_recall_micro'],scoresSVM_HOG['test_f1_macro'],
    scoresSVM_HOG['test_accuracy']],
81                  'ModelCNN_mobilenet': [scoresCNN_mobilenet['fit_time'],
     scoresCNN_mobilenet['score_time'], scoresCNN_mobilenet['
    test_precision_macro'], scoresCNN_mobilenet['test_recall_macro'],
    scoresCNN_mobilenet['test_precision_micro'],scoresCNN_mobilenet['
    test_recall_micro'],scoresCNN_mobilenet['test_f1_macro'],
    scoresCNN_mobilenet['test_accuracy']],
                    'ModelCNN_resnet': [scoresCNN_resnet['fit_time'],
    scoresCNN_resnet['score_time'], scoresCNN_resnet['test_precision_macro'
    ], scoresCNN_resnet['test_recall_macro'],scoresCNN_resnet['
    test_precision_micro'],scoresCNN_resnet['test_recall_micro'],
    scoresCNN_resnet['test_f1_macro'],scoresCNN_resnet['test_accuracy']],
```

```
                    'ModelCNN_inception': [scoresCNN_inception['fit_time'],
 scoresCNN_inception['score_time'], scoresCNN_inception['
test_precision_macro'], scoresCNN_inception['test_recall_macro'],
scoresCNN_inception['test_precision_micro'],scoresCNN_inception['
test_recall_micro'],scoresCNN_inception['test_f1_macro'],
scoresCNN_inception['test_accuracy']],
                    'ModelCNN_efficientnet': [scoresCNN_efficientnet['
fit_time'], scoresCNN_efficientnet['score_time'], scoresCNN_efficientnet
['test_precision_macro'], scoresCNN_efficientnet['test_recall_macro'],
scoresCNN_efficientnet['test_precision_micro'],scoresCNN_efficientnet['
test_recall_micro'],scoresCNN_efficientnet['test_f1_macro'],
scoresCNN_efficientnet['test_accuracy']]
                    })
 df = df[['Modelo', 'ModelLR_canny','ModelLDA_canny','ModelKNN_canny','
ModelDT_canny','ModelRF_canny','ModelSVM_canny', 'ModelLR_HOG','
ModelLDA_HOG','ModelKNN_HOG','ModelDT_HOG','ModelRF_HOG','ModelSVM_HOG',
'ModelCNN_mobilenet','ModelCNN_resnet','ModelCNN_inception','
ModelCNN_efficientnet']]
 path = os.path.join(os.getcwd(), 'Escritorio/trabajo_aprendizaje/excel/
Metricas.xlsx')
 print(path)
 writer = ExcelWriter(path)
 df.to_excel(writer, 'Hoja de datos', index=False)
 writer.save()

 df = pd.DataFrame({'Modelo': ['fit_time', 'score_time', '
test_precision_macro', 'test_recall_macro', 'test_precision_micro','
test_recall_micro','test_f1_macro','test_accuracy'],
                    'ModelLR_canny': [np.mean(scoresLR_canny['fit_time']),
np.mean(scoresLR_canny['score_time']), np.mean(scoresLR_canny['
test_precision_macro']), np.mean(scoresLR_canny['test_recall_macro']),np
.mean(scoresLR_canny['test_precision_micro']),np.mean(scoresLR_canny['
test_recall_micro']),np.mean(scoresLR_canny['test_f1_macro']),np.mean(
scoresLR_canny['test_accuracy'])],
                    'ModelLDA_canny': [np.mean(scoresLDA_canny['fit_time'])
, np.mean(scoresLDA_canny['score_time']), np.mean(scoresLDA_canny['
test_precision_macro']), np.mean(scoresLDA_canny['test_recall_macro']),
np.mean(scoresLDA_canny['test_precision_micro']),np.mean(scoresLDA_canny
['test_recall_micro']),np.mean(scoresLDA_canny['test_f1_macro']),np.mean
(scoresLDA_canny['test_accuracy'])],
                    'ModelKNN_canny': [np.mean(scoresKNN_canny['fit_time'])
, np.mean(scoresKNN_canny['score_time']), np.mean(scoresKNN_canny['
test_precision_macro']), np.mean(scoresKNN_canny['test_recall_macro']),
np.mean(scoresKNN_canny['test_precision_micro']),np.mean(scoresKNN_canny
['test_recall_micro']),np.mean(scoresKNN_canny['test_f1_macro']),np.mean
(scoresKNN_canny['test_accuracy'])],
```

```
97                      'ModelDT_canny': [np.mean(scoresDT_canny['fit_time']),
    np.mean(scoresDT_canny['score_time']), np.mean(scoresDT_canny['
    test_precision_macro']), np.mean(scoresDT_canny['test_recall_macro']),np
    .mean(scoresDT_canny['test_precision_micro']),np.mean(scoresDT_canny['
    test_recall_micro']),np.mean(scoresDT_canny['test_f1_macro']),np.mean(
    scoresDT_canny['test_accuracy'])],
                        'ModelRF_canny': [np.mean(scoresRF_canny['fit_time']),
    np.mean(scoresRF_canny['score_time']), np.mean(scoresRF_canny['
    test_precision_macro']), np.mean(scoresRF_canny['test_recall_macro']),np
    .mean(scoresRF_canny['test_precision_micro']),np.mean(scoresRF_canny['
    test_recall_micro']),np.mean(scoresRF_canny['test_f1_macro']),np.mean(
    scoresRF_canny['test_accuracy'])],
99                      'ModelSVM_canny': [np.mean(scoresSVM_canny['fit_time'])
    , np.mean(scoresSVM_canny['score_time']), np.mean(scoresSVM_canny['
    test_precision_macro']), np.mean(scoresSVM_canny['test_recall_macro']),
    np.mean(scoresSVM_canny['test_precision_micro']),np.mean(scoresSVM_canny
    ['test_recall_micro']),np.mean(scoresSVM_canny['test_f1_macro']),np.mean
    (scoresSVM_canny['test_accuracy'])],
                        'ModelLR_HOG': [np.mean(scoresLR_HOG['fit_time']), np.
    mean(scoresLR_HOG['score_time']), np.mean(scoresLR_HOG['
    test_precision_macro']), np.mean(scoresLR_HOG['test_recall_macro']),np.
    mean(scoresLR_HOG['test_precision_micro']),np.mean(scoresLR_HOG['
    test_recall_micro']),np.mean(scoresLR_HOG['test_f1_macro']),np.mean(
    scoresLR_HOG['test_accuracy'])],
101                     'ModelLDA_HOG': [np.mean(scoresLDA_HOG['fit_time']), np
    .mean(scoresLDA_HOG['score_time']), np.mean(scoresLDA_HOG['
    test_precision_macro']), np.mean(scoresLDA_HOG['test_recall_macro']),np.
    mean(scoresLDA_HOG['test_precision_micro']),np.mean(scoresLDA_HOG['
    test_recall_micro']),np.mean(scoresLDA_HOG['test_f1_macro']),np.mean(
    scoresLDA_HOG['test_accuracy'])],
                        'ModelKNN_HOG': [np.mean(scoresKNN_HOG['fit_time']), np
    .mean(scoresKNN_HOG['score_time']), np.mean(scoresKNN_HOG['
    test_precision_macro']), np.mean(scoresKNN_HOG['test_recall_macro']),np.
    mean(scoresKNN_HOG['test_precision_micro']),np.mean(scoresKNN_HOG['
    test_recall_micro']),np.mean(scoresKNN_HOG['test_f1_macro']),np.mean(
    scoresKNN_HOG['test_accuracy'])],
103                     'ModelDT_HOG': [np.mean(scoresDT_HOG['fit_time']), np.
    mean(scoresDT_HOG['score_time']), np.mean(scoresDT_HOG['
    test_precision_macro']), np.mean(scoresDT_HOG['test_recall_macro']),np.
    mean(scoresDT_HOG['test_precision_micro']),np.mean(scoresDT_HOG['
    test_recall_micro']),np.mean(scoresDT_HOG['test_f1_macro']),np.mean(
    scoresDT_HOG['test_accuracy'])],
                        'ModelRF_HOG': [np.mean(scoresRF_HOG['fit_time']), np.
    mean(scoresRF_HOG['score_time']), np.mean(scoresRF_HOG['
    test_precision_macro']), np.mean(scoresRF_HOG['test_recall_macro']),np.
    mean(scoresRF_HOG['test_precision_micro']),np.mean(scoresRF_HOG['
```

```
    test_recall_micro'])  ,np.mean(scoresRF_HOG['test_f1_macro'])  ,np.mean(
    scoresRF_HOG['test_accuracy'])],
105                 'ModelSVM_HOG': [np.mean(scoresSVM_HOG['fit_time']),  np
    .mean(scoresSVM_HOG['score_time']),  np.mean(scoresSVM_HOG['
    test_precision_macro']),  np.mean(scoresSVM_HOG['test_recall_macro'])  ,np.
    mean(scoresSVM_HOG['test_precision_micro'])  ,np.mean(scoresSVM_HOG['
    test_recall_micro'])  ,np.mean(scoresSVM_HOG['test_f1_macro'])  ,np.mean(
    scoresSVM_HOG['test_accuracy'])],
                    'ModelCNN_mobilenet': [np.mean(scoresCNN_mobilenet['
    fit_time']),  np.mean(scoresCNN_mobilenet['score_time']),  np.mean(
    scoresCNN_mobilenet['test_precision_macro']),  np.mean(
    scoresCNN_mobilenet['test_recall_macro'])  ,np.mean(scoresCNN_mobilenet['
    test_precision_micro'])  ,np.mean(scoresCNN_mobilenet['test_recall_micro'
    ])  ,np.mean(scoresCNN_mobilenet['test_f1_macro'])  ,np.mean(
    scoresCNN_mobilenet['test_accuracy'])],
107                 'ModelCNN_resnet': [np.mean(scoresCNN_resnet['fit_time'
    ]),  np.mean(scoresCNN_resnet['score_time']),  np.mean(scoresCNN_resnet['
    test_precision_macro']),  np.mean(scoresCNN_resnet['test_recall_macro']),
    np.mean(scoresCNN_resnet['test_precision_micro'])  ,np.mean(
    scoresCNN_resnet['test_recall_micro'])  ,np.mean(scoresCNN_resnet['
    test_f1_macro'])  ,np.mean(scoresCNN_resnet['test_accuracy'])],
                    'ModelCNN_inception': [np.mean(scoresCNN_inception['
    fit_time']),  np.mean(scoresCNN_inception['score_time']),  np.mean(
    scoresCNN_inception['test_precision_macro']),  np.mean(
    scoresCNN_inception['test_recall_macro'])  ,np.mean(scoresCNN_inception['
    test_precision_micro'])  ,np.mean(scoresCNN_inception['test_recall_micro'
    ])  ,np.mean(scoresCNN_inception['test_f1_macro'])  ,np.mean(
    scoresCNN_inception['test_accuracy'])],
109                 'ModelCNN_efficientnet': [np.mean(
    scoresCNN_efficientnet['fit_time']),  np.mean(scoresCNN_efficientnet['
    score_time']),  np.mean(scoresCNN_efficientnet['test_precision_macro']),
    np.mean(scoresCNN_efficientnet['test_recall_macro'])  ,np.mean(
    scoresCNN_efficientnet['test_precision_micro'])  ,np.mean(
    scoresCNN_efficientnet['test_recall_micro'])  ,np.mean(
    scoresCNN_efficientnet['test_f1_macro'])  ,np.mean(scoresCNN_efficientnet[
    'test_accuracy'])]
                    })
111
     df = df[['Modelo',  'ModelLR_canny','ModelLDA_canny','ModelKNN_canny','
    ModelDT_canny','ModelRF_canny','ModelSVM_canny',  'ModelLR_HOG','
    ModelLDA_HOG','ModelKNN_HOG','ModelDT_HOG','ModelRF_HOG','ModelSVM_HOG',
    'ModelCNN_mobilenet','ModelCNN_resnet','ModelCNN_inception','
    ModelCNN_efficientnet']]
113  path = os.path.join(os.getcwd(),  'Escritorio/trabajo_aprendizaje/excel/
    Metricas_medias.xlsx')
     print(path)
```

```
    writer = ExcelWriter(path)
    df.to_excel(writer, 'Hoja de datos', index=False)
    writer.save()

    #Metodos_evaluacion.comparacion_modelos(scoresLR_canny,scoresLDA_canny,
    scoresKNN_canny,scoresDT_canny,scoresRF_canny,scoresSVM_canny,
    scoresLR_HOG,scoresLDA_HOG,scoresKNN_HOG,scoresDT_HOG,scoresRF_HOG,
    scoresSVM_HOG)
    Metodos_evaluacion.hipotesis(scoresLR_canny,scoresLDA_canny,
    scoresKNN_canny,scoresDT_canny,scoresRF_canny,scoresSVM_canny,
    scoresLR_HOG,scoresLDA_HOG,scoresKNN_HOG,scoresDT_HOG,scoresRF_HOG,
    scoresSVM_HOG)


if __name__ == "__main__":
    tamano = 64
    print('Opciones:')
    print('1 - Ejecutar el código principal')
    print('2 - Ejecutar código de EfficientNet B0-B4')
    print('3 - Aplicación')
    print('4 - Salir')

    bucle = 1
    while bucle == 1:
        opcion = int(input('Introduce el número de la opción deseada: '))
        if opcion == 1:
            main()
            bucle = 0
        elif opcion == 2:
            efficientnet_completo.main_efficientnet()
            bucle = 0
        elif opcion == 3:
            bucle = 0
            import aplicacion
        elif opcion == 4:
            bucle = 0

        if bucle == 1:
            print('Error,introduzca un número del 1 al 4.')
```

Código main.

16.2. Script preprocesado

```
'''Importación de librerías'''
'''librerías para trabajar con los paths'''
import os
import re
import numpy as np

import matplotlib.pyplot as plt
import cv2

from sklearn.model_selection import train_test_split

#
    ------------------------------------------------------------------------

def lectura_imagenes(carpeta):
    '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
    '''Se obtiene la ubicación del script actual y se le añade el nombre de
    la carpeta donde se encuentran las imágenes'''
    '''En jupyter no era necesario poner más que el nombre de la carpeta,
   mientras que en python es necesario poner todo desde la carpeta de
   usauario'''
    dirname = os.path.join(os.getcwd(), carpeta)
    imgpath = dirname + os.sep
    print(imgpath)
    '''Inicialización de variables empleadas en la lectura'''
    images = []
    directories = []
    dircount = []
    prevRoot=''
    cant=0

    '''Inicialización de listas para los paths de las imágenes'''
    lista_uno = []
    lista_dos = []
    lista_tres = []
    lista_cuatro = []
    lista_cinco = []
    lista_parriba = []
    lista_pabajo = []

    print("leyendo imagenes de ",imgpath)
    '''Se recorren todas las imagenes y se van leyendo'''
    for root, _, filenames in os.walk(imgpath):
```

```
        '''print(root,'+',filenames)'''
        #valor =valor_imagen(root)
        #print(filename)
        for filename in filenames:
            if re.search("\.(jpg|jpeg|png|txt)$", filename):
                cant=cant+1
                filepath = os.path.join(root, filename)

                '''
                image = plt.imread(filepath)

                images.append(image)
                '''

                '''En windows la barra diagonal inversa \\ en linux /'''
                ruta=root.split('/')
                '''print(ruta)'''
                '''Tras separar la ruta, según el nombre de la carpeta, se incluye la imagen en su lista correspondiente'''
                if ruta[len(ruta)-1]=='uno':
                    lista_uno.append(filepath)

                elif ruta[len(ruta)-1] == 'dos':
                    lista_dos.append(filepath)

                elif ruta[len(ruta)-1] == 'tres':
                    lista_tres.append(filepath)

                elif ruta[len(ruta)-1] == 'cuatro':
                    lista_cuatro.append(filepath)

                elif ruta[len(ruta)-1] == 'cinco':
                    lista_cinco.append(filepath)

                elif ruta[len(ruta)-1] == 'pulgar_abajo':
                    lista_pabajo.append(filepath)

                elif ruta[len(ruta)-1] == 'pulgar_arriba':
                    lista_parriba.append(filepath)


                b = "Leyendo..." + str(cant)
                print (b, end="\r")
                if prevRoot !=root:
                    print(root, cant)
                    prevRoot=root
```

```
                    directories.append(root)
                    dircount.append(cant)
                    cant=0


    dircount.append(cant)
    '''
    print(images[0])
    print(images[1])
    '''
    dircount = dircount[1:]
    dircount[0]=dircount[0]+1
    print('Directorios leidos:',len(directories))
    print("Imagenes en cada directorio", dircount)
    print('Total de imagenes en subdirs:',sum(dircount))

    '''Se hace un diccionario con todas las listas y el nombre del gesto correspondiente'''
    gestos_images_dict = {
        'cinco': lista_cinco,
        'cuatro': lista_cuatro,
        'tres': lista_tres,
        'dos': lista_dos,
        'uno': lista_uno,
        'pulgar_abajo': lista_pabajo,
        'pulgar_arriba': lista_parriba,
    }

    '''Se hace un diccionario con los valores numéricos asignados a cada gesto'''
    gestos_labels_dict = {
        'cinco': 5,
        'cuatro': 4,
        'tres': 3,
        'dos': 2,
        'uno': 1,
        'pulgar_abajo': 6,
        'pulgar_arriba': 0,
    }

    return gestos_images_dict,gestos_labels_dict

#
    ------------------------------------------------------------------------
```

```
125 def separacion_normalizacion_datos_CNN(gestos_images_dict,
    gestos_labels_dict,tamano):
    '''Se introducen las imagenes y se reescalan y se normalizan'''
127     X=[]
    y=[]
129
    for gestos_name, images in gestos_images_dict.items():
131         for image in images:
            img = plt.imread(image)
133
            resized_img = cv2.resize(img,(tamano,tamano))
135         #print(resized_img)
            X.append(resized_img)
137         y.append(gestos_labels_dict[gestos_name])

139     X = np.array(X)
    y = np.array(y)
141     #print(np.shape(X))
    X_train, X_test, y_train, y_test = train_test_split(X, y)
143
    X_aplanado = X/255
145
    X_train_scaled = X_train / 255
147     X_test_scaled = X_test / 255
    return X_aplanado,X_train_scaled,X_test_scaled,y,y_train,y_test
149
def separacion_normalizacion_datos_CV(gestos_images_dict,gestos_labels_dict
    ,tamano):
151
    '''Se introducen las imagenes y se reescalan y se normalizan'''
153     X=[]
    y=[]
155
    for gestos_name, images in gestos_images_dict.items():
157         for image in images:

159             img= cv2.imread(image,0)

161             resized_img = cv2.resize(img,(tamano,tamano))
            #print(resized_img)
163             X.append(resized_img)
            y.append(gestos_labels_dict[gestos_name])
165
    X = np.array(X)
167     y = np.array(y)
    #print(np.shape(X))
```

```
169     X_train, X_test, y_train, y_test = train_test_split(X, y)

171     X_aplanado=[]
        for i in range(len(X)):
173         X_aplanado.append(X[i].ravel())
        X_aplanado=np.array(X_aplanado)
175
        X_train_aplanado=[]
177     X_test_aplanado=[]
        for i in range(len(X_test)):
179         X_test_aplanado.append(X_test[i].ravel())
        for i in range(len(X_train)):
181         X_train_aplanado.append(X_train[i].ravel())

183     X_train_aplanado=np.array(X_train_aplanado)
        X_test_aplanado=np.array(X_test_aplanado)
185


187     X_aplanado = X_aplanado/255

189     X_train_aplanado = X_train_aplanado / 255
        X_test_aplanado = X_test_aplanado / 255
191


193     return X_aplanado,X_train_aplanado,X_test_aplanado,y,y_train,y_test

195 def separacion_normalizacion_datos_CV_HOG(gestos_images_dict,
    gestos_labels_dict):

197     '''Se introducen las imagenes y se reescalan y se normalizan'''
        X=[]
199     y=[]

201     for gestos_name, images in gestos_images_dict.items():
            for image in images:
203
                img= np.loadtxt(image)
205
                X.append(img)
207             y.append(gestos_labels_dict[gestos_name])

209     X = np.array(X)
        y = np.array(y)
211     #print(np.shape(X))
        X_train, X_test, y_train, y_test = train_test_split(X, y)
213
```

```
    X_aplanado=X
215
    X_train_aplanado=X_train
217 X_test_aplanado=X_test

219

221 return X_aplanado, X_train_aplanado, X_test_aplanado, y, y_train, y_test
```

Código de preprocesado.

16.3. Script busqueda_hiperametros

```
'''Importación de librerías'''
2 '''librerías para trabajar con los paths'''
import os
4 import re

6 from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
8 from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
10 from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
12
import Preprocesado
14 import CNN_transfer
import Metodos_evaluacion
16 import escribir_excel

18 from sklearn.model_selection import RandomizedSearchCV

20 import pandas as pd
from pandas import ExcelWriter
22

24 def RL_hiper(X_aplanado, y , canny_hog):

26     n_comb=30
    ''','sag','saga'''
28     hiperparam = {'solver': ['newton-cg','lbfgs'],
                'penalty': ['l2','none'],
```

```
                'multi_class':['multinomial'],
                'n_jobs': [-1]}
    modelLR = LogisticRegression( max_iter=1000)
    model_search = RandomizedSearchCV(modelLR, param_distributions=
    hiperparam, n_iter=n_comb, n_jobs=-1, cv=10, verbose=3, random_state
    =1001)
    model_search.fit(X_aplanado, y)

    modelo_final = model_search.best_estimator_
    resultados = model_search.cv_results_
    if canny_hog==0:
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_LR_canny.xlsx','Hiper_LR_canny',resultados)
    else:
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_LR_hog.xlsx','Hiper_LR_hog',resultados)


    return modelo_final

def LDA_hiper(X_aplanado, y , canny_hog):

    n_comb = 4

    hiperparam = {'solver': ['lsqr','svd']}

    modelLDA = LinearDiscriminantAnalysis()

    model_search = RandomizedSearchCV(modelLDA, param_distributions=
    hiperparam, n_iter=n_comb, n_jobs=-1, cv=10, verbose=3, random_state
    =1001)
    model_search.fit(X_aplanado, y)
    modelo_final = model_search.best_estimator_
    print(model_search.best_estimator_)
    resultados = model_search.cv_results_
    if canny_hog==0:
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_LDA_canny.xlsx','Hiper_LDA_canny',resultados)
    else:
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_LDA_hog.xlsx','Hiper_hog',resultados)

    return modelo_final

def KNN_hiper(X_aplanado, y , canny_hog):
```

```
68
    n_comb = 30
70
    hiperparam = {'n_jobs': [-1],
72              'n_neighbors': list(range(1, 50)),
                'weights': ['uniform', 'distance'],
74              'algorithm':['ball_tree', 'kd_tree', 'brute'],
                "leaf_size": [5,7,11,15,20,30,40,50,70,80,90],
76              "metric": ['minkowski'],
                "p": [1, 2]}
78
    modelKNN = KNeighborsClassifier()
80
    model_search = RandomizedSearchCV(modelKNN, param_distributions=
    hiperparam, n_iter=n_comb, n_jobs=-1, cv=10, verbose=3, random_state
    =1001)
82  model_search.fit(X_aplanado, y)
    modelo_final = model_search.best_estimator_
84  print(model_search.best_estimator_)
    resultados = model_search.cv_results_
86
    if canny_hog==0:
88      escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_KNN_canny.xlsx','Hiper_KNN_canny',resultados)
    else:
90      escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_KNN_hog.xlsx','Hiper_KNN_hog',resultados)

92  return modelo_final

94


96
def DT_hiper(X_aplanado, y , canny_hog):
98
    n_comb = 100
100
    hiperparam = {"max_depth": [100],
102             'splitter': ['best', 'random'],
                "max_features": ['sqrt','log2'],
104             "min_samples_split": [5,7,11,15,20,30,40,50,70,80,90],
                "min_samples_leaf": [1,5,7,11,15,20,30,40,50,70,80,90],
106             "criterion": ["gini", "entropy"]}

108
```

```
110     modelDT = DecisionTreeClassifier(criterion='gini',max_depth=30)

112     model_search = RandomizedSearchCV(modelDT, param_distributions=
        hiperparam, n_iter=n_comb, n_jobs=-1, cv=10, verbose=3, random_state
        =1001)
        model_search.fit(X_aplanado, y)
114     modelo_final = model_search.best_estimator_
        print(model_search.best_estimator_)
116     resultados = model_search.cv_results_

118     if canny_hog==0:
            escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        Hiper_DT_canny.xlsx','Hiper_DT_canny',resultados)
120     else:
            escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        Hiper_DT_hog.xlsx','Hiper_DT_hog',resultados)
122
        return modelo_final
124
    def RF_hiper(X_aplanado, y , canny_hog):
126


128     n_comb = 30

130     hiperparam = {"max_depth": [100],
                "max_features": ['sqrt','log2'],
132             "min_samples_split": [5,7,11,15,20,30,40,50,70,80,90],
                "min_samples_leaf": [1,5,7,11,15,20,30,40,50,70,80,90],
134             "bootstrap": [True, False],
                "criterion": ["gini", "entropy"]}
136
        modelRF = RandomForestClassifier(n_estimators=30)
138
        model_search = RandomizedSearchCV(modelRF, param_distributions=
        hiperparam, n_iter=n_comb, n_jobs=-1, cv=10, verbose=3, random_state
        =1001)
140     model_search.fit(X_aplanado, y)
        modelo_final = model_search.best_estimator_
142     print(model_search.best_estimator_)
        resultados = model_search.cv_results_
144     if canny_hog==0:
            escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        Hiper_RF_canny.xlsx','Hiper_RF_canny',resultados)
146     else:
            escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        Hiper_RF_hog.xlsx','Hiper_RF_hog',resultados)
```

```
148
    return modelo_final
150
def SVM_hiper(X_aplanado, y , canny_hog):
152
    n_comb = 30
154
    hiperparam = {"kernel": ['linear','poly','rbf','sigmoid'],
156             "gamma": [0.01,'auto'],
             "decision_function_shape": ['ovo','ovr']}
158
    modelRF = SVC()
160
    model_search = RandomizedSearchCV(modelRF, param_distributions=
    hiperparam, n_iter=n_comb, n_jobs=-1, cv=10, verbose=3, random_state
    =1001)
162 model_search.fit(X_aplanado, y)
    modelo_final = model_search.best_estimator_
164
    print(model_search.best_estimator_)
166 resultados = model_search.cv_results_

168 if canny_hog==0:
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_SVM_canny.xlsx','Hiper_SVM_canny',resultados)
170 else:
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    Hiper_SVM_hog.xlsx','Hiper_SVM_hog',resultados)
172
    return modelo_final
```

Código de búsqueda de hiperparámetros.

16.4. Script modelos_aprendizaje

```
1 '''Importación de librerías'''
  '''librerías para trabajar con los paths'''
3 import os
  import re
5 '''librerias para manejar las imagenes'''
  import numpy as np
7 import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import cross_validate

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler


from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc, accuracy_score, recall_score,
    precision_score, plot_confusion_matrix, classification_report

import Preprocesado
import CNN_transfer
import Metodos_evaluacion
import busqueda_hiperparametros
import escribir_excel

from pandas import ExcelWriter
'''CNN MEDIANTE TRANSFER LEARNING'''
def CNN_FINAL(tamano):

    '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
    gestos_images_dict,gestos_labels_dict = Preprocesado.lectura_imagenes(
    carpeta = 'Escritorio/trabajo_aprendizaje/muchasimagenes')

    #
    ----------------------------------------------------------------------------


    '''Se introducen las imagenes y se reescalan y se normalizan.'''
    X_aplanado, X_train_scaled, X_test_scaled,y , y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CNN(gestos_images_dict,
    gestos_labels_dict,tamano)
    '''Inception pide las imagenes a 299x299'''
    X_aplanado_inc, X_train_scaled_inc, X_test_scaled_inc,y_inc ,
    y_train_inc, y_test_inc = Preprocesado.
    separacion_normalizacion_datos_CNN(gestos_images_dict,gestos_labels_dict
    ,299)
    '''EfficientNet pide las imagenes a 380x380'''
    X_aplanado_ef, X_train_scaled_ef, X_test_scaled_ef,y_ef , y_train_ef,
    y_test_ef = Preprocesado.separacion_normalizacion_datos_CNN(
    gestos_images_dict,gestos_labels_dict,380)
    #-------------------------------------------------------------------
    '''CNN'''
    '''La primera función realiza la validación cruzada además de entrenar
    el modelo,
```

```
41      mientras que para solo entrenarlo habría que ejecutar las dos
        siguientes.'''

43      model_efficientnet, history_efficientnet,results_efficientnet=
        CNN_transfer.CNN_transferlearning(X_aplanado_ef, X_train_scaled_ef,
        X_test_scaled_ef,y_ef , y_train_ef, y_test_ef,80,5,red_preentrenada='
        efficientnet')
        model_mobilenet, history_mobilenet,results_mobilenet= CNN_transfer.
        CNN_transferlearning(X_aplanado, X_train_scaled, X_test_scaled,y ,
        y_train, y_test,80,5,red_preentrenada='mobilenet')
45      model_resnet, history_resnet,results_resnet= CNN_transfer.
        CNN_transferlearning(X_aplanado, X_train_scaled, X_test_scaled,y ,
        y_train, y_test,80,5,red_preentrenada='resnet')
        model_inception, history_inception,results_inception= CNN_transfer.
        CNN_transferlearning(X_aplanado_inc, X_train_scaled_inc,
        X_test_scaled_inc,y_inc , y_train_inc, y_test_inc,80,5,red_preentrenada=
        'inception')
47

49      '''
        model = CNN_transfer.CNN_creacion()
51      modelo, history = CNN_transfer.CNN_fit(model,X_train_scaled, y_train,
        X_test_scaled, y_test, 5)
        '''
53      model_efficientnet.evaluate(x=X_test_scaled_ef,y=y_test_ef)
        Metodos_evaluacion.graficas_acc_loss(history_efficientnet, '/home/
        nunuel/Escritorio/trabajo_aprendizaje/curva_acc_loss/
        acc_loss_efficientnetb4.jpg')
55      Metodos_evaluacion.curva_ROC(model_efficientnet,X_test_scaled,y_test,'/
        home/nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/
        ROC_CNN_efficientnetb7.jpg')
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        CNN_results_efficientnet.xlsx','CNN_results',results_efficientnet)
57

        model_mobilenet.evaluate(x=X_test_scaled,y=y_test)
59      Metodos_evaluacion.graficas_acc_loss(history_mobilenet, '/home/nunuel/
        Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_mobilenet.jpg')
        Metodos_evaluacion.curva_ROC(model_mobilenet,X_test_scaled,y_test,'/
        home/nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_CNN_mobilenet.
        jpg')
61      escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        CNN_results_mobilenet.xlsx','CNN_results',results_mobilenet)

63      model_resnet.evaluate(x=X_test_scaled,y=y_test)
        Metodos_evaluacion.graficas_acc_loss(history_resnet, '/home/nunuel/
        Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_resnet.jpg')
```

```
65      Metodos_evaluacion.curva_ROC(model_resnet,X_test_scaled,y_test,'/home/
        nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_CNN_resnet.jpg')
        escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        CNN_results_resnet.xlsx','CNN_results',results_resnet)
67
        model_inception.evaluate(x=X_test_scaled_inc,y=y_test_inc)
69      Metodos_evaluacion.graficas_acc_loss(history_inception, '/home/nunuel/
        Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_inception.jpg')
        Metodos_evaluacion.curva_ROC(model_inception,X_test_scaled_inc,
        y_test_inc,'/home/nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/
        ROC_CNN_inception.jpg')
71      escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        CNN_results_inception.xlsx','CNN_results',results_inception)


73


75      #Metodos_evaluacion.predicción_imagenes_aleatorias(model,X_test_scaled,
        y_test)
        #escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/CNN_history
        .xlsx','CNN_history',(history[acc])
77
        return results_mobilenet,results_resnet,results_inception,
        results_efficientnet
79
def RL_FINAL(tamano,path,canny_hog):
81      '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
        gestos_images_dict,gestos_labels_dict = Preprocesado.lectura_imagenes(
        carpeta = path)
83      '''Se introducen las imagenes y se reescalan y se normalizan.'''
        if canny_hog==0:
85          X_aplanado, X_train_scaled, X_test_scaled,y , y_train, y_test =
        Preprocesado.separacion_normalizacion_datos_CV(gestos_images_dict,
        gestos_labels_dict,tamano)
        else:
87          X_aplanado, X_train_scaled, X_test_scaled,y , y_train, y_test =
        Preprocesado.separacion_normalizacion_datos_CV_HOG(gestos_images_dict,
        gestos_labels_dict)
        '''REGRESIÓN LOGÍSTICA'''
89
        scoring = ['precision_macro', 'recall_macro', 'precision_micro', '
        recall_micro', 'f1_macro','accuracy']
91
        modelo_final = busqueda_hiperparametros.RL_hiper(X_aplanado, y ,
        canny_hog)
93
        modelos = []
```

```
95      modelos.append(('LR', modelo_final))
97
        for (nombre, modelo) in modelos:
99          modelo = make_pipeline(StandardScaler(), modelo)
            scores = cross_validate(modelo, X_aplanado, y, cv=10, scoring=
        scoring, n_jobs=-1)
101         print('Media del ', nombre, ': ', np.mean(scores['test_accuracy']))
103
        modelLR = modelo_final
105     modelLR.fit(X_train_scaled, y_train)
107     y_pred = modelLR.predict(X_test_scaled)
109     if canny_hog==0:
            Metodos_evaluacion.matriz_confusion(modelLR, X_test_scaled, y_test, '/
        home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
        matriz_RL_canny.jpg')
111         Metodos_evaluacion.metricas_evaluacion(y_test, y_pred)
            Metodos_evaluacion.curva_ROC(modelLR, X_test_scaled, y_test, '/home/
        nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_RL_canny.jpg')
113     else:
            Metodos_evaluacion.matriz_confusion(modelLR, X_test_scaled, y_test, '/
        home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
        matriz_RL_HOG.jpg')
115         Metodos_evaluacion.metricas_evaluacion(y_test, y_pred)
            Metodos_evaluacion.curva_ROC(modelLR, X_test_scaled, y_test, '/home/
        nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_RL_HOG.jpg')
117
        return scores
119
    def LDA_FINAL(tamano, path, canny_hog):
121     '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
        gestos_images_dict, gestos_labels_dict = Preprocesado.lectura_imagenes(
        carpeta = path)
123     '''Se introducen las imagenes y se reescalan y se normalizan.'''
        if canny_hog==0:
125         X_aplanado, X_train_scaled, X_test_scaled, y , y_train, y_test =
        Preprocesado.separacion_normalizacion_datos_CV(gestos_images_dict,
        gestos_labels_dict, tamano)
        else:
127         X_aplanado, X_train_scaled, X_test_scaled, y , y_train, y_test =
        Preprocesado.separacion_normalizacion_datos_CV_HOG(gestos_images_dict,
        gestos_labels_dict)
```

```
129     '''LDA'''

131     scoring = ['precision_macro', 'recall_macro', 'precision_micro', '
    recall_micro', 'f1_macro','accuracy']
        '''SE REALIZA LA BÚSQUEDA DE HIPERPARÁMETROS'''
133     modelo_final = busqueda_hiperparametros.LDA_hiper(X_aplanado, y ,
    canny_hog)

135     modelos = []
        modelos.append(('LDA', modelo_final))
137

139     for (nombre, modelo) in modelos:
            modelo = make_pipeline(StandardScaler(), modelo)
141         scores = cross_validate(modelo, X_aplanado, y, cv=10, scoring=
    scoring,n_jobs=-1)
            print('Media del ', nombre, ': ', np.mean(scores['test_accuracy']))
143
        modelLDA = modelo_final # Se define el modelo
145     modelLDA.fit(X_train_scaled,y_train)

147     y_pred = modelLDA.predict(X_test_scaled)

149     if canny_hog==0:
            Metodos_evaluacion.matriz_confusion(modelLDA,X_test_scaled,y_test,'
    /home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
    matriz_LDA_canny.jpg')
151         Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
            Metodos_evaluacion.curva_ROC(modelLDA,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_LDA_canny.jpg')
153     else:
            Metodos_evaluacion.matriz_confusion(modelLDA,X_test_scaled,y_test,'
    /home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
    matriz_LDA_HOG.jpg')
155         Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
            Metodos_evaluacion.curva_ROC(modelLDA,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_LDA_HOG.jpg')
157

159     return scores

161 def KNN_FINAL(tamano,path,canny_hog):
        '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
163     gestos_images_dict,gestos_labels_dict = Preprocesado.lectura_imagenes(
    carpeta = path)
        '''Se introducen las imagenes y se reescalan y se normalizan.'''
```

```
165     if canny_hog==0:
            X_aplanado, X_train_scaled, X_test_scaled,y , y_train, y_test =
        Preprocesado.separacion_normalizacion_datos_CV(gestos_images_dict,
        gestos_labels_dict,tamano)
167     else:
            X_aplanado, X_train_scaled, X_test_scaled,y , y_train, y_test =
        Preprocesado.separacion_normalizacion_datos_CV_HOG(gestos_images_dict,
        gestos_labels_dict)
169
        '''KNN'''
171
        scoring = ['precision_macro', 'recall_macro', 'precision_micro', '
        recall_micro', 'f1_macro','accuracy']
173
        modelos = []
175
        modelo_final = busqueda_hiperparametros.KNN_hiper(X_aplanado, y ,
        canny_hog)
177
        modelos.append(('KNN', modelo_final))
179
181     for (nombre, modelo) in modelos:
            modelo = make_pipeline(StandardScaler(), modelo)
183         scores = cross_validate(modelo, X_aplanado, y, cv=10, scoring=
        scoring,n_jobs=-1)
            print('Media del ', nombre, ': ', np.mean(scores['test_accuracy']))
185
        modelKNN = modelo_final
187     modelKNN.fit(X_train_scaled,y_train)

189     y_pred = modelKNN.predict(X_test_scaled)

191     if canny_hog==0:
            Metodos_evaluacion.matriz_confusion(modelKNN,X_test_scaled,y_test,'
        /home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
        matriz_KNN_canny.jpg')
193         Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
            Metodos_evaluacion.curva_ROC(modelKNN,X_test_scaled,y_test,'/home/
        nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_KNN_canny.jpg')
195     else:
            Metodos_evaluacion.matriz_confusion(modelKNN,X_test_scaled,y_test,'
        /home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
        matriz_KNN_HOG.jpg')
197         Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
```

```
        Metodos_evaluacion.curva_ROC(modelKNN, X_test_scaled, y_test, '/home/nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_KNN_HOG.jpg')

    return scores

def DT_FINAL(tamano, path, canny_hog):
    '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
    gestos_images_dict, gestos_labels_dict = Preprocesado.lectura_imagenes(carpeta = path)
    '''Se introducen las imagenes y se reescalan y se normalizan.'''
    if canny_hog==0:
        X_aplanado, X_train_scaled, X_test_scaled, y , y_train, y_test = Preprocesado.separacion_normalizacion_datos_CV(gestos_images_dict, gestos_labels_dict, tamano)
    else:
        X_aplanado, X_train_scaled, X_test_scaled, y , y_train, y_test = Preprocesado.separacion_normalizacion_datos_CV_HOG(gestos_images_dict, gestos_labels_dict)

    '''REGRESIÓN LOGÍSTICA'''

    scoring = ['precision_macro', 'recall_macro', 'precision_micro', 'recall_micro', 'f1_macro','accuracy']

    modelos = []
    modelo_final = busqueda_hiperparametros.DT_hiper(X_aplanado, y , canny_hog)
    modelos.append(('DT', modelo_final))


    for (nombre, modelo) in modelos:
        modelo = make_pipeline(StandardScaler(), modelo)
        scores = cross_validate(modelo, X_aplanado, y, cv=10, scoring=scoring, n_jobs=-1)
        print('Media del ', nombre, ': ', np.mean(scores['test_accuracy']))


    modelDT = modelo_final
    modelDT.fit(X_train_scaled, y_train)

    y_pred = modelDT.predict(X_test_scaled)

    if canny_hog==0:
        Metodos_evaluacion.matriz_confusion(modelDT, X_test_scaled, y_test, '/home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/matriz_DT_canny.jpg')
```

```
233             Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
                Metodos_evaluacion.curva_ROC(modelDT,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_DT_canny.jpg')
235         else:
                Metodos_evaluacion.matriz_confusion(modelDT,X_test_scaled,y_test,'/
    home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
    matriz_DT_HOG.jpg')
237             Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
                Metodos_evaluacion.curva_ROC(modelDT,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_DT_HOG.jpg')
239
        return scores
241
def RF_FINAL(tamano,path,canny_hog):
243     '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
        gestos_images_dict,gestos_labels_dict = Preprocesado.lectura_imagenes(
    carpeta = path)
245     '''Se introducen las imagenes y se reescalan y se normalizan.'''
        if canny_hog==0:
247         X_aplanado, X_train_scaled, X_test_scaled,y , y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CV(gestos_images_dict,
    gestos_labels_dict,tamano)
        else:
249         X_aplanado, X_train_scaled, X_test_scaled,y , y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CV_HOG(gestos_images_dict,
    gestos_labels_dict)

251     '''RANDOM FOREST'''

253     scoring = ['precision_macro', 'recall_macro', 'precision_micro', '
    recall_micro', 'f1_macro','accuracy']

255     modelos = []

257     modelo_final = busqueda_hiperparametros.RF_hiper(X_aplanado, y ,
    canny_hog)
        modelos.append(('RF', modelo_final))
259

261     for (nombre, modelo) in modelos:
            modelo = make_pipeline(StandardScaler(), modelo)
263         scores = cross_validate(modelo, X_aplanado, y, cv=10, scoring=
    scoring,n_jobs=-1)
            print('Media del ', nombre, ': ', np.mean(scores['test_accuracy']))
265
    modelRF = modelo_final
```

```
267
    modelRF.fit(X_train_scaled, y_train)
269
    y_pred = modelRF.predict(X_test_scaled)
271    if canny_hog==0:
        Metodos_evaluacion.matriz_confusion(modelRF, X_test_scaled, y_test, '/
    home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
    matriz_RF_canny.jpg')
273        Metodos_evaluacion.metricas_evaluacion(y_test, y_pred)
        Metodos_evaluacion.curva_ROC(modelRF, X_test_scaled, y_test, '/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_RF_canny.jpg')
275    else:
        Metodos_evaluacion.matriz_confusion(modelRF, X_test_scaled, y_test, '/
    home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
    matriz_RF_HOG.jpg')
277        Metodos_evaluacion.metricas_evaluacion(y_test, y_pred)
        Metodos_evaluacion.curva_ROC(modelRF, X_test_scaled, y_test, '/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_RF_HOG.jpg')
279
    return scores
281
def SVM_FINAL(tamano, path, canny_hog):
283    '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
    gestos_images_dict, gestos_labels_dict = Preprocesado.lectura_imagenes(
    carpeta = path)
285    '''Se introducen las imagenes y se reescalan y se normalizan.'''
    if canny_hog==0:
287        X_aplanado, X_train_scaled, X_test_scaled, y , y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CV(gestos_images_dict,
    gestos_labels_dict, tamano)
    else:
289        X_aplanado, X_train_scaled, X_test_scaled, y , y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CV_HOG(gestos_images_dict,
    gestos_labels_dict)

291    '''SVM'''

293    scoring = ['precision_macro', 'recall_macro', 'precision_micro', '
    recall_micro', 'f1_macro','accuracy']

295    modelo_final = busqueda_hiperparametros.SVM_hiper(X_aplanado, y ,
    canny_hog)

297    modelos = []
    modelos.append(('SVCrbf', modelo_final))
299
```

```
    for (nombre, modelo) in modelos:
301         modelo = make_pipeline(StandardScaler(), modelo)
            scores = cross_validate(modelo, X_aplanado, y, cv=10, scoring=
    scoring,n_jobs=-1)
303         print('Media del ', nombre, ': ', np.mean(scores['test_accuracy']))
            #test_sc.append(scores['test_accuracy'])
305
    modelSVM = modelo_final
307 modelSVM.fit(X_train_scaled,y_train)

309 y_pred = modelSVM.predict(X_test_scaled)

311 if canny_hog==0:
        Metodos_evaluacion.matriz_confusion(modelSVM,X_test_scaled,y_test,'
    /home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
    matriz_SVM_canny.jpg')
313     Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
    else:
315     Metodos_evaluacion.matriz_confusion(modelSVM,X_test_scaled,y_test,'
    /home/nunuel/Escritorio/trabajo_aprendizaje/matrices_confusion/
    matriz_SVM_HOG.jpg')
        Metodos_evaluacion.metricas_evaluacion(y_test,y_pred)
317
    return scores
```

Código con los modelos de aprendizaje.

16.5. Script CNN_transfer

```
'''Importación de librerías'''
2 '''librerías para trabajar con los paths'''
import os
4 import re

6 import numpy as np

8 import tensorflow as tf
import tensorflow_hub as hub
10
from tensorflow import keras
12 from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

```
14
16 from sklearn.model_selection import cross_validate
   from sklearn.model_selection import KFold
18
   from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
20

22 '''CNN'''
   '''Creación del modelo'''
24
   def cnn_mobilenet():
26
       '''Se obtiene la red preentrenada sin la última capa del repositorio de
       tensorflow'''
28
       feature_extractor_model = "https://tfhub.dev/google/tf2-preview/
      mobilenet_v2/feature_vector/4"
30     '''Cambiando el parámetro trainable se pueden entrenar casi todos los
      parámetros'''
       pretrained_model_without_top_layer = hub.KerasLayer(
32         feature_extractor_model, input_shape=(224, 224, 3), trainable=False
      )
       '''numero de clase'''
34     num_of_gestos = 7
       '''se junta el modelo con la capa para clasificar'''
36     model = tf.keras.Sequential()
       model.add(pretrained_model_without_top_layer)
38     model.add(tf.keras.layers.Dense(num_of_gestos))

40     opt = keras.optimizers.Adam(learning_rate=0.001)
       model.compile(optimizer="adam", loss=tf.keras.losses.
      SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])
42
       model.summary()
44     return model

46 def cnn_resnet():

48     '''Se obtiene la red preentrenada sin la última capa del repositorio de
       tensorflow'''

50     feature_extractor_model = "https://tfhub.dev/tensorflow/resnet_50/
      feature_vector/1"
       '''Cambiando el parámetro trainable se pueden entrenar casi todos los
      parámetros'''
```

```
    pretrained_model_without_top_layer = hub.KerasLayer(
        feature_extractor_model, input_shape=(224, 224, 3), trainable=False
    )
    '''numero de clase'''
    num_of_gestos = 7
    '''se junta el modelo con la capa para clasificar'''
    model = tf.keras.Sequential()
    model.add(pretrained_model_without_top_layer)
    model.add(tf.keras.layers.Dense(num_of_gestos))

    opt = keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer="adam", loss=tf.keras.losses.
    SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])

    model.summary()
    return model

def cnn_inception():

    '''Se obtiene la red preentrenada sin la última capa del repositorio de
    tensorflow'''

    feature_extractor_model = "https://tfhub.dev/google/tf2-preview/
    inception_v3/feature_vector/4"
    '''Cambiando el parámetro trainable se pueden entrenar casi todos los
    parámetros'''
    pretrained_model_without_top_layer = hub.KerasLayer(
        feature_extractor_model, input_shape=(299, 299, 3), trainable=False
    )
    '''numero de clase'''
    num_of_gestos = 7
    '''se junta el modelo con la capa para clasificar'''
    model = tf.keras.Sequential()
    model.add(pretrained_model_without_top_layer)
    model.add(tf.keras.layers.Dense(num_of_gestos))

    opt = keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer="adam", loss=tf.keras.losses.
    SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])

    model.summary()
    return model

def cnn_b4():
```

```
90      '''Se obtiene la red preentrenada sin la última capa del repositorio de
        tensorflow'''

92      feature_extractor_model = "https://tfhub.dev/tensorflow/efficientnet/b4
        /feature-vector/1"
        '''Cambiando el parámetro trainable se pueden entrenar casi todos los
        parámetros'''
94      pretrained_model_without_top_layer = hub.KerasLayer(
            feature_extractor_model, input_shape=(380, 380, 3), trainable=False
        )
96      '''numero de clase'''
        num_of_gestos = 7
98      '''se junta el modelo con la capa para clasificar'''
        model = tf.keras.Sequential()
100     model.add(pretrained_model_without_top_layer)
        model.add(tf.keras.layers.Dense(num_of_gestos))
102
        opt = keras.optimizers.Adam(learning_rate=0.001)
104     model.compile(optimizer="adam", loss=tf.keras.losses.
        SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])

106     model.summary()
        return model
108
    def cnn_b3():
110
        '''Se obtiene la red preentrenada sin la última capa del repositorio de
        tensorflow'''
112
        feature_extractor_model = "https://tfhub.dev/tensorflow/efficientnet/b3
        /feature-vector/1"
114     '''Cambiando el parámetro trainable se pueden entrenar casi todos los
        parámetros'''
        pretrained_model_without_top_layer = hub.KerasLayer(
116         feature_extractor_model, input_shape=(300, 300, 3), trainable=False
        )
        '''numero de clase'''
118     num_of_gestos = 7
        '''se junta el modelo con la capa para clasificar'''
120     model = tf.keras.Sequential()
        model.add(pretrained_model_without_top_layer)
122     model.add(tf.keras.layers.Dense(num_of_gestos))

124     opt = keras.optimizers.Adam(learning_rate=0.001)
        model.compile(optimizer="adam", loss=tf.keras.losses.
        SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])
```

```
126
    model.summary()
128     return model

130 def cnn_b2():

132     '''Se obtiene la red preentrenada sin la última capa del repositorio de
        tensorflow'''

134     feature_extractor_model = "https://tfhub.dev/tensorflow/efficientnet/b2
        /feature-vector/1"
        '''Cambiando el parámetro trainable se pueden entrenar casi todos los
        parámetros'''
136     pretrained_model_without_top_layer = hub.KerasLayer(
            feature_extractor_model, input_shape=(260, 260, 3), trainable=False
        )
138     '''numero de clase'''
        num_of_gestos = 7
140     '''se junta el modelo con la capa para clasificar'''
        model = tf.keras.Sequential()
142     model.add(pretrained_model_without_top_layer)
        model.add(tf.keras.layers.Dense(num_of_gestos))
144
        opt = keras.optimizers.Adam(learning_rate=0.001)
146     model.compile(optimizer="adam", loss=tf.keras.losses.
        SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])

148     model.summary()
        return model
150
def cnn_b1():
152
        '''Se obtiene la red preentrenada sin la última capa del repositorio de
        tensorflow'''
154
        feature_extractor_model = "https://tfhub.dev/tensorflow/efficientnet/b1
        /feature-vector/1"
156     '''Cambiando el parámetro trainable se pueden entrenar casi todos los
        parámetros'''
        pretrained_model_without_top_layer = hub.KerasLayer(
158         feature_extractor_model, input_shape=(240, 240, 3), trainable=False
        )
        '''numero de clase'''
160     num_of_gestos = 7
        '''se junta el modelo con la capa para clasificar'''
162     model = tf.keras.Sequential()
```

```
    model.add(pretrained_model_without_top_layer)
    model.add(tf.keras.layers.Dense(num_of_gestos))

    opt = keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer="adam", loss=tf.keras.losses.
    SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])

    model.summary()
    return model

def cnn_b0():

    '''Se obtiene la red preentrenada sin la última capa del repositorio de
    tensorflow'''

    feature_extractor_model = "https://tfhub.dev/tensorflow/efficientnet/b0
    /feature-vector/1"
    '''Cambiando el parámetro trainable se pueden entrenar casi todos los
    parámetros'''
    pretrained_model_without_top_layer = hub.KerasLayer(
        feature_extractor_model, input_shape=(224, 224, 3), trainable=False
    )
    '''numero de clase'''
    num_of_gestos = 7
    '''se junta el modelo con la capa para clasificar'''
    model = tf.keras.Sequential()
    model.add(pretrained_model_without_top_layer)
    model.add(tf.keras.layers.Dense(num_of_gestos))

    opt = keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer="adam", loss=tf.keras.losses.
    SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])

    model.summary()
    return model


'''Validación cruzada'''
def validacion_cruzada(funcion_modelo,X_aplanado,y,n_epochs,n_splits):
    '''
    n_epochs: número de epochs
    funcion_modelo: modelo creado para el clasificador
    '''
    scoring = ['precision_macro', 'recall_macro', 'precision_micro', '
    recall_micro', 'f1_macro','accuracy']
```

```
    model = KerasClassifier(build_fn=funcion_modelo, epochs=n_epochs)
    kfold = KFold(n_splits=n_splits, shuffle=True)

    results = cross_validate(model, X_aplanado, y, cv=kfold,scoring=scoring
    )

    print('Media: ', np.mean(results['test_accuracy']))
    return results

#-----------------------------------------------------------------
def CNN_transferlearning(X_aplanado,X_train_scaled,X_test_scaled,y,y_train,
    y_test,n_epochs,n_splits,red_preentrenada):
    if red_preentrenada=='mobilenet':
        model = cnn_mobilenet()
        results = validacion_cruzada(cnn_mobilenet,X_aplanado,y,n_epochs,
    n_splits)
    if red_preentrenada=='resnet':
        model = cnn_resnet()
        results = validacion_cruzada(cnn_resnet,X_aplanado,y,n_epochs,
    n_splits)
    if red_preentrenada=='inception':
        model = cnn_inception()
        results = validacion_cruzada(cnn_inception,X_aplanado,y,n_epochs,
    n_splits)
    if red_preentrenada=='efficientnet':
        model = cnn_b4()
        results = validacion_cruzada(cnn_b4,X_aplanado,y,n_epochs,n_splits)

    model, history = CNN_fit(model,X_train_scaled, y_train, X_test_scaled,
    y_test, n_epochs)
    return model,history,results

'''Entrenamiento'''
def CNN_fit(model,X_train_scaled,y_train,X_test_scaled,y_test,n_epochs):

    history= []
    modelo=model.fit(X_train_scaled, y_train, epochs=n_epochs,
    validation_data=(X_test_scaled,y_test))
    history.append(modelo)
    return model, history


def CNN_transferlearning_efficientnet(X_aplanado,X_train_scaled,
    X_test_scaled,y,y_train,y_test,n_epochs,n_splits,red_preentrenada):
    if red_preentrenada=='b0':
        model = cnn_b0()
```

```
        results = validacion_cruzada(cnn_b0,X_aplanado,y,n_epochs,n_splits)
    if red_preentrenada=='b1':
        model = cnn_b1()
        results = validacion_cruzada(cnn_b1,X_aplanado,y,n_epochs,n_splits)
    if red_preentrenada=='b2':
        model = cnn_b2()
        results = validacion_cruzada(cnn_b2,X_aplanado,y,n_epochs,n_splits)
    if red_preentrenada=='b3':
        model = cnn_b3()
        results = validacion_cruzada(cnn_b3,X_aplanado,y,n_epochs,n_splits)
    if red_preentrenada=='b4':
        model = cnn_b4()
        results = validacion_cruzada(cnn_b4,X_aplanado,y,n_epochs,n_splits)

    model, history = CNN_fit(model,X_train_scaled, y_train, X_test_scaled,
    y_test, n_epochs)
    return model,history,results

def CNN_elegida_entrenamiento(X_train_scaled,X_test_scaled,y,y_train,y_test
    ,n_epochs):
    model = cnn_b1()
    model, history = CNN_fit(model,X_train_scaled, y_train, X_test_scaled,
    y_test, n_epochs)
    return model
```

Código de la red CNN.

16.6. Script efficientnet_completo

```
'''Importación de librerías'''
'''librerías para trabajar con los paths'''
import os
import re
'''librerias para manejar las imagenes'''
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import cross_validate

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

```
import scipy.stats as stats
from statsmodels.stats.multicomp import pairwise_tukeyhsd, MultiComparison

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc, accuracy_score, recall_score,
    precision_score, plot_confusion_matrix, classification_report

import Preprocesado
import CNN_transfer
import Metodos_evaluacion
import escribir_excel

import pandas as pd
from pandas import ExcelWriter
'''CNN MEDIANTE TRANSFER LEARNING'''
def CNN_EFF_FINAL():

    '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
    gestos_images_dict,gestos_labels_dict = Preprocesado.lectura_imagenes(
    carpeta = 'Escritorio/trabajo_aprendizaje/muchasimagenes')

    #
    ----------------------------------------------------------------------------

    '''Se introducen las imagenes y se reescalan y se normalizan.'''

    X_aplanado, X_train_scaled, X_test_scaled,y, y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CNN(gestos_images_dict,
    gestos_labels_dict,224)
    model_b0, history_b0,results_b0= CNN_transfer.
    CNN_transferlearning_efficientnet(X_aplanado, X_train_scaled,
    X_test_scaled,y, y_train, y_test,80,5,red_preentrenada='b0')
    model_b0.evaluate(x=X_test_scaled,y=y_test)
    Metodos_evaluacion.graficas_acc_loss(history_b0, '/home/nunuel/
    Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_b0.jpg')
    Metodos_evaluacion.curva_ROC(model_b0,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_CNN_b0.jpg')
    escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    CNN_results_b0.xlsx','CNN_results',results_b0)


    X_aplanado, X_train_scaled, X_test_scaled,y, y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CNN(gestos_images_dict,
    gestos_labels_dict,240)
    model_b1, history_b1,results_b1= CNN_transfer.
    CNN_transferlearning_efficientnet(X_aplanado, X_train_scaled,
```

```
    X_test_scaled,y, y_train, y_test,80,5,red_preentrenada='b1')
46  model_b1.evaluate(x=X_test_scaled,y=y_test)
    Metodos_evaluacion.graficas_acc_loss(history_b1, '/home/nunuel/
    Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_b1.jpg')
48  Metodos_evaluacion.curva_ROC(model_b1,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_CNN_b1.jpg')
    escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    CNN_results_b1.xlsx','CNN_results',results_b1)
50

52  X_aplanado, X_train_scaled, X_test_scaled,y, y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CNN(gestos_images_dict,
    gestos_labels_dict,260)
    model_b2, history_b2,results_b2= CNN_transfer.
    CNN_transferlearning_efficientnet(X_aplanado, X_train_scaled,
    X_test_scaled,y, y_train, y_test,80,5,red_preentrenada='b2')
54  model_b2.evaluate(x=X_test_scaled,y=y_test)
    Metodos_evaluacion.graficas_acc_loss(history_b2, '/home/nunuel/
    Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_b2.jpg')
56  Metodos_evaluacion.curva_ROC(model_b2,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_CNN_b2.jpg')
    escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    CNN_results_b2.xlsx','CNN_results',results_b2)
58

60  X_aplanado, X_train_scaled, X_test_scaled,y, y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CNN(gestos_images_dict,
    gestos_labels_dict,300)
    model_b3, history_b3,results_b3= CNN_transfer.
    CNN_transferlearning_efficientnet(X_aplanado, X_train_scaled,
    X_test_scaled,y, y_train, y_test,80,5,red_preentrenada='b3')
62  model_b3.evaluate(x=X_test_scaled,y=y_test)
    Metodos_evaluacion.graficas_acc_loss(history_b3, '/home/nunuel/
    Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_b3.jpg')
64  Metodos_evaluacion.curva_ROC(model_b3,X_test_scaled,y_test,'/home/
    nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_CNN_b3.jpg')
    escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
    CNN_results_b3.xlsx','CNN_results',results_b3)
66

    X_aplanado, X_train_scaled, X_test_scaled,y, y_train, y_test =
    Preprocesado.separacion_normalizacion_datos_CNN(gestos_images_dict,
    gestos_labels_dict,380)
68  model_b4, history_b4,results_b4= CNN_transfer.
    CNN_transferlearning_efficientnet(X_aplanado, X_train_scaled,
    X_test_scaled,y, y_train, y_test,80,5,red_preentrenada='b4')
    model_b4.evaluate(x=X_test_scaled,y=y_test)
```

```
70      Metodos_evaluacion.graficas_acc_loss(history_b4, '/home/nunuel/
        Escritorio/trabajo_aprendizaje/curva_acc_loss/acc_loss_b4.jpg')
        Metodos_evaluacion.curva_ROC(model_b4,X_test_scaled,y_test,'/home/
        nunuel/Escritorio/trabajo_aprendizaje/curvas_roc/ROC_CNN_b4.jpg')
72      escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/
        CNN_results_b4.xlsx','CNN_results',results_b4)


74
        #Metodos_evaluacion.predicción_imagenes_aleatorias(model,X_test_scaled,
        y_test)
76      #escribir_excel.excel('Escritorio/trabajo_aprendizaje/excel/CNN_history
        .xlsx','CNN_history',(history[acc])

78      return results_b0,results_b1,results_b2,results_b3,results_b4

80  def hipotesis_eff(scores_b0,scores_b1,scores_b2,scores_b3,scores_b4):
        alpha = 0.05
82      CV = 10
        F_statistic, pVal = stats.kruskal(scores_b0['test_accuracy'],scores_b1[
        'test_accuracy'],scores_b2['test_accuracy'],scores_b3['test_accuracy'],
        scores_b4['test_accuracy'])
84      F_statistic2, pVal2 = stats.f_oneway(scores_b0['test_accuracy'],
        scores_b1['test_accuracy'],scores_b2['test_accuracy'],scores_b3['
        test_accuracy'],scores_b4['test_accuracy'])
        print ('p-valor KrusW:', pVal)
86      print ('p-valor ANOVA:', pVal2)
        if pVal <= alpha:
88          print('Rechazamos la hipótesis: los modelos son diferentes\n')
            stacked_data = np.vstack((scores_b0['test_accuracy'],scores_b1['
        test_accuracy'],scores_b2['test_accuracy'],scores_b3['test_accuracy'],
        scores_b4['test_accuracy'])).ravel()
90          stacked_model = np.vstack((np.repeat('model_b0',CV),np.repeat('
        model_b1',CV),np.repeat('model_b2',CV),np.repeat('model_b3',CV),np.
        repeat('model_b4',CV))).ravel()
            MultiComp = MultiComparison(stacked_data, stacked_model)
92          comp = MultiComp.allpairtest(stats.ttest_rel, method='Holm')
            print (comp[0])
94          print(MultiComp.tukeyhsd(alpha=0.05))
        else:
96          print('Aceptamos la hipótesis: los modelos son iguales')


98
    def main_efficientnet():
100     scores_b0,scores_b1,scores_b2,scores_b3,scores_b4 = CNN_EFF_FINAL()
        test_sc=[scores_b0['test_accuracy'],scores_b1['test_accuracy'],
        scores_b2['test_accuracy'],scores_b3['test_accuracy'],scores_b4['
```

```
    test_accuracy']]

    lab = ['EfficientNet_B0', 'EfficientNet_B1','EfficientNet_B2','
    EfficientNet_B3','EfficientNet_B4']
    fig, ax = plt.subplots()

    ax.boxplot(test_sc,labels=lab)
    ax.set_ylabel('Test_accuracy')
    ax.set_xlabel('Modelos')
    plt.xticks(rotation = 90)

    plt.savefig('/home/nunuel/Escritorio/trabajo_aprendizaje/boxplot/
    boxplot_efficientnet.jpg', bbox_inches='tight')


    df = pd.DataFrame({'Modelo': ['fit_time', 'score_time', '
    test_precision_macro', 'test_recall_macro', 'test_precision_micro','
    test_recall_micro','test_f1_macro','test_accuracy'],
                       'Model_B0': [scores_b0['fit_time'], scores_b0['
    score_time'], scores_b0['test_precision_macro'], scores_b0['
    test_recall_macro'],scores_b0['test_precision_micro'],scores_b0['
    test_recall_micro'],scores_b0['test_f1_macro'],scores_b0['test_accuracy'
    ]],
                       'Model_B1': [scores_b1['fit_time'], scores_b1['
    score_time'], scores_b1['test_precision_macro'], scores_b1['
    test_recall_macro'],scores_b1['test_precision_micro'],scores_b1['
    test_recall_micro'],scores_b1['test_f1_macro'],scores_b1['test_accuracy'
    ]],
                       'Model_B2': [scores_b2['fit_time'], scores_b2['
    score_time'], scores_b2['test_precision_macro'], scores_b2['
    test_recall_macro'],scores_b2['test_precision_micro'],scores_b2['
    test_recall_micro'],scores_b2['test_f1_macro'],scores_b2['test_accuracy'
    ]],
                       'Model_B3': [scores_b3['fit_time'], scores_b3['
    score_time'], scores_b3['test_precision_macro'], scores_b3['
    test_recall_macro'],scores_b3['test_precision_micro'],scores_b3['
    test_recall_micro'],scores_b3['test_f1_macro'],scores_b3['test_accuracy'
    ]],
                       'Model_B4': [scores_b4['fit_time'], scores_b4['
    score_time'], scores_b4['test_precision_macro'], scores_b4['
    test_recall_macro'],scores_b4['test_precision_micro'],scores_b4['
    test_recall_micro'],scores_b4['test_f1_macro'],scores_b4['test_accuracy'
    ]]
                       })
```

```
    df = df[['Modelo', 'Model_B0','Model_B1','Model_B2','Model_B3','
Model_B4']]
    path = os.path.join(os.getcwd(), 'Escritorio/trabajo_aprendizaje/excel/
Metricas_todo_efficientnet.xlsx')
    print(path)
    writer = ExcelWriter(path)
    df.to_excel(writer, 'Hoja de datos', index=False)
    writer.save()

    df = pd.DataFrame({'Modelo': ['fit_time', 'score_time', '
test_precision_macro', 'test_recall_macro', 'test_precision_micro','
test_recall_micro','test_f1_macro','test_accuracy'],
                       'Model_B0': [np.mean(scores_b0['fit_time']), np.mean(
scores_b0['score_time']), np.mean(scores_b0['test_precision_macro']), np
.mean(scores_b0['test_recall_macro']),np.mean(scores_b0['
test_precision_micro']),np.mean(scores_b0['test_recall_micro']),np.mean(
scores_b0['test_f1_macro']),np.mean(scores_b0['test_accuracy'])],
                       'Model_B1': [np.mean(scores_b1['fit_time']), np.mean(
scores_b1['score_time']), np.mean(scores_b1['test_precision_macro']), np
.mean(scores_b1['test_recall_macro']),np.mean(scores_b1['
test_precision_micro']),np.mean(scores_b1['test_recall_micro']),np.mean(
scores_b1['test_f1_macro']),np.mean(scores_b1['test_accuracy'])],
                       'Model_B2': [np.mean(scores_b2['fit_time']), np.mean(
scores_b2['score_time']), np.mean(scores_b2['test_precision_macro']), np
.mean(scores_b2['test_recall_macro']),np.mean(scores_b2['
test_precision_micro']),np.mean(scores_b2['test_recall_micro']),np.mean(
scores_b2['test_f1_macro']),np.mean(scores_b2['test_accuracy'])],
                       'Model_B3': [np.mean(scores_b3['fit_time']), np.mean(
scores_b3['score_time']), np.mean(scores_b3['test_precision_macro']), np
.mean(scores_b3['test_recall_macro']),np.mean(scores_b3['
test_precision_micro']),np.mean(scores_b3['test_recall_micro']),np.mean(
scores_b3['test_f1_macro']),np.mean(scores_b3['test_accuracy'])],
                       'Model_B4': [np.mean(scores_b4['fit_time']), np.mean(
scores_b4['score_time']), np.mean(scores_b4['test_precision_macro']), np
.mean(scores_b4['test_recall_macro']),np.mean(scores_b4['
test_precision_micro']),np.mean(scores_b4['test_recall_micro']),np.mean(
scores_b4['test_f1_macro']),np.mean(scores_b4['test_accuracy'])]
                       })

    df = df[['Modelo', 'Model_B0','Model_B1','Model_B2','Model_B3','
Model_B4']]
    path = os.path.join(os.getcwd(), 'Escritorio/trabajo_aprendizaje/excel/
Metricas_medias_todo_efficientnet.xlsx')
    print(path)
    writer = ExcelWriter(path)
    df.to_excel(writer, 'Hoja de datos', index=False)
```

```
142     writer.save()

144     hipotesis_eff(scores_b0,scores_b1,scores_b2,scores_b3,scores_b4)
```

Código de escritura en excel.

16.7. Script aplicación

```
    '''Importación de librerías'''
2   import os
    import re
4   import cv2
    import matplotlib.pyplot as plt
6   import tkinter
    from tkinter import *
8   from tkinter import filedialog
    import imutils
10  from PIL import Image , ImageTk
    import numpy as np
12  import CNN_transfer
    import Preprocesado
14
    import tensorflow as tf
16  import tensorflow_hub as hub

18  from tensorflow import keras

20  def hacer_foto():
        global imagen
22
        foto = cv2.VideoCapture(0)
24      cv2.waitKey(50)
        leido,frame = foto.read()
26      if leido == True:
            imagen = frame
28
            cv2.imwrite('Escritorio/trabajo_aprendizaje/foto_aplicacion/foto.
       jpg',frame)
30          print('Foto realizada')

32      cv2.waitKey(50)
```

```
34      '''LABEL Foto realizada'''
        label_texto1 = Label(ventana,text= 'Foto realizada:')
36      label_texto1.grid(column=0,row=1,padx=5,pady=5)
        label_texto1.config(font=("Arial", 30))
38      '''Foto a etiquetar'''
        label_foto = Label(ventana)
40      label_foto.grid(column=0,row=2)

42      foto =plt.imread('Escritorio/trabajo_aprendizaje/foto_aplicacion/foto.
        jpg')
        foto = cv2.flip(foto,1)
44

46      '''Redimensionado de la imagen'''
        imagen = imutils.resize(foto,height=800)
48      imagen = imutils.resize(imagen,width=500)

50      '''Se muestra la imagen'''
        imagen_mostrar = Image.fromarray(imagen)
52      img = ImageTk.PhotoImage(image=imagen_mostrar)

54      label_foto.configure(image=img)
        label_foto.image = img
56      prediccion(modelo)

58


60  def carga_modelo():
        '''LECTURA DE IMÁGENES Y GUARDADO DE SUS PATHS EN LISTAS'''
62      gestos_images_dict,gestos_labels_dict = Preprocesado.lectura_imagenes(
        carpeta = 'Escritorio/trabajo_aprendizaje/muchasimagenes')
        X_aplanado, X_train_scaled, X_test_scaled,y, y_train, y_test =
        Preprocesado.separacion_normalizacion_datos_CNN(gestos_images_dict,
        gestos_labels_dict,240)
64      modelo =CNN_transfer.cnn_b1()
        modelo.train_on_batch(X_train_scaled[:1], y_train[:1])
66      modelo.load_weights('Escritorio/trabajo_aprendizaje/modelo/
        modelo_elegido.h5')
        return modelo
68
    def prediccion(modelo):
70      class_names = ['pulgar arriba','uno','dos','tres','cuatro','cinco','
        pulgar abajo']
        image = plt.imread('Escritorio/trabajo_aprendizaje/foto_aplicacion/foto
        .jpg')
72      resized_img = cv2.resize(image,(240,240))
```

```
74      X_pred = np.array(resized_img)
        X_pred = [X_pred/255]
76      X_pred = np.array(X_pred)

78      pred = modelo.predict(X_pred)
        class_pred = np.argmax(pred)
80
        if class_pred == 0:
82          imagen_etiqueta = cv2.imread('Escritorio/trabajo_aprendizaje/
    Imagenes_etiquetas/pulgar_arriba.png')
        elif class_pred == 1:
84          imagen_etiqueta = cv2.imread('Escritorio/trabajo_aprendizaje/
    Imagenes_etiquetas/uno.png')
        elif class_pred == 2:
86          imagen_etiqueta = cv2.imread('Escritorio/trabajo_aprendizaje/
    Imagenes_etiquetas/dos.png')
        elif class_pred == 3:
88          imagen_etiqueta = cv2.imread('Escritorio/trabajo_aprendizaje/
    Imagenes_etiquetas/tres.png')
        elif class_pred == 4:
90          imagen_etiqueta = cv2.imread('Escritorio/trabajo_aprendizaje/
    Imagenes_etiquetas/cuatro.png')
        elif class_pred == 5:
92          imagen_etiqueta = cv2.imread('Escritorio/trabajo_aprendizaje/
    Imagenes_etiquetas/cinco.png')
        elif class_pred == 6:
94          imagen_etiqueta = cv2.imread('Escritorio/trabajo_aprendizaje/
    Imagenes_etiquetas/pulgar_abajo.png')


96
        imagen_etiqueta = imutils.resize(imagen_etiqueta,height=925)
98      imagen_etiqueta = imutils.resize(imagen_etiqueta,width=625)

100     label_texto2 = Label(ventana,text= 'El gesto de la imagen es:')
        label_texto2.grid(column=5,row=1,padx=5,pady=5)
102     label_texto2.config(font=("Arial", 30))

104     label_foto2 = Label(ventana)
        label_foto2.grid(column=5,row=2)
106
        imagen_etiqueta = imagen_etiqueta
108     print(imagen_etiqueta)
        '''Se muestra la imagen'''
110     imagen_mostrar2 = Image.fromarray(imagen_etiqueta)
        img_etiqueta = ImageTk.PhotoImage(image=imagen_mostrar2)
```

```
112
    label_foto2.configure(image=img_etiqueta)
114     label_foto2.image = img_etiqueta

116 modelo = carga_modelo()

118 '''Se crea la ventana'''
ventana = Tk()
120 imagen = None
'''Botón para hacer foto'''
122 boton = Button(ventana,text = 'Hacer foto',width = 25,command=hacer_foto)
boton.grid(column=0,row=0,padx=5,pady=5)
124
'''Label para mostrar predicción'''
126 label_prediccion = Label(ventana)
label_prediccion.grid(column=1,row=1)
128
ventana.mainloop()
```

Código de escritura en excel.

16.8. Script Metodos_evaluacion

```
1 '''Importación de librerías'''
'''librerías para trabajar con los paths'''
3 import os
import re
5 '''librerias para manejar las imagenes'''
import numpy as np
7 import random
import matplotlib.pyplot as plt
9
import tensorflow as tf
11 ''' se importa el repositorio de tensorflow para la red preentrenada'''
import tensorflow_hub as hub
13
from tensorflow import keras
15 from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
17
import scipy.stats as stats
19 from statsmodels.stats.multicomp import pairwise_tukeyhsd, MultiComparison
```

```
from sklearn.model_selection import cross_validate

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler


from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc, accuracy_score, recall_score, precision_score, plot_confusion_matrix, classification_report

from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


import pandas as pd
from pandas import ExcelWriter
#-----------------------------------------------------------------
def CNN_transferlearning(X_aplanado,X_train_scaled,X_test_scaled,y,y_train,y_test):
    '''CNN'''
    '''Creación del modelo'''
    feature_extractor_model = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"

    pretrained_model_without_top_layer = hub.KerasLayer(
        feature_extractor_model, input_shape=(224, 224, 3), trainable=False)

    def cnn():
        num_of_gestos = 7
        model = tf.keras.Sequential()
        model.add(pretrained_model_without_top_layer)
        model.add(tf.keras.layers.Dense(num_of_gestos))

        opt = keras.optimizers.Adam(learning_rate=0.001)
        model.compile(optimizer="adam", loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])
        return model
    num_of_gestos = 7
    model=cnn()
    model.summary()

    '''Validación cruzada'''
    scoring = ['precision_macro', 'recall_macro', 'precision_micro', 'recall_micro', 'f1_macro','accuracy']
```

```
    CV = 10

    model = KerasClassifier(build_fn=cnn, epochs=50)
    kfold = KFold(n_splits=5, shuffle=True)

    results = cross_val_score(model, X_aplanado, y, cv=kfold)


    '''Entrenamiento'''
    history= []
    modelo=model.fit(X_train_scaled, y_train, epochs=50,validation_data=(
    X_test_scaled,y_test))
    history.append(modelo)

    '''Gráficas de exactitud y error'''

def graficas_acc_loss(history,path):
    plt.figure(figsize=(20,7))

    plt.subplot(1, 2, 1)
    plt.plot(history[0].history['acc'], label='Entrenamiento')
    plt.plot(history[0].history['val_acc'], label = 'Validación')
    plt.xlabel('Iteración (epoch)')
    plt.ylabel('Exactitud (accuracy)')
    plt.ylim([0, 1])
    plt.grid()
    plt.title('Modelo 1')
    plt.legend(loc='lower right')

    plt.subplot(1, 2, 2)
    plt.plot(history[0].history['loss'], label='Entrenamiento')
    plt.plot(history[0].history['val_loss'], label = 'Validación')
    plt.xlabel('Iteración (epoch)')
    plt.ylabel('Error (loss)')
    plt.ylim([0, 1])
    plt.grid()
    plt.title('Modelo 1')
    plt.legend(loc='lower right')

    plt.savefig(path)

def predicción_imagenes_aleatorias(model, X_test_scaled, y_test):
    '''Prueba con unas imágenes aleatorias: Si la etiqueta es correcta se
   ve en negro y si es incorrecta en rojo'''
    class_names = ['pulgar arriba','uno','dos','tres','cuatro','cinco','
   pulgar abajo']
```

```
103     plt.figure(figsize=(12,12))
        for i in range(25):
105         index = random.randint(0, X_test_scaled.shape[0]) # Se elige un nú
    mero de imagen al azar
            image = X_test_scaled[index:index+1]
107
            plt.subplot(5, 5, i+1)
109         plt.imshow(image[0])        # Se muestra la imagen elegida al azar
            plt.axis('off')
111
            pred = model.predict(image)  # Se obtiene la predicción del modelo
    para la imagen elegida
113         class_pred = np.argmax(pred) # Se obtiene la clase para la imagen
    elegida

115
            if y_test[index] == class_pred:   # Si hay acierto en la clase
    predicha: se muestra en el título solo el nombre de clase
117             plt.title(class_names[class_pred])
            else:                                   # Si hay un error en la
    clasificación: se muestra en rojo en el título ambas clases
119             plt.title(class_names[class_pred], color='#ff0000')

121
def curva_ROC(model,X_test_scaled,y_test,path):
123     '''Realización de la curva ROC'''
        pred = model.predict(X_test_scaled)
125     class_pred=[]
        for i in range(len(X_test_scaled)):
127         class_pred.append(np.argmax(pred[i]))

129     n_classes =  len(np.unique(y_test))                              #
        Calcula el número de clases del problema
        t_test_bin = label_binarize(y_test, classes=np.arange(0,n_classes,1)) #
        Recodifica las etiquetas de clase en valores binarios
131     print(t_test_bin)

133     y_score = model.predict_proba(X_test_scaled)

135     fpr_micro, tpr_micro, _ = roc_curve(t_test_bin.ravel(), y_score.ravel()
    )
        roc_auc_micro = auc(fpr_micro, tpr_micro)
137
        plt.figure(figsize=(10, 8))
139     colors = ['aqua','blue','violet','gold','orange','pink','tan','purple',
    'lime','red']
```

```
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(t_test_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
        plt.plot(fpr[i], tpr[i], color=colors[i], lw=1, label='ROC clase %i (area = %0.3f)' %(i,roc_auc[i]))

    plt.plot(fpr_micro, tpr_micro, color='red', lw=2, linestyle=':', label='Curva ROC micro-average (AUC = %0.3f)' %roc_auc_micro)
    plt.plot([0, 1], [0, 1], color='k', lw=2, linestyle='--')
    plt.xlabel('Tasa de Falsos Positivos')
    plt.ylabel('Tasa de Verdaderos Positivos')
    plt.title('Curva ROC por clase')
    plt.legend(loc="lower right");


    plt.savefig(path)

def matriz_confusion(modelo,X_test,t_test,path):

    disp = plot_confusion_matrix(modelo, X_test, t_test) # Muestra gráficamente la matriz de confusión
    disp.figure_.suptitle("Matriz de confusión");         # Añade un título a la figura de la matriz de confusión
    disp.figure_.set_dpi(100)                             # Establece el tamaño de la figura
    plt.xlabel("Clase predicha")
    plt.ylabel("Clase real");
    plt.savefig(path)
    #print(f"Matriz de confusión:\n{disp.confusion_matrix}")


def metricas_evaluacion(t_test,y_pred):

    print("Exactitud     : %.2f %%" %(100*accuracy_score(t_test, y_pred)))
    print("Sensibilidad  : %.2f %%" %(100*recall_score(t_test, y_pred, average='macro')))
    print("Precisión     : %.2f %%" %(100*precision_score(t_test, y_pred, average='macro')))
    metricas = classification_report(t_test, y_pred)
    print("Informe de evaluación del clasificador sobre el conjunto de test:\n", metricas)

    return metricas
```

```
179 def comparacion_modelos(scoresLR_canny, scoresLDA_canny, scoresKNN_canny,
    scoresDT_canny, scoresRF_canny, scoresSVM_canny, scoresLR_HOG, scoresLDA_HOG
    , scoresKNN_HOG, scoresDT_HOG, scoresRF_HOG, scoresSVM_HOG):
    data = [scoresLR_canny['test_accuracy'], scoresLDA_canny['test_accuracy'
    ], scoresKNN_canny['test_accuracy'], scoresDT_canny['test_accuracy'],
    scoresRF_canny['test_accuracy'], scoresSVM_canny['test_accuracy'],
    scoresLR_HOG['test_accuracy'], scoresLDA_HOG['test_accuracy'],
    scoresKNN_HOG['test_accuracy'], scoresDT_HOG['test_accuracy'],
    scoresRF_HOG['test_accuracy'], scoresSVM_HOG['test_accuracy']]
181     fig7, ax = plt.subplots()
    ax.set_title('Modelos')
183     ax.boxplot(data, labels=['LR_canny', 'LDA_canny', 'KNN_canny', 'DT_canny', '
    RF_canny', 'SVM_canny', 'LR_HOG', 'LDA_HOG', 'KNN_HOG', 'DT_HOG', 'RF_HOG', '
    SVM_HOG']);

185 def hipotesis(scoresLR_canny, scoresLDA_canny, scoresKNN_canny, scoresDT_canny
    , scoresRF_canny, scoresSVM_canny, scoresLR_HOG, scoresLDA_HOG, scoresKNN_HOG
    , scoresDT_HOG, scoresRF_HOG, scoresSVM_HOG):
    alpha = 0.05
187     CV = 10
    F_statistic, pVal = stats.kruskal(scoresLR_canny['test_accuracy'],
    scoresLDA_canny['test_accuracy'], scoresKNN_canny['test_accuracy'],
    scoresDT_canny['test_accuracy'], scoresRF_canny['test_accuracy'],
    scoresSVM_canny['test_accuracy'], scoresLR_HOG['test_accuracy'],
    scoresLDA_HOG['test_accuracy'], scoresKNN_HOG['test_accuracy'],
    scoresDT_HOG['test_accuracy'], scoresRF_HOG['test_accuracy'],
    scoresSVM_HOG['test_accuracy'])
189     F_statistic2, pVal2 = stats.f_oneway(scoresLR_canny['test_accuracy'],
    scoresLDA_canny['test_accuracy'], scoresKNN_canny['test_accuracy'],
    scoresDT_canny['test_accuracy'], scoresRF_canny['test_accuracy'],
    scoresSVM_canny['test_accuracy'], scoresLR_HOG['test_accuracy'],
    scoresLDA_HOG['test_accuracy'], scoresKNN_HOG['test_accuracy'],
    scoresDT_HOG['test_accuracy'], scoresRF_HOG['test_accuracy'],
    scoresSVM_HOG['test_accuracy'])
    print('p-valor KrusW:', pVal)
191     print('p-valor ANOVA:', pVal2)
    if pVal <= alpha:
193         print('Rechazamos la hipótesis: los modelos son diferentes\n')
            stacked_data = np.vstack((scoresLR_canny['test_accuracy'],
    scoresLDA_canny['test_accuracy'], scoresKNN_canny['test_accuracy'],
    scoresDT_canny['test_accuracy'], scoresRF_canny['test_accuracy'],
    scoresSVM_canny['test_accuracy'], scoresLR_HOG['test_accuracy'],
    scoresLDA_HOG['test_accuracy'], scoresKNN_HOG['test_accuracy'],
    scoresDT_HOG['test_accuracy'], scoresRF_HOG['test_accuracy'],
    scoresSVM_HOG['test_accuracy'])).ravel()
```

```
195         stacked_model = np.vstack((np.repeat('modelLR_canny',CV),np.repeat(
    'modelLDA_canny',CV),np.repeat('modelKNN_canny',CV),np.repeat('
    modelDT_canny',CV),np.repeat('modelRF_canny',CV),np.repeat('
    modelSVM_canny',CV),np.repeat('modelLR_HOG',CV),np.repeat('modelLDA_HOG'
    ,CV),np.repeat('modelKNN_HOG',CV),np.repeat('modelDT_HOG',CV),np.repeat(
    'modelRF_HOG',CV),np.repeat('modelSVM_HOG',CV))).ravel()
            MultiComp = MultiComparison(stacked_data, stacked_model)
197         comp = MultiComp.allpairtest(stats.ttest_rel, method='Holm')
            print (comp[0])
199         print(MultiComp.tukeyhsd(alpha=0.05))
    else:
201         print('Aceptamos la hipótesis: los modelos son iguales')
```

Código de escritura en excel.

16.9. Script escribir_excel

```
1 '''Importación de librerías'''
  '''librerías para trabajar con los paths'''
3 import os
  import re
5
  import pandas as pd
7 from pandas import ExcelWriter


9


11 def excel(path,hoja,diccionario):
      df = pd.DataFrame(diccionario)
13
      path = os.path.join(os.getcwd(), path)
15    print(path)
      writer = ExcelWriter(path)
17    df.to_excel(writer, hoja, index=False)
      writer.save()
```

Código de escritura en excel.