Modelización de problemas UCM

Management Solutions

Detección de emociones a partir de imágenes faciales

Equipo: **You Only Lose Once**

Integrantes: **Alberto Maurel Serrano**

Eduardo Rivero Rodríguez

Pablo Villalobos Sánchez

1. Introducción

Somos un equipo de 5º del Doble Grado en Ingeniería Informática y Matemáticas de la UCM. A pesar de que estamos familiarizados con Jupyter y hemos trabajado previamente temas de IA, no teníamos experiencia en el uso de CNN.

Este dataset apareció hace una competición en Kaggle hace 8 años, llamada *Challenges in Representation Learning: Facial Expression Recognition Challenge* [15]. El objetivo, al igual que en este caso, era reconocer las emociones presentes en una serie de fotografías de rostros.

Queremos empezar el proyecto exponiendo cuál ha sido nuestro pipeline de trabajo:

1. Investigación acerca del problema. Hemos buscado información acerca del dataset y mirado implementaciones de otras personas para resolver este problema. Estas implementaciones serán nuestra primera aproximación, nuestro baseline.

2. Buscar papers que resuelvan este problema e introducir las mejoras que nos han parecido más prometedoras (y que estuvieran a nuestro alcance).

3. Ajustar la arquitectura de estos modelos y los hiperparámetros, creando ya unos modelos propios que mejoren el rendimiento inicial.

4. Tratar de comprender qué estaba ocurriendo e implementar nuevas ideas

2. Fundamento teórico

Preprocesado

En primer lugar tenemos que preprocesar los datos. Esto lo hacemos por varios motivos:

1. Comprender los datos

Es importante intentar entender nuestro modelo, y ver por qué se está equivocando. No es lo mismo que nuestro modelo se confunda entre dos clases entre las que las expresiones son claramente distintas y que lo haga entre dos clases muy similares. En el primer caso, nuestra red no estaría aprendiendo demasiado, mientras que en el segundo sí que lo está haciendo, y lo que necesitamos es refinarla.

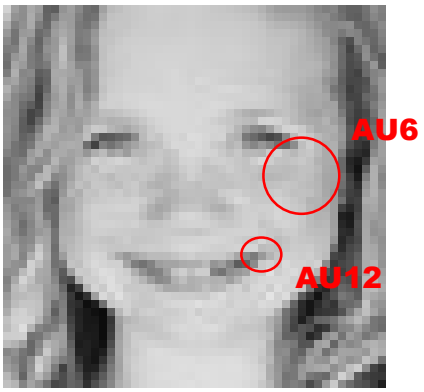
Para ello, nosotros hemos utilizado como referencia el trabajo de Paul Ekman. Es un psicólogo ampliamente reconocido por su estudio de las emociones y la expresión facial. Adoptó un sistema llamado Sistema de Codificación Facial. Los movimientos de la cara se codifican mediante unidades de acción (AUs), que agrupan movimientos de músculos. La combinación de ellas da lugar a la expresión. La siguiente tabla recoge la relación entre las emociones y las AUs involucradas[1]:

Emoción	AUs involucradas	Significado
Felicidad	AU6	Levantamiento de mejillas
	AU12	Ascenso comisuras labios
Sorpresa	AU1	Levantamiento interior de ceja
	AU2	Levantamiento exterior de ceja
	AU5	Levantamiento del párpado superior
	AU26	Caída de la mandíbula
Tristeza	AU1	Levantamiento interior de ceja
	AU4	Bajar cejas
	AU15	Depresión labial esquinal
	AU17	Levantamiento de barbilla
Miedo	AU1	Levantamiento interior de ceja
	AU2	Levantamiento exterior de ceja
	AU4	Juntar cejas
	AU5	Levantamiento del párpado superior
	AU7	Apretar parpado(s)
	AU20	Apretar los labios
	AU26	Caída de la mandíbula
Enfado	AU4	Juntar cejas
	AU5	Levantamiento del párpado superior
	AU7	Apretar parpado(s)
	AU29	Tracción de la mandíbula
Asco	AU9	Arrugar la nariz
	AU10	Levantamiento del labio superior
	AU15	Depresión labial esquinal
	AU16	Depresión labial frontal

[1] Obtenido de [1] y [2]

Hay que tener en cuenta que una persona con un cierto sentimiento no tiene por qué mostrar todos los rasgos, pero sí que mostrará la mayoría. Y de forma más visual, colocamos un diagrama y una imagen del dataset con los rasgos identificados:

Felicidad:



train/happy/im13.pn

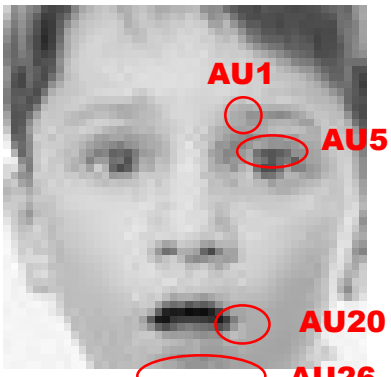
Sorpresa

train/surprised/im14.

Tristeza

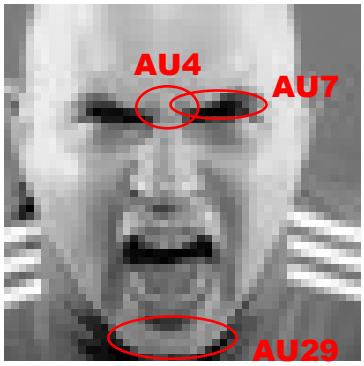
train/sad/im7.png

Miedo



train/fearful/im91.pn

Enfado



train/angry/im3.png

Asco

train/disgusted/im22.png

Neutral

En este caso, se caracterizan por la ausencia de expresiones marcadas:

train/neutral/im84.png

En base a esto, hemos construido esta tabla con lo probable que consideramos nosotros que se puedan confundir dos expresiones:

	Enfado	Asco	Miedo	Felicidad	Neutral	Tristeza	Sorpresa
Enfado							
Asco							
Miedo							
Felicidad							
Neutral							
Tristeza							
Sorpresa							

Esta tabla nos permitirá entender la matriz de confusión el comportamiento del modelo (poco similares, algo similares, muy similares).

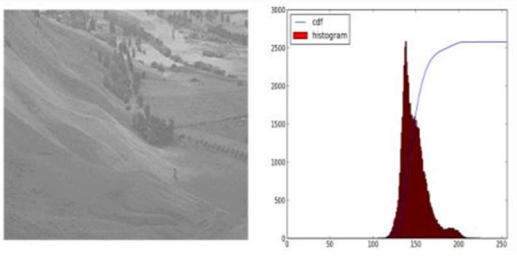
2. Localizar posibles problemas que puede tener nuestro modelo

Luminosidad

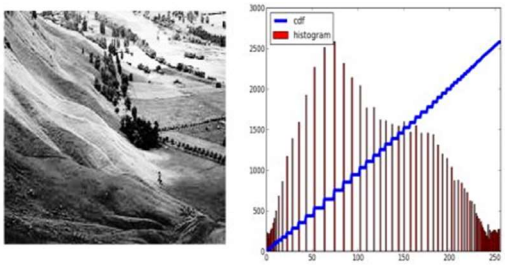
Las siguientes imágenes expresan tristeza:

Sin embargo, es altamente probable que nuestra red, al quedarse con el máximo/promedio en las capas de agrupación, quiera juntar las fotos con menor luminosidad en una clase y las de mayor luminosidad en otra. Incluso, es difícil percibir los rasgos en las fotos excesivamente oscuras.

Para lidiar con esto, podemos emplear una técnica conocida como *Histogram Equalization*. Esta técnica incrementa el contraste de la imagen, especialmente cuando hay poco contraste en la misma. Esto se hace tomando el histograma de una imagen y obteniendo uno con una distribución uniforme[2].



Antes de aplicar Histogram equalization



Después de aplicar Histogram equalization

Un segundo método sería CLAHE (Contrast Limited Adaptive Histogram Equalization). En la técnica anterior considerábamos el contraste de toda la imagen. Sin embargo, esto puede conducir a resultados artificiales, y que a cambio de ganar contraste en una parte lo perdamos

[2] Ejemplos y definición extraído de [3]

en otra. Por ello, vamos a adoptar la técnica anterior pero solo en pequeños fragmentos de la imagen. De esta forma, aumentamos el contraste localmente.

Ángulos

Las imágenes además no son solo de caras rectas de frente, sino que tenemos caras torcidas y algunas fotos de lado:

Algunas imágenes de lado

Y torcidas

Esto es más complicado de solucionar, especialmente con las imágenes laterales. Para agregar más robustez a las imágenes torcidas se puede añadir además de cada foto, su imagen especular:

Esto además no solo introduce más robustez a las caras torcidas, sino que también proporciona más ejemplos de entrenamiento.

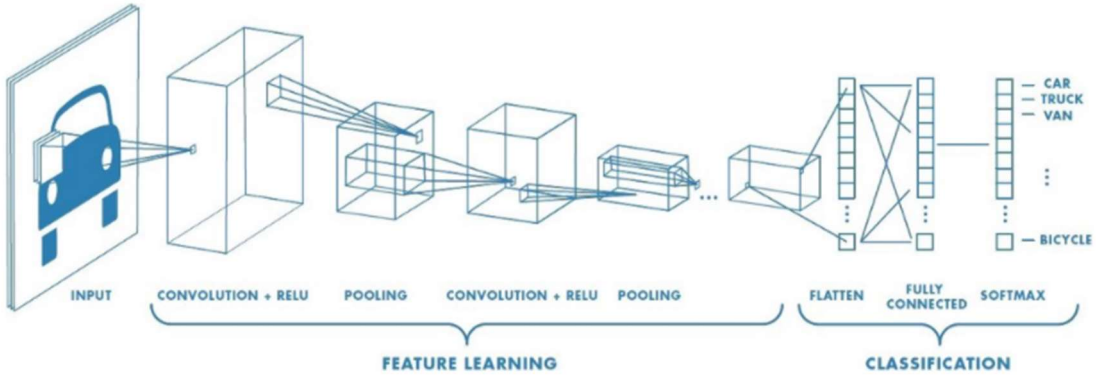
3. Eliminar los datos corruptos

Dado que el dataset tiene un tamaño comedido, podemos inspeccionarlo rápidamente en busca de fotografías corruptas, y que solo vayan a confundir al modelo entrenado. Por ejemplo:

son imágenes presentes en el dataset y que queremos eliminar. Hemos visto que en el conjunto de test también hay imágenes corruptas, pero no las hemos eliminado porque si no estaríamos probando el modelo sobre un dataset distinto al de los otros equipos, influyendo esto en la tasa de acierto final.

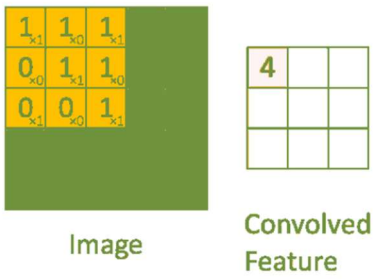
Procesado de la imagen

El procesado de la imagen vamos a realizarlo empleando una red neuronal convolucional (CNN por sus siglas en inglés). Las redes convolucionales se valen de dos operaciones (convoluciones y agrupaciones) para transformar los píxeles en un vector que recoge las principales "características" de la imagen (a estas se las llama *features*). Al vector obtenido nos referiremos a lo largo del trabajo como vectorización o *featurización* de la imagen. Posteriormente, esta vectorización pasa por una serie de capas densas de neuronas (en las que todas las neuronas están conectadas con las neuronas de las capas adyacentes). Con los ejemplos del conjunto de entrenamiento se establecen pesos en las neuronas para predecir en la última capa a qué clase pertenece la imagen[3].



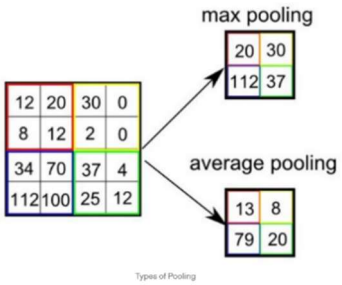
En primer lugar, explicaremos un poco más en detalle el funcionamiento de las capas:

- **Capas convolucionales:**
  Toman una porción de la fotografía y la combinan mediante una operación llamada convolución. Cada píxel tiene un parámetro, el peso del píxel, que es los que aprenderemos en estas capas. En la convolución, se multiplica el valor de cada pixel por el parámetro aprendido, y se suman todos estos valores para dar lugar al valor de la siguiente capa.



  De las capas convolucionales nos centraremos en dos parámetros: el tamaño de la convolución (dimensión del área sobre la que aplicamos la operación) y el número de filtros, que es el número de veces distintas que se aplica la operación sobre la misma zona, dando lugar a "subcapas" distintas.

- **Capas de agrupación:**
  Produce una métrica agregada del área seleccionada. Hay de 2 clases: *MaxPool*, que se queda con el máximo valor de la porción, y *AveragePool*, que se queda con la media de la región.



  Mientras que en las capas convolucionales empleábamos una ventana deslizante, completando con 0 a los lados para que no

[3] Las imágenes de esta página, junto a gran parte de la explicación proceden de [5], un genial artículo en el que se explica bastante en detalle el funcionamiento de las CNN. Aquí simplemente haremos una breve introducción a las CNN pero sin entrar en detalles.

disminuyese el tamaño de la fotografía, aquí de cada región se saca una métrica agregada, disminuyendo el tamaño de la imagen y condensando la información.

- **Capa de flattening:**
  Las capas anteriores son capas tridimensionales (capas bidimensionales y tenemos varias copias de cada una, las "subcapas" que mencionábamos en las capas convolucionales). Esta capa simplemente transforma la salida tridimensional en un vector con las mismas componentes.

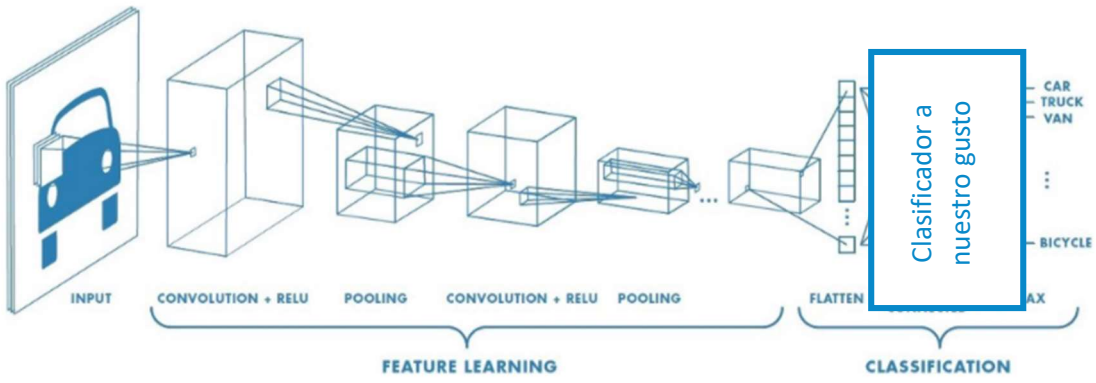
- **Capa densa:**
  Todas las neuronas de esta capa están conectadas con todas las neuronas de las capas previa y posterior. Esto nos permite darle pesos a cada una de las conexiones, aprendidos durante el entrenamiento, para predecir la clase a la que pertenece nuestra imagen.

Una vez que tenemos una intuición sobre cómo funciona la CNN, vemos en la imagen que la red se divide en 2 partes: *feature learning* y *classification*:

- ***Feature learning* (Aprendizaje de características):**
  Se encarga de "aprender" cuáles son las principales características (agrupaciones de píxeles con un determinado patrón) de las imágenes. Comprende las capas convolucionales y las de agrupación.
- ***Classification* (Clasificación):**
  Una vez que tenemos una representación compacta de nuestra imagen, esta parte se encarga de predecir a qué clase pertenece basándose en la representación.

Sin embargo, una vez que tenemos una CNN entrenada, no solo podemos usarla al completo para predecir la clase a la que pertenece la imagen. También podemos extraer las salidas de las capas intermedias. La idea de la parte de aprendizaje de características es ir poco a poco condensando la información de la imagen. Capas más cercanas al comienzo captan los detalles, mientras que las últimas capas concentran los rasgos más importantes de la imagen.

Por ello, tras encontrar la mejor arquitectura para nuestra CNN, vamos a experimentar con diversos clasificadores, que trabajen sobre la vectorización de las imágenes. Para ello, nos quedaremos con la parte de aprendizaje de características y sustituiremos el clasificador por otro que funcione mejor. Es decir:



Clasificadores

En concreto, vamos a probar con 4 tipos de clasificadores distintos:

Softmax

Este es el clasificador propio de la CNN, del que ya hablamos antes. Toma como entrada la vectorización y las capas densas se encargan de comprimirla, combinando la entrada de las capas anteriores ponderadas por los parámetros aprendidos hasta llegar a un único vector con 7 componentes (cada una representando una emoción) con un valor entre 0 y 1. La emoción predicha será la de la componente con un valor mayor.

SVM

Viendo la vectorización de nuestra imagen como un punto en el espacio, se intenta separar las distintas clases mediante hiperplanos. Esto muchas veces no es posible, así que la solución pasa por aumentar la dimensión de los puntos, de forma que pasen a ser casi separables. Como hiperplanos se toman aquellos que maximicen la distancia mínima a los puntos de cada clase. Se tolera además que ciertos puntos no estén correctamente clasificados, y la cantidad de estos puntos depende de hiperparámetros.

Con un hiperplano podemos separar dos clases, así que para generalizarlo a varias clases tenemos que emplear variantes, que hacen uso de estas mismas ideas, pero comparando cada clase con las restantes.

Random Forest

Cogiendo muestras aleatorias del conjunto de entrenamiento, se construyen múltiples árboles de decisión (de ahí el concepto de bosque), que para ese conjunto clasifican las diferentes muestras. Cuando queremos hacer una predicción, se introduce la imagen y cada uno de los árboles hace una predicción. Después, los árboles “votan”, y se escoge como predicción aquella que ha sido elegida por más árboles.

K - nearest neightbours

Viendo de nuevo cada uno de los vectores con las features de las imágenes como un punto en un espacio n-dimensional, tomando una métrica podemos determinar cuáles son los puntos más cercanos. Este algoritmo no construye un modelo, sino que dado un punto (imagen vectorizada) del conjunto de prueba, mira cuáles son las k más cercanas en el conjunto de entrenamiento, y toma como predicción la clase a la que más puntos pertenecen de esos k.

Red neuronal convolucional preentrenada

Hemos hablado anteriormente de usar la red convolucional únicamente para extraer las características más importantes de la imagen, para posteriormente usar un clasificador diferente.

Existen múltiples redes neuronales preentrenadas, cuyo objetivo es sacar de las imágenes las características más representativas posibles. Aunque no sean redes entrenadas sobre nuestro dataset, ni cuyo objetivo inicial fuese extraer las emociones de las caras, pueden funcionar mejor que una red creada ad hoc. El motivo es que se han entrenado sobre conjuntos más grandes de datos y disponen de muchos más parámetros. Por ello, también exploraremos la posibilidad de usar una red preentrenada para sacar las características de las imágenes y después un clasificador.

Detección de las caras

Por último, para la detección de las caras vamos a utilizar un clasificador en cascada. Estos clasificadores se basan en la idea de que muchos clasificadores que son malos en general pueden construir un clasificador robusto, si nos centramos en detectar en qué situación cada uno de los clasificadores suele acertar.

Básicamente, vamos a extraer muchas *features* de nuestra foto, que serán simplemente porciones de nuestra imagen, en las cuáles sumamos una parte de los valores de los píxeles y otra la restamos. Pero la mayoría de estas *features* no contendrán información relevante. Con una cantidad grande de ejemplos, vemos cuáles se equivocan menos, y cuáles predicen mejor en grupo.

A la hora de trabajar con una imagen, aplicaremos primero unos pocos clasificadores, los que mejor funcionan. En caso de que estos no puedan dar una respuesta, se encargan un grupo mayor de clasificadores. Así hasta que consigamos detectar con certeza si hay o no una cara en una región de la imagen [6].

3. Modelado de los datos

De esta fase en adelante vamos a considerar 2 itinerarios: construir una CNN desde 0 y ajustar el clasificador o tomar una CNN preentrenada y ajustar el clasificador.

Itinerario 1: CNN desde 0

Como propuesta inicial, hemos usado el tutorial de Keras: *Facial Expression Classification Keras*[4]. Esta será la base de nuestro código, y la arquitectura en él obtiene un 56,14% de precisión. Se corresponde con la arquitectura B del punto siguiente.

CNN

Una vez decidido que queremos utilizar una red convolucional, tenemos que decidir su arquitectura. Las arquitecturas que hemos planteado son las siguientes:

A	B	C	D	E	F	G
Input						
Conv3-32	Conv3-32 Conv3-32 Conv3-32	Conv3-32 Conv3-32	Conv3-32 Conv3-32	Conv3-32 Conv3-32	Conv3-32 Conv3-32 Conv3-32	Conv3-32 Conv3-32
Maxpool	Maxpool	Maxpool	Averagepool	Maxpool	Averagepool	Averagepool
Conv3-64	Conv3-64 Conv3-64 Conv3-64	Conv3-64 Conv3-64 Conv3-64	Conv3-64 Conv3-64 Conv3-64	Conv3-64 Conv3-64 Conv3-64	Conv3-64 Conv3-64 Conv3-64	Conv3-64 Conv3-64 Conv3-64
Maxpool	Maxpool	Maxpool	Averagepool	Maxpool	Averagepool	Averagepool
Conv3-128	Conv3-128 Conv3-128 Conv3-128	Conv3-128 Conv3-128 Conv3-128 Conv3-128	Conv3-128 Conv3-128 Conv3-128 Conv3-128	Conv3-128 Conv3-128 Conv3-128 Conv3-128	Conv3-128 Conv3-128 Conv3-128	Conv3-128 Conv3-128 Conv3-128 Conv3-128
Maxpool	Maxpool	Maxpool	Averagepool	Averagepool	Averagepool	Averagepool
Flatten						
FC-64						
FC-64						
FC-7	FC-7	FC-7	FC-7	FC-7	FC-7 L2 (1e-4)	FC-7 L2 (1e-4)
softmax	softmax	softmax	softmax	softmax	linear	linear

	A	B	C	D	E	F	G
Número de parámetros	392.263	779.783	918.119	918.119	918.119	779.783	918.119
Número de capas	11	17	17	17	17	17	17

En primer lugar, todas las arquitecturas van a tener 3 grupos de capas convolucionales, con 32, 64 y 128 filtros respectivamente. Además, la convolución se hará sobre grupos de 3x3 píxeles

[4] Disponible en: https://www.kaggle.com/ashishpatel26/tutorial-facial-expression-classification-keras

Haciendo pruebas con más grupos de capas aumentamos sustancialmente la cantidad de parámetros sin obtener una ganancia, por lo que mantendremos en todos los modelos 3 grupos.

Las capas convolucionales utilizarán además la opción padding = "same", que mantiene el tamaño de las capas agregando 0 en las filas de los extremos para evitar disminuir la dimensión. Emplearemos además como función de activación la función ReLu.

Lo único que nos queda discutir de las capas convolucionales es por qué hemos colocado varias capas convolucionales sucesivas, en vez de emplear la arquitectura típica de bloques con 1 capa convolucional seguida por una capa de agregación. Esta idea se presenta en el paper *Very deep convolutional networks for large-scale image recognition* [7]. Colocar 2 capas convolucionales seguidas con área 3x3 hace que en realidad actúe de forma similar a una única capa de dimensión 5x5. Sin embargo, tiene 2 efectos positivos frente a usar directamente una capa 5x5:

1. Reduce la cantidad de parámetros necesaria.
2. Incrementa la no linearidad introducida por la función de rectificación, puesto que en caso de tener una única capa tendríamos solo una rectificación, y al tener 2 capas tenemos 2 funciones de rectificación.

Por último, siguiendo también las ideas de este paper, probamos agrupando más capas al final de la red en vez de al principio.

Entre distintas arquitecturas, también probaremos a modificar las capas de agrupación. Generalmente, las capas de MaxPooling funcionan mejor que las de AveragePooling, pero nosotros en los experimentos vimos que para este caso particular las capas de AveragePooling lograban mejores resultados, por lo que también experimentamos con ellas.

Tras la fase de extracción tendremos la de clasificación. En todos los modelos tendremos dos capas densas con 64 neuronas, y luego una capa densa con 7, empleando regularización en las 2 últimas arquitecturas. Por último, la activación será softmax en las 5 primeras arquitecturas y linear en las 2 últimas.

Todas ellas las compilaremos con el optimizador 'Adam'. Para las 5 primeras emplearemos como métrica 'categorical_crossentropy', y en las dos últimas 'squared_hinge'.

Cada arquitectura es el producto del refinamiento de la anterior, y llevamos en paralelo la realización de los puntos 3 y 4 (modelado y resultados).

Clasificador

Tras obtener una representación vectorial de una imagen, tenemos que asignarle a esa representación una categoría. Vamos a probar con los 4 tipos de clasificadores mencionados anteriormente:

Softmax

El Softmax es el clasificador que hemos utilizado en la parte de clasificación de nuestra CNN, por lo que no tendremos que hacer ninguna modificación.

SVM

Los principales parámetros son:

- **Kernel**: tipo de kernel usado en la SVM, que se encarga de calcular lo similares que son dos puntos. El más utilizado es el "rbf" o kernel gaussiano, que es el que emplearemos nosotros.
- **C**: parámetro de regularización. Cuanto mayor sea el parámetro C, menos fuerte será la regularización. Se corresponde a la penalización L2 al cuadrado.
- **Gamma**: coeficiente del kernel.

Random Forest

Nos centraremos en los siguientes parámetros:

- **N_estimators**: número de árboles de decisión que votarán.
- **Criterion**: función que mide la calidad de cada paso. Con *gini* medimos la impureza y con *entropy* la ganancia de información.
- **Max_depth**: profundidad máxima de los árboles de decisión. Una profundidad excesiva lleva al sobreaprendizaje.
- **Min_samples_split**: número mínimo de muestras que debe tener un nodo interno para partirse. Un número muy pequeño conduce al sobreaprendizaje.
- **Min_samples_leaf**: número mínimo de muestras que ha de tener un nodo hoja. De nuevo un número pequeño conduce al sobreaprendizaje.

K - nearest neightbours

Sus parámetros principales son:

- **N_neighbors**: número de vecinos en los que nos vamos a basar para hacer las predicciones
- **Weights:** pesos que les damos a cada uno de los vecinos. Si queremos dar a todos los vecinos el mismo peso emplearemos *uniform*, mientras que si queremos ponderarlos por la distancia usaremos *distance.*
- **Algorithm**: algoritmo que empleamos para calcular cuáles son los vecinos más cercanos. Dado que tenemos un conjunto grande de datos, vamos a usar un BallTree, una estructura de datos avanzada se emplea para computar los vecinos más cercanos en un conjunto de puntos. Está además especialmente indicado para espacios con un número alto de dimensiones.
- **Metric**: métrica que empleamos para calcular la distancia. Vamos a probar con las métricas más conocidas: manhattan, euclídea y minkowski.

Itinerario 2: CNN preentrenada

Para esta aproximación vamos a emplear un clasificador de los que ya hemos visto (SVM, Random Fores o K-nn). Solo nos queda discutir como extraemos las features de las imágenes proporcionadas.

Al igual que en el anterior itinerario, vamos a emplear una red neuronal convolucional. Sin embargo, al estar ya preentrenada y no tener que entrenarla nosotros, podemos utilizar redes mucho más grandes y complejas.

La que nosotros vamos a emplear es VGGFace2. Está basada en la arquitectura de restnet50 (al menos la versión que estamos utilizando, la original se apoyaba en VGG16), cuenta con 23 millones y medio de parámetros (es unas 25 veces más grande que las propuestas por nosotros) y ha sido entrenada sobre 3,31 millones de caras pertenecientes a 9131 personas [11].

Al tener un tamaño mayor y más ejemplos, será capaz de extraer las características principales de la cara de una forma mucho más refinada que una CNN de propósito general.

4. Resultados

Itinerario 1: CNN desde 0

A la hora de buscar el modelo final, nuestro pipeline ha sido el siguiente:

1. Decidir la arquitectura de la red
2. Examinar la influencia del preprocesado (contraste y ángulos)
3. Escoger el mejor clasificador y ajustar sus parámetros
4. Reentrenar toda la red tanto con los datos de test como con los de validación

Lo ideal sería estudiar para cada arquitectura de la CNN su sinergia con los distintos clasificadores. Sin embargo, la gran cantidad de posibilidades llevaría a hacer una cantidad de tests excesivamente grande. Por ello primero decidiremos la arquitectura de la CNN y luego trabajaremos sobre la arquitectura fijada.

Importante:
Los datos de entrenamiento y test los hemos dividido en 3 partes: entrenamiento, validación y test.

- Los datos de **entrenamiento** sirven para que nuestro modelo aprenda (90% conjunto train).
- Los datos de **validación** sirven para elegir los hiperparámetros del modelo y evitar el sobreaprendizaje (10% conjunto train).
- Los datos de **test** sirven para ver el comportamiento de nuestro modelo sobre datos limpios.

En primer lugar vamos a entrenar nuestra red con diferentes hiperparámetros, solo con los datos de entrenamiento, y vamos a probar los resultados sobre el conjunto de validación. Una vez escogido el mejor, vamos a entrenar la red con los conjuntos de entrenamiento y validación y a ver los resultados definitivos sobre el test.

1. Decidir la arquitectura de la red

Un punto muy importante a tener en cuenta es que estamos entrenando nuestros modelos sobre una GPU, lo que emplea la librería cuDNN. Esta librería inicializa de forma arbitraria ciertos parámetros, y hace que los experimentos no sean reproducibles (puesto que cada ejecución, pese a fijar una semilla, se ignora) [12]. Por ello, para probar las arquitecturas hemos realizado varias pruebas y tomado la media como medida. Esto además no afectará al modelo final, puesto que una vez hemos entrenado el modelo, guardamos los pesos de la red y al cargarlos sí que obtendremos los mismos resultados.

Para cada una de las arquitecturas presentadas, hemos obtenido los siguientes resultados[5]

Arquitectura	Función de pérdida	Precisión media	Precisión máxima	Precisión mínima	Varianza
A	categorical_crossentropy	51,694	53,07	50,14	0,988
B	categorical_crossentropy	56,143	57,98	54,50	1,145
C	categorical_crossentropy	56,490	58,09	54,67	1,475
D	categorical_crossentropy	57,570	59,03	55,96	1,333
E	categorical_crossentropy	57,339	59,14	54,57	1,645
F	squared_hinge	58,679	59,66	56,76	0,919
G	squared_hinge	57,937	59,34	57,01	0,586

[5] Esta tabla recoge los estadísticos obtenidos en los experimentos. En la sección de anexos se puede encontrar una tabla (7.1) detallada con los resultados de las pruebas.

Vemos que la arquitectura que nos da mejores resultados es la F. Por un lado, en cuanto a precisión media aventaja en más de medio punto a la siguiente arquitectura, la G. Además, el mejor modelo de todos los entrenados ha sido también de esta arquitectura, logrando un 59,66% de precisión en el conjunto de validación. Por último, también ha sido una de las arquitecturas más consistentes. Por ello, la arquitectura escogida para nuestra CNN será la arquitectura F.

2. Preprocesado

Realizamos ahora pruebas sobre la arquitectura F para ver la influencia de las diferentes técnicas de preprocesado[6]:

Preprocesado	Precisión media	Precisión máxima	Precisión mínima	Varianza
Sin preprocesado	58,679	59,66	56,76	0,918521
Histogram equalization	57,493	58,65	56,03	1,055757
CLAHE size=(4,4) limit=4	58,604	60,77	56,38	2,302004
Incluir las imágenes especulares	**60,867**	**61,85**	59,38	0,712112
Imágenes especulares + CLAHE	60,304	61,51	57,81	1,77516

Vemos que al añadir histogram equalization, los resultados empeoran notablemente, perdiendo más de un punto de precisión media. Sin embargo, al utilizar la técnica CLAHE, la media se mantiene, pero se dispara la varianza, obteniéndose resultados mucho menos consistentes. Eso lleva a que la precisión máxima aumente en más de un punto. En caso de que la usemos finalmente, deberemos tener cuidado y escoger un modelo que haya obtenido buenos resultados.

Por otro lado, doblar el número de imágenes de entrenamiento, añadiendo cada imagen y su reflejo sí que provoca una mejora sustancial del funcionamiento de la CNN. La precisión media mejora y máxima mejoran en 2 puntos frente a la CNN sin preprocesado. Además, se reduce la varianza, obteniendo en todas las ejecuciones modelos más consistentes.

Por último, vamos a probar el efecto combinado de añadir la imagen especular y usar CLAHE. Vemos que empeoramos medio punto la precisión media respecto al caso anterior, aumentando además la varianza, y no logrando mejorar la precisión máxima. Por ello, prescindiremos de las técnicas de mejora de contraste en nuestro modelo final.

3. Clasificador

En este último paso elegimos el clasificador que vamos a utilizar.

Un problema con el que nos hemos encontrado es que en la capa de Flatten, el vector obtenido va a tener tamaño 6x6x128 = 4608. Dado que en el entrenamiento usamos unas 25000 fotografías, los clasificadores tienen que ajustar una cantidad ingente de puntos de gran dimensionalidad. Por ello, usando como entrada de nuestros clasificadores la salida de la capa de Flatten requiere mucho tiempo y no obtienen muy buenos resultados. Para obtener mejores

[6] De nuevo en la sección de anexos, tabla 7.2, tenemos los resultados detallados

resultados, las vectorizaciones de las imágenes que vamos a usar van a ser la salida tanto de la primera como de la segunda capas densas (capas 13 y 14 del modelo, indexando desde 0), que tienen 64 componentes. En estas capas tenemos menor capacidad de aprendizaje en el clasificador (puesto que estamos más cerca de la capa de predicción en la CNN) pero tenemos los datos más sintetizados. Las pocas pruebas que hemos hecho usando como vectorización la salida de Flatten (dado que requieren una gran cantidad de tiempo) nos indican que esta es la mejor aproximación.

Vemos los resultados para cada uno de los clasificadores[7]:

Softmax

Para este clasificador ya hemos visto cuál es la arquitectura que vamos a utilizar: la F preprocesando las imágenes e incluyendo su reflejo respecto al eje vertical. Sin embargo, ahora vamos a entrenarla empleando tanto los conjuntos de train como el de validación y vamos a probarlo sobre el conjunto de test. Obtenemos los siguientes resultados:

Datos de entrenamiento	Train	Validación	Test
Solo test	0.8652	0.6185	0.6105
Test + Validación	0.8209		0.6226

Vemos que al entrenarlo tanto con el conjunto de entrenamiento como de validación logramos mejorar los resultados hasta un 62,26% de acierto, lo cual ya es una ganancia de un 6% de acierto frente al modelo que usamos como baseline.

Las pruebas con el resto de clasificadores las hemos hecho sobre la CNN entrenada solo con datos de entrenamiento. Como ya dijimos antes, el conjunto de validación se emplea para seleccionar hiperparámetros y para evitar el sobreaprendizaje. Si entrenásemos el modelo directamente sobre todos los datos de entrenamiento, el modelo tendería a ajustarse solamente a los datos de entrenamiento, obteniendo peores datos en los datos de prueba. Los datos de prueba únicamente se utilizan para ver el rendimiento final de nuestro modelo.

SVM

Utilizamos como Kernel siempre el Gaussiano (`kernel = RBF`), y ajustamos la SVM empleando también las imágenes especulares. Como CNN usamos la arquitectura F con preprocesado, que obtenía un 61,85% en validación y 61,05% en test. Ajustamos los valores de los hiperparámetros (`C` y `gamma`), obteniendo:

[7] Para este punto nos hemos apoyado en el código del repositorio: https://github.com/shibuiwilliam/Keras_Sklearn

Features	C	Gamma	Train	Validación	Test
Segunda densa	1000	0.01	0.92789	0.60286	0.60867
Segunda densa	100	0.01	0.91315	0.60669	0.60922
Segunda densa	100	0.1	0.97892	0.59623	0.59599
Segunda densa	10	0.01	0.90567	0.60844	0.61117
Segunda densa	1	0.01	0.90191	0.61367	0.60964
Segunda densa	0.1	0.01	0.89503	**0.61820**	0.61145
Segunda densa	0.1	0.01	0.87619	0.60739	0.60309
Primera densa	0.1	0.01	0.87850	0.61402	0.60504
Primera densa	0.1	0.01	0.89616	**0.62029**	0.61006
Primera densa	1	0.01	0.90416	0.61262	0.61048
Primera densa	0,1	0.001	0.88203	0.61785	0.60671
Primera densa	0.1	0.1	0.87934	0.60530	0.59515

Vemos que para la featurización extraída de la primera y segunda capa densa, los valores que mejor funcionan son `C = 0,1` y `Gamma = 0,01`, rozando en ambos casos el 62% de acierto en validación y obteniendo 61% en test.

Para estos parámetros, entrenaremos la SVM empleando tanto el conjunto de entrenamiento como el de validación y lo probaremos sobre el conjunto de prueba. Ahora sí que utilizaremos como base la CNN con la que obteníamos un 62,26% de acierto:

Features	C	Gamma	Train	Test
Segunda densa	0.1	0.01	0.82735	**0.62483**
Primera densa	0.1	0.01	0.83031	0.62413

Vemos una mejora muy pequeña desde el clasificador Softmax, pasando de 62,26% a 62,48% de acierto en el mejor de los casos.

K-nearest neighbours

Usando la misma CNN, ajustaremos el K-nn. Usaremos el parámetro `algorithm = ball_tree`, como ya habíamos dicho. Ajustaremos los parámetros `metric`, `n_neighbors` y weights. Como este algoritmo tarda mucho menos tiempo en entrenar, para ajustarlos utilizaremos GridSearchCV. Es una función de Sklearn que nos permite introducir distintos valores de parámetros y ver cuál de las combinaciones funciona mejor. En nuestro caso, probamos usando como features la salida de la segunda capa densa, y los siguientes valores para los parámetros:

- N_neighbors: 1, 5, 10, 30, 50, 70, 100
- Weights: 'uniform', 'distance'
- Metric: 'minkowski', 'euclidean', 'manhattan'

Como resultado, obtenemos que el mejor k-nn usa `n_neighbors` = 30, `weights` = 'distance' y `metric` = 'minkowski'. Solo nos queda ajustar el número de vecinos y ver los resultados para la featurización con la primera capa.

Featurización	n_neigh	weights	metric	Train	Validación	Test
Segunda densa	30	distance	Minkowski	0.99864	0.62308	0.61633
Segunda densa	25	distance	Minkowski	0.99864	0.62134	0.61577
Segunda densa	20	distance	Minkowski	0.99864	0.62204	0.61535
Segunda densa	35	distance	Minkowski	0.99864	**0.62413**	0.61675
Segunda densa	40	distance	Minkowski	0.99864	**0.62622**	0.61633
Segunda densa	45	distance	Minkowski	0.99864	**0.62796**	0.61633
Primera densa	35	distance	Minkowski	0.99864	0.62378	0.61605
Primera densa	40	distance	Minkowski	0.99864	0.62448	0.61591

Una vez que encontramos los mejores parámetros, ajustamos el entrenamiento definitivo. En este caso hemos visto que la segunda densa se comporta mejor que la primera, con lo que vamos a trabajar solo con el output de la segunda densa:

Featurización	N_neigh	Weights	Metric	Train	Test
Segunda densa	35	distance	Minkowski	0.99864	0.62817
Segunda densa	40	distance	Minkowski	0.99864	0.62859
Segunda densa	45	distance	Minkowski	0.99864	**0.62873**

Vemos que entre las tres posibilidades hay diferencias muy poco significativas. Nos quedaremos con la última opción, que logra un 62,87% de precisión, mejorando en 4 décimas la precisión del Softmax.

Random forest

Usamos la misma CNN para extraer las features. Además, escogemos `max_features = 'auto'` y buscando ajustar los hiperparámetros obtenemos:

Features	max depth	boostrap	min samples split	min samples leaf	criterion	n estimators	Train	Validación	Test
Primera densa	20	True	5	5	entropy	1000	0.93473	0.61890	0.61247
Primera densa	20	True	5	5	gini	1000	0.93289	0.61925	0.61271
Primera densa	20	True	5	5	gini	1250	0.93305	0.61820	0.61201
Primera densa	20	True	3	5	gini	1000	0.93318	0.61924	0.61145
Primera densa	20	True	5	3	gini	1000	0.95523	0.62169	0.61257
Primera densa	None	True	5	3	gini	1000	0.95657	0.62133	0.61396
Primera densa	None	False	5	3	gini	1000	0.98409	**0.62377**	0.61437
Segunda densa	None	False	5	3	gini	1000	0.98785	**0.61681**	0.61590
Segunda densa	20	False	5	3	gini	1000	0.95566	**0.61541**	0.61605

Los resultados que obtenemos con Random Forest son ligeramente mejores a los que obteníamos con SVM. Vemos además que de nuevo los valores obtenidos con la featurización extraída de la primera y de la segunda capas densas son muy similares.

Por último, cogemos los modelos más prometedores y los probamos sobre el conjunto tanto de train como de validación. Vemos además que en este caso no hay una correlación directa entre los mejores resultados en validación y en test.

Features	max depth	boostrap	min samples split	min samples leaf	criterion	n_estimators	Train	test
Primera densa	None	False	5	3	gini	1000	0.95338	**0.63096**
Segunda densa	None	False	5	3	gini	1000	0.98380	0.62859
Segunda densa	20	False	5	3	gini	1000	0.96443	0.62831

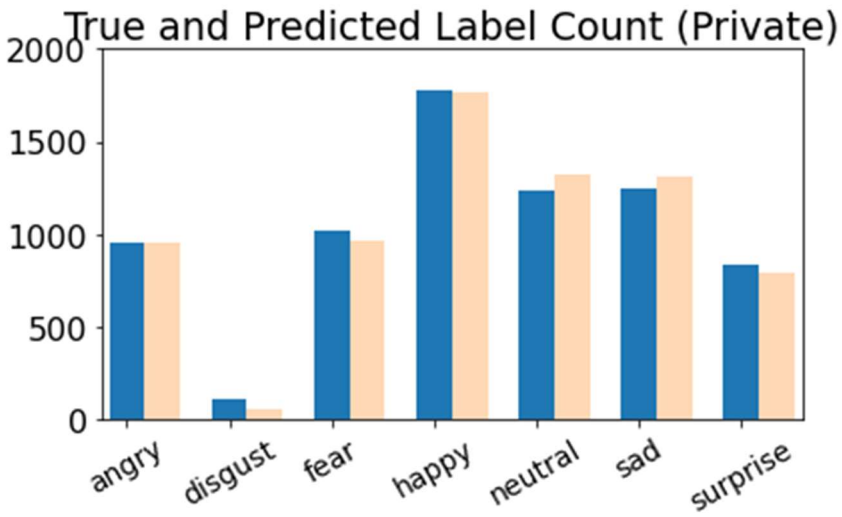
Con el mejor Random Forest sí que obtenemos una mejora sustancial sobre el Softmax, llegando a un 63,10% de precisión. Al contrario que en el caso anterior, aquí obtenemos mejores resultados extrayendo las features de la primera capa densa.

Resultados itinerario 1

Ahora que hemos visto los resultados para los diferentes clasificadores, vemos que el que mejor funciona es la CNN con arquitectura F con la que obteníamos un 62,26% en test, utilizando como vectorización de la imagen el output de la primera capa densa. Esta vectorización la pasaremos por un clasificador Random Forest, con los parámetros que acabamos de ver:

- `Max_depth = none`
- `Boostrap = False`
- `Min_samples_split = 5`
- `Min_samples_leaf = 3`
- `Criterion = Gini`
- `N_estimators = 1000`

Con el cual obtenemos un 63,1% de precisión en la clasificación de las emociones. Vamos a examinar un poco más estos resultados. Comenzamos viendo el número de imágenes predichas por cada clase:



En azul tenemos el número de fotos que hay por clase y en beige el número de fotos que predecimos de cada tipo. Una primera consideración que tenemos que hacer es que las clases de felicidad y asco están bastante desbalanceadas respecto al resto, la de felicidad teniendo muchas muestras y la de asco teniendo muy pocas.

Sin embargo, es buena señal que la cantidad de fotos predichas de cada clase sea similar al número de fotos que realmente están en dicha clase. Al haber por ejemplo tantas fotos de rostros que expresen felicidad (1774, frente a las 1247 de tristeza, la siguiente clase mayoritaria), nuestra red podría haberse centrado en predecir muchas fotos como felices (ya que la probabilidad de que una cara sea feliz es mayor que de que exprese otra emoción) para maximizar el porcentaje de acierto, cosa que no ha hecho. Estos datos respecto al tamaño de las clases son para el conjunto de test, pero la prevalencia de cada clase en ambos conjuntos (train y test) es similar.

Examinamos ahora las métricas agregadas para cada una de las clases. Para cada clase, además del número de muestras en esa clase tendremos 3 métricas:

- **Precision** (valor predictivo positivo o precisión):

$$precision = \frac{verdaderos\ positivos}{positivos\ predichos}$$

  Es decir, de los que hemos dicho que son de una clase, cuántos son efectivamente de ella.

- **Recall** (tasa de verdaderos positivos o exhaustividad):

$$recall\ = \frac{verdaderos\ positivos}{positivos\ reales}$$

  Es decir, de los que son de una clase, cuántos hemos predicho que son de dicha clase.

- **F1-score**: es una combinación de la precisión y el recall. Cuanto más alta sea el valor F1, mayor será el equilibrio obtenido entre precisión y recall, es decir, entre las fotos que predecimos correctamente de una clase y las fotos que encontramos de una clase.

$$F1\ = 2 \cdot \frac{precision \cdot\ recall}{precision\ +\ recall}$$

  Esta métrica es importante porque es muy sencillo encontrar todas las fotos de una clase (elevado recall) a costa de sacrificar la precisión de esa clase, clasificando muchas muestras en esa clase. O viceversa, tener una precisión muy elevada a costa de encontrar pocas fotografías de dicha clase

Los resultados obtenidos son:

Clase	Precision	Recall	F1-score	Número de muestras
Enfado	0.55	0.55	0.55	958
Asco	0.72	0.44	0.51	111
Miedo	0.47	0.44	0.45	1024
Felicidad	0.84	0.84	0.84	1774
Neutral	0.58	0.62	0.60	1233
Tristeza	0.48	0.50	0.49	1247
Sorpresa	0.80	0.77	0.78	831
Todas las clases (ponderadas)	0.63	0.63	0.63	7178

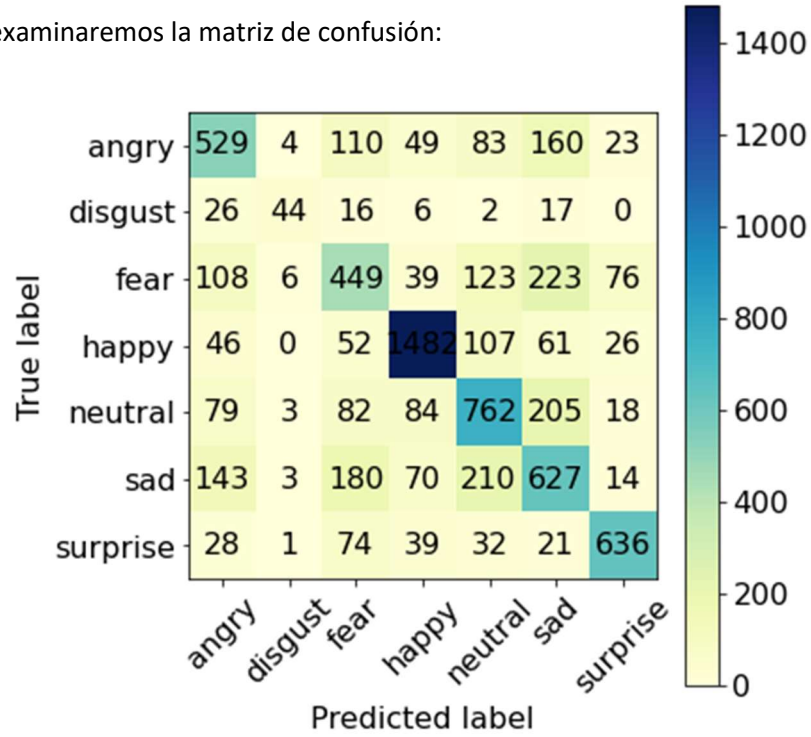
Vemos que, efectivamente feliz es la clase que estamos prediciendo mejor. No solo es la que más acertamos cuando decimos que una foto representa a una persona feliz, sino que también encontramos a la mayor parte de personas felices. Como ya anticipamos antes, esto se debe probablemente a la mayor cantidad de ejemplos de personas felices, unido a que durante el entrenamiento a nuestro modelo le conviene maximizar el acierto en esta clase para maximizar la puntuación general.

La siguiente clase en la que obtenemos buenos resultados es sorpresa. Esto es algo inesperado, puesto que tenemos varias clases con un número similar de fotos (y bastantes más) en las que obtenemos resultados notablemente peores. Sin embargo, si nos fijamos en la matriz de confusión que hicimos en el apartado 2, vemos que catalogamos sorpresa como la emoción más reconocible. Tan solo consideramos que era muy confundible con miedo y algo confundible con enfado. Con la matriz de confusión veremos si efectivamente estos buenos resultados se deben a que es una expresión bastante reconocible o hay algo más.

A continuación, tenemos las clases de enfado, miedo, neutral y triste. Son clases con un número similar de muestras (entre 958 y 1247) y con valores de F1 similares (entre 0.45 y 0.60), estando las puntuaciones de precisión y exhaustividad bastante equilibradas.

Por último, tenemos la clase de asco. Tenemos una precisión muy alta (de los que decimos que son de esta clase, acertamos un 72%, el tercer mejor porcentaje de acierto con muchos menos ejemplos de entrenamiento que el resto) a costa de encontrar muy pocas imágenes de esta clase (tan solo un 44%). Esto concuerda con el histograma anterior, de todas las clases predecíamos una cantidad muy similar de fotos a las que había en realidad. Pero de asco predecimos aproximadamente la mitad. Prediciendo tan pocas fotos de esa clase, no podemos encontrar la mayoría. Sin embargo, poco podemos hacer por mejorar esta situación. En nuestra investigación vimos que en la competición de Kaggle había gente que eliminaba esta categoría, juntándola con la que ellos consideraban que era más similar (y nunca predecían esta emoción) [13]. Sin embargo, esta es una idea que no nos gusta mucho, por lo que simplemente dejamos que nuestro modelo haga lo que pueda con ella.

Por último, examinaremos la matriz de confusión:



Un primer comentario acerca de la matriz es que es bastante simétrica. Esto es bueno, porque significa que no estamos tratando de maximizar las veces que acertamos prediciendo una clase a costa de fallar más en las restantes. Es decir, el modelo está tratando de obtener una buena puntuación F1, equilibrando la cantidad de fotos que encuentra de una clase (recall) con las que acierta de las predichas para esa clase (precisión), como ya vimos en las métricas por clase.

Comparamos ahora la matriz de confusión obtenida con la que predijimos en base a las AUs. Eliminamos las casillas correspondientes al asco, ya que hemos visto que en nuestro modelo tenemos problemas con ella:

	Enfado	Asco	Miedo	Felicidad	Neutral	Triste	Sorpresa
Enfado							
Asco							
Miedo							
Felicidad							
Neutral							
Triste							
Sorpresa							

Real

	Enfado	Asco	Miedo	Felicidad	Neutral	Triste	Sorpresa
Enfado							
Asco							
Miedo							
Felicidad							
Neutral							
Triste							
Sorpresa							

Predicha

Vemos que, si bien ambas matrices no coinciden exactamente, sí que son bastante consistentes, y que no hay ninguna clase que considerásemos que no fuese a ser muy confundida y no haya sido confundida en absoluto o viceversa. De hecho, nuestra predicción era más pesimista y en líneas generales el modelo se ha equivocado menos de lo esperado.

Itinerario 2: CNN preentrenada

Al trabajar con VGGFace, la vectorización de las imágenes tiene 2048 componentes, frente a las 64 que teníamos antes. Por ello, van a ser modelos mucho más lentos de ajustar Sobre la vectorización aplicamos una regularización L2, con parámetro 1e-10. Tras ello ajustamos los siguientes clasificadores:

SVM

A pesar de que para nuestra arquitectura de red ha tenido los peores resultados (exceptuando el Softmax), es en el que tenemos más confianza. Esto se debe a que es una combinación que se ha utilizado habitualmente en la literatura para resolver este problema, como por ejemplo en [14].

Al trabajar con el embedding, el tiempo que necesitamos para ajustar los modelos es mucho más grande y por lo tanto podemos hacer muchas menos pruebas. Mantendremos `gamma = "scale"` y variaremos el valor de `C`:

C	Gamma	Validación	Test	Tiempo ent (s)
1	scale	0.66548	0.66392	6027
10	scale	**0.67900**	0.68320	7107
100	scale	**0.67970**	0.68445	6730

Vemos que para los valores `C=10` y `C=100` obtenemos un resultado similar en validación, así que probaremos ambos valores:

C	Gamma	Test	Tiempo ent (s)
10	scale	**0.68933**	8962
100	scale	0,68557	8962

Finalmente vemos que con `C=10` obtenemos el mejor resultado, rozando el 69% de acierto. Este es una mejora sustancial respecto a nuestra CNN, de casi un 6%, y un 13% frente a nuestra baseline.

Random Forest

Ahora utilizamos como clasificador un Random Forest. Tenemos mucho interés en obtener buenos resultados con el Random Forest, puesto que, a diferencia de la SVC, puede entrenarse de forma paralela. El ordenador en el que estamos trabajando tiene 12 hilos, con lo que poder trabajar en paralelo supone una aceleración considerable con respecto a trabajar en un único hilo. Obtenemos los siguientes resultados:

max depth	boostrap	min samples split	min samples leaf	criterion	n_estimators	Validación	Test	Tiempo entrenamiento
None	False	5	3	gini	1000	0.63360	0.64015	131
None	False	5	3	gini	1500	0.63011	0.64125	193
None	False	5	3	gini	2000	0.63430	0.64196	257
30	False	5	3	gini	2000	0.63709	0.64154	256
35	False	5	3	gini	2000	0.63604	0.63890	258
25	False	5	3	gini	2000	**0.64059**	0.64196	260
25	False	7	3	gini	2000	0.63360	0.63918	262

Para el entrenamiento con los conjuntos de entrenamiento y validación obtenemos:

max depth	boostrap	min samples split	min samples leaf	criterion	n_estimators	Test	Tiempo entrenamiento
25	False	5	3	gini	2000	**0.64711**	323

Vemos que con todas las imágenes de entrenamiento logramos un respetable 64,71% de precisión, aumentando en casi 0,7% desde el entrenamiento con el conjunto reducido. Además, este modelo acierta un 1,6% más que el mejor Random Forest del itinerario 1, manteniendo un tiempo de entrenamiento comedido, lo que nos muestra la potencia del embedding preentrenado.
Sin embargo, el cálculo de la vectorización es mucho más lenta, en torno a 55 minutos para todas las imágenes. Además, es más rápido de entrenar porque podemos paralelizarlo, también podemos paralelizarlo en el itinerario 1 y obtener un tiempo mucho menor. Y teniendo un SVM que obtiene una precisión mucho mayor, no consideramos que nos aporte ni un aumento de precisión considerable respecto al itinerario 1 ni una reducción del tiempo de entrenamiento frente a la SVM previa.

Ensemble SVM

Una idea intermedia a estas dos podría ser una ensemble SVM. En vez de tener una única SVM grande, vamos a tener múltiples más pequeñas. Cada una de estas SVM pequeñas se va a entrenar con una porción de la información (no de forma excluyente), y a la hora de hacer una predicción votarán entre ellas. El entrenamiento de cada SVM es paralelizable, lo que nos permite reducir además el tiempo de entrenamiento.

Los parámetros con los que trabajaremos son `n_estimators`, el número de SVM que contemplamos, `max_samples`, el número de muestras con las que será entrenada cada SVM, `C`, que ya vimos antes, y fijaremos `Gamma='scale'`:

n_estimators	max_samples	C	Validación	Test	Tiempo ent
100	Tam train*0.1	1	0.60356	0.59668	2593
10	Tam train*0.2	1	0.61719	0.61619	863
5	Tam train*0.5	1	0.64024	0.64085	2064
10	Tam train*0.2	10	0.63989	0.64266	1064
5	Tam train*0.5	10	**0.66748**	0.67052	2575
5	Tam train*0.5	100	0.66678	0.66941	2591

De nuevo obtenemos unos resultados peores que con una única SVM, y además el tiempo de entrenamiento es bastante elevado. Sin embargo, sí que obtenemos una mejora considerable con respecto al Random Forest, con un tercio del tiempo de entrenamiento de una única SVM. Entrenando con el conjunto de entrenamiento y validación obtenemos:

n_estimators	max_samples	C	Test	Tiempo ent
5	Train*0.5	10	0.67679	6061

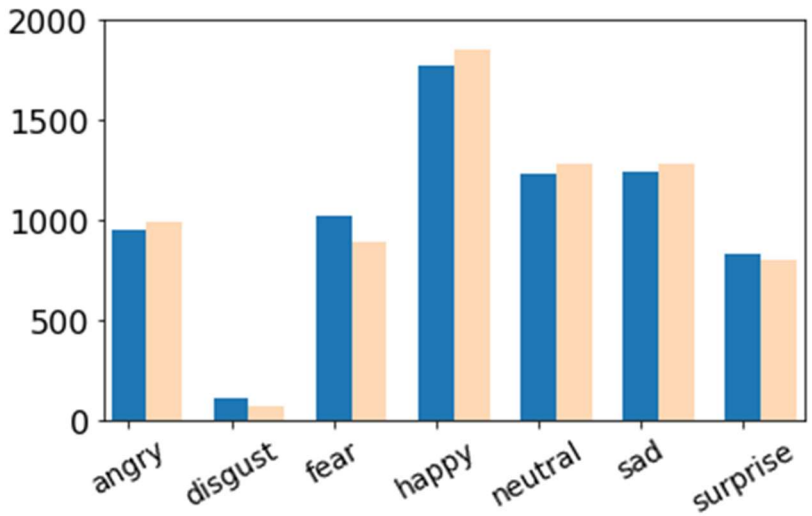
Tras entrenarlo con todos los datos la precisión sube hasta el 67,68%, quedando muy cerca de una única SVM. Sin embargo, el tiempo de entrenamiento también crece mucho, en nuestra opinión demasiado para la cantidad de datos añadida (2863 imágenes que estaban en validación). Aunque no remplaza al modelo conseguido con una única SVM, sí que consigue muy buenos resultados, y es un modelo con el que estamos también bastante contentos. Con un poco más de ajuste de hiperparámetros (número de SVM y datos con los que entrenamos cada SVM) es posible que incluso superase a nuestro mejor modelo.

K-nearest neighbours

Para VGGFace no vamos a hacer pruebas con este clasificador. Al tener la representación de las imágenes mucho mayor tamaño, los puntos estarán mucho más dispersos y es más complicado que dos imágenes de la misma clase tengan como vectorización puntos en un espacio 2048-dimencional que se encuentren próximos.

Resultados itinerario 2

Tras ver los resultados obtenidos, nos vamos a quedar con la SVM con parámetros `C=10` y `Gamma='scale'`. Vamos a repetir el mismo análisis que hicimos en el primer itinerario, relacionándolo con los resultados que obtuvimos en aquel. Para ello consultamos en primer lugar la distribución por clases:



De nuevo vemos que la cantidad de fotos predichas es muy similar a la cantidad de fotos reales en cada clase. Apreciamos una mayor descompensación que en el itinerario 1, viendo que el

sistema ha decidido predecir más imágenes de las clases mayoritarias (feliz, neutral, triste), y de menos en las clases con menos muestras (sorpesa y asco). En el caso de enfado y miedo, tienen una cantidad similar de fotografías.

Pasamos ahora a los estadísticos de cada clase:

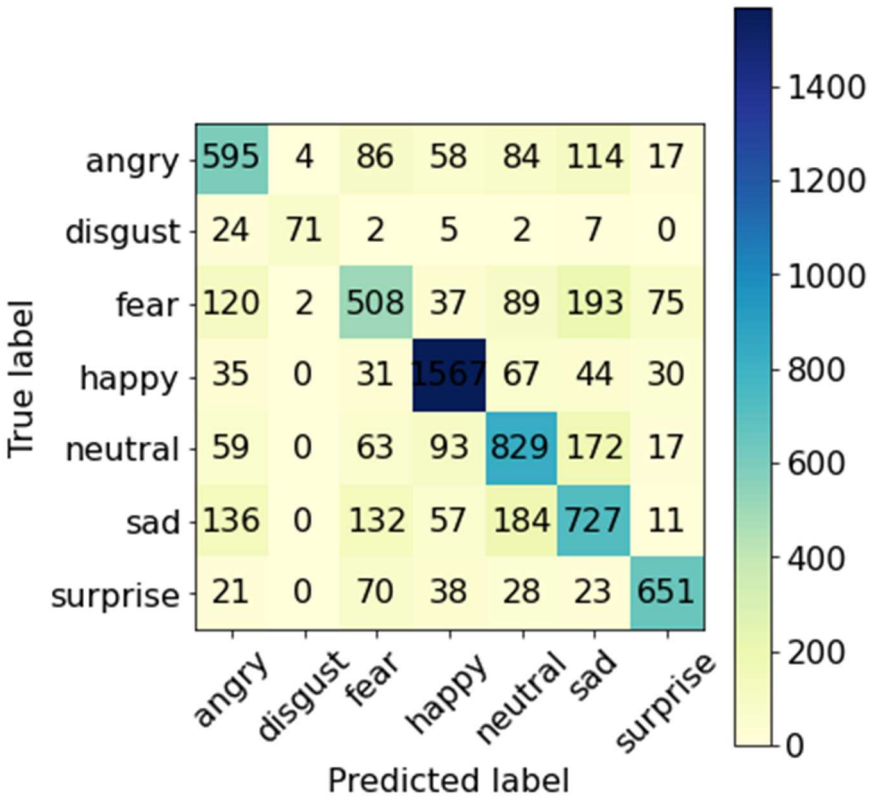
Clase	Precision	Recall	F1-score	Número de muestras
Enfado	0.60	0.62	0.61	958
Asco	0.92	0.64	0.76	111
Miedo	0.57	0.50	0.53	1024
Felicidad	0.84	0.88	0.86	1774
Neutral	0.65	0.67	0.66	1233
Tristeza	0.57	0.58	0.58	1247
Sorpresa	0.81	0.78	0.80	831
Todas las clases (ponderadas)	0.69	0.69	0.69	7178

En primer lugar, vemos que no ha empeorado ninguna métrica respecto al itinerario 1. De nuevo las clases felicidad y sorpresa son las claras vencedoras, estando además muy equilibradas. En ellas prácticamente no se produce mejora.

Donde sí que se produce mejora es en la "tabla media". Las emociones de enfado, miedo, neutral y tristeza experimentan un crecimiento en la precisión de entre 0.05 y un 0.1. En el recall 0.05 y 0.08, obteniendo consistentemente mejores resultados.

Por último, la clara ganadora es la clase de asco, que incrementa su puntuación F1 en 0.25, gracias a mejorar 0.2 tanto en precisión como en recall. Esto lleva a que prácticamente de la totalidad de fotos predichas como asco realmente contengan esa expresión (92%) y que se encuentre esta emoción en casi dos tercios de las veces (64%).

Para finalizar, examinamos la matriz de confusión:



Vemos que prácticamente entre cualesquiera dos clases se reduce el número de muestras confundidas. Además, esta reducción es bastante más acusada entre aquellas clases en las que teníamos un alto porcentaje de muestras incorrectamente clasificadas. Por otro lado, vemos un incremento consistente en todas las clases en el número de muestras en la diagonal principal, que se corresponden a las imágenes correctamente predichas.

En los casos en los que confundimos pocas muestras además tenemos que pensar en que quizás la culpa no la tiene totalmente nuestro modelo. Entre los datos de prueba, como ya dijimos anteriormente, tenemos cierta información corrupta, que es imposible de clasificar. Además, puede haber imágenes mal clasificadas, o varias caras con expresiones similares, que se encuentren en el limbo entre dos expresiones y que en ocasiones estén etiquetadas con una expresión y en otras con otra.

5. Definición del pipeline final

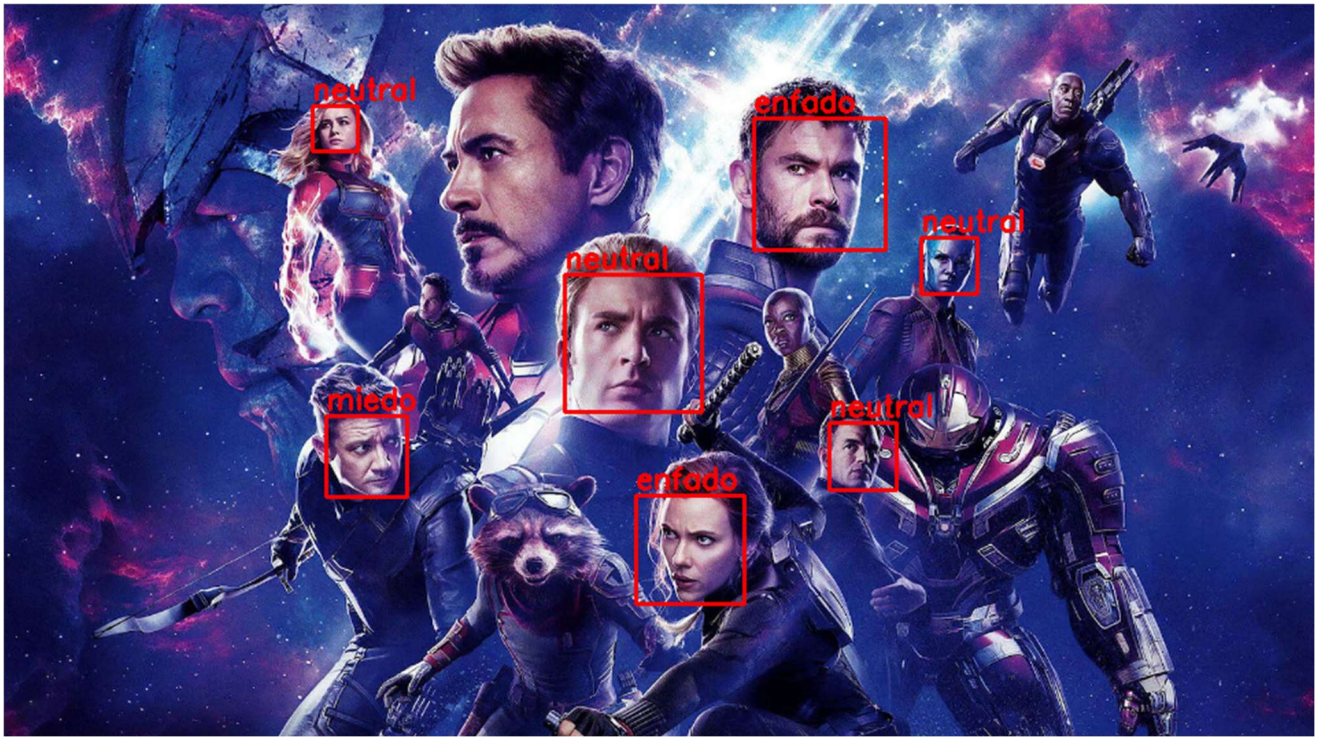
El pipeline final consta de 2 pasos:

1. Recortar de una imagen las caras presentes
2. Predecir la expresión de las caras recortadas

Ya hemos hablado largo y tendido del modelo que resuelve el paso 2. Además, tras entrenar el modelo, lo podemos almacenar. De esta forma, en el código final simplemente hay que cargarlo desde memoria y pedirle la predicción.

Para el paso 1 vamos a utilizar OpenCV, una librería de visión artificial que ya hemos utilizado en múltiples ocasiones en el paso 2. Concretamente, una función llamada CascadeClassifier, que detecta las caras en una fotografía mediante una cascada de clasificadores (cuya teoría ya explicamos previamente). Hay que pasarle como parámetro un fichero con el entrenamiento, que en este caso es "haarcascade_frontalface_default.xml".

Una vez tenemos hecho esto, ya lo tenemos todo. Algunos de los ejemplos que hemos probado son los siguientes:

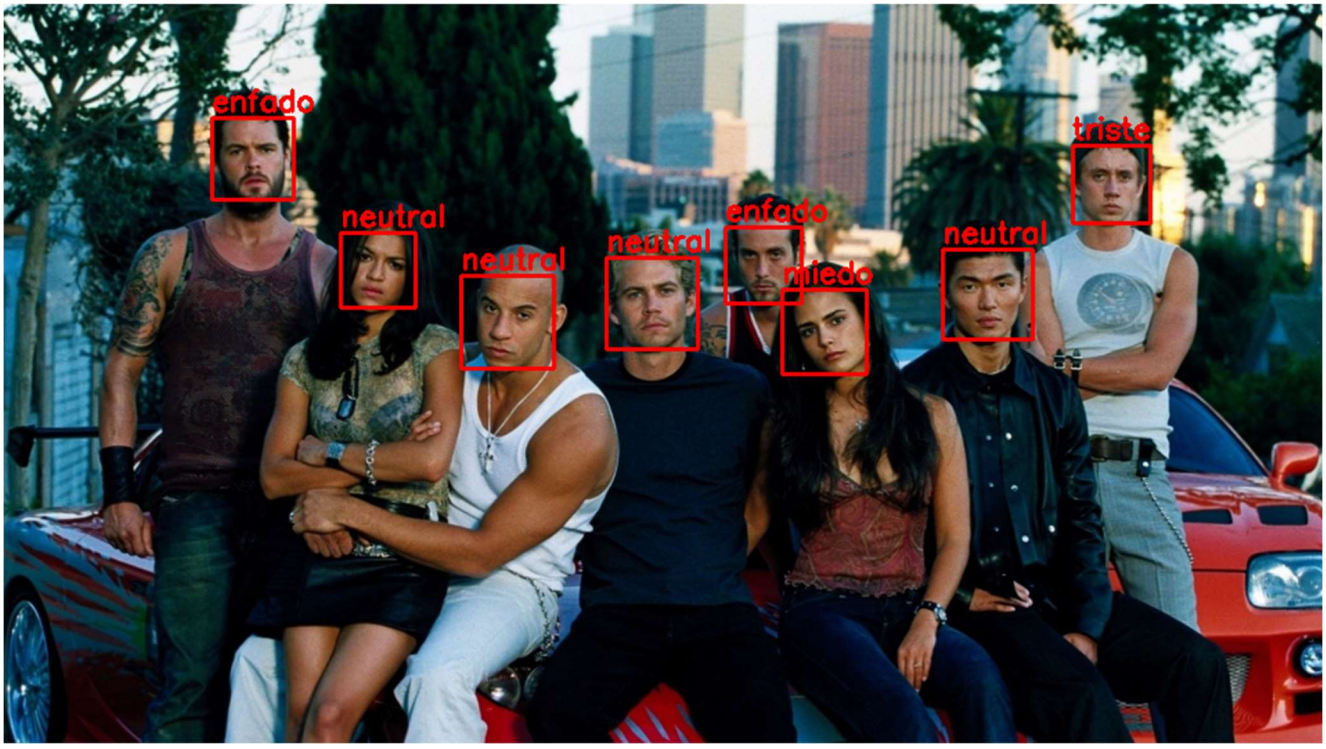


Comenzamos con una imagen promocional de Avengers: Endgame. En primer lugar, vemos que de los rostros humanos (sin contar a Iron Man y a Rocket), el clasificador en cascada preentrenado que hemos incluido en el proyecto detecta 7 de 11 caras. Se equivoca en Tony Stark, que a pesar de ser la cara más grande, está prácticamente de lado, y los tres personajes más al fondo, cuyo tamaño es muy pequeño. De hecho, la resolución de la imagen parece jugar un papel crucial en la detección de caras, ya que en varias pruebas que hemos hecho, las caras demasiado grandes o no las detecta o las detecta mal y las caras demasiado pequeñas no las detecta.

En cuanto a la detección de emociones sobre las caras detectadas, vemos que lo hace bastante bien. En primer lugar, detecta tanto a la Viuda Negra como a Thor enfadados. En el caso de la Viuda negra presenta claramente cara de enfado, mientras que en Thor podríamos dudar si tiene cara enfadada o neutral. Sin embargo, la presencia de AU4 (cejas fruncidas) es un rasgo distintivo del enfado, por lo que nos decantamos por la primera.

Por otro lado, Ojo de Halcón es catalogado como miedo, expresión con la que nosotros también coincidimos. Por último, tenemos 4 personajes catalogados como neutrales: Capitana Marvel, Capitán América, Hulk y Nébula. En el caso de Hulk y Capitana Marvel creemos su expresión es claramente neutral, expectante. En los casos del Capitán América y Nébula dudamos entre si su expresión es neutral o de enfado. En cualquier caso, los rasgos faciales de Nébula son poco visibles (al estar alejada y media cara en la penumbra), y al no ser clara ninguna de estas dos clases, creemos que la predicción no es mala. En el caso del Capitán América sí que nos habríamos decantado por enfado antes que neutral. Sin embargo, al estar media cara en penumbra es más complicada la predicción, y neutral sería nuestra segunda predicción.

Pasamos a otra foto grupal, en la que aparece el reparto de Fast and Furious 1. La foto es la siguiente:



En primer lugar, detecta correctamente todas las caras. Si bien están todas mirando de frente, el cuarto desde la derecha está ligeramente oculto tras Mia (tercera desde la derecha), lo cual añade dificultad. Además, Leti (segunda desde la izquierda) y Dom (tercero desde la izquierda), así como Mia tienen la cabeza ligeramente escorada. La detección es por lo tanto satisfactoria.

En cuanto a las expresiones, consideramos que esta foto es extremadamente difícil de analizar, incluso para un humano. Comenzamos desde la derecha. El primer actor es catalogado como triste, predicción con la que estamos de acuerdo. A pesar de tener el semblante serio, las

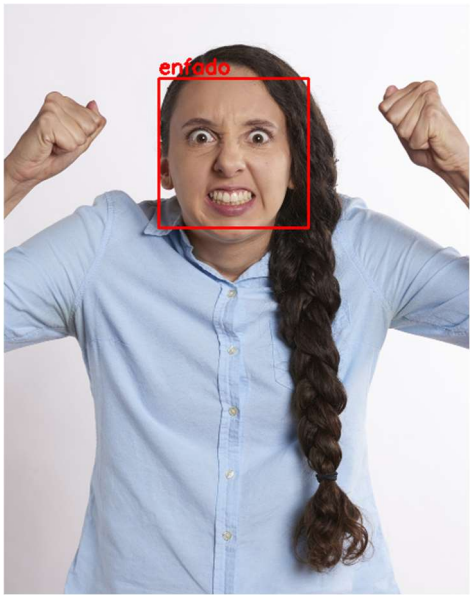
comisuras de los labios así como las cejas bajas (AU4 + AU15) nos hacen decantarnos por tristeza como primera expresión. En el segundo vemos un semblante inexpresivo, por lo que nuestra predicción también concuerda con neutral.

Llegamos a Mia, que el sistema predice con miedo. Esta es para nosotros la primera expresión en la que dudamos, en nuestro caso entre tristeza y miedo. Puesto que no es una expresión exagerada ni de tristeza ni de miedo (están presentes solo algunas unidades de acción de ambas expresiones) no creemos que sea una equivocación flagrante, aunque nosotros nos decantaríamos por tristeza. Además, miedo y tristeza hemos visto que son unas de las expresiones que más se confunden, tanto en nuestra tabla como en el sistema.

El cuarto lo clasifica como enfadado. Quizás no sería la emoción que nosotros pondríamos, pero desde luego no es una clasificación desastrosa. A continuación, tenemos a O'Connor y Dom, ambos neutrales, clasificación con la que coincidimos completamente. Solo nos quedan los dos primeros desde la derecha. Leti, la segunda desde la derecha es clasificada como neutral. Nosotros lo habríamos clasificado como asco, debido a la nariz torcida y fruncida (AU9). Sin embargo, ya hemos visto que el clasificador tiene problemas con la gente asqueada, debido a la falta de imágenes de entrenamiento. Por último, Vince, el primero desde la izquierda se clasifica como enfadado. Faltando contexto, dudamos si clasificarlo como enfadado o asqueado. Sin embargo, dado el semblante serio de todos los integrantes probablemente nos decantásemos por enfadado.

Consideramos por tanto que el sistema ha rendido bastante bien en una fotografía muy complicada, aunque ha cometido ciertos fallos, y nosotros también tenemos dudas en ciertas expresiones.

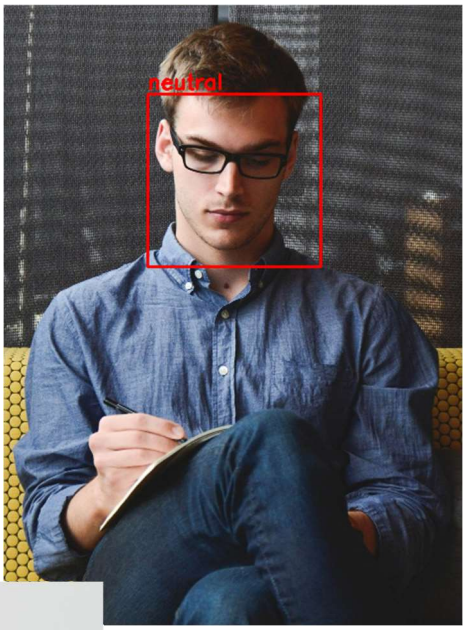
Por último, vamos a examinar unas fotos donde haya una única persona y las expresiones estén claras, para centrarnos en el funcionamiento del predictor de emociones.



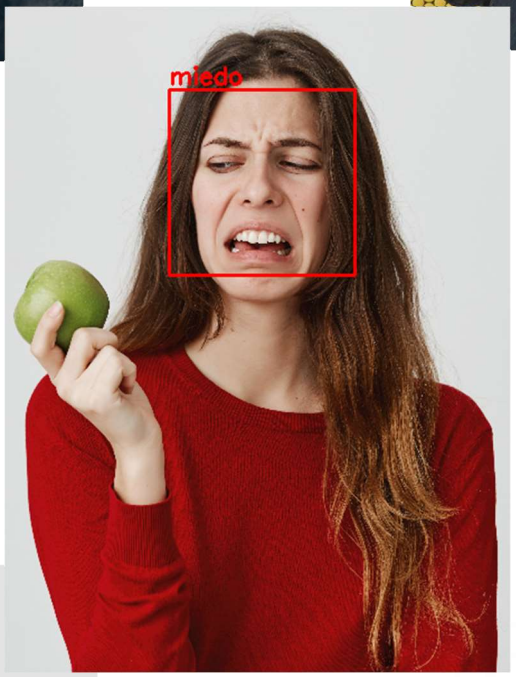
Enfado

Tristeza

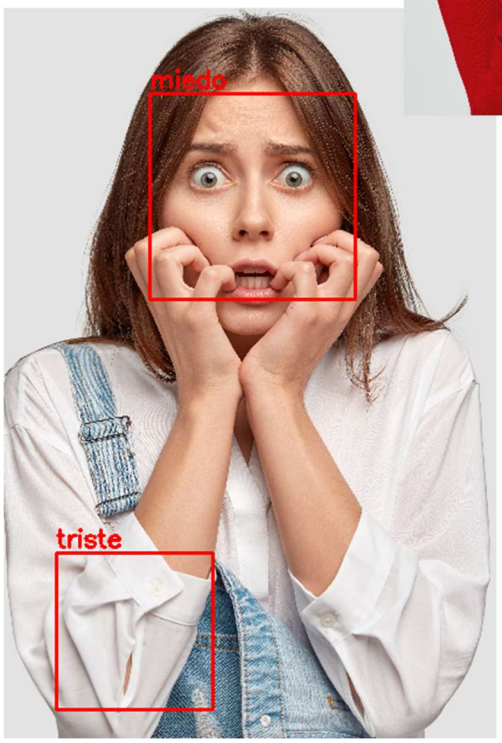
Felicidad



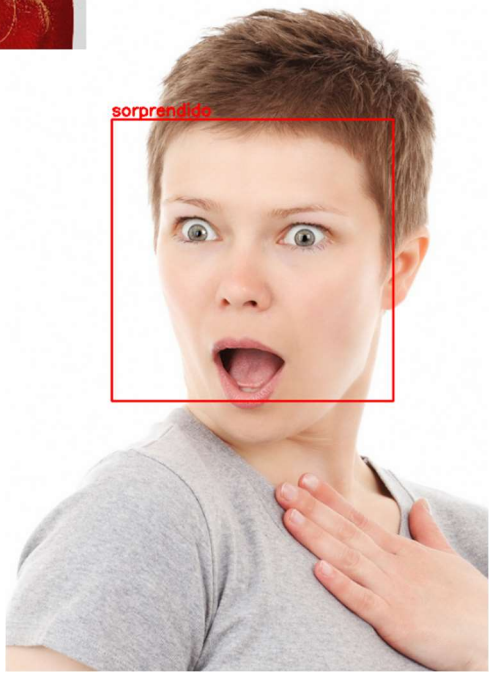
Neutro



Asco



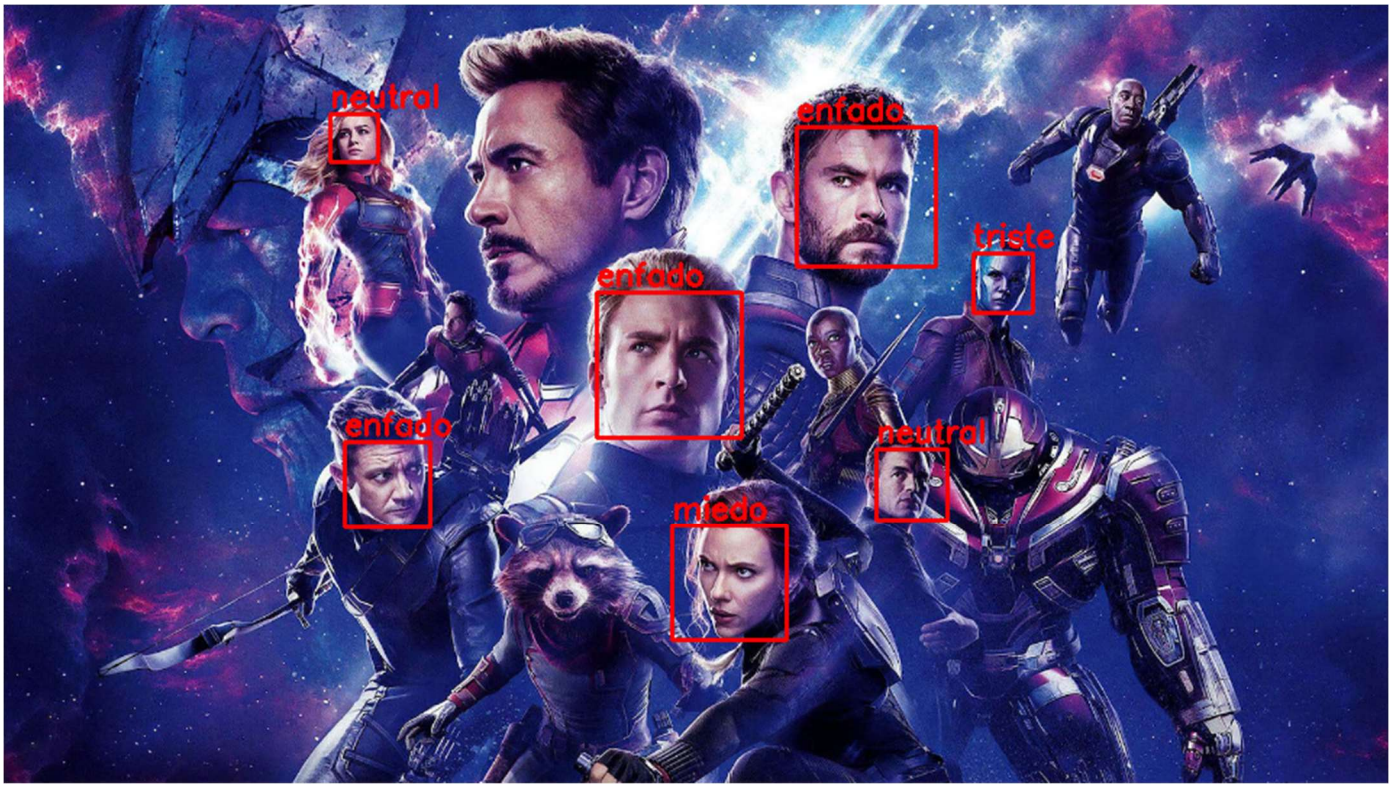
Miedo



Sorpresa

Vemos que en estas 7 fotografías comete 2 fallos. El primero es de nuevo con la imagen de asco, que la clasifica como miedo. De nuevo vemos que el sistema es poco robusto con la clase de asco. Por otro lado, en la foto de miedo detecta correctamente la expresión, pero también detecta una cara en la manga de la blusa. Ahí el error ha sido del detector de caras, no del clasificador de expresiones. Además, en la fotografía de felicidad el detector de caras también se equivoca, detectando caras donde no las hay (en el tronco del árbol).

Los ejemplos anteriores los hemos sacado con la *Propuesta 2* (en el punto 6 se indica cual es la configuración exacta del modelo, y lo que significa esto exactamente). La Propuesta 1 (también explicada en el punto 6), clasifica todas las imágenes individuales igual, salvo la de asco, que en este caso también falla confundiéndola con tristeza. En cuanto a las imágenes grupales tenemos:

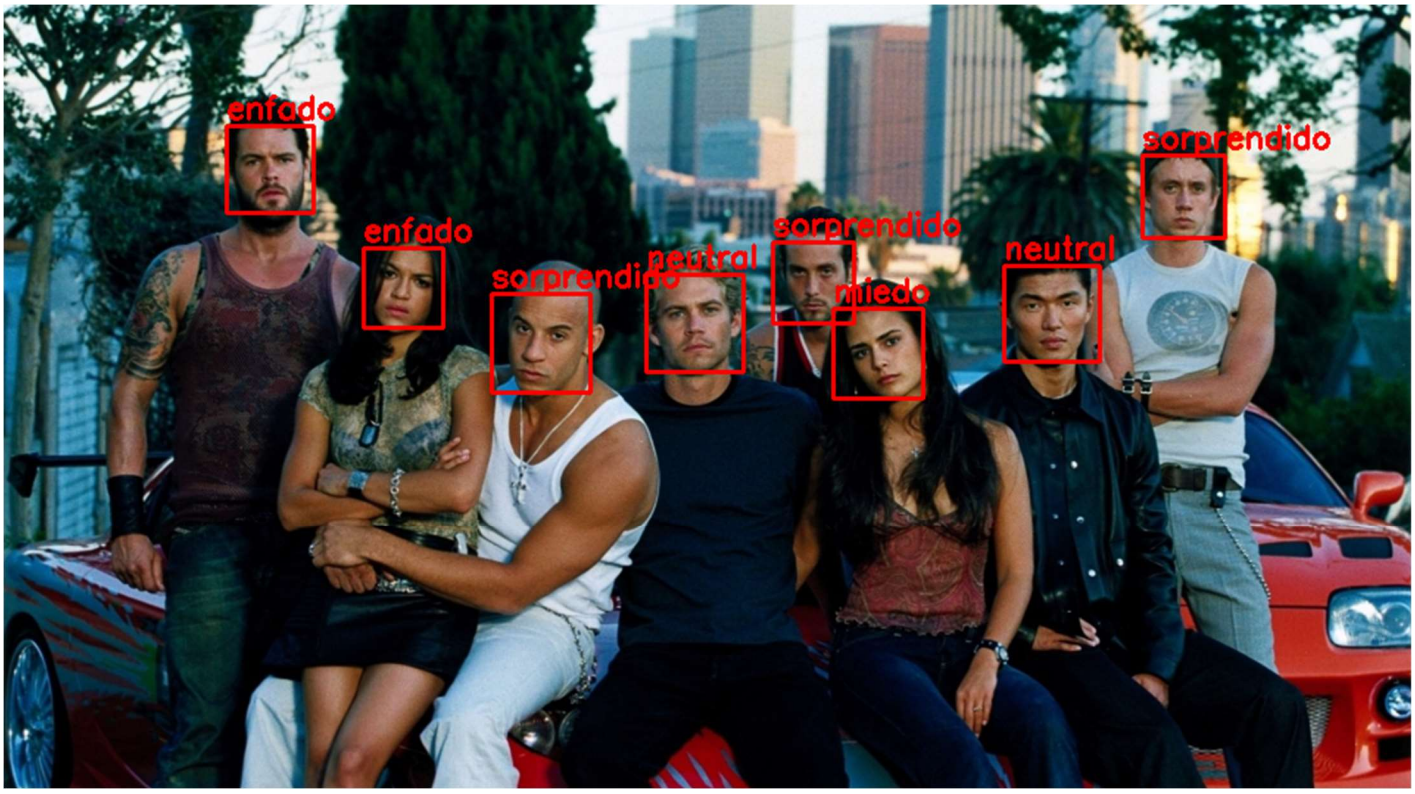


Vemos que la predicción varía bastante respecto a la primera propuesta, coincidiendo únicamente las predicciones de Capitana Marvel, Thor y Hulk. En el caso de Capitán América, lo predice esta vez sí como enfadado, que como ya dijimos antes es la expresión que más nos encaja a nosotros.

Por otro lado, predice a Nébula como triste. Esta es una predicción que tampoco nos desagrada, puesto que tiene las cejas elevadas y las comisuras de los labios caídas.

Por último, tenemos las predicciones de Ojo de Halcón y la Viuda Negra, que nos gustan menos que las anteriores. Por un lado, predice a Ojo de halcón como enfadado, cuando nosotros consideramos que es más una cara de miedo. Aunque el mentón fruncido es característico del enfado, por lo que entendemos esta clasificación. Por otro lado, predice a la Viuda Negra con miedo, cuando creemos que es un rostro claramente enfadado.

Por otro lado, en la segunda imagen grupal obtenemos:



Comenzando desde la derecha, se mantiene la predicción de la 2ª, 5ª y 8ª personas, con las que ya dijimos que coincidíamos, además de la 3ª, con la que teníamos algunas dudas.

Para la primera persona creemos que se equivoca, y que la anterior predicción (tristeza) era la adecuada. Para la 4ª persona también creemos que era más plausible la expresión de enfado que la de sorpresa. Para la Vin Disel (6ª persona) sí que nos encaja la predicción de sorprendido, debido al arqueamiento de las cejas, aunque nos gustaba más la de neutral. Por último, para Leti (7ª persona) nos gusta más la predicción de enfado que la de neutral que hacía antes, aunque seguiríamos decantándonos por asco.

Aunque en ambos modelos se han cometido fallos (o al menos se han escogido emociones que nosotros no creemos las más indicadas), creemos que el rendimiento del sistema ha sido bastante bueno, y que es evidente que no se están haciendo las predicciones sin un motivo. Además, tenemos que recordar que la red se está entrenando sobre un dataset. Si este dataset tiene sesgos, nuestra red adquirirá esos sesgos. Esto hará que obtenga mejores puntuaciones en la evaluación, pero que a nosotros nos parezcan unas predicciones más “artificiales”.

6. Conclusiones

En primer lugar, queríamos poner un poco en contexto los resultados obtenidos. Dado que hay 7178 imágenes en el conjunto de prueba, y de ellas 1774 son de caras felices (la clase mayoritaria), un sistema que predijese cualquier imagen como feliz tendría una precisión del 24,71%.

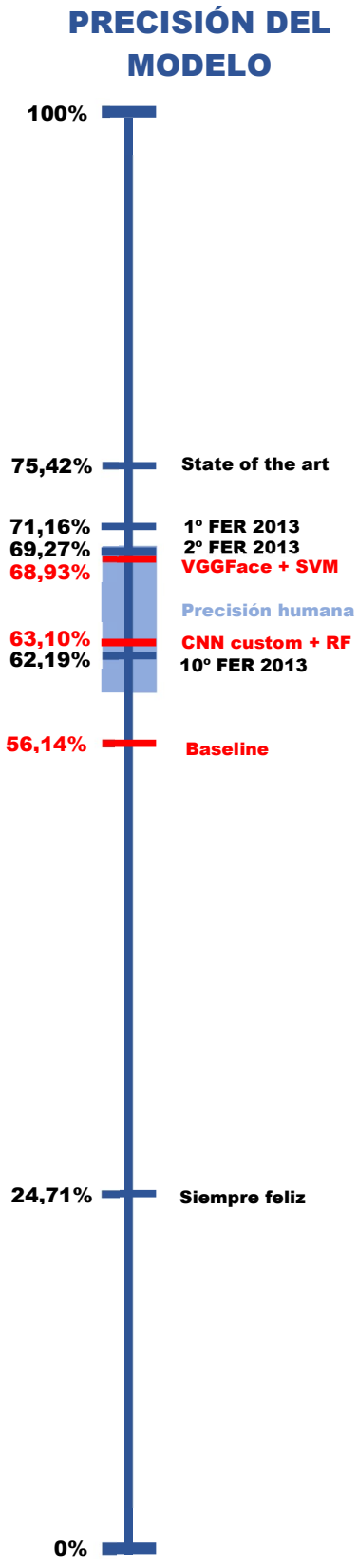
Este dataset apareció hace una competición en Kaggle hace 8 años, llamada *Challenges in Representation Learning: Facial Expression Recognition Challenge* [15]. En ella, el primer puesto consiguió un 71,16% de precisión, el segundo un 69,27% y el tercero un 68,82%. Se estima además que la precisión humana en este dataset es de 65±5% [16].

Por último, el *state of the art* en este dataset, que nosotros sepamos, es de un paper de marzo de este año [17], donde se obtenía un 75.42% de precisión.

Por ello, estamos muy contentos con los resultados obtenidos. Por un lado, obtuvimos un 63,1% de precisión con el modelo completamente construido por nosotros utilizando únicamente imágenes del dataset. Este modelo nos habría valido para quedar en décima posición en la competición FER2013.

Por otro lado, nuestro mejor modelo consigue un 68,93% de precisión. Esto nos habría valido para quedar en tercera posición en FER2013, y además mejora el promedio de acierto humano. Sin embargo, hay que tener en cuenta que usamos VGGFace, una red preentrenada (con fotografías externas al dataset), es mucho más pesada y no existía en el momento de la competición. Este modelo mejora además en un 12,79% la precisión del modelo del que partimos.

Por ello, hemos decidido hacer dos propuestas, cada una con sus propias ventajas, para que se adapten a todas las necesidades.



Propuesta 1

- **Máxima precisión: 68,93%**
- **Mayor tiempo de entrenamiento: 12275 s ≈ 3h 25min**
  - **Cálculo de la vectorización: 3313 s**
  - **Ajuste del clasificador: 8962 s**
- **Menor tamaño del modelo + clasificador: 90MB + 384MB**
- **Tiempo de predicción por fotografía: 40 ms**

Propuesta 2

- **Menor tiempo de entrenamiento: 70 s = 1 min 10 s**
  - **Cálculo de la vectorización: 53 s**
  - **Ajuste del clasificador: 17 s**
- **Menor precisión: 63,10%**
- **Mayor tamaño del modelo + clasificador: 9MB + 1,06GB**
- **Tiempo de predicción por fotografía: 1,93 ms**

7. Anexos

7.1 Resultados de los experimentos

Tabla 1: Pruebas arquitectura CNN

Arquitectura	Función pérdida	Nº experimento									
		1	2	3	4	5	6	7	8	9	10
A	entropy	52,82	51,22	51,12	52,41	51,39	50,14	52,68	50,63	51,46	53,07
B	entropy	56,59	56,87	57,98	56,8	54,88	55,75	55,51	56,94	55,61	54,5
C	entropy	55,82	57,78	54,67	57,32	57,43	55,93	57,22	55,58	55,06	58,09
D	entropy	55,96	56,94	58,68	56,35	56,42	57,04	58,72	59,03	57,95	58,61
E	entropy	56,69	57,67	57,08	59,14	58,05	54,57	58,47	58,09	57,78	56,45
F	hinge	59,10	57,29	**59,66**	59,14	59,03	56,76	59,14	58,86	59,55	58,26
G	hinge	58,12	57,95	59,34	57,18	57,78	57,09	57,64	58,47	57,01	58,79

En esta tabla se representa, para cada arquitectura del apartado 3, la precisión obtenida sobre el conjunto de validación en cada uno de los experimentos. El conjunto de validación se obtiene a partir del conjunto de entrenamiento proporcionado, que se ha dividido el 90% de las imágenes para entrenamiento y el 10% para validación.

Tabla 2: Pruebas preprocesado

Técnica	Nº experimento									
	1	2	3	4	5	6	7	8	9	10
Sin preprocesado	59,10	57,29	59,66	59,14	59,03	56,76	59,14	58,86	59,55	58,26
Histogram equalization	56,03	56,03	58,09	58,47	56,56	58,65	58,51	56,97	58,12	57,50
CLAHE size=(4,4) limit=4	56,38	60,63	57,46	57,57	59,27	59,45	60,77	56,83	59,14	58,54
Imágenes especulares	59,66	61,26	**61,85**	60,98	61,12	61,75	61,19	59,38	60,18	61,3
Im especulares + CLAHE	57,81	60,81	59,45	61,4	61,3	60,91	60,39	61,51	61,16	58,3

En esta tabla vemos el resultado de aplicar las distintas técnicas de preprocesado de los datos.

1. No se aplica preprocesado.
2. Se aplica la técnica de histogram equalization a toda la imagen al mismo tiempo.
3. Se aplica la técnica CLAHE, sobre recuadros de tamaño 4x4 y limitando la modificación de contraste para evitar amplificar el ruido con limit=4.
4. Se introducen tanto las imágenes como su reflejo respecto al eje central vertical.
5. Por último, se combinan utiliza CLAHE y se introducen las imágenes especulares, para ver el efecto conjunto de ambos preprocesados.

7.2 Entorno

Para entrenar el modelo, hemos empleado unos entornos con las siguientes características:

Hardware:

CPU: Ryzen 5 3600 6 núcleos 12 hilos

RAM: 16Gb 3200MHz CL15

GPU: Nvidia 1660 Ti

Software:

Hemos usado dos entornos. El primero, para entrenar las redes neuronales convolucionales aceleradas por GPU (PythonGPU):

Librería	Versión
python	3.7.1
numpy	1.19.1
keras	2.3.1
keras-gpu	2.3.1
tensorflow	2.0.0
tensorflow-gpu	2.0.0
scikit-learn	0.23.2
openCv	3.4.1
cuDNN	7.6.5

El segundo, en el que ajustábamos los clasificadores, lo hacíamos por CPU (PythonCPU), puesto que sklearn no admite el uso de GPU. El entorno tenía instaladas las mismas librerías y versiones, salvo keras-gpu, tensorflow-gpu y cuDNN, que no teníamos instaladas.

7.3 Fotografías usadas para la predicción

Las fotografías usadas para la predicción se pueden encontrar en:

- https://hipertextual.com/analisis/avengers-endgame
- https://www.bolsamania.com/cine/asi-han-cambiado-los-actores-de-fast-furious-casi-20-anos-despues/
- https://pixabay.com/es/photos/furioso-malestar-persona-mujer-2514031/
- https://pixabay.com/es/photos/lindo-sorprendido-mujeres-ni%C3%B1a-18927/
- https://pixabay.com/es/photos/l%C3%A1grimas-llanto-una-l%C3%A1grima-dolor-4551435/
- https://www.freepik.es/foto-gratis/chica-aterradora-ve-expresion-miedo-asustada-mantiene-manos-cerca-boca-tiene-expresion-asustada_10584635.htm#page=1&query=asustado&position=0
- https://pixabay.com/es/photos/hermosa-sonrisa-ni%C3%B1a-mujer-feliz-1274360/
- https://pixabay.com/es/photos/empresario-inicio-593358/
- https://www.freepik.es/foto-gratis/mujer-linda-asqueada-que-mira-manzana-aversion-aversion-haciendo-muecas_9473918.htm#page=1&query=asco&position=2

8. Bibliografía

[1] colaboradores de Wikipedia. (2020, 28 noviembre). *Sistema de Codificación Facial*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Sistema_de_Codificaci%C3%B3n_Facial

[2] A. (2020, 10 junio). *Cómo reconocer las 7 emociones básicas en la comunicación no verbal*. analisisnoverbal.com. https://www.analisisnoverbal.com/las-7-emociones-basicas-en-la-comunicacion-no-verbal/

[3] OpenCV: Histograms - 2: Histogram Equalization. (s. f.). OpenCV Tutorials. https://docs.opencv.org/master/d5/daf/tutorial_py_histogram_equalization.html

[4] Wikipedia contributors. (2020, 8 octubre). *Histogram equalization*. Wikipedia. https://en.wikipedia.org/wiki/Histogram_equalization

[5] Saha, S. (2020, 15 octubre). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6] *OpenCV: Cascade Classifier*. (s. f.). OpenCV. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[7] Simonyan, K., & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition*. ICLR-2015.

[8] Rodrigo, J. A. (2017, abril). *Máquinas de Vector Soporte (Support Vector Machines, SVMs)*. cienciadedatos.net. https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines#Support_Vector_Machines_para_m%C3%A1s_de_dos_clases

[9] Sreenivasa, S. (2020, 12 octubre). *Radial Basis Function (RBF) Kernel: The Go-To Kernel*. Medium. https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a

[10] *Random Forests Classifiers in Python*. (2018, mayo). DataCamp Community. https://www.datacamp.com/community/tutorials/random-forests-classifier-python#algorithm

[11] Cao, Q., Shen, L., Xie, W., Parkhi, O. M., & Zisserman, A. (2018). VGGFace2: A Dataset for Recognising Faces across Pose and Age. *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, 1-11. https://doi.org/10.1109/fg.2018.00020

[12] Brownlee, J. (2019, 19 agosto). *How to Get Reproducible Results with Keras*. Machine Learning Mastery. https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

[13] A. (2018, 16 octubre). *Tutorial : Facial Expression Classification Keras*. Kaggle. https://www.kaggle.com/ashishpatel26/tutorial-facial-expression-classification-keras

[14] Guo, S., Chen, S., & Li, Y. (2016). Face recognition based on convolutional neural network and support vector machine. *2016 IEEE International Conference on Information and Automation (ICIA)*, 1. https://doi.org/10.1109/icinfa.2016.7832107

[15] *Challenges in Representation Learning: Facial Expression Recognition Challenge | Kaggle*. (2013). Kaggle. https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

[16] Giannopoulos, P., Perikos, I., & Hatzilygeroudis, I. (2018). Deep Learning Approaches for Facial Emotion Recognition: A Case Study on FER-2013. *Smart Innovation, Systems and Technologies*, 1-16. https://doi.org/10.1007/978-3-319-66790-4_1

[17] Georgescu, M.-I., Ionescu, R. T., & Popescu, M. (2019). Local Learning With Deep and Handcrafted Features for Facial Expression Recognition. *IEEE Access*, *7*, 64827-64836. https://doi.org/10.1109/access.2019.2917266