DETECCIÓN DE SEÑALES DE TRÁFICO.

Tratamiento Digital de Imágenes

Francisco José López Carrillo
3º Ingeniería Informática TIC

Contenido

Introducción.

En esta práctica voy a tratar de usar filtros vistos en la asignatura para tratar imágenes con señales de tráfico que podemos encontrarnos por carretera o por la calle.

En principio, esta no sería la forma más recomendable, y hay una cantidad enorme de información que da para TFG, es por ello por lo que no me voy a centrar en la visión artificial y tampoco voy a pararme en los problemas que podíamos tener al procesar imágenes o videos de señales, sino que el trabajo principal será dar una introducción extensa a como sería aplicar diferentes filtros para la detección y traducción de señales para futura extensión del asunto si fuera necesario.

Motivación.

El principal motivo por el que desarrollo este trabajo es porque lo principal o lo más notorio de la visión artificial es trasladarse al mundo de los automóviles y en su conducción autónoma, es por ello por lo que este tema tiene infinidad de puntos de vista y de formas de resolverse y me gustaría añadir mi granito de arena.

Imágenes de uso.

Para el trabajo, voy a utilizar las siguientes imágenes:

Figura 1: Limite100.jpg

Figura 2: Bicis.jpg

Figura 3: SenalLetras.jpg

Figura 4: Limite90.jpg

Software utilizado.

Para la realización de la práctica usaré Matlab. Todos los códigos están alojados en https://github.com/pcoloc/Matlab-Filters

Weka para el análisis de datos del datasheet CIFAR-10.

Aplicado de filtros.

Imagen en escala de grises.

El problema que tenemos muchas veces es que no podemos procesar una imagen correctamente si la tenemos en color, es por ello por lo que lo primero que vamos a hacer es transformarla a escala de grises para que los siguientes procesos den mejores resultados.

Lo que esta fórmula hace es transformar el número que forman el color de los pixeles a uno que esté en la escala de grises (0 a 255)

El código de Matlab es el siguiente:

```
[x, map] = imread('limite100.jpg');
newGray = rgb2gray(map);
```

Imagen en negativo.

Para darle el valor negativo lo que tenemos que hacer es invertir sus pixeles al antagónico de su gama, con esto podemos conseguir que los contrastes en las imágenes sean más claros o que consigamos ver detalles que anteriormente no podíamos ver con claridad. En las siguientes imágenes vemos el uso de del filtro negativo para la imagen en color (primera imagen) y para la imagen en blanco y negro.

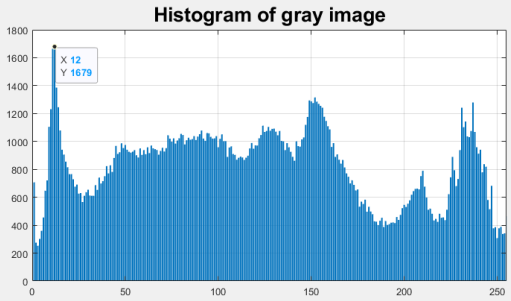
Figura 5. Limite.jpg en invertido con color.

Figura 6. Limite100.jpg invertido en blanco y negro.

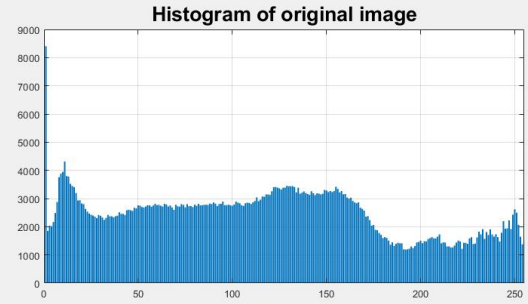
Histograma.

Un histograma es una representación de la distribución del color en una imagen, representa el número de píxeles que tienen colores en cada una de las listas de rangos de colores. El histograma puede ser construido por cualquier espacio de color.

Las desventajas de los histogramas para clasificar es que la representación depende del color del objeto, ignorando su forma y textura.




```
% Just for fun, let's get its histogram and display it.
[pixelCount, grayLevels] = imhist(grayImage);
subplot(2, 2, 2);
bar(pixelCount);
title('Histogram of gray image', 'FontSize', fontSize);
xlim([0 grayLevels(end)]); % Scale x axis manually.
grid on;
```




```
% Just for fun, let's get its histogram and display it.
[pixelCount, colorLevels] = imhist(colorImage);
subplot(2, 2, 4);
bar(pixelCount);
title('Histogram of original image', 'FontSize', fontSize);
xlim([0 colorLevels(end)]); % Scale x axis manually.
grid on;
```

Ajuste de contraste.

Aumentamos el contraste de una imagen asignado los valores de la imagen e intensidad de entrada a nuevos valores consiguiendo que se sature a intensidades altas y bajas de los datos de entrada.

Ecualización del histograma es otra de las formas que nos permite la mejora en el contraste de una imagen. Esto crea una distribución del contraste uniforme.

Realizamos una ecualización del histograma de forma adaptativa del contraste limitado. A diferencia de los filtros anteriores, este funciona en regiones de datos pequeñas en lugar

de toda la imagen. La mejora del contraste se puede limitar par evitar amplificar el ruido que podría estar presente en la imagen.

Ruido

El ruido es uno de los problemas que nos podemos encontrar en las imágenes, ya sea porque ha tenido una mala captura, por pérdidas de calidad o porque se nos ha colado ruido después de múltiples tratamientos.

Sal y pimienta.

Este tipo de ruido se caracteriza por cubrir toda la imagen, suele producirse cuando la señal es afectada por intensas o repentinas perturbaciones o impulsos. Para reducir este ruido solemos utilizar el filtro de la mediana. Podemos aplicarlo tanto a imágenes en Color como en Blanco y negro como vemos a continuación.

Color Image with Salt and Pepper Noise

Restored Image

Image with Salt and Pepper Noise

Realce de detalles.

Estos filtros los utilizamos para cuando queremos realzar partes de las imágenes que están en un primer plano.

Filtro paso alto.

Atenúa las frecuencias bajas manteniendo las frecuencias altas. Estas frecuencias corresponden a cambios bruscos en la densidad, es por ello por lo que usamos este filtro cuando queremos realzar los bordes de las imágenes, reforzando así los detalles más importantes.

El primer filtro lo utilizamos con la siguiente matriz:

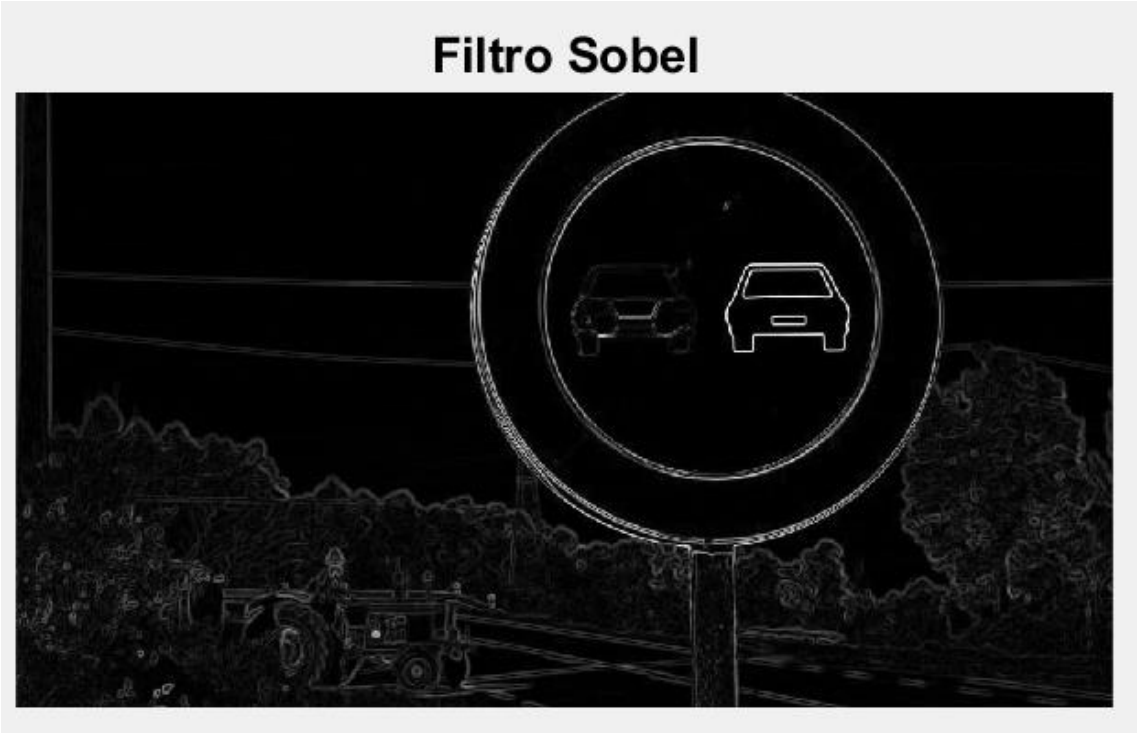
```
kernel1 = -1 * ones(3)/9;
kernel1(2,2) = 8/9;
```

Y este filtro lo usamos con la matriz siguiente:

```
kernel2 = [-1 -2 -1; -2 12 -2; -1 -2 -1]/16;
```

Sobel.

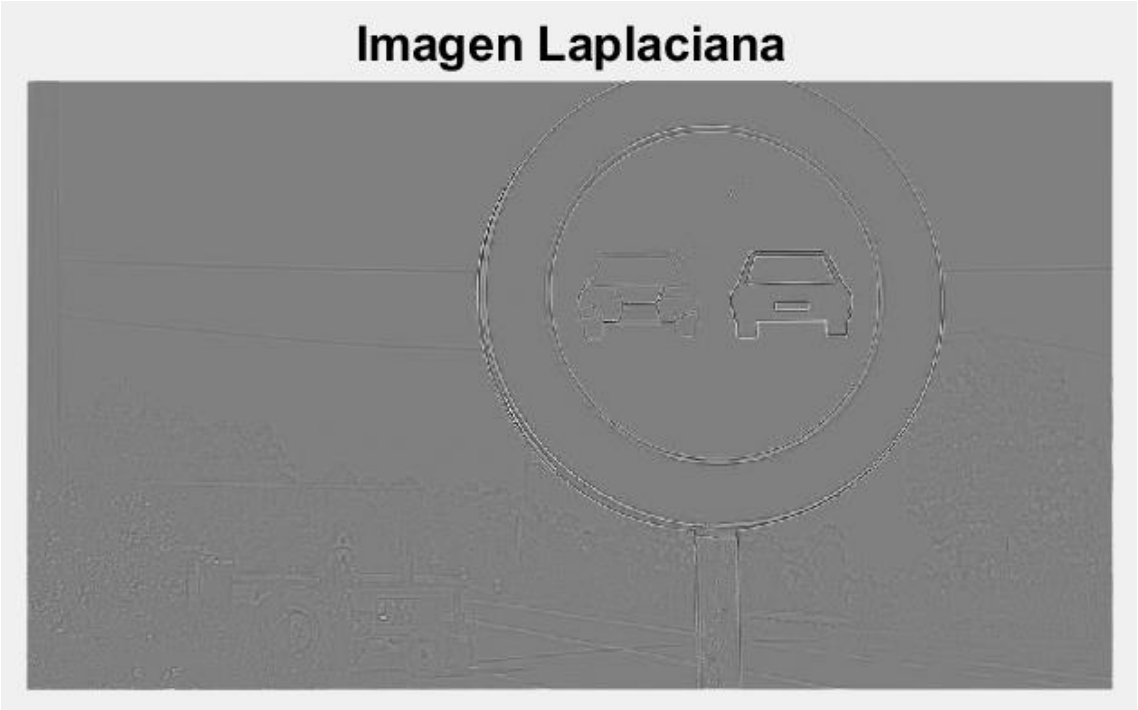
Con el filtro de Sobel lo que tratamos es de clacular el gradiente de intensidad de la imagen en cada punto consiguiendo que de el mayor cambio posible. Es por eso que es de los mejores para la detección de bordes.




```
% Computer Sobel image
[magnitudeImage, directionImage] = imgradient(grayImage, 'Sobel');
% Display the image.
subplot(2, 2, 4);
imshow(magnitudeImage, []);
title('Filtro Sobel', 'FontSize', fontSize);
```

Laplace.

Este filtro realza los bordes en todas direcciones, o sea, el resultado de la imagen es la suma de los resultados obtenidos al aplicar realce Horizontal, Vertical y Diagonal. Se trabaja con la segunda derivada y su inconveniente es que aumenta el ruido.




```
% Compute Laplacian
laplacianKernel = [-1,-1,-1;-1,8,-1;-1,-1,-1]/8;
laplacianImage = imfilter(double(grayImage), laplacianKernel);
% Display the image.
subplot(2, 2, 2);
imshow(laplacianImage, []);
title('Imagen Laplaciana', 'FontSize', fontSize);
```

Sharpened.

Este filtro se crea a partid de la suma entre el filtro de Laplace y la imagen original, dando como resultado una imagen que realza los bordes, pero no es tan exagerada como el Laplaciano.


```
% simply add the original image to the laplacianImage.
sharpenedImage = double(grayImage) + laplacianImage;
% Display the image.
subplot(2, 2, 3);
imshow(sharpenedImage, []);
title('Imagen Afilada (Sharpened)', 'FontSize', fontSize);
```

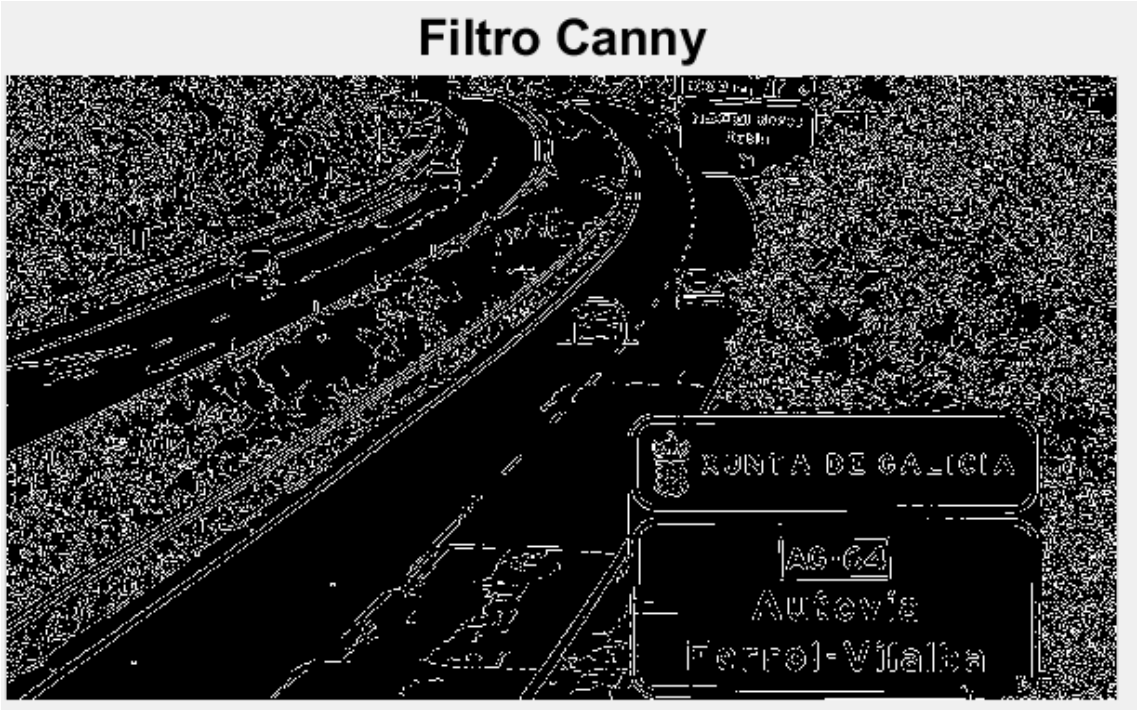
Detección de Bordes.

Para la detección de bordes tenemos el algoritmo de Canny y el de Sobel, este ultimo utilizado más arriba.

Canny

El filtro de Canny se compone de diferentes etapas. La primera trata de reducir el ruido realizando la primera derivada de una matriz gaussiana.

La siguiente etapa trata de detectar la intensidad del gradiente de la imagen, en esto utilizamos 4 filtros diferentes ya que los bordes pueden estar en cuatro direcciones diferentes.




```
% Computer Canny image
[magnitudeImage, directionImage] = edge(grayImage, 'canny');
% Display the image.
subplot(2, 2, 4);
imshow(magnitudeImage, []);
title('Filtro Cany', 'FontSize', fontSize);
```

Segmentación.

Segmentación basada en Clúster.

Para realizar esta segmentación necesitamos realizar diversos pasos.

El primer paso es leer la imagen.

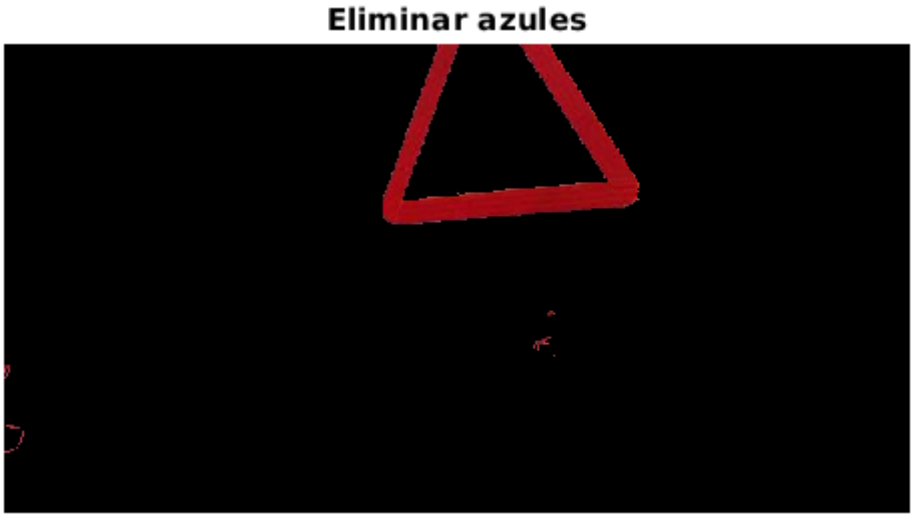
El segundo paso es transformar la imagen de color RGB a un espacio de color que nos permita trabajar correctamente con la imagen.

El tercer paso trata de clasificar la imagen del espacio anteriormente creado usando el filtro de agrupamiento K-medias. Este filtro crea una imagen en blanco y negro donde los diferentes objetos se agrupan en la imagen con diferentes colores.

El cuarto paso trata de segmentar la imagen por los colores que la componen. Siendo en este caso 3 regiones diferentes de colores.

Si nos fijamos en la ultima imagen vista obtenemos que hemos encontrado que nos da información acerca de nuestra señal.

Quinto paso trata de eliminar los azules en la imagen dando una imagen más concreta de la parte principal de esta.



Así es como conseguimos sacar el tipo de señal que tenemos en la imagen.

Filtro Gabor

El filtro de Gabor trata de segmentar la imagen en las diferentes texturas que la componen. Para esto necesitamos realizar los siguientes pasos.

El primero es leer la imagen.

El segundo paso es diseñar una matriz de filtro que estén sintonizados a diferentes frecuencias y orientaciones, subconjuntos y aproximaciones ortogonales de la información de frecuencia y orientación de la imagen de entrada.

Postprocesamos la imagen mediante suavizado con filtros gaussianos y filtros *kmeans* y normalizamos la información con la varianza y la media común.

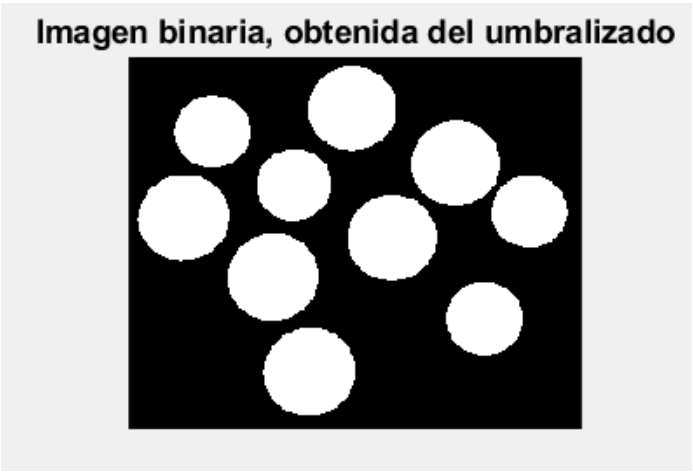
Clasificamos las características de textura mediante *kmeans*

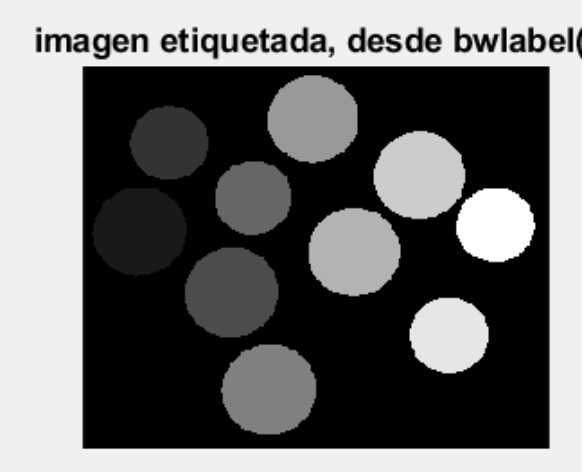
Y para finalizar visualizamos las dos regiones de la imagen donde vemos que una es la que necesitamos, la señal, que es la parte relevante de la imagen.

Recorte de imágenes.

Para el recorte de una imagen necesitamos clasificar en primera instancia la posición de los objetos que queremos recortar mediante el histograma.

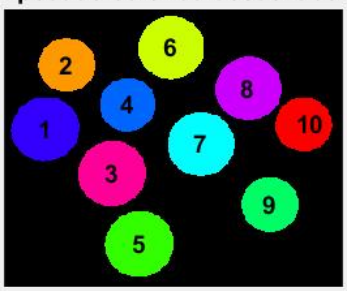
Mediante umbralizado eliminamos los huecos de las monedas y mediante bwlabel los etiquetamos.



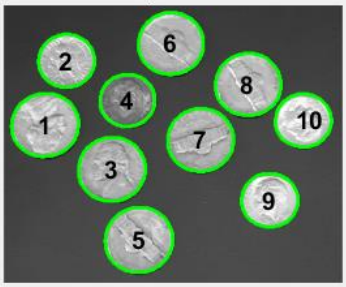


Después coloreamos la imagen mediante label2rgb y dibujamos las líneas de contorno.

Imagen pseudo colorida desde label2rgb().

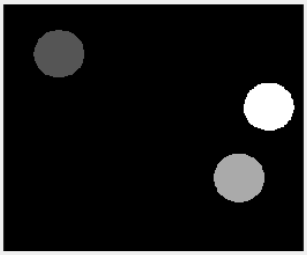


Lineas de contorno, desde bwboundaries()

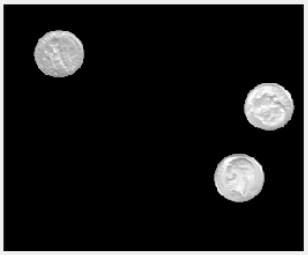


Clasificamos las monedas pequeñas y las mostramos.

Monedas pequeñas separadas y clasificadas

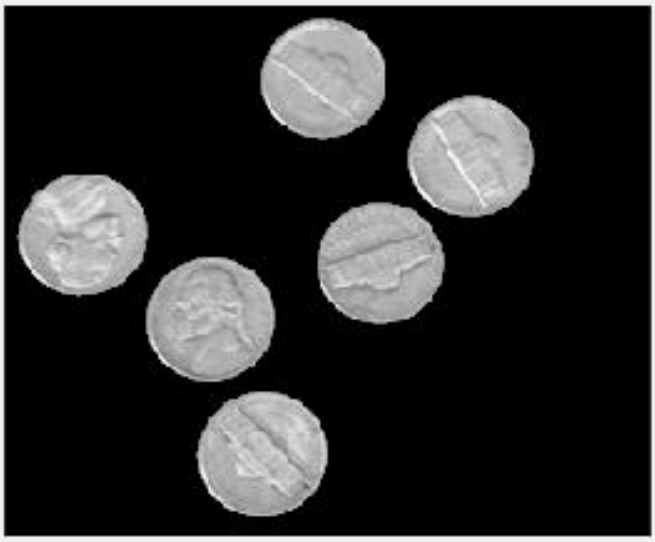


Solo las monedas pequeñas de la imagen principal

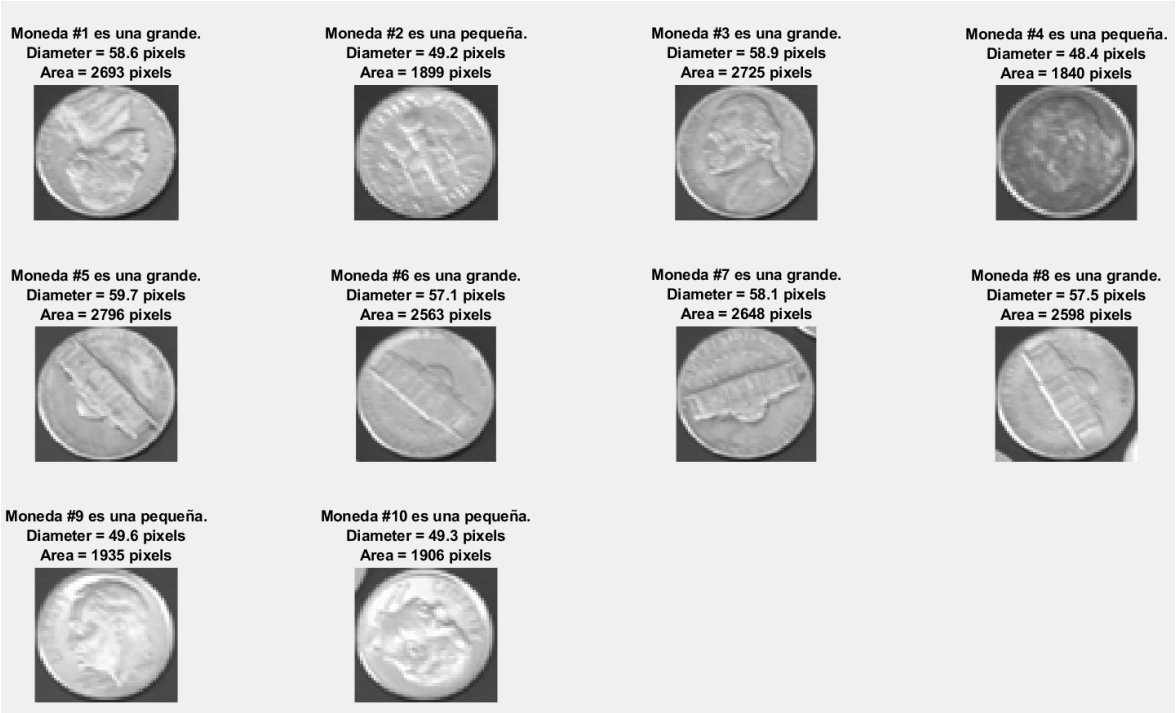


Después las monedas grandes.

Solo las monedas grandes de la imagen principal



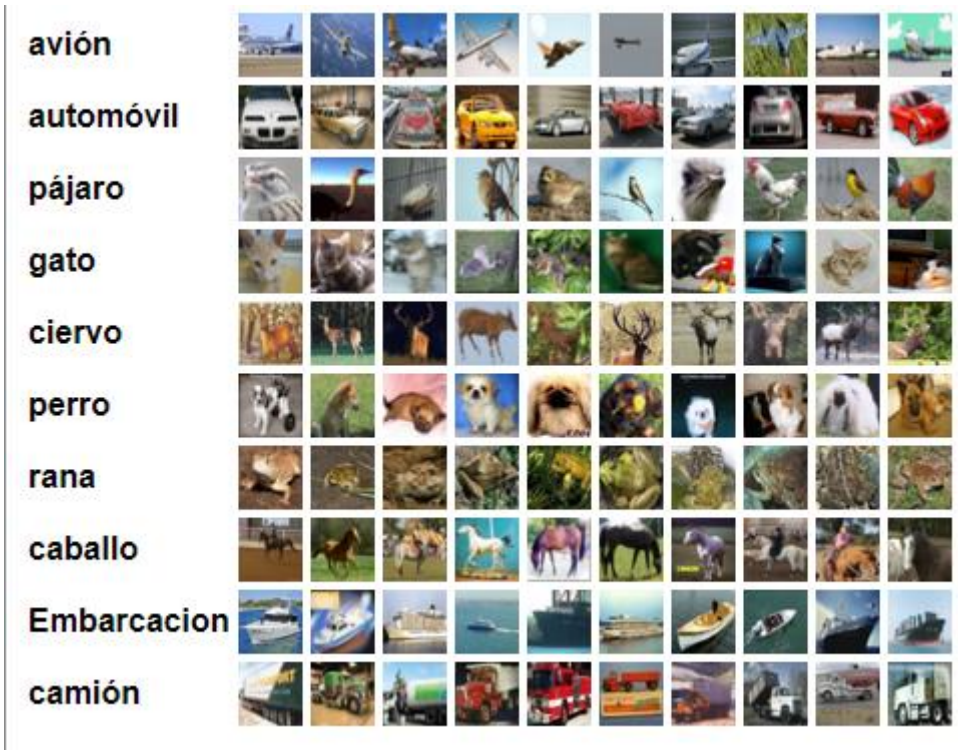
Para terminar. Recortamos las monedas en diferentes imágenes y señalamos su tamaño en el que se ha basado el recorte.



Esta parte del trabajo no está realizada con las señales porque no ha dado tiempo de implementar el código para este fin, pero podemos observar la manera de realizarlo con la imagen de stock que tiene Matlab llamada Coins.png.

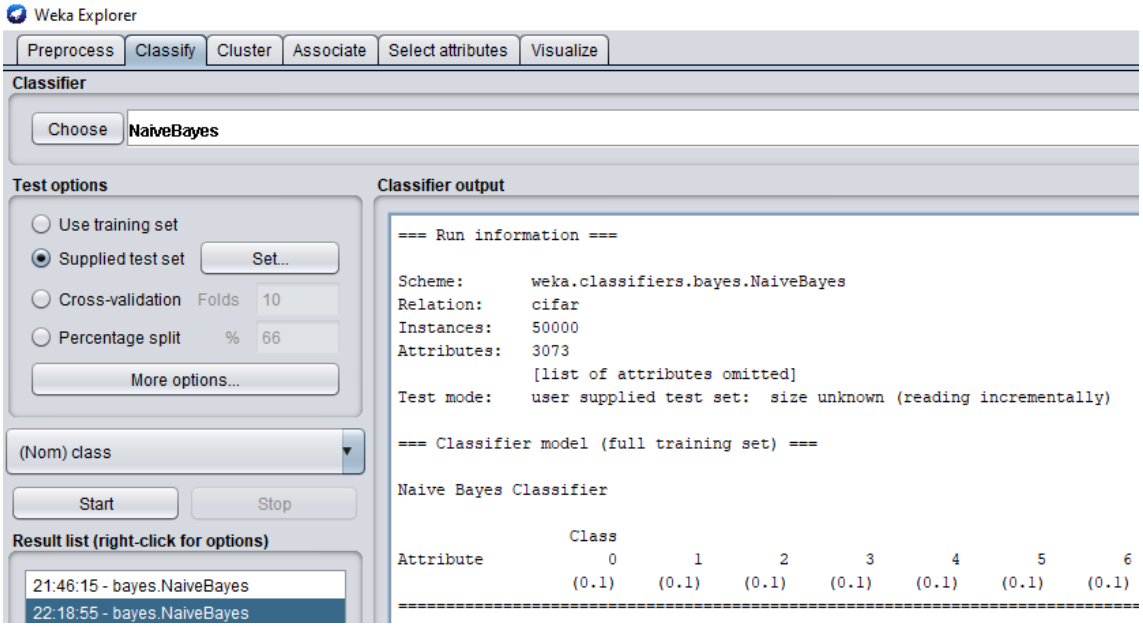
Análisis de datos.

Para analizar datos voy a utilizar Weka, en este caso voy a utilizar el datasheet CIFAR-10, que es una base de datos de imágenes preparada para la clasificación de distintos tipos de imágenes, se compone por 50.000 imágenes de entrenamiento y 10.000 imágenes de prueba. Tenemos 10 tipos de clases por lo que tenemos 6.000 imágenes de cada clase para ver como funcionan. Las clases son las siguientes:



en la pagina principal he descargado los archivos train.arff y test.arff y los he introducido en Weka para trabajar con el clasificador de NaivesBayes.

Hemos clasificado las imágenes y hemos añadido el archivo de prueba en la opción que vemos en la siguiente imagen.

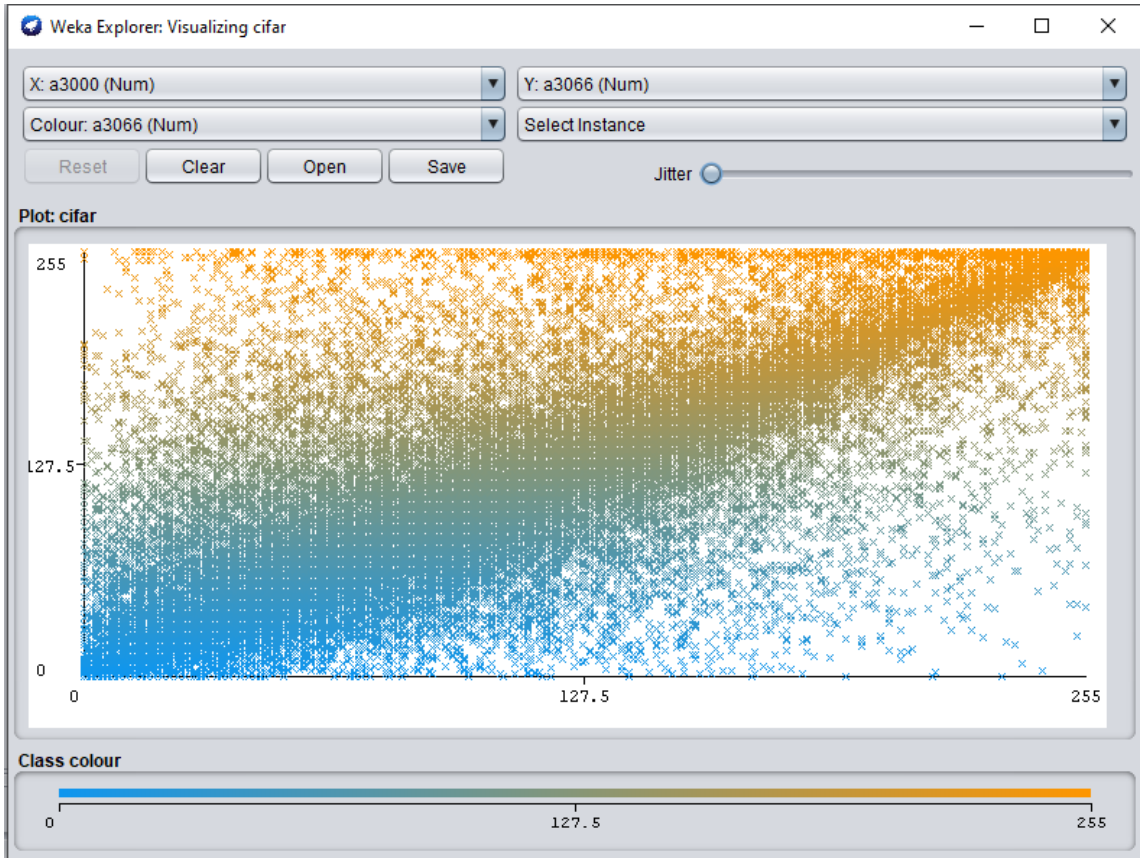


Los resultados que obtenemos son de un 70% de clasificación correcta frente a un 29.74% de clasificación errónea, por lo que podemos obtener que este tipo de clasificación con tiene mucho error.

```
=== Summary ===

Correctly Classified Instances        2974               29.74    %
Incorrectly Classified Instances      7026               70.26    %
Kappa statistic                          0.2193
Mean absolute error                      0.1406
Root mean squared error                  0.3741
Relative absolute error                 78.1084 %
Root relative squared error            124.7097 %
Total Number of Instances            10000
```

Como observamos, las clases más cercanas entre si tienen una tasa de acierto mayor.



Conclusión

Este ha sido un trabajo donde se ha visto como encontrar diferentes formas de separar partes de las imágenes, se han utilizado la mayoría de los filtros vistos en clase, aunque ha habido alguno que se ha encontrado en internet y se ha propuesto ya que daban un resultado satisfactorio al plan que seguía para este fin. La idea se queda un poco en el aire, ya que no se han utilizado todas las posibles formas para detectar las señales porque había un tiempo reducido y realmente esta práctica podría ser una pretendiente a TFG o trabajo más desarrollado, al igual que para realizar el implementado para la detección de

señales necesitaríamos un datasheet de todas las señales de España (o del mundo) para poder obtener así unos resultados donde se nos dijera si la imagen que estamos viendo pertenece o no a nuestra búsqueda y que nos de así un resultado, para ello deberíamos hacer comparaciones de imágenes y una búsqueda recursiva de estas, esto entra en el tema de análisis de datos y demás, que no se ha podido llevar a cabo.

Para terminar, ha sido una práctica muy didáctica que ha recorrido todos los temas de temario de Tratamiento Digital de Imágenes, aunque algunos filtros no han sido implementados por no tener relevancia en el trabajo que he realizado.

Bibliografía.

- Aguirre Dobernack, N. (2013). Implementación de un sistema de detección de señales de tráfico mediante visión artificial basado en FPGA (TFG). Universidad de Sevilla, Sevilla, España. Recuperado de: http://bibing.us.es/proyectos/abreproy/12112/fichero/Documento_completo%252FProyecto+Fin+de+Carrera-Nicol%C3%A1s+Aguirre+Dobernack.pdf
- Cruzado Hernando, D. (2015). Detección y reconocimiento de señales de tráfico (TFG). Universidad Carlos III, Madrid, España. Recuperado de: https://e-archivo.uc3m.es/bitstream/handle/10016/23616/TFG_Daniel_Cruzado_Hernando_2015.pdf?sequence=1&isAllowed=y
- Lillo Castellano, J. M. (2010). Detección automática de señales de tráfico mediante procesamiento digital de imagen (TFG). Universidad Rey Juan Carlos, Madrid, España. Recuperado de: https://eciencia.urjc.es/handle/10115/7851?show=full
- MathWorks (2016). Add and Reduce Salt & Pepper Noise. Recuperado de: https://es.mathworks.com/matlabcentral/answers/273098-add-and-reduce-salt-pepper-noise
- MathWorks (2019). Detección de bordes. Recuperado de: https://es.mathworks.com/help/images/edge-detection.html
- MathWorks (2015). Image Segmentation Tutorial. Recuperado de: https://es.mathworks.com/matlabcentral/fileexchange/25157-image-segmentation-tutorial

- MathWorks (2019). Segmentación basada en color utilizando el espacio de color L*a*b. Recuperado de: https://es.mathworks.com/help/images/color-based-segmentation-using-the-l-a-b-color-space.html
- MathWorks (2019). Segmentación de texturas mediante filtros Gabor. Recuperado de: https://es.mathworks.com/help/images/texture-segmentation-using-gabor-filters.html
- Wikipedia (2020). Filtro de Gabor. Recuperado de: https://es.wikipedia.org/wiki/Filtro_de_Gabor
- Wikipedia (2020). Operador Sobel. Recuperado de: https://es.wikipedia.org/wiki/Operador_Sobel
- Wikipedia (2020). Procesamiento digital de imágenes. Recuperado de: https://es.wikipedia.org/wiki/Procesamiento_digital_de_im%C3%A1genes