Universidad Politécnica
de Cartagena

Escuela Técnica Superior de
Ingeniera de Telecomunicación

Máster Universitario en Ingeniería Telemática

Trabajo Fin de Máster

Preclasificación online de objetos en flujos de video mediante técnicas GMM

Autor: Luis Pérez Martinez

Tutor: Javier Vales Alonso

Cartagena, 2020

Este Trabajo Fin de Máster se ha depositado en la ETSIT de la Universidad Politécnica de Cartagena para su defensa.

«Agradezco enormemente al tutor de este proyecto Javier Vales la enorme ayuda que me han proporcionado en estos últimos meses para facilitarme la elaboración de este trabajo de fin de máster.»

Trabajo Fin de Máster
Máster Universitario en Ingeniería Telemática
Título: Preclasificación online de objetos en flujos de video mediante técnicas GMM
Diciembre 2020

Autor(a): Pérez Martinez, Luis

Tutor(a):
Vales Alonso, Javier
ETSIT
Universidad Politécnica de Cartagena

Resumen

Cada día hay disponibles más y más datos y esta tendencia no para de crecer. Por ponerlo en contexto, en 2019 cada minuto son subidas a YouTube más de 400 horas de video. El 99,5% de los datos generados no se analizan debido a falta de recursos [1]. Toda esta información se hace imposible de analizar y visualizar manualmente, por lo que es necesario estudiar y emplear nuevas técnicas de análisis de información y videos.

El objetivo de este proyecto es estudiar la viabilidad de aplicar técnicas de inteligencia artificial para obtener, analizar y clasificar información extraída de videos de modo desatendido. Es decir, sin partir de unos datos etiquetados previos.

En concreto, nuestro objetivo es ser capaz de identificar y agrupar objetos en movimiento en un video, y todo ello del modo más simple posible y con la mínima intervención humana.

Para la realización del estudio hemos utilizado videos disponibles online. En este documento se recoge en concreto la aplicación para detectar peces en un acuario y diferenciar cuántas especies distintas hay. Ésta es simplemente una aplicación demostrativa, pero que ilustra potencialmente los pasos de resolución para otro tipo de problemas similares.

En un primer paso hemos detectado objetos en movimiento extrayendo un modelo del background a partir del video. Los objetos en primer plano son aquellos que se mueven entre frames, y pueden ser peces, plantas u otros objetos móviles. En un segundo paso estos objetos se uniformizan a un tamaño fijo y como imágenes en escala de grises. Tras ello se compara el objeto con un modelo Gaussiano multidimensional creado a partir de un grupo de imágenes de peces. Obsérvese que este paso requiere intervención humana, pero es más simple que un modelo supervisado donde se requieren también etiquetar imágenes negativas. Por último, aquellas imágenes que superan este test son agrupadas con un modelo de mixturas gaussianas (GMM), usando el método del codo para determinar el número total de clusters (especies). Todo este algoritmo se ha desarrollado mediante Python con las librerías de OpenCV y Keras/TF.

Para concluir se puede afirmar que con la tecnología actual y utilizando las capacidades del lenguaje de programación Python y las librerías disponibles de inteligencia artificial es posible realizar un sistema que de forma semiautónoma pueda realizar un análisis de la información contenido en videos o imágenes.

Palabras clave: Big data, Inteligencia artificial, Python, GMM, Autoencoder, OpenCV, Clasificación, Aprendizaje Supervisado.

Abstract

Every day more and more data are generated, this trend does not stop growing up. In 2019, more than 400 hours of video are uploaded to YouTube every minute. 99.5% of the data generated are not analysed due to lack of resources or techniques[1]. All this information becomes impossible to analyse and visualize by hand. Therefore, is necessary an evolution of information and video analysis techniques to apply.

The objective of this project is to study the viability of apply artificial intelligence techniques to obtain, analyse and classify information extracted from videos.

To carry out the study we have been used online videos available. This document specifically collects the application to detect fish in an aquarium and differentiate how many different species there are. This is a simple demonstration application, but potentially illustrates the resolution steps for other similar types of problems.

In a first step we have detected moving objects by extracting a model of the background from the video. The objects in the foreground are those that move between frames, and can be fish, plants or other moving objects. In a second step these objects are standardized to a fixed size and as grayscale images. The object is then compared with a multidimensional Gaussian model created from a group of fish images. Note that this step requires human intervention but is simpler than a supervised model where negative images are also required to be labelled. Finally, those images that pass this test are grouped with a Gaussian mixture model (GMM), using the elbow method to determine the total number of clusters (species). All this algorithm has been developed using Python with the OpenCV and Keras / TF libraries.

To conclude, it can be stated that with current technology and using the capabilities of the Python programming language and the available artificial intelligence libraries, it is possible to create a semi-autonomous system that can perform an analysis of the information contained in videos or images.

Keyword: Big Data, Artificial Intelligence, Python, GMM, Autoencoder, OpenCV, Clustering, Classification, Supervised learning.

Tabla de contenidos

Tabla de ilustraciones

1 Introducción

Hoy en día la sociedad se ha digitalizado de tal forma que se crea tal cantidad de contenidos diariamente que es imposible de analizar. Tradicionalmente todos los contenidos creados se tenían que analizar manualmente para la obtención de información relevante de ellos, pero con la con la evolución de las técnicas matemáticas que han llevado a la creación del aprendizaje maquina (machine learning) y de la inteligencia artificial cada se va haciendo más posible este análisis de forma automatizada.

Con este trabajo titulado *"Preclasificación online de objetos en flujos de video mediante técnicas GMM"* pretendo realizar un estudio de las distintas técnicas de análisis de la información contenida en videos o imágenes, para extraer las características e información relevante para crear un conjunto e imágenes normalizadas, dataset. Posteriormente con este dataset realizare un estudio comparativo de las diferentes técnicas de clusterización para comprobar su eficacia y precisión. De esta manera demostrar la posibilidad de crear un sistema que de forma autónoma pueda analizar los videos de manera online, sacar información y conclusiones de ellas.

Los objetivos de este trabajo son los siguientes:

- Demostrar la viabilidad de realizar un sistema que de forma autónoma pueda analizar videos y extraer la información relevante del mismo, como por ejemplo las especies de animales que aparecen.

- Comprobar las diferentes capacidades de la librería OpenCV sobre código Pyhton para detectar movimiento en video y para eliminar elementos no deseados de una imagen para centrase solo en el objeto deseado, como la eliminación de un fondo.

- Verificar la aplicabilidad y precisión de las diferentes técnicas de inteligencia artificial disponibles para la clusterizacion de una serie de objetos extraídos de las imágenes para este caso en particular.

- Realizar de una forma precisa y mediante algoritmos de inteligencia artificial no supervisados una clasificación y posterior clusterizacion según las características de los objetos en las imágenes.

El ejemplo utilizado en este trabajo ha sido con un video de un acuario de peces, en este video hay diferentes objetos en movimiento, peces, plantas, agua etc. El objetivo es ser capaz de identificar los peces en movimiento y poder separarlos en la misma especie o similares.

En el primer capítulo de este trabajo explico los diferentes métodos disponibles en la librería OpenCV 3 sobre lenguaje Python para realizar un análisis de video, es decir detectar los objetos en movimiento, recortar solo ese objeto, eliminarle el fondo a esa imagen y guardarlo. De esta manera que estas imágenes

separadas sirvan como banco de datos, dataset, para poder aplicar sobre ellas posteriormente las técnicas de clusterización.

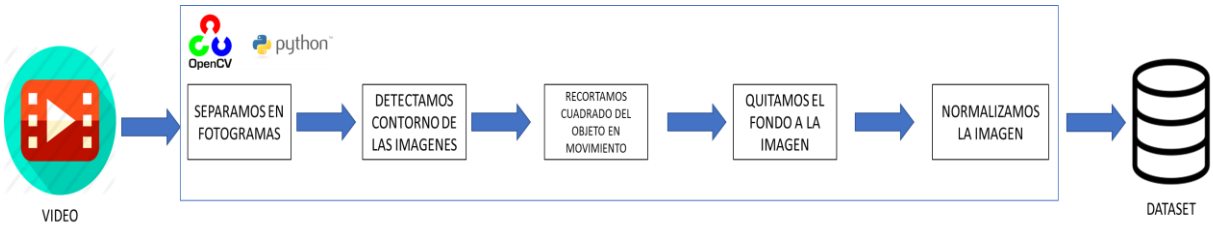


Figura 1: Esquema del flujo de trabajo realizado con OpenCV

En el segundo capítulo realizaré un estudio de las distintas técnicas disponibles para la separación de las imágenes los grupos deseados. Primero explicaré los conceptos teóricos utilizados. A continuación, explicare la creación de un preclasificador entrenado con un conjunto de imágenes para la separación de las imágenes en 2 grupos. Y por último aplicaré diferentes técnicas de clusterizacion, es decir separación de objetos similares, para poder separar los peces por las diferentes especies utilizando la técnica GMM.

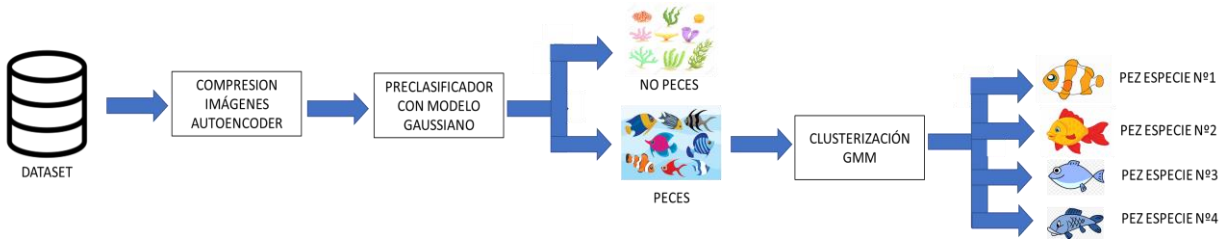


Figura 2: Esquema del flujo de trabajo utilizado para la clasificación y clusterización de las imágenes.

Finalmente, concluiré este trabajo verificando la precisión de cada uno de los métodos de clusterizacion y evaluando si se ha cumplido los objetivos de este trabajo.

2 Desarrollo

2.1 Análisis del video y creación del dataset

2.1.1 Introducción a la librería OpenCV

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. OpenCV significa Open Computer Vision (Visión Artificial Abierta). Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en una gran cantidad de aplicaciones, y hasta 2020 se la sigue mencionando como la biblioteca más popular de visión artificial.1 Detección de movimiento, reconocimiento de objetos, reconstrucción 3D a partir de imágenes, son sólo algunos ejemplos de aplicaciones de OpenCV. [2]

Sus características principales son:

- Open-source, publicada bajo licencia BSD.
- Multiplataforma, para los sistemas operativos GNU/Linux, Mac OS X, Windows y Android, y para diversas arquitecturas de hardware como x86, x64 (PC), ARM (celulares y Raspberry Pi).
- Interfaces en múltiples lenguajes: Python, Java y C++ [3].
- Contiene implementaciones de más de 2500 algoritmos [4].

Se puede utilizar OpenCV para realizar operaciones simples con imágenes como [3]:

- Abrir y guardar imágenes
- Dibujar formas simples en imágenes
- Escribir en imágenes

Las posibilidades de análisis y tratamiento de imágenes con la biblioteca OpenCV son enormes, desde detectar caras y clasificarlas según género hasta crear modelos de realidad aumentada o usar clasificadores para detectar objetos [5].

Las áreas de aplicación de OpenCV incluyen:

- Características 2D y 3D
- Estimación de pose de cámara
- Reconocimiento facial
- Reconocimiento de gestos
- Interacción persona-computadora
- Robótica móvil
- Comprensión de movimientos
- Reconocimiento de objetos
- Segmentación
- Estereoscopía
- Structure from motion (SFM)
- Tracking
- Realidad aumentada

2.1.2 Detección del movimiento

En este subapartado explicare tanto cual es proceso seguido para la detección del movimiento en una imagen utilizando las diferentes funciones disponibles de la librería opencv.

Las funciones necesarias para crear el código que detecte movimiento son las siguientes:

- cv2.VideoCapture: Esta función abre un archivo de video o un dispositivo de captura o una transmisión de video IP para capturar video con Preferencia API. En esta función hay diferentes variantes según los parámetros de entrada que acepta. La versión utiliza en este trabajo es la que tiene un único parámetro de entrada *filename*. El *filename* acepta diferentes formatos y opciones de entrada que son:
  - Nombre del archivo de video (por ejemplo, video.avi) o en caso de que no está en la misma ubicación que el script de Python la ruta completa.
  - Secuencia de imágenes (por ejemplo, img_% 02d.jpg, que leerá ejemplos como img_00.jpg, img_01.jpg, img_02.jpg, ...).
  - URL de la transmisión de video (p. ej., protocolo: // host: puerto / script_name? script_params | auth).
  - GStreamer en formato de herramienta gst-launch en caso de que GStreamer se utilice como backend. [6]

- cv2.erode: Erosiona una imagen mediante el uso de un elemento específico. La función erosiona la imagen de origen utilizando el elemento de estructuración especificado que determina la forma de una vecindad de píxeles sobre la que se toma el mínimo.

$$\texttt{dst}(x,y) = \min_{(x',y'):\, \texttt{element}(x',y')\neq 0} \texttt{src}(x+x', y+y')$$

Figura 3: Formula de la erosión utilizada por OpenCV [7]

Es decir, la función de erosión de una imagen es similar a la convolución 2D. Un píxel de la imagen original (1 ó 0) sólo se considerará 1 si todos los píxeles que caen dentro de la ventana del kernel son 1, de lo contrario se modifica a 0 (se erosiona). Por tanto, todos los píxeles cerca de los bordes de los objetos en la imagen serán descartados dependiendo del tamaño del kernel [8]

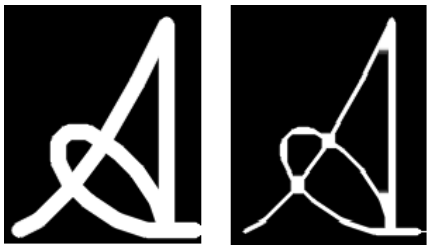


Figura 4: Ejemplo de la utilización Erode de OpenCV.

Los parámetros de esta función son los siguientes:

- Src: Es la imagen de entrada; el número de canales puede ser arbitrario, pero la profundidad debe ser una de los siguientes CV_8U, CV_16U, CV_16S, CV_32F o CV_64F
- Dst: Es la imagen de salida del mismo tamaño y tipo que src.
- Kernel: Es el elemento utilizado para la erosión, si el elemento = Mat (), se utiliza un elemento rectangular de 3 x 3. El kernel se puede crear usando la función getStructuringElement.
- Anchor: Es la posición del ancla dentro del elemento; El valor predeterminado (-1, -1) significa que el ancla está en el centro del elemento.
- Iterations: Es el número de veces que la erosión es aplicada.
- BorderType: Es el método de extrapolación de pixeles utilzado.
- BorderValue: En caso de que el borde sea una constante es el valor utilizado [7].

- cv2.dilate: Dilata una imagen mediante el uso de un elemento específico. La función dilata la imagen de origen utilizando el elemento de estructuración especificado que determina la forma de una vecindad de píxeles sobre la que se toma el máximo.

$$\texttt{dst}(x,y) = \max_{(x',y'):\, \texttt{element}(x',y') \neq 0} \texttt{src}(x+x', y+y')$$

Figura 5: Formula de la dilatación utilizada por OpenCV [7]

El proceso de dilatación es el opuesto a la erosión. Aquí, un elemento de píxel es '1' si al menos un píxel de la imagen de los que caen dentro de la ventana del kernel es '1'. Por lo tanto, la dilatación aumenta el tamaño de los objetos de primer plano. Normalmente, en casos como la eliminación del ruido, la erosión es seguida de dilatación. La razón para esto es que, aunque la erosión elimina los ruidos blancos también encoge los objetos, por lo tanto, es necesario aplicar una dilatación después para recuperar el tamaño habitual de los objetos [8].

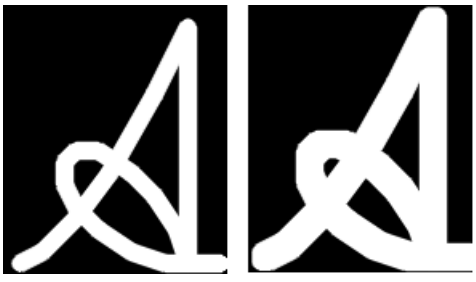


Figura 6: Ejemplo de la utilización Dilate de OpenCV.

Los parámetros de esta función son los siguientes:

- Src: Es la imagen de entrada; el número de canales puede ser arbitrario, pero la profundidad debe ser una de los siguientes CV_8U, CV_16U, CV_16S, CV_32F o CV_64F
- Dst: Es la imagen de salida del mismo tamaño y tipo que src.
- Kernel: Es el elemento utilizado para la erosión, si el elemento = Mat (), se utiliza un elemento rectangular de 3 x 3. El kernel se puede crear usando la función getStructuringElement.
- Anchor: Es la posición del ancla dentro del elemento; El valor predeterminado (-1, -1) significa que el ancla está en el centro del elemento.
- Iterations: Es el número de veces que la erosión es aplicada.
- BorderType: Es el método de extrapolación de pixeles utilzado.
- BorderValue: En caso de que el borde sea una constante es el valor utilizado [7].

- cv2.findContours: Esta es la función utilizada para poder encontrar y dibujar contornos de algún objeto o región de interés. Una limitación de esta función es que la imagen de entrada debe ser binaria, es decir tiene que estar en blanco y negro. Los contornos son una herramienta útil para el análisis de formas y la detección y reconocimiento de objetos.

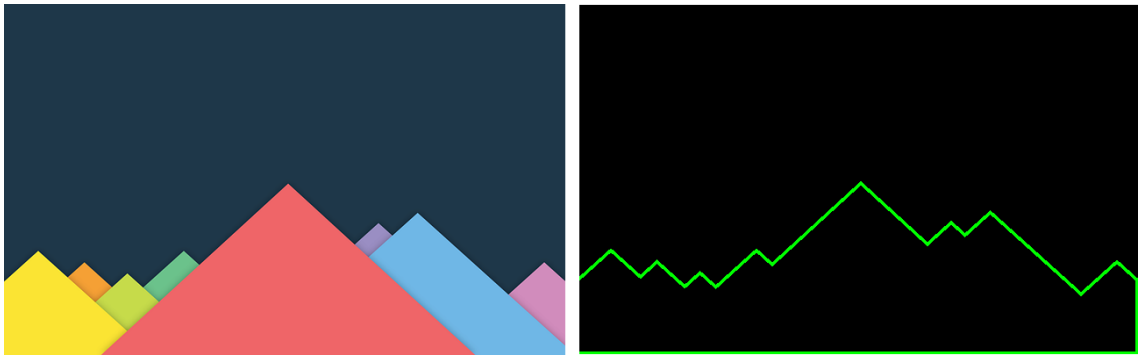


Figura 7: Ejemplo de utilización de findContours sobre una imagen sencilla en OpenCV [9].

Los parámetros de esta función son los siguientes:

- Image: Es la imagen de entrada, debe ser de un solo canal de 8 bits. Los píxeles distintos de cero se tratan como unos. Cero píxeles siguen siendo 0, por lo que la imagen se trata como binaria. Puede usar comparar, inRange , umbral , adaptiveThreshold , Canny y otros para crear una imagen binaria a partir de una escala de grises o de color.
- Contours: Es la salida de la función y son los contornos detectados. Cada contorno se almacena como un vector de puntos.
- Hierarchy: Es un vector opcional de salida que contiene información sobre la topología de la imagen. Tiene tantos elementos como el número de contornos.
- Mode: Modo de recuperación de contorno.

- Method: Es el método de aproximación de contorno utilizado. Hay 4 métodos diferentes que se pueden utilizar que son [10]:
  - chain_approx_none: Almacena absolutamente todos los puntos de contorno. Es decir, 2 puntos subsiguientes cualesquiera (x1, y1) y (x2, y2) del contorno serán vecinos horizontales, verticales o diagonales.
  - chain_approx_simple: Comprime segmentos horizontales, verticales y diagonales y deja solo sus puntos finales. Por ejemplo, un contorno rectangular hacia arriba a la derecha se codifica con 4 puntos.
  - chain_approx_tc89_l1.
  - chain_approx_tc89_kcos.

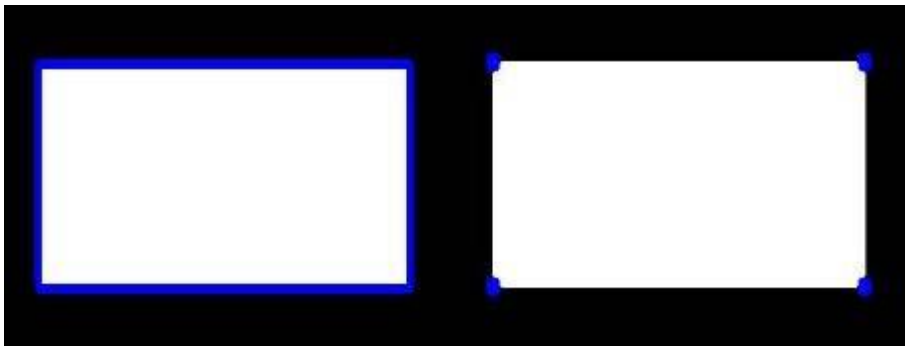


Figura 8: Diferencia entre la utilización del meto de aproximación de contorno con el aprox_none (izquierda) y el approx_simple (derecha).

- Offset: Es el desplazamiento opcional por el que se desplaza cada punto del contorno [11].

- cv2.contourArea: Esta función calcula un área de contorno. De manera similar a los momentos, el área se calcula usando la fórmula de Green. Los parámetros de esta función son los siguientes:
  - Contour: Es el vector de entrada de puntos 2D (vértices de contorno).
  - Oriented: Es la bandera de zona orientada. Si es verdadero, la función devuelve un valor de área con signo, según la orientación del contorno (en sentido horario o anti horario). Con esta función, puede determinar la orientación de un contorno tomando el signo de un área. De forma predeterminada, el parámetro es falso, lo que significa que se devuelve el valor absoluto [12].

- cv2.boundingRect: Esta función calcula y devuelve el rectángulo delimitador mínimo para el conjunto de puntos especificado o píxeles distintos de cero de la imagen en escala de grises.
  El parámetro de entrada de esta función es:
  - Array: Es la imagen en escala de grises o el conjunto de puntos en 2-D para calcular el rectángulo delimitador entre esos puntos [12].

- cv2.rectangle: Esta función dibuja un rectángulo simple, grueso o relleno. Es utilizada para destacar zonas relevantes de una imagen como en nuestro caso los objetos en movimiento.
  Los parámetros de esta función son los siguientes:

  - Img: Es la imagen de entrada donde se va a dibujar el rectángulo
  - Pt1: Es el vértice del rectángulo a dibujar
  - Pt2: Es el vértice del rectángulo opuesto a pt1.
  - Color: Es el Color o brillo del rectángulo a dibujar.
  - Thickness: Será el grosor de las líneas que componen el rectángulo. Los valores negativos, como FILLED, significan que la función tiene que dibujar un rectángulo relleno.
  - LineType: Es el tipo de línea a dibujar.
  - Shift: Es el número de bits fraccionarios en las coordenadas del punto [13].

Una vez explicadas todas las funciones utilizadas para detectar el movimiento en un video. Pasaré a detallar los pasos seguidos, así como la utilidad de cada uno de ellos. Aunque iré añadiendo los fragmentos de código, este se encuentra completo en el anexo de esta memoria.

En nuestro código lo primero que hemos realizado es una función que realizará toda la creación del dataset de forma iterativa según los parámetros de entrada establecidos.

La definición de la función y los parámetros de entrada son los siguientes como se puede leer en el código siguiente:

```
def CreateImages (path, tipo, learning_step, pathOut):

    """ Función que dado un video situado en una ruta de entrada le elimina el
background haciendo uso de la librería opencv y añade en un recuadro el objeto
en movimiento. Recorta de este recuadro el objeto en movimiento, lo normaliza y
lo guarda en la ruta de salida con fondo y sin fondo de la imagen.

        :params path: Ruta en local donde se situa el video
        :type path: string
        :params tipo: Metodo a aplicar para eliminar el background KNN ó MOG2
        :type tipo: string
        :params learning_rate:
        :type learning_rate: double
        :params pathOut: Ruta de salida donde guarda los imagenes resultado
        :type learning_rate: string

    """
```

Una vez que tenemos definido la función así como los parámetros de entrada pasamos a leer el video de entrada con la función cv.VideoCapture() guardado el resultado en la variable frames, ya que se guara fotograma a fotograma. Además, se realiza una comprobación para ver si el video se ha leído correctamente. El código realizado es el siguiente:

```
## Utilizamos videoCapture para leer el video origen
cap = cv2.VideoCapture(path)
success, frame = cap.read()
count = 0

## Comprobación de que sea abierto bien el video
if not cap.isOpened:
    print('Unable to open: ' + path)
    exit(0)
```

Lo siguiente que realizamos es un bucle utilizando la variable creada al leer el video success para que cuando vaya leyendo fotograma a fotograma se vaya haciendo el análisis de la imagen para detectar el movimiento.

Dentro de este bucle lo primero que hacemos para detectar el movimiento es aplicar la función cv2.Erode() para eliminar el ruido de la imagen, ya que elimina los ruidos blancos, de esta forma será más fácil detectar los contornos y su movimiento. Y después tal como se ha explicado previamente aplicamos la función cv2.dilate para recuperar el tamaño original de los objetos de la imagen. En ambos casos se aplica utilizando el kernel por defecto y con 2 iteraciones ya que después de realizar diferentes pruebas es el que mejor resultado proporcionaba sin que afectase considerablemente a la velocidad de la función. Los parámetros de entrada a utilizar no serán los fotogramas directamente, sino que serán la salida de ese fotograma eliminando el fondo, esto se detallara en el próximo apartado. El código utilizado es el siguiente:

```
while success:
## Detectamos el contorno de las imágenes para ver los objetos en movimiento
        cv2.erode(thresh, erode_kernel, thresh, iterations=2)
        cv2.dilate(thresh, dilate_kernel, thresh, iterations=2)
```

A continuación, vamos a aplicar la parte más importante de este código que es aplicar la función cv2.findContours para poder detectar el movimiento, para ello utilizará como parámetro de entrada la salida de la función de eliminación del fondo después de aplicar la reducción de ruido, con el código anterior.

En la función utilizamos en el parámetro de entrada Hierarchy (jerarquía) el método RETR_EXTERNAL ya que solo quedemos que nos devuelva el contorno completo sin importar los contornos hijos que tenga el contorno principal [14], es decir con esto solo queremos diferenciar los peces pero que nos devuelva cada parte del pez como un contorno independiente. Y el método utilizado es el chain_approx_simple, ya que reduce mucho el coste computacional sin afectar al resultado. Ya que se realizó diferentes pruebas y no había mucha diferencia con los otros métodos. El código utilizado es el siguiente:

```
        contours, hier = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
```

Para finalizar a partir de los contornos encontrados en la función anterior vamos a dibujar un rectángulo alrededor de ese objeto para identificar en el video si la

detección de movimiento se está realizando correctamente y para terminar poder recortar ese rectángulo y guardarlo como una imagen separada para utilizarlo como imagen para nuestro dataset.

```
        ## A partir de los contornos dibujamos el rectangulo alrededor de ese objeto
            for c in contours:
                if cv2.contourArea(c) > 1000:
                    x, y, w, h = cv2.boundingRect(c)
                    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 0), 2)

        ## Recortamos el cuadrado y lo guardamos en una nueva imagen
            frame1 = frame[y:y+h, x:x+w]
            frame1_Mask = fg_mask[y:y+h, x:x+w]
```

Y con esto ya quedaría identificado y guardado los objetos en movimiento en nuestra imagen que luego utilizaremos para utilizarlo como elemento de entrada para nuestro clasificador de imágenes.

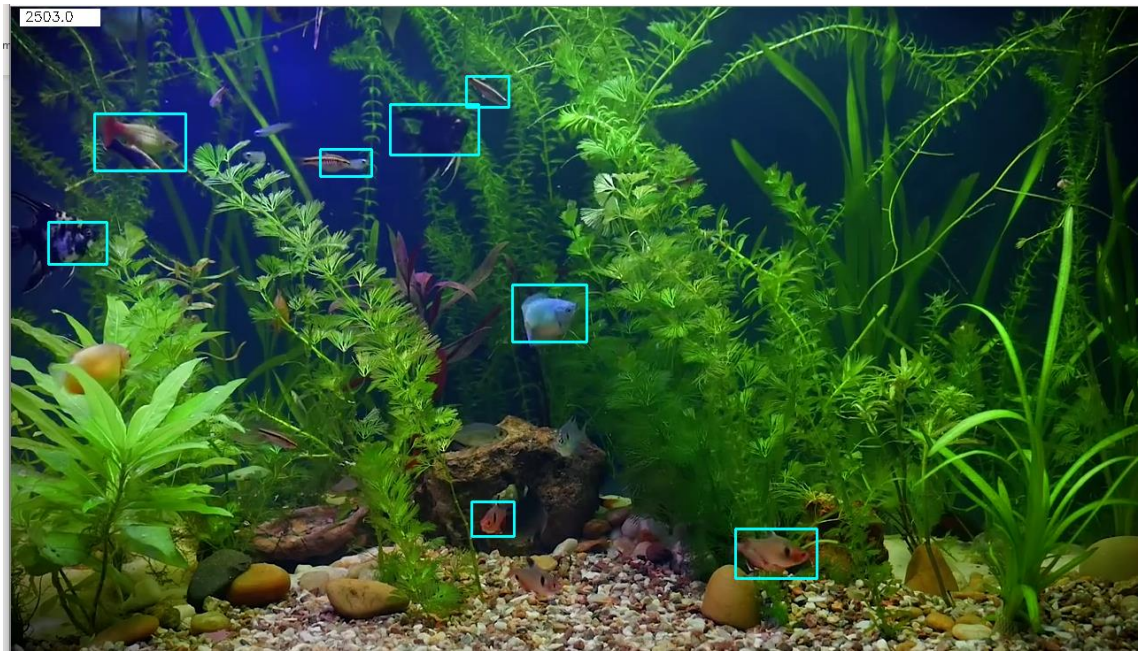


Figura 9: Fotograma del video aplicando la función de detección de movimiento.

2.1.3 Eliminación del fondo

Una parte esencial del trabajo realizado con la librería OpenCV es utilizar las funciones disponibles para poder eliminar el fondo de las imágenes, es decir el background de la imagen y centrarse solo en el objeto en movimiento de esta forma cuando se utilice esta imagen como argumento de entrada del clasificador sea mucho más fácil identificar si se trata de un pez y poder agrupar los peces de especies similares.

La eliminación de fondo es una técnica común y ampliamente utilizada para generar una máscara de primer plano (es decir, una imagen binaria que contiene los píxeles que pertenecen a los objetos en movimiento en la escena) mediante el uso de cámaras estáticas. Para ello calcula la máscara de primer plano realizando una resta entre el fotograma actual y un modelo de fondo, que contiene la parte estática de la escena, todo lo que se puede considerar como fondo.

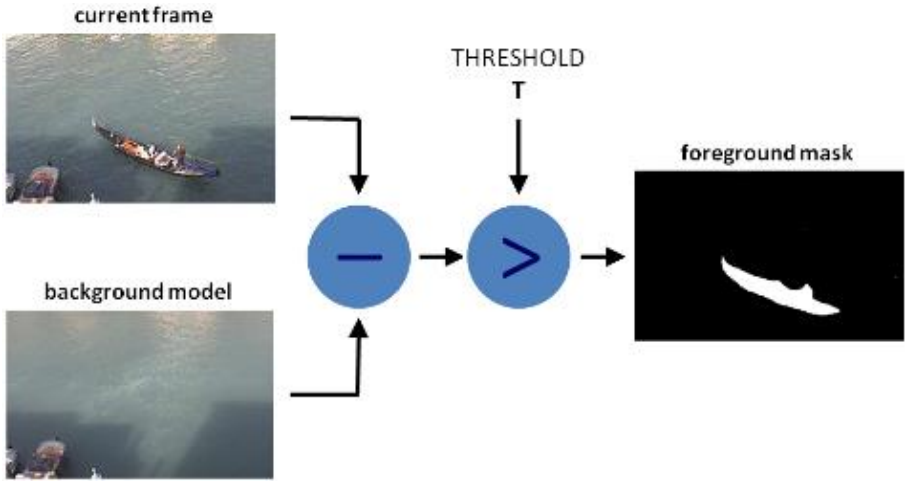


Figura 10: Diagrama de funcionamiento de la técnica de eliminación del fondo [15].

Para la realización de este proceso utilizamos las siguientes funciones de la librería OpenCV:

- cv2.createBackgroundSubtractorKNN: Esta función crea un sustractor de fondo KNN. Algoritmo de segmentación de fondo/primer plano basado en K vecinos más cercanos. Es un algoritmo muy eficiente si el número de píxeles de primer plano es bajo.
  Esta función tiene los siguientes parámetros [16]:
  - History: Es la duración de la historia.
  - Dist2Threshold: Es el umbral de la distancia al cuadrado entre el píxel y la muestra para decidir si un píxel está cerca de esa muestra.
  - DetectShadows: Si este parámetro está en True algoritmo detectará sombras y las marcará.

- cv2.createBackgroundSubtractorMOG2: Esta función crea un sustractor de fondo MOG2. Algoritmo de segmentación de fondo/primer plano basado en Gaussian mixture (GMM). Esta función tiene los siguientes parámetros [16]:

    - History: Es la duración de la historia.
    - Dist2Threshold: Es el umbral en la distancia de Mahalanobis (su utilidad radica en que es una forma de determinar la similitud entre dos variables aleatorias multidimensionalesal [17]) al cuadrado entre el píxel y el modelo para decidir si un píxel está bien descrito por el modelo de fondo.

$$d_m(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y})}.$$

Figura 11: Fórmula de la distancia de Mahalanobis [17].

    - DetectShadows: Si este parámetro está en True algoritmo detectará sombras y las marcará.

- bg_subtractor.apply: Esta función aplicada sobre la clase BackgroundSubtractor calcula una máscara de segundo plano, es decir elimina el fondo de una imagen. Esta función tiene los siguientes parámetros [18]:
    - Image: Siguiente fotograma de video.
    - Fgmask: Es la máscara resultante de primer plano del primer plano del objeto, tiene el formato de una imagen binaria de 8 bits.
    - Learning Rate: Es un valor entre 0 y 1 que indica como de rápido aprende el modelo. El valor del parámetro negativo hace que el algoritmo utilice una tasa de aprendizaje elegida automáticamente. 0 significa que el modelo de fondo no se actualiza en absoluto, 1 significa que el modelo de fondo se reinicializa completamente desde el último fotograma.

- cv2.threshold: Esta función aplica un umbral de nivel fijo a cada elemento de la matriz. La función se usa generalmente para obtener una imagen binaria de una imagen en escala de grises o para eliminar un ruido, es decir, filtrar píxeles con valores demasiado pequeños o demasiado grandes. Para cada píxel, se aplica el mismo valor de umbral. Si el valor de píxel es menor que el umbral, se establece en 0; de lo contrario, se establece en un valor máximo. Los parámetros de esta función son los siguientes [19]:
    - Src: Es la matriz de entrada.
    - Dst: Es la matriz de salida del mismo tamaño y tipo y el mismo número de canales que src.
    - Trillar: Es el valor umbral establecido.
    - Maxval: Es el valor máximo para usar con los tipos de umbral THRESH_BINARY y THRESH_BINARY_INV.
    - Type: Es el tipo de umbral. Los tipos de umbrales disponibles son los siguientes:
        - cv.THRESH_BINARY

- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO
- cv.THRESH_TOZERO_INV

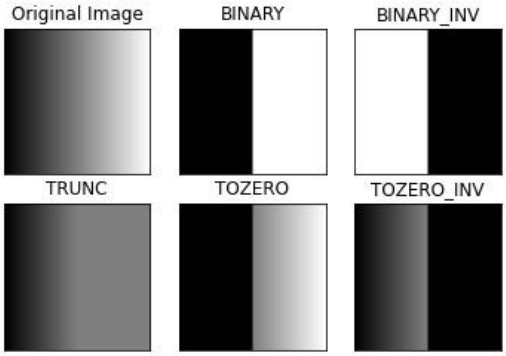


Figura 12: Resultado producido por cada uno de los diferentes umabres de la función cv2.threshold [20].

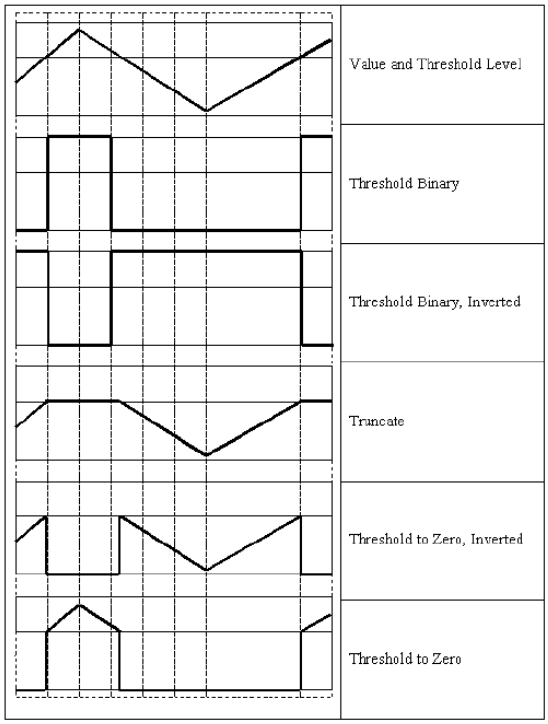


Figura 13: Diferencias de funcionamiento de cada uno de los diferentes umbrales de la función cv2.threshold [19].

Una vez detalladas y explicadas todas las funciones a utilizar, procedo a detallar los pasos seguidos, así como el código necesario para poder realizar estas funciones.

Lo primero a realizar será crear un objeto para generar la máscara del fondo, subtractor, que se creará según los parámetros de entrada. Como se ha explicado anteriormente hay diferentes tipos, MOG2, KNN entre otros. Por

defecto utilizamos el método de KNN, K vecinos más cercanos, ya que en las pruebas realizadas es que me proporcionaba un resultado más preciso. Además, también configuramos la función para que detecte sombras y también las identifique como parte del elemente, ya las eliminaremos mas tarde. El código utilizado es el siguiente:

```
## Creamos un objeto que va a generar la foreground mask.

    if tipo == 'MOG2':

        bg_subtractor = cv2.createBackgroundSubtractorMOG2(detectShadows=True)

    else:
        bg_subtractor = cv2.createBackgroundSubtractorKNN(detectShadows=True)
```

Una vez que se ha creado la clase para eliminar el fondo según el parámetro de entrada y dentro del bucle que va recorriendo nuestro video fotograma a fotograma aplicamos la función para eliminar el fondo. Se utiliza un learning rate de 0.01 ya que las pruebas realizadas es el que mejor resultado producía. El código utilizado para aplicar la función es el siguiente:

```
while success:

## Cada fotograma se utiliza para calcular la foreground mask y actualizar el
fondo.

        fg_mask = bg_subtractor.apply(frame, learningRate=learning_rate)
```

Por últimos vamos a aplicar la función thresh para poder crear una imagen binaria del fotograma con el fondo eliminado, de esta forma se tiene un fotograma adicional que se puede utilizar también como entrada a nuestro clasificador. Para aplicar el umbral para la transformación binaria tenemos en cuenta que la sombra la tratamos con el background ya que no nos proporciona información relevante, por lo que establecemos el umbral en un valor cercano al blanco de la máscara, que es de 255, por lo que establecemos el umbral a 244.

El código utilizado es el siguiente:

```
# Cuando pasamos el fotograma al método de eliminación del background, el
substractor actualiza su modelo interno del del background y devuelve una máscara.
La máscara es blanca (255) para los segmentos del foreground, gris (127) para
las sombras y negro (0) para el background. En este caso tratamos las sombras
como el background, así que aplicamos un límite cercano a blanco a la máscara
(244).

        _, thresh = cv2.threshold(fg_mask, 244, 255, cv2.THRESH_BINARY)
```

Y con esto ya estaría terminada la parte de eliminación del fondo, ya solo faltaría adaptar y normalizer las imágenes para poder utilizarlas como dataset de entrada a nuestro clasificador.

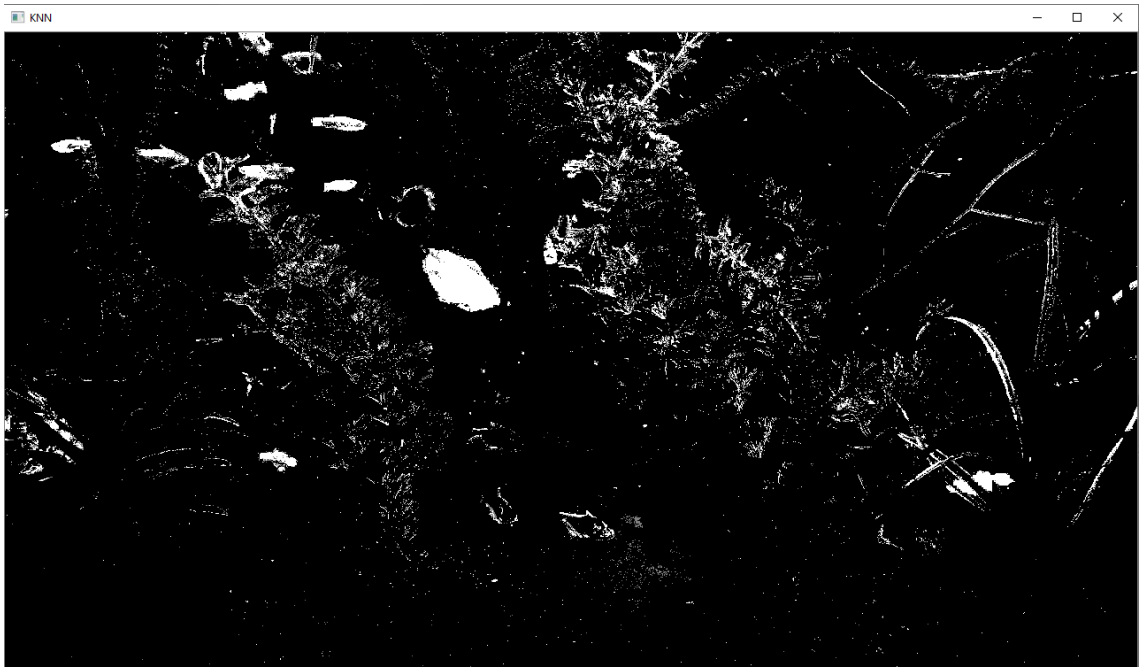


Figura 14: Fotograma del video aplicando la función de eliminación del fondo.

2.1.4 Creación del dataset

El último paso realizado con la librería openCV es adaptar, normalizar y guardar todas las imágenes extraídas de los fotogramas del video, detectando el movimiento y eliminando el fondo.

Este es un paso muy importante ya que, si todas las imágenes no se quedan de la misma forma, mismo tamaño, mismos colores etc., el clasificado no va a funcionar de forma óptima.

Para la realización de este proceso utilizamos las siguientes funciones:

- cv2.resize: Esta función cambia el tamaño de una imagen. Permite aumentar o reducir el tamaño original. Tiene diferentes métodos de interpolación de imágenes dependiendo del uso. Los parámetros de esta función son los siguientes:

    - Src: Es la imagen de entrada.
    - Dst: Es la imagen de salida; tiene el tamaño dsize (cuando no es cero) o el tamaño calculado a partir de src.size () , fx y fy; el tipo de dst es el mismo que el de src.
    - Dsize: Es el tamaño de la imagen de salida; si es igual a cero, se calcula como:
      dsize = Tamaño (redondo (fx * src.cols), round (fy * src.rows))
    - Fx: Es el factor de escala a lo largo del eje horizontal.
    - Fy: Es el factor de escala a lo largo del eje vertical.

- Interpolación: Es el método de interpolación a utilizar. Por defecto utiliza INTER_LINEAR, que es una interpolación bilineal [21].

- cv2.cvtColor: Esta función convierte una imagen de entrada de un espacio de color a otro. En el caso de una transformación de un espacio de color RGB, el orden de los canales debe especificarse explícitamente (RGB o BGR). Hay que tener en cuenta que el formato de color predeterminado en OpenCV a menudo se denomina RGB, pero en realidad es BGR (los bytes están invertidos). En nuestro caso que queremos convertir una imagen a escala de grises hay que utilizar BGR a Gray. Esta función tiene los siguientes parámetros de entrada:

  - Src: Es la imagen de entrada.
  - Dst: Es la imagen de salida del mismo tamaño y profundidad que src.
  - Code: Es el código de conversión de espacio de color.
  - DstCn: Es el número de canales en la imagen de destino; si el parámetro es 0, el número de canales se deriva automáticamente de src y código [22].

- cv2.imwrite: Esta función guarda la imagen en el archivo especificado. El formato de la imagen se elige en función de la extensión del nombre de archivo. En general, solo se pueden guardar imágenes de un canal de 8 bits o de 3 canales usando esta función, aunque hay excepciones. Tiene los siguientes parámetros de entrada:
  - Filename: Es el nombre del archivo a guardar.
  - Img: Es la imagen o imágenes a guardar.
  - Params: Parámetros específicos del formato codificados como pares [23].

Las transformaciones a realizar con las imágenes son las siguientes:

1. Establecemos un tamaño fijo para todas las imágenes.
2. Convertimos la imagen a escala de grises.
3. La guardamos con un nombre único dependiendo del origen.

Lo primero a realizar seria modificar el tamaño de la imagen para que todos tengan un tamaño fijo de la imagen, en este caso 128 x 128 pixeles. Para poder realizar esta transformación utilizamos el siguiente código, que hace uso de la función resize y del método de interpolación por defecto, el inter_lineal, que aun que no es el mejor para un caso concreto cuando es necesario hacer ampliaciones y reducciones de tamaño es el más polivalente. El código es el siguiente:

```
#Modificamos el tamaño de la imagen para que tengan un valor fijo
frame1_resized = cv2.resize(frame1,(128,128))
frame1_Mask_resized = cv2.resize(frame1_Mask,(128,128))
```

Lo siguiente a realizar será convertir la imagen es escala de grises para que podamos convertir la imagen a un vector y el clasificador pueda trabajar con ellas. Para ello utilizamos la función de cvt_color y aplicamos la tranformacion

del tipo de color por defecto BGR a escala de grises. El código utilizado es el siguiente:

```
## Convertimos la imagen recortada en escala de grises, solo la que tiene fondo
        frame1_resized_gray = cv2.cvtColor(frame1_resized, cv2.COLOR_BGR2GRAY)
```

Y por último lo que tenemos que hacer es guardar la imagen en diferentes carpetas según sea el recorte del objeto en que sea detectado movimiento o bien este mismo recorte, pero aplicando la eliminación de fondo, para poder utilizar ambas imágenes como nuestra entrada al clasificador. Para realizar esto hacemos uso de la función imwrite y utilizamos el siguiente código:

```
        ##Guardamos esa imagen en la carpeta
        cv2.imwrite( pathOut + "\\frameCrop%d.jpg" % count, frame1_resized_gray)
        cv2.imwrite(pathOut+"\\frameCropMask%d.jpg"%count,frame1_Mask_resized)
```

2.2 Separación de las imágenes

2.2.1 Conceptos teóricos

En este proyecto se han realizado 2 técnicas diferentes para poder separar las imágenes, se ha hecho uso de un preclasificador mediante un modelo gaussiano, realizado para separar en peces y no peces y de una clusterizacion mediante GMM para separar las distintas especies de peces encontrados en el video.

La definición de clusterizacion aplicada a la inteligencia artificial consiste en la segmentación y delimitación de grupos de elementos que pueden ser unidos por características o patrones comunes que comparten.

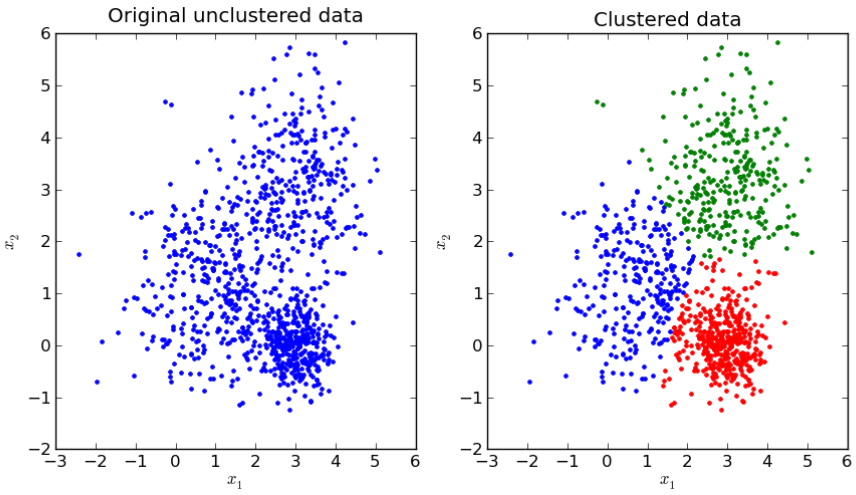


Figura 15: Esquema de funcionamiento de la técnica de clusterización

Esta asociación es realizada mediante algoritmos de agrupamiento que analizan el valor del conjunto de los datos y recalculan una y otra vez la estructura del grupo en base al análisis de los elementos que se incluyen dentro de ellos

mismos y la media de los valores que lo conforman Según las características de sus elementos, el conjunto de datos que conforman el cluster irán definiendo al conjunto. Cada cluster contará con un hipotético punto intermedio (el centroide), resultado de definir la media entre los objetos presentes en el grupo, que irá adaptándose hacia otro cluster según su cercanía a la media. [24].

Un algoritmo de clustering tiene como objetivo agrupar los objetos de un dataset según su similaridad, de forma que los objetos que hay dentro de un grupo (cluster) sean más similares que aquellos que caen en grupos distintos. Normalmente, para poder hablar de similaridad se suele acudir a algún tipo de distancia, con el fin de poder asociar la similitud de los objetos analizados con la distancia que hay entre ellos [25].

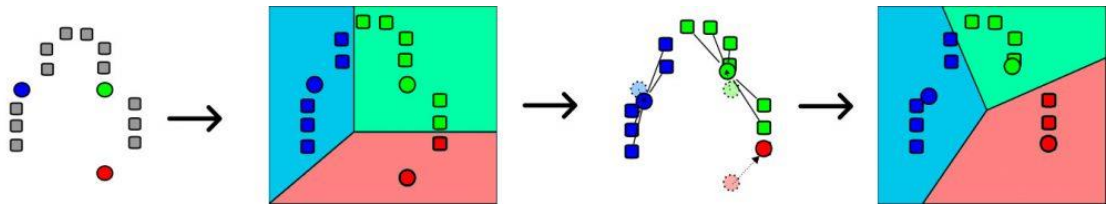


Figura 16: Esquema de funcionamiento de un algoritmo de clustering [24].

Cuando se inicia la creación de un cluster, la ubicación de los centroides puede ser totalmente aleatoria. Un centroide es el punto que ocupará la posición media del grupo final. Es muy probable que la primera asignación de un centroide no guarde ninguna relación con su ubicación final, por lo que ira cambiando a lo largo que el algoritmo vaya avanzando.

El propio algoritmo se encargará de ir realizando cálculos, modificando la ubicación del centroide hasta que se logre el equilibrio entre la media de sus valores y el conjunto que lo compone.

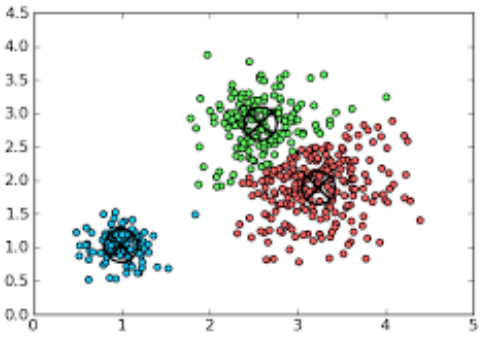


Figura 17: Ubicación final de los centroides en un conjunto de datos [26].

Según la forma en que los clusters se relacionan entre sí y con los objetos del dataset, podemos establecer una primera división entre los diversos algoritmos existentes:

- Clustering Duro: Cada objeto pertenece a un único cluster, por lo que los clusters se convierten en divisiones del conjunto de datos, dataset.
- Clustering Blando: Los objetos pertenecen a los clusters según un grado de confianza, por lo que pueden pertenecer a más de un cluster a la vez.

También es posible clasificar las técnicas disponibles dependiendo de cómo se relacionan con detalle:

- Partición estricta: Cada objeto pertenece únicamente y estrictamente a un cluster.
- Partición estricta con valor atípico: Puede haber objetos que no pertenecen a ningún cluster (los valores atípicos).
- Clustering con superposiciones: Un objeto puede pertenecer a más de un cluster.
- Clustering Jerárquico: Los clusters se ordenan jerárquicamente de forma que los objetos que pertenecen a un cluster también pertenecen a su cluster padre [25].

Una vez que se ha explicado el concepto teórico de la clusterización, vamos a pasar a explicar la técnica GMM que es la utilizada para realizar la clusterizacon de las diferentes especies de peces.

Los Gaussian Mixture Model (GMM), modelos de mezcla gaussiana, son un modelo probabilístico para representar subpoblaciones distribuidas normalmente dentro de una población general. Los modelos de mezcla en general no requieren saber a qué subpoblación pertenece un punto de datos, lo que permite que el modelo aprenda las subpoblaciones automáticamente. Dado que no se conoce la asignación de subpoblaciones, esto constituye una forma de aprendizaje no supervisado.

$$p(X) = \sum_{i=1}^{K} \phi_{yo} \mathcal{N}(X \mid \vec{\mu}_{yo}, \Sigma_{yo})$$

$$\mathcal{N}(X \mid \vec{\mu}_{yo}, \Sigma_{yo}) = \frac{1}{\sqrt{(2\pi)^K \mid \Sigma_{yo} \mid}} \operatorname{Exp}\left(-\frac{1}{2}(X - \vec{\mu}_{yo})^{\mathrm{T}} \Sigma_{yo}^{-1} (X - \vec{\mu}_{yo}) \right)$$

$$\sum_{i=1}^{K} \phi_{yo} = 1$$

Figura 18: Fórmula multidimensional del modelo de mezcla gaussiana (GMM) [27].

Los modelos de mezcla gaussiana se pueden usar para agrupar datos sin etiquetar de la misma manera que k-means. Sin embargo, hay dos ventajas importantes:

- En primer lugar, k-means no tiene en cuenta la covarianza. En dos dimensiones, la covarianza determina la forma de la distribución. El modelo K-means lo que hace es colocar un círculo en el centro de cada grupo, con un radio definido por el punto más distante en el clúster. Esto funciona bien para cuando sus datos son circulares. Sin embargo, cuando sus datos adquieren una forma diferente no es muy exacto. Por el contrario, los modelos de mezcla gaussianos pueden manejar incluso conglomerados muy alargados.

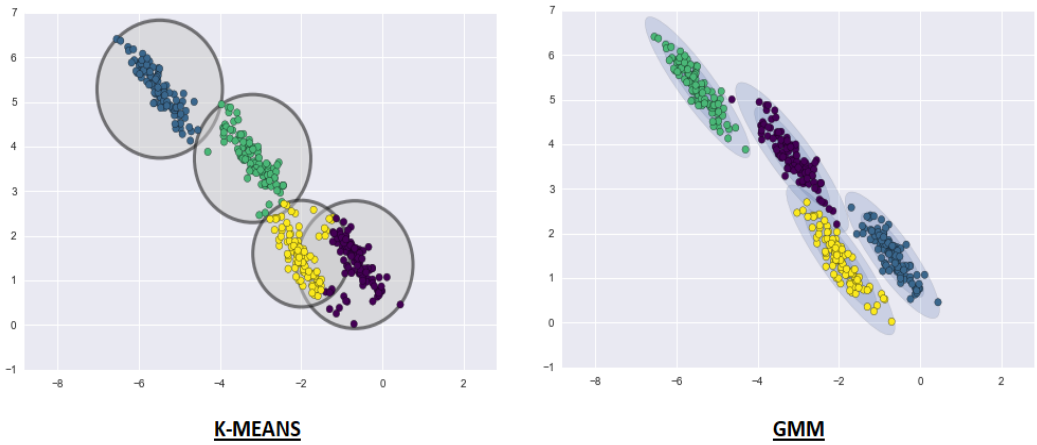


Figura 19: Comparativa de clusters creados con K-means y GMM.

- La segunda diferencia entre los modelos K-means y GMM es que el primero realiza una clasificación estricta mientras que el segundo realiza una clasificación suave. En otras palabras, K-means nos dice qué punto de datos pertenece a qué conglomerado, pero no nos proporcionará las probabilidades de que un punto de datos determinado pertenezca a cada uno de los posibles conglomerados [28].

Una parte esencial en la clusterizacion de objetos es determinar el número de cluster óptimos, para ello existe una forma llamada el método del codo.

El método del codo es una heurística que se utiliza para determinar el número de conglomerados en un conjunto de datos. El método consiste en graficar la variación explicada en función del número de grupos y elegir el codo de la curva como el número de grupos a utilizar. El mismo método se puede utilizar para elegir el número de parámetros en otros modelos basados en datos, como el número de componentes principales para describir un conjunto de datos. Usar el "codo" o la " rodilla de una curva " como punto de corte es una heurística común en la optimización matemática para elegir un punto en el que los rendimientos decrecientes ya no justifiquen el costo adicional. En la agrupación en clústeres, esto significa que uno debe elegir una serie de clústeres para que la adición de otro clúster no proporcione un mejor modelado de los datos [29].

Este método utiliza los valores de la inercia obtenidos tras aplicar el K-means a diferente número de Clusters (desde 1 a N Clusters), siendo la inercia la suma de las distancias al cuadrado de cada objeto del Cluster a su centroide.

$$Inercia = \sum_{i=0}^{N} \|x_i - \mu\|^2$$

Figura 20:Fórmula para calcular la inercia en el método del codo [30].

Una vez obtenidos los valores de la inercia tras aplicar el K-means de 1 a N Clusters, representamos en una gráfica lineal la inercia respecto del número de Clusters. En esta gráfica se debería de apreciar un cambio brusco en la evolución de la inercia, teniendo la línea representada una forma similar a la de un brazo y su codo. El punto en el que se observa ese cambio brusco en la inercia nos dirá el número óptimo de Clusters a seleccionar para ese data set [30].

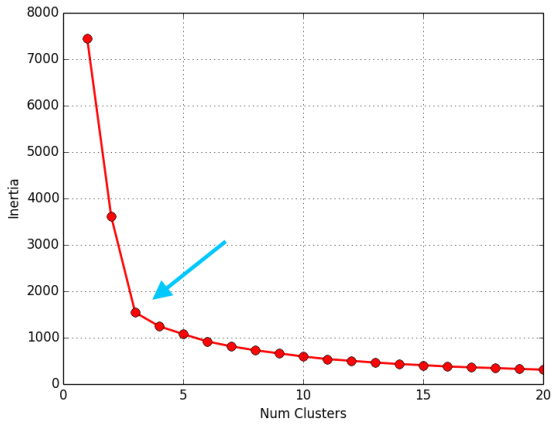


Figura 21: Gráfica que muestra la aplicación del método del codo.

El preclasificador lo que realiza es la separación de las imágenes entre 2 grupos, es decir para separar las imágenes entre peces y no peces. Esto se realiza mediante un modelo gaussiano a partir de imágenes positivas, conjunto de entrenamiento de 1000 imágenes de peces separadas manualmente y eliminadas del conjunto de imágenes de validación. Estas imágenes son comprimidas mediante un autoencoder.

Los autoencoders son redes neuronales con el objetivo de generar nuevos datos primero comprimiendo la entrada en un espacio de variables latentes y luego reconstruyendo la salida en base a la información adquirida. Este tipo de red consta de dos partes [31]:

1. Encoder: la parte de la red que comprime la entrada en un espacio de variables latentes y que puede representarse mediante la función de codificación h = f (x).
2. Decoder: la parte que trata de reconstruir la entrada en base a la información recolectada previamente. Se representa mediante la función de decodificación r = g (h).

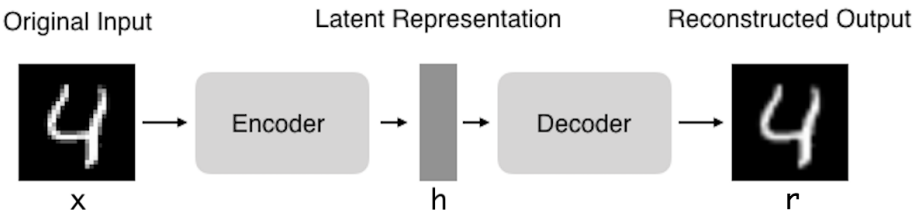


Figura 22: Esquema de funcionamiento de un autoencoder [31].

2.2.2 Preclasificador de imágenes mediante un modelo gaussiano

En este apartado explicare el procedimiento que he seguido para la realización del preclasificador de imágenes mediante un modelo gaussiano comprimiendo los datos mediante una red neuronal autoencoder.

Aunque antes voy a explicar las funciones principales utilizadas, así como sus parámetros para conocer mejor su funcionamiento y ejecutarlas de la manera más óptima para los requisitos de este proyecto. Y son las siguientes:

- sklearn.preprocessing.MinMaxScaler: Esta función transforma características escalando cada característica a un rango determinado. Este estimador escala y traduce cada característica individualmente de modo que esté en el rango dado en el conjunto de entrenamiento, por ejemplo, entre cero y uno. Esta transformación se usa a menudo como una alternativa a la escala de varianza unitaria de media cero. Los parámetros que tiene esta función son los siguientes [32]:

    - Feature_range: tupla (mínimo, máximo), predeterminado = (0, 1)
      Es el Rango deseado de datos transformados.

    - Copy: bool, predeterminado = True
      Se establece en False para realizar la normalización de filas en la misma variable y de esta manera evitar una copia.

- TensorFlow.keras.Sequential: Esta función agrupa una pila lineal de capas en a tf.keras.Model. Es decir, es una función de modelado de una red neuronal, en nuestro caso utilizada para crear la compresión del autoencoder. Tiene los siguientes parámetros [33]:

    - Layers: Es la lista opcional de capas para agregar al modelo.

    - Name: Es el nombre opcional del modelo.

- TensorFlow.keras.callbacks.EarlyStopping: Esta función detiene el entrenamiento cuando una métrica monitoreada haya dejado de mejorar. Tiene los siguientes parámetros [34]:

    - Monitor: Es la cantidad a monitorear.

    - Min_delta: Es el cambio mínimo en la cantidad monitoreada para calificar como una mejora, es decir, un cambio absoluto de menos de min_delta, no contará como mejora.

    - Patience: Es el número de épocas sin mejora después de las cuales se detendrá el entrenamiento.

    - Verbose: Es la activación del modo verbose.

    - Mode: Son los diferentes modos disponibles.
        - "auto"
        - "min"
        - "max

- Baseline: Es el valor de línea base para la cantidad monitoreada. El entrenamiento se detendrá si el modelo no muestra una mejora con respecto a la línea de base.

- Restore_best_weights: Si es False, se utilizan los pesos del modelo obtenidos en el último paso del entrenamiento.

- sklearn.mixture.GaussianMixture: Esta función realiza una representación de una distribución de probabilidad del modelo de mezcla gaussiana. Esta clase permite estimar los parámetros de una distribución de mezcla gaussiana, tal como se ha explicado en el apartado anterior. Esta función tiene los siguientes parámetros [35]:

  - N_components: int, el valor predeterminado es 1. Es el número de componentes de la mezcla.

  - Covariance_type: Es una cadena de texto que describe el tipo de parámetros de covarianza a utilizar. Debe ser uno de estos:
    - 'full': Cada componente tiene su propia matriz de covarianza general
    - 'tied': Todos los componentes comparten la misma matriz de covarianza general
    - 'diag': Cada componente tiene su propia matriz de covarianza diagonal
    - 'spherical': Cada componente tiene su propia varianza única

  - Tol: float, el valor predeterminado es 1e-3. Es el umbral de convergencia. Las iteraciones se detendrán cuando la ganancia promedio del límite inferior esté por debajo de este umbral.

  - Reg_covar: float, por defecto es 1e-6. Es la regularización no negativa añadida a la diagonal de covarianza. Permite asegurar que las matrices de covarianza sean todas positivas.

  - Max_iter: int, el valor predeterminado es 100. Es el número de iteraciones a realizar.

  - N_init: int, el valor predeterminado es 1. Es ll número de inicializaciones a realizar. Se guardan los mejores resultados.

  - Init_params: por defecto es 'kmeans'. Es el método utilizado para inicializar los pesos, las medias y las precisiones. Debe ser uno de:
    - 'kmeans'
    - 'random'

  - Weights_init: tipo matriz. Son los pesos iniciales proporcionados por el usuario, predeterminados son Ninguno. Si es None, los pesos se inicializan utilizando el método init_params.

  - Means_init: tipo matriz. Son los medios iniciales proporcionados por el usuario.

- Precisions_init tipo matriz. Son las precisiones iniciales proporcionadas por el usuario (inversas de las matrices de covarianza).

- Random_state int. Controla la semilla aleatoria dada al método elegido para inicializar los parámetros. Además, controla la generación de muestras aleatorias a partir de la distribución ajustada.

- Warm_start bool, por defecto False. Si 'warm_start' es True, la solución del último ajuste se usa como inicialización para la siguiente llamada de fit (). Esto puede acelerar la convergencia cuando se llama al ajuste varias veces en problemas similares. En ese caso, se ignora 'n_init' y solo se produce una inicialización en la primera llamada.

- Verbose int, predeterminado en 0. Habilite la salida detallada. Si es 1, imprime la inicialización actual y cada paso de iteración. Si es mayor que 1, también imprime la probabilidad de registro y el tiempo necesario para cada paso.

- Verbose_interval int, predeterminado en 10. Es el número de iteraciones realizadas antes de la siguiente impresión.

- sklearn.neighbors.DistanceMetric: Esta clase proporciona una interfaz uniforme para funciones métricas de distancia rápida. Se puede acceder a las diversas métricas a través del get_metric método de clase y el identificador de cadena de métricas.

Ahora que ya hemos explicado todos las funciones y métodos necesarios. Voy a explicar el desarrollo seguido, así como el código utilizado.

Lo primero que realice fue una separación manual de 1000 imágenes de peces para poder entrenar el preclasificador mediante un modelo gaussiano, es decir obtener la distancia gaussiana para las imágenes de peces y después aplicarlo a todo el conjunto de datos original sin las 1000 imágenes de peces.

Para realizarlo lo primero que hacemos es abrir la carpeta con las imágenes guardándola en una lista, para abrir las imágenes utilizamos la función *load_images_from_folder(path)* que cree para este fin. Y para poder utilizar las imágenes en cualquier función de machine learning como nuestro preclasificador tenemos que convertir las imágenes de una lista con un elemento por cada uno de los pixeles de la imagen en un matriz de 3 dimensiones con una fila por cada imagen aplanada. Tal como se muestra en el código a continuación:

```
##Leemos las imagenes de los peces
folderPeces = "C:/Users/David/Desktop/OTROS/TFM/PECES"

x = load_images_from_folder(folderPeces)

# convert list to numpy array
imagesize = len(x[0])*len(x[0][0])
print(len(x), imagesize)


X = np.zeros(shape=(len(x), imagesize)) # Este 3 es porque la imagen está en
RGB.
for i in range(len(x)):
    X[i] = np.array(x[i]).reshape(1,-1)

# X es vector donde cada fila es una imagen aplanada
print(X.shape)
```

Lo siguiente a realizar fue la compresión de las imágenes usando una red neuronal tipo autoencoder, para poder en los siguientes pasos saber la distribución gaussiana de estas imágenes. El autoencoder lo he realizado usando el transformador *MinMaxScaler()*, y utilizando 8 variables de compresión de los datos. Para saber que 8 era el numero optimo, realice diferentes pruebas con otros valores 4, 16, 32, con 4 valores la clasificación no era precisa y con valores más grandes de 8 no cambiaba sustancialmente. Y esta es el código realizado, ayudándome para crearlo en los manuales de la propia librería:

```
# AUTOENCODER
transformer = preprocessing.MinMaxScaler().fit(X)
X_transform = transformer.transform(X)
overfitCallback = EarlyStopping(monitor='loss', min_delta=0, patience = 100)
encoder            =            keras.models.Sequential([keras.layers.Dense(8,
input_shape=[X.shape[1]])])     # El 8 es el numero de variables de compresion.
Esta adaptado a nuestro sistema
decoder        =        keras.models.Sequential([keras.layers.Dense(X.shape[1],
input_shape=[8])])     # Idem con el 8
autoencoder = keras.models.Sequential([encoder, decoder])
autoencoder.compile(loss="mse", optimizer='adam')
history    =    autoencoder.fit(X_transform,    X_transform,    epochs=10000000,
callbacks=[overfitCallback], verbose=0)
X_reduced = encoder.predict(X_transform)
```

A continuación, lo siguiente que realizamos fue calcular el modelo gaussiano que tenían las imágenes de peces, es decir separamos manualmente 1000 imágenes de peces y las usamos para que nuestro algoritmo de preclasificación calculase la distancia de mahalanobis, para saber las características de las imágenes de peces. De esta manera obtenemos un modelo bastante exacto de las imágenes de peces que luego podemos para clasificar las demás imágenes. Para ello lo realizamos con la función de clusterizacion de GMM con el fit a una única variable. Además, como margen de error utilizamos 0,5 es decir admitimos que la distancia hasta nuestro conjunto de datos sea la mitad que el punto más alejado de esta manera, evitamos que las imágenes que no son muy exactas como las que contienen varios peces entre dentro de nuestro conjunto de peces. El código utilizado es el siguiente:

```
# GAUSSIANA
gmm = GaussianMixture(n_components=1).fit(X_reduced)
mu = gmm.means_[0].tolist()
mahalanobis = DistanceMetric.get_metric('mahalanobis', V=gmm.covariances_[0])

distanciaumbral = float('-inf')
for i in range(X_reduced.shape[0]):
     x = X_reduced[i]
     puntos = [mu, x]
     distancia = mahalanobis.pairwise(puntos)[0, 1]
     if distancia>distanciaumbral:
         distanciaumbral = distancia

distanciaumbral *= 0.5   # Admitimos un margen del 50% con respecto a la distancia
umbral.
```

Ahora lo siguiente que hay que hacer es abrir todo el conjunto de imágenes extraídas del video, no únicamente las imágenes de peces, para que de esta forma podamos aplicar el clasificador que acabamos de crear sobre estas imágenes para separar en peces y no peces. Una vez que hemos abierto las imágenes como hemos hecho anteriormente, utilizando la función *load_images_from_folder(path)* y convirtiendo las de una lista a un array para poder hacer el clustering. Las imágenes están listan para pasarlas por nuestro clasificador.

El clasificador funciona de la siguiente manera, una vez que ya tenemos el modelo GMM creado tenemos que clasificar las imágenes como pez/no pez. Para ello cargamos en una matriz nueva X con esas imágenes, la codificamos con el autoencoder, utilizando los mismos criterios que antes. Y voy recorriendo imagen a imagen con el bucle y si la distancia de esa imagen es mayor a la distancia umbral la imagen no es un pez si es menor si lo es. El código creado es el siguiente:

```
Peces= []
IndexPeces = numpy.zeros(shape=(1))
XX_transform = transformer.transform(XX)
XX_reduced = encoder.predict(XX_transform)
count = 0
for i in range(XX_reduced.shape[0]):
     xx = XX_reduced[i]
     puntos = [mu, xx]
     distancia = mahalanobis.pairwise(puntos)[0, 1]
     if distancia>distanciaumbral:
         # NO PEZ
         # NO HAGO NADA
       count = count
     else:
        # SI PEZ
         # AÑADO esa imagen (XX[i]) a la matriz de imagenes PECES
        count+=1
        Peces.append(xx1[i])
```

Lo último que hay que hacer es guardar las imágenes clasificadas en una carpeta para poder utilizarla para los siguientes pasos. Y en caso de que ejecutar el código por partes poder tenerlas como copia. Para ello vamos recorriendo en un bucle la lista de imágenes resultante y utilizando la función de guardar de openCV se guardan en la ruta indicada.

```
## Ahora guardamos los peces en una carpeta
count= 0
for imagen in Peces:
    cv2.imwrite("C:/Users/David/Desktop/OTROS/TFM/SalidaPeces/"              +
"\\frame%d.jpg" % count, Peces[count])
    count+=1
```

Y para terminar y poder comprobar que el resultado es satisfactorio, hemos utilizado una función que dado una matriz de imágenes te muestra un rango temporal de ellas.

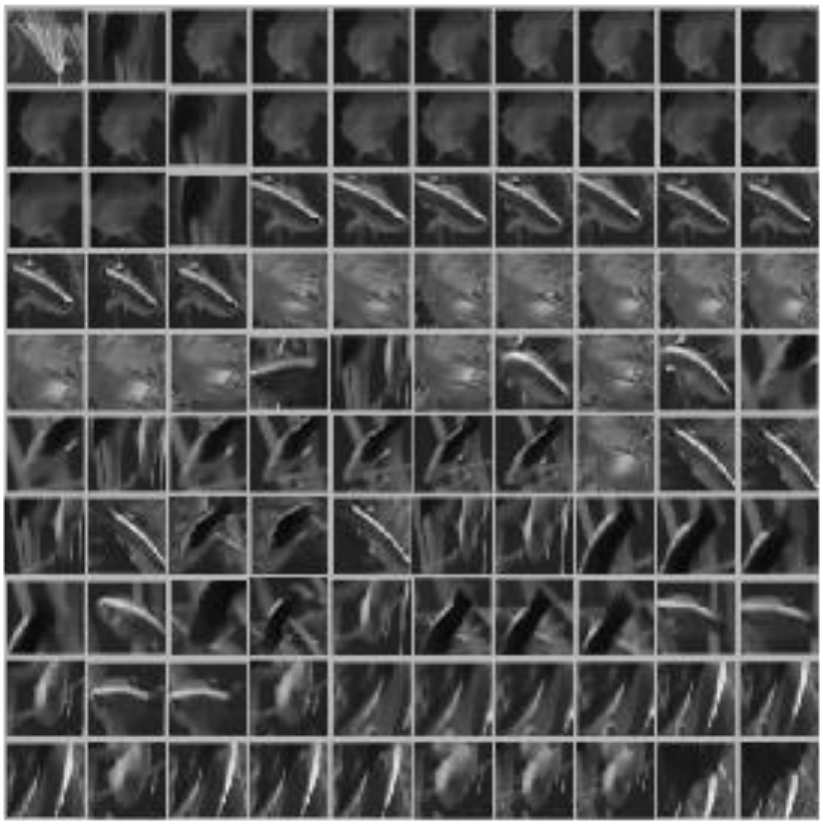


Figura 23: Resultado de la preclasificación, 100 primeras imágenes de peces.

2.2.3 Clusterizacion de las distintas imágenes usando GMM

En este último apartado explicare el procedimiento seguido para realizar la clusterizacion de los datos ya preclasificados utilizando para ello el método GMM. De esta manera separe los diferentes peces según sus especies de una manera no supervisada.

Las funciones necesarias para este apartado y que no se hayan explicado en puntos anteriores de este trabajo es [36]:

- sklearn.cluster.KMeans: Esta clase permite realizar una clusterizacion mediante K-means. Los parámetros que utiliza son los siguientes:
    - N_clusters int, es el número de clústeres que se formarán, así como el número de centroides que se generarán.
    - Init: Es el método de inicialización elegido para la selección de los centroides.
    - N_init int, predeterminado = 10. Es el número de veces que se ejecutará el algoritmo de k-medias con diferentes semillas de centroide.
    - Max_iter int, predeterminado = 300. Es el número máximo de iteraciones del algoritmo k-means para una sola ejecución.
    - Tol float, predeterminado = 1e-4. Es la tolerancia relativa con respecto a la norma de Frobenius de la diferencia en los centros de los conglomerados de dos iteraciones consecutivas para declarar convergencia.
    - Precompute_distances: Es para activar o desactivar el calculo previo de las distancias, que acelera el procesamiento, pero requiere más memoria.
    - Verbose: Es la activación del modo verbose.
    - Random_state int, instancia de RandomState. Determina la generación de números aleatorios para la inicialización del centroide.
    - Copy_x bool, predeterminado = True. Cuando se calculan previamente las distancias, es numéricamente más preciso centrar los datos primero. Si copy_x es True (predeterminado), los datos originales no se modifican. Si es Falso, los datos originales se modifican y se devuelven antes de que vuelva la función, pero se pueden introducir pequeñas diferencias numéricas restando y luego sumando la media de los datos
    - N_jobs int, predeterminado = Ninguno. Es la cantidad de subprocesos OpenMP que se utilizarán para el cálculo.

- Algoritmo: Es el algoritmo K-means a utilizar. El algoritmo clásico de estilo EM es "completo". La variación "elkan" es más eficiente en datos con grupos bien definidos, al usar la desigualdad del triángulo. Sin embargo, consume más memoria debido a la asignación de una matriz adicional de formas (n_samples, n_clusters).

Ahora que ya hemos explicado las funciones necesarias vamos a pasar a explicar paso a paso todos los procesos realizados para obtener la clusterizacion de los peces en especies.

Lo primero a realizar es abrir s imágenes resultantes del algoritmo de preclasificación gaussiano para saber si una imagen es un pez o no. Para ello utilizamos la función creada previamente para abrir las imágenes dado una ruta. Y después las convertimos de una lista a un array de numpy. El código utilizado es el siguiente:

```
##Lo primero que haremos será abrir las imágenes de SalidaPeces y transformarlos.
##Leemos las imágenes de los peces
folderTodos = "C:/Users/David/Desktop/OTROS/TFM/SalidaPeces"

yy = load_images_from_folder(folderTodos)

import numpy as np
# convert list to numpy array
imagesize = len(yy[0])*len(yy[0][0])
print(len(yy), imagesize)


YY = np.zeros(shape=(len(yy), imagesize)) # Este 3 es porque la imagen está en
RGB. DEBE CAMBIARSE A ESCALA DE GRISES Y ELIMINARLO
# Cada fila debe ser una imagen.
for i in range(len(yy)):
    YY[i] = np.array(yy[i]).reshape(1,-1)

# YY está formada. ¡ES UNA MATRIZ DONDE CADA FILA ES UNA IMAGEN APLANADA!
print(YY.shape)
```

Lo siguiente que realizaremos será aplicar el método del codo para saber el número de agrupaciones óptimas para realizar la clusterización. Para ello haremos uso del algoritmo de K-means que introduciremos en un bucle que ira variando el número de clústers y lo graficaremos junto con la inercia.

```
##Ahora aplicamos el metodo del codo con SalidaPeces para encontrar el numero
óptimo de clústers.
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 0).fit(YY)
    wcss.append(kmeans.inertia_)

# Grafica de la suma de las distancias
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

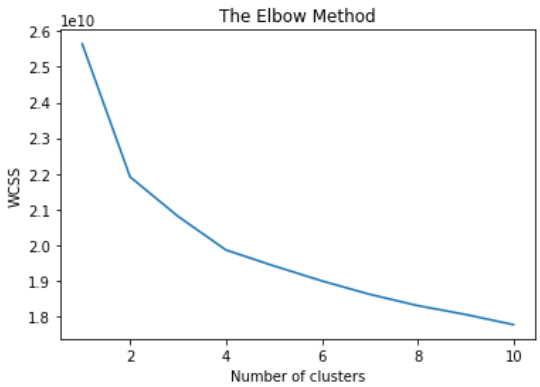


Figura 24: Salida de la aplicación del método del codo sobre las imágenes de Peces.

Como se puede comprobar en la figura de arriba el número de clusters óptimos según este método será 4. Que es el numero de agrupaciones que utilizaremos a continuación.

Lo siguiente a realizar será aplicar el método de clusterizacion de GMM con 4 agrupaciones a todo el conjunto de datos. El código utilizado es el siguiente:

```
## Ahora que ya sabemos que el número de clúster optimo es 4, aplicamos una
clusterizacion utilizando GMM con 4 clusters
## Hacemos el clustering con la tecnica GaussianMixture

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=4, random_state=0).fit(YY) # EL FIT ES CON
LA MATRIZ YY
labels = gmm.predict(YY)
```

Y lo último realizado es utilizando la función creada previamente para graficar un conjunto de imágenes, visualizar todas las imágenes de los clusters para comprobar que la agrupación ha tenido éxito. Para ello filtramos los datos según el cluster asignado en la variable labels, en este caso 0,1,2 y 3.

```
## Mostramos el 1º cluster para comprobar si la separación ha funcionado

plt.figure(figsize=(20,5))
aux=YY[labels==0]
plot_digits(aux[:100], images_per_row=10)
plt.show()
```

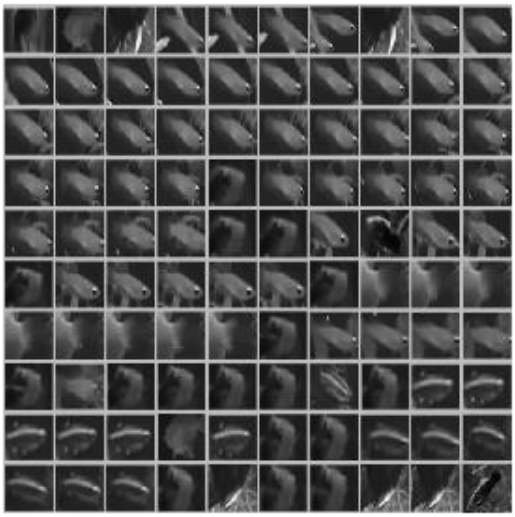


Figura 25: Imágenes pertenecientes al 2º cluster.

3 Revisión y análisis de los resultados obtenidos

En este apartado analizaré los resultados obtenidos tanto en el apartado de análisis y extracción de las características de un video como en la clasificación y posterior agrupación de las imágenes.

En el apartado de análisis de las características del video se ha realizado una función que analiza el movimiento en cada fotograma según los contornos que se mueve y recorta en un rectángulo ese objeto en movimiento.

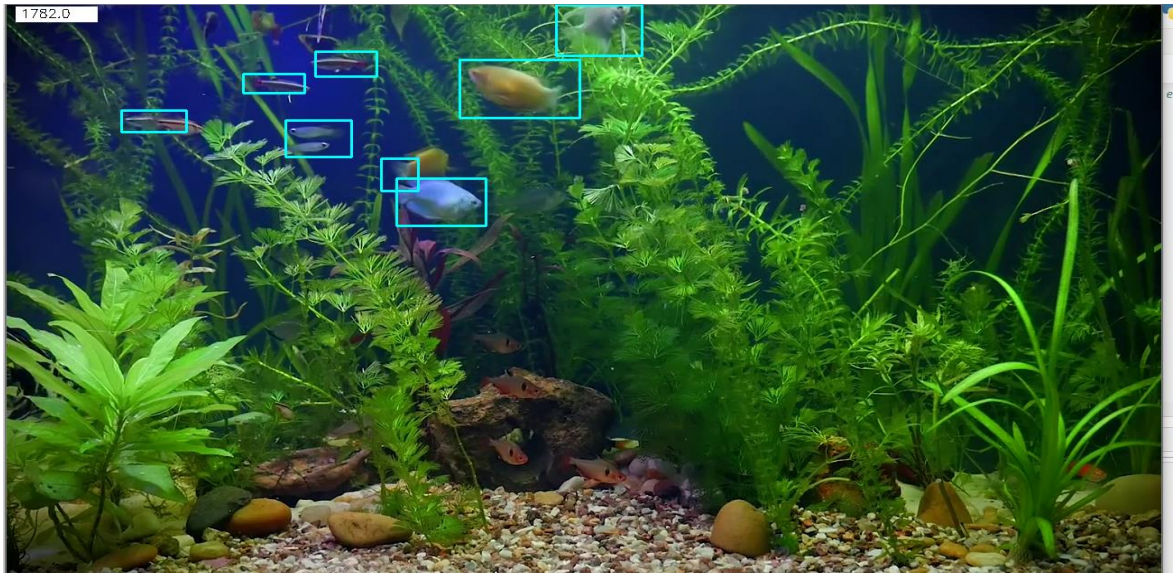


Figura 26: Fotograma resultante 1 del análisis de movimiento con OpenCV.

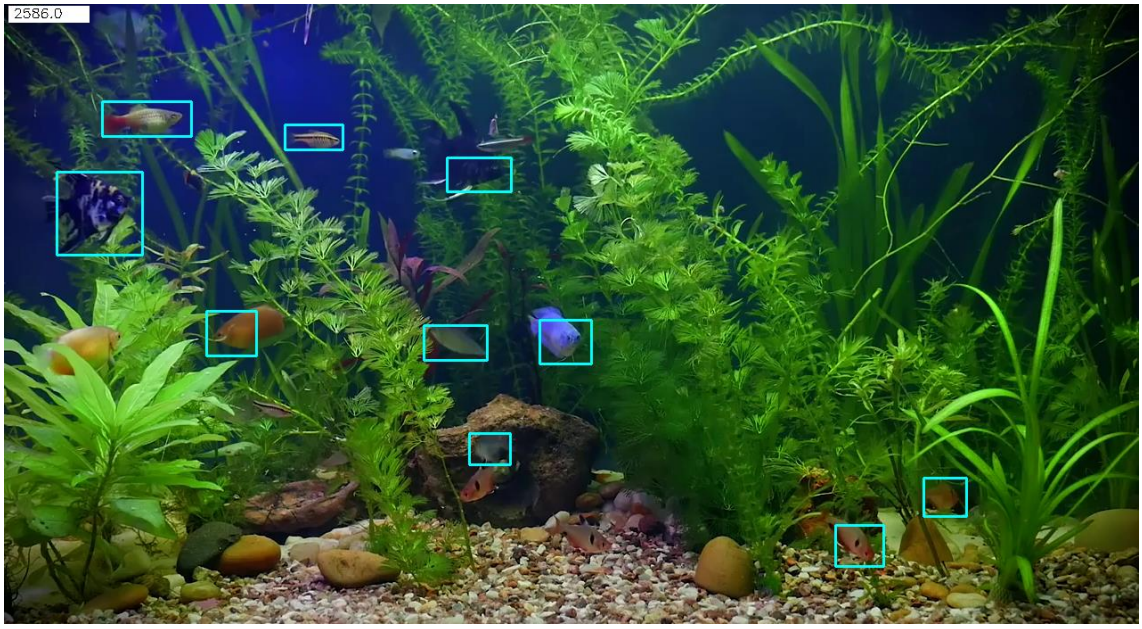


Figura 27: Fotograma resultante 2 del análisis de movimiento con OpenCV.

Como se puede comprobar en las figuras arriba expuestas nuestro sistema función de una manera bastante precisa para la detección de movimiento, es decir para la detección de los nuevos peces que aparecen en la escena. Aunque fue necesario un ajuste inicial de los parámetros, tal como se ha explicado en apartados anteriores de este trabajo, actualmente nuestro sistema funciona de una forma muy precisa y rápida realizándolo en tiempo real. Para poner los resultados en perspectiva en el video utilizado de ejemplo de unos 5 minutos de duración nuestra función nos detecte más de 11.000 objetos en movimiento.

En el apartado siguiente voy a analizar el resultado de la preclasificación de imágenes realizada mediante el modelo gaussiano para separar las imágenes en 2 grupos, peces y no peces. Anteriormente esta clasificación se realizaba mediante una clusterizacion sencilla de K-means o GMM pero al comprobar que no daba los resultados tan precisos y satisfactorios como se requería se paso a realizar un preclasificación codificando las imágenes con una red neuronal autoencoder, a continuación entrenar nuestro algoritmo que realiza un modelo gaussiano con 1000 imágenes de peces para a continuación realizar esta clasificación con todas las imágenes eliminando las 1000 utilizadas para entrenar el sistema.

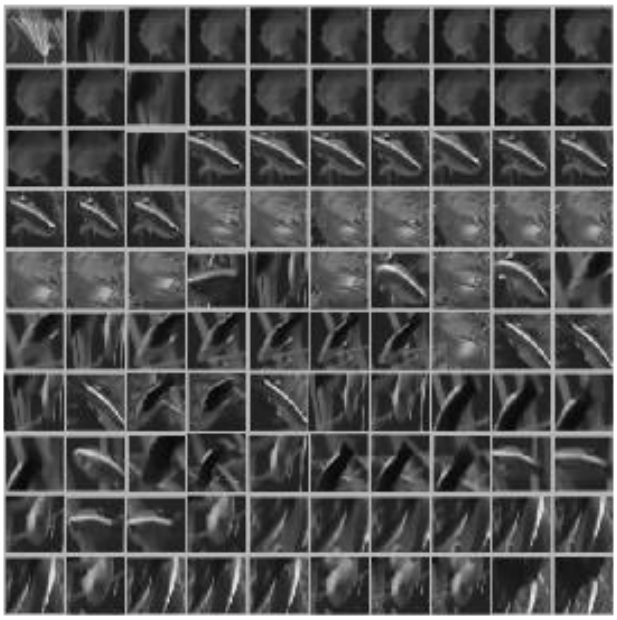


Figura 28: Resultado 1 de la preclasificación de las imágenes para saber si son Peces o No.

Figura 29 Resultado 2 de la preclasificación de las imágenes para saber si son Peces o No.

Como se puede comprobar en las imágenes de arriba, la clasificación función de una manera bastante precisa pudiendo eliminar las imágenes capturadas erróneamente por nuestra función del análisis del video. Como es normal hay alguna imagen que no es totalmente de peces que esta dentro y es debido al ajuste de nuestro sistema. Se han ajustado los parámetros del sistema para que permita pasar únicamente las imágenes que se alejan un 0,5 de la imagen clasificada como pez mas distante: Es decir se permite una similitud de la imagen al 50% con la imagen mas alejada. Y se ha configurado de la siguiente forma para no dejar pasar las imágenes que contienen varios peces en el mismo objeto, esto se produce cuando los peces se cruzan y las imágenes se sobreponen.

Con todo esto nuestro sistema funciona de una manera muy precisa, de las 10.000 imágenes introducidas en nuestro clasificador, nuestro algoritmo es

capaz de detectar más de 5.000 imágenes de peces, siendo la mayoría de ellas correctas. Por lo que una vez visto los resultados puedo afirmar que nuestro clasificador funciona muy bien y que esta parte del proyecto ha sido un éxito.

Y para terminar este apartado el ultimo análisis de resultado a realizar es el de la clusterizacion de diferentes especies o formas de peces. Para realizar esta clusterizacion primero se realiza el método del codo para obtener el numero de clusters óptimo para nuestro conjunto de datos, es este caso particular 4. Y a continuación mediante el algoritmo GMM se realiza ese clusterizacion, anteriormente se realizaba con el algoritmo de K-means pero se modifico por el GMM ya que nos proporcionaba mejores resultados.

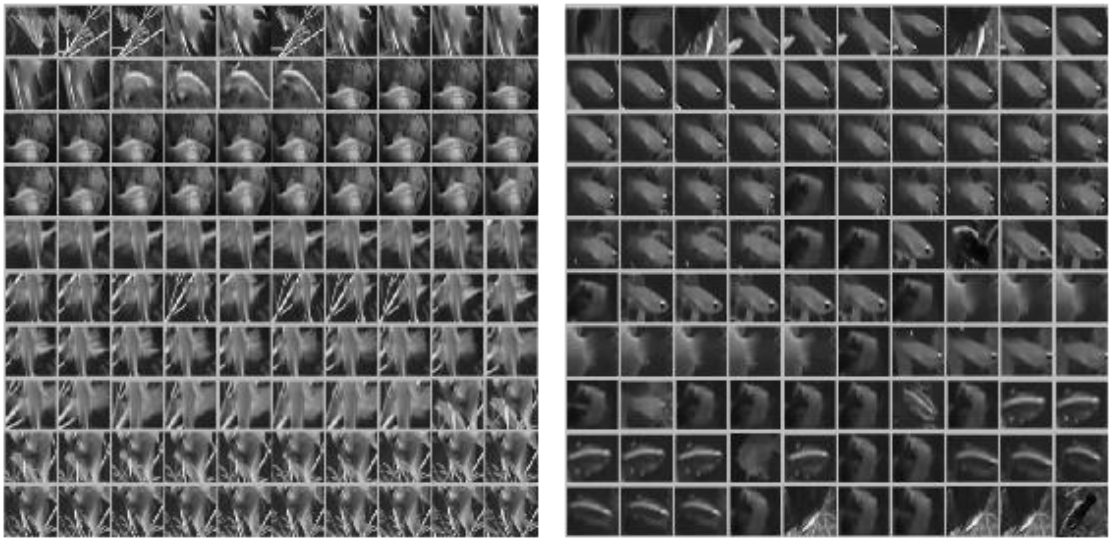


Figura 30: Resultados de los Cluster 1 y 2 de la separación de los peces en especies.

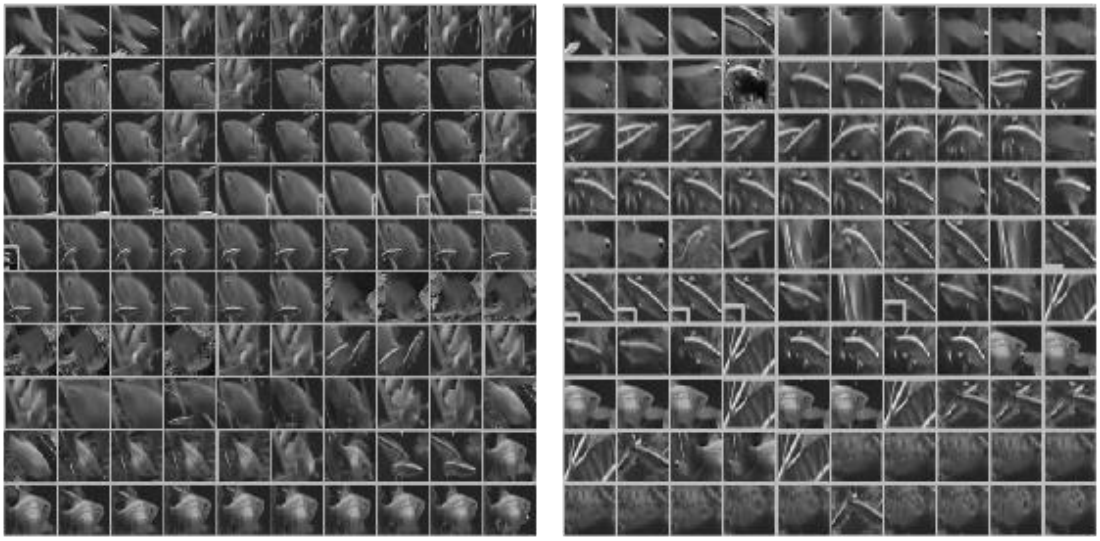


Figura 31: Resultados de los Cluster 3 y 4 de la separación de los peces en especies.

Como se puede ver en las figuras anteriores nuestro algoritmo es capaz de detectar y separar la mayoría de las especies de peces que aparecen en nuestras imágenes. El primer cluster es la especie de pez mediana amarilla, en el segundo cluster hay varias especies peces pequeños de colores, que son muy similares, en el tercer cluster son los peces grandes y en el ultimo son los peces mas pequeños, la especie llamada neones, aunque en este cluster y debido a la similitud de estos peces con algunas de las ramas pasadas erróneamente del algoritmo anterior también las añade a este cluster.

Por lo que se puede establecer de que una manera bastante precisa nuestro sistema es capaz de separar las especies o grupos de especies que más se parecen, por lo que personalmente que esta parte era la mas complicada de poder realizar se ha cumplido con total éxito.

Para resumir este apartado y teniendo en cuenta todos los resultados obtenidos, así como la salida del sistema, se puede afirmar que nuestro sistema en su conjunto funciona de una manera muy precisa pudiendo funcionar de una manera semiautónoma, es posible si añadimos mas algoritmos como una clusterizacion previa nuestro sistema funcione de una manera aun mas precisa. Pero dado el alcance de este proyecto creo que personalmente los resultados son muy satisfactorios.

4 Conclusión

Para concluir el trabajo es necesario evaluar si se ha tenido éxito en los objetivos propuestos inicialmente y en este caso así es. Se han cumplido con éxito todos los objetivos planteados ya que se ha podido realizar todo el sistema con éxito usando solo las capacidades del lenguaje de programación Python y las librerías mas importantes.

Se han demostrado y estudiado con creces las características de la librería de OpenCV permitiendo con esta librería poder realizar en tiempo real el análisis de los contenidos de un video y su posterior transformación y adaptación de las imágenes para poder convertirlas en nuestro banco de datos con el que realizar la posterior clasificación y clusterización.

Además, se ha demostrado y comprobado la posibilidad de realizar una clasificación y agrupación de las imágenes según sus características, en este caso las especies de peces aparecidas en la imagen con bastante precisión. Abriendo la puerta a que nuestro sistema de forma autónoma vaya trabajando para poder extraer información de todos los videos.

Por lo que para cerrar el trabajo se puede afirmar que se han cumplido todos los objetivos propuestos, y otros que no estaban previstos inicialmente como ampliar mis conocimientos de las diferentes librerías utilizadas y la aplicación de los algoritmos de inteligencia artificial.

5 Bibliografía

[1] J. M. LÓPEZ-ZAFRA, «El Confidencial,» 31 12 2019. [En línea]. Available: https://blogs.elconfidencial.com/economia/big-data/2019-12-31/20-datos-para-2020_2394980/. [Último acceso: 29 09 2020].

[2] Wikipedia, «OpenCV,» 7 Agosto 2020. [En línea]. Available: https://es.wikipedia.org/wiki/OpenCV. [Último acceso: 18 Noviembre 2020].

[3] R. Marín, «¿Qué es OpenCV? Instalación en Python y ejemplos básicos,» Revistadigital, 12 Febrero 2020. [En línea]. Available: https://revistadigital.inesem.es/informatica-y-tics/opencv/. [Último acceso: 18 Noviembre 2020].

[4] Open CV, «About,» Open CV, [En línea]. Available: https://opencv.org/about/. [Último acceso: 11 Noviembre 2020].

[5] L. M. Garcia, «¿Qué es OpenCV?,» unpocodejava.com, 9 Octubre 2013. [En línea]. Available: https://unpocodejava.com/2013/10/09/que-es-opencv/. [Último acceso: 18 Noviembre 2020].

[6] OpenCV, «cv::VideoCapture Class Reference,» OpenCV Documentation, 21 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html#a949d90b766ba42a6a93fe23a67785951.

[7] O. Documentation, «Open CV Documentation Erode and Dilate,» OpenCV, 21 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaeb1e0c1033e3f6b891a25d0511362aeb.

[8] Unipython, «Transformaciones Morfologicas,» Unipython, 5 Abril 2018. [En línea]. Available: https://unipython.com/transformaciones-morfologicas/. [Último acceso: 21 Noviembre 2020].

[9] Python Examples, «Python OpenCV cv2 Find Contours in Image,» 23 Noviembre 2020. [En línea]. Available: https://pythonexamples.org/python-opencv-cv2-find-contours-in-image/.

[10] OpenCV, «Contour Approximation Modes OpenCV,» 23 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga4303f45752694956374734a03c54d5ff.

[11] OpenCV DOCS, «Structural Analysis and Shape Descriptors,» OpenCV, 23 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0.

[12] OpenCV Docs, «Structural Analysis and Shape Descriptors,» OpenCV, 26 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#ga2c759ed9f497d4a618048a2f56dc97f1.

[13] OpenCV Docs, «Drawing Functions,» OpenCV, 2020 Noviembre 25. [En línea]. Available: https://docs.opencv.org/master/d6/d6e/group__imgproc__draw.html.

[14] OpenCV, «Contours Hierarchy,» OpenCV, 25 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html.

[15] OpenCV Docs, «How to Use Background Subtraction Methods,» OpenCV, 28 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html.

[16] OpenCV Docs, «Motion Analysis,» OpenCV, 28 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/de/de1/group__video__motion.html.

[17] Wikipedia, «Distancia de Mahalanobis,» Wikipedia, 5 Abril 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Distancia_de_Mahalanobis. [Último acceso: 28 Noviembre 2020].

[18] OpenCV Doc, «BackgroundSubtractor Class Reference,» OpenCV, 28 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/3.4/d7/df6/classcv_1_1BackgroundSubtractor.html.

[19] OpenCV Docs, «Miscellaneous Image Transformations,» OpenCV, 29 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57.

[20] OpenCV Docs, «Image Thresholding,» OpenCV, 29 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html.

[21] OpenCV Docs, «Geometric Transformations of Images,» OpenCV, 30 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/da/d6e/tutorial_py_geometric_transformations.html.

[22] OpenCV Docs, «Color Space Conversions,» OpenCV, 30 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab.

[23] OpenCV Docs, «Image file reading and writing,» OpenCV, 30 Noviembre 2020. [En línea]. Available: https://docs.opencv.org/master/d4/da8/group__imgcodecs.html#gabbc7ef1aa2edfaa87772f1202d67e0ce.

[24] Avansis, «¿QUÉ SIGNIFICA CLUSTERING Y K MEANS EN INTELIGENCIA ARTIFICIAL?,» Avansis, 2 Diciembre 2020. [En línea]. Available: https://www.avansis.es/inteligencia-artificial/que-significa-clustering-y-k-means-en-inteligencia-artificial/.

[25] F. S. Caparrini, «Algoritmos de Clustering,» Dpto. de Ciencias de la Computación e Inteligencia Artificial Universidad de Sevilla , 12 Diciembre 2019. [En línea].

[25] Available: http://www.cs.us.es/~fsancho/?e=230. [Último acceso: 2 Diciembre 2020].

[26] J. Rocha, «¿Qué es el clustering ?,» Planetachatbot, 28 Febrero 2020. [En línea]. Available: https://planetachatbot.com/que-es-clustering-5568d52286a1. [Último acceso: 2 Diciembre 2020].

[27] G. P. A. D. John McGonagle, «Gaussian Mixture Model,» Brilliant, 5 Diciembre 2020. [En línea]. Available: https://brilliant.org/wiki/gaussian-mixture-model/#:~:text=Gaussian%20mixture%20models%20are%20a,subpopulations%20within%20an%20overall%20population.&text=A%20model%20making%20this%20assumption,have%20more%20than%20two%20components..

[28] C. Maklin, «Gaussian Mixture Models Clustering Algorithm Explained,» Towards Data Science, 5 Diciembre 2020. [En línea]. Available: https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e.

[29] Wikipedia, «Elbow_method_(clustering),» Wikipedia, 21 Abril 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Elbow_method_(clustering). [Último acceso: 5 Diciembre 2020].

[30] R. Moya, «Selección del número óptimo de Clusters,» Jarroba, 26 Septiembre 2016. [En línea]. Available: https://jarroba.com/seleccion-del-numero-optimo-clusters/. [Último acceso: 5 Diciembre 2020].

[31] N. Hubens, «Introducción al autoencoder,» DLI, 8 Mayo 2018. [En línea]. Available: https://www.deeplearningitalia.com/introduzione-agli-autoencoder-2/. [Último acceso: 5 Diciembre 2020].

[32] Scikit learn, «sklearn.preprocessing.MinMaxScaler,» Scikit learn, 5 Diciembre 2020. [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[33] TensorFlow, «tf.keras.Sequential,» TensorFlow, 6 Diciembre 2020. [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential#args_1.

[34] TensorFlow, «tf.keras.callbacks.EarlyStopping,» TensorFlow, 6 Diciembre 2020. [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping#arguments_1.

[35] Scikit learn, «sklearn.mixture.GaussianMixture,» Scikit learn, 5 Diciembre 2020. [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html.

[36] Scikit learn, «sklearn.cluster.KMeans,» Scikit learn, 8 Diciembre 2020. [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

6 Anexo I: Código completo del proyecto

In [245]:

```
import cv2 as cv
import cv2
import matplotlib.pyplot as plt
```

In [246]:

```
## Definimoslos parametros
path = "C:/Users/David/Desktop/OTROS/TFM/pecera.mp4"
tipo = "KNN"
learning_rate = 0.01
pathOut = "C:/Users/David/Desktop/OTROS/TFM/FinalFramesNew"
```

In [249]:

```
def CreateImages (path, tipo, learning_step, pathOut):

    """ Funcion que dado un video situado en una ruta le elimina el background haciendo uso de la libreria opencv
        y mete en un recuadro el objeto en movimiento. Recorta de este recuadro el objeto en movimeinto, lo normaliza y lo guarda
        en la ruta de salida con fondo y sin fondo de la imagen.

        :params path: Ruta en local donde se situa el video
        :type path: string
        :params tipo: Metodo a aplicar para eliminar el background KNN ó MOG2
        :type tipo: string
        :params learning_rate:
        :type learning_rate: double
        :params pathOut: Ruta de salida donde guarda los imagenes resultado
        :type learning_rate: string

    """
    ## Creamos un objeto que va a generar la foreground mask. Aqui utilizamos los parametros por defecto, pero se pueden parametrizar.

    if tipo == 'MOG2':

        bg_subtractor = cv2.createBackgroundSubtractorMOG2(detectShadows=True)

    else:
        bg_subtractor = cv2.createBackgroundSubtractorKNN(detectShadows=True)

    erode_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 5))
    dilate_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (17, 11))

    ## Utilizamos videocapture para leer el video origen
    cap = cv2.VideoCapture(path)
    success, frame = cap.read()
    count = 0


    ## Comprobacion de que sea leido bien el video
    if not cap.isOpened:
        print('Unable to open: ' + path)
        exit(0)


    while success:

         ## Cada fotograma se utiliza para calcular la foreground mask y actualizar el fondo.
         ### Se ha utilizado el learning rate por defecto pero se podria cambiar pasandoselo como parametro de entrada a la funcion apply.
        fg_mask = bg_subtractor.apply(frame, learningRate=learning_rate)

        ## Cuando pasamos el fotograma al metodo de eliminacion del background, el substractor actualiza su modelo interno del
        ## del background y devuelve una mascara. La mascara es blanca (255) para los segmentos del foreground, gris(127) para
        ## las sombras  y negro(0) para el background. En este caso tratamos las sombras como el background, asi que aplicamos
```

```
        ## un liminte cercano a blanco a la mascara(244).
        _, thresh = cv2.threshold(fg_mask, 244, 255, cv2.THRESH_BINARY)


        ## Detectamos el contorno de las imagenes para ver los objetos en movimiento
        cv2.erode(thresh, erode_kernel, thresh, iterations=2)
        cv2.dilate(thresh, dilate_kernel, thresh, iterations=2)
        contours, hier = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)

        ## A partir de los contornos dibujamos el rectangulo alrededor de ese objeto
        for c in contours:
            if cv2.contourArea(c) > 1000:
                x, y, w, h = cv2.boundingRect(c)
                cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 0), 2)

        ## Recortamos el cuadrado y lo guardamos en una nueva imagen
        frame1 = frame[y:y+h, x:x+w]
        frame1_Mask = fg_mask[y:y+h, x:x+w]

        #Modificamos el tamño de la imagen para que tengan un valor fijo
        frame1_resized = cv2.resize(frame1,(128,128))
        frame1_Mask_resized = cv2.resize(frame1_Mask,(128,128))

        ## Convertimos la imagen recortada en escala de grises, solo la que tiene fondo
        frame1_resized_gray = cv2.cvtColor(frame1_resized, cv2.COLOR_BGR2GRAY)

        ##Guardamos esa imagen en la carpeta
        cv2.imwrite( pathOut + "\\frameCrop%d.jpg" % count, frame1_resized_gray)
        cv2.imwrite( pathOut + "\\frameCropMask%d.jpg" % count, frame1_Mask_resized)

        ## Extraemos el numero de fotograma del objeto cv::VideoCapture y lo ponemos en la esquina
superior izquierda en un rectangulo blanco.
        cv2.rectangle(frame, (10, 2), (100,20), (255,255,255), -1)
        cv2.putText(frame, str(cap.get(cv2.CAP_PROP_POS_FRAMES)), (15, 15),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5 , (0,0,0))

        ## Mostramos el video de entrada con el recuadro y los resultados
        cv2.imshow(tipo, fg_mask)
        cv2.imshow('thresh', thresh)
        cv2.imshow('detection', frame)


        ## Guardamos la imagen de fondo con el recuadro para realizar comprobaciones
        cv2.imwrite( pathOut + "\\frame%d.jpg" % count, frame)

        count = count + 1

        keyboard = cv2.waitKey(30)
        if keyboard == 'q' or keyboard == 27:
            break
        success, frame = cap.read()
```

In [250]:

```
CreateImages(path, tipo, learning_rate, pathOut)
```

In [3]:

```
## Definimos la funcion de carga de imagenes
import cv2
import os

def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        if img is not None:
            images.append(img)
    return images
```

In [188]:

```
##Definimos la funcion para graficar un conjunto de imagenes

def plot_digit(data):
    image = data.reshape(64, 64)
    plt.imshow(image, cmap = mpl.cm.binary,
               interpolation="nearest")
    plt.axis("off")

def plot_digits(instances, images_per_row=10, **options):
    size = 64
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = 'gray', **options)
    plt.axis("off")
```

In [4]:

```
### Nuevo codigo
```

In [76]:

```
##1. Eligimos 1000 imagenes de peces VARIADAS y la metemos en una carpeta MODELO_PECES eliminando
de la carpeta de IMAGENES
#2. Creamos un modelo gaussiano de las imagenes MODELO_PECES.
#Para ello hacemos una autoencoder de keras y un fit a una gmm de 1 componente


import keras
from keras import layers
from keras import regularizers
from keras.callbacks import EarlyStopping
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import DistanceMetric
import numpy as np
```

In [6]:

```
##Leemos las imagenes de los peces
folderPeces = "C:/Users/David/Desktop/OTROS/TFM/PECES"


x = load_images_from_folder(folderPeces)

## Convertimos nuestro array en un vector para poder hacer el clustering
import numpy as np
# convert list to numpy array
imagesize = len(x[0])*len(x[0][0])
print(len(x), imagesize)


X = np.zeros(shape=(len(x), imagesize)) # Este 3 es porque la imagen está en RGB.
# Cada fila debe ser una imagen.
for i in range(len(x)):
    X[i] = np.array(x[i]).reshape(1,-1)

# X está formada. ES UNA MATRIZ DONDE CADA FILA ES UNA IMAGEN APLANADA!
print(X.shape)
```

```
1000 4096
(1000, 4096)
```

In [63]:

```
# AUTOENCODER
transformer = preprocessing.MinMaxScaler().fit(X)
X_transform = transformer.transform(X)
overfitCallback = EarlyStopping(monitor='loss', min_delta=0, patience = 100)
encoder = keras.models.Sequential([keras.layers.Dense(8, input_shape=[X.shape[1]])])      # El 8 es
el numero de variables de compresion. Esta adaptado a nuestro sistema
decoder = keras.models.Sequential([keras.layers.Dense(X.shape[1], input_shape=[8])])      # Idem con
el 8
autoencoder = keras.models.Sequential([encoder, decoder])
autoencoder.compile(loss="mse", optimizer='adam')
history = autoencoder.fit(X_transform, X_transform, epochs=10000000, callbacks=[overfitCallback], v
erbose=0)
X_reduced = encoder.predict(X_transform)
```

In [219]:

```
# GAUSSIANA
gmm = GaussianMixture(n_components=1).fit(X_reduced)
mu = gmm.means_[0].tolist()
mahalanobis = DistanceMetric.get_metric('mahalanobis', V=gmm.covariances_[0])

distanciaumbral = float('-inf')
for i in range(X_reduced.shape[0]):
     x = X_reduced[i]
     puntos = [mu, x]
     distancia = mahalanobis.pairwise(puntos)[0, 1]
     if distancia>distanciaumbral:
         distanciaumbral = distancia

distanciaumbral *= 0.5   # Admitimos un margen del 50% con respecto a la distancia umbral.
```

In [220]:

```
## AHORA VOY A PROBAR LA SEPARACION CON TODO EL CONJUNTO DE DATOS NO SOLOS LOS DE ENTRENAMIENTO

##Leemos las imagenes de los peces
folderTodos = "C:/Users/David/Desktop/OTROS/TFM/FinalFrames"


xx1 = load_images_from_folder(folderTodos)

import numpy as np
# convert list to numpy array
imagesize = len(xx1[0])*len(xx1[0][0])
print(len(xx1), imagesize)


XX = np.zeros(shape=(len(xx1), imagesize)) # Este 3 es porque la imagen está en RGB. DEBE
CAMBIARSE A ESCALA DE GRISES Y ELIMINARLO
# Cada fila debe ser una imagen. Tal y como lo hacías, aplanabas todo el "tensor". Las imagenes la
s cargas en una lista de 3 dimensiones
for i in range(len(xx1)):
    XX[i] = np.array(xx1[i]).reshape(1,-1)

# X está formada. ES UNA MATRIZ DONDE CADA FILA ES UNA IMAGEN APLANADA!
print(XX.shape)
```

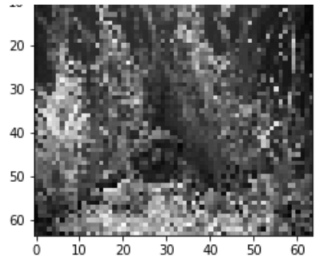
```
9656 4096
(9656, 4096)
```

In [221]:

```
##Mostramos la 1º imagen de los datos para comprobar que se ha cargado bien
plt.imshow(xx1[1], cmap='gray')
```

Out[221]:

```
<matplotlib.image.AxesImage at 0x2089e215d08>
```



In [222]:

```
## 3. Con el modelo (gm) creado tenemos que clasificar como PEZ/NO PEZ las imagenes de los demás P
ECES.
#Para ello cargamos en una matriz nueva X con esas imágenes, la codificamos con el autoencoder,
#y recorro con en el código anterior imagen por imagen
#, si la distancia es MAYOR a la umbral no es PEZ si es menor SI
Peces= []
IndexPeces = numpy.zeros(shape=(1))
XX_transform = transformer.transform(XX)
XX_reduced = encoder.predict(XX_transform)
count = 0
for i in range(XX_reduced.shape[0]):
     xx = XX_reduced[i]
     puntos = [mu, xx]
     distancia = mahalanobis.pairwise(puntos)[0, 1]
     if distancia>distanciaumbral:
         # NO PEZ
         # NO HAGO NADA
       count = count
     else:
        # SI PEZ
         # AÑADO esa imagen (XX[i]) a la matriz de imagenes PECES
        count+=1
        Peces.append(xx1[i])
```

In [223]:

```
## Imprimimos el numero de peces encontrados en el conjunto de datos
count
```

Out[223]:

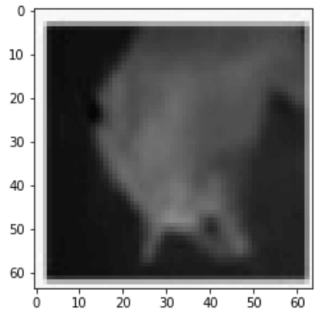
```
5798
```

In [244]:

```
## Graficamos una imagen de peces para comprobar que se han guardado bien
plt.imshow(Peces[6], cmap='gray')
```

Out[244]:

```
<matplotlib.image.AxesImage at 0x208b2c0fa08>
```

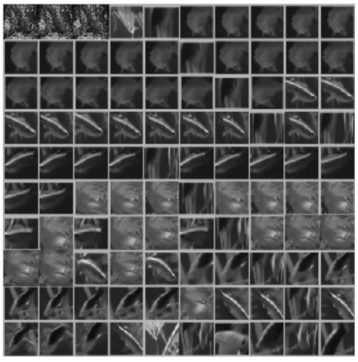


In [225]:

```
## Ahora guardamos los peces en una carpeta
count= 0
for imagen in Peces:
    cv2.imwrite("C:/Users/David/Desktop/OTROS/TFM/SalidaPeces/" + "\\frame%d.jpg" % count, Peces[co
unt])
    count+=1
```

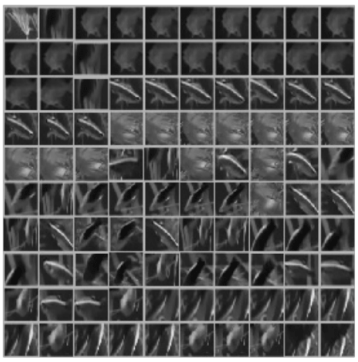
In [226]:

```
## Mostramos la imagenes originales

plt.figure(figsize=(20,5))
plot_digits(xx1[:100], images_per_row=10)
plt.show()
```



In [227]:

```
## Mostramos la imagenes separadas de peces
plt.figure(figsize=(20,5))
plot_digits(Peces[:100], images_per_row=10)
plt.show()
```



In [200]:

```
## AQUI REALIZAMOS LA CLUSTERIZACION EN ESPECIES DE LOS PECES
```

In [173]:

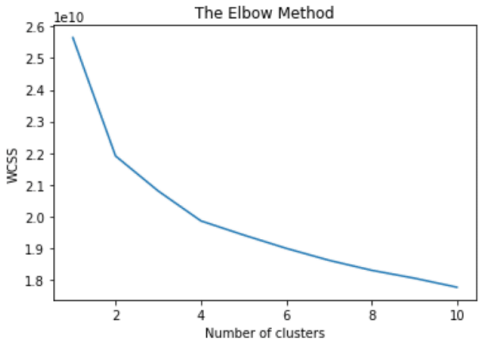
```
##Lo primero que haremos sera abrir las imagenes de SalidaPeces y transformarlos.
##Leemos las imagenes de los peces
folderTodos = "C:/Users/David/Desktop/OTROS/TFM/SalidaPeces"


xx = load_images_from_folder(folderTodos)
```

```
yy = load_images_from_folder(foldernodos)

import numpy as np
# convert list to numpy array
imagesize = len(yy[0])*len(yy[0][0])
print(len(yy), imagesize)


YY = np.zeros(shape=(len(yy), imagesize)) # Este 3 es porque la imagen está en RGB.
# Cada fila debe ser una imagen. for i in range(len(yy)):
    YY[i] = np.array(yy[i]).reshape(1,-1)

# YY está formada. ES UNA MATRIZ DONDE CADA FILA ES UNA IMAGEN APLANADA!
print(YY.shape)
```

```
5798 4096
(5798, 4096)
```

In [176]:

```
##Ahora aplicamos el metodo del codo con SalidaPeces
# Metodo del Codo para encontrar el numero optimo de clusters

from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 0).fit(YY)
    wcss.append(kmeans.inertia_)

# Grafica de la suma de las distancias
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

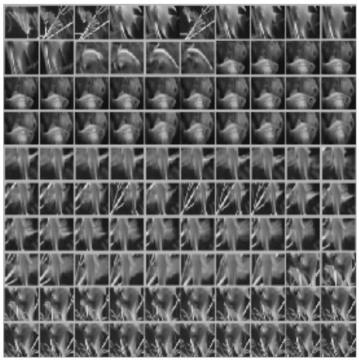


In [177]:

```
## Ahora que ya sabemos que el numero de cluster optimo es 4, aplicamos una clusterizacion utiliza
ndo GMM con 4 clusters
## Hacemos el clustering con la tecnica GaussianMixture

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=4, random_state=0).fit(YY) # EL FIT ES CON LA MATRIZ YY
labels = gmm.predict(YY)
```

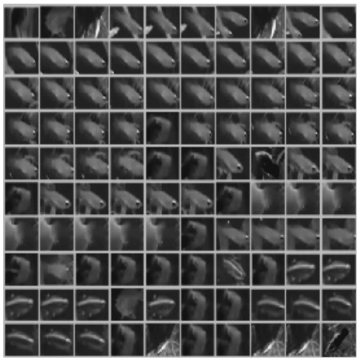
In [183]:

```
## Mostramos el 1° cluster para comprobar si la separacion ha funcionado

plt.figure(figsize=(20,5))
aux=YY[labels==0]
plot_digits(aux[:100], images_per_row=10)
plt.show()
```



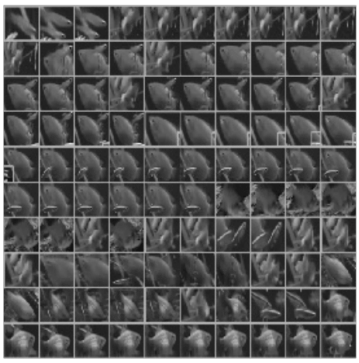
In [184]:

```
## Mostramos el 2° cluster para comprobar si la separacion ha funcionado

plt.figure(figsize=(20,5))
aux=YY[labels==1]
plot_digits(aux[:100], images_per_row=10)
plt.show()
```



In [185]:

```
## Mostramos el 3° cluster para comprobar si la separacion ha funcionado

plt.figure(figsize=(20,5))
aux=YY[labels==2]
plot_digits(aux[:100], images_per_row=10)
plt.show()
```



In [186]:

```
## Mostramos el 4° cluster para comprobar si la separacion ha funcionado

plt.figure(figsize=(20,5))
aux=YY[labels==3]
plot_digits(aux[:100], images_per_row=10)
plt.show()
```