

Reconocimiento de ojos en imágenes mediante técnicas de Machine Learning

Teodoro Rodrigues, João Víctor Valcarce Ríos, Diego
victor.teodoro@udc.es d.valcarce@udc.es

Abstract: Uno de los problemas de reconocimiento más clásico es aquel que aborda los ojos. En este paper haremos uso de redes neuronales, extracción de características de imágenes, operaciones morfológicas, momentos de Hu y de Zernike para atacar el problema, evaluando los resultados con diversas métricas y validación cruzada.

Palabras clave: K-Fold Crossvalidation, Hu-Moments, Zernike-Moments, advanced-features, eye detection.

1. Introducción

Uno de los ámbitos de la inteligencia artificial que se encuentra en mayor auge es la visión por computador; debido tanto a la mejora en prestaciones de los ordenadores modernos para ser capaces de tratar imágenes como por la importancia que estos procesos toman en nuestro día a día.

El caso que nos incumbe hace referencia a la detección de ojos en imágenes cotidianas, lo cual puede tener diversas aplicaciones. Por ejemplo podría ser la base para un sistema de autenticación vía iris, parte de un sistema para corregir ojos rojos en imágenes con flash o parte de otro que se encargue de, en función del estado de los ojos, determinar su cansancio para, por ejemplo, colocar una cámara en un vehículo y detectar si el conductor está en condiciones para la tarea que lo concierne.

2. Descripción del problema

Para aplicar las técnicas que serán descritas más adelante, es necesario conocer *a priori* qué requisitos se han utilizado a la hora de afrontar este problema; para el proceso de entrenamiento se han utilizado imágenes que contienen ojos para los positivos y otras partes de la cara para los negativos, de tamaños (en resolución) muy diversos.

2.1. Descripción de la base de datos

Las figuras [1][2] representan dos muestras de la base de datos. Cabe resaltar que en los ejemplos de negativos tenemos únicamente partes de la cara, por lo que a la hora de evaluar las imágenes de test, esta casuística se verá reflejada en los resultados.

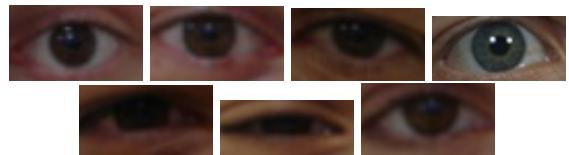


Figura 1: Ejemplos de positivos



Figura 2: Ejemplos de negativos

Para comprobar el funcionamiento de la red, disponemos de un conjunto de 12 imágenes de test, de las cuales se observa un subconjunto en la figura [3]. Tal y como se acaba de anticipar, nótese la presencia de elementos para los cuales no tenemos conjunto de entrenamiento (todas las zonas que no son cara).



Figura 3: Ejemplo imágenes de test

3. Materiales y recursos

Durante este artículo, utilizaremos varios lenguajes de programación: R (v4.0.3) para realizar la mayoría de las gráficas y Julia (v1.5) para ejecutar los procesos de entrenamiento y todo lo relacionado con tareas de computación exigentes. Todo ello será ejecutado en un Intel Core i9 con 32 GiB de RAM.

4. Análisis bibliográfico

Cuando se trata de problemas de clasificación con imágenes, la primera opción que se viene a la mente son las redes convolucionales, tal y como proponen en su trabajo Tivive and Bouzerdoum, 2005, con esta aproximación obtienen una red capaz de clasificar correctamente en el 99% de las ocasiones.

De forma distinta, Sabancı and Köklü, 2015, afrontan este mismo problema haciendo uso del algoritmo k -NN, que obtienen el mejor resultado con $k = 3$ con un 84% de accuracy. Existen otras aproximaciones, que se basan en la extracción de las características de las imágenes, como es nuestro caso; en esta rama es resaltable el trabajo de Kim and Kim, 2008, que hacen uso de los momentos de Zernike (se profundizará sobre este concepto [5.3]) y máquinas de soporte vectorial (SVM), con precisiones superiores al 90%.

5. Desarrollo

Durante las próximas aproximaciones, se tratará de resolver el problema propuesto enfocándolo desde diversas perspectivas. En la primera aproximación [5.1] se utilizarán

métodos estadísticos simples, mientras que en las siguientes aproximaciones se tratará de afrontar el problema desde otro punto de vista: localizar formas similares a los ojos en vez de estadísticas sobre los colores.

De esta forma, en la segunda aproximación [5.2] se introducirán técnicas para el reconocimiento de patrones en imágenes, mientras que en la tercera [5.3] se hará algo similar, pero esta vez se añadirán a mayores técnicas de reconocimiento de ojos exclusivas para las imágenes de test, al ser estas muy diferentes al conjunto de entrenamiento.

5.1. Aproximación 1

En esta primera aproximación, entrenaremos y analizaremos los resultados de reconocimiento de ojos con *features* simples: la media de los colores de la imagen, su desviación típica, su media armónica y la covarianza entre los canales RGB (definidas en [1], [2], [3] y [4], respectivamente).

$$\hat{\mu} = \frac{1}{N} \sum_{i=0}^N x_i \quad (1)$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=0}^N (x_i - \hat{\mu})^2} \quad (2)$$

$$\text{Media Armónica} = \frac{N}{\sum_{i=0}^N \frac{1}{x_i}} \quad (3)$$

$$\text{COV}(X, Y) = \frac{1}{N} \sum_{i=0}^N (x_i - \hat{\mu}_X)(y_i - \hat{\mu}_Y) \quad (4)$$

5.1.1. Análisis exploratorio

Con este procedimiento, obtenemos para cada ventana un total de 12 características. En el caso de la media, la figura [4] representa las sobresedimentación de las densidades de las medias de los canales RGB separadas por Ojos y No ojos (tonos más apagados y más brioso, respectivamente).

Esta gráfica puede interpretarse de la siguiente manera: los ojos tienen intensidades más bajas en media por canal que los no ojos; es decir, los Ojos están en media más presentes (tienen más área)

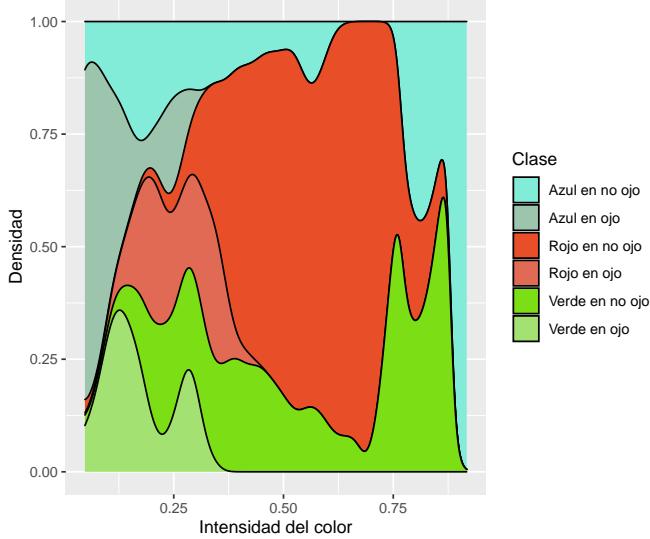


Figura 4: Sobresedimentación de las densidades de las medias de los canales para las clases Ojo y No ojo

en zonas con menor intensidad del color, mientras que los No ojos ocupan zonas más altas de dicha escala.

No obstante, es reseñable que aunque en el caso de los ojos las intensidades están acotadas a niveles bajos de intensidad (no tiene área en zonas altas), en el caso de los no ojos tenemos una distribución más presente a lo largo de todo el rango de intensidades (con menor presencia en la zona ocupada por los ojos, pero con cierta presencia).

Para el caso de la desviación típica [5] sucede prácticamente lo mismo, pero para los no ojos, la presencia a lo largo del rango de intensidades es mucho mayor, lo cual genera un solapamiento entre los distintos canales de las dos clases.

En este caso, hemos de destacar que la media armónica [6] produce resultados muy similares a la media usada anteriormente (la aritmética), aunque no del todo iguales (la media armónica presenta más colas a la izquierda en los canales rojo y verde que no están presentes en la aritmética), por lo que probaremos a utilizar esta característica por si las pequeñas diferencias que presenta respecto a la media aportan información relevante al modelo que vamos a entrenar.

Podemos observar en la gráfica [7], que las covarianzas generalmente no se distinguen

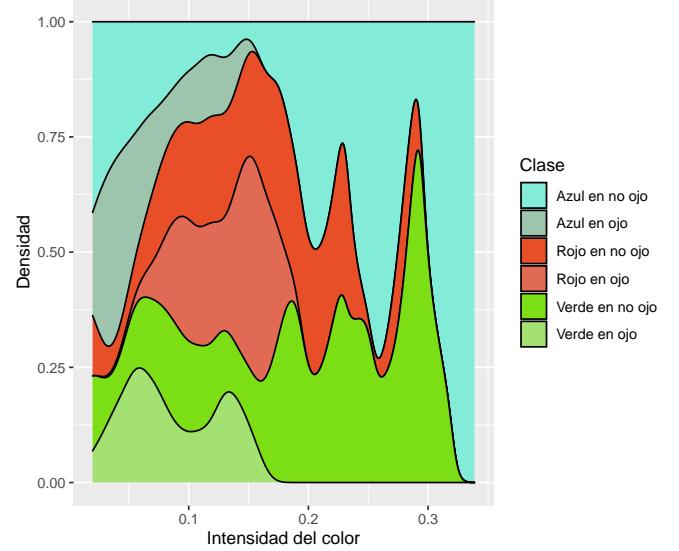


Figura 5: Sobresedimentación de las densidades de las desviaciones típicas de los canales para las clases Ojo y No ojo

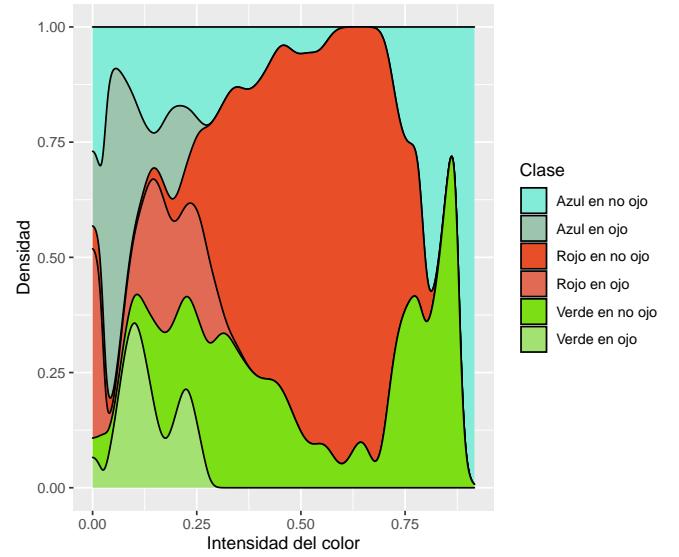


Figura 6: Sobresedimentación de las densidades de las medias armónicas de los canales para las clases Ojo y No ojo.

mucho entre sí, con la salvedad de que en el grupo de negativos hay mayor presencia de atípicos, presentando asimetría positiva. Esto en cierta forma es esperable, ya que al tratar con conjuntos de datos provenientes de fotos de la cara, aquellas fotos en las que haya mucha piel tenderán a presentar mayor covarianza que las que representan ojos.

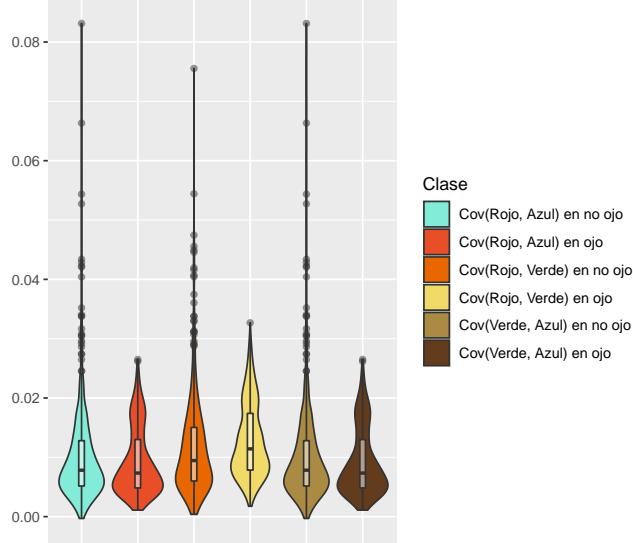


Figura 7: Violin plot de las covarianzas entre los canales RGB para las clases Ojo y No ojo.

5.1.2. Entrenamiento

En esta aproximación construiremos una Red de neuronas Artificial (ann) que tendrá 12 neuronas en la capa de entrada, 5 en la capa oculta y 1 en la capa de salida. Entrenaremos usando como función de error la entropía cruzada binaria y utilizando una tasa de aprendizaje de 0.01 con el algoritmo Adam, tal y como se muestra en [8].

Para el entrenamiento de la ann extraeremos las características anteriormente mencionadas de cada fotografía (media, desviación típica, media armónica de cada canal y covarianza entre canales) y las pasaremos como parámetros a nuestra red de neuronas.

Tras haber entrenado la red partiendo el conjunto de entrenamiento en evaluación y test obtenemos las métricas de la tabla [1]. Como podemos observar, obtenemos métricas muy buenas para unas características tan sencillas como las que hemos escogido para esta primera aproximación. Luego de esto procedemos a utilizar validación cruzada para evaluar que tan bueno es realmente el modelo.

5.1.3. Validación Cruzada

El modelo entrenado arroja las métricas que se muestran en la tabla [2] usando validación cruzada con 10 grupos.

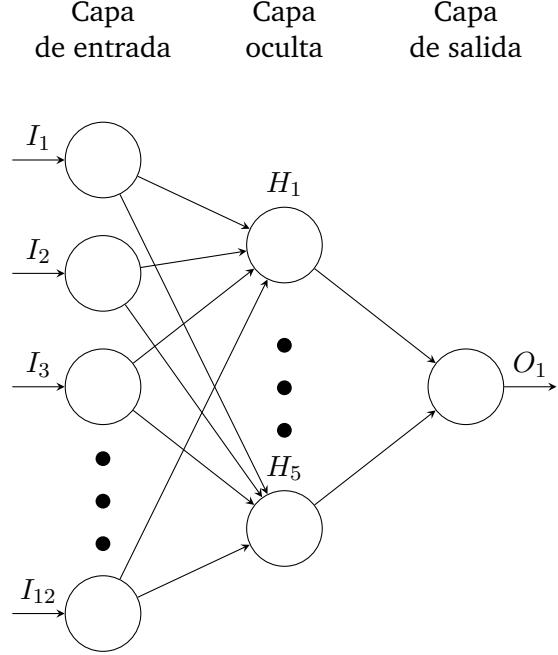


Figura 8: Red neuronal utilizada en la primera aproximación

Accuracy	Recall	F1	Specificity	NPV
0.914	0.916	0.897	0.844	0.921

Tabla 1: Métricas de una ann con la topología propuesta para la Aproximación I

	Accuracy	Recall	F1	Specificity	NPV
1	0.906	0.896	0.907	0.916	0.893
2	0.893	0.960	0.903	0.822	0.952
3	0.871	0.863	0.869	0.879	0.867
4	0.927	0.972	0.937	0.872	0.962
5	0.895	0.921	0.878	0.877	0.942
6	0.879	0.924	0.890	0.827	0.905
7	0.882	0.962	0.893	0.800	0.952
8	0.950	0.979	0.949	0.924	0.980
9	0.868	0.931	0.869	0.811	0.930
10	0.916	0.946	0.913	0.889	0.949
μ	0.899	0.935	0.901	0.862	0.933
σ	0.025	0.034	0.026	0.042	0.033

Tabla 2: Métricas de K -Fold Crossvalidation para la Aproximación I

Cabe destacar que se obtienen unos buenos resultados, pues todas las métricas dan valores superiores a un 85% y además la desviación típica

sobre las métricas obtenidas para las 10 folds es baste pequeña.

5.1.4. Resultados en postproducción

Una vez confirmado por validación cruzada que el modelo propuesto arroja buenos resultados, procedemos a entrenar el modelo final que se entregaría en la fase de postproducción. Este modelo se entrenará utilizando todos los datos disponibles, por lo que las únicas que podemos calcular son las métricas del propio conjunto con el que se ha entrenado el modelo, obteniendo los resultados de la tabla [3]. Nuevamente, se obtienen buenas métricas, y están entre los mismos rangos observados anteriormente.

Accuracy	Recall	F1	Specificity	NPV
0.907	0.911	0.807	0.825	0.901

Tabla 3: Métricas de una ann con la topología propuesta para la Aproximación I

Para probar cómo se comportaría el modelo en postproducción disponemos de 12 imágenes como las mostradas anteriormente en [3]. Lo que haremos será implementar un sistema de enventanado que cree ventanas de longitud variable y recorra la imagen cogiendo subconjuntos de píxeles. Se evaluará cada una de esas ventanas en el modelo, que las clasificará como “Ojo” o “No ojo”. Finalmente marcaremos con un recuadro rojo en la imagen original todas las ventanas que han dado positivo como ojo en nuestro modelo.

Tras aplicar este procedimiento con la ann entrenada en esta aproximación, obtenemos los resultados de la figura [9]. Cabe recalcar que el mayor problema que podemos observar en estos resultados es que el modelo clasifica como ojos bastante pelo, ropa y fondo de la imagen. Esto es más pronunciado en el ejemplo de la figura [10].

Realmente, esto es una consecuencia del conjunto de entrenamiento que recibe la red de neuronas, ya que en ningún momento ha recibido imágenes de ropa, objetos, paredes, etc. como para poder aprender a diferenciarlos de los verdaderos ojos. Además sólamente contamos con dos imágenes representando pelo en todo el conjunto de negativos, por lo que la capacidad del

modelo de discernir entre un verdadero ojo y una imagen con pelo es muy baja.

En cierta forma, esta incapacidad de distinguir entre pelo y ojo viene dado además porque las fotografías de ojos presentes en nuestro conjunto de entrenamiento son muy oscuras. Como hemos podido ver en [5.2.1] los ojos presentan generalmente poca intensidad de color (imágenes oscuras) y esto dificulta el diferenciarlos del pelo, que es oscuro en naturaleza.



Figura 9: Resultados de la Aproximación 1 para algunas fotos de test

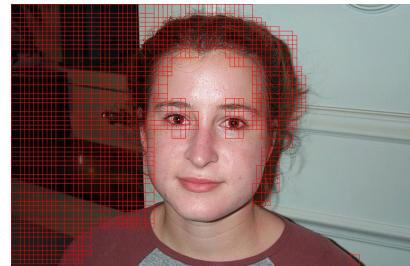


Figura 10: Ejemplo de imagen con mucho fondo como falso positivo

5.1.5. Discusión de los resultados

Como consecuencia de utilizar métricas simples, basadas únicamente en los colores, creamos modelos muy sensibles al ruido de la imagen y sobretodo, a imágenes que, aun siendo exageradamente distintas, tengan valores similares a nivel estadístico, esto es muy significativo en el caso de la desviación típica, ya que obtendrá el mismo valor en una imagen que es completamente roja que una que sea completamente blanca.

Para afrontar este problema que se ha comentado, se utilizarán en las siguientes secciones métricas que tengan en cuenta la forma de los ojos en vez de los colores de las imágenes.

5.2. Aproximación 2

Utilizando medidas estadísticas simples relacionadas con las distribuciones de los colores, como se ha observado en los resultados anteriores, da lugar a mucho margen de mejora; como consecuencia, trataremos en esta aproximación de entrenar con las formas presentes en la imagen en vez de con los colores *per se*.

Para ello, definiremos a continuación una serie de características relacionadas con las formas intrínsecas en la imagen.

Contraste

Como su nombre indica, nos da una medida del contraste de la imagen. Esto es, nos dice cuán sesgada está la distribución de la escala de grises de la imagen hacia el blanco o hacia el negro.

Para su cálculo hacemos uso del cuarto momento estandarizado de la imagen. Definimos el cuarto momento estandarizado como en estadística:

$$\alpha_4 = \frac{\mu_4}{\sigma^4} \quad (5)$$

Donde μ_4 es el cuarto momento univariante centrado en torno a la media y σ es la desviación típica de la imagen en escalas de grises.

Así pues obtenemos el contraste de la imagen dividiendo la desviación típica entre el cuarto momento estandarizado:

$$\text{Contraste} = \frac{\sigma}{\alpha_4} \quad (6)$$

Coarseness (tosquedad)

Está relacionada con la distancia en escala de grises de las variaciones espaciales. Calcula repeticiones y tamaños de patrones primitivos en la imagen, por lo que nos da una medida que nos permite discernir la presencia de distintas texturas en una imagen.

Su cálculo formal viene dado por la siguiente ecuación:

$$A_k(x, y) = \sum_{i=x-2k-1}^{x+2k-1} \sum_{j=y-2k-1}^{y+2k-1} \frac{I(i, j)}{2^{2k}} \quad (7)$$

Donde $2^k \cdot 2^k$ es el tamaño de una vecindad de píxeles en torno al píxel (i, j) .

Aunque realmente implementaremos el siguiente algoritmo, que nos permite entender mejor cómo funciona su cómputo:

Algoritmo

1. Para cada píxel $I(x, y)$ de una imagen calcularemos distintas submatrices de píxeles de la misma, $E_k(x, y) \in \mathcal{M}_{2^k \times 2^k}$ centradas en torno al píxel escogido que llamaremos escala de tamaño k . A lo largo de este algoritmo se usarán los valores $k = 0, 1, \dots, 5$.
2. Calcularemos las diferencias absolutas entre las esquinas de las matrices escala en diagonal.

Es decir, sean

- $A_k^1 = E_k(x, y)_{[1,1]}$
- $A_k^2 = E_k(x, y)_{[2^k, 2^k]}$
- $A_k^3 = E_k(x, y)_{[2^k, 1]}$
- $A_k^4 = E_k(x, y)_{[1, 2^k]}$

Construiremos:

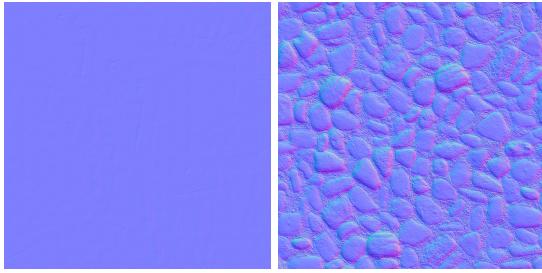
- $E_{k,a}(I) = |A_k^1 - A_k^2|$
- $E_{k,b}(I) = |A_k^3 - A_k^4|$
- 3. Sea $E_k = \max \{E_{k,a}(I), E_{k,b}(I)\}$.
- 4. Escogeremos el k correspondiente a la escala en la que haya más variación, es decir, tal que $k = \operatorname{argmax}_k \{E_0, E_1, \dots, E_5\}$ y calcularemos $S = 2^k$ como mejor tamaño de ventana para el píxel $I(x, y)$.
- 5. Finalmente, se calcula *coarseness* promediando S para todos los píxeles de la imagen.

Como ejemplo para ver los resultados de *coarseness*, procederemos a aplicar el algoritmo a dos mapas normales. Los mapas normales son un tipo de textura presente en el ámbito de la modelización 3D que permite dar rugosidad a los modelos. En esta ocasión compararemos la tosquedad del mapa normal de un metal (figura [11a]) y el de una roca (figura [11b]).

Para el metal, *coarseness* nos da un valor de 44, mientras que para la piedra de 54,7. Esto es lo esperado, ya que el mapa normal de la roca es más rugoso, más “tosco” que el del metal.

Momentos geométricos o espaciales

Sea $I(x, y)$ la intensidad de la imagen en el píxel (x, y) , definimos los momentos geométricos



(a) Mapa normal de un metal (b) Mapa normal de una roca

Figura 11: Mapas normales

o espaciales de una imagen de la siguiente forma:

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q \rho(x, y) dx dy \quad (8)$$

Los cuales pueden ser estimados empleando la siguiente fórmula (Donde N indica el ancho de la imagen en píxeles y M el alto):

$$m_{pq} = \sum_{i=0}^N \sum_{j=0}^M x_i^p y_j^q I(x_i, y_j) \quad (9)$$

Momentos centrales (bivariantes)

También haremos uso de los momentos centrales bivariantes. Sea $f(x, y)$ una función de densidad de una variable aleatoria bivariante, el momento central bivariante teórico de orden (p, q) es

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \mu_X)^p (y - \mu_Y)^q dx dy \quad (10)$$

En nuestro caso la versión adaptada para imágenes es:

$$\mu_{pq} = \sum_{i=0}^N \sum_{j=0}^M (x_i - \bar{X})^p (y_j - \bar{Y})^q I(x_i, y_j) \quad (11)$$

Momentos de hu

Haciendo uso de estas medidas (momentos geométricos y centrales bivariantes), podemos introducir la siguiente característica que utilizaremos en esta aproximación, los momentos introducidos por Hu, 1962.

Estos momentos tienen ciertas propiedades de mucho interés para nuestro modelo. Presentan

invarianza a rotaciones. Su estandarización viene dada por (Chaki and Dey, 2019):

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{\gamma}} \quad (12)$$

Donde:

$$\gamma = \frac{i, j}{2} + 1 \quad (13)$$

Los momentos de Hu son 7, y se definen como sigue:

$$I_1 = \eta_{20} + \eta_{02} \quad (14)$$

$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \quad (15)$$

$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \quad (16)$$

$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \quad (17)$$

$$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \quad (18)$$

$$I_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \quad (19)$$

$$I_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \quad (20)$$

El resultado que se obtiene de calcular cada uno de los momentos de Hu, presenta grandes diferencias de magnitud entre ellos, por lo que procederemos a transformarlos de la siguiente manera:

$$- sign(I_i) \cdot \log(|I_i|) \quad (21)$$

Usamos la función logarítmica al ser esta de continua y lentamente creciente, por lo que es adecuada para nuestro caso en particular.

En el caso de los 6 primeros momentos, son invariantes a la simetría, mientras que el

séptimo es capaz de distinguir entre imágenes reflejadas. Para entender las implicaciones que esto supone hemos decidido poner un ejemplo: en las imágenes [12][13] se muestran rotaciones, inversiones y escalamientos de dos imágenes, un ojo de Horus y el símbolo Ankh, respectivamente.

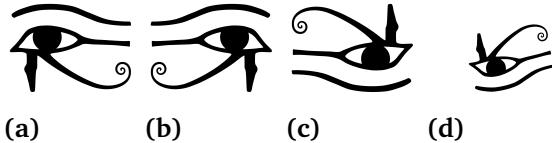


Figura 12: Rotaciones, inversiones y escalamientos sobre un ojo Wadjet (ojo de Horus)

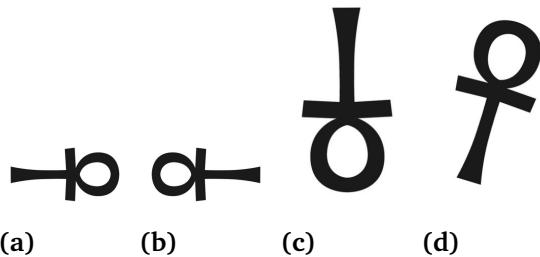


Figura 13: Rotaciones, inversiones y escalamientos sobre Ankh (símbolo de la vida)

Las tablas [4][5] contienen los valores obtenidos para cada momento de Hu dada una imagen de las mencionadas. Se observa que los momentos de Hu son capaces de encontrar diferencias entre las imágenes, y sin embargo, otorgar valores muy similares entre sí cuando se trata de la misma imagen. Nótese que las diferencias (en forma) entre el ojo y el símbolo no son tan excesivas, y aún así se encuentran las diferencias, por lo que queda justificado su uso en este problema que nos concierne.

5.2.1. Análisis exploratorio

Vamos a analizar las diferencias que genera entre las clases las features presentadas en la sección anterior. En el caso de los contrastes sobre los canales RGB individualmente (figura [14]), se obtienen conclusiones muy similares a las de la sección [5.2.1]: los ojos tienen contrastes similares y acotados entre canales, mientras que el resto de las imágenes de entrenamiento (los no

Momentos	Imágenes			
	[12a]	[12b]	[12c]	[12d]
$I_1[14]$	1.425	1.427	1.428	1.589
$I_2[15]$	5.66	5.674	5.665	5.841
$I_3[16]$	10.732	10.735	10.652	11.919
$I_4[17]$	9.242	9.17	9.161	10.625
$I_5[18]$	19.318	19.207	19.214	24.099
$I_6[19]$	12.265	12.183	12.232	-15.759
$I_7[20]$	20.137	-20.051	19.756	21.903

Tabla 4: Momentos de Hu para el ojo de Horus

Momentos	Imágenes			
	[13a]	[13b]	[13c]	[13d]
$I_1[14]$	1.371	1.371	1.371	1.415
$I_2[15]$	4.292	4.292	4.292	4.283
$I_3[16]$	10.126	10.126	10.125	10.48
$I_4[17]$	9.457	9.457	9.455	10.351
$I_5[18]$	19.248	19.248	19.246	21.052
$I_6[19]$	11.603	11.603	11.601	12.84
$I_7[20]$	-24.517	24.517	-24.532	21.184

Tabla 5: Momentos de Hu para el símbolo Ankh

ojos), como es de esperar, están más dispersas por todo el rango de contrastes.

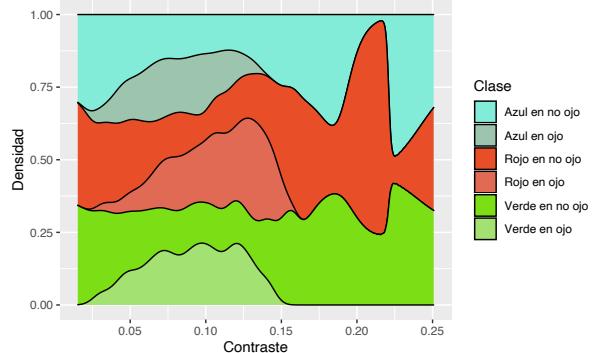


Figura 14: Densidades sobresedimentadas para los canales RGB

En el caso del coarseness, el análisis resulta más interesante: vemos comportamientos de las densidades que, en vez de solaparse, en las zonas donde una función crece, la otra decrece y así a lo largo de toda la gráfica [15].

Para los momentos Hu (6 y 7), figura [16], las diferencias no son tan claras, pero si existentes. Se observa que las densidades de los ojos están

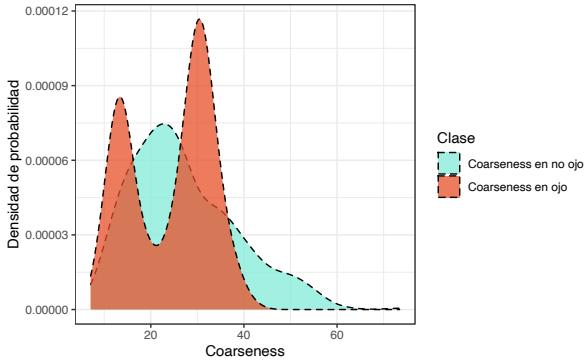
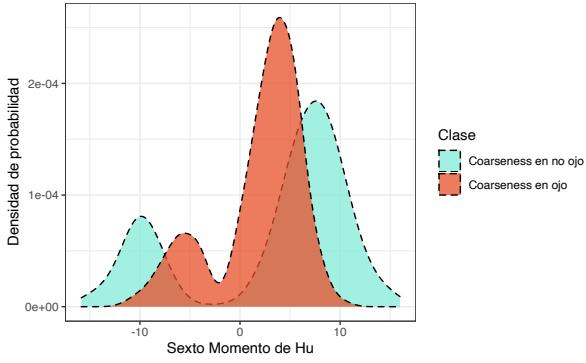
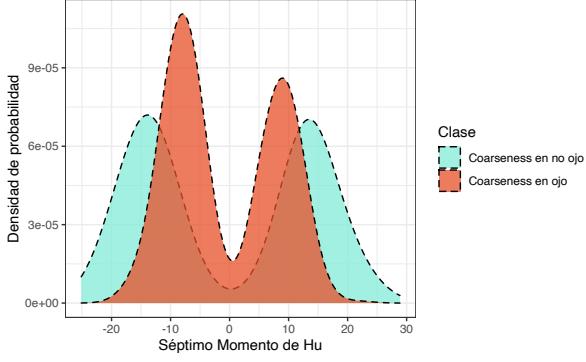


Figura 15: Superposición de las densidades para la variable coarseness

más concentrados en el centro de la gráfica, mientras que los no ojos están más dispersas hacia las colas de la misma.



(a) Momento Hu 6



(b) Momento Hu 7

Figura 16: Superposición de las densidades para los últimos momentos de Hu

5.2.2. Entrenamiento

Para el entrenamiento de la red hemos optado por topología de 5 neuronas en la capa oculta

[17], algoritmo Adam, con tasa de aprendizaje de 0.01 y una función de error de entropía cruzada binaria. Para el entrenamiento de la red le pasaremos como parámetros: el contraste por canal de la ventana, el coarseness de la imagen en blanco y negro y los momentos de Hu 6 y 7. Tras haber entrenado la red hemos obtenido las métricas que se muestran en la tabla [6].

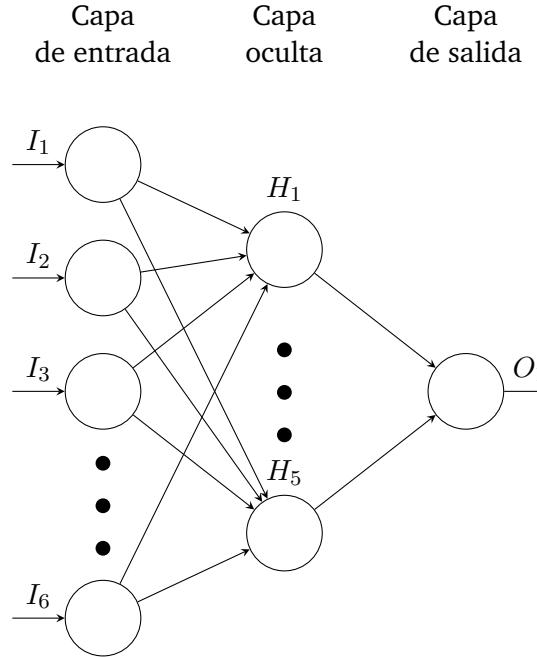


Figura 17: Red neuronal utilizada en la segunda aproximación

Accuracy	Recall	F1	Specificity	NPV
0.793	0.785	0.764	0.821	0.785

Tabla 6: Métricas de una ann con la topología propuesta para la Aproximación II

Para aumentar la robustez sobre el modelo propuesto, realizaremos validación cruzada en la siguiente sección [5.2.3].

5.2.3. Validación cruzada

Los resultados obtenidos mediante validación cruzada con la topología mencionada se muestra en la tabla [7].

Cabe destacar que los resultados son, en validación cruzada, sustancialmente peores que los obtenidos en la aproximación anterior [5.1.3],

	Accuracy	Recall	F1	Specificity	NPV
1	0.801	0.809	0.779	0.795	0.850
2	0.792	0.814	0.789	0.772	0.822
3	0.727	0.768	0.668	0.705	0.846
4	0.739	0.731	0.755	0.749	0.694
5	0.771	0.769	0.777	0.773	0.756
6	0.763	0.738	0.749	0.787	0.767
7	0.749	0.785	0.736	0.720	0.811
8	0.793	0.819	0.814	0.760	0.782
9	0.762	0.781	0.784	0.738	0.733
10	0.791	0.746	0.791	0.842	0.748
μ	0.769	0.776	0.764	0.764	0.781
σ	0.024	0.030	0.038	0.038	0.048

Tabla 7: Métricas de K -Fold Crossvalidation para la Aproximación II

no obstante, como se comprobará en la siguiente sección [5.2.4], los resultados en test, son ligeramente mejores que los obtenidos en la anterior aproximación [5.1.4].

5.2.4. Resultados en postproducción

Tras realizar validación cruzada y comprobar la robustez de nuestro modelo, procedemos a entrenar una nueva red que ahora disponga de todos los datos. Las métricas obtenidas con este nuevo conjunto de entrenamiento se muestran en la tabla [8]. Estas métricas son bastante similares a las obtenidas en validación cruzada, presentando unos resultados buenos comparados con la mayoría de resultados obtenidos en las folds.

Accuracy	Recall	F1	Specificity	NPV
0.783	0.774	0.756	0.810	0.797

Tabla 8: Métricas de una ann con la topología propuesta para la Aproximación II

Un subconjunto de los resultados obtenidos en las imágenes de test se muestran en la figura [18]. Se observa una ligera mejora en cuanto a la detección de ruido en las zonas no referidas a la cara, pero el resultado sigue sin ser el óptimo.

5.2.5. Discusión

Aunque los resultados son mejores que en la anterior aproximación (para el conjunto de



Figura 18: Resultados de la aproximación 2 para algunas imágenes de test

test), todavía distan mucho de ser ideales. Por esta razón trataremos de sofisticar las features empleadas sobre la detección de formas, introduciendo para ello los momentos de Zernike.

5.3. Aproximación 3

Introduciremos unos momentos (más) complejos (momentos de Zernike) y, añadiremos una capa de preprocessado en las imágenes de test para obtener mejores resultados y mejorar la eficiencia computacional.

Momentos de Zernike

- Polinomio radial R_{nm}

$$R_{nm}(\rho) = \sum_{s=0}^{(n-|m|)/2} (-1)^s \frac{(n-s)!}{s! \left(\frac{n+|m|}{2} - s\right)! \left(\frac{n-|m|}{2} - s\right)!} \rho^{n-2s} \quad (22)$$

- Polinomio de Zernike

$$V_{nm}(x, y) = R_{nm} \exp(jm\theta) \quad (23)$$

- Momento de Zernike

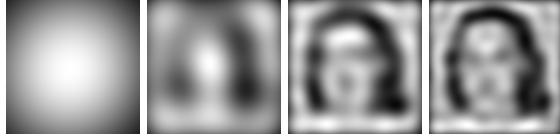
$$A_{mn} = \frac{n+1}{\pi} \sum_{x=1}^N \sum_{y=1}^N I(x, y) [V_{mn}(x, y)]^* \quad (24)$$

Si disponemos de todos los momentos A_{nm} de la imagen hasta un orden n_{max} podremos hacer una reconstrucción $\hat{f}(x, y)$ de la imagen como comentan de la siguiente forma:

$$\hat{f}(x, y) \approx \sum_{n=0}^{n_{max}} \sum_m A_{nm} V_{nm}(\rho, \theta) \quad (25)$$

Como es natural, a medida que subamos el orden de los momentos de Zernike, mejor será la reconstrucción de la imagen.

Podemos ver un ejemplo que ilustra los resultados de la reconstrucción de una imagen a medida que sube el orden máximo que escogemos para los momentos de Zernike en la figura [19].



(a) $g = 5$ (b) $g = 15$ (c) $g = 30$ (d) $g = 40$

Figura 19: Reconstrucción con los momentos de Zernike

La imagen original correspondiente a la reconstrucción se puede observar en la imagen [10]. Nótese la similitud que ya existe con grado igual a 30. Por motivos computacionales, no se muestran grados superiores, pero la reconstrucción sería cada vez más pareja (en escala de grises).

Algoritmo EyeMap

Se aplicará Zernike sobre una imagen tanto en escala de grises como binarizada. Para binarizar las imágenes vamos a usar el algoritmo EyeMap; nos basaremos en el trabajo de Nasiri et al., 2008. Procederemos como sigue:

1. El primer paso consiste en calcular la componente “EyeMapC”, que parte del espacio de colores YCbCr y aprovecha que los ojos se caracterizan por ser zonas con altos valores del canal Cb y bajos del canal Cr. Se computa como se muestra en la ecuación [26].

$$\text{EyeMapC} = \frac{1}{3} \left[C_b^2 + C_r^2 + \frac{C_b}{C_R} \right] \quad (26)$$

2. Además los ojos se caracterizan por tener tanto zonas con tonos blancos como negros, por lo que se puede explotar esta característica. Para ello se hace uso de las operaciones morfológicas de dilatación [27] y erosión [28].

$$(f \oplus g)(x) = \sup_{t \in G \cap \mathcal{D}_{-x}} \{f(x - t) + g(t)\} \quad (27)$$

$$(f \ominus g)(x) = \inf_{t \in G \cap \mathcal{D}_{-x}} \{f(x + t) - g(t)\} \quad (28)$$

Siendo:

$$f : \mathcal{D} \subset \mathbb{R}^n \rightarrow \mathbb{R}$$

$$g : \mathcal{G} \subset \mathbb{R}^n \rightarrow \mathbb{R}$$

Partiendo de estas definiciones, podemos resaltar las zonas con posibles ojos “filtrando” la imagen con la siguiente operación [29]:

$$\text{EyeMapL} = \frac{Y(x, y) \oplus g(x, y)}{Y(x, y) \otimes g(x, y)} \quad (29)$$

Donde $Y(x, y)$ es la imagen, y $g(x, y)$ es una función de estructuración que jugará un papel clave a la hora de detectar (o no) las zonas de interés (Jackway and Deriche, 1996). Para nuestro caso, hemos utilizado una función en forma de disco, por relación obvia con la forma del ojo.

3. El último paso del algoritmo reside únicamente en aplicar una operación lógica *and* [30] para comprobar que se cumplen las dos restricciones propuestas.

$$\text{Eye Map} = \text{EyeMapC} \wedge \text{EyeMapC} \quad (30)$$

Las implicaciones que esto provoca sobre nuestro conjunto de test se pueden observar en la imagen [20].

5.3.1. Análisis exploratorio

Comprobaremos las implicaciones que tiene, a la hora de separar las clases, las características que hemos comentado. Vamos a analizar los resultados que se obtienen de comparar un momento Zernike, en cuanto a su parte real y a su parte imaginaria.

Para el caso de la parte real (figura [21]), se observa la casuística que se ha comentado a lo largo de todo este documento: en el caso de los ojos, es cierto que tenemos los rangos acotados, y son frecuentes en comparación de los no ojos, sin embargo, en el caso de los no ojos, como es de esperar, está repartido aleatoriamente sobre todo el rango; por este motivo, este tipo de variables no es válida como un separador lineal de ambas clases.

En el caso de la parte imaginaria (figura [22]), sucede otra vez lo mismo. Aunque por separado

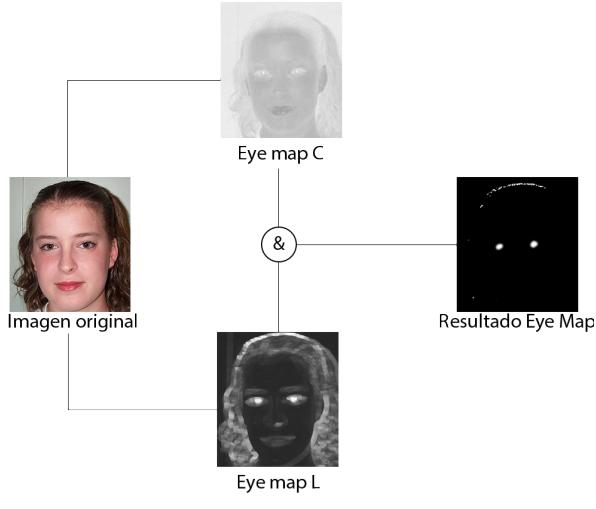


Figura 20: Funcionamiento algoritmo EyeMap

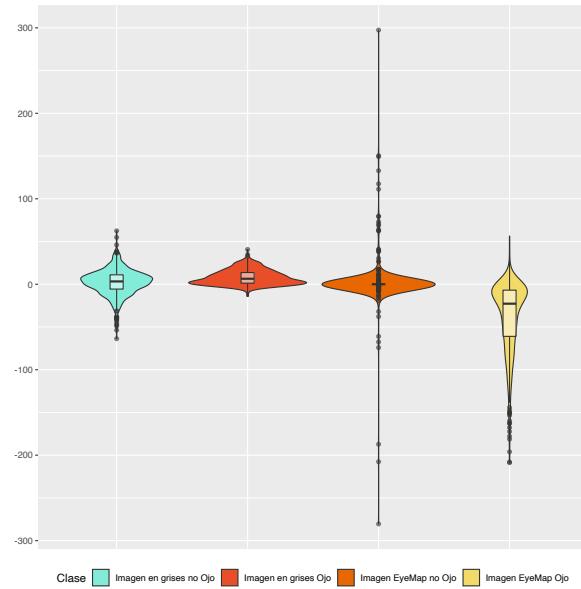


Figura 21: Violin plot de la parte real de un momento Zernike

puede que no funcionen bien, lo que hay que tener en cuenta es su funcionamiento como conjunto, que es lo que se analizará en secciones posteriores.

5.3.2. Entrenamiento

Para entrenar el modelo hemos calculado los momentos de Zernike hasta orden 5 usando radio 20 píxeles para las imágenes en escala de grises

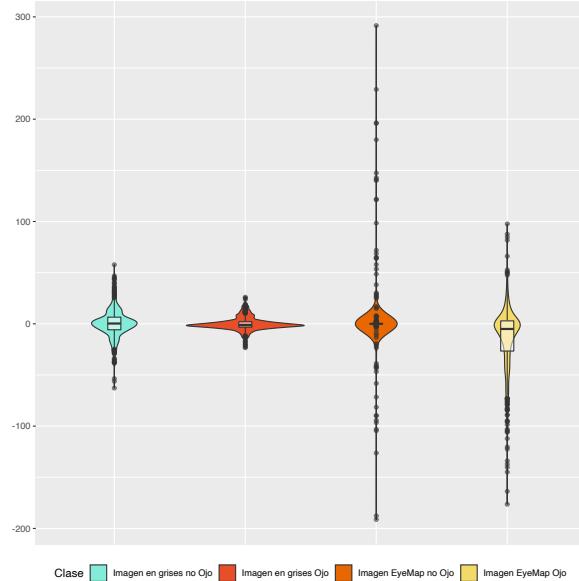


Figura 22: Violin plot de la parte imaginaria de un momento Zernike

y orden 5 y radio 30 para las imágenes tras aplicarles el algoritmo de Eye Map. Las features que hemos escogido son:

- La parte real de cada momento
- La parte imaginaria
- La media armónica entre la parte real del momento sobre la imagen original y la imagen con EyeMap.
- La media armónica entre la parte imaginaria del momento sobre la imagen original y la imagen con EyeMap.

Hemos utilizado una RNA con 126 neuronas en la capa de entrada (tantas como features tenemos), dos capas ocultas de 100 y 50 neuronas respectivamente, y finalmente 1 neurona en la capa de salida. Escogimos un Adam con learning rate de 0.08.

5.3.3. Validación cruzada

Como en aproximaciones anteriores, procederemos a evaluar el comportamiento de la topología escogida mediante validación cruzada; los resultados son los que se muestran en la tabla [9].

En este caso, los resultados son mejores que los obtenidos en la aproximación 2 [5.2.3], pero ligeramente peores todavía que las obtenidas en la primera aproximación [5.1.3]. No obstante,

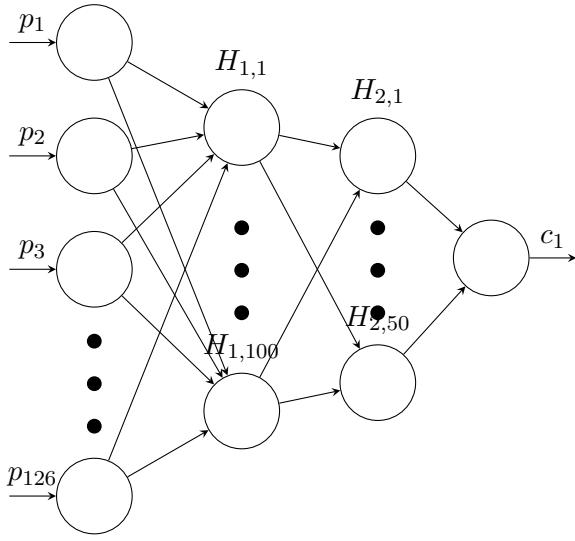


Figura 23: Red neuronal artificial densa con dos capas ocultas.

	Accuracy	Recall	F1	Specificity	NPV
1	0.850	0.898	0.794	0.880	0.866
2	0.892	0.868	0.918	0.883	0.887
3	0.872	0.947	0.818	0.963	0.861
4	0.884	0.881	0.888	0.883	0.885
5	0.881	0.888	0.875	0.908	0.871
6	0.858	0.867	0.851	0.894	0.847
7	0.870	0.822	0.926	0.830	0.868
8	0.866	0.894	0.842	0.907	0.860
9	0.875	0.907	0.836	0.889	0.889
10	0.860	0.932	0.798	0.934	0.860
μ	0.871	0.890	0.854	0.897	0.869
σ	0.012	0.033	0.044	0.033	0.013

Tabla 9: Métricas de K -Fold Crossvalidation para la Aproximación III

con las técnicas que se presentarán en la sección de postproducción [5.3.4], los resultados de test serán claramente superiores a los obtenidos en cualquiera de las aproximaciones anteriores.

5.3.4. Resultados en postproducción

Nuevamente, luego de hacer validación cruzada y comprobar la robustez de nuestro modelo, procedemos a entrenar una nueva red con todos los datos. Las métricas obtenidas con este nuevo conjunto de entrenamiento se muestran en la tabla [10]. Estas métricas muy buenas, incluso mejores que las obtenidas en validación cruzada, lo cual era de esperar, ya que

estamos evaluando con el mismo conjunto con el que entrenamos.

Accuracy	Recall	F1	Specificity	NPV
0.867	0.963	0.861	0.732	0.952

Tabla 10: Métricas de una ann con la topología propuesta para la Aproximación III

Asimismo, al evaluar el modelo usando un umbral de 0.5 para la clasificación como ojo, obtenemos la matriz de confusión de la figura [24]. Sobre esta matriz de confusión (y como habíamos observado anteriormente en las métricas), cabe destacar que obtenemos valores altos para los verdaderos positivos y negativos (488 y 380 respectivamente). También obtenemos valores bajos para los falsos negativos (19), es decir, es poco probable que nuestro modelo clasifique como “no ojo” cuando realmente sí es un ojo. Sin embargo, algo que podemos ver también en la matriz de confusión, y que por los resultados obtenidos en anteriores aproximaciones nos es de especial interés combatir, es que obtenemos un número relativamente alto de falsos positivos, i.e. la red tiene una leve tendencia a clasificar como ojos cosas que realmente no lo son.

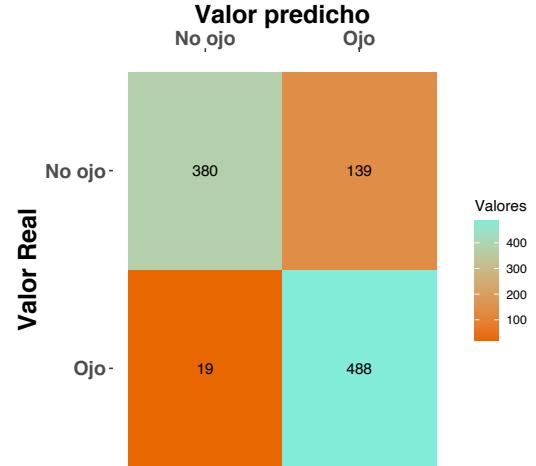


Figura 24: Matriz de confusión ($threshold = 0.5$)

En vista de esto, procederemos a realizar una curva ROC (figura [25]), para escoger un mejor $threshold$ que nos proporcione un mejor

compromiso entre la tasa de verdaderos positivos y la de falsos positivos.

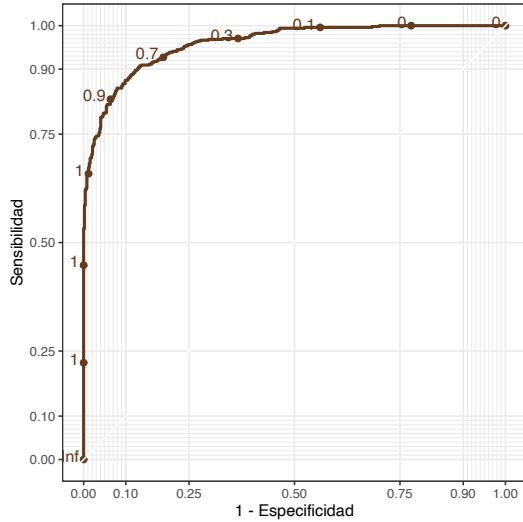


Figura 25: Curva ROC para el modelo

Hemos decidido escoger un *threshold* de 0.7, ya que es el que nos garantiza conservar una sensibilidad superior al 90%, mientras que reducimos la tasa de falsos positivos con respecto al *threshold* de 0.5. La matriz de confusión calculada utilizando este umbral se puede ver en la figura [26]. Como era de esperar, hemos reducido ligeramente los verdaderos positivos y aumentado los falsos negativos, pero a cambio hemos conseguido aumentar los verdaderos negativos y mitigar un poco la cantidad de falsos positivos, que era nuestro objetivo.

Además de construir un modelo menos sensible al ruido, hemos optado por llevarlo un paso más allá teniendo en cuenta también las imágenes de test. El principal problema al que nos tenemos que entrenar con las imágenes de test es al ruido del fondo de la imagen, para el cual no existe un conjunto de entrenamiento, por ello vamos a construir un filtro que facilite este trabajo. Haremos uso para ello del algoritmo EyeMap presentado en el inicio de la sección [5.3], con la diferencia de que ahora, tenemos una imagen completa sobre la que queremos construir ventanas para evaluar nuestra red. Para afrontar este problema aplicaremos el siguiente algoritmo de agrupación:

1. Guardamos las coordenadas de los píxeles blancos de la imagen en una colección de

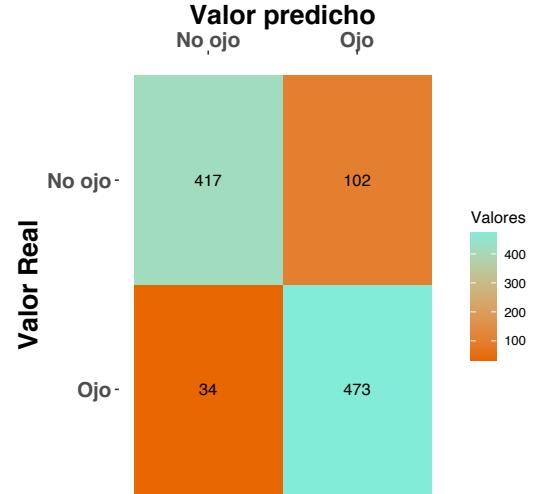


Figura 26: Matriz de confusión (*threshold* = 0.7)

datos, una lista por ejemplo.

2. En cada iteración del algoritmo cogemos un píxel cualquiera de esa lista y creamos un clúster.
3. Para las coordenadas escogidas (x, y) comprobamos los píxeles que tiene a su alrededor. Esto es, comprobamos los píxeles correspondientes a las coordenadas $(x + i, y + j) \forall i, j \in \{-1, 0, 1\}$. Para cada píxel blanco, añadimos sus coordenadas al clúster, las retiramos de la colección de datos que contiene las coordenadas de los píxeles blancos y repetimos esta iteración, buscando los nuevos vecinos de cada una de estas nuevas coordenadas vecinas.

De esta forma vamos creando clústeres recursivamente. Una vez creados estos clústeres de coordenadas, procederemos a calcular el centroide de cada uno (esto es, el punto medio de los mismos) y crearemos una ventana centrada en ese punto, formando así las ventanas que necesitamos para evaluar nuestro modelo en la fase de postproducción. El usar este procedimiento nos lleva a la primera de las ventajas de usar este algoritmo en vez de un bucle cuádruple, como en las aproximaciones anteriores:

1. A la hora de construir ventanas para evaluar la ann, no depende de la imagen para construir ventanas con buenos resultados: al ir generando la ventana se crea el

riesgo de que ninguna ventana contenga perfectamente los ojos, mientras que con esta técnica se garantiza que, si se cumplen las características esenciales del ojo, siempre habrá, como mínimo, una ventana centrada y del tamaño adecuado para cada ojo de la imagen (aunque esto pueda suponer la introducción de ruido).

2. Tiempos de ejecución. La tabla [11] muestra los tiempos de ejecución de la creación de las ventanas.

	Fuerza bruta	Eye Map
Tiempo de ejecución(s)	0.726	2.18

Tabla 11: Tiempo de generación de las ventanas para las 12 imágenes de test

En una primera instancia, no justifica el uso del algoritmo descrito anteriormente. No obstante, este tiempo es para la generación de las ventanas que posteriormente deberá evaluar la red. Es en este punto donde puede resultar interesante por motivos de eficiencia.

	Fuerza bruta	EyeMap
Nº ventanas	1600	15

Tabla 12: Comparación de algoritmo por fuerza bruta

La tabla [12] muestra el número de ventanas que genera cada uno de los algoritmos propuestos; teniendo en cuenta que el tiempo de evaluación de la red por cada una de las ventanas es de, aproximadamente, 100 microsegundos, esto coloca al algoritmo EyeMap en un beneficio absoluto de 0.15 segundos por imagen, lo que implica, que en el conjunto de test es, aproximadamente 1.7 segundos más rápido.

No obstante, estos resultados debemos penalizarlos por el tiempo de creación de las ventanas, por lo que el resultado final obtenido, relacionado con la eficiencia de los algoritmos es el mostrado en la tabla [13].

Filtrado de tonos de piel

EyeMap tiene un problema y es su sensibilidad al ruido, al “recoger” la información de blancos.

	Fuerza bruta	EyeMap
Tiempo ejecución (s)	2.68	2.19

Tabla 13: Comparación de algoritmo por fuerza bruta con EyeMap

Por este motivo, añadiremos una capa más de filtrado, que busque tonos de piel, tal y como realizó Tuba et al., 2014. Este procedimiento se basa una vez más en aplicar un filtro sobre el espacio de color YCbCr, concretamente los siguientes:

$$\begin{array}{lll} 69 & < Y & < 215 \\ 94 & < Cb & < 255 \\ 136 & < Cr & < 179 \end{array}$$

Hemos realizado modificaciones de los rangos sobre los propuestos para un funcionamiento más genérico (funciona con tonos de piel más diversos). Para entender las implicaciones que este filtro supone, lo aplicaremos sobre la misma imagen que la mostrada usando EyeMap (figura [20]). El resultado de aplicar este simple algoritmo lo observamos en la imagen [27].



Figura 27: Resultado de filtrar tonos de piel

Si juntásemos ambos resultados (EyeMap y el filtrado de tono de piel), obtendríamos el resultado de la figura [28].

No obstante, este procedimiento no es perfecto, como se observará en las imágenes de test, ya que sigue siendo sensible al ruido, pero en este caso, es capaz de filtrar mayoritariamente el ruido del fondo de las imágenes.

Los resultados obtenidos para todo el conjunto de test aplicando este preprocesado junto a nuestra red neuronal lo podemos observar en la figura [29].

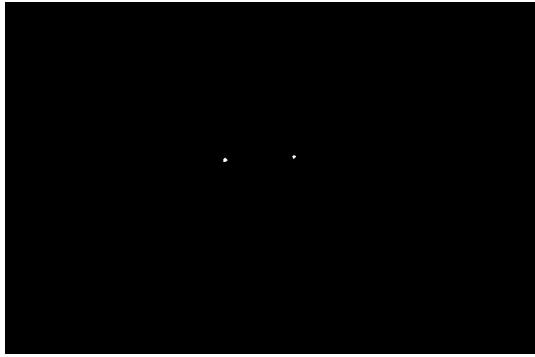


Figura 28: Resultado de filtrar mediante tono de piel y EyeMap

5.3.5. Discusión

Los resultados son claramente mejores que los obtenidos en ocasiones anteriores, ya que observamos en todas las imágenes que mayoritariamente se seleccionan ambos ojos y, también en la mayoría de las imágenes, se hace sin seleccionar ruido de fondo.

6. Conclusiones

Podemos concluir que el mayor beneficio a la hora de evaluar las imágenes de test se da gracias a realizar un preprocesado de las imágenes de test.

Es probable que, utilizando las mismas técnicas de preprocesado se lleguen a resultados similares con aproximaciones radicalmente distintas, esto es debido a que el reconocimiento de los ojos se hace realmente en la parte de preprocesado del test, ya que con las imágenes de entrenamiento, no fuimos capaces de construir ningún modelo que, de por sí solo, fuese capaz de equiparar los resultados obtenidos con la última de las aproximaciones.

7. Trabajo futuro

En futuras aproximaciones, podría tratarse de afrontar este mismo problema con perspectivas más modernas e innovadoras, como por ejemplo usando técnicas de *Deep Learning*. Para ello deberíamos prestar especial atención a las dimensiones de las imágenes.

Una vía que queda abierta después de este trabajo, es la implementación de SVM sobre los momentos de Zernike o incluso árboles de



Figura 29: Resultados de la aproximación 3 para todas las imágenes de test

decisión, ya que estos, podrían ser útiles a la hora de partir el espacio dadas las *features* que hemos empleado.

Tabla de contenidos

1	Introducción	1
2	Descripción del problema	1
2.1	Descripción de la base de datos	1
3	Materiales y recursos	2
4	Análisis bibliográfico	2
5	Desarrollo	2
5.1	Aproximación 1	2
5.1.1	Análisis exploratorio	2
5.1.2	Entrenamiento	4
5.1.3	Validación Cruzada	4
5.1.4	Resultados en postproducción	5
5.1.5	Discusión de los resultados	5
5.2	Aproximación 2	6
5.2.1	Análisis exploratorio	8
5.2.2	Entrenamiento	9
5.2.3	Validación cruzada	9
5.2.4	Resultados en postproducción	10
5.2.5	Discusión	10
5.3	Aproximación 3	10
5.3.1	Análisis exploratorio	11
5.3.2	Entrenamiento	12
5.3.3	Validación cruzada	12
5.3.4	Resultados en postproducción	13
5.3.5	Discusión	16
6	Conclusiones	16
7	Trabajo futuro	16
	Referencias	17

Referencias

- Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *IRE transactions on information theory*, 8(2), 179–187.
- Jackway, P. T., & Deriche, M. (1996). Scale-space properties of the multiscale morphological dilation-erosion. *IEEE transactions on pattern analysis and machine intelligence*, 18(1), 38–51.
- Tivive, F. H. C., & Bouzerdoum, A. (2005). A fast neural-based eye detection system. *2005 International Symposium on Intelligent Signal Processing and Communication Systems*, 641–644.
- Kim, H.-J., & Kim, W.-Y. (2008). Eye detection in facial images using zernike moments with svm. *ETRI journal*, 30(2), 335–337.
- Nasiri, J. A., Khanchi, S., & Pourreza, H. R. (2008). Eye detection algorithm on facial color images. *2008 Second Asia International Conference on Modelling & Simulation (AMS)*, 344–349.
- Tuba, M., Capor-Hrosik, R., & Vukovic, M. (2014). Face detection based on invariant moments classified by neural network. *International Journal of Circuits, Systems and Signal Processing*.
- Sabancı, K., & Köklü, M. (2015). The classification of eye state by using knn and mlp classification models according to the eeg signals.
- Chaki, J., & Dey, N. (2019). *A beginner's guide to image shape feature extraction techniques*. CRC Press.