ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Sistema de Identificación de Bacterias para Microbiología

(Bacterial Identification System for Microbiology)

Para acceder al Título de

Graduado en
Ingeniería de Tecnologías de Telecomunicación

Autor: Ana Ruiz Oviedo

Julio – 2016

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Ana Ruiz Oviedo
Director del TFG: Pablo Pedro Sánchez Espeso
Pablo González de Aledo Marugán
Título: "Sistema de Identificación de Bacterias para Microbiología"
Title: "Bacterial Identification System for Microbiology"

Presentado a examen el día: 27 de Julio de 2016

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Sánchez Espeso, Pablo
Secretario (Apellidos, Nombre): Martínez Fernández, Mª Carmen
Vocal (Apellidos, Nombre): Fernández Solórzano, Víctor

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del TFG (sólo si es distinto del Secretario)
Vº Bº del Subdirector	Trabajo Fin de Grado Nº (a asignar por Secretaría)

Agradecimientos

A mis padres, por su apoyo y confianza incondicional. Por haber estado ahí siempre que los he necesitado. Sin ellos no habría sido posible.

A Pepe, por su paciencia infinita conmigo sus buenos consejos y su ayuda en los momentos difíciles.

A mi tutor Pablo por ayudarme en este proyecto y animarme en los momentos difíciles.

A Pablo González, porque incluso desde Australia ha estado ahí hasta el último momento ayudándome.

A Álvaro y a todos los chicos del departamento de Ingeniería Microelectrónica.

Índice

Lista de Figuras

1. Introducción

1.1 Objetivos

Este proyecto se desarrolla en un entorno de colaboración con el IBBTEC (Instituto de Biomedicina y Biotecnología de Cantabria), por lo que el principal objetivo es conseguir dar solución a la especificación planteada por este grupo de investigación.

En esta línea, uno de los objetivos de este proyecto es el de proporcionar soporte a parte de las actividades que se llevan a cabo en el IBBTEC. Estas actividades están relacionadas con el recuento e identificación de bacterias en imágenes tomadas por un microscopio. Por ello, se ha propuesto como objetivo desarrollar un sistema que procese imágenes tomadas por un microscopio de contraste de fase. Además del procesado de imagen se proporcionará información acerca del número de bacterias que aparecen en la imagen. También será necesario individualizar todas las bacterias que aparecen en el campo de visión.

Un ingeniero debe ser capaz de resolver los problemas que se le planteen, intentando proporcionar una solución óptima tanto en prestaciones como en coste. Es por ello, que otro de los objetivos del proyecto es explorar diversas soluciones alternativas al problema planteado, identificando la que proporcione los mejores resultados.

En resumen, los objetivos principales de este proyecto son cumplir la especificación dada y desarrollar una herramienta que proporcione soporte a las labores de investigación en el campo de la microbiología, al tiempo que se demuestra que la autora es lo suficientemente autónoma como para proporcionar una solución que satisfaga estas necesidades de la manera más óptima posible.

1.2 Motivación

La investigación en el campo de la microbiología es un tema de gran actualidad y en constante desarrollo. La idea de realizar este proyecto surge de un problema que los científicos del IBBTEC llevan tiempo intentando solucionar.

La identificación de bacterias en una imagen microscópica es un paso previo esencial para la realización de análisis posteriores, tales como el recuento de especímenes de un determinado tipo o el cálculo de densidades de población. Todos estos procesos son actividades habituales en un laboratorio de microbiología.

Sin embargo, debido al tamaño de los especímenes y en ocasiones la calidad de las imágenes que proporcionan los microscopios, el trabajo de identificación de bacterias mediante el procesado de imagen no resulta una tarea fácil.

Poder realizar una colaboración con otros grupos, como el IBBTEC, resulta muy interesante. Más todavía si se trata de un tema, como es la investigación en el campo de la microbiología, que está constantemente avanzando y donde cualquier herramienta que pueda facilitar el trabajo supone una gran ayuda. Poder contribuir, desde el campo de la ingeniería de telecomunicación, a mejorar y agilizar las investigaciones en los laboratorios de microbiología así como proporcionarles soporte y nuevas herramientas que faciliten su trabajo, supone una de las principales motivaciones para llevar a cabo este proyecto.

Otra motivación, es la de ampliar conocimientos en el campo del procesado de señal. Más concretamente, ampliar y adquirir nuevos conocimientos relacionados con el procesado de imágenes, especialmente en el caso de imágenes biológicas.

En definitiva, las motivaciones que han impulsado este proyecto han sido, por un lado, contribuir activamente en la colaboración con otros grupos de investigación. Por otra parte, ayudar en la investigación en un campo, como es el de la investigación y la microbiología, que se encuentra constantemente avanzando y cuyas investigaciones derivan en grandes avances en terrenos de vital importancia como puede ser la medicina. Y por último, el poder aprender y ampliar los conocimientos adquiridos sobre el procesado de imagen y el procesado de señal, explorando técnicas de segmentación, clasificación y reconocimiento de objetos en imágenes, entre otras.

2. Especificación

A continuación, en este capítulo se detallará la especificación de la que se disponía y que este proyecto debe satisfacer.

En primer lugar, se necesita crear un software de procesado de imagen que sea capaz de corregir las imágenes tomadas por un microscopio de contraste de fase. El programa será capaz de corregir los defectos de las imágenes tales como falta de luminosidad y de contraste.

En segundo lugar, se asumirá que el tamaño de bacterias que el software tendrá que identificar estará acotado. En consecuencia, el programa no tendrá que determinar el factor de escala óptimo de la imagen. Siempre y cuando las bacterias que se quieran detectar se encuentren en el rango de tamaños de funcionamiento del software, este será capaz de detectarlas, sin necesidad de re-escala o ampliar/reducir la imagen.

En tercer lugar, el programa que se desarrolle debe ser capaz de adaptarse a distintos tipos de bacterias, aprendiendo de forma autónoma como identificarlas. Esto significa que se necesitará un algoritmo de entrenamiento que sea capaz de crear un clasificador que consiga detectar bacterias.

Por último, se necesita que el software realice un trabajo de identificación y recuento de bacterias. Además, se pretende implementar posteriormente este programa en un sistema embebido, de modo que se consiga diseñar un sistema portable. Se valorará positivamente la velocidad del software y el uso de recursos a la hora de poder implementarlo en el sistema embebido.

En definitiva, lo que se especifica es un programa capaz de realizar todo el procesado de las imágenes tomadas por un microscopio de contraste de fase, así como el entrenamiento de un algoritmo que sea capaz de generar un clasificador que se utilizará en la identificación y el recuento de las bacterias. Además, este programa debe ser implementable en un sistema embebido, de modo que se consiga un software portable. Es decir, que el propio sistema se encargue de recibir, procesar, identificar y contar las bacterias.

3. Estado del Arte

Como se ha comentado anteriormente, el objetivo principal de este proyecto es diseñar un sistema que permita identificar y contar el número de bacterias que hay en una imagen tomada por un microscopio. A continuación, se explicarán las diferentes técnicas de captura de imágenes así como de procesado de las mismas.

3.1 Captura de Imágenes

3.1.1 Microscopio

La captura de imágenes de las bacterias se realiza a través de un microscopio. Uno de los microscopios más utilizados para la captura de imágenes de bacterias es el microscopio de contraste de fase. Este microscopio basa su funcionamiento en la emisión de dos haces de luz sobre la muestra a capturar, y en la posterior medida del desfase entre los dos haces, como se puede ver en la Figura 1.

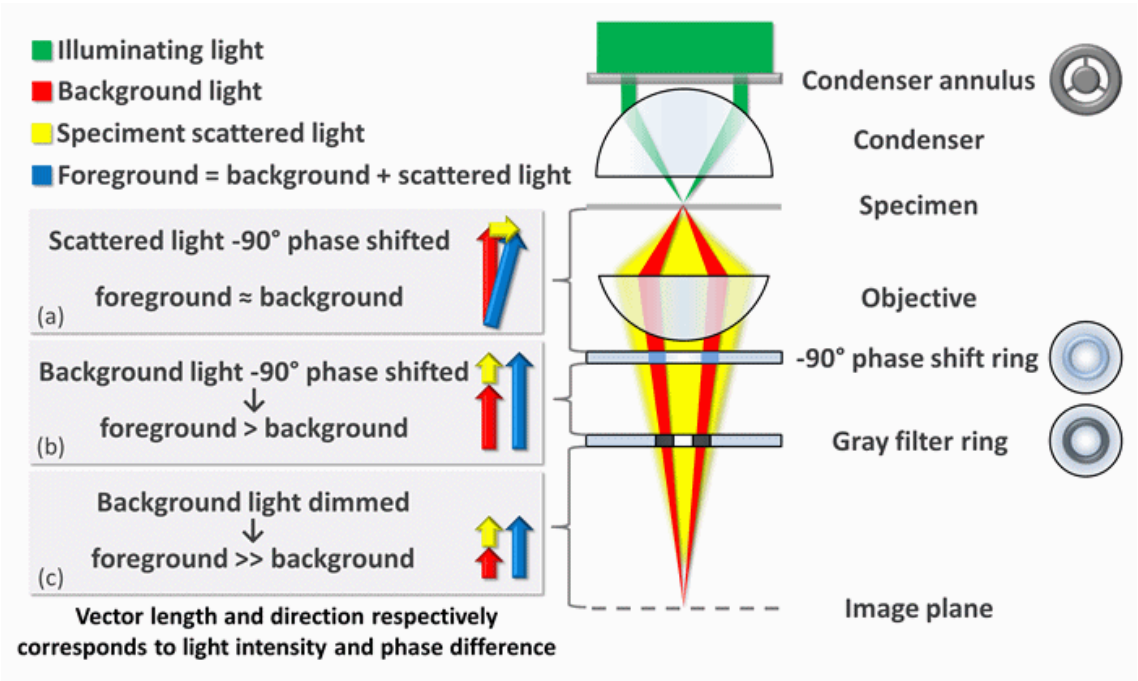


Figura 1: Funcionamiento del Microscopio de Contraste de Fase [1]

El microscopio de contraste de fase se utiliza para conseguir una imagen más nítida de las bacterias. Esto es necesario las mismas tienen un índice de refracción muy similar al del agua en el que viven, por lo que cuando se capta una imagen con un microscopio normal es difícil distinguir la bacteria del fondo. Sin embargo, con el uso del microscopio de contraste de fase, se consigue una mejor definición de los bordes de las bacterias.

Los bordes de las bacterias están formados por una membrana. Esta membrana, tiene un índice de refracción mayor que el del resto de la célula, por lo que al incidir luz sobre la membrana se reduce su velocidad, provocando un desfase entre la luz que ha pasado a través de la membrana y la que lo ha hecho a través del medio acuoso que la rodea. Utilizando este desfase, se genera un mejor contraste en la imagen de salida, quedando los bordes de las bacterias más marcados. Esto resulta muy útil, para contar las bacterias mediante la detección de sus bordes, ya que existe una diferencia notable entre las imágenes tomadas con un microscopio tradicional frente a las imágenes que se obtienen con un microscopio de contraste de fase, como se muestra en la Figura 2.

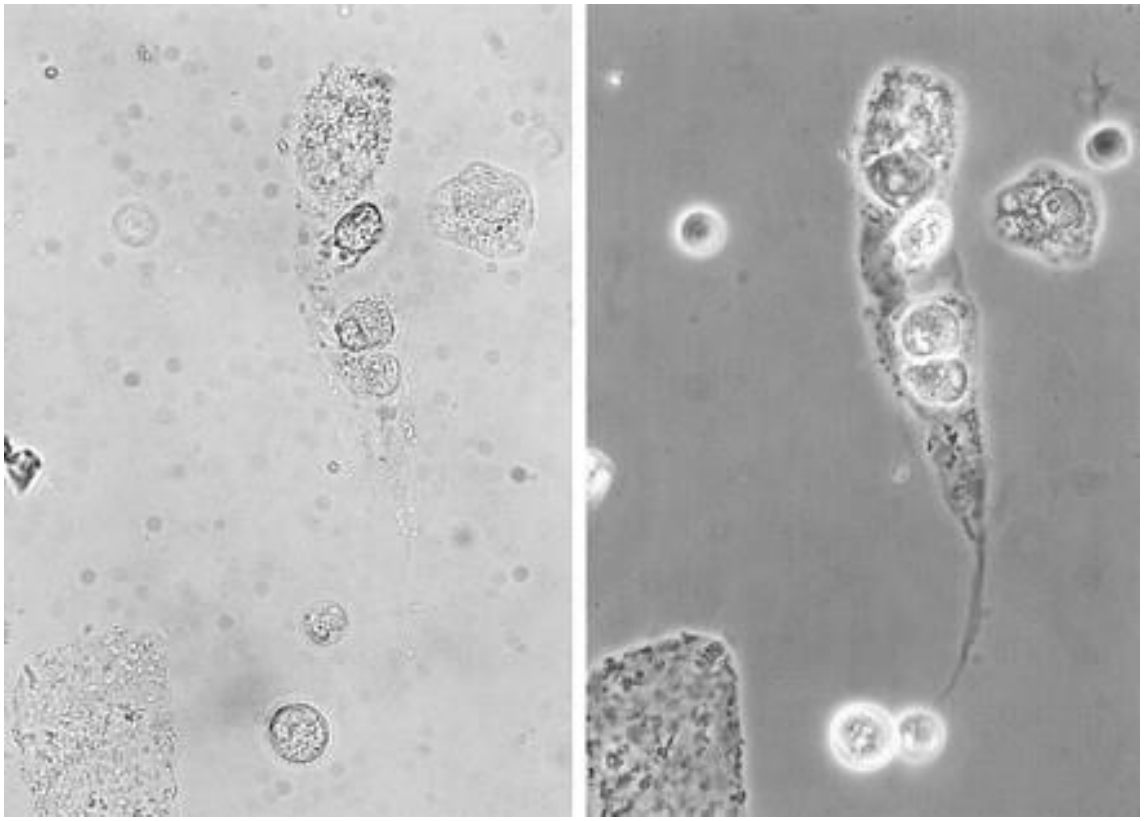


Figura 2: Microscopio Tradicional (izquierda) Microscopio de Contraste de Fase (derecha) [2]

3.1.2 Cámara

Una vez se han generado las imágenes del microscopio, es necesario capturar dichas imágenes con una cámara. Esta cámara se acopla al microscopio y permite obtener tanto fotografías como vídeo de lo que en ese momento se esté observando. Existen varios tipos de cámaras para realizar la captura de las imágenes en un microscopio. Por un lado están las cámaras digitales de alta resolución, que son capaces de tomar imágenes en el espectro visible, por lo que capturan la misma información que el investigador observa directamente en el microscopio. Por otro lado, hay cámaras más específicas como pueden ser las cámaras multi-espectrales. Estas cámaras permiten capturar imágenes a distintas longitudes de onda, lo que proporciona como resultado, varias imágenes de

salida. Como se observa en la Figura 3, en función de la longitud de onda a la que se capture cada imagen, se pueden apreciar diferentes propiedades de las bacterias.

Generalmente, las células o bacterias vivas no pueden ser teñidas para obtener mejores imágenes ya que los tintes utilizados suelen matar dichos organismos vivos. Es por ello, que la opción más utilizada suele ser la de marcar las bacterias con un gen fluorescente. Este gen hace que si se observa la bacteria a determinadas longitudes de onda, esta resalte sobre el fondo, obteniéndose así imágenes con una mejor definición. La obtención de las imágenes a la longitud de onda adecuada es posible gracias a las cámaras multi-espectrales.

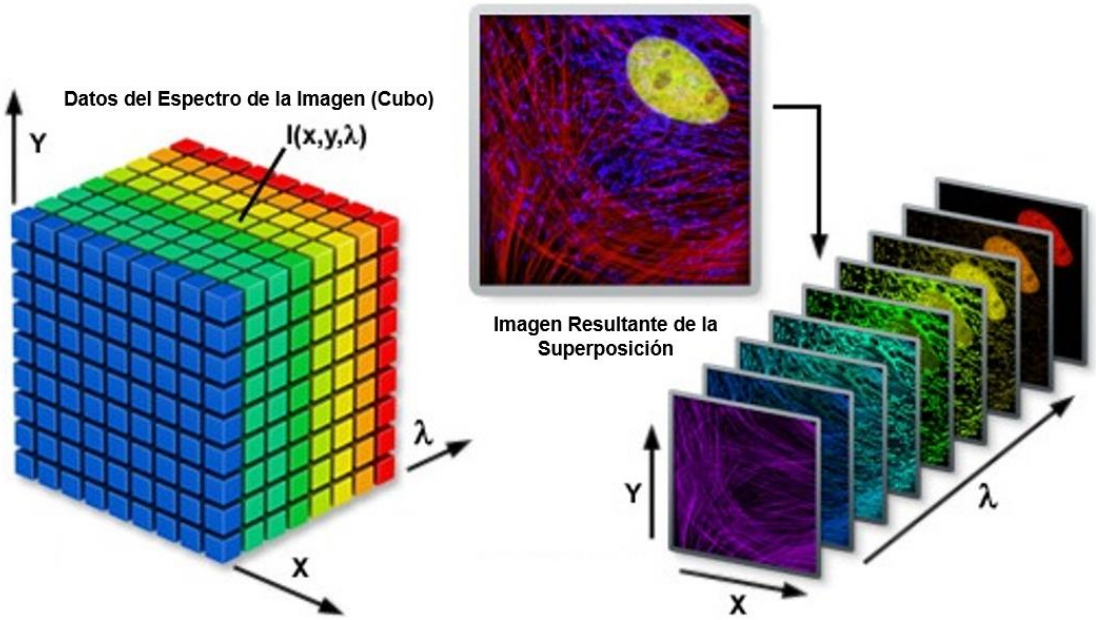


Figura 3: Esquemático de una Imagen Multi-espectral

3.2 Técnicas Básicas

3.2.1 Corrección del Histograma

Para trabajar con imágenes es importante conocer su histograma o distribución de valores entre los píxeles que la forman. En el eje X se representa la escala tonal de la imagen que va desde el negro, a la izquierda, hasta el blanco, a la derecha, en el caso de imágenes en escala de grises. En el eje Y, se representan el número de píxeles que hay en cada región de la escala tonal.

Cuando se utilizan imágenes tomadas con un microscopio, generalmente el histograma de dichas imágenes está desajustado. Esto provoca que, al visualizar las imágenes,

parezca que no contienen información o que dicha información es insuficiente. Es por ello, que existen técnicas para realizar un ajuste del histograma. Entre las más utilizadas se encuentran el centrado del histograma y la ecualización del histograma. A continuación, se detallarán ambas técnicas.

En la mayoría de los casos, el histograma de las imágenes se asemeja a una función de distribución normal o gaussiana. Esto es debido a que gran parte de las imágenes que se manejan tienen una iluminación uniforme. Más aún si se trata de imágenes de microscopía en donde los microscopios con los que se capturan las imágenes suelen tener una fuente potente de luz que ilumina uniformemente la superficie a observar.

Esta cualidad del histograma hace que el centrado del histograma de la imagen sea equivalente a hacer la media de la gaussiana cero. Realmente, lo que se busca al centrar el histograma es situar cercanos al centro del eje X los valores con mayor densidad de pixeles. Con este centrado se consigue que imágenes muy oscuras o muy claras se sitúen en un rango de tonos más grises.

Por otra parte, el otro ajuste del histograma más utilizado es la ecualización. Consiste en forzar que la varianza de la gaussiana sea uno, con lo que se consigue distribuir todos los valores de la función de distribución a lo largo del eje X. Con ello se consigue obtener un mayor contraste de la imagen ya que las diferencias entre los valores de los pixeles de la imagen son mayores.

3.2.2 Filtrado

En muchas ocasiones, las imágenes con las que se está trabajando contienen una gran cantidad de ruido. Este ruido provoca que la imagen no esté tan nítida como debería o que al realizar cualquier tipo de procesado en ella, se confunda la información real con el ruido.

Generalmente, el ruido que tienen las imágenes suele ser ruido blanco por lo que se puede eliminar o reducir utilizando técnicas de filtrado. Estas técnicas de filtrado permiten eliminar el ruido de la imagen sin distorsionar apreciablemente la información relevante de la misma.

Es importante eliminar el ruido de una imagen antes de realizar cualquier otra transformación (como aumento del contraste) ya que la misma puede modificar la distribución del ruido y cambiar el efecto del filtro, pudiendo provocar que el ruido esté aún más presente en la imagen que antes de haber realizado la transformación.

Los filtros más adecuados para reducir el ruido de la imagen son los filtros de tipo paso bajo. Dichos filtros, como su propio nombre indica, permiten pasar las frecuencias bajas

y atenúan las altas frecuencias. Son adecuados para el filtrado de ruido ya que las altas frecuencias se corresponden con los cambios más bruscos de intensidad en la imagen y al ser filtradas, estos cambios quedan suavizados. Por tanto, cuando se realiza un filtrado paso bajo de una imagen, el resultado es una imagen más suavizada. Dentro de los filtros paso bajo, los dos principales filtros utilizados para eliminar el ruido son el filtro de valor medio y el filtro gaussiano.

El filtro de valor medio suma el valor de un pixel al de todos los píxeles adyacentes y divide el resultado entre nueve, que es el número de píxeles que se han tomado para obtener el valor medio. Es decir, se le aplica al pixel a procesar una máscara que determina el peso con el que cada pixel se suma al valor resultante y se divide dicho valor entre nueve, como se observa en la Figura 4. Las imágenes que se obtienen al aplicar este filtro son imágenes suavizadas y por tanto menos nítidas. Por tanto, aunque elimina parte del ruido también difumina la imagen.

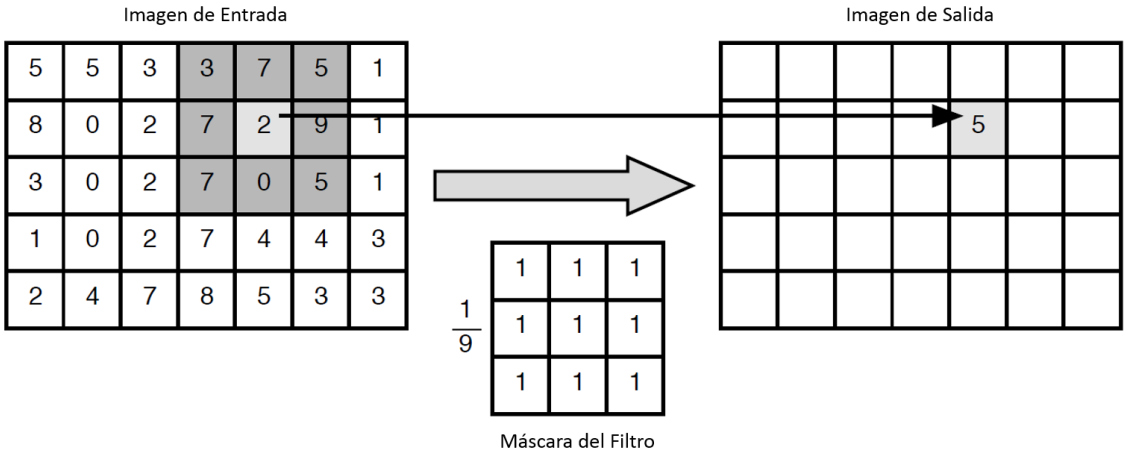


Figura 4: Filtro de Valor Medio [4] ***¡Error! No se encuentra el origen de la referencia.***

El filtro gaussiano se aplica igual que el filtro de valor medio, es decir, se le aplica una máscara al pixel a procesar, como se observa en la Figura 5. Esta máscara se calcula a partir de la función de Gauss:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Para obtener los valores de la máscara, se utiliza σ = 1 como valor de la varianza. Además, al igual que en el filtro de valor medio, una vez aplicada la máscara, se normaliza. En este caso, se dividirá el valor del pixel entre 16 debido a los coeficientes de la máscara mostrados en la Figura 5.

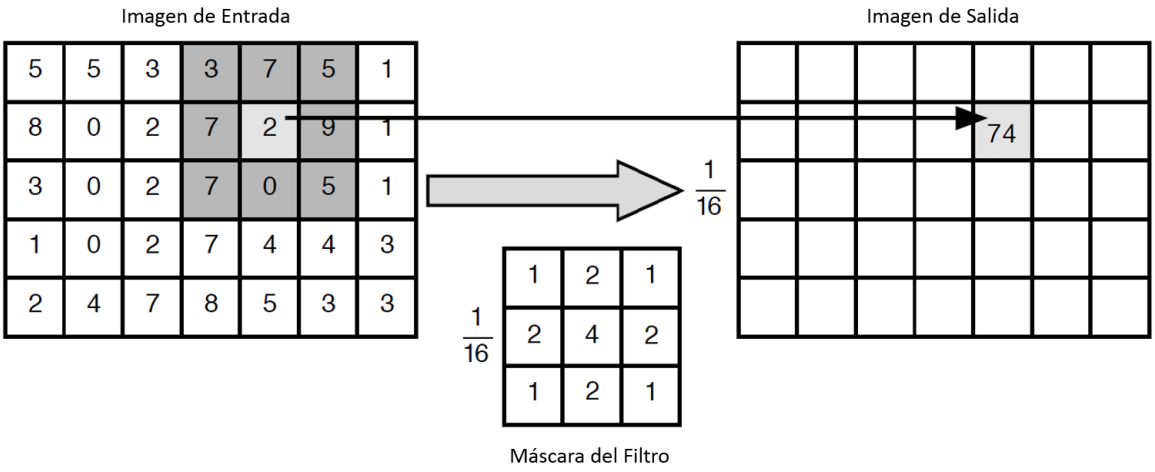


Figura 5: Filtro Gaussiano

En este ejemplo, se ha utilizado una matriz de 3x3 como máscara de filtrado pero se pueden utilizar matrices de mayor tamaño para obtener mejores resultados. Es importante resaltar que si se varía el valor de la varianza, σ, se deberá variar el tamaño de la máscara de modo que cuanto mayor sea la varianza mayor deberá ser el tamaño de la matriz utilizada para el filtrado.

La diferencia entre ambos filtros es que el filtro de media difumina y suaviza más la imagen que el filtro gaussiano. Este último es capaz de eliminar el ruido sin difuminar tanto la imagen por lo que normalmente se suelen utilizar filtros gaussianos.

3.2.3 Detección de Bordes

Existen otros tipos de filtros, como los de detección de bordes. Dentro de estos filtros se pueden distinguir dos grupos: los que no dependen de la dirección (como los filtros de Laplace) y los que dependen de la dirección (como el filtro Sobel). Ambos filtros basan su funcionamiento en encontrar las discontinuidades, es decir, en determinar donde se producen cambios bruscos en el brillo de una imagen en escala de grises.

La principal diferencia que existe entre los filtros de borde que dependen de la dirección y los que no es que los primeros solo necesitan utilizar una máscara para encontrar los bordes mientras que los segundos (como el Sobel) utilizan dos máscaras: una para encontrar los bordes con respecto al eje X y otra para encontrar los bordes con respecto al eje Y. Los valores obtenidos con las dos máscaras son combinados para proporcionar el resultado del filtrado.

3.3 Segmentación

Una vez se han presentado las técnicas básicas de procesado de imagen más utilizadas, a continuación se introducirá la segmentación. Dicha técnica consiste en extraer, de una imagen las partes o áreas de interés. Aunque en principio puede parecer algo sencillo de realizar, para una herramienta de procesado de imagen esto no resulta nada fácil.

Diferenciar un objeto, una forma o un color dentro de una imagen es sencillo para una persona ya que, debido a hábitos adquiridos desde la infancia, le resulta fácil identificar lo que se está buscando. Al buscar algo concreto dentro de una imagen, los humanos somos capaces de reconocer el entorno de la imagen y distinguir con facilidad qué es lo que se está buscando y qué elementos pertenece al entorno y al fondo.

Sin embargo, para un sistema de procesado, cada imagen solo es un conjunto de pixeles que contienen información, no diferenciando los objetos y elementos de la misma. Aun así, existen diversas técnicas de segmentación que permiten resolver este problema en función de las condiciones de la imagen y de lo que se necesite extraer de la misma.

3.3.1 Segmentación por Umbralización

Una de las técnicas más utilizadas en la segmentación de imágenes es la umbralización. Para segmentar utilizando esta técnica se necesita partir de una imagen en escala de grises. Partiendo de esta imagen se calcula un umbral que puede obtenerse por medio de algoritmos o fijarse manualmente. El algoritmo de segmentación por umbral es sencillo; lee cada pixel de la imagen y lo compara con el umbral: si el valor de dicho pixel es mayor que el umbral, al mismo se le asigna un valor elevado (generalmente 255, el valor blanco cuando se trata de imágenes de 8 bits), y si el valor del pixel es menor que el umbral este se fija a un valor 0 (negro). Como resultado se obtiene una imagen binaria donde todo el fondo estará en negro y la parte de la imagen relevante estará en blanco, como en el ejemplo de la Figura 6.

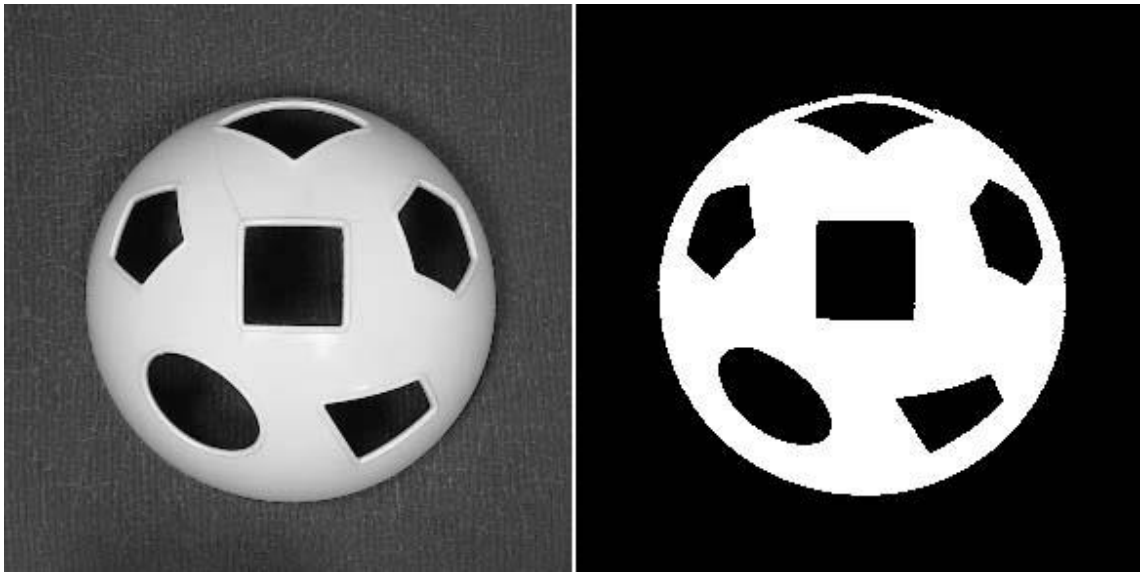


Figura 6: Ejemplo de Umbralización

Se puede realizar también la umbralización de forma inversa, es decir, fijar a blanco todos los píxeles que se encuentren por debajo del umbral y a negro los que se encuentren por encima de él, como en la Figura 7. A este tipo de umbralización se le denomina umbralización inversa.

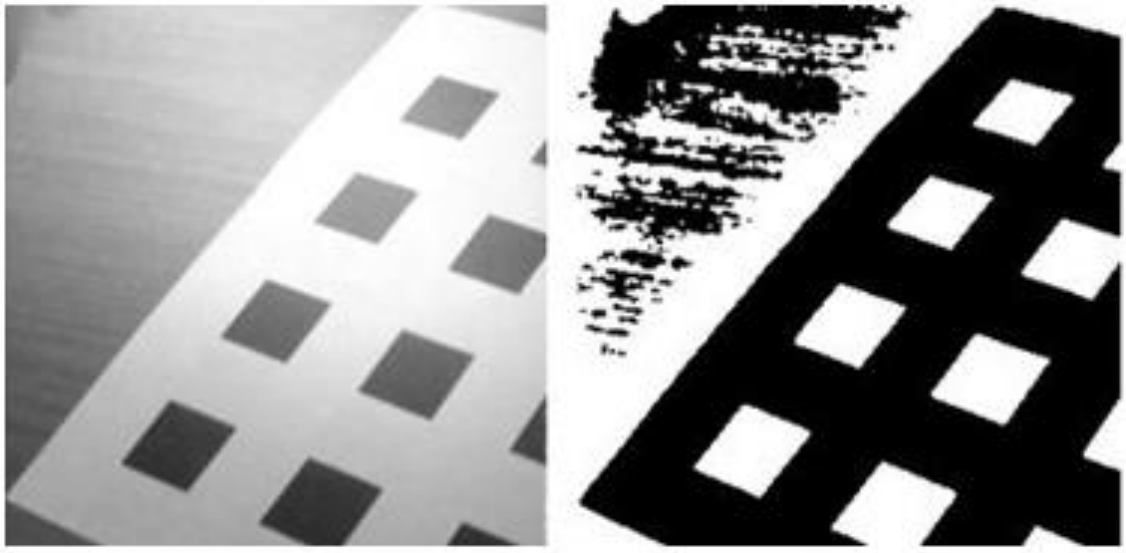


Figura 7: Ejemplo de Umbralización Inversa

Como se observa en la Figura 7, al realizar la umbralización, parte del fondo ha quedado blanca y otra parte negra cuando lo ideal habría sido que todo el fondo estuviese blanco. Esto se debe a que el umbral escogido no es el más adecuado ya que no es capaz de segmentar totalmente la imagen. Este es uno de los problemas que presenta la técnica de segmentación por umbralización, la elección del valor umbral adecuado.

Además, existen casos en los que los tonos que contiene el fondo son los mismos que los de partes de la imagen a segmentar, provocando ello que la segmentación por umbralización no sea de gran utilidad en estas situaciones.

3.3.2 Segmentación por Color

Cuando se tienen imágenes en color, se puede recurrir a la segmentación por color en lugar de la segmentación por umbralización. El objetivo de ambas segmentaciones es el mismo (conseguir aislar un objeto, área o sección de la imagen del fondo) y su estrategia es similar, pero en lugar de comparar con un umbral se comparan los píxeles con el valor de un cierto color.

Al utilizar segmentación por color, es importante saber qué tipo de imagen se tiene. Es decir, no es lo mismo trabajar con una imagen en RGB que con una en HSV. La gran diferencia entre estos dos tipos de imágenes reside en que en el HSV la luminosidad y la saturación están separadas del valor del color mientras que en las imágenes RGB estos parámetros no se pueden separar. Esta característica hace que a la hora de realizar una segmentación por color sea mucho más fácil trabajar con imágenes en formato HSV o YUV.

Para segmentar una imagen por color, lo que se hace es fijar unos valores máximos y mínimos para cada componente, es decir, para H, S y V. A continuación, se comprueba si el píxel a segmentar se encuentra dentro del rango de valores elegido. Si es así, se fijará a blanco y si no es así, se fijará a negro.

Existen otras formas de realizar la segmentación por color, como la basada en la función de distribución gaussiana de los colores o la basada en su histograma pero resultan más complejas por lo que se utilizan menos.

3.3.3 Región de Crecimiento

La técnica de segmentación mediante región de crecimiento consiste en unir áreas con un tono de gris similar o con un color parecido. Para ello, se coge un pixel y se comprueba si los adyacentes tienen el mismo color. Si esto se cumple, se añade el pixel adyacente a una lista de pixeles que será procesada posteriormente. Además, se crea otra lista de pixeles con todos los pixeles comprobados para evitar comprobar varias veces un mismo pixel. Con este algoritmo, se consiguen conectar las regiones con un color similar y así segmentarlas.

3.3.4 Segmentación por Formas Geométricas

Este tipo de segmentación consiste en identificar formas geométricas utilizando la transformada de Hough. La principal condición que deben cumplir estas formas es que puedan ser expresadas mediante ecuaciones matemáticas. Dichas ecuaciones incluyen parámetros que caracterizan a la forma geométrica. Por ejemplo, en el caso de una recta en 2 dimensiones (x e y), la figura geométrica puede ser caracterizada por la ecuación de la recta en forma polar "$\rho = x.\cos(\theta) + y.sen(\theta)$", en donde los parámetros "ρ" (distancia al origen del coordenadas) y "θ" (ángulo) definen la recta. La transformada de Hough convierte el plano de la imagen, con coordenadas x e y, en un plano de parámetros, con coordenadas "ρ" y "θ". En la imagen del plano de parámetros, cada pixel mostrará la existencia de rectas con los parámetros asociados a dichas coordenadas. De esta forma, si en la imagen existen 2 rectas, la transformada de Hough mostrará 2 puntos, que se corresponderán con los parámetros "ρ" y "θ" de cada una de las rectas. Además de la transformada que detecta rectas, también se suele utilizar la transformada para identificar arcos de circunferencia, caracterizándose en este caso la figura geométrica mediante 3 parámetros: el radio del circulo y las coordenadas (x,y) de su origen.

La transformada de Hough se aplica a una imagen binaria, que normalmente se obtiene después de haber aplicado un filtro de bordes a la imagen. De esta forma, la transformada identifica la forma del objeto. La transformada también permite identificar los puntos de la imagen que verifican ciertas propiedades, como por ejemplo ser punto de cruce de varias rectas.

3.4 Clasificación

Los clasificadores son algoritmos capaces de identificar dentro de una imagen ciertos elementos a partir de algunas de sus características. La principal diferencia entre la segmentación y la clasificación, es que mientras que la segmentación es una técnica de procesado de imagen, la clasificación utiliza técnicas de aprendizaje máquina para llevar a cabo la tarea. Además, en la segmentación se utilizan características de la imagen tales como luminosidad o color, mientras que en la clasificación se busca identificar elementos con unas características comunes, sin tener tan en cuenta los valores absolutos de los píxeles de la imagen.

Desde el punto de vista matemático, un clasificador es una técnica que parte de un conjunto de datos marcados o datos de entrenamiento. Cada elemento del conjunto de datos de entrenamiento (por ejemplo, imágenes) posee una marca que le asocia una propiedad (por ejemplo, la marca es 1 si la imagen es una bacteria y 0 si no lo es). El

clasificador analiza el conjunto de elementos marcados y define una serie de parámetros que permiten particionar el conjunto de datos de entrenamiento conforme a la marca. Para ello, el clasificador normalmente utiliza técnicas estadísticas. Una vez ha determinado los parámetros que permiten dividir los elementos del conjunto de entrenamiento en sub-conjuntos con la misma marca (proceso de entrenamiento), el clasificado es capaz de predecir a que conjunto será asociada una nueva imagen (proceso de clasificación).

Existen varios tipos de clasificadores, cada uno de ellos basados en diferentes técnicas de aprendizaje y entrenamiento. A continuación, se presentarán algunos clasificadores y métodos para el entrenamiento de los mismos.

3.4.1 AdaBoost

El AdaBoost (Adaptative Boosting) es un algoritmo de aprendizaje de máquinas. Este algoritmo pretende entrenar iterativamente clasificadores débiles, de modo que cada nuevo clasificador se centre en los datos erróneamente identificados por el clasificador anterior. De este modo, se consigue obtener un clasificador robusto, basado en clasificadores débiles, con poca capacidad de discriminación de objetos.

Se dice que un clasificador es débil, cuando su capacidad de identificar una propiedad es ligeramente superior a la que tendría un clasificador aleatorio. Por ejemplo, un clasificador que reconozca bacterias con una probabilidad del 55% sería un clasificador débil mientras que si lo hiciese con el 50% sería aleatorio. Un clasificador robusto es aquel que proporciona tasas de acierto en la clasificación elevadas (por ejemplo, identifica correctamente una bacteria en el 90% de los casos).

La forma que tiene AdaBoost de combinar los clasificadores débiles es asignándoles un peso a la salida, en función del acierto que proporcionen. El algoritmo selecciona en primero lugar el clasificador con mayor tasa de acierto, y a continuación, el que tiene mayor acierto en el subconjunto de casos en el cual falla el clasificador anterior. De esta forma, se consigue obtener un clasificador final robusto como una combinación lineal de todos los clasificadores débiles seleccionados durante el proceso.

La gran ventaja que presenta AdaBoost frente a otros clasificadores es la reducción del tiempo y del volumen de datos necesario. De ahí, el nombre de Boosting o aceleración del entrenamiento utilizando una combinación lineal de clasificadores débiles.

3.4.2 Viola-Jones

Viola-Jones es un algoritmo diseñado originalmente para la detección de caras. Fue de los primeros en obtener buenos resultados en dicho campo. El algoritmo utiliza una serie de clasificadores en cascada que realizan operaciones muy simples en la imagen (sumas y restas de píxeles adyacentes). Utilizando un algoritmo de entrenamiento de tipo AdaBoost se determinan los clasificadores a utilizar, su orden y su peso. Pese a que originalmente se utilizó para la detección de caras, puede entrenarse para detectar cualquier otro tipo de objetos como por ejemplo coches en una carretera.

La ventaja que presenta Viola-Jones frente a otros clasificadores es el ahorro de tiempo ya que no analiza minuciosamente toda la imagen sino que se centra en analizar las zonas donde hay una gran probabilidad de encontrar el objeto a clasificar.

El algoritmo presenta una gran robustez, además de un coste computacional significativamente menor que sus predecesores. Es por ello, que fue el primer algoritmo de detección facial en tiempo real.

3.4.3 SVM

SVM o Support Vector Machines es un conjunto de modelos y algoritmos de entrenamiento capaces de generar un clasificador partiendo de un conjunto de datos dado. A partir de un conjunto de datos de entrenamiento, en donde cada elemento es clasificado en un grupo, SVM crea un modelo capaz de asignar un elemento nuevo a la categoría que le corresponde. Para ello, SVM realiza transformaciones de los parámetros del conjunto de datos y determina condiciones de separación. En la Figura 8 se observa cómo a partir de un conjunto de datos de entrenamiento, o “Espacio Inicial”, en el cual los elementos fueron clasificados como “azules” y “rojos”, SVM ha realizado una transformación introduciendo nuevas dimensiones (espacio “Transformación”) que permite dividir el conjunto de datos en dos subconjuntos: elementos rojos y elementos azules.

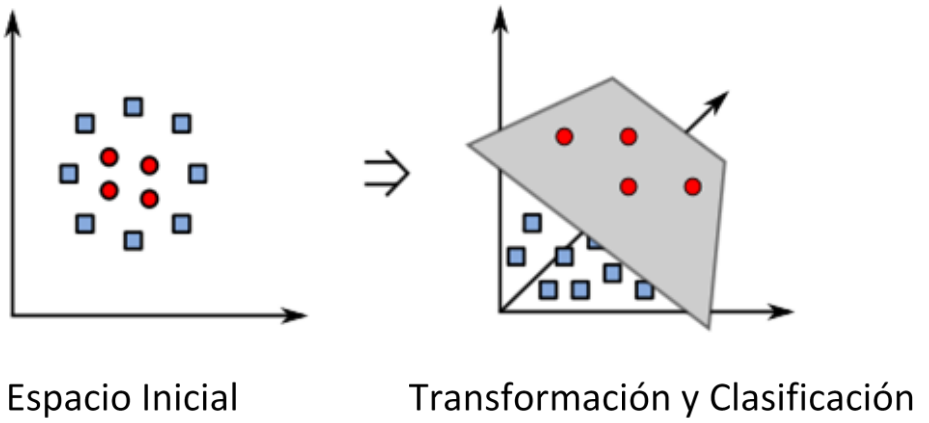


Figura 8: Clasificación de Elementos

SVM pertenece al grupo de los clasificadores lineales (la condición de separación es una ecuación lineal, como un plano) y no responde a un patrón de probabilidad sino que sigue un modelo generado a partir de datos de entrenamiento. Para separa las imágenes, SVM genera un conjunto de híper-planos (planos con dimensión mayor de 3) que se aplican en un espacio de datos transformado, con nuevas dimensiones generadas a partir de las iniciales

3.4.4 Simulated Annealing

El algoritmo Simulated Annealing (recocido simulado) es un algoritmo estocástico de búsqueda local. Está basado en el proceso de recocido y templado de los metales. En este proceso, si el templado del metal se realiza a una velocidad adecuada se consigue un estado final de mínima energía. Por tanto, lo que se busca obtener con el algoritmo Simulated Annealing es el estado de mínima energía.

El algoritmo consta de dos bucles, un bucle interno donde se buscan nuevos estados que minimicen la energía y un bucle externo, donde se desciende la temperatura. La búsqueda de nuevos estados es un proceso que depende de la temperatura, del estado actual en el que se encuentre el algoritmo y de la probabilidad de cambiar de estado aun cuando el nuevo estado tenga mayor energía que el anterior. Así, si el proceso se enfría adecuadamente, se consigue obtener una configuración de la posición de los objetos a buscar que minimice la energía de la función.

Este algoritmo puede utilizarse para buscar la solución óptima de una asignación. Las técnicas de clasificación de imágenes pueden modelarse como problemas de optimización y ser resueltos aplicando técnicas de Simulated Annealing.

El principal problema que presenta este algoritmo es la dificultad de optimizarlo ya que depende de múltiples variables. Esto hace que en ocasiones se requiera mucho tiempo de enfriamiento, convirtiendo esta técnica en un método bastante lento para la búsqueda de mínimos en una función.

3.5 Entornos Específicos de Procesado de Imagen

Además de las técnicas presentadas anteriormente, existen otro tipo de herramientas para llevar a cabo el procesado de imagen y la detección de bacterias. Algunas de ellas, se basan alguno de los algoritmos descritos anteriormente mientras que otras utilizan técnicas propias.

3.5.1 OpenCV

Open Source Computer Vision, más conocido como OpenCV es una librería con funciones de programación destinadas principalmente a aplicaciones de tiempo real y de procesado de imagen. Su uso está muy extendido por integrar multitud de funciones y proporcionar buenas prestaciones en tiempo real, al estar la implementación de la librería paralelizada con OpenMP, OpenCL y códigos específicos de la plataforma hardware (por ejemplo, código con extensiones SIMD).

El uso de OpenCV en el procesado de imágenes es de gran interés ya que esta librería contiene funciones que realizan parte de los algoritmos anteriormente comentados como la segmentación, el filtrado o la corrección del histograma sin necesidad de que el usuario conozca el algoritmo ni necesite desarrollar el código. Simplemente utilizando las funciones que proporciona OpenCV, el usuario es capaz de realizar un filtrado o una segmentación con rapidez sin necesidad de preocuparse por la optimización del algoritmo. OpenCV también incluye funciones para realizar la clasificación utilizando AdaBoost o Viola-Jones.

Una desventaja de utilizar OpenCV es el grado de control de los parámetros a aplicar. Esto quiere decir que, cuando se utiliza una función de librería, los parámetros que se pueden ajustar son los que la función permita modificar. Si el usuario desarrolla

íntegramente el algoritmo, el control sobre dichos parámetros es total, aunque el tiempo de desarrollo es mayor.

Sin embargo, las ventajas son mayores que las desventajas y es por ello que resulta de gran ayuda y utilidad su uso en aplicaciones que requieran el uso de imágenes. Además de todas las funciones para el procesado de las imágenes, incluye funciones que facilitan la lectura, escritura y manipulación de dichas imágenes.

3.5.2 Matlab

MAtrix LABoratory, es un lenguaje y un entorno de programación que cuenta con una librería específica para el procesado de imagen. Además, cuenta con un entorno gráfico donde visualizar datos y resultados.

La ventaja que ofrece Matlab frente a OpenCV es que además de las funciones de procesado de imagen también proporciona un entorno de programación, aunque hay que aprender su lenguaje específico. Pese a la gran capacidad de cómputo que ofrece Matlab, la librería de procesado de imagen resulta mucho menos extensa que la de OpenCV por lo que su uso en estas aplicaciones es menor.

Una desventaja del uso de Matlab es que se trata de un lenguaje de más alto nivel por lo que puede provocar que las funciones que se realicen no sean tan buenas como las que puedan desarrollarse en otros lenguajes de más bajo nivel.

Aun así, en Matlab se han desarrollado herramientas para la detección de bacterias como puede ser MicrobeTracker [3].

4. Diseño de la Solución

En este capítulo se presentará la solución desarrollada para verificar la especificación pedida. Se va a detallar tanto el diseño final, como los pasos seguidos hasta llegar a él. En la Figura 9 se puede ver un pequeño diagrama de flujo de la metodología seguida para el diseño de la solución.

Se parte de las imágenes capturadas por un microscopio de contraste de fase. Dichas imágenes incluyen tres capas o planos: fase, rojo y verde. Cada plano es una imagen monocromática (con solo un canal) y típicamente tiene 8 bits de profundidad de píxel. En este proyecto se ha utilizado únicamente la capa de fase.

En primer lugar, la imagen es acondicionada para su posterior clasificación (bloque procesado de las imágenes). A continuación, la imagen se segmenta utilizando un clasificador (bloque segmentación de las imágenes). La segmentación produce una imagen en donde las bacterias se diferencian del fondo, pero no se individualizan. La identificación de cada una de las bacterias es realizada por el bloque "Identificación y Recuento de las Bacterias". Por último, se generan los resultados del proceso.

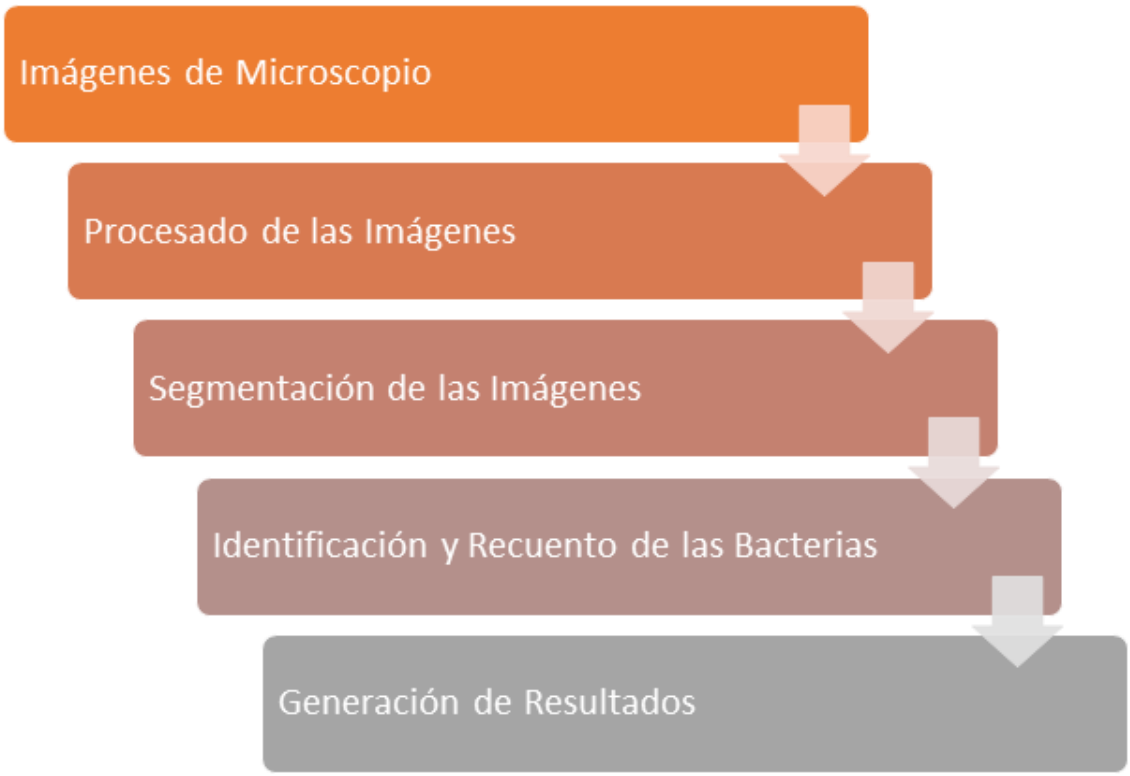


Figura 9: Esquema del Diseño de la Solución

4.1 Posibles Soluciones

Para resolver el problema planteado en la especificación de este proyecto, se han planteado varias soluciones que se expondrán más adelante. Todas estas soluciones incluyen una parte común, que consiste en el procesado previo de las imágenes que se explicará a continuación.

Para realizar este proyecto, se dispone de unas imágenes de bacterias en donde a priori no parece haber información ya que son prácticamente negras. Por ello, la primera transformación que se aplica es una corrección del histograma de la imagen, de forma que esté centrado.

Además, al centrar el histograma de la imagen, se observó ruido de fondo. Este ruido de fondo es causado por las cámaras, óptica del microscopio y preparación de las bacterias. Como dicho ruido se asemeja a un ruido blanco, se ha reducido (y en ocasiones eliminado) utilizando un filtro paso bajo para la eliminación del ruido.

Con este procesado previo de las imágenes se han ensayado cuatro tipos de soluciones que se explicarán a continuación. Para poder realizar la segmentación, se han utilizados dos tipos de clasificadores: Viola-Jones y un algoritmo específico que utiliza clasificadores lineales de la librería SVM. Para poder reconocer las bacterias, una vez segmentada la imagen, se han utilizado dos técnicas: Simulated Annealing y una técnica específica basada en “dilatación y erosión de objetos”.

4.1.1 Viola-Jones

El primer algoritmo que se ha probado como solución a la especificación es el Viola-Jones. Dado que dicho algoritmo resulta efectivo en el reconocimiento y la detección de caras, se ha pensado que resultaría de gran interés probar su efectividad como detector de bacterias.

Como se comentará más adelante, aunque se han probado dos herramientas basadas en Viola-Jones, no ha sido posible realizar una segmentación adecuada con esta técnica. En el apartado de resultados se propondrá una posible causa de este problema. Por ello, se han explorado otras alternativas, como los clasificadores lineales.

4.1.2 Clasificador Lineal

Como alternativa al Viola-Jones, se ha optado por utilizar una aproximación basada en un clasificador lineal. En este punto, se ha utilizado como soporte la librería SVM, por lo

que el clasificador se ha implementado con funciones de dicho entorno. Este tipo de clasificadores requiere un elevado tiempo de entrenamiento. Para poder reducir el tiempo de entrenamiento y mejorar los resultados, se ha optado por un análisis y optimización del conjunto de datos de entrenamiento. Para ello, se ha utilizado el módulo PCA (Principal Component Analysis) de OpenCV. Este módulo realiza un análisis de la similitud (co-varianza) entre las distintas imágenes del conjunto de datos de entrenamiento y, a partir del mismo, obtiene los vectores característicos. Con esta información genera un conjunto reducido de imágenes de entrenamiento (50 por tipo de imagen).

Además, para utilizar este tipo de clasificador ha sido necesario añadir una transformación adicional a la corrección del histograma, en el paso de procesado de imagen. En este caso, se ecualizará el histograma para obtener una imagen más definida.

4.1.3 Simulated Annealing

Una vez la imagen se ha segmentado, de forma que las bacterias de diferencian del fondo, el paso siguiente es individualizar cada bacteria. La imagen segmentada es una imagen binaria, en donde cada píxel solo puede tomar dos valores: 0 (fondo) o 1 (bacteria).

Para individualizar las bacterias en la imagen segmentada, se ha desarrollado un algoritmo del tipo Simulated Annealing. Este algoritmo intenta colocar una “bacteria virtual” encima de cada bacteria física. El proceso de colocación es optimizado gracias al algoritmo de Simulated Annealing. Una vez se han colocado células virtuales encima de todas las existentes, se consigue obtener el número total de células.

En el capítulo de resultados se explicará cual ha sido la calidad de los datos obtenidos mediante esta técnica. Además, en este capítulo, se explicará con más detalle en que consiste el algoritmo utilizado.

4.1.4 Algoritmo de Dilatación y Erosión

Por último, para individualizar las bacterias en la imagen segmentada, también se ha utilizado un algoritmo basado en las técnicas de dilatación y erosión de imágenes. En el capítulo dedicado a este algoritmo se explicará en qué consisten estas técnicas y cómo se han utilizado en este algoritmo. Esta técnica para identificar bacterias se ha combinado con el clasificador lineal para desarrollar la solución propuesta en el proyecto.

En el apartado de resultados, se comparará esta técnica con las anteriores y se detallarán los resultados proporcionados por dicho algoritmo.

4.2 Procesado de las Imágenes

Uno de los principales problemas encontrados a la hora de realizar este proyecto ha sido el procesado de las imágenes.

Se disponía de un catálogo de imágenes proporcionadas por el IBBTEC, con tres componentes: fase, verde y rojo. Sin embargo, cuando en un primer momento se observan las imágenes de cualquiera de las tres componentes, resulta muy difícil ver algo en ellas. Es por ello, que lo primero que se hizo fue analizar el histograma de la imagen para determinar la distribución de valores de los píxeles.

Este estudio determinó que el problema que presentaban estas imágenes es que el histograma se encontraba desplazado hacia la izquierda y, por tanto, toda la imagen se veía mucho más oscura de lo que en realidad era. Por esto, se decidió centrar el histograma de la imagen y obtener una imagen mucho más clara, como se ve en la Figura 10.

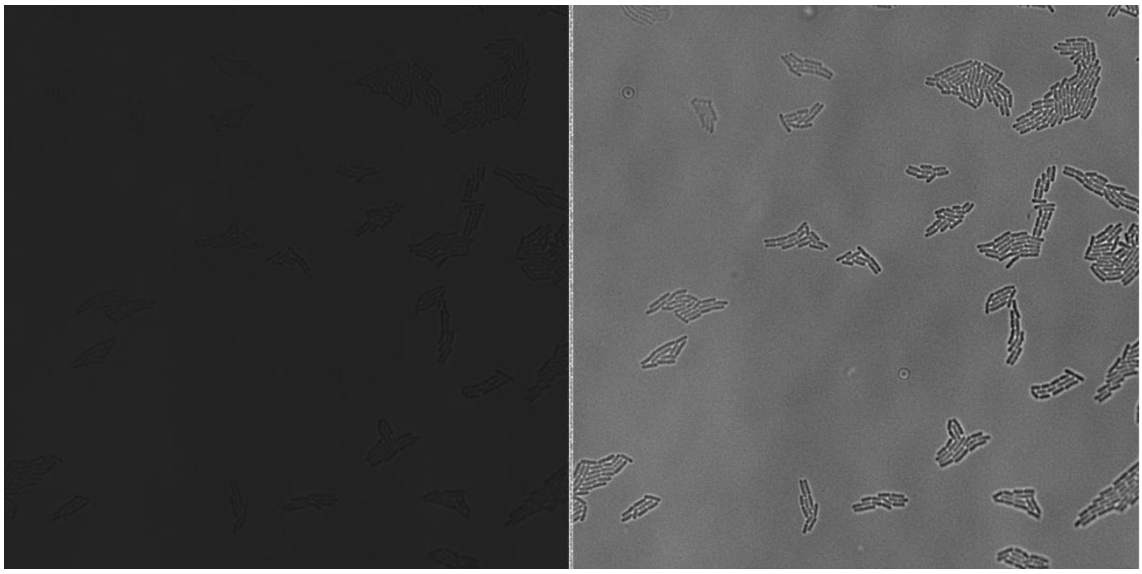


Figura 10: Imagen Original (izquierda) Imagen con Histograma Centrado (derecha)

Además, al centrar el histograma de las imágenes se detectó ruido de fondo debido, principalmente, a las cámaras, preparación de las bacterias y la óptica del microscopio. Como se trata de un ruido blanco, se ha eliminado utilizando una función de OpenCV

(*fastNlMeansDenoising*) que implementa un filtro de reducción de ruido. El problema que se ha encontrado al aplicar este filtro, es que si se intenta eliminar todo el ruido, se distorsiona también parte de la imagen perdiéndose nitidez. Por tanto, es importante buscar el compromiso entre eliminación de ruido y nitidez.

Como resultado de aplicar el filtro de reducción de ruido a la imagen anterior se obtiene la imagen mostrada en la Figura 11. En ella, se puede apreciar que el fondo de la imagen es mucho más liso, lo que significa que el ruido de fondo se ha eliminado. Esto provoca que las bacterias queden más resaltadas sin distorsionarse.

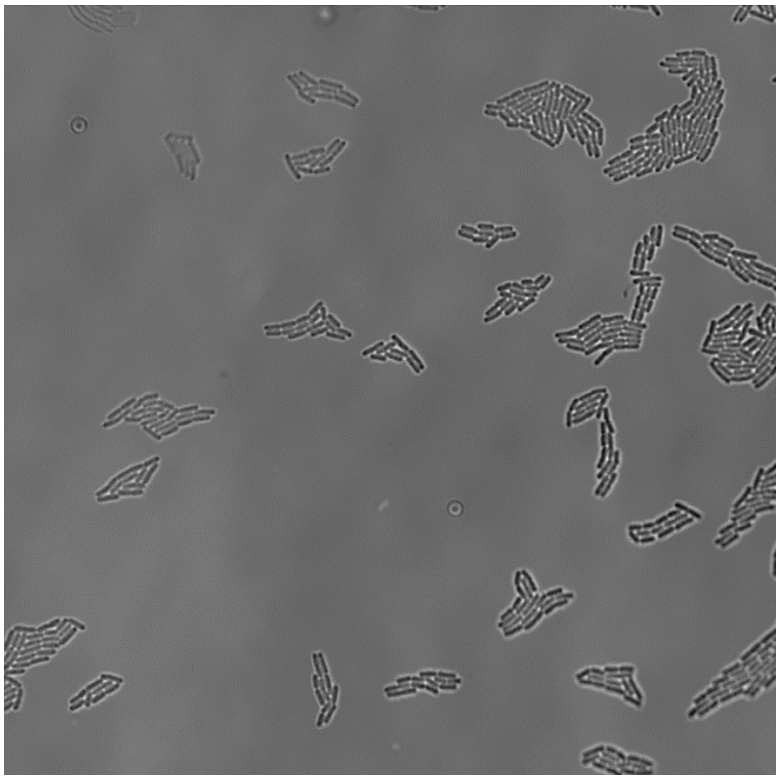


Figura 11: Imagen Filtrada

Aun con un buen ajuste del filtro, existen zonas de la imagen donde el filtro distorsiona la misma. Estas zonas ya aparecían desenfocadas en la imagen inicial, por lo que a la hora de realizar la detección de bacterias estas zonas son obviadas.

4.3 Soluciones Exploradas

Tras procesar inicialmente las imágenes, se han explorado tres soluciones para resolver el problema del recuento de las bacterias. En primer lugar, se utilizó un algoritmo de tipo Viola-Jones para segmentar la imagen y reconocer bacterias. Como dicho proceso no obtuvo los resultados esperados, se optó por implementar un clasificador lineal. Este clasificador fue combinado con dos tipos de algoritmo para individualizar bacterias: "Simulated Annealing" y "Dilatación y Erosión". Los mejores resultados se han obtenido combinando el clasificador lineal con la técnica de "dilatación y erosión".

A continuación se presentan las distintas técnicas exploradas.

4.3.1 Viola-Jones

Como se comentó en el estado del arte de este proyecto, el algoritmo Viola-Jones se utiliza en muchas aplicaciones de detección facial, por lo que a priori parece una técnica interesante para alcanzar los objetivos de este proyecto.

Se ha elegido este algoritmo como primera opción para realizar la detección de bacterias debido a que es un algoritmo robusto, rápido y que cuenta con mucho soporte, lo que facilita el trabajo de desarrollo.

Dentro de todas los posibles variantes que existen del algoritmo de Viola-Jones, se han evaluado dos diferentes: una versión típicamente utilizada en detección facial [8] y otra basado en funciones de OpenCV [10].

4.3.1.1 Algoritmo Viola-Jones de Detección Facial

La primera de las dos técnicas ensayadas, está basada en un algoritmo Viola-Jones diseñado para la detección facial. El entorno incluye un entrenador y un detector, así como una parte de post-procesado de la imagen que se encargará de individualizar cada cara eliminando falsos positivos. El entrenador se utilizará únicamente una vez para configurará el entorno, siendo utilizado el detector para segmentar la imagen y reconocer las bacterias.

El entrenador de Viola-Jones está basado en AdaBoost, es decir, genera un clasificador robusto a partir de muchos clasificadores débiles. El algoritmo utiliza un método de votación para generar el clasificador final. La técnica parte de los resultados que generan cada uno de los clasificadores débiles y que miden la presencia de una cara en la ventana que se está analizando. Estos resultados pueden ser positivos o negativos, es decir, un

clasificador puede decir que hay una cara en un punto concreto (resultado positivo) mientras que otro dice que no la hay (resultado negativo). Finalmente, para saber si realmente hay una cara o no, el algoritmo utiliza un método de votación.

Por otra parte, el detector se basa en un algoritmo que se encarga de prestar mayor atención a las zonas donde es más probable encontrar una cara que a las zonas donde la probabilidad es baja. Esto se hace mediante una cascada de múltiples capas. El uso de capas permite definir zonas en donde la probabilidad de encontrar una cara es mayor. En estas zonas, se emplearán más recursos computacionales para la detección de caras mientras que en las zonas donde la probabilidad es baja, se prestará menor atención. Esto genera un algoritmo más eficiente, desde el punto de vista de computación.

Por último, este algoritmo incluye una parte de post-procesado de la imagen, cuya función principal es la de reconocer caras individuales, eliminando los falsos positivos. Resulta interesante la utilización de esta parte de post-procesado ya que este algoritmo tiende a ser redundante, es decir, a dar positivos muchas veces en el mismo sitio como se observa en la Figura 12. Sin embargo, tras aplicarle el post-procesado a la imagen se puede ver en la Figura 13 que se han eliminado falsos positivos así como positivos redundantes.

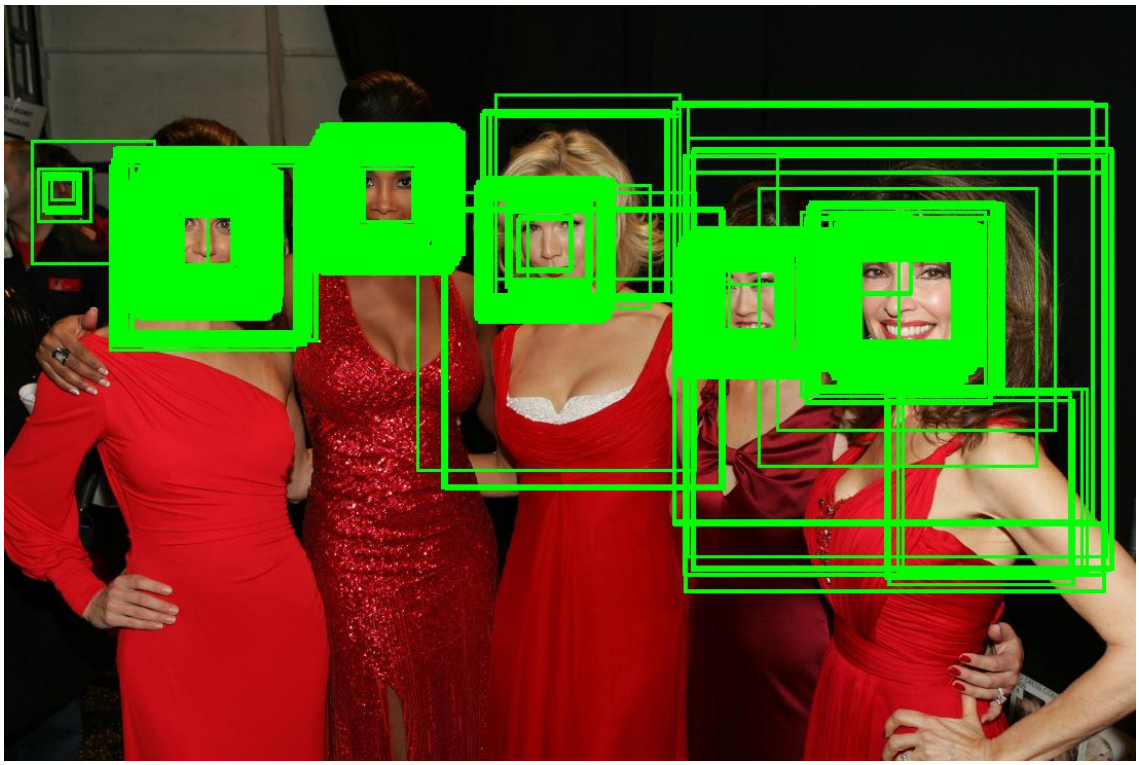


Figura 12: Ejemplo de Falsos Positivos y Positivos Redundantes [8]

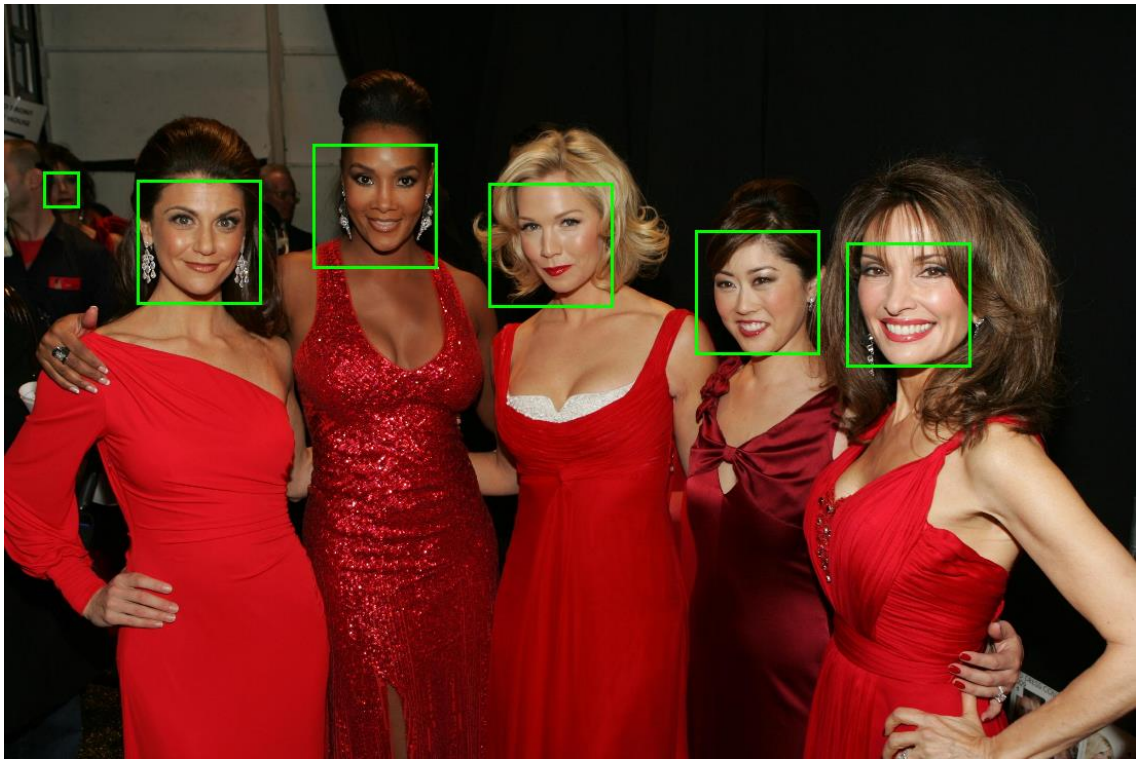


Figura 13: Ejemplo de Funcionamiento del Algoritmo de Post-Procesado [8]

Para conseguir un detector de bacterias utilizando este algoritmo, en primer lugar, es necesario entrenar el algoritmo para, posteriormente, utilizar el detector que reconoce (segmenta) las bacterias.

Para entrenar el algoritmo, es necesario proporcionarle dos bases de datos: una de ellas incluye imágenes con bacteria (muestras positivas) mientras que la otra solo incluye muestras negativas. Es decir, la base de datos de las muestras positivas contendrá imágenes con una bacteria mientras que la otra base de datos, la de las muestras negativas, contendrá imágenes de lo que no son bacterias.

Todas las imágenes que se introducen en la base de datos tienen que tener el mismo tamaño y, en el caso de las muestras positivas deben ajustarse a lo que es una bacteria. Es importante incluir imágenes de diversos tipos, como por ejemplo bacterias que tengan otras bacterias pegadas, ya que esto hará que el detector generado sea más robusto.

Las imágenes que conforman la base de datos son pequeños fragmentos de las imágenes tomadas en el microscopio y que tenían que ser clasificadas. Además, en las muestras negativas se han incluido zonas ruidosas de las imágenes y manchas existentes en ellas. La base de datos final, la resultante de unir las dos bases de datos anteriores, contiene 800 imágenes de las cuales 400 son muestras positivas y 400 muestras negativas.

Dado que se trata de un algoritmo desarrollado previamente, se han seguido todas las instrucciones proporcionadas para su correcto funcionamiento. Además, ha sido necesaria la instalación de una librería llamada libpng. Todo el desarrollo se ha realizado sobre Ubuntu 14.04 ya que el código utilizado requería ciertas librerías únicamente disponibles en Linux.

Para utilizar este código, en primer lugar es necesario entrenar la cascada de clasificadores, para lo que se ejecutará el comando:

./startTraining.sh

Este comando genera un archivo llamado *Detector.cxx*. Para utilizar este detector, se le cambiará el nombre al archivo de *Detector.cxx* a *Detector.cpp* y se compilará de nuevo el código. Seguidamente se ejecutarán los siguientes comandos:

./train > trainLog 2>&1

./detect imagen.png

El comando *detect* necesita como argumento de entrada la imagen a clasificar. Como argumentos adicionales, se pueden ajustar dos parámetros. El primer parámetro es el número de capas que utilizará en la cascada de clasificadores y el segundo parámetro es un umbral que se utiliza para darle robustez al algoritmo en la parte de post-procesado de la imagen.

Los resultados obtenidos con este algoritmo, se expondrán y compararán con las otras técnicas y algoritmos utilizados en el capítulo de resultados.

4.3.1.2 Algoritmo Viola-Jones Basado en OpenCV

Es importante contrastar los resultados obtenidos con varias técnicas diferentes para obtener la solución que proporcione un mejor detector. Es por ello, que se ha probado otro algoritmo de tipo Viola-Jones. En este caso, el algoritmo está basado en la utilización de unas funciones que proporciona OpenCV para entrenar algoritmos Viola-Jones.

OpenCV proporciona dos funciones, "*opencv_createsamples*" y "*opencv_traincascade*" para entrenar y obtener un clasificador de tipo cascada. Para realizar el entrenamiento con estas funciones, se necesita crear nuevamente una base de datos con muestras positivas y muestras negativas.

En primer lugar, se necesita ejecutar "*opencv_createsamples*". Esta función prepara el conjunto de datos para la posterior ejecución de "*opencv_traincascade*". El comando que se ha ejecutado es:

opencv_createsamples -info bact.info -num 25 -w 46 -h 20 -vec bacts.vec

El argumento "*–info*" se utiliza para indicarle el fichero donde se especifica el catálogo de imágenes positivas. Este fichero tiene que tener la siguiente estructura:

pos/image338.png 1 0 0 46 20

En primer lugar se especifica la ruta de la imagen, a continuación el número de objetos que contiene, en este caso la imagen contiene una bacteria. Seguidamente las coordenadas de la imagen donde se encuentra el principio del objeto y por último el tamaño de la ventana que engloba al objeto.

El argumento *–num* indica el número de imágenes que hay en la carpeta pos, donde se han guardado las muestras positivas. A continuación, *-w* y *-h* definen la anchura y altura de las muestras de salida, es decir, el tamaño de las imágenes de la base de datos. Por último, el argumento *–vec* indica el fichero .vec donde se almacenará el dataset creado.

OpenCV proporciona dos funciones cuya función es la misma, entrenar un clasificador de tipo cascada. La principal diferencia que existe entre *opencv_haartraining* y *opencv_traincascade* es que *opencv_traincascade* es una versión más nueva de la otra función por lo que soporta los clasificadores de tipo Haar y los de tipo LBP (Local Binary Pattern). Esta versatilidad hace que se haya elegido utilizar la función *opencv_traincascade* para entrenar el clasificador.

La diferencia de resultados obtenida entre los clasificadores Haar y LBP depende tanto de la calidad del dataset como de los parámetros del entrenamiento. Sin embargo, en la mayoría de los casos los clasificadores basados en LBP son más rápido que los basados en Haar.

Una vez se ha creado el dataset con *opencv_createsamples* se realizará el entrenamiento para obtener el clasificador final que se va a utilizar. Para ello, se ha empleado el siguiente comando:

opencv_traincascade -data data -vec bacts.vec -bg nobact.txt -numStages 6 -minhitrate 0.999 -maxfalsealarm 0.3 -numPos 25 -numNeg 400 -w 46 -h 20 -featureType LBP

El argumento *–data* especifica el directorio donde se guardarán los resultados. El siguiente argumento, *-vec* sirve para precisar el nombre del archivo correspondiente al dataset generado por *opencv_createsamples*. Seguidamente, *-bg* determina el archivo de background. Este archivo contiene la ruta de las imágenes que se utilizarán como muestras negativas. El formato usado en este archivo es el siguiente:

neg/image_no037.png

Simplemente, se pone la ruta a las imágenes, cada una en una línea diferente. El siguiente argumento de la función *opencv_traincascade* es *–numStages* que determina el número de fases que se utilizarán para el entrenamiento. Aunque a priori se puede llegar a pensar que cuanto mayor sea este número mejor será el resultado, en ocasiones un número de fases muy elevado puede derivar en un entrenamiento muy lento. Además, generalmente llega un punto en el que aumentar el número de fases no implica un aumento considerable en la fiabilidad del clasificador.

El siguiente argumento de la función es *–minhitrate* que es la mínima tasa de éxito que se le permite a cada clasificador. A continuación se define el *–maxfasealarm* que es la tasa máxima de falsos positivos permitidos a cada clasificador de la cascada. Por último, se definen *–numPos* y *–numNeg*, -w y *–h* que se corresponden con el número de muestras positivas, el número de muestras negativas y la anchura y la altura de las muestras, respectivamente. Finalmente, el argumento *–featureType* define el tipo de clasificador que se quiere usar ya sea Haar o LBP.

Al ejecutar el entrenamiento, se generan numerosos ficheros con extensión .xml. Cada fichero de nombre *stage* se corresponde con una fase del entrenamiento. Además, al finalizar todo el entrenamiento se genera un archivo llamado *cascade.xml* donde se encuentra el clasificador generado.

Es importante comentar, que los parámetros del entrenamiento tienen que ser optimizados en función del tipo de imágenes del que se disponga, así como de la robustez que se quiera obtener. Además, estos parámetros utilizados han sido optimizados siguiendo la guía de la referencia [10].

Para comprobar si el clasificador funciona, se ha utiliza un script [9] que toma como entrada la imagen a clasificar y el archivo cascade.xml y proporciona como salida, la imagen de entrada con los objetos encontrados por el clasificador.

En el apartado de resultados de este proyecto, se comentará la calidad de la solución obtenida mediante este método y se comparará con el resto de soluciones.

4.3.2 Descripción del Clasificador Propuesto

Como el algoritmo de Viola-Jones no proporcionó los resultados esperados, se ha desarrollado un nuevo clasificador que utiliza técnicas de la librería SVM. Este clasificador transforma los datos a un espacio de dimensión alta, mediante una función (kernel) de base radial gaussiana. En dicho espacio se utiliza un híper-plano para separar los distintos tipos de datos, en este proyecto los distintos tipos de imágenes.

Partiendo de las imágenes con el procesado previo, presentado en el apartado 4.2, realizado se ha implementado una transformación adicional: la ecualización del histograma de las imágenes. Aunque previamente se había centrado, resulta necesario ecualizarlo para obtener una imagen con más contraste.

Para ecualizar el histograma de las imágenes, se ha utilizado una función de la librería OpenCV (*cvEqualizeHist*). Esta función simplemente toma una imagen de entrada y proporciona como salida la imagen con el histograma corregido. Un ejemplo del funcionamiento de esta función se puede ver en la Figura 14, donde se observa claramente que al ecualizar el histograma se obtiene un mayor contraste en la imagen.

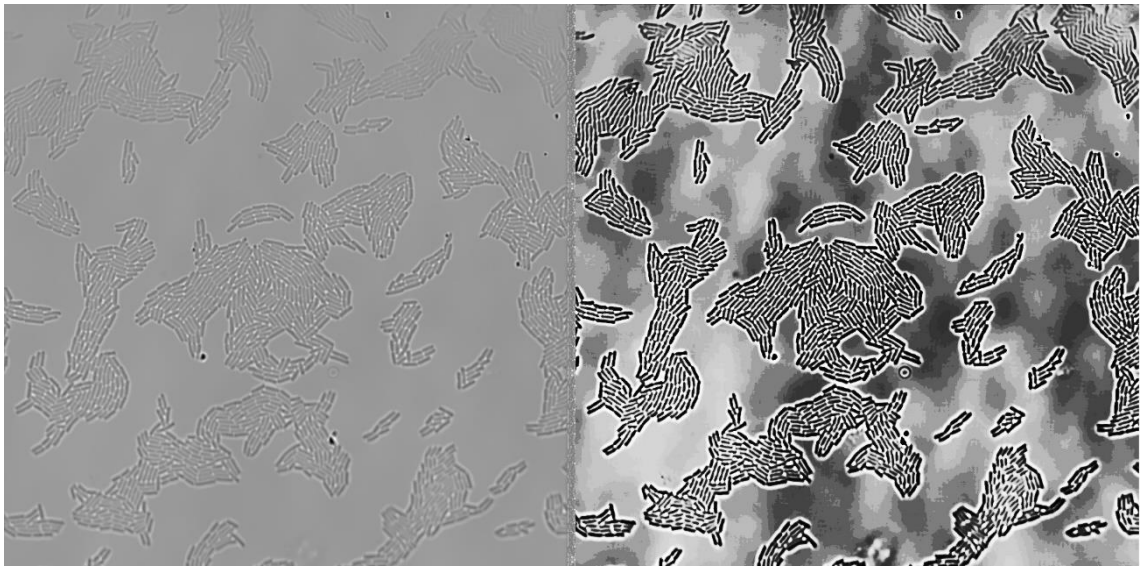


Figura 14: Imagen con Histograma Centrado y Ruido Filtrado (izquierda) Imagen con el Histograma Ecualizado (derecha)

A continuación se aplica el clasificador desarrollado. La presentación se divide en dos pasos: proceso de entrenamiento y algoritmo de clasificación utilizando la técnica propuesta.

4.3.2.1 Entrenamiento del Algoritmo Propuesto

En la Figura 15 se muestra un diagrama donde se puede ver cómo funciona el algoritmo de entrenamiento.

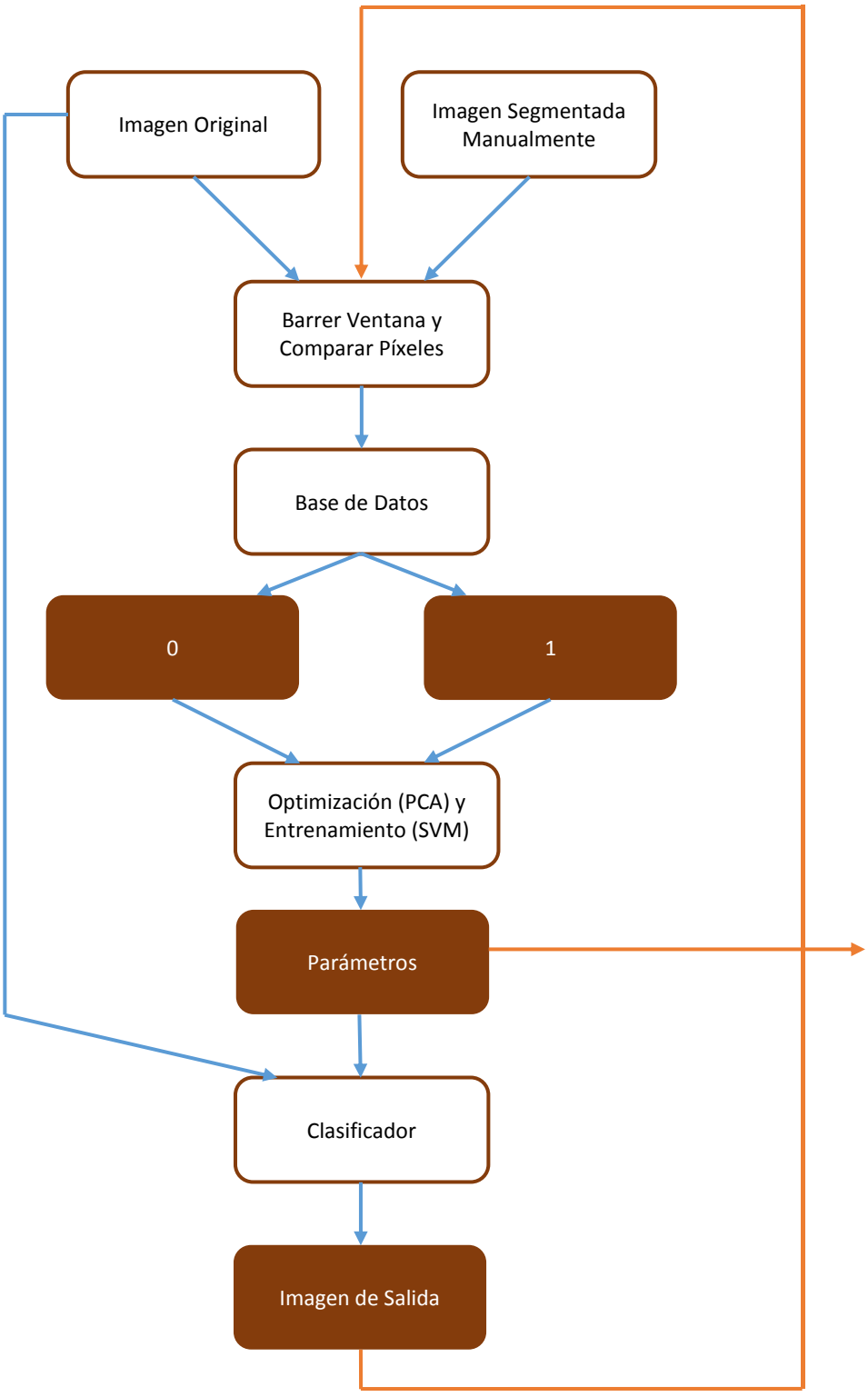


Figura 15: Diagrama de Funcionamiento del Algoritmo de Entrenamiento

En primer lugar se ejecuta una función que crea dos bases de datos en dos carpetas: carpeta 0 (imágenes no segmentadas) y carpeta 1 (imágenes segmentadas). Estas bases de datos servirán para entrenar posteriormente el clasificador. Para crear la base de datos, la función recibe como entrada una imagen y su segmentación manual, así como el tamaño de la ventana que se quiere utilizar para recorrer la imagen y el ángulo de rotación. Por tanto, para entrenar al algoritmo se necesita segmentar previamente una imagen a mano.

Para crear las bases de datos, la función recorre la imagen utilizando una ventana de T x T píxeles, donde T es el tamaño de la ventana especificado como argumento de entrada por el usuario. Esta ventana se va desplazando hasta que se ha recorrido toda la imagen. Para cada ventana, se lee el valor del píxel central de dicha ventana en la imagen segmentada. En el caso de que el píxel sea blanco (seleccionado), se almacena la ventana correspondiente de la imagen no segmentada en la carpeta 1 mientras que si el píxel es negro, la imagen de la ventana se almacena en la carpeta 0.

De esta forma, al terminar de ejecutar esta función, se obtienen dos bases de datos con imágenes de tamaño T x T, que corresponden con fragmentos de la imagen sin segmentar. Por lo tanto, en la base de datos 1 se almacenan imágenes de T x T píxeles que se corresponden con regiones de la imagen sin segmentar, cuyo píxel central ha sido seleccionado (valor 1) en la imagen segmentada. Cuando esta condición no se verifique, la imagen de T x T píxeles será almacenada en la base de datos 0. El objetivo del clasificador será, por lo tanto, identificar ventanas de tamaño T x T cuyo píxel central tenga que aparecer como 1 en la imagen segmentada. A la hora de generar una base de datos, es recomendable utilizar más de un ángulo de rotación de la imagen. Esto hace que la base de datos contenga casos distintos y que así se genere un entrenamiento más robusto. Además, el clasificador será independiente del ángulo de rotación de la bacteria. A continuación, se ejecutará el algoritmo de entrenamiento, que hace uso de las bases de datos anteriormente generaras. Dicho proceso consta de dos pasos:

1. Análisis de componentes principales, con objeto de reducir las dimensiones del problema. Las bases de datos antes comentadas contienen una imagen por cada píxel de la imagen sin segmentar. A su vez, cada imagen tiene T x T píxeles, por lo que el número de dimensiones a estudiar es muy elevado. De hecho, las bases de datos se transforman en un nuevo conjunto de datos con dos ejes: imagen y píxeles de la imagen. Si la base de datos contiene 700 imágenes y cada una tiene 20 x 20 = 400 píxeles, se crea un problema con 1 (imágenes) + 400 (valor del píxel) = 401 dimensiones. Para reducir la complejidad del problema se utiliza un análisis de tipo PCA (Principal Component Analysis). Dicho análisis permite identificar las imágenes y los píxeles más relevantes de las bases de datos. Para ello calcula la covarianza entre los conjuntos de datos y calcula los vectores y valores propios. Como resultado, se obtiene un conjunto reducido (en este proyecto formado por

tan solo 50 elementos) de imágenes y píxeles que modelan las características del conjunto inicial. Por lo tanto la dimensión del problema se ha reducido de 401 a solo 51 dimensiones.

2. Entrenamiento del clasificador. Una vez reducido el conjunto de datos de entrenamiento, estos se utilizan para crear un modelo de clasificador usando SVM.

En este punto, el algoritmo de entrenamiento utiliza un clasificador para generar una imagen segmentada de la imagen sin segmentar. El funcionamiento de este módulo se puede ver en la Figura 16. Esta función recibe como argumento de entrada la imagen a segmentar y, utilizando los parámetros generados por el entrenamiento, genera una imagen de salida que será la imagen original segmentada. El clasificador utiliza los resultados del análisis PCA y del entrenamiento con SVM para generar una imagen segmentada.

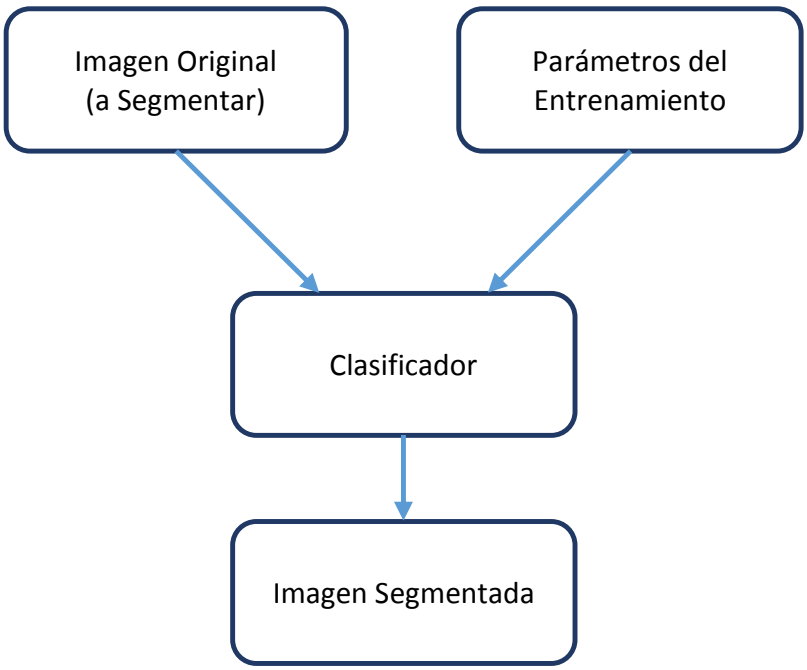


Figura 16: Diagrama de Funcionamiento del Clasificador

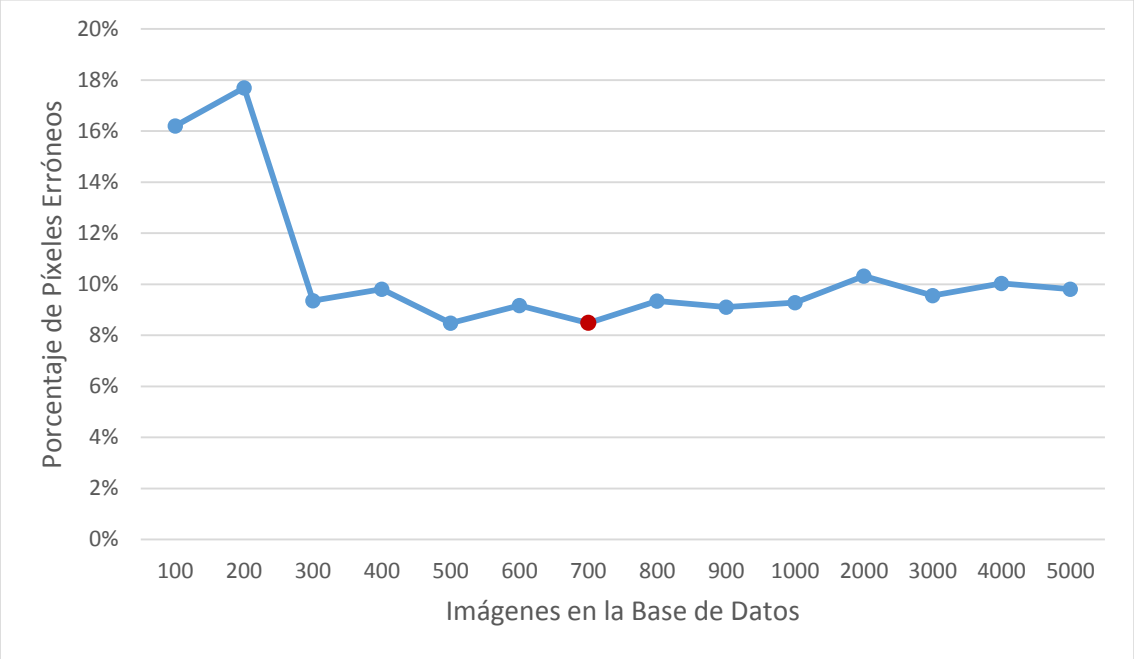
Sin embargo, hay ocasiones en las que esto no resulta suficiente. Es por ello, que se ha añadido una nueva iteración al proceso de entrenamiento (línea naranja en la Figura 15), que incrementa las bases de datos basándose en los píxeles erróneos generados en la primera clasificación.

El funcionamiento es muy similar al de la primera iteración. La principal diferencia es que en este caso, además de la imagen a segmentar y la imagen manualmente segmentada, también se utilizará la imagen de salida del clasificador. En esta ocasión, la

función lo que hace es volver a recorrer la imagen con la ventana seleccionada pero el píxel central de la ventana no es comparado solo con el píxel correspondiente de la imagen segmentada, sino que también se tiene en cuenta la salida obtenida en la primera iteración (imagen segmentada automáticamente). Si el valor de los píxeles coincide en la imagen con segmentación manual y automática, la imagen de la ventana no se añade a la base de datos. Sin embargo, si el valor del píxel no coincide en ambas imágenes, se añadirá a la base de datos la imagen de la ventana en la carpeta 0 ó 1 según corresponda.

Para la realización del entrenamiento, se ha creado una base de datos con imágenes tanto de la primera como de la segunda iteración. Se ha decidido que el 30% de las imágenes de la base de datos pertenecen a la primera iteración mientras que el 70% restante pertenecen a la segunda. Esta proporción ha sido tomada basándose en el tipo de imágenes que añade a la base de datos cada iteración. Por un lado, la primera genera imágenes de casos más triviales mientras que en la segunda iteración, las imágenes que se incluyen se corresponden con casos más difíciles de detectar ya que está basada en los errores de la primera iteración. Es por ello, que se ha escogido esta proporción a la hora de crear las bases de datos y entrenar el algoritmo.

Aunque en un principio se puede pensar que cuantas más imágenes tenga la base de datos mejor será el resultado del entrenamiento, esto no es cierto. Para ver cuál es el número de imágenes en la base de datos con el que se obtiene un mejor resultado, se ha realizado el entrenamiento y la clasificación para varias bases de datos obteniéndose los resultados mostrados en la siguiente gráfica:



En la anterior gráfica se puede observar que pese a aumentar el número de imágenes en la base de datos, no se consiguen mejores resultados. Es más, el mejor resultado se ha conseguido con 700 imágenes por lo que es innecesario aumentar la bes de datos ya que conllevaría al empeoramiento de los resultados. Además, cuanto mayor sea el número de imágenes en la base de datos, mayor es el tiempo que se necesita para entrenar el algoritmo.

Pese a que las imágenes segmentadas solo tienen en torno a un 8% - 9% de píxeles erróneos, existen zonas en las que la segmentación proporcionada por el algoritmo propuesto no es del todo correcta. Como se observa en la Figura 17 , existen algunas partes de la imagen donde el clasificador ha coloreado de blanco zonas del fondo próximas a las bacterias creando un efecto de doble borde. Además, las bacterias aparecen pegadas, no diferenciándose donde termina una y empieza la siguiente. Para solucionar estos problemas se ha combinado el clasificador con otros algoritmos, como Simulated Annealing o técnicas de búsqueda de contornos cerrados. Estas alternativas se presentarán a continuación.

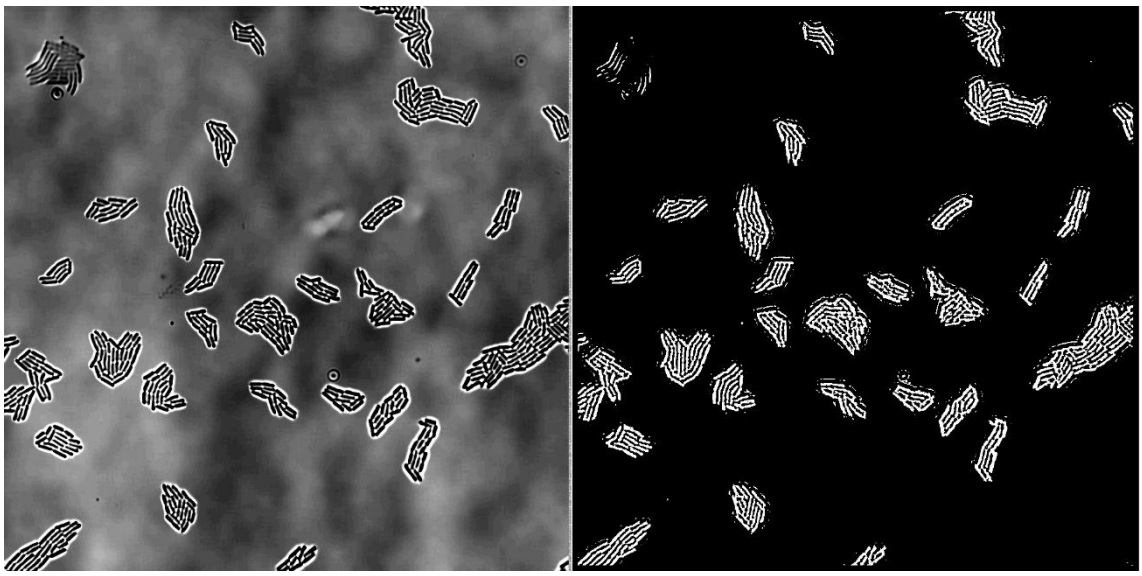


Figura 17: Imagen de Entrada del Entrenamiento (izquierda) Imagen de Salida del Entrenamiento (derecha)

4.3.2.2 Algoritmo de Clasificación

Una vez el algoritmo ha sido entrenado, solo se necesita utilizar el clasificador para segmentar una imagen. El esquema del clasificador fue presentado en la Figura 16. Como se observa en dicha figura, el clasificador recibe una imagen procesada y, teniendo en cuenta los parámetros del entrenamiento, genera una imagen segmentada.

4.3.3 Clasificador Propuesto con Simulated Annealing

Como se ha visto en el punto anterior, las características de la imagen segmentada generada por el clasificador propuesto hacen que un algoritmo de detección de contornos o de búsqueda de contornos cerrados, no pueda ser aplicable a este tipo de imágenes ya que no proporcionaría un resultado adecuado. Es por ello, que se ha ensayado una optimización utilizando el algoritmo de Simulated Annealing.

Como se explicó en el estado del arte de este proyecto, el algoritmo Simulated Annealing lo que busca es encontrar el mínimo de una función. Aplicado a este caso concreto, el mínimo se obtiene cuando todas las bacterias "virtuales" están en la posición de las "bacterias reales". Para conseguir minimizar la función, se ha desarrollado un algoritmo que se describirá a continuación.

La idea fundamental de este algoritmo consiste en ser capaz de posicionar una bacteria "virtual" donde realmente se encuentra la real en la imagen y así obtener el número total de ellas. La forma en que funciona el algoritmo es básicamente, generar una bacteria "virtual" aleatoria e ir "enfriándola" hasta que se encuentre en el estado que minimice la energía (bacteria virtual sobre una bacteria real). Este proceso se deberá repetir hasta que no queden estados que minimicen la energía de nuevas bacterias (todas las bacterias estarán cubiertas).

La siguiente pregunta que se plantea es: ¿cómo se obtiene la energía que proporciona una bacteria?, y por tanto ¿cómo se minimiza la energía? La respuesta a la primera pregunta es sencilla: si se posiciona una bacteria "virtual" en una zona aleatoria de la imagen, con un tamaño aleatorio, comprendido entre un rango de valores coherente, y un ángulo con respecto a la horizontal también aleatorio, la energía que esta bacteria virtual tiene es el número de píxeles blancos (píxeles de las bacterias reales) de la imagen que esté cubriendo. Es decir, si la bacteria se posiciona sobre el fondo tendrá una energía muy elevada ya que no cumbre ningún pixel blanco. Sin embargo, si la bacteria virtual se coloca justamente sobre otra bacteria, la energía es muy pequeña ya que está cubriendo muchos píxeles blancos e incluso podría llegar a darse el caso de que toda la superficie que ocupa esta bacteria posicionada sea de píxeles blancos.

Como se puede ver en la Figura 18, la bacteria virtual inicialmente se había posicionado sobre el fondo. Sin embargo, al ir enfriándose, se ha conseguido posicionar sobre otra bacteria, adquiriendo un estado que minimiza la energía. Este proceso se repetirá hasta que todas las bacterias queden detectadas. Como las bacterias virtuales se generan una a una, cuando la ejecución ha acabado, se puede saber el número exacto de bacterias detectadas. Además se guardará la imagen de salida con todas las bacterias marcadas por lo que se ha realizad un recuento y un marcaje (identificación) de las bacterias.

Figura 18: Imagen Original (izquierda) Imagen con Bacteria en Fondo (centro) Imagen con Bacteria Detectada (derecha)

Más adelante, en el capítulo de resultados, se comentará como de preciso resulta este algoritmo y se comparará con el resto de técnicas ensayadas.

4.3.3 Clasificador Propuesto con Algoritmo de Dilatación y Erosión

El esquema de funcionamiento de esta técnica puede observarse en la Figura 19. En ella se muestra el clasificador propuesto conjuntamente con los algoritmos de dilatación y erosión combinados con técnicas de región de crecimiento. Para la identificación de las bacterias, esta metodología utiliza imágenes segmentadas y busca las zonas donde se separan las bacterias para contarlas e identificarlas. Además, como existen ciertas zonas donde dos bacterias colindantes no se han separado del todo, el algoritmo también busca estrechamientos dentro de las áreas segmentadas para separar dos bacterias que a priori parecían una.

Ya que el algoritmo busca las pequeñas separaciones entre las bacterias, se necesita tener una segmentación de la imagen de la mayor calidad posible. Para ello, se ha utilizado el algoritmo propuesto en el apartado anterior. Dado que este algoritmo presenta pequeños fallos, se ha decidido combinarlo con la segmentación que proporciona OpenCV (*cvTrheshold*), basada en segmentación por umbral.

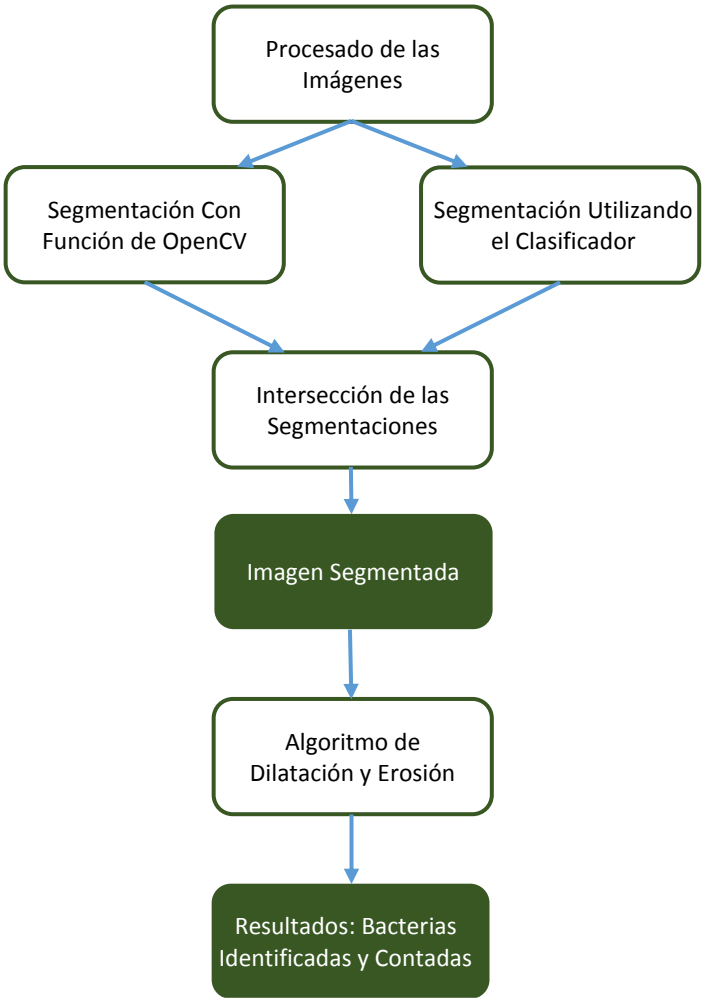


Figura 19: Diagrama del Algoritmo de Dilatación y Erosión

El principal problema de la segmentación por umbral es que, como las imágenes no tienen las mismas condiciones de iluminación ni de enfoque, resulta muy difícil fijar un valor umbral que proporcione una buena segmentación en todas las imágenes. Como se puede ver en la Figura 20, la segmentación por umbral que se obtiene cuando se utiliza el mismo valor umbral en dos imágenes con diferente iluminación es muy diferente. Mientras que en la imagen de la izquierda la segmentación es bastante buena y sin manchas en el fondo, en la imagen de la derecha, se detectan grandes manchas. Esto se debe a que al utilizar el mismo umbral en imágenes que no tienen las mismas condiciones de iluminación o condiciones similares, el umbral fijado puede no ser el adecuado.

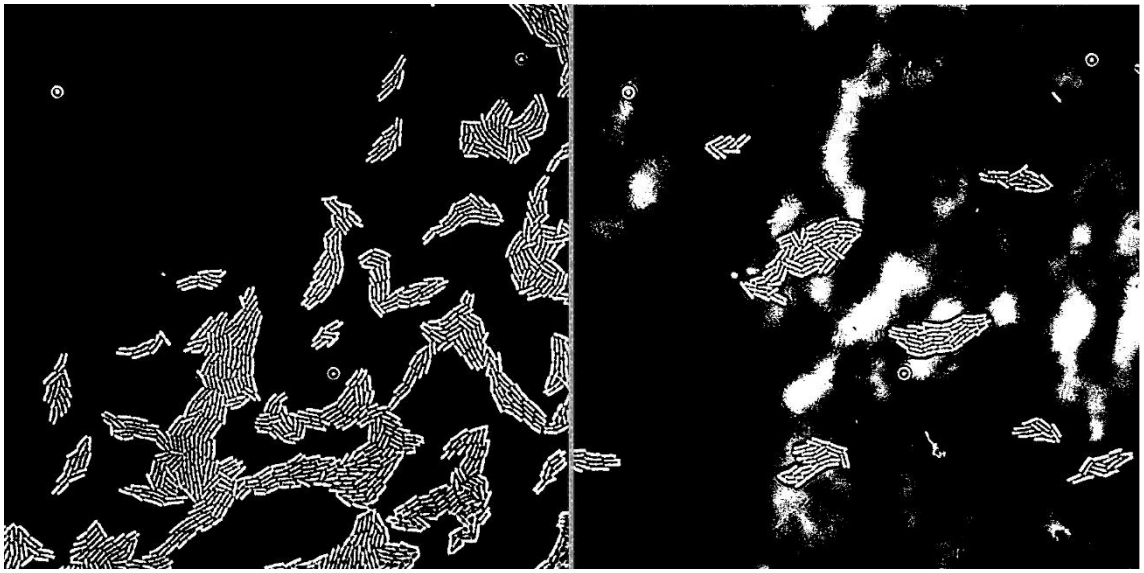


Figura 20: Comparación de Imágenes Segmentadas con OpenCV

Sin embargo, y pese a los inconvenientes que presentan el clasificador propuesto y las técnicas de segmentación por umbral por separado, si se combinan ambas, se consigue obtener unas imágenes con una buena calidad de segmentación. Esto se debe a que el segmentador por umbral de OpenCV es capaz de segmentar bastante bien las células pero generalmente provoca manchas en el fondo cuando se ajusta mucho el umbral. Por otro lado, el algoritmo propuesto, no produce manchas en el fondo y es capaz de separar bacterias que la segmentación por umbral no detecta. Sin embargo, el clasificador propuesto no produce contornos de bacterias tan detallados con la segmentación por umbral.

Por tanto, si se segmentan todas las imágenes utilizando ambas técnicas y posteriormente se combinan los resultados, se consigue una segmentación de alta calidad. Cuando se habla de combinar las imágenes, lo que se quiere decir es que se generará una nueva imagen, en la cual un píxel tomará valor blanco solo si es blanco en ambas imágenes. De esta forma, las manchas en el fondo que suelen aparecer en la segmentación por umbral se corrigen, ya que el clasificador propuesto segmenta bien el fondo. Por otro lado, los bordes que el algoritmo propuesto no segmenta correctamente, se ven corregidos por la segmentación de umbral propuesta por OpenCV.

En la Figura 21 se pueden ver los resultados que proporcionan ambas técnicas para la misma imagen. A la izquierda, se observa la imagen que ha proporcionado como salida el algoritmo de clasificación propuesto mientras que a la derecha se puede ver la proporcionada por la segmentación de umbral de OpenCV. Por un lado, la imagen que se ha obtenido con clasificador propuesto, tiene un fondo sin manchas y proporciona una buena segmentación de las bacterias aunque genera un efecto de doble borde alrededor de las colonias de bacterias. Sin embargo, la segmentación de umbral con

OpenCV no crea ese efecto de doble borde y es capaz de segmentar las bacterias como el algoritmo propuesto. Por contra, genera unas manchas en el fondo que pueden ser confundidas con bacterias a la hora de utilizar un algoritmo que realice el recuento de dichas bacterias.

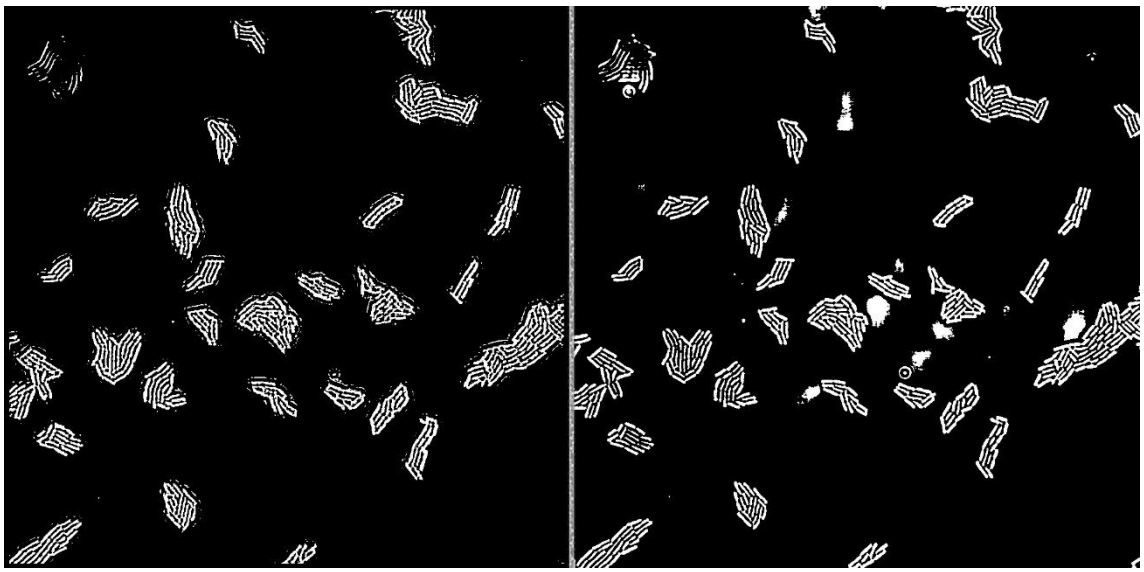


Figura 21: Imagen Segmentada por Clasificador Propuesto (Izquierda) Imagen Segmentada por OpenCV (Derecha)

Aunque ambos métodos de segmentación de las imágenes tienen sus defectos, si se combinan, se consigue obtener una manera de segmentar estas imágenes de manera más fiable y robusta. Un buen ejemplo de esto se puede observar en la Figura 22. En esta imagen se muestra el resultado de combinar las dos imágenes anteriores, las de la Figura 21. Al observar con detenimiento la imagen, se puede comprobar que las manchas del fondo han desaparecido y el efecto de doble borde también. Esta imagen, la resultante de combinar ambas técnicas, se utilizará como imagen de entrada al algoritmo que realizará el recuento de las bacterias.

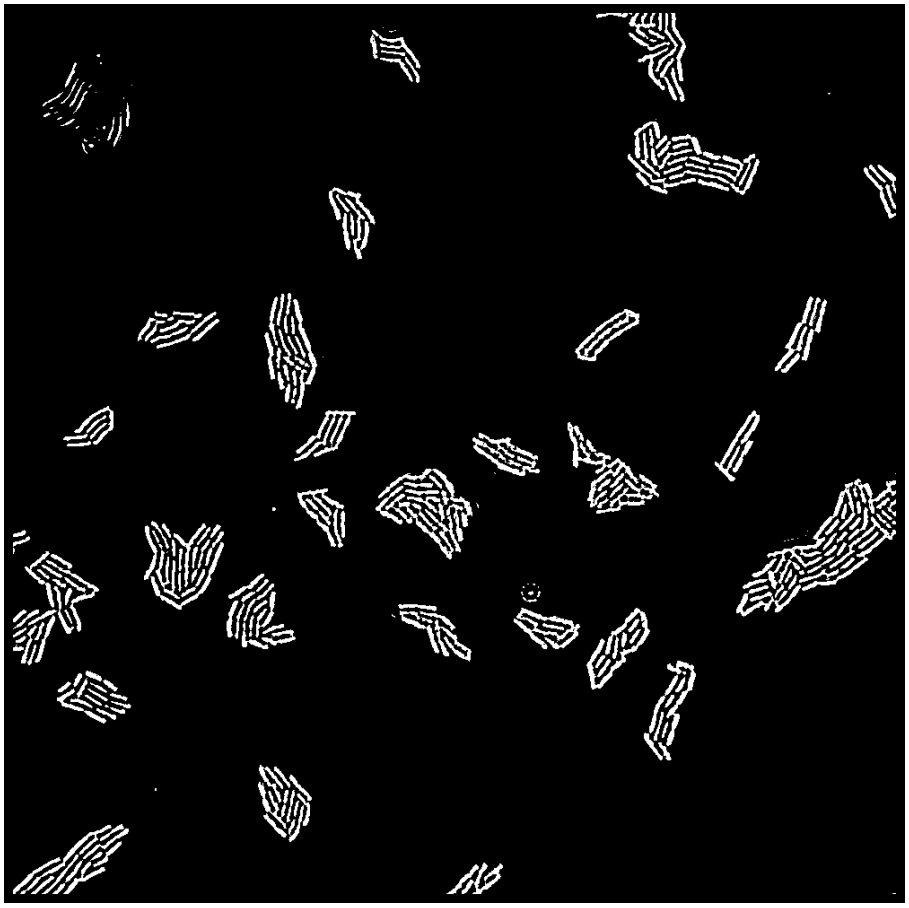


Figura 22: Ejemplo de Segmentación Combinando Clasificador y OpenCV

El algoritmo que se ha utilizado para realizar el recuento y la identificación de las bacterias es un algoritmo que utiliza las técnicas de dilatación y erosión, así como la de región de crecimiento. El algoritmo proporciona como resultado final el número total de bacterias existentes en la imagen así como una imagen de salida donde se puedan observar todas las bacterias identificadas.

La técnica de erosión consiste en reducir los bordes de una imagen segmentada. En este caso, esta técnica lo que hace es disminuir los bordes de las bacterias, es decir, hacerlas más delgadas. Lo que se consigue con esto es que las bacterias que aparecen pegadas entre sí se separen y sea más fácil contarlas.

Por otra parte, la técnica de dilatación es la opuesta a la de erosión. Esto significa que su función es aumentar el tamaño de los bordes de las bacterias. A priori, esto puede parecer poco útil o incluso un inconveniente. Sin embargo, se utiliza para detectar aquellas bacterias que al segmentarlas hayan reducido su tamaño. Al aumentar sus bordes, estas bacterias pueden ser detectadas por el algoritmo. Si esto no se aplicase, el algoritmo las despreciaría debido a su pequeño tamaño.

Además de la erosión y la dilatación, el algoritmo utiliza una técnica de región de crecimiento. Como se explicó en el estado del arte de este proyecto, esta técnica

segmenta imágenes uniendo áreas con tonos de grises iguales o similares. En este caso, ya se dispone de una imagen segmentada por lo que el objetivo es construir una bacteria uniendo las áreas blancas. Si todas las bacterias estuviesen totalmente separadas, es decir, que no hubiese ningún pixel blanco que conectase dos bacterias colindantes, sería muy sencillo identificarlas utilizando este método. Para ello, simplemente, se seleccionaría un pixel blanco y se iría "creciendo" a su alrededor buscando el resto de pixeles blancos adyacentes. Este proceso se repetiría hasta que no quedase ningún pixel blanco por recorrer.

Sin embargo, debido a las características de las imágenes y de las bacterias de las que se dispone, ha sido necesario desarrollar un algoritmo que proporcione el recuento de bacterias de una forma más compleja. A continuación, se detallará el funcionamiento de este algoritmo.

En primer lugar, lo que se va a hacer es evitar trabajar con las imágenes como tales. Por ello, se crea un array de dos dimensiones (matriz) que contiene todos los píxeles de la imagen. Estos pixeles se han codificado de la siguiente forma, si el pixel es negro se almacenará en el array un 1. Si por el contrario el pixel es blanco, en el array se almacenará un 0.

Seguidamente, se creará otro array pero en este caso la codificación que se utilizará será diferente. Este nuevo array contendrá "pesos" que se utilizarán posteriormente para detectar las bacterias. La manera de calcular los pesos es la siguiente: se recorre el array creado anteriormente, de modo que si se encuentra un 1, se pondrá un 9 en el nuevo array. Si lo que se encuentra es un 0, lo que se hará será sumar el número de 1's que tiene alrededor dicho 0. El número de 1's puede oscilar entre 8 y 0. El valor de esta suma, será el que se almacenará en el nuevo array. Por tanto, cuando se termine este paso se tendrán dos arrays, uno con 0's y 1's y otro con números comprendidos entre 0 y 9.

A continuación, se realiza la primera búsqueda de células. Para lo que se utilizará el array de pesos. La búsqueda se lleva a cabo de la siguiente manera, si se encuentra un 0 en el array se crea una célula y se buscan todos los 0's adyacentes. El 0 encontrado se sustituye por un 1 para evitar añadirlo a la célula infinitamente. Este paso se repite recursivamente con los 0's adyacentes hasta que no quede ningún 0. La razón por la que se buscan 0's y no otros números es que al generar el array de pesos, se ha realizado una erosión a las bacterias ya que los bordes de dichas bacterias no serán 0 sino que tendrán un peso ya que tienen píxeles negros alrededor. Sin embargo, pixeles centrales de las bacterias serán 0 ya que están rodeados de más pixeles blancos. Al realizar la búsqueda de bacterias solo teniendo en cuenta los píxeles de valor 0 lo que se está haciendo es erosionar las bacterias para conseguir separarlas.

Es muy importante en este algoritmo no recorrer el mismo pixel varias veces. Es por ello, que se van creando células. Cada célula es una lista de todos los píxeles que la

conforman. A continuación, cada píxel añadido a la célula se cambia de valor de 0 a 1 para evitar volver a recorrerlo.

Dado que existen pequeñas imperfecciones en la segmentación tales como manchas, el siguiente paso consiste en borrar las células pequeñas. Se sabe que las células tienen que tener un tamaño mínimo por lo que se ha calculado un valor aproximado de los píxeles que tienen que contener. Todas las células creadas en el paso anterior que no contengan ese número mínimo de píxeles, se eliminarán ya que realmente no son células sino que son imperfecciones de la segmentación.

Tras la primera búsqueda o primera iteración, se realizará una segunda. En esta segunda búsqueda, se creará una copia del primer array (el que contenía 0's y 1's) y se trabajará con esta copia. Dentro de este array, lo primero que se hará será borrar las células que ya habían sido detectadas en la primera iteración. Esto significa que todas los píxeles que se encuentren en cualquiera de las células creadas, cambiará su valor de 0 a 1 para así evitar ser detectado otra vez. Además, también se eliminarán los bordes de las células detectadas. Esto significa que se aplicará una dilatación a las células ya detectadas en la primera iteración. Cuando se haya terminado este proceso, se procederá a buscar de nuevo células. La búsqueda es igual que la de la primera iteración, es decir, se buscarán 0's y sus 0's adyacentes hasta que no haya más.

A primera vista puede parecer que no existe diferencia alguna entre la primera y la segunda iteración pero lo cierto es que, mientras que en la primera iteración los bordes de las bacterias se habían eliminado utilizando el array con pesos, en esta segunda iteración estos bordes se siguen conservando, lo que permite al algoritmo detectar bacterias más delgadas.

A continuación, se vuelven a eliminar las células pequeñas. En este caso, se buscan células aún más pequeñas que en la primera iteración. De esta forma, se eliminan gran cantidad de falsos positivos.

Pese a los esfuerzos realizados en la primera y en la segunda iteración, existe otro problema: las células demasiado grandes. Es por ello, que se ha realizado una tercera iteración donde se procesarán estas células para determinar si realmente es una sola célula o son dos juntas y en este último caso, donde se encuentra la división de ambas.

Lo primero que se hace es una búsqueda, entre todas las células ya detectadas, de células que sean mayores de un tamaño determinado. Las células que cumplan la condición se eliminarán de la lista de células detectadas y se procesarán.

El procesado de las células grandes tiene como objetivo partir las células grandes en células más pequeñas. Es decir, determinar si una célula muy grande está realmente formada por dos células más pequeñas y, a continuación, ser capaz de saber en qué punto está la división entre ambas células.

Para llevar a cabo esta tarea, se recorrerá la célula grande en busca de estrechamientos. Un estrechamiento se detecta porque tiene menos número de pixeles pertenecientes a la célula alrededor. Si se detecta un estrechamiento, los píxeles que cumplen la condición de que se encuentran en el estrechamiento, serán borrados de la célula grande produciendo así una fractura. Esta fractura servirá para determinar cuántas células formaban realmente la célula grande y añadirlas como nuevas células a la lista de células detectadas.

Tras realizar este procesado de las células grandes, se volverán a eliminar las células demasiado pequeñas. En esta ocasión, se reducirá aún más el número de píxeles a partir del cual una célula es considerada demasiado pequeña, ya que al dividir células grandes es posible que hayan quedado células pequeñas pero que realmente son células.

Finalmente, se realizará una cuarta y última iteración. En esta iteración, lo primero que ser realizará será una copia del array de 0's y 1's para trabajar con él. Seguidamente, se borrarán todas las células ya detectadas. En esta ocasión se realizará una doble dilatación a la hora de borrar las células ya detectadas. Esto significa que además de los píxeles pertenecientes a las células detectadas se eliminarán también los píxeles adyacentes, es decir, los bordes y los píxeles adyacentes de los adyacentes (los bordes de los bordes). Lo que se pretende conseguir con esto es detectar ciertas células que pudiesen estar pegadas a otras y que no se hubiesen visto en las anteriores iteraciones.

A continuación, se vuelve a realizar una búsqueda de células igual que en la segunda iteración y se borran las células pequeñas para evitar falsos positivos.

Por último, para saber cuántas bacterias, o células, existen en la imagen, solo es necesario contar cuántas células se han creado. Además, para proporcionar la imagen de salida, se pintarán todos los píxeles que pertenezcan a cada célula del mismo color y cada célula en un color diferente. De este modo, además de saber cuántas bacterias hay también se sabe su posición.

En el siguiente apartado, el de resultados, se comparará esta técnica de detección con las anteriormente descritas y se explicarán los resultados obtenidos.

5. Resultados

En este capítulo del proyecto, se describirán los resultados obtenidos con cada método utilizado para el recuento y la detección de bacterias. Además, se compararán los resultados proporcionados por cada técnica para explicar cuál ha sido el método finalmente elegido.

5.1 Viola-Jones

El primer algoritmo evaluado como solución al problema planteado fue el Viola-Jones. Como se ha explicado en el apartado del diseño de la solución, este algoritmo se basa en la combinación de una serie de entrenadores débiles para construir un entrenador robusto. Por tanto, es una técnica basada en probabilidades y estadística. Dentro de los algoritmos de tipo Viola-Jones, en este proyecto se utilizaron dos. Cada uno de ellos proporcionó una solución diferente que se comentará a continuación.

5.1.1 Algoritmo Viola-Jones para Detección Facial

En primer lugar se probó un algoritmo utilizado para la detección facial. La idea era cambiar la base de datos de caras por una base de datos de bacterias. Esta base de datos debía contener muestras positivas y negativas para entrenar el algoritmo.

En un primer momento, se pensó que cambiando la base de datos de caras por una base de datos de bacterias se obtendrían unos buenos resultados. Sin embargo, al cambiar esta base de datos, el algoritmo no es capaz de proporcionar una solución. Después de analizar los resultados, se concluyó que la baja textura de las imágenes (especialmente en la base de datos con muestras negativas) dificultaba enormemente la selección de los clasificadores débiles. Al ser todas las muestras muy parecidas, prácticamente todos los clasificadores débiles proporcionaban la misma solución, por lo que el proceso de entrenamiento no podía encontrar una solución razonable en un tiempo reducido.

Además, este algoritmo requiere mucho tiempo de cómputo por lo que resulta poco adecuado para ser implementado en un sistema embebido.

5.1.2 Algoritmo Viola-Jones con Funciones de OpenCV

Debido a los malos resultados proporcionados por el primer algoritmo Viola-Jones y la lentitud del proceso de entrenamiento, se decidió buscar una alternativa basada en el mismo tipo de algoritmos.

En esta ocasión, el algoritmo está diseñado utilizando funciones de OpenCV. Como OpenCV es una librería optimizada para el manejo de imágenes, su utilización resulta altamente satisfactoria en cuanto al tiempo necesario para entrenar el algoritmo. Esto significa, que este algoritmo es mucho más rápido que el anterior y ofrece muchas más alternativas, por lo que podría solucionar el problema planteado.

Sin embargo, y pese a los esfuerzos realizados, esta técnica tampoco obtuvo resultados. Aunque se optimizaron todos los parámetros de las funciones de OpenCV el algoritmo tampoco proporcionó ningún resultado relevante.

La causa de esta falta de resultados reside en la naturaleza del algoritmo. Un algoritmo de tipo Viola-Jones está basado en clasificadores débiles, con un entrenamiento de tipo AdaBoost. El entrenamiento es una técnica estadística, por lo que es sensible al ruido y a situaciones en las que hay gran diversidad de posibles soluciones y todas ellas no están muy correladas.

En esta ocasión, los problemas en el entrenamiento del algoritmo vienen derivado del tipo de imágenes que forman la base de datos de muestras. Estas imágenes, tanto las muestras positivas como las negativas, se obtuvieron del conjunto de imágenes de microscopio disponible. El problema que presentan estas imágenes es que existe una gran similitud entre las muestras negativas y las positivas.

Cuando el algoritmo comienza a buscar y a entrenar la cascada de clasificadores, llega un punto en el que no encuentra suficiente diferencia entre muestras negativas y positivas. Esta ausencia de diferencia está causada por el ruido de las imágenes y su baja textura. Cuando el algoritmo intenta buscar las diferencias entre las muestras positivas y negativas no es capaz de encontrarlas o simplemente, las diferencias encontradas no son suficientes como para poder entrenar el siguiente clasificador débil de la cascada. Esto hace que durante la ejecución del entrenamiento, este se detenga antes de terminar y sin proporcionar parámetros para el clasificador. Como se puede ver en la Figura 23, en la fase 9, el algoritmo ha dado como resultado un 0, indicando que no es capaz de encontrar una solución y abortando el proceso de entrenamiento.

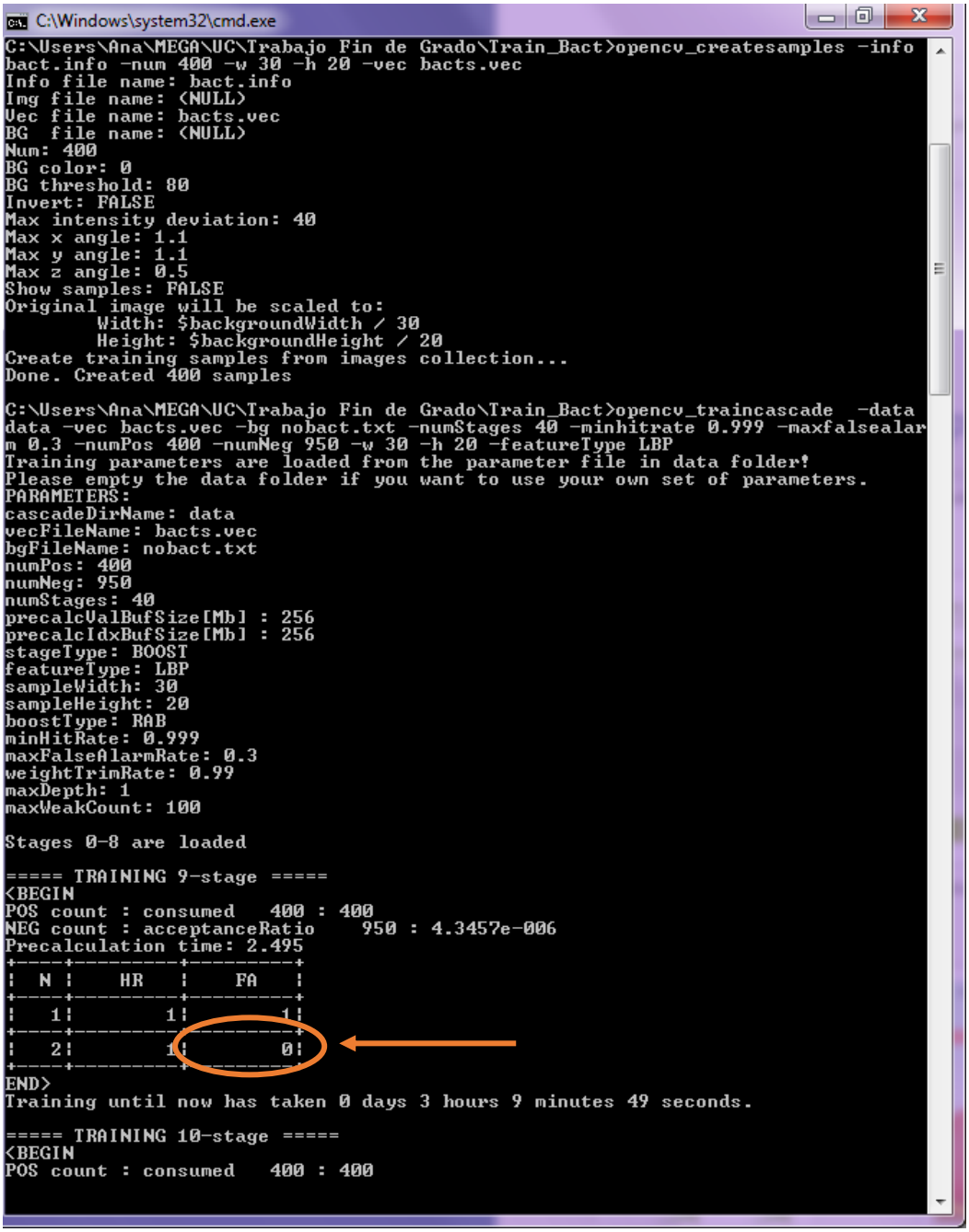


Figura 23: Fallo en la Ejecución del Algoritmo Viola-Jones Basado en OpenCV

En algunas ocasiones, dependiendo del número de imágenes y de los parámetros, este fallo se producía en una fase tardía del proceso de entrenamiento. Para intentar solucionar el problema, se pensó en reducir fases. Con esta solución, se consiguió que el algoritmo fuese capaz de generar un clasificador. El problema, es que este clasificador no era demasiado bueno, por lo que los resultados que proporcionaba no eran relevantes.

Ya que los resultados presentados por este algoritmo no han sido satisfactorios, se desechó la posibilidad de utilizarlo como solución final del proyecto.

5.2 Clasificador Propuesto con Simulated Annealing

Al no lograrse los resultados previstos con el algoritmo de Viola-Jones, se desarrolló una nueva metodología que combinaba un clasificador lineal (basado en SVM) con el algoritmo Simulated Annealing.

Tras entrenar el algoritmo de segmentación, se pudo observar que los resultados que proporcionaba eran bastante aceptables ya que de media solo existía entre un 8% y un 9% de píxeles erróneos en la segmentación.

Utilizando estas imágenes segmentadas, se desarrolló un algoritmo de identificación basado en la técnica de Simulated Annealing. Este algoritmo fue presentado en una sección anterior.

Pese a que a priori parecía una buena técnica para proporcionar una solución al problema, finalmente, los resultados proporcionados no han sido los esperados.

En primer lugar, existen ciertos casos en los que las bacterias virtuales no consiguen posicionarse en el lugar de las bacterias reales. Esta situación se produce cuando hay varias bacterias paralelas y el algoritmo cruza la bacteria virtual sobre dos o tres bacterias reales paralelas, lo que impide que las mismas sean cubiertas por otra bacteria virtual. Como se puede ver en la Figura 24, existen colonias donde el algoritmo prácticamente ha acertado el 100% de bacterias mientras que existen otras zonas donde no ha habido prácticamente acierto. Las bacterias virtuales aparecen en la Figura 24 en color verde mientras que las reales son blancas.

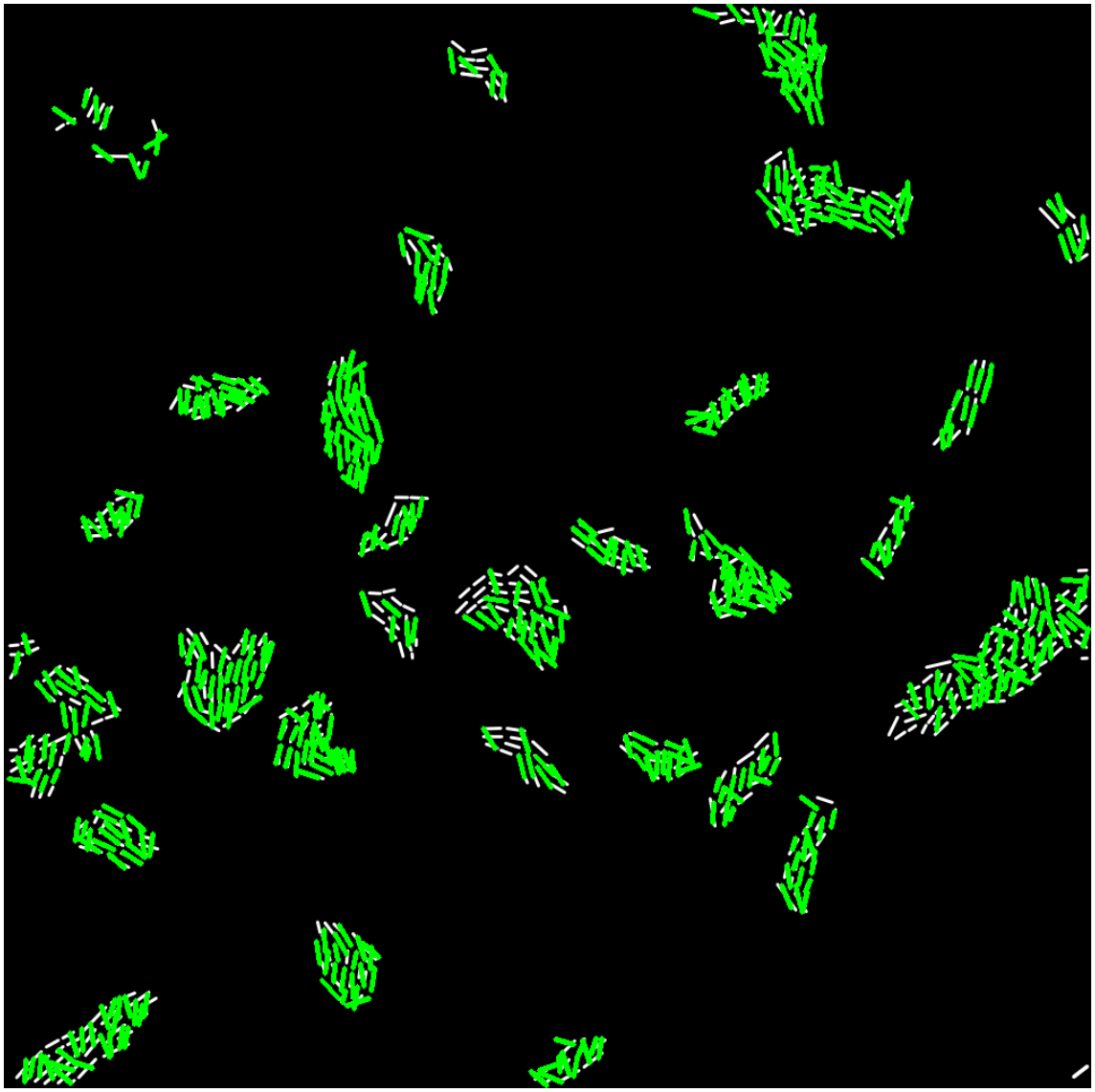


Figura 24: Resultados del Algoritmo Simulated Annealing

Otro hándicap que tiene este algoritmo es el tiempo de ejecución. Como se trata de un algoritmo cuyo funcionamiento se basa en la simulación de un proceso de enfriamiento, se necesitan realizar muchas operaciones y muchos enfriamientos hasta conseguir la solución final. Es decir, es un algoritmo muy lento el cual, en caso de haber funcionado correctamente, habría sido necesario paralelizar antes de introducirlo en un sistema embebido.

Por tanto, ha quedado descartado su uso como solución final a la especificación ya que, aunque se han obtenido resultados, estos no han sido todo lo buenos que se necesitaban.

5.3 Clasificador Propuesto con Algoritmo de Dilatación y Erosión

Como se ha comentado, esta técnica aprovecha el clasificador propuesto para segmentar las imágenes. Además, se refuerza la segmentación combinando los resultados del clasificador propuesto y la función de segmentación por umbral de OpenCV.

El algoritmo de dilatación y erosión desarrollado en este proyecto, necesita que las imágenes de entrada estén bien segmentadas ya que, en caso contrario, los resultados que proporcionará no serán satisfactorios.

Durante el desarrollo del algoritmo se fueron haciendo diferentes modificaciones de modo que cada nueva iteración corrigiese parte de los errores de la iteración anterior. Finalmente, se concluyó que con cuatro iteraciones era suficiente para conseguir unos resultados muy buenos.

Dado que en este proyecto, la imagen ha sido ampliamente procesada antes de ser evaluada por el clasificador y que la segmentación que se ha realizado es de gran calidad, los resultados obtenidos utilizando esta técnica han sido satisfactorios.

Como se puede ver en la Figura 25, la salida del algoritmo proporciona una imagen donde cada bacteria está representada de un color diferente. Esta característica permite que la identificación sea más sencilla y visual.

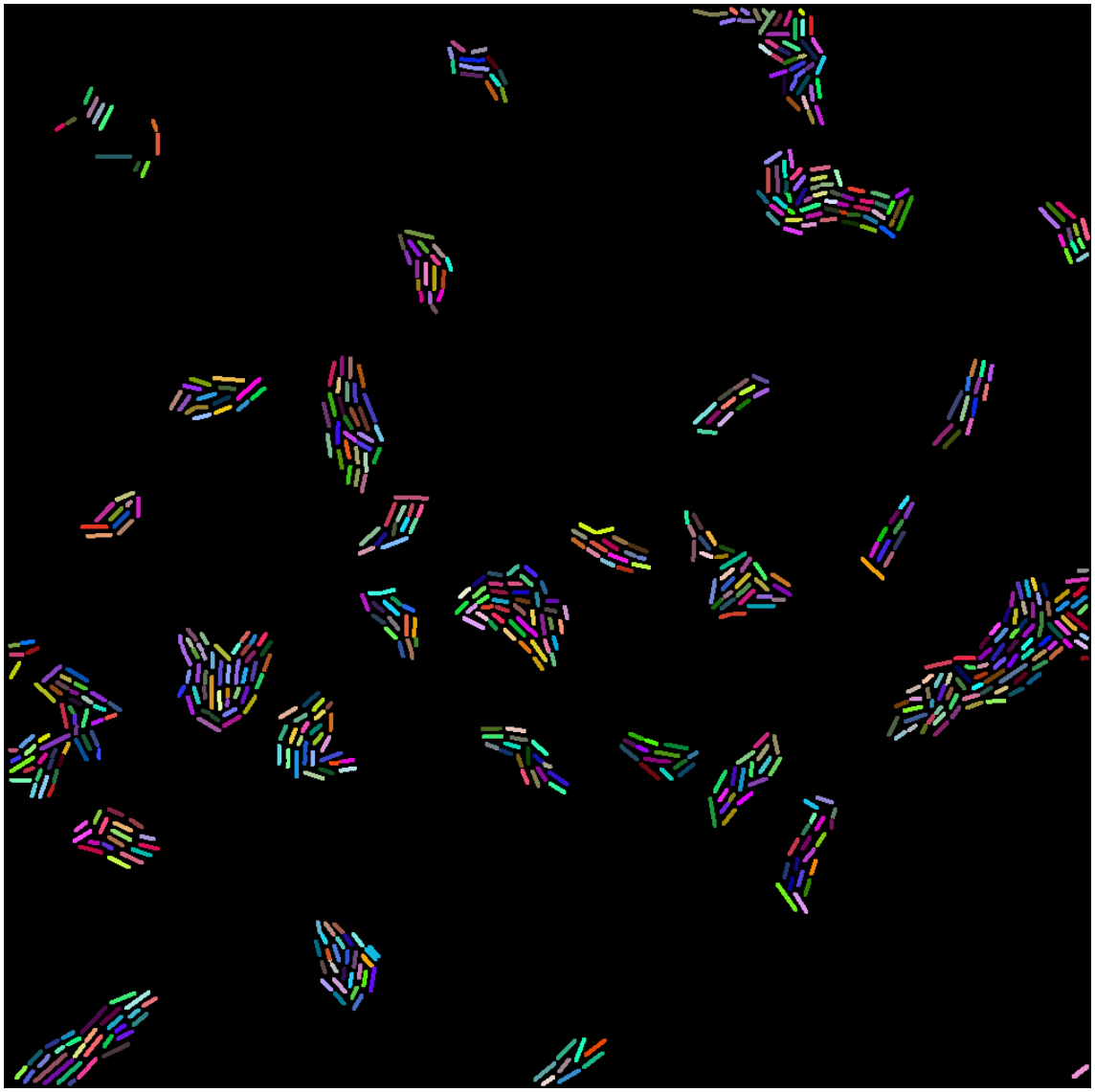


Figura 25: Resultados del Algoritmo de Dilatación y Erosión

Además, durante la ejecución del algoritmo, en la consola de comandos se pueden ir viendo el número de bacterias detectadas en cada fase antes y después de eliminar las bacterias demasiado pequeñas, como se muestra en la Figura 26. Cuando el programa acaba su ejecución, guarda el resultado en ficheros y muestra en la consola el número final de bacterias detectadas.

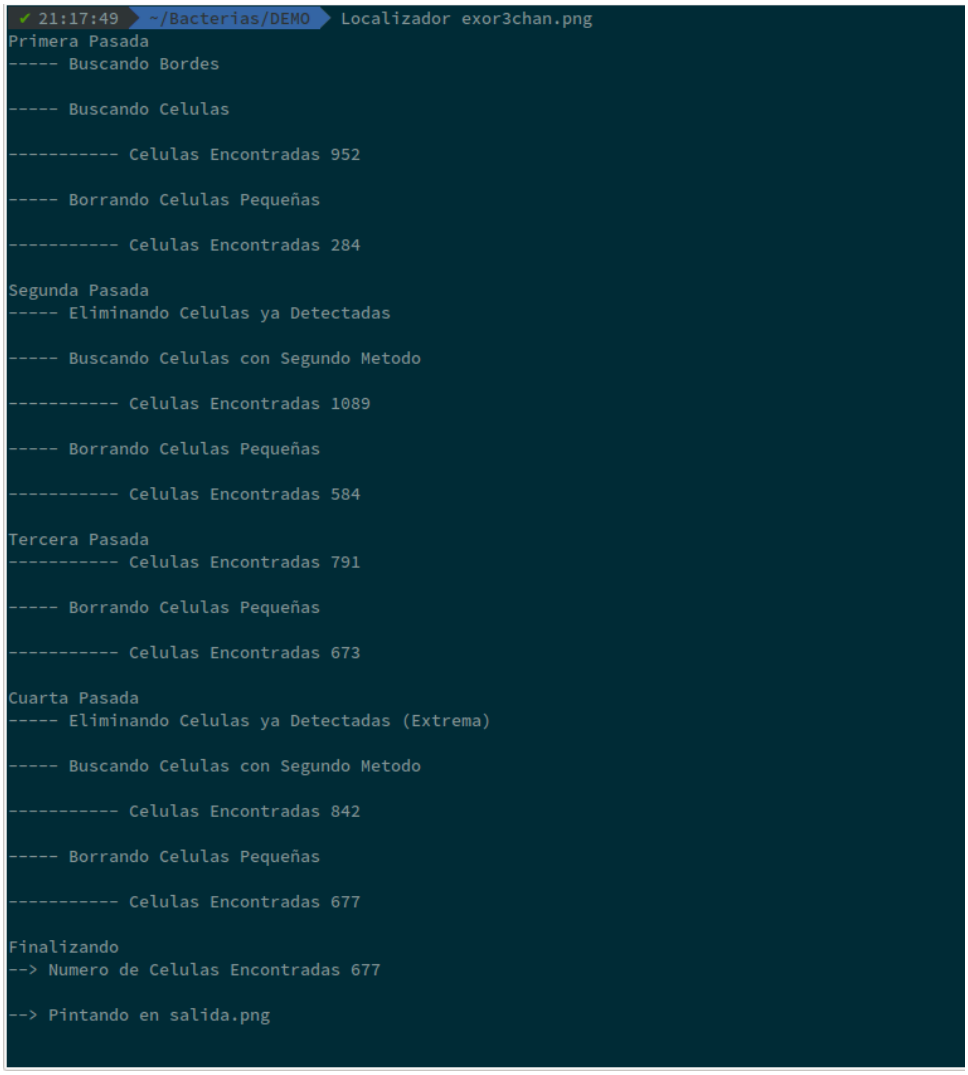


Figura 26: Resultados Mostrados en Consola de Comandos por el Algoritmo de Dilatación y Erosión

En conclusión, se ha decidido utilizar este algoritmo como solución final a la especificación por varios motivos. En primer lugar, de todas las soluciones probadas es la que mejores resultados ha proporcionado. Además, estos resultados obtenidos con el uso del algoritmo de dilatación y erosión, se aproximan bastante a la realidad. Es decir, el número de bacterias final obtenido al ejecutar el código, se aproxima fielmente al número total de bacterias en la imagen.

En segundo lugar, la elección de esta solución por encima de las anteriormente explicadas se basa en la buena calidad de la identificación de las bacterias. Cabe recordar, que en la especificación se pedía que, además de contar las bacterias había que identificarlas.

En tercer lugar, es un algoritmo cuya ejecución es muy rápida por lo que se podría implementar en un sistema embebido sin mayor dificultad. Esto haría que la herramienta fuese portable de modo que solo se necesitaría conectar el sistema embebido al microscopio para proporcionar lecturas del número de bacterias.

Por último, es un algoritmo robusto ya que combina varias técnicas de procesado de imagen, tanto de corrección de histograma, filtrado de ruido, ecualización de histograma, y posteriormente, segmentación. Es importante recordar, que la segmentación utilizada es la combinación de dos técnicas diferentes de segmentación: por un lado la proporcionada por el algoritmo propuesto (basado en un clasificador SVM) y por el otro la que se obtiene utilizando la función de segmentación de umbral de la librería OpenCV. Esta particularidad aporta robustez y fiabilidad al resultado final.

En conclusión, el algoritmo desarrollado en este proyecto ha cumplido ampliamente los requisitos que se especificaban al inicio.

6. Conclusiones

En este proyecto se ha desarrollado una herramienta que permite dar soporte a las actividades de procesado de imagen de microscopio, identificación de ejemplares (bacterias) y recuento de los mismos.

Para llevar a cabo del desarrollo de la herramienta, en primer lugar se ha sido necesario tener en cuenta el tipo de imágenes del que se disponía. Estas imágenes han sido capturadas con un microscopio de contraste de fase, lo que hace que se necesite realizar un procesado previo de las mismas.

La aplicación de las diferentes correcciones a las imágenes ha resultado un paso previo clave para obtener los resultados finales. Esto es debido a que las imágenes que en un principio se tenían, tenían baja luminosidad y contraste. Gracias a la corrección del histograma, se obtuvieron imágenes mucho más claras y luminosas. Posteriormente, se eliminó parte del ruido de fondo, causado principalmente por la cámara, preparación de las bacterias y óptica del microscopio. Finalmente, ser realizó una ecualización del histograma, lo que proporcionó un mayor contraste en las imágenes.

Además se han combinado dos técnicas diferentes de segmentación, una basada en clasificadores de SVM y la otra basada en las funciones de segmentación por umbral de OpenCV. La primera técnica (clasificador propuesto) permite diferenciar claramente las bacterias del fondo. La utilización de las funciones de segmentación por umbral de OpenCV han permitido conseguir una mejor definición de las bacterias y sus bordes. Al combinar ambas técnicas se consigue una segmentación de muy buena calidad, que contribuye a que el recuento de bacterias sea correcto.

Finalmente, se le ha aplicado a la imagen el algoritmo de dilatación y erosión para conseguir identificar y contar las bacterias de las imágenes. Este algoritmo, además de proporcionar muy buenos resultados, tiene un bajo tiempo de ejecución. Por otra parte, la complejidad computacional del algoritmo no es muy alta, por lo que puede ser implementado con facilidad en cualquier sistema embebido.

Por tanto, se puede decir que el resultado final obtenido cumple la especificación requerida y realiza el trabajo de recuento e identificación de bacterias de manera rápida, robusta y fiable. El error de segmentación es menor del 8% y los tiempos de ejecución son del orden de minutos.

Como conclusión final, en este proyecto se ha conseguido desarrollar un sistema que realice el procesado de imágenes tomadas con un microscopio de contraste de fase y que proporcione soporte a las tareas de recuento e identificación de bacterias

cumpliendo la especificación dada. Este sistema puede ser implementado en un sistema embebido para evitar el uso de un PC y así obtener un sistema portable.

Por último, este proyecto ha abierto nuevas líneas de investigación, basadas en el reconocimiento de bacterias y otros patrones en imágenes de microscopía, además de una vía de colaboración con el IBBTEC.

7. Referencias

[1] https://en.wikipedia.org

[2] http://www.phiab.se/technology/phase-contrast-microscopy

[3] O. Sliusarenko, J. Heinritz, T. Emonet and C. Jacobs-Wagner, "High-throughput, subpixel precision analysis of bacterial morphogenesis and intracellular spatio-temporal dynamics", *Molecular Microbiology*, vol. 80, no. 3, pp. 612-627, 2011.

[4] P. Azad, T. Gockel and R. Dillmann, *Computer vision*. [Netherlands]: Elektor International Media, 2008.

[5] http://www.ipol.im/pub/art/2011/bcm_nlm/

[6] http://g2pi.tsc.uc3m.es/es/Boosting-es

[7] http://www.cs.upc.edu/~mabad/IA/SIMULATED%20ANNEALING.pdf

[8] Y. Wang, "An Analysis of the Viola-Jones Face Detection Algorithm", *Image Processing On Line*, vol. 4, pp. 128-148, 2014.

[9] https://abhishek4273.com/2014/03/16/traincascade-and-car-detection-using-opencv

[10] Annamraju, Abhishek Kumar, and Akash Deep Singh. "Analysis and optimization of parameters used in training a cascade classifier." *Advances in Image and Video Processing* 3.2 (2015): 25.