UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Desarrollo de una aplicación forense para la clasificación de gránulos de pólvora

Autor: Noelia Campos Puertas

Tutor: José Luis Martín Sánchez

Cotutor: Fernando E. Ortega Ojeda

TRIBUNAL:

Presidente: Sira Elena Palazuelos Cagigas

Vocal 1º: José Luis Martín Sánchez

Vocal 2º: Jesús Ureña Ureña

CALIFICACIÓN:

FECHA

ESCUELA POLITECNICA SUPERIOR

Índice

Índice de Figuras 3
Índice de Códigos 5
I. Resumen 6
II. Abstract 7
III. Palabras Clave 8
IV. Glosario de Acrónimos 9
V. Resumen Extendido 10
VI. Memoria 12
1. Introducción 12
2. Estado del Arte 15
2.1. Visión Artificial 15
2.2. Deep Learning. Redes Neuronales Convolucionales 16
2.3. Algoritmo Bag of Features 28
2.4. Máquina de Soporte Vectorial 31
2.5. Interfaz Gráfica de Usuario 34
2.6 Python 36
3. Descripción del Sistema 37
3.1 Base de Datos 37
3.2 Arquitectura 39
3.3 Etapa de Pre-procesamiento 41
3.4 Etapa de Clasificación. AlexNet 67
3.5 Etapa de Adaptación de la Red AlexNet 70
3.6 Etapa de Clasificación. MyNet 85
3.7 Etapa de Extracción de Características 87
3.8 Etapa de Clasificación. SVM 91
3.9 Etapa de Testeo y Predicción 94
3.10 Interfaz Gráfica de Usuario 98
3.11 Desarrollo del Sistema Sobre Python 103

4. Evaluación 115
4.1 Resultados 115
4.2 Mejor Resultado 130
4.3 Conclusiones 133
VII. Presupuesto 134
VIII. Pliego de Condiciones 136
IX. Requerimientos en la Adquisición 137
IX.I Método 1 139
IX.II Método 2 140
IX.III Método 3 141
X. Manual de Usuario 142
XI. Bibliografía 146
XII. Bibliografía Auxiliar 148

Índice de Figuras

Figura 1. Red Neuronal Feed Forward Simple 17
Figura 2.Ejemplos de Funciones de Activación.....18
Figura 3. Arquitectura Red AlexNet.....21
Figura 4. Proceso Realizado en la Capa de Convolución 22
Figura 5. Operación de Pooling 25
Figura 6. Capa Totalmente Conectada.....26
Figura 7. Operación de Softmax 27
Figura 8. Proceso de Extracción y Representación en Espacio Multidimensional de Puntos de Interés 29
Figura 9. Hiperplano Óptimo para SVM con 2 Características 31
Figura 10. Suavizado del Margen con Variables de Holgura 32
Figura 11. Caso de un Conjunto de Datos no Linealmente Separable.....33
Figura 12. Ubicación de GUIDE 34
Figura 13. Clases de Gránulos Tratados 38
Figura 14. Arquitectura del Sistema 40
Figura 15. Relación de Aspecto. Imagen Original (dcha), Imagen Tratada (izda) 42
Figura 16. Relación de Tamaño. Imagen Original (superior), Imagen Tratada (inferior).....42
Figura 17. Segmentación del Gránulo con la Función Imadd.....47
Figura 18. Segmentación del Gránulo con Perdida de Información 48
Figura 19.Gradiente para la Detección de Bordes 50
Figura 20. Obtención de Bordes con los Operadores Descritos.....52
Figura 21. Histograma con los Niveles de Gris de los Pixeles de la Imagen.....55
Figura 22. Imágenes Binarizadas Bajo Umbral Establecido.....56
Figura 23. Ejemplo Operación de Dilatación.....57
Figura 24. Ejemplo Operación de Erosión.....59
Figura 25. Ejemplo Operación de Etiquetado de Objetos.....60
Figura 26. Segmentación del Gránulo.....63
Figura 27. Gránulo Escala RGB Recuperado 63
Figura 28. Ángulo de Orientación Devuelto por la Función.....64
Figura 29. Objeto en Posición Original BW y Objeto Final en Escala RGB.....65
Figura 30. Modificaciones sobre las Capas Totalmente Conectada de AlexNet 72
Figura 31. Pérdida vs Pesos 77
Figura 32. Efectos de una Tasa de Aprendizaje demasiado Grande 78
Figura 33. Sobre-entrenamiento vs Buena Capacidad de Generalización 80
Figura 34. División del Conjunto de Datos.....81
Figura 35. Evolución del Error cuando se produce Sobre Entrenamiento 81
Figura 36. Perdida durante el Entrenamiento de la Red.....83
Figura 37. Espacio de Color L*A*B.....90
Figura 38. Ventana de la Aplicación.....98
Figura 39. Tabla de Confusión.....99

Figura 40. Ventana de la Aplicación....113
Figura 41. estructuras del Sistema 139
Figura 42. Estructura del sistema....140
Figura 43. Componentes para Sistemas de Adquisición de Imágenes....141
Figura 44. Interfaz Gráfica....142
Figura 45. Ventana con la Matriz de Confusión 143
Figura 46. Cuadro de Diálogo y Mensaje de Error 143
Figura 47. Visualizado de la Información de la Imagen Seleccionada 144
Figura 48. Visualizado de la Etiqueta de Predicción 144
Figura 49. Acción de Cierre de la Ventana....145

Índice de Códigos

Código 1. Rellenado de Imágenes.....45
Código 2. Cantidad de Objetos y Tamaño de los mismos 61
Código 3. Fragmento de Código Final para Supresión de Objetos.....62
Código 4. Predicciones de AlexNet para Nuestros datos de Text.....68
Código 5. Capas que Componen la Nueva Red 75
Código 6.Opciones de Entrenamiento y Nueva Red.....84
Código 7. Predicciones de AlexNet para Nuestros datos de Text.....85
Código 8. Características extraídas para el Set de Entrenamiento y de Test 88
Código 9. Entrenamiento del Clasificador SVM para CNN 92
Código 10. Entrenamiento del Clasificador SVM para BOF.....92
Código 11. Matriz de Confusión.....95
Código 12. Almacenamiento de Imágenes Clasificadas de Forma Incorrecta.....97
Código 13. Proceso Seguido en la Operación de Selección de Nueva Imagen.....100
Código 14. Cuadro de Diálogo para Cerrar la APP 101
Código 15. Carga de Archivos con Extensión .mat 106
Código 16. Carga de las Matrices de Pesos y Sesgos 106
Código 17. Sesión Tensorflow.....107
Código 18. Tabla de Resultados Normalizados 110
Código 19. Ventana Principal de la GUI 111
Código 20. Botón para cerrar una ventana.....112

I. Resumen

En la actualidad existen varios campos de estudio y trabajo que utilizan la visión artificial como motor de desarrollo, haciendo que esta disciplina científica se encuentre en continuo auge.

La aplicación de métodos de aprendizaje automático de características en imágenes puede permitir la construcción de clasificadores de elevada precisión. La finalidad de este proyecto se centra en realizar un sistema en el que se permita diferenciar distintas clases de gránulos de pólvora, de forma que se pueda extraer información útil para los cuerpos de seguridad en casos de delitos reales.

II. Abstract

Nowadays, machine vision is being used in several research and work areas. Which makes it a scientific discipline in continuous growth.

The study of automatic learning methods may allow the development of highly accurate images classifiers. The objective of this project is to create a system, which shall distinguish different kinds of gunpowder granules based on their specific characteristics, so it could be useful to law enforcement agencies for solving actual criminal cases.

III. Palabras Clave

AlexNet

Bag of Features

Máquinas de Vectores de Soporte

Matlab

Python

Red Neuronal Convolucional

IV. Glosario de Acrónimos

AI – Artificial Intelligence → Inteligencia Artificial

CCD – Charge Coupled Device → Dispositivo de Carga Acoplada

CMOS – Complementary Metal Oxide Semiconductor → Semiconductor Complementario de Óxido Metálico

CNN – Convolutional Neural Network → Red Neuronal Convolucional

GPU – Graphics Processing Unit → Unidad de Procesamiento Gráfico

GUI – Graphical User Interface → Interfaz Gráfica de Usuario

SGD – Sthochastic Gradient Descent → Descenso de Gradiente Estocástico

SIFT – Scale Invariant Features Transforms

SURF – Speeded Up Robust Features

SVM – Support Vector Machines → Máquinas de Soporte Vectorial

V. Resumen Extendido

Este trabajo se presenta con el objetivo de diseñar un sistema que sea capaz de reconocer diferentes tipos de gránulos de pólvora. Para ello, se implementará un algoritmo utilizando el entorno de desarrollo de Matlab, de forma que, a través de una imagen de entrada de una de estas muestras de pólvora, el sistema sea capaz de detectar de forma automática a qué clase pertenece dicha muestra.

Es importante tener en cuenta que el conjunto de imágenes del trabajo va a presentar variaciones de posición, iluminación, nitidez y tamaño dentro de cada uno de los tipos que la componen. Por lo que se ofrecerán una serie de recomendaciones para la captación de nuevas imágenes de manera que faciliten la futura extracción de características sobre las mismas.

El proceso seguido estará compuesto por las siguientes fases de trabajo:

1. El Preprocesamiento del Conjunto de Datos, formado por una serie de operaciones que mejoren la calidad de la imagen y ayuden a realizar una extracción adecuada de las características que presenta cada tipo de gránulo.

2. La Extracción de Características, que se realizará por medio de dos sistemas distintos. El primero se corresponderá con una red neuronal convolucional, concretamente la red AlexNet preentrenada, que será optimizada para el caso de estudio de este trabajo. El segundo se basará en el sistema Bag of features también conocido como Bag of words o bolsa de palabras.

3. La Clasificación, en la cual, una vez se haya obtenido el conjunto de características apropiado para todas las clases de gránulos, se diseñará un clasificador basado en máquinas de soporte vectorial (SVM). Este clasificador será entrenado y evaluado para los datos obtenidos de los extractores mencionados anteriormente.

4. La Evaluación del Sistema, en la que se llevará a cabo una exhaustiva serie de pruebas sobre los algoritmos diseñados. En función de los resultados alcanzados, se escogerá aquel caso que presente un mejor rendimiento para nuestro conjunto de imágenes.

   Por otro lado, se posibilitará la opción de almacenar todos aquellos casos en los que las muestras sean clasificadas de forma incorrecta.

5. El Diseño de una Interfaz Gráfica de Usuario, mediante la cual el sistema podrá ser evaluado con nuevas imágenes mostrando la clase a la que pertenece la nueva muestra introducida. Además, se incluirá cierta información al usuario a través de una muestra ilustrativa de cada tipo de gránulo, la cantidad de imágenes que compone cada clase y la precisión que ofrece el sistema en cada caso.

6. El Desarrollo del Trabajo en Python. Con el fin de dar más accesibilidad al conjunto de tareas desarrolladas inicialmente en Matlab, este será traducido al lenguaje de programación de Python.

VI. Memoria

1. Introducción

La visión artificial es la disciplina que abarca métodos de adquisición, procesado y análisis para el tratamiento de imágenes digitales. Este es uno de los campos más importantes y de mayor auge de los últimos tiempos dentro de la AI. Hoy por hoy, se recurre a sistemas de visión artificial en diversos ámbitos de trabajo, como, por ejemplo, dentro de la industria, la seguridad, las redes sociales o el propio hogar, haciéndolo muy atractivo para un sin fin de aplicaciones.

El procesamiento o tratamiento digital de imágenes consiste en aplicar procesos algorítmicos que transformen una imagen en otra donde se resalta cierta información de interés, y/o se atenúan o eliminan detalles irrelevantes para la aplicación desarrollada. Así, las tareas del procesamiento de imágenes comprenden la supresión de ruidos, mejoras del contraste, eliminación de efectos no deseados en la etapa de captura, transformaciones de color, etc.

La complejidad de la tarea que se desea realizar será determinante para establecer el número de procesos necesarios para resolver el problema, siendo los más habituales:

1. La adquisición de la imagen, puede ser un paso muy importante a tener en cuenta puesto que, dependiendo de cómo se lleve a cabo, las dificultades con las que habrá que enfrentarse en etapas posteriores pueden variar significativamente. En este punto influirán parámetros tales como el tipo de iluminación, la óptica de la cámara, el entorno de captación, etc.

2. La mejora o restauración de la imagen, procesos en los que se hará un manipulado de la misma para resaltar ciertos detalles mejorando lo que se tenía anteriormente:

Procesado del color, modificaciones que en determinado espacio de color puedan resaltar información de relevancia.

I. Operaciones morfológicas, herramientas para extraer componentes de la imagen, tales como descriptores de forma.

II. Segmentación, de forma que se pueda extraer o aislar algunos de los objetos representados en la imagen.

3. La representación y descripción, etapa en la cual se recibe una imagen segmentada para extraer atributos que resulten claves a la hora de diferenciar distintos tipos de objetos.

4. El reconocimiento, donde se asignará una etiqueta a un objeto basándose en sus descriptores.

Para implementar todas y cada una de las técnicas mencionadas se puede hacer uso de algún software que posea herramientas especializadas en este campo de trabajo, siendo uno de los más populares, la herramienta de software matemático Matlab.

En el presente proyecto, se realizará un sistema de clasificación en el que se puedan diferenciar distintos tipos de gránulos de pólvora que serán extraídos por parte de los cuerpos de seguridad nacional de escenas de crímenes reales, de manera que estos puedan obtener información específica acerca de posibles fechas y lugares en los que se suministraron dichos objetos e, incluso, información de la persona que los adquirió.

Para ello, se partirá de un conjunto de datos compuesto por imágenes representativas de dichos gránulos, y por medio del entorno de programación de Matlab se realizará un sistema de clasificación que posteriormente será traducido al lenguaje de código abierto Python, de manera que presente una mayor accesibilidad al usuario.

2. Estado del Arte

2.1. Visión Artificial

La visión artificial surge en la década de los 60 con la idea de conectar una cámara de video a un computador. Esto supuso la posibilidad de comprender lo que las imágenes captadas por la misma representaban.

Las técnicas utilizadas en visión artificial se han desarrollado a gran velocidad en las últimas décadas. Los primeros sistemas se basaron en imágenes binarias que se procesaban en bloques, ventanas o píxeles. Gracias al desarrollo de nuevos algoritmos, se consiguió reconocer el contorno de los objetos y su posición dentro de la imagen con la limitación de mantener fijo el tipo de iluminación.

Posteriormente, se introdujeron sistemas que trabajaban con intensidades de gris, de manera que se originó la posibilidad de tratar con distintos tipos de iluminación, localizando los bordes de los objetos a través de los cambios de intensidad de los píxeles que los formaban.

Actualmente, este subcampo de la AI comprende tanto la obtención como la caracterización e interpretación de los objetos contenidos en una imagen, siendo uno de los elementos escogidos para incrementar la autonomía en el mundo de la robótica, como parte de algoritmos utilizados en reconocimiento o detección de patrones en aplicaciones y redes sociales, etc.

2.2. Deep Learning. Redes Neuronales Convolucionales

2.2.1 Red Neuronal

Dentro del campo de la AI, el Deep Learning es una técnica de aprendizaje automático mediante la cual es posible que un ordenador aprenda a realizar tareas de clasificación de imágenes, texto o sonido, de la misma forma que lo hacemos las personas.

Algunos métodos de aprendizaje automático emplean arquitecturas de redes neuronales que pueden contener una gran cantidad de capas ocultas, razón por la cual estas se conocen como redes neuronales profundas.

Estos modelos de Deep Learning se entrenan haciendo uso de extensos conjuntos de datos etiquetados y arquitecturas de redes neuronales que aprenden a partir de dichos datos. Considerando el conjunto de datos etiquetados $\{(x_n, y_n)\}$ donde x_n representa las características de los objetos e y_n las etiquetas del ejemplo de entrenamiento n-ésimo, las redes neuronales proporcionan una forma de representar una función compleja no lineal de la variable de entrada $h_w(x)$. Esta función esta parametrizada por una matriz de pesos que tienen asociados cada una de las entradas w_n, pudiendo tomar cualquier valor positivo o negativo.

Un ejemplo de una red neuronal simple como la mostrada en la Figura 1 consta de tres capas, siendo estas: una primera capa con x neuronas de entrada, una capa oculta y una de salida tal que $y = h_w(x_{11}, x_{12}, x_{13} \dots x_{1n})$.

Como se puede apreciar, las neuronas se enlazan entre capas adyacentes de forma que todas quedan interconectadas, conformando así lo que se denomina arquitectura de la red, siendo posibles distintas configuraciones de la misma, en función del número de capas o del sentido del avance de la información.

Una neurona calcula primero una combinación lineal ponderada de sus entradas, tal que:

$$a_j = \sum_{i=1}^{n} w_{ji} x_i + w_0 \qquad [1]$$

Donde $\sum_{i=1}^{n} w_{ji} x_i$ es la suma de todos los valores de entrada a la neurona multiplicados por sus correspondientes pesos, mientras que w_0 se corresponderá con un offset o polarización que se debe sumar al conjunto.

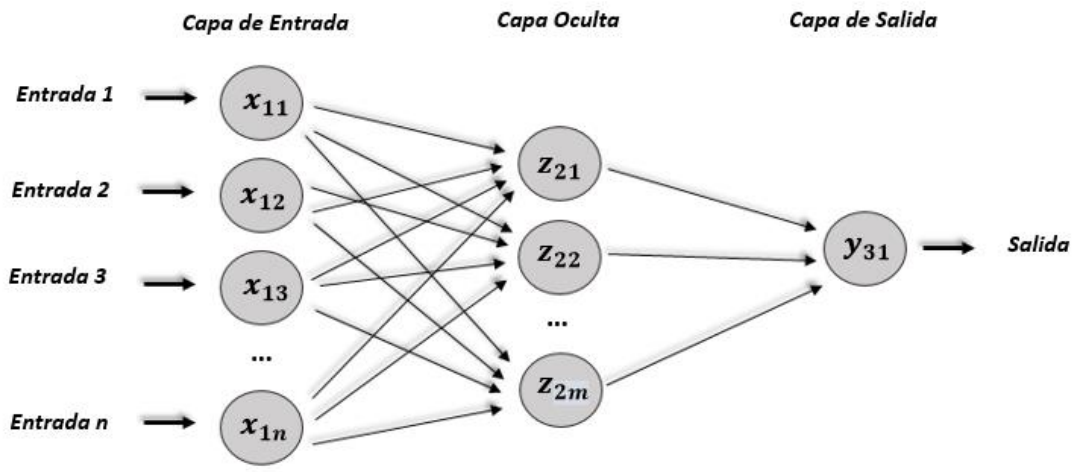


FIGURA 1. RED NEURONAL FEED FORWARD SIMPLE

Como la activación de cada neurona solo depende de los valores de las capas anteriores, estas se van a calcular aplicando sobre a_j su función de transferencia, también llamada función de activación, definida como: $z_j = h(a_j)$. Algunas funciones de activación típicas podrían ser las mostradas en la Figura 2.

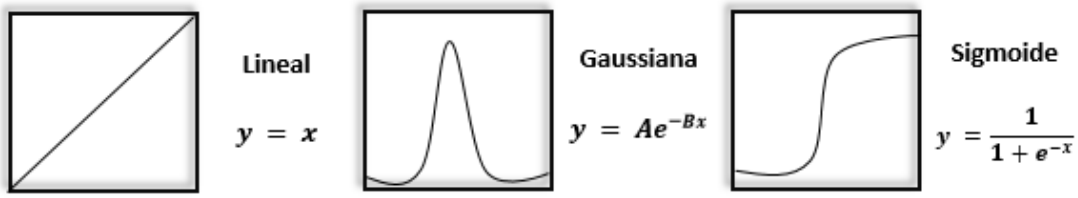


FIGURA 2.EJEMPLOS DE FUNCIONES DE ACTIVACIÓN

De forma general, el proceso mediante el cual la información se propagará a través de las distintas capas de la red tiene el objetivo de aprender y ajustar los parámetros $\boldsymbol{W}$ a sus valores óptimos para un determinado conjunto de datos tratando de minimizar una función de pérdida y obteniendo así un conjunto de salidas:

$$\boldsymbol{y} = \boldsymbol{h_W}(\boldsymbol{x}).$$

2.2.2 Red Neuronal Convolucional

Una red neuronal convolucional (CNN) puede definirse como una red neuronal multicapa jerárquica, puesto que las CNNs se obtienen apilando múltiples capas de características. Estas son similares a las redes neuronales simples, descritas en el punto anterior, y vienen a solucionar los problemas de escalado que surgen cuando se desea trabajar con imágenes de elevada definición.

Un ejemplo de este problema podría partir del uso de una red neuronal simple donde las imágenes a clasificar tengan dimensiones 28x28x1 píxeles, de forma que para una sola neurona completamente conectada en una primera capa oculta supondría un total de 784 pesos. En el momento que se desee trabajar con imágenes más grandes o en espacios de color, como RGB, daría lugar a neuronas con una cantidad de pesos muy elevada, lo cual podría conducir a sobreajustes y desperdicio de recursos.

Las redes neuronales convolucionales trabajan modelando de forma consecutiva pequeñas piezas de información para más tarde combinar dicha información en capas profundas de la misma dando lugar a patrones más simples. En general, estarán constituidas por tres capas específicas para este tipo de redes: una capa convolucional que le da su nombre, una capa de reducción o pooling, con la cual se reducirá la cantidad de parámetros quedándose con las características más comunes, y una capa de clasificación totalmente conectada, con la que se obtendrá el resultado final de la red.

Matlab, proporciona una Toolbox que recibe el nombre de MatConvNet con la que se puede implementar una gran variedad de CNNs para aplicaciones de visión por computadora, además de diversas CNNs pre-entrenadas para la clasificación de imágenes, reconocimiento facial, detección de texto, etc.

2.2.3 CNN Pre-entrenada. AlexNet

El entrenamiento de un modelo de aprendizaje profundo desde cero implica el uso de redes con decenas de capas y de miles a millones de observaciones o imágenes, lo que se traduce en la necesidad de emplear potentes herramientas de cálculo que pueden llevar desde días hasta semanas en completar dicho proceso, existiendo la posibilidad de agilizarlo si se emplea una unidad de procesamiento gráfico (GPU).

El modelo del cual se va a partir para este caso de estudio concreto está entrenado en un subconjunto de la base de datos ImageNet, que utiliza más de un millón de imágenes que puede clasificar hasta en mil categorías de objetos distintos.

La red consta de veinticinco capas en total entre las cuales cabe destacar la existencia de cinco capas convolucionales y tres capas totalmente conectadas. El esquema que sigue su arquitectura se corresponde con el mostrado en la Figura 3.

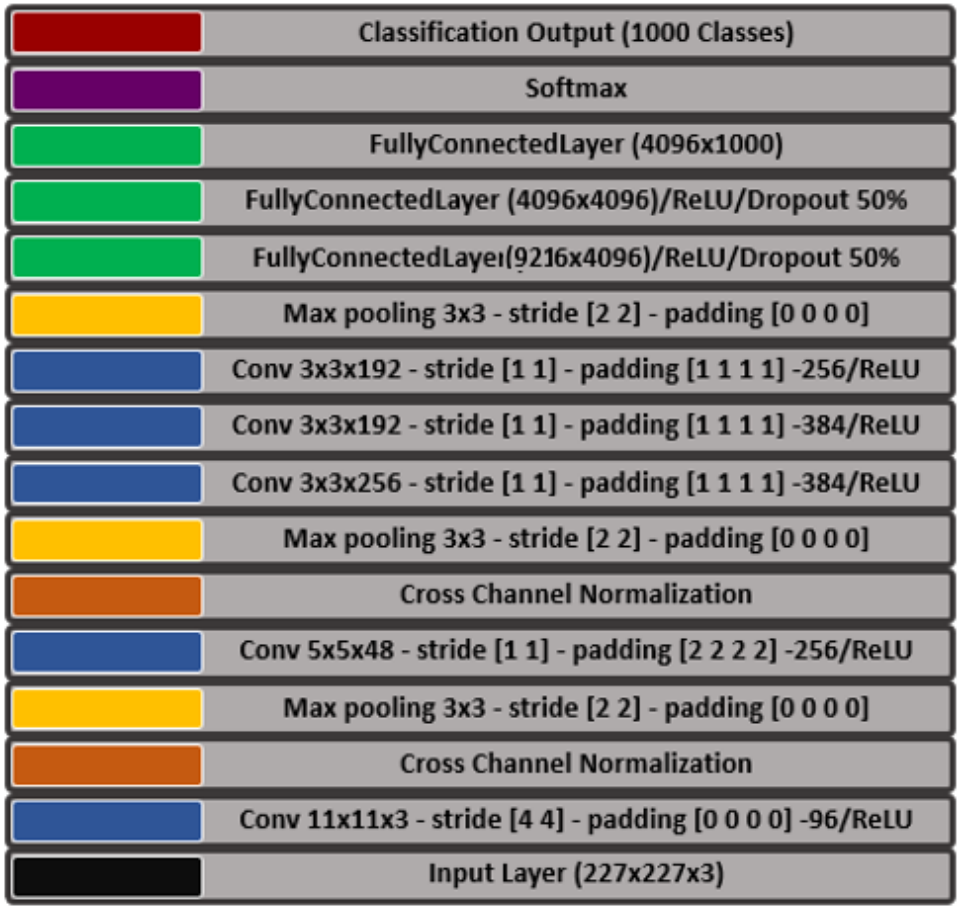


FIGURA 3. ARQUITECTURA RED ALEXNET

A continuación, se proporcionará una descripción detallada de cada una de las capas que forman esta red.

2.2.3.1 ***Input Layer***

En primer lugar, se localiza la capa de entrada, que va a definir el tamaño de las imágenes que se deseen utilizar en la red. Dicho tamaño se corresponderá con la altura, la anchura y la cantidad de canales de color de que disponga la imagen. En este caso concreto, se ha diseñado para imágenes de 227x227 en la escala de color RGB.

2.2.3.2 ***Convolution Layer***

Como se expuso anteriormente, lo que distingue a las redes neuronales convolucionales de cualquier otra red es utilizar la convolución en algunas de sus capas. Esta recibe como entrada la imagen y aplica sobre ella un filtro o kernel que devuelve un mapa de las características de la imagen original, logrando así reducir la cantidad de parámetros que se deben aprender en las siguientes capas.

Este tipo de capas constan de neuronas que se conectan a pequeñas regiones o filtros de la entrada, tal y como muestra la Figura 4. Dicho filtro se mueve a lo largo de toda la imagen, y, en cada caso, se calcula el producto de los pesos por las entradas al que posteriormente se le añade un offset o sesgo como se explicó en el apartado ***2.2.1 Red Neuronal.***

Es posible escoger el tamaño de paso con el que el filtro se va a mover por la imagen, especificándose con la propiedad *Stride*, disponible en la función *convolution2dLayer* con la cual se crean este tipo de capas. Por ejemplo, si el valor escogido es 1, saltará de píxel en píxel hasta recorrer toda la imagen. Por otro lado, se puede utilizar la propiedad *PaddingSize*, que representa el tamaño del relleno para aplicar a los bordes de entrada, de forma que, si se decide fijar en 1, se añadirá a cada uno de los bordes de la imagen una fila más de píxeles de relleno.

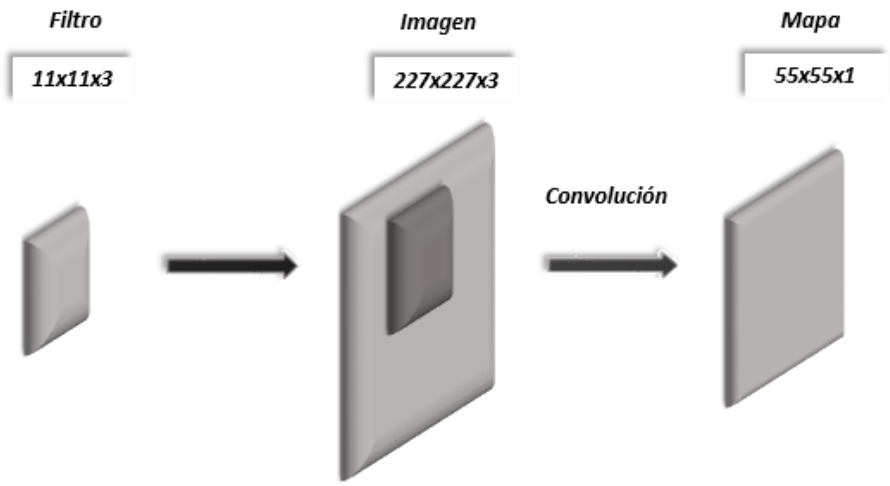


FIGURA 4. PROCESO REALIZADO EN LA CAPA DE CONVOLUCIÓN

La cantidad de filtros utilizados es un parámetro que necesariamente se debe configurar en la capa, para ello, se proporciona la propiedad *NumFilters***,** y va a determinar el número de mapas de características. Generalmente se dispondrá de múltiples mapas, cada uno con un conjunto diferente de pesos y un sesgo.

Muchas CNNs aprenden a detectar características tales como color y bordes en su primera capa de convolución. Cuanto más profundas sean las capas convolucionales que conforman la red, más complejas serán las características que la red aprende a detectar.

2.2.3.3 ***ReLU Layer***

Las capas de convolución siempre van seguidas de una capa ReLU que realiza una operación de umbralizado para cada elemento, de forma que cualquier valor de entrada menor que cero se iguala a cero, tal que:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad [2]$$

Este tipo de capa no modifica el tamaño de su entrada y se crea utilizando la función *reluLayer*. Existiría la posibilidad de utilizarla con algunas variantes como, por ejemplo, filtrando las entradas negativas a la salida multiplicando los valores de entrada menores de 0 por un escalar fijo, capa ReLU con fugas, o estableciendo los valores de entrada superiores a un cierto valor a ese límite, capa ReLU recortada.

2.2.3.4 *Cross Channel Normalization*

Esta capa realiza una normalización de respuesta local por canal. Por lo general, sigue a la capa de activación ReLU, y se encarga de reemplazar cada elemento de entrada con un valor normalizado que obtiene usando los elementos de un cierto número de canales vecinos. Esto se conoce como ventana de normalización y su tamaño se especifica con la propiedad *windowChannelSize* de la función *crossChannelNormalizationLayer***.**

Matemáticamente, el parámetro normalizado se obtiene mediante la ecuación [3], siendo x el elemento de entrada, ss la suma de cuadrados de los elementos de la ventana de normalización y K, α y β hiperparámetros de la normalización.

$$x' = \frac{x}{(K + \frac{\alpha * ss}{windowChannelSize})^{\beta'}} \qquad [3]$$

2.2.3.5 *Pooling Layer*

La capa de reducción o pooling se coloca generalmente después de la capa convolucional. Se utiliza principalmente para reducir las dimensiones ancho x alto del volumen de entrada, sin afectar a la profundidad del mismo, reduciendo así el número de conexiones y por tanto la cantidad de parámetros que se deben aprender en capas sucesivas.

Esta reducción de tamaño significará una pérdida de información de la imagen. No obstante, esto puede resultar beneficioso para la red debido a que se tendrá una menor sobrecarga de cálculo para las sucesivas capas de la red, pudiendo además contribuir a la reducción del sobreajuste.

Existe la posibilidad de utilizar dos tipos distintos de capas de reducción pese a que estas utilicen las mismas propiedades para su formación. En este caso se utilizará aquella que divide a la imagen de entrada en un conjunto de rectángulos y respecto a cada subregión, se va quedando con el valor máximo. El tamaño de dichas regiones podrá ser establecido, de forma que, si a este se le da un valor de 3x3, la capa devolverá el valor máximo de cada región de la imagen de entrada tal y como se muestra en la Figura 5.

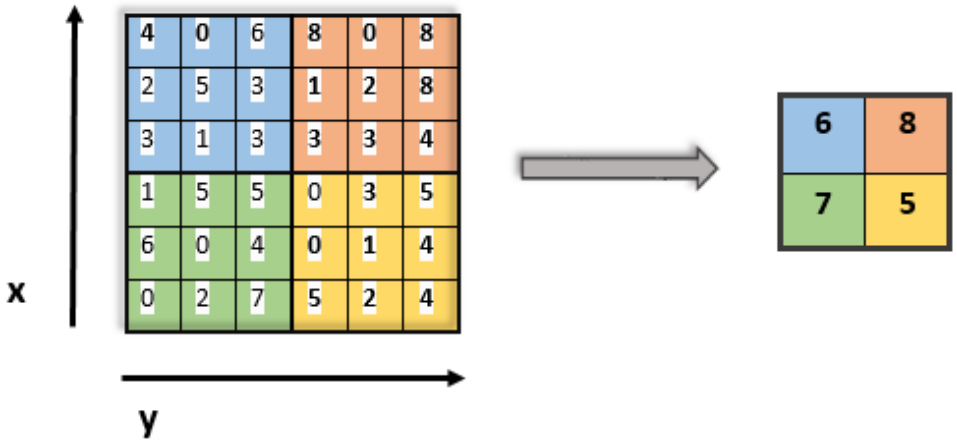


FIGURA 5. OPERACIÓN DE POOLING

2.2.3.6 ***Fully Connected Layer***

Las capas fully connected o totalmente conectadas como su nombre indica, conectan todas las neuronas de la capa anterior con las entradas de la siguiente. Normalmente son utilizadas al final de las capas convolucionales.

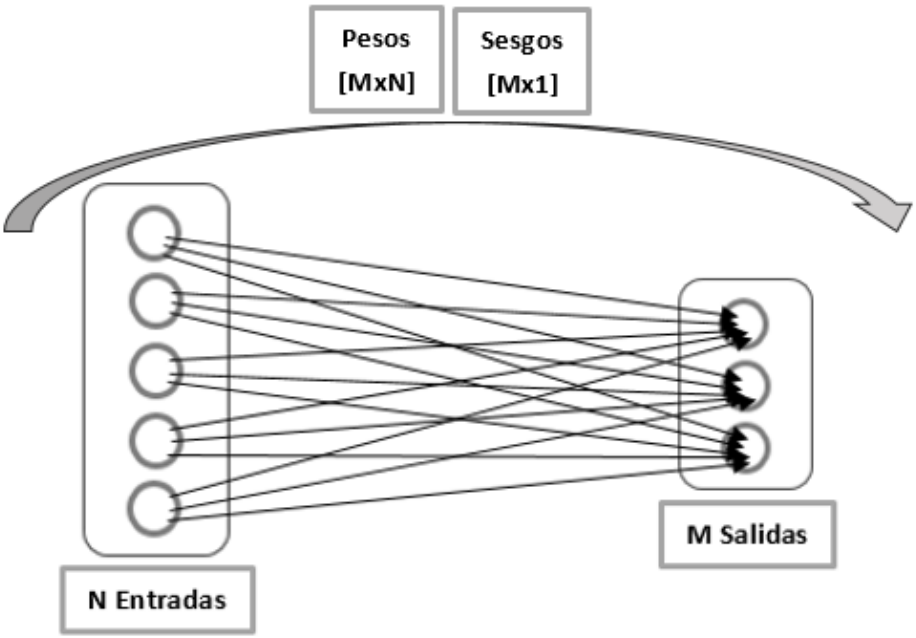


FIGURA 6. CAPA TOTALMENTE CONECTADA

Las salidas obtenidas de las capas convolucionales representarán características de alto nivel de la imagen de entrada. El objetivo de esta capa es utilizar estas características combinándolas para clasificar la imagen de entrada en distintas clases en función de los datos obtenidos.

Para ello, se utilizarán unas matrices de pesos y sesgos, parámetros de aprendizaje que inicialmente tomarán valores aleatorios y se irán ajustando durante el entrenamiento de la red en función de los datos de entrada y el número de clases o salidas que se desea diferenciar.

2.2.3.7 ***Dropout Layer***

La capa de exclusión o dropout tiene la función de intentar evitar el problema de sobreajuste en la red, de forma que los pesos y sesgos de las capas convolucionales y totalmente conectadas no se ajusten en exceso a las imágenes de entrenamiento provocando un mal funcionamiento de la red con las muestras de test.

El proceso seguido en este tipo de capa es eliminar un conjunto aleatorio de activaciones con una probabilidad dada estableciendo sus valores a cero.

2.2.3.8 ***Softmax Layer***

Este tipo de capa trunca las salidas de cada elemento de la matriz formada en la última capa totalmente conectada, de forma que, tal y como se muestra en la Figura 7, estas se encuentren contenidas entre 0 y 1 y que, además, la suma de todas ellas sea la unidad. Para ello, aplica la función sigmoide o función logística tal que:

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{k} e^{x_k}} \qquad [4]$$

Donde x es un vector compuesto por las entradas a la capa, y j indexa cada elemento de salida, $j = 1,2, \ldots., k$

$$\begin{bmatrix} 4 \\ 2.1 \\ 0.5 \end{bmatrix} \rightarrow SoftMax \rightarrow \begin{bmatrix} 0.85 \\ 0.13 \\ 0.02 \end{bmatrix}$$

FIGURA 7. OPERACIÓN DE SOFTMAX

2.3. Algoritmo Bag of Features

El algoritmo denominado como Bag of features o bolsa de características está basado en un método utilizado para el procesado del lenguaje denominado bag of words o bolsa de palabras.

De forma que se construirá un vocabulario de palabras visuales, es decir, un conjunto de características o descriptores que sean representativos de cada una de las clases con las que cuente el conjunto de datos evaluado.

Este algoritmo se puede describir bajo las siguientes etapas de trabajo:

1. La extracción de características. Para describir las partes que componen cada objeto se podrán utilizar distintos detectores:

    1. SIFT. Este algoritmo transforma los datos de la imagen en coordenadas invariantes a la escala, rotación y en cierta medida al cambio de iluminación y al punto de vista 3D, [*Ref.1*]. Está basado en cuatro etapas diferenciadas: detección de máximos y mínimos de la escala, localización de puntos de interés, asignación de la orientación y descriptor del punto de interés.

    2. SURF. Otro de los algoritmos más utilizados para la extracción de puntos de interés en la imagen. Realiza la detección de estos posibles puntos y localiza su posición dentro de la imagen. Seguidamente se representa la vecindad del punto de interés como un vector de características siendo posible regular el tamaño del mismo.

    3. Funciones de extracción personalizadas. Se podrán aplicar algoritmos de detección específicos para el conjunto de imágenes tratadas.

2. Vocabulario Visual. Etapa en la que se conformará un diccionario partiendo de la extracción de los puntos de interés mencionados anteriormente, estos puntos podrán ser representados en el espacio multidimensional del descriptor, Figura 8.

   El objetivo para poder encontrar las palabras visuales con las que crear el vocabulario pasará por localizar grupos de puntos que posean una descripción similar. Es necesario tener en cuenta que en este tipo de extractor, para dar con la frontera entre distintos grupos se partirá de los valores de sus descriptores sin tener conocimiento de las clases o etiquetas de objetos de las que se dispone.

   La tarea de encontrar una separación óptima entre las muestras sin conocer las etiquetas de las mismas, es lo que se conoce como aprendizaje no supervisado, siendo el método empleado en este caso, el algoritmo de clustering o agrupamiento k-means.

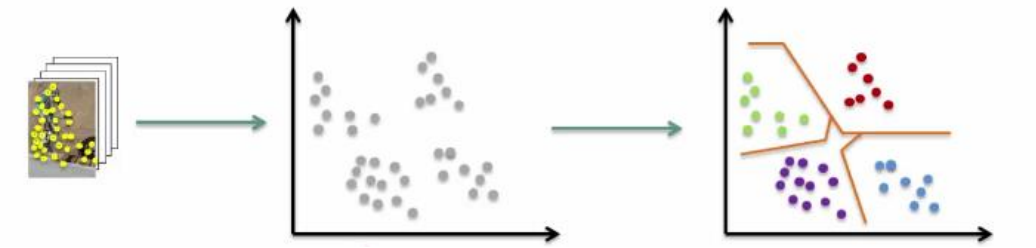


FIGURA 8. PROCESO DE EXTRACCIÓN Y REPRESENTACIÓN EN ESPACIO MULTIDIMENSIONAL DE PUNTOS DE INTERÉS

Este algoritmo tomará como entradas un conjunto de muestras representadas con algún descriptor y el número de agrupaciones que se desean obtener, en este caso el número de objetos distintos a diferenciar en el conjunto de datos. Como resultado final, el proceso seguido devolverá un representante de cada grupo a determinar.

Dicho algoritmo seguirá un proceso iterativo, de forma que tras una primera inicialización de los puntos representantes de los clusters, cada nueva muestra se asignará al grupo cuyo punto representante quede más cercano de la misma, y teniéndola en cuenta, se recalculará la ubicación de los representantes de cada grupo o cluster.

$$Representantes\ de\ cada\ Clúster\ \rightarrow \{c_1, c_2, \dots, c_k\}$$

$$Muestras\ \rightarrow \{x_1, x_2, \dots, x_n\}$$

$$Clúster.\ Paso\ de\ Asignación \rightarrow\ C_j = \{x_i\ |d(x_i, c_j) = \min d(x_i, c_j)\}$$

$$Paso\ para\ Recalcular\ los\ Representantes\ \rightarrow\ c_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

[5]

El objetivo final consistirá en encontrar una agrupación que minimice la distancia de todas las muestras al representante de cada uno de los grupos. Y estos representantes se corresponderán con las palabras visuales en la formación del vocabulario.

2.4. Máquina de Soporte Vectorial

Un clasificador tendrá la tarea de separar los clusters formados con los vectores de características durante el proceso de entrenamiento, de tal forma que a partir del vector de características de una nueva muestra este sea capaz de asignarla a la clase conocida correspondiente.

Los clasificadores podrán utilizar hiperplanos basados en una función lineal para separar los clusters que se hayan formado, estos serán denominados como clasificadores lineales. En caso contrario, se utilizarán clasificadores no lineales, construyendo superficies arbitrarias que permitan separar aquellos que se encuentren solapados.

Una Máquina de Soporte Vectorial (SVM), se basa principalmente en la idea de minimizar el error en la separación de objetos dados y maximizar el margen de separación entre ellos a partir de los vectores de soporte, obteniendo un hiperplano óptimo, Figura 9, de forma que se mejore la capacidad de generalización del clasificador haciéndolo más robusto frente al problema de overfitting.

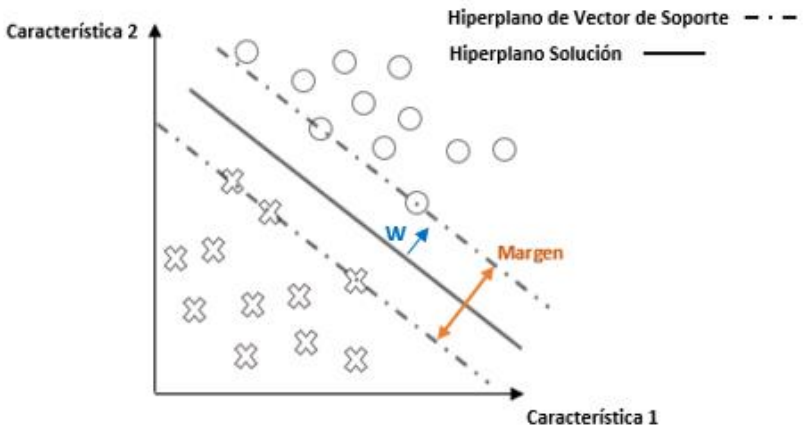


FIGURA 9. HIPERPLANO ÓPTIMO PARA SVM CON 2 CARACTERÍSTICAS

De tal manera que las muestras de cada una de las clases, tendrán una etiqueta con la que a través de la formulación del hiperplano se obtendrá a qué clase pertenece cada muestra en función de si esta es mayor o menor que cero, ecuación [6]. Por lo tanto, la función de clasificación se podrá expresar a partir del signo al aplicar el hiperplano a la muestra x.

$$y_1 = +1 \qquad Muestras\ Positivas: W^T x_i + b > 0$$
$$y_2 = -1 \qquad Muestras\ Negativas: W^T x_i + b < 0$$

$$y_i(W^T x_i + b) \geq 1 \quad para\ todo\ i$$

$$Hiperplano:\ W^T x_i + b = 0 \qquad\qquad f(x) = sgn(W^T x_i + b)$$

[6]

Para que el SVM pueda ser eficiente en conjuntos de datos no linealmente separables, será necesario relajar la condición del margen, lo que implica añadir una cierta tolerancia a errores, ya sean debidos a ruidos o a muestras complejas. La forma de incluir una toleración a errores, es decir, a muestras que queden ubicadas en el espacio de margen, se puede llevar a cabo definiendo una serie de variables de holgura o slack variables, $\xi_i \geq 0$ tal que:

$$y_i(W^T x_i + b) \geq 1 - \xi_i$$

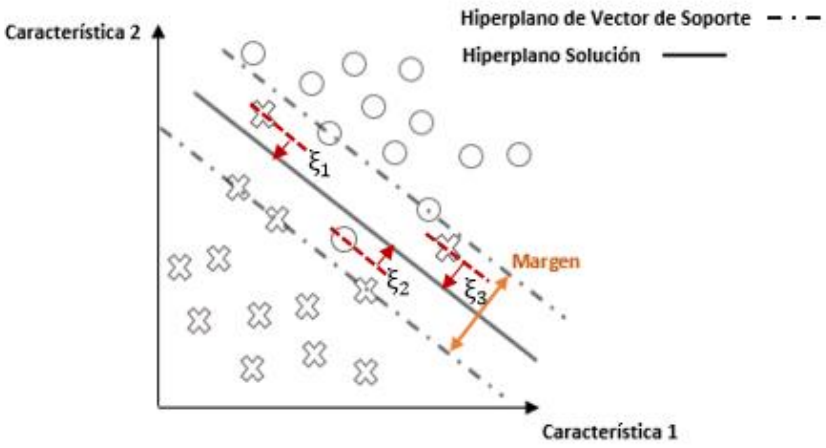


FIGURA 10. SUAVIZADO DEL MARGEN CON VARIABLES DE HOLGURA

El otro punto clave para que el SVM funcione adecuadamente en conjuntos no lineales, pasa por transformar el espacio de características en otro linealmente separable de dimensión superior como se muestra a continuación, Figura 11. Para ello, se utilizaran funciones kernel [*Ref.2*], habiendo una gran variedad de ejemplos de las mismas, siendo algunas de las más habituales: la polinomial, las funciones de base radial o la sigmoide, etc.

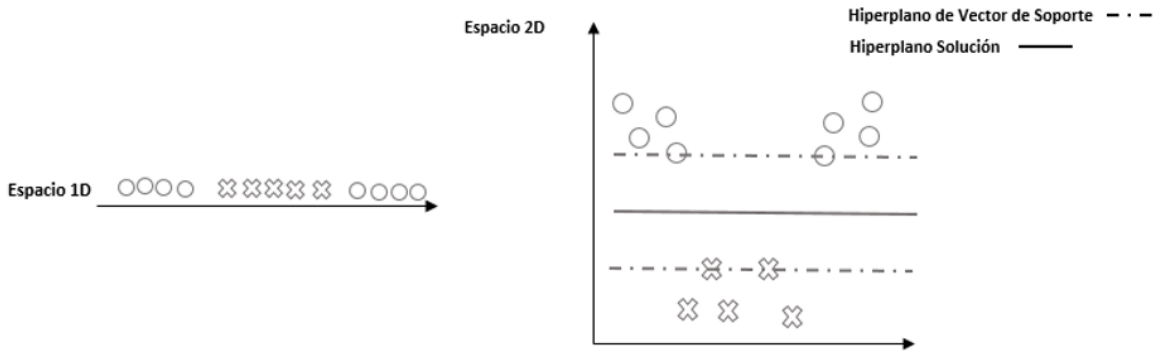


FIGURA 11. CASO DE UN CONJUNTO DE DATOS NO LINEALMENTE SEPARABLE

Este clasificador está desarrollado para trabajar teniendo en cuenta solo dos clases distintas de objetos, no obstante se podrá utilizar para resolver problemas multiclase de dos formas diferentes:

1. Construyendo pares de clases y creando el clasificador binario para cada par, siendo la clase elegida como la más probable, aquella que gane la mayoría de comparaciones.

2. Comparando cada clase con el resto y seleccionando aquella que se encuentre a mayor distancia del hiperplano.

2.5. Interfaz Gráfica de Usuario

El entorno de desarrollo de Matlab incluye un medio de programación visual para aprender a crear interfaces de usuario que recibe el nombre de GUIDE. Para acceder al mismo se podrá hacer a través de su icono, Figura 12, o ejecutando la instrucción guide dentro de la ventana de comandos.

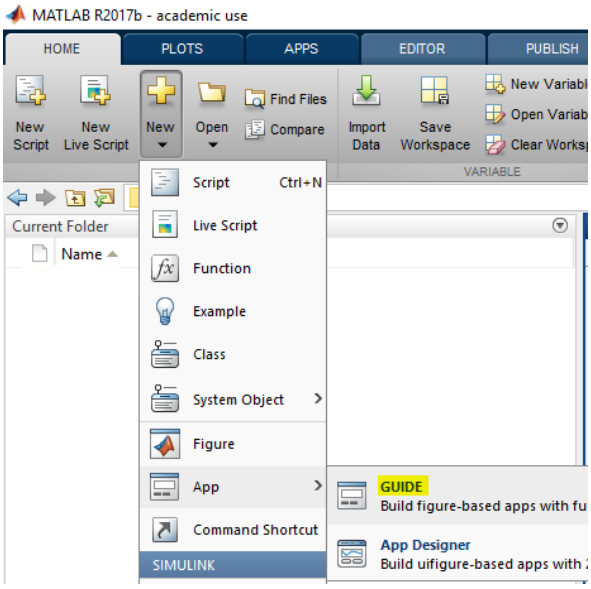


FIGURA 12. UBICACIÓN DE GUIDE

Se seleccionará la opción de utilizar una GUI en blanco para diseñar el programa deseado, donde se podrán encontrar una serie de útiles como, opciones de diseño para tablas, texto, botones, figuras, etc. El editor permitirá agregar estos elementos arrastrándolos al área de la ventana en el que se deseen ubicar, y dentro de cada elemento se podrá acceder a un conjunto de propiedades para personalizar algunos de sus valores.

Una aplicación GUIDE constará de dos archivos que serán generados cuando se realice la acción de guardado, uno con extensión .m que contendrá el código con las correspondencias de los botones de control de la interfaz, y otro .fig con el contenido de los elementos gráficos.

Todos los valores que se encuentran en las propiedades de los elementos agregados a la GUI, y los valores de las variables transitorias del programa se almacenarán a través de la sentencia de salvado de datos *guidata*, en una estructura a la que se podrá acceder a través de un único identificador asignado en: Handles.output = hObject.

Para que se puedan cargar los datos requeridos en la ventana, se dispone de la función *set*, donde será necesario indicar el identificador del elemento y los datos que se desean incluir. Esta operación será definida dentro de uno de los Callback, es decir, una de las funciones disponibles para cada elemento de tipo variable o editable añadido a la GUI.

2.6 Python

El lenguaje de programación Python fue creado a principios de los años 90 por Guido van Rossum. Este es un lenguaje de propósito general que puede usarse de forma efectiva para construir casi cualquier tipo de programa que no necesite acceso directo al hardware.

Python puede no ser adecuado para el desarrollo de programas que requieran altas restricciones de fiabilidad. Sin embargo, este posee ciertas ventajas frente a otros lenguajes de programación entre las cuales se puede destacar su simplicidad, facilitando el aprendizaje del mismo a un usuario medio.

Se trata de un lenguaje interpretado de tipado dinámico, es decir, no será necesario declarar el tipo de dato que va a contener una variable, se podrá determinar durante la ejecución del código siendo posible además, modificarlo para usarlo con diferentes tipos de datos.

Entre los principales motivos por los que se ha decidido utilizarlo para este trabajo, cabe destacar que se trata de un lenguaje de programación Open Source o de código abierto multiplataforma, lo que lo hace accesible a cualquier tipo de usuario sin que este tenga que realizar ninguna inversión para utilizarlo. Por otro lado, ofrece una gran cantidad de bibliotecas de manera que las posibilidades de desarrollo pueden ser muy amplias.

Para el pre-procesamiento de imágenes y la realización de sistemas de clasificación con redes neuronales artificiales, máquinas de soporte vectorial u otro tipo de algoritmos, hay dedicadas distintas bibliotecas entre las cuales se pueden destacar: Pillow, Numpy, Keras, Mxnet, Sklearn o Tensorflow. La información relativa al contenido de las mismas, donde se podrán encontrar una gran variedad de funciones, será accesible por medio de una serie de excelentes fuentes y recursos que se podrán encontrar en línea, y que describirán en detalle cada aspecto relevante de estas.

3. Descripción del Sistema

3.1 Base de Datos

En el caso de estudio que se plantea en este documento, el set de imágenes original está formado por un total de 13 clases que representarán los distintos tipos de gránulos de pólvora a tratar, Figura 13. Cada grupo va a contener un número diferente de elementos, como se puede apreciar en la Tabla 1, sumando un total de 5388 imágenes.

Tipo de Gránulo	Cantidad de Imágenes
f1f331	596
f2f343	536
f6f383	405
f9f360	309
f11f369	311
f15f310	310
f16f336	311
f17f356vn	532
f19f377vn	426
f20f315vn	482
f22f363vn	511
f23f364	311
f24f367	348
TOTAL	5388

TABLA 1. CONJUNTO DE IMÁGENES ORIGINAL

La adquisición de las imágenes de cada uno de los elementos capturados fue tomada desde una dirección y distancia fija. Sin embargo, la posición que presenta cada gránulo en las distintas imágenes sigue un patrón aleatorio, lo que se traduce en la aparición de brillos sobre algunas zonas del propio gránulo y sombras en el fondo de la imagen. Además, la nitidez presentará igualmente una gran aleatoriedad en el conjunto de imágenes debido al enfoque de la cámara. Por ello, se hará más complejo realizar un tratamiento óptimo para todos los casos.

A pesar de ello, se podrán diferenciar tres elementos dentro de cada imagen, donde dos de los mismos se corresponderán con el fondo, una zona blanca y otra grisácea perteneciente al recorte tomado, y un tercero que será el gránulo de pólvora que se desea clasificar.

Cada uno de los tipos de objetos estudiados presentará un tamaño diferente, por lo que, tal y como se puede apreciar en la Figura 13, el tamaño del propio recorte tendrá una estrecha relación con el tamaño real del gránulo.

Además, es conveniente mencionar que aparte de la diferencia de tamaños que presentan las imágenes entre los distintos tipos de gránulos, existirá una gran aleatoriedad de las dimensiones alto x ancho dentro de cada conjunto de estos, lo que puede llegar a suponer un problema para trabajar de forma adecuada sobre estas imágenes.

FIGURA 13. CLASES DE GRÁNULOS TRATADOS

3.2 Arquitectura

El sistema estará compuesto por las fases expuestas en el diagrama mostrado en la Figura 14. Es necesario señalar que el proceso de adquisición de las imágenes fue realizado con anterioridad a este trabajo, por lo que la explicación dará comienzo en la fase de pre-procesamiento. Sin embargo, en uno de los siguientes capítulos, se darán una serie de pautas para realizar la toma de futuras muestras.

La fase de pre-procesamiento de las imágenes englobará todas las propuestas desarrolladas tratando de obtener un conjunto de datos de mayor calidad, cuyo fin será el de facilitar su posterior clasificación.

En la etapa de clasificación se diseñarán y probarán tres variantes. Por un lado, se va a hacer uso de la CNN pre-entrenada explicada en el apartado **2.2.3 CNN Pre-entrenada. AlexNet** aplicándole las imágenes que se desean evaluar en este trabajo. A continuación, se tratará de adaptar la red al conjunto de datos disponible, buscando optimizar tanto los resultados de la clasificación, como la cantidad de parámetro y el uso de memoria necesarios.

Por otro lado, se utilizará un clasificador de tipo máquina de soporte vectorial, que se entrenará bajo un conjunto de características obtenidas de las imágenes a través de: la red pre-entrenada AlexNet, la red adaptada al conjunto de datos del trabajo y el método Bag of Features, usando dos extractores diferentes.

Una vez resuelto este paso, se utilizará un conjunto de imágenes de test para evaluar los clasificadores en los distintos casos que se han mencionado, comprobando los resultados predichos.

Tras finalizar el desarrollo de las etapas representadas dentro de la sección izquierda de Matlab, Figura 14, se diseñará una interfaz gráfica para que cuando el usuario final desee clasificar una nueva muestra, el sistema pueda ser fácilmente utilizado.

En un último bloque, se proporcionará todo el trabajo desarrollado con Matlab bajo el lenguaje de programación Python, será necesario tener en cuenta que no existe una forma sencilla de portarlo de un lenguaje al otro. Por lo que se tratará de reproducir de la forma más exacta posible.

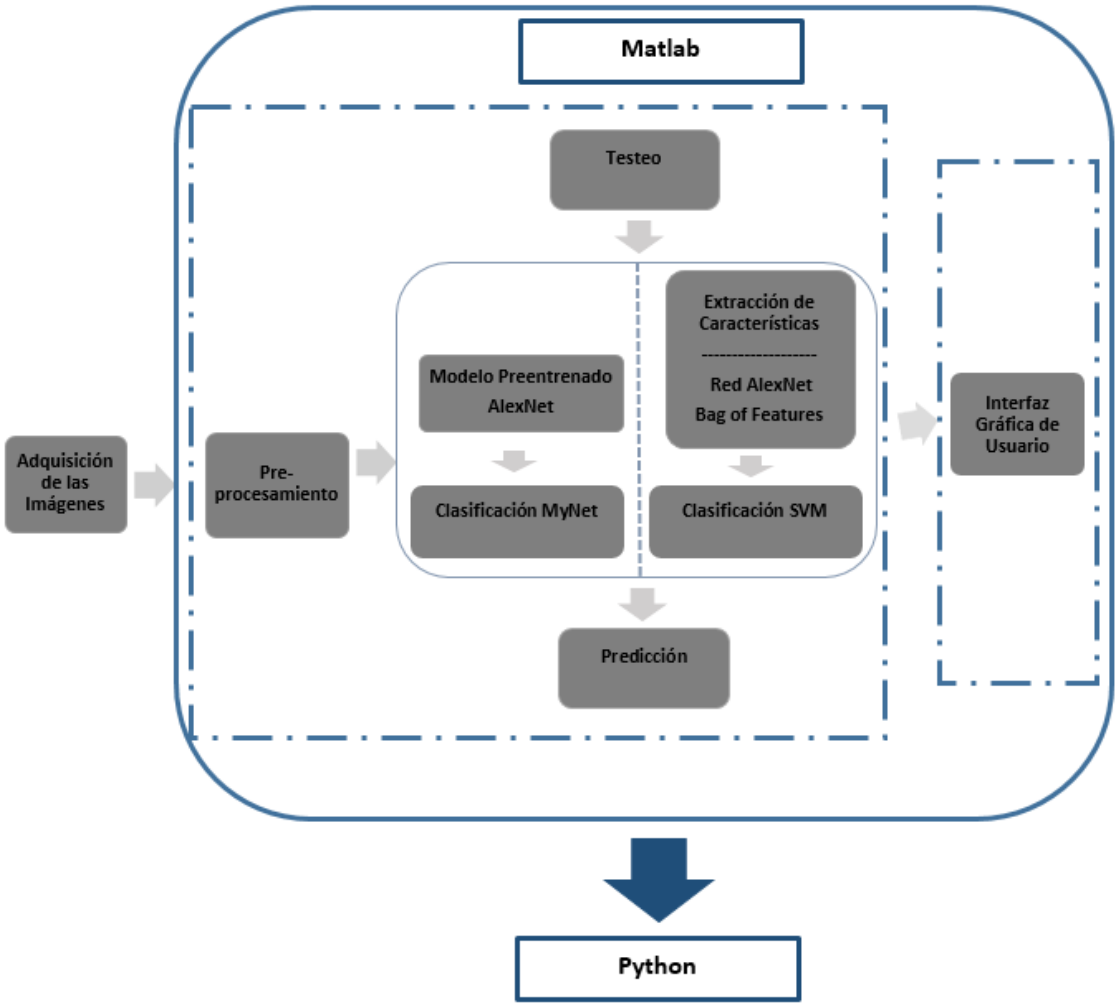


FIGURA 14. ARQUITECTURA DEL SISTEMA

3.3 Etapa de Pre-procesamiento

Este será uno de los puntos clave en el desarrollo del trabajo, puesto que el hecho de tratar las imágenes de una u otra forma influye considerablemente en la tarea de realizar una buena extracción de características y, por lo tanto, en la precisión que pueda presentar el algoritmo de clasificación.

Para este trabajo se cuenta con un conjunto de imágenes del que será necesario extraer ciertas propiedades que caractericen las distintas clases en las que estas se encuentran divididas. De manera que para facilitar y optimizar dicha extracción, las imágenes serán tratadas con los procesos y operaciones explicados a continuación.

3.3.1 Dimensionado

En primer lugar, y como ya se ha comentado en el punto ***3.1 Base de Datos,*** será necesario solventar el problema que conlleva la gran variabilidad de las dimensiones alto x ancho en cada grupo de imágenes. Por lo tanto, antes de realizar cualquier operación a nivel de píxel se precisará unificar todos los elementos del set a un mismo tamaño.

3.3.1.1 Primera Propuesta

En un primer paso, se planteó la opción de redimensionar todo el conjunto de datos al tamaño que requiere la red CNN pre-entrenada utilizando la función *imresize* que brinda Matlab en la toolbox de procesamiento de imágenes [*Ref.3*]. No obstante, el primer problema que se encontró es que, al tener una variabilidad muy grande en el tamaño de las imágenes, el resultado de aplicar esta transformación suponía en muchos caso la perdida de la relación de aspecto del objeto, como se

puede apreciar en la Figura 15 y, por otro lado, la pérdida del tamaño real de los mismos, como se muestra en la Figura 16.

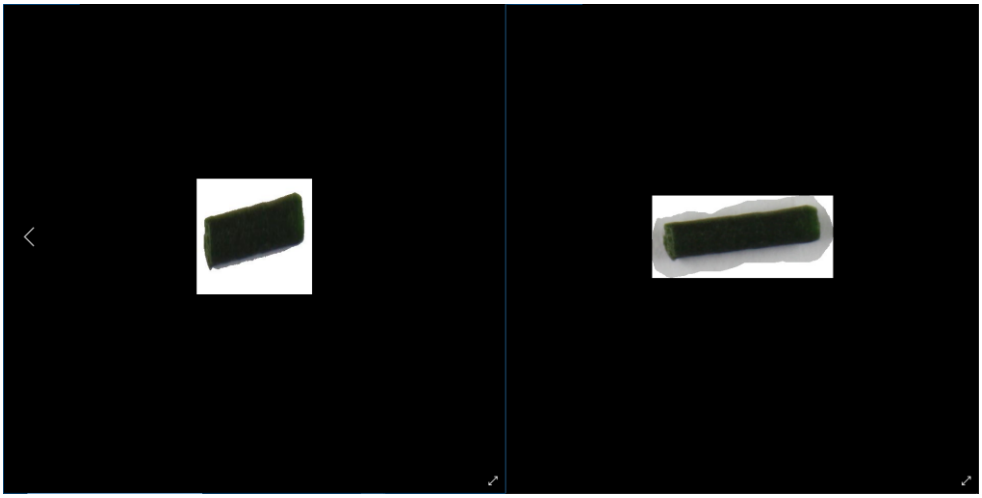


FIGURA 15. RELACIÓN DE ASPECTO. IMAGEN ORIGINAL (DCHA), IMAGEN TRATADA (IZDA)

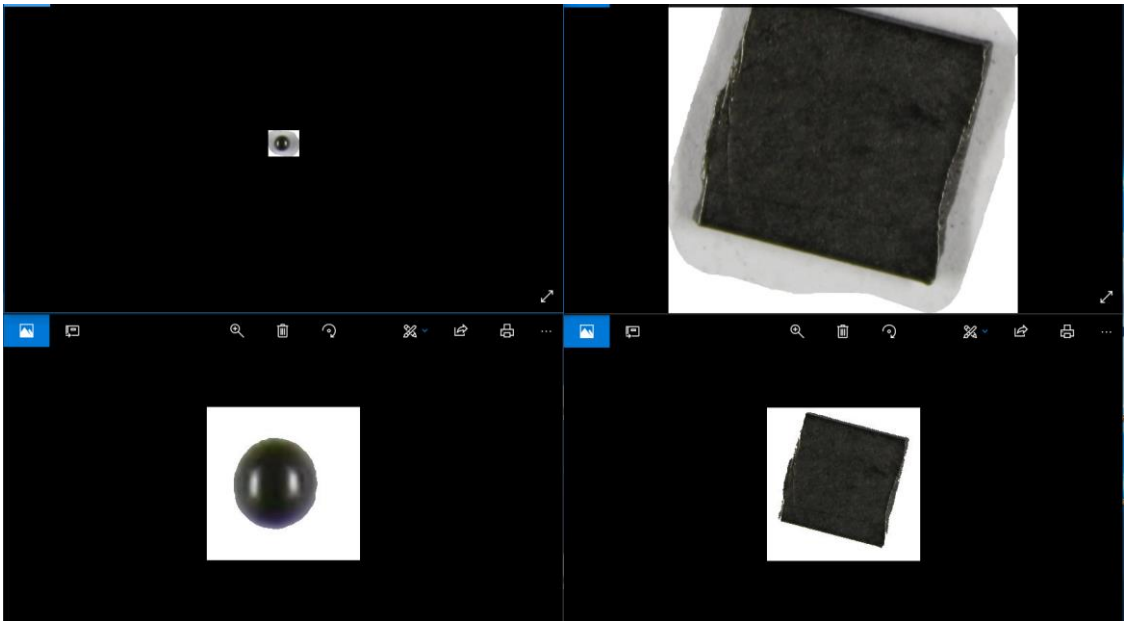


FIGURA 16. RELACIÓN DE TAMAÑO. IMAGEN ORIGINAL (SUPERIOR), IMAGEN TRATADA (INFERIOR)

3.3.1.2 Segunda Propuesta

Se debe partir de la condición de que existe relación entre el tamaño que tiene la imagen, es decir, las dimensiones alto x ancho que esta presenta según el tipo de gránulo, y el tamaño real del objeto capturado. Por lo tanto, esta segunda alternativa planteada tiene el objetivo de redimensionar las imágenes sin crear o perder información del gránulo de pólvora, de forma que se pueda mantener la relación de aspecto y el tamaño real del mismo**.**

El proceso seguido consta de varias operaciones incluidas en las líneas 2 a 11 del Código 1. En primer lugar, se busca la dimensión máxima que presenta el conjunto de imágenes de estudio. A continuación, es necesario conocer las dimensiones originales de cada imagen para posteriormente realizar un rellenado de píxeles en cada uno de los canales que la componen hasta la máxima calculada.

Para ello, se utiliza la función *padarray*, donde se debe especificar un vector de enteros no negativos con la cantidad de relleno que se desee agregar y la dimensión a lo largo de la cual agregarlo teniendo presente que se debe mantener al gránulo centrado. Puesto que, en el conjunto de datos, la gran mayoría de las imágenes presentan un corte cuadrado, el rellenado hasta la dimensión máxima calculada se ha efectuado tanto en el ancho como en el alto de la imagen.

Esta función incluye varios parámetros para escoger las condiciones en las que se desea realizar el relleno. Por defecto, este se lleva a cabo con píxeles de valor 0 (negro), por lo que, como se cuenta con imágenes cuyo fondo se corresponde con píxeles de valor 255 (blanco), se ha considerado conveniente incluir este valor de relleno para aplicarlo en nuestras imágenes. Por otro lado, se ofrece la posibilidad de añadirlo en tres puntos diferentes de la imagen original: antes del primer elemento del array, después del último elemento o bien en ambos casos. Como será indispensable mantener el gránulo centrado, se ha utilizado la última de las opciones expuestas, que coincide con el valor por defecto de este parámetro.

El principal inconveniente de utilizar esta función es que la cantidad de píxeles que se incluyan debe expresarse como un número entero positivo. Por lo que, al realizar el cálculo para obtener dicho valor, en función de si este es par o impar se hace necesario realizar un redondeo a 0 utilizando la función *fix*, lo que puede significar que las dimensiones finales difieran de las esperadas en una pequeña cantidad de píxeles, ecuación [7].

$$Dimensiones\ Finales\ Esperadas\ \rightarrow\ [Dmax, Dmax, 3]$$

$$\begin{cases} \dfrac{(Dmax - (Alto\ o\ Ancho)Original)}{2} = n^{\underline{o}}\ Par & \rightarrow [Dmax, Dmax, 3] \\ \dfrac{Dmax - (Alto\ o\ Ancho)Original}{2} = n^{\underline{o}}\ Impar & \rightarrow [Dmax - 1, Dmax - 1, 3] \end{cases}$$

[7]

La forma más sencilla de solucionar este conflicto pasa por realizar una comprobación del valor de las dimensiones finales y, en caso de no ser las esperadas, incluir tras el último elemento del array un relleno cuyo tamaño se corresponda con un vector fila y/o columna como se muestra en las líneas 14 a 19 del Código 1.

```
1. ImgOrig = imread(RutaimgRGB);        % Lectura de la Imagen
2. [m,n,s] = size (ImgOrig);            % Dimensiones Originales
3. ValRell = 255;                       % Valor de Relleno
4.
5. if m < Dmax && n < Dmax              % Se comprueba si las
   Dimensiones Originales son Inferiores a la Dmax
6.
7.       MmaxR =fix((Dmax-m)/2);
         % Cálculo del N° de píxeles de relleno, Alto de la Imagen
8.       NmaxR =fix((Dmax-n)/2);          % Cálculo del N° de píxeles
   de relleno, Ancho de la Imagen
9.
10.      imgRell= zeros(Dmax,Dmax,3);
         % Nuevo array de Dimensiones Máximas
11.      imgRell = padarray(ImgOrig, [MmaxR NmaxR],ValRell);
         % Rellenado de la Imagen
12.       [Mr,Nr,s] = size (imgRell);
         % Dimensiones Imagen Rellenada
13.
14.      if (Mr ~=Dmax)    % Comprobamos Alto de la Imagen
15.            imgRell = padarray(imgRell, [1 0],ValRell,'post');
               %Se incluye píxel Después del último elemento
16.      end
17.      if (Nr ~=Dmax)    % Comprobamos Ancho de la Imagen
18.            imgRell = padarray(imgRell, [0 1],ValRell,'post');
               %Se incluye píxel Después del último elemento
19.      end
20.
21.      end
```

CÓDIGO 1. RELLENADO DE IMÁGENES

3.3.2 Segmentación

Esta fase es, probablemente, la etapa más importante del pre-procesado de las imágenes, ya que su objetivo final es el de obtener el objeto de estudio en cuestión con todas aquellas modificaciones que impliquen una mejora en el mismo.

En primer lugar, es imprescindible tener en cuenta las particularidades que presentan las imágenes, descritas en el apartado ***3.1 Base de Datos***, con respecto al proceso de adquisición seguido.

Existen diversas técnicas de segmentación basadas en bordes, en regiones, etc. En este trabajo se han utilizado operaciones que trabajan sobre los píxeles y el histograma que componen la imagen.

3.3.2.1 Primera Propuesta

Originalmente, las imágenes se encuentran en el espacio de color RGB, lo que implica que el tratamiento que se aplique sobre estas tendrá que ser efectuado para los tres canales que la componen. Teniendo en cuenta esto, se planteó la posibilidad de utilizar la función *imadd.* Esta función realiza una suma de matrices que tendrán que ser del mismo tamaño y tipo de datos. Existiendo la posibilidad de realizar la suma de la imagen de entrada RGB que se desea segmentar consigo misma, con otra imagen distinta o con un valor constante que será añadido a cada elemento de la matriz de entrada de la forma mostrada en la ecuación [8].

$$(Img_{Salida})Pixel[i,j,c] = (Img_{Entrada})Pixel[i,j,c] + (Img_{Entrada})Pixel[i,j,c]$$

$$(Img_{Salida})Pixel[i,j,c] = (Img_{Entrada})Pixel[i,j,c] + Valor\ Constante\ (uint8)$$

[8]

En este caso, se trabaja sobre una imagen en la cual el objeto que se desea segmentar es considerablemente más oscuro que el resto de la misma, por lo que este se puede diferenciar perfectamente del fondo.

De tal manera que si se suma la imagen RGB consigo misma o con un valor constante, el resultado de aplicar esta función se corresponde con una imagen más clara. Teniendo en cuenta que en aquellos casos en los que la suma supere el valor máximo posible para los datos de tipo uint8, estos serán truncados tomando un valor de 255, y obteniendo así, el gránulo segmentado tal y como se muestra en la Figura 17.

FIGURA 17. SEGMENTACIÓN DEL GRÁNULO CON LA FUNCIÓN IMADD

Sin embargo, hay que tener en cuenta algunos problemas que surgen al aplicar esta función. Como se puede observar tanto en la Figura 17 como en la Figura 18, para conseguir la eliminación del fondo de la imagen ha sido necesario modificar el valor de los píxeles del objeto, lo que puede llegar a ser un inconveniente si se tiene un objeto muy brillante o con una sombra muy pronunciada. Por lo tanto, al modificar el valor de estos píxeles, se corre el riesgo tanto de perder información relevante del objeto, como de incluir información del fondo que no se requiere para su posterior clasificación, siendo uno de estos casos el representado en la Figura 18.

FIGURA 18. SEGMENTACIÓN DEL GRÁNULO CON PERDIDA DE INFORMACIÓN

Para las siguientes propuestas se planteó la opción de hacer uso de la imagen sobre un solo canal, ya que por un lado puede facilitar el tratamiento de la misma y, por otro lado, se reduce el número de datos que se deben administrar. Debido a esto, la primera transformación que se aplicará sobre el conjunto será una conversión a escala de grises de tal forma que para cada píxel:

$$PixelGray\,[i,j] = 0.2989 \cdot PixelR\,[i,j] + 0.587 \cdot PixelG\,[i,j] + 0.1 \cdot PixelB\,[i,j]$$

[9]

Existen varios métodos para realizar esta operación. En Matlab, la función *rgb2gray* realiza la suma ponderada de las tres componentes de la imagen con los pesos expuestos en la ecuación [9]. Estos pesos varían en función de la sensibilidad con la que el ojo humano percibe cada color [*Ref.4*].

3.3.2.2 Segunda Propuesta

En la mayoría de los casos, el gránulo se puede diferenciar del fondo con bastante exactitud, por lo que se puede tratar de separar del resto de la imagen de diversas formas.

En una segunda hipótesis se pensó realizar una segmentación mediante el uso de la detección de bordes, este elemento es utilizado en muchas aplicaciones de procesamiento de imágenes o reconocimiento de patrones.

Los bordes de una imagen, en este caso en escala de grises, son píxeles con una variación elevada de su valor en una pequeña área. Estos se detectan calculando el gradiente en cada punto de la imagen, siendo este un vector perpendicular al borde tal que:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{d}{dx} f(x,y) \\ \frac{d}{dy} f(x,y) \end{bmatrix}$$

$$G = \sqrt{G_x^2 + G_y^2} \quad ; \quad \Phi(x,y) = \tan^{-1}\frac{G_y}{G_x}$$

[10]

Como se puede apreciar en la Figura 19, el vector G apunta en la dirección donde se produce una mayor variación de f en el punto (x, y) por unidad de distancia, con la magnitud y dirección dadas en la ecuación [10]. Se considerará que existe borde si la magnitud del gradiente supera un determinado umbral.

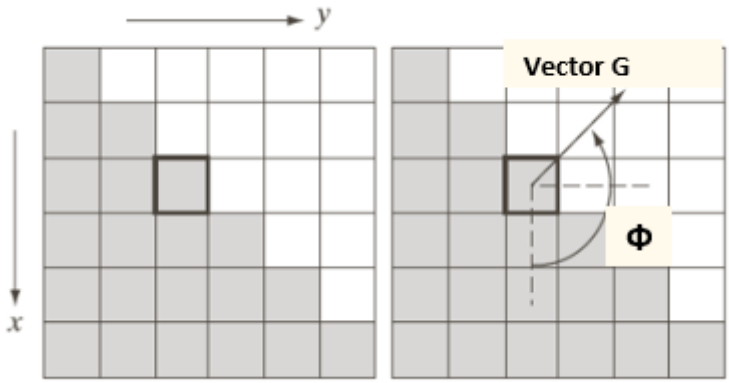


FIGURA 19.GRADIENTE PARA LA DETECCIÓN DE BORDES

En este punto se han probado tres operadores distintos basados en la primera derivada.

- Operador Prewitt. Este tipo de operador implica a los vecinos de filas/columnas adyacentes, de manera que proporcionará cierta inmunidad al ruido que puede tener la imagen. Utiliza dos máscaras de convolución, una para el gradiente fila y otra para el gradiente columna, ecuación [11], con los cuales se obtienen los bordes verticales y horizontales de la imagen.

$$Mascara\ Gx \rightarrow \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} ; \quad Mascara\ Gy \rightarrow \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad [11]$$

- Operador Sobel. Muy similar al anterior, con la particularidad de poseer una mayor sensibilidad a la localización de bordes diagonales. En la práctica se puede observar que las diferencias son mínimas en nuestras imágenes.

$$Mascara\ Gx \rightarrow \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} ; \quad Mascara\ Gy \rightarrow \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad [12]$$

- Operador Canny. Este tipo de operador se desglosa en varias tareas:

  1. Calcula el gradiente de cada píxel de la imagen.

  2. Realiza una supresión no máxima al resultado del gradiente, es decir, para cada píxel $p(x, y)$ comprueba cuál de las direcciones d_k: 0°, 45°, 90°, 135° es más parecida a la dirección $\Phi(x, y)$ del gradiente. En el caso de que el valor de la magnitud de $G(x, y)$ sea más pequeño que un vecino en la dirección d_k, el píxel final de salida será $s(x, y) = 0$, y en caso contrario será $s(x, y) = 1$. Siendo el objetivo final la obtención de bordes de un píxel de grosor.

  3. Realiza un umbralizado con histéresis fijando dos umbrales tal que $Umbral_1 < Umbral_2$. Evitando de esta forma la perdida de bordes y la aparición de ruidos.

     3.1 Se localizan los puntos en los que $p(x, y) > Umbral_2$ y en los que $Umbral_1 < p(x, y) < Umbral_2$.

     3.2 Se busca conectividad entre los píxeles mayores al $Umbral_2$, añadiendo aquellos que se encuentran entre ambos umbrales con los denominados como píxeles fuertes, en el caso de que posean una 8-conectividad. La salida será un conjunto de bordes conectados que describen las propiedades de los puntos de borde.

A continuación, se muestran una serie de ejemplos obtenidos haciendo uso de la función *edge* con los operadores explicados anteriormente.

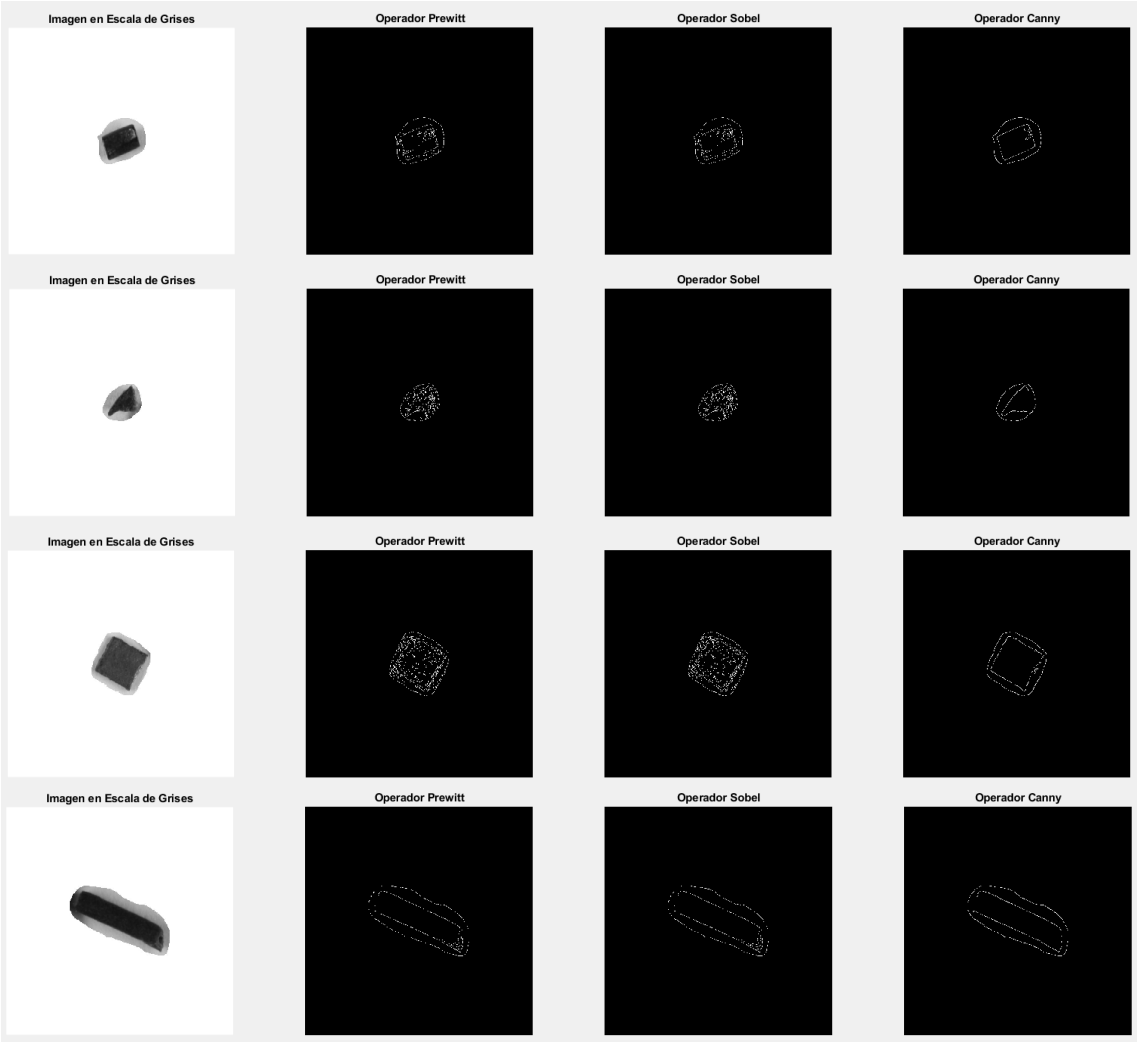


FIGURA 20. OBTENCIÓN DE BORDES CON LOS OPERADORES DESCRITOS

Como se puede apreciar en la Figura 20, los mejores resultados se han dado para el caso del uso de la extracción de bordes con el operador Canny, teniendo en cuenta que se han seleccionado los valores umbral de:

$$Umbral_1 = 0 \;\; y \;\; Umbral_2 = 0{,}4.$$

No obstante, surgen una serie de problemas al utilizar este tipo de algoritmos de segmentación. El primer inconveniente que aparece es que, junto con el contorno del gránulo, en la mayoría de los casos se obtiene también el contorno del recorte.

En segundo lugar, en una gran parte de las imágenes, los contornos del gránulo no terminan de cerrarse, por lo que se hace necesario aplicar funciones de dilatación para cerrarlos. Sin embargo, el uso de estas operaciones supone nuevamente que el contorno del recorte quede cerrado de igual manera que queda para el gránulo.

Por último, se añade el caso en el que los gránulos presentan tonos de gris muy parecidos a los que forman la parte de fondo del recorte, por lo que la extracción del borde se realiza con parte del gránulo y parte del recorte.

En definitiva, se puede concluir que en cualquiera de los tres casos va a resultar difícil separar ambos objetos de forma óptima para el conjunto de datos.

3.3.2.3 Tercera Propuesta

3.3.2.3.1 Binarización

En este último punto se pensó en segmentar el objeto utilizando técnicas de binarización sobre la imagen. Partiendo de esta en escala de grises se han tratado dos opciones:

1. El uso de la función *imbinarize* estableciendo un valor umbral tal que:

$$\begin{cases} p(x,y) > Umbral \ \rightarrow \ s(x,y) = 1 \\ p(x,y) \leq Umbral \ \rightarrow \ s(x,y) = 0 \end{cases} \qquad [13]$$

2. El uso de la función *imadd*, comentada en el apartado ***3.3.2.1 Primera Propuesta***, combinada con la función *Imbinarize* estableciendo un valor umbral, ecuación [13].

La primera pregunta que se debe plantear para aplicar estos métodos es: ¿cómo se puede establecer el umbral con un valor adecuado? Una de las opciones para ajustar de la forma más rigurosa posible el valor de este parámetro, puede llevarse a cabo haciendo un estudio del tipo de histograma que presentan las imágenes de las distintas clases de gránulos.

El histograma de una imagen representa la distribución de la cantidad de píxeles que se encuentran en los distintos niveles de color, en este caso niveles de gris. La segmentación basada en el histograma se lleva a cabo utilizando procesos de umbralización, que se pueden clasificar:

1. Según el número de umbrales a aplicar en la imagen: global, local o multiespectral para imágenes RGB.

2. Según la forma de calcularlos: supervisados o no supervisados.

Así pues, se han estudiado todas las distribuciones que presentan cada una de las clases del set para los dos métodos propuestos. De tal forma que, como se puede apreciar en la Figura 21, en el primer caso se tendrán tres lóbulos diferenciados. Uno se corresponderá con los píxeles pertenecientes al gránulo, otro con la parte de fondo del recorte de la imagen que se desea eliminar y, por último, el que engloba todos los píxeles blancos del resto de la imagen.

En el segundo caso, al haber utilizado la función *imadd*, el lóbulo correspondiente al recorte de la imagen será eliminado, debido a que esos píxeles pasarán a ser un valor máximo del tipo de dato uint8, es decir, un valor de 255. Por lo que se producirá una imagen con mayor contraste, dando la opción de afinar más el valor de umbral a seleccionar que en el caso anterior.

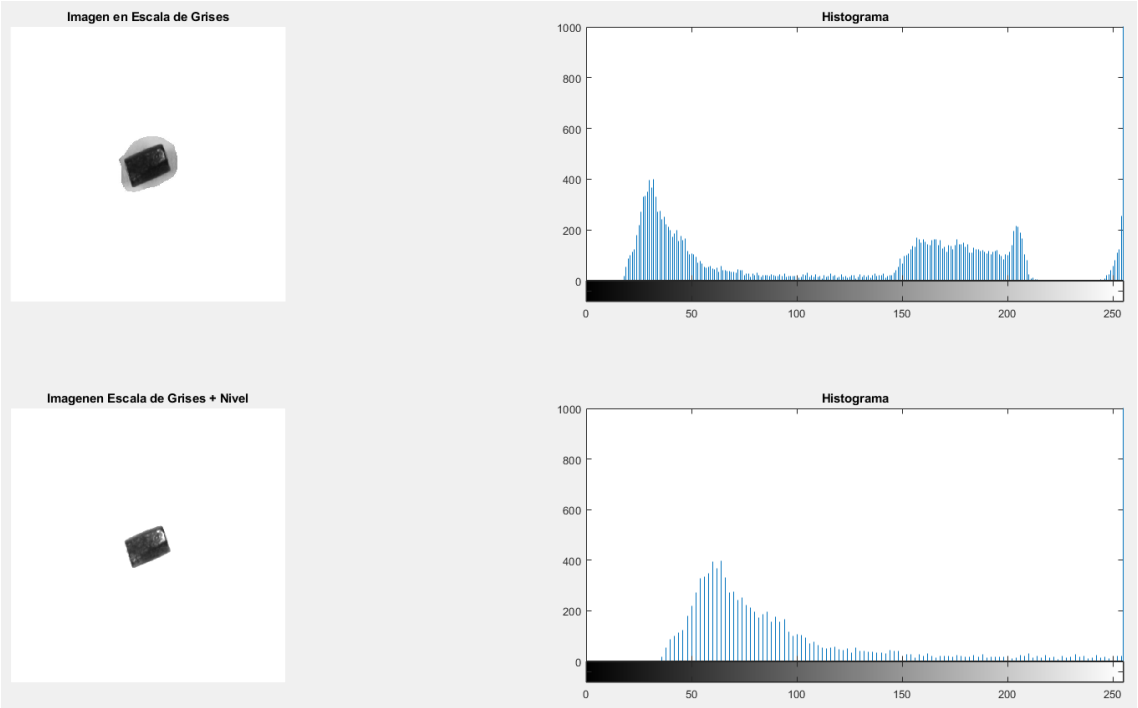


FIGURA 21. HISTOGRAMA CON LOS NIVELES DE GRIS DE LOS PIXELES DE LA IMAGEN

Pese a tener trece clases diferentes de objetos, las distribuciones de los histogramas que estos presentan tienden a mostrar en la mayoría de los casos, los tres lóbulos mencionados anteriormente, aunque con distinta cantidad de píxeles dependiendo del tamaño del gránulo. Por lo tanto, se podrá seleccionar un único valor umbral para todo el conjunto de imágenes.

Como este valor se ha seleccionado a partir de la imagen con mayor contraste, y en un alto porcentaje de casos las sombras habrán sido eliminadas casi en su totalidad, se puede escoger un umbral ajustado en la zona máxima de valores del histograma.

Procediendo de esta forma, se evitarán problemas que pueden llegar a surgir al realizar dilataciones, erosiones y otras operaciones para rellenar y cerrar contornos que se vean afectados por los brillos. A continuación, se han expuesto algunos de los resultados obtenidos, mostrados en la Figura 22.

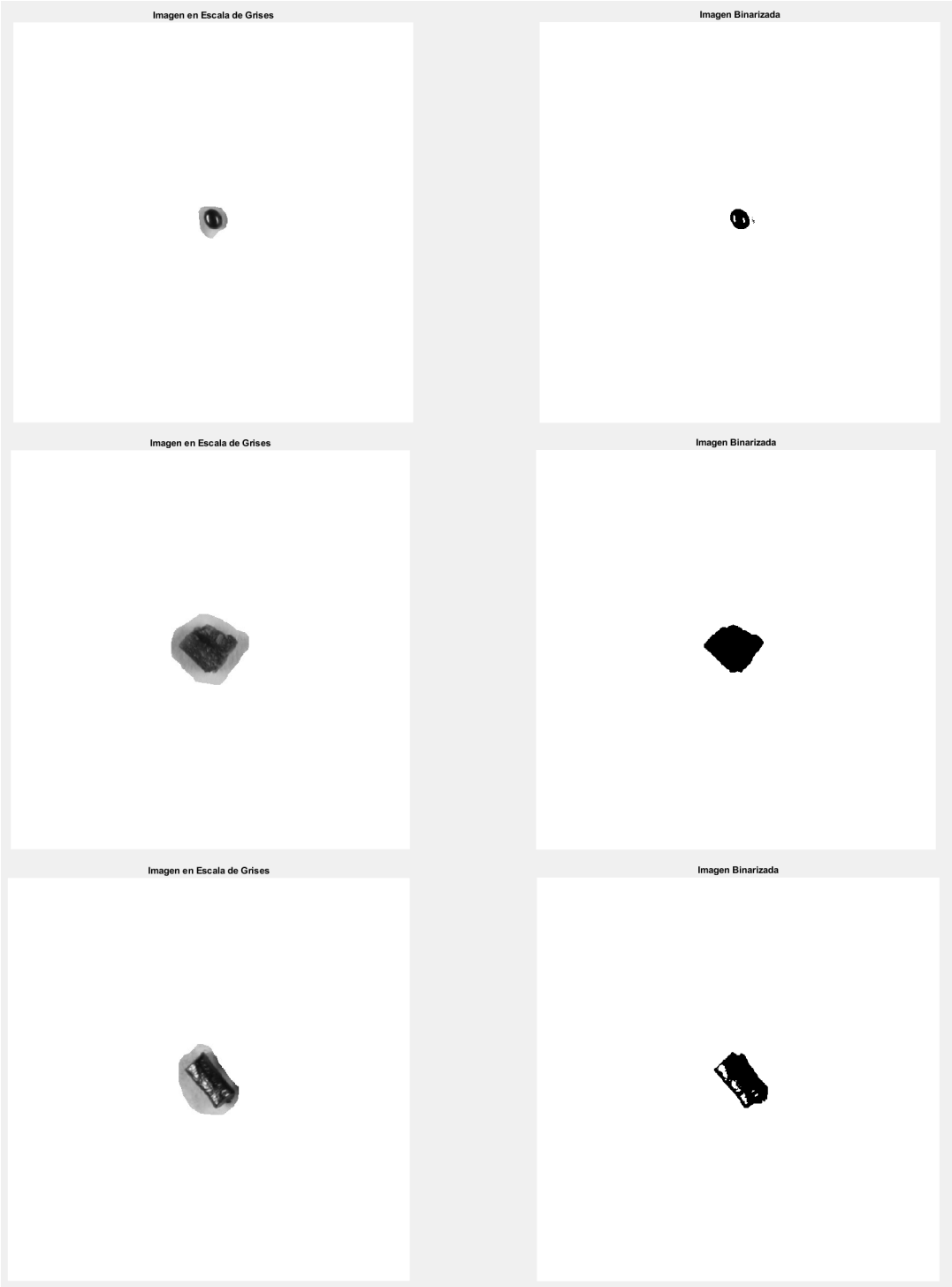


FIGURA 22. IMÁGENES BINARIZADAS BAJO UMBRAL ESTABLECIDO

Cuando en el apartado ***3.3.2.1 Primera Propuesta*** se hacía referencia a la pérdida de información de algunos píxeles del gránulo, se puede apreciar que para el primer y tercer caso expuesto en la Figura 22 los brillos del gránulo se han interpretado como fondo.

Estos efectos hacen necesario el uso de las operaciones explicadas a continuación para recuperar esa información perdida y, en otros casos, para eliminar posibles restos de sombras que no se hayan descartado totalmente.

3.3.2.3.2 Operaciones Morfológicas

Las operaciones morfológicas simplifican las imágenes y conservan las principales características de forma de los objetos, permitiendo hacer reconstrucciones de las mismas de manera óptima. Estas pueden ser utilizada en imágenes binarias o en escala de grises para: supresión de ruidos, simplificación de formas, destacar estructuras de los objetos, obtener características de estos, etc.

3.3.2.3.2.1 Operación de Dilatación

La operación de dilatación dada una imagen binaria A y un elemento estructural B se puede definir como:

$$A \oplus B = \{x \mid A \cap B_x \neq \emptyset\} \quad [14]$$

De tal forma que, como se ejemplifica en la Figura 23, si se sitúa el elemento estructural sobre un píxel de valor ‘1’ de la imagen de entrada, todos aquellos que se incluyan en el área de B tomarán el valor de ‘1’ en la imagen de salida.

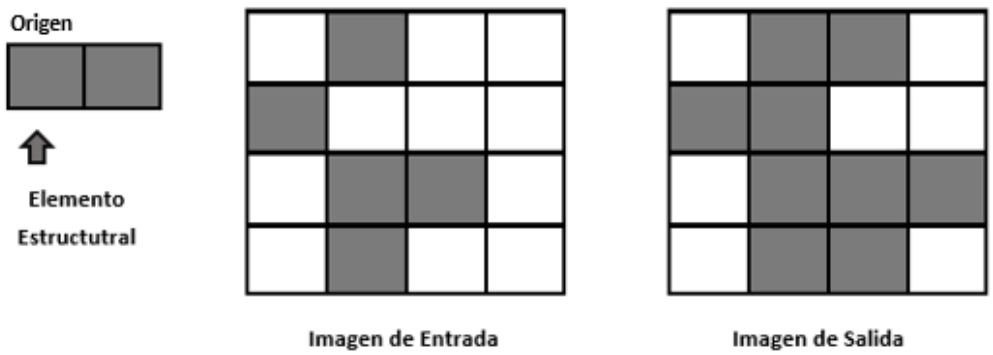


FIGURA 23. EJEMPLO OPERACIÓN DE DILATACIÓN

En Matlab, se dispone de la función *imdilate* para llevar a cabo esta operación. Esta función recibe como parámetros: la imagen que se desea dilatar, es decir, aquella que ha sido segmentada en el punto ***3.3.2.3 Tercera Propuesta***, y un elemento estructural escogido adecuadamente para cada caso.

Para escoger el elemento estructural de forma adecuada, se podrá utilizar la función *strel*. Esta ofrece una gran variedad de estructuras, siendo necesario facilitar el tipo de forma y el tamaño del elemento que mejor se puede adecuar a los objetos sobre los que se desea trabajar.

Particularmente para este estudio, se dan casos en los que, por el efecto de los brillos, el área perteneciente al gránulo que ha sido interpretada como fondo no termina de cerrarse, lo que supone una complicación en un intento de recuperarla. Por consiguiente, se ha aplicado un pequeño elemento estructural de tipo disco del cual será necesario indicar el radio y el entorno vecindad, habiéndose seleccionado un $r = 2 \ y \ vecindad = 4$. Estos valores fijados son suficientes para cerrar los huecos existentes en nuestras imágenes, sin dar lugar a que se puedan unir pequeños restos de sombras que hayan podido quedar al realizar el proceso de binarizado.

3.3.2.3.2.2 Operación de Relleno

La operación de relleno se encarga de localizar aquellas pequeñas áreas de valor ‘0’ dentro de un objeto cerrado en una imagen binaria y darles el valor ‘1’. Para ello, utiliza un algoritmo basado en la reconstrucción morfológica [*Ref.5*].

3.3.2.3.2.3 Operación de Erosión

La operación de erosionado dada una imagen A y un elemento estructural B, se produce cuando los píxeles de B no están contenido en píxeles del mismo valor en A. Es decir, como se puede apreciar en la Figura 24, si el píxel de la imagen de

entrada que se está evaluando es ‘1’ y los píxeles del elemento estructural de valor ‘1’ están contenidos en píxeles de valor ‘1’ en la imagen original, el píxel de salida será ‘1’. En caso contrario, tomará el valor de ‘0’.

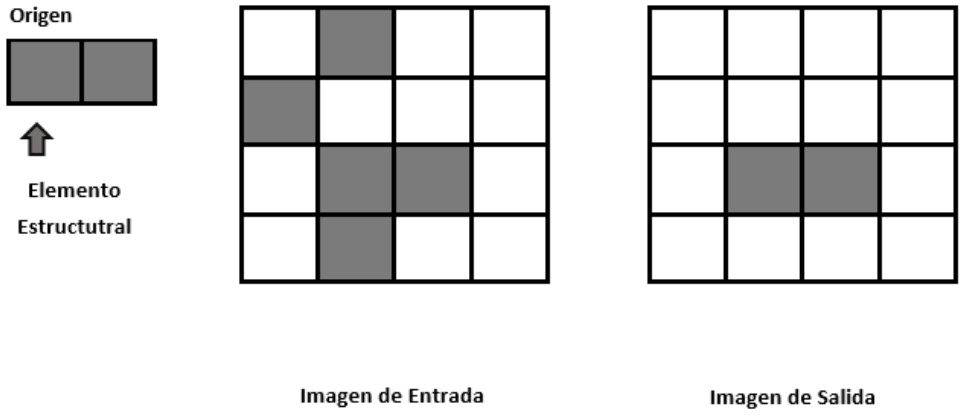


FIGURA 24. EJEMPLO OPERACIÓN DE EROSIÓN

Matlab aporta la función *imerode* para realizar esta operación. Igual que en el caso de la operación de dilatación, es necesario indicar el tipo de elemento estructural que se desea utilizar. En este caso, será uno de tipo disco de radio y entorno vecindad, $r = 3 \;\; y \;\; vecindad = 4$.

Existe otra función para realizar la misma secuencia de operaciones, dilatación seguida de erosión, denominada *imclose*. Esta realiza el cierre morfológico de la imagen en escala de grises o binaria y se puede definir como:

$$A \bullet B = (A \oplus B) \ominus B \qquad [15]$$

La diferencia con respecto a la aplicación de las funciones de dilatación y erosión que se han realizado, es que en este caso el elemento estructural utilizado tiene que ser el mismo para ambas operaciones. No obstante, el hecho de utilizar distintos elementos estructurales en los datos de este trabajo supone una mejora en la tarea correspondiente al aislamiento de los pequeños objetos pertenecientes al fondo de la imagen, que han quedado de forma residual hasta este punto del proceso.

3.3.2.3.2.4 Operación de Supresión de Objetos Pequeños

Existen varias funciones para localizar los objetos que componen la imagen y seleccionar aquellos sobre los que interese trabajar en cada momento, o simplemente para eliminar aquellos que no aporten información relevante a la aplicación a desarrollar. La transición de operaciones realizada sigue el orden expuesto a continuación:

1. Etiquetar los elementos que componen la imagen.
2. Calcular el área de los objetos.
3. Localizar el área máxima de interés y suprimir el resto de los objetos.

3.3.2.3.2.4.1. Etiquetar los elementos que componen la imagen

La función *bwlabel* de Matlab se utiliza para localizar los distintos objetos que componen una imagen binaria. Esta devuelve una matriz de tipo double con el número de objetos localizados en la imagen de entrada, y las etiquetas de cada uno de ellos, tal y como se puede apreciar en la Figura 25.

1	1	1			
1	1	1			1
1	1	1		1	1
1	1			1	1
1			1	1	1
				1	1

Imagen Binaria

1	1	1			
1	1	1			2
1	1	1		2	2
1	1			2	2
1			2	2	2
				2	2

Matriz de Etiquetas

FIGURA 25. EJEMPLO OPERACIÓN DE ETIQUETADO DE OBJETOS

3.3.2.3.2.4.2. Calcular el área de los objetos

Para obtener el tamaño de los objetos se ha hecho uso de *regionprops*. Esta función devuelve el valor de cada una de las propiedades que se especifiquen como parámetros a calcular, pudiendo encontrar una gran variedad de atributos distintos para el tipo de datos de entrada establecidos: la imagen binaria, la matriz de etiquetas obtenidas en el punto anterior, etc.

En este caso, se ha calculado el área de cada etiqueta, del tal forma que se obtenga para cada objeto un valor escalar con la cantidad de píxeles que lo componen, tal y como se ha expuesto en el fragmento de Código 2.

```
1. [L,num]=bwlabel(img_erode);          % Etiquetar los Objetos
2.  prop=regionprops(L,{'Area'});       % Calcular Área de Objetos
3.  ObjMax=max([prop.Area]);            % Tomar el área Máxima
```

CÓDIGO 2. CANTIDAD DE OBJETOS Y TAMAÑO DE LOS MISMOS

3.3.2.3.2.4.3. Localizar área máxima de interés y suprimir el resto de los objetos

En los casos en los que la imagen erosionada resulte en varios objetos, se aplicará la función *bwareaopen*, con la que todos los objetos que cuenten con una cantidad de píxeles inferior a un valor dado quedarán eliminados, como se muestra en el Código 3. Este parámetro o valor límite será el del objeto de mayor tamaño.

```
1. if num ~= 1
2.       % Supresión de Objetos Pequeños
3.       Granulo = bwareaopen(img_erode,ObjMax);
4. else
5.       Granulo = img_erode;
6. end
```

CÓDIGO 3. FRAGMENTO DE CÓDIGO FINAL PARA SUPRESIÓN DE OBJETOS

Todas estas operaciones se han llevado a cabo a partir de la imagen binaria que se obtuvo en el punto ***3.3.2.3.1 Binarización***. Obteniendo como resultado final de este proceso una imagen binaria en la cual el gránulo ha sido segmentado quedando libre de brillos y sombras, tal y como se muestra en la Figura 26.

Una vez se haya finalizado el proceso de segmentación de la imagen, será el momento preciso en el que se recuperará el área de píxeles RGB de la imagen original que conforman el objeto segmentado, obteniendo el gránulo final mostrado en la Figura 27.

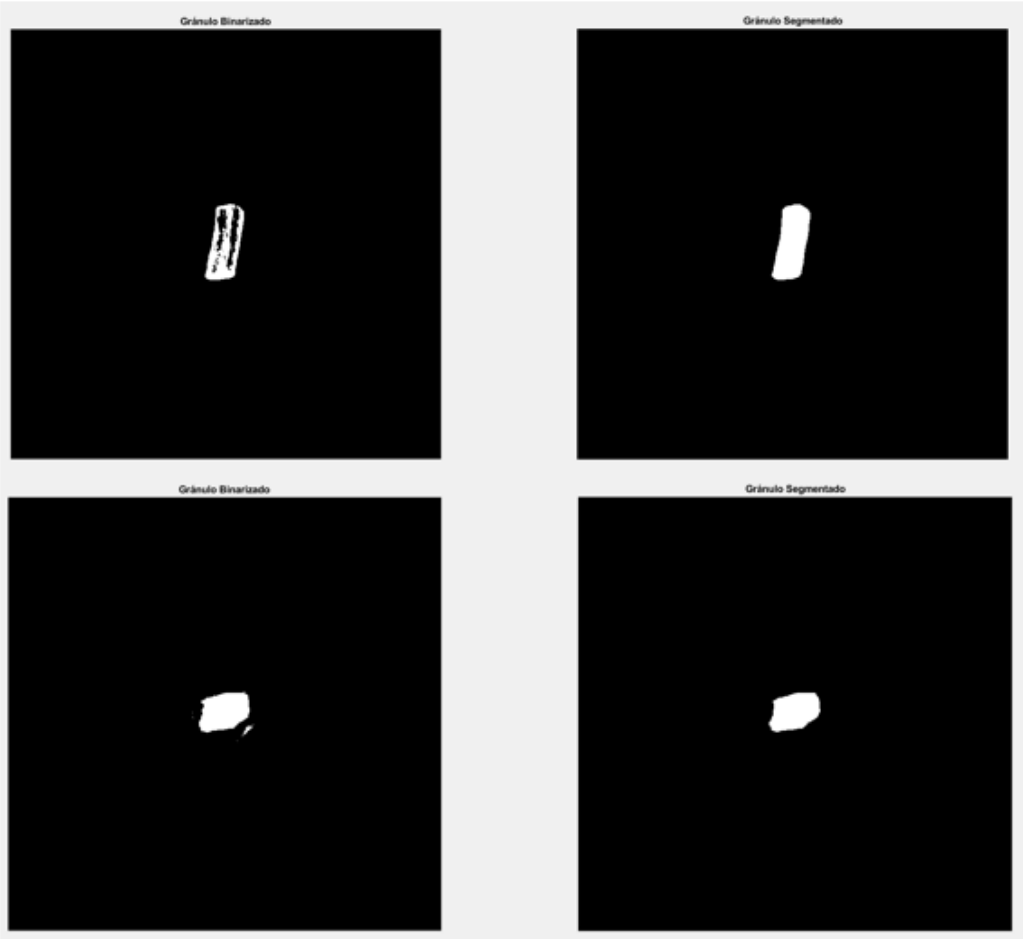


FIGURA 26. SEGMENTACIÓN DEL GRÁNULO

FIGURA 27. GRÁNULO ESCALA RGB RECUPERADO

3.3.2.3.2.5. Orientación

El hecho de que las imágenes hayan sido tomadas con los objetos colocados aleatoriamente supone algunos problemas, siendo uno de ellos el tratado en este punto.

Las entradas de la red neuronal se corresponden con cada uno de los píxeles de la imagen de entrada, de forma que, si los objetos de una clase concreta se encuentran en posiciones aleatorias, pese a estar centrados, se estará pasando a una misma entrada diversos valores en función de la posición del objeto. Esto puede significar un problema a la hora de extraer ciertas características de esa clase de objeto y, por consiguiente, de cara a realizar una buena clasificación de estos.

Así pues, el próximo paso se realiza con la finalidad de orientar todos los objetos del set de datos en una posición concreta. Para ello, la función *regionprop* comentada anteriormente en el punto ***3.3.2.3.2.4.2***, dispone de la propiedad *orientation*. Esta crea la elipse mínima que contiene al objeto, y obtiene el valor del ángulo formado por el eje mayor de la misma y la horizontal como un escalar en grados que puede ir desde -90° hasta 90°.

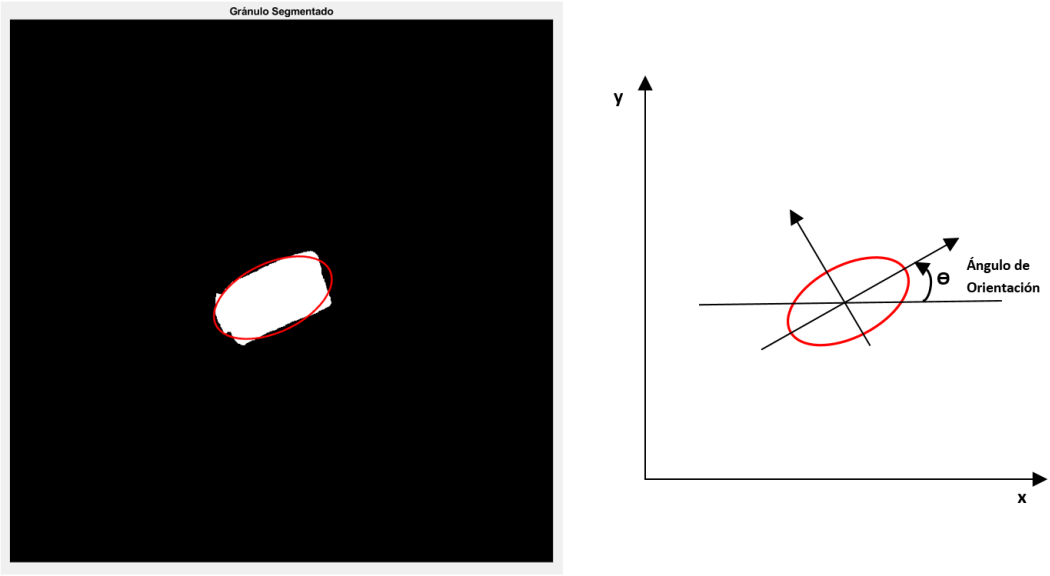


FIGURA 28. ÁNGULO DE ORIENTACIÓN DEVUELTO POR LA FUNCIÓN

Una vez calculado el valor del ángulo, Figura 28, se aplica la función *imrotate* sobre la imagen RGB recuperada tras la segmentación del gránulo. Esta requiere la imagen a rotar y el ángulo en grados de rotación, y ofrece la posibilidad de escoger el método de interpolación a utilizar y el tamaño de la imagen de salida.

El sentido de giro se hace de forma antihoraria según el ángulo Θ que se le desee aplicar como parámetro de entrada. En caso de que se requiera girar en el sentido horario será necesario que el valor del ángulo sea negativo.

Por defecto, esta función aplicará un método de interpolación del vecino más cercano, pero en este caso se ha decidido utilizar una interpolación bicúbica. Con este método, se cuentan los valores en los píxeles conocidos que rodean a uno dado en una vecindad de los 4x4 píxeles más cercanos, se toma el promedio ponderado de estos 16 píxeles y se calcula el valor interpolado, produciendo imágenes más nítidas que en el caso del método del vecino más cercano y del método bilineal.

Por último, es necesario indicar que, al realizar el giro, la imagen de salida que se desea obtener debe poseer las mismas dimensiones que la imagen de entrada, tal y como se muestra en la Figura 29. Para ello, se ha utilizado el parámetro *bbox* para recortarla. De lo contrario, se tendrá una imagen de salida tan grande como sea necesario para incluir totalmente la imagen rotada, cuyos píxeles exteriores a la imagen girada tomaran el valor de 0.

FIGURA 29. OBJETO EN POSICIÓN ORIGINAL BW Y OBJETO FINAL EN ESCALA RGB

Una vez se haya completado todo el tratamiento de las imágenes con las mejoras y operaciones que se han explicado a lo largo de este apartado, será el momento preciso en el que se podrá comenzar a tratar todos los aspectos y procesos relativos a la etapa de clasificación: los clasificadores que se van a usar, las modificaciones que van a sufrir, la forma de configurarlos, etc.

3.4 Etapa de Clasificación. AlexNet

La red AlexNet, explicada en el apartado ***2.2.3 CNN Pre-entrenada. AlexNet***, está entrenada para imágenes de entrada de dimensiones 227x227x3 y tiene la capacidad de distinguir hasta 1000 clases distintas de objetos. Por lo tanto, si se desea emplear esta red, que ha sido diseñada bajo unas condiciones concretas, será necesario adaptar los datos a dichas condiciones.

Las imágenes obtenidas en la etapa de procesado, apartado ***3.3 Etapa de Pre-procesamiento***, van a tener un tamaño final de 735x735 píxeles.

En este caso, se aplicará la función *imresize* sobre las imágenes en escala RGB con el valor de escalado que permita obtener las dimensiones que requiere AlexNet en su capa de entrada. Este parámetro debe expresarse como un escalar real, y de la misma forma que se hacía en la operación de rotación, punto ***3.3.2.3.2.5.,*** será necesario establecer un método de interpolación, siendo el método bicúbico el que presenta mejores resultados debido al rango de vecindad que considera para dar el valor óptimo a cada píxel evaluado de la imagen.

La clasificación se va llevar a cabo utilizando la función *classify* que Matlab ofrece para obtener los resultados de salida de una red neuronal. Esta función cuenta con la posibilidad de trabajar con distintos parámetros. Entre ellos, es necesario indicar la red que se va a utilizar y el conjunto de imágenes que se desea clasificar. Además, se podrán expresar otra serie de preferencias como, el entorno de ejecución, el tamaño de los lotes por medio de los cuales se evaluaran las imágenes, etc.

En primer lugar, se ha utilizado la función *alexnet* tal y como se muestra en el fragmento de Código 4. Esta función se encuentra incluida en el paquete de Neural Network Toolbox [*Ref.6*] y devuelve el modelo pre-entrenado de AlexNet.

El conjunto de imágenes de test viene dado por un almacén de datos, creado a partir de la función *imageDatastore* que Matlab ofrece para poder administrar grandes cantidades de imágenes y que incluye principalmente, los archivos de imagen y las etiquetas de cada una de las mismas.

```
1. net = alexnet;
2. [predictedLabels,MatrixResult] = classify(net,imdsPolvorasTest);
```

CÓDIGO 4. PREDICCIONES DE ALEXNET PARA NUESTROS DATOS DE TEXT

Esta función requiere principalmente de los parámetros que se han mencionado, aunque será conveniente tener en cuenta otros aspectos de la misma. El entorno de ejecución se selecciona de forma automática utilizando una GPU, si esta se encuentra disponible y cumple con los requisitos expuestos [*Ref.7*], o en caso contrario utilizando la CPU. El tamaño de los lotes de imágenes debe ser un número entero positivo y estará definido de forma predeterminada en un valor de 128. La elección del mismo dependerá de la cantidad de memoria que se pueda utilizar, es decir, cuanto mayor sea el tamaño del lote, mayor será la cantidad de memoria requerida, pero también se posibilitará una predicción más rápida.

La ejecución de esta función dará como argumentos de salida las etiquetas o clases, a las que la red considera que pertenece esa imagen y la matriz NxM de pesos (variables entre 0 y 1), donde N se corresponde con el número de muestras o imágenes evaluadas, y M con el número de clases de objetos.

En cualquier caso, hay que tener en cuenta una serie de observaciones a partir de las operaciones realizadas en este apartado. En primer lugar, el escalado realizado sobre las imágenes va a implicar la perdida de una cantidad considerable de píxeles, y por lo tanto, de una posible pérdida de información del gránulo.

En segundo lugar, la red va a tratar de clasificar los objetos de las imágenes disponibles para este trabajo, en aquellos que posean características similares a los que se han utilizado para entrenarla. Sin embargo, entre las clases de objetos que se han utilizado en AlexNet no se encuentran los gránulos de pólvora que se quiere clasificar en nuestro sistema. Por lo tanto, es probable que la clasificación no se realice de forma correcta.

A continuación, y como solución a este problema, se va a exponer una alternativa para tratar de adaptar la CNN al conjunto de datos disponible.

3.5 Etapa de Adaptación de la Red AlexNet

Entrenar una red desde cero supone trabajar con unos pesos inicializados al azar, y dependiendo del conjunto de datos y del tamaño de la red, esto puede suponer un proceso en mayor o menor medida duradero. En Matlab, existe la posibilidad de usar una red pre-entrenada como punto de partida para aprender una nueva tarea.

Así pues, en este apartado se ha planteado la posibilidad de aplicar entradas de diversas dimensiones acotando la salida de la red a los 13 tipos diferentes de objetos con las que se cuenta en el trabajo. Para ello, va a ser necesario realizar una serie de modificaciones que se explicarán a continuación.

3.5.1 Escalado de las Imágenes

En este caso se aplica la función *imresize* sobre la imagen en escala RGB de dimensiones 735x735x3 bajo distintos valores de escalado, estableciendo este parámetro como un escalar real y bajo el método de interpolación bicúbico. La finalidad de realizar estas variaciones consistirá en comprobar si el tamaño de las imágenes influye en la clasificación de los distintos objetos.

3.5.2 Modificaciones en la Red AlexNet

3.5.2.1. Capa de Entrada

Para obtener la capa de entrada a la red se utiliza la función *imageInputLayer* en la cual se debe especificar el tamaño (ancho x alto) y el número de canales que estas presentan.

Esta función permite además la posibilidad de realizar una serie de transformaciones a la imagen de entrada durante el entrenamiento de la red. Por defecto, se aplica la normalización de cada imagen de entrada, pasando a la primera capa de convolución el resultado de substraer la imagen media del set de entrenamiento a la imagen de entrada para prevenir la saturación de la red.

3.5.2.2. Transfer Learning. Últimas Capas de la Red

Como se mencionó anteriormente, las últimas capas de la red AlexNet están configuradas de tal forma que esta es capaz de clasificar las imágenes de entrada de dimensiones 227x227x3 en 1000 tipos de objetos diferentes. Estas capas se encuentran formadas por: tres capas totalmente conectadas, una capa de activación softmax y una última capa de clasificación. De manera que, para adaptar la estructura de la red a nuestras necesidades, se procederá realizando la extracción de las mismas para reemplazarlas de la siguiente forma:

3.5.2.2.1. Capas FullyConnected

Para establecer este tipo de capa se hace uso de la función *fullyConnectedLayer*, que requiere como mínimo especificar el número de salidas que se desea obtener expresado como un número entero positivo. No obstante, esta función ofrece una gran variedad de parámetros para configurar la capa adecuadamente según las especificaciones del trabajo desarrollado.

En primer lugar, es necesario mencionar que el número de entradas a la primera capa totalmente conectada depende del tamaño de las imágenes que se pasen a la capa de entrada.

En este trabajo, se ha propuesto la opción de utilizar el conjunto de imágenes con distintos tamaños, por lo que se ha indicado que el número de entradas a esta capa se configure automáticamente en función del tamaño del conjunto de datos de entrada a la red.

En segundo lugar, como ya se ha mencionado y se pueda apreciar en la Figura 30, la salida de la red está entrenada para clasificar 1000 objetos diferentes. Por lo tanto, el tamaño de salida en la última capa totalmente conectada se debe modificar para las 13 clases de gránulos de pólvora.

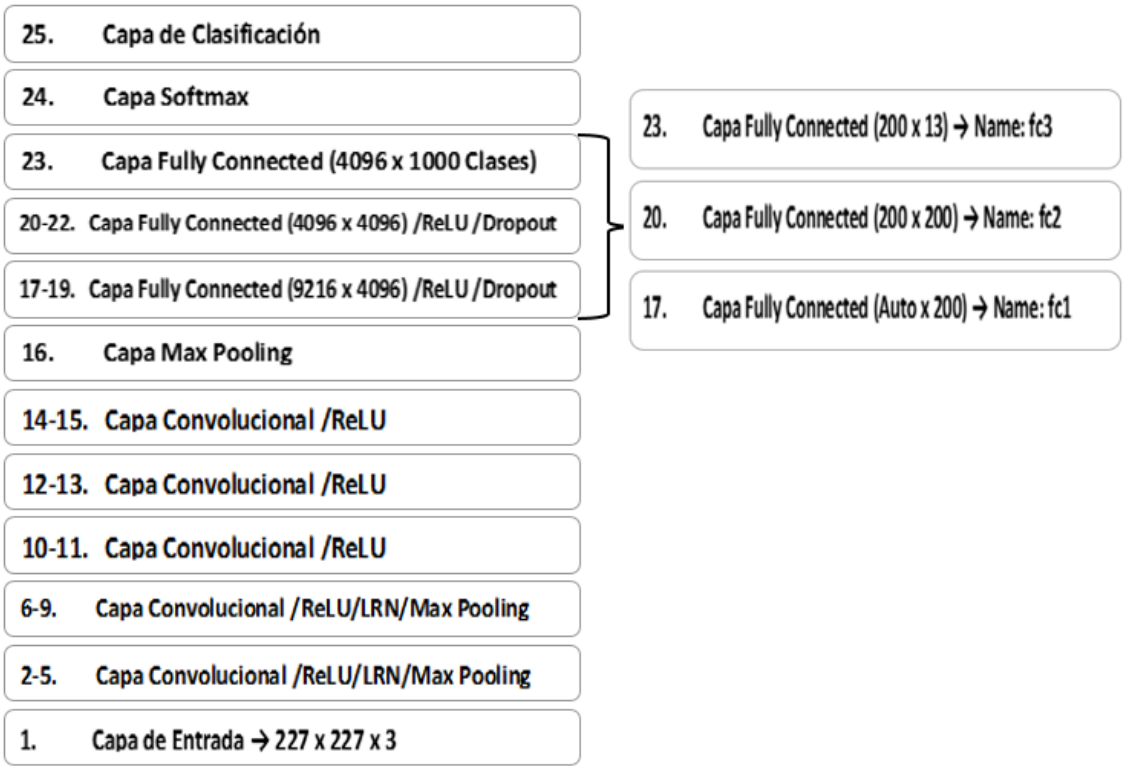


FIGURA 30. MODIFICACIONES SOBRE LAS CAPAS TOTALMENTE CONECTADA DE ALEXNET

Por otro lado, es necesario contar con varios factores para establecer el número de salidas y entradas en el resto de casos, siendo estos las salidas de la primera y segunda capa, y las entradas de la segunda y tercera capa.

La cantidad de conexiones que deben tener estas capas es un problema no resuelto. A priori, no hay un método consolidado para conocer de forma precisa la conectividad necesaria para resolver una tarea concreta [*Ref.8*]. Por lo que, el proceso ha consistido en realizar varios entrenamientos variando el número de conexiones y seleccionando el menor número con el que se han producido los mejores resultados. De esta forma, debido a que el número de datos con los que se ha de trabajar es notablemente inferior, se facilita un entrenamiento más rápido.

Además, se debe tener en cuenta que la base de datos y la variabilidad de objetos utilizados en este trabajo son considerablemente más pequeños que los utilizados en la red AlexNet. Por lo tanto, es lógico pensar que no va a ser necesario extraer la misma cantidad de características para el caso de estudio de este trabajo, puesto que como se puede ver en el Gráfico 1 ha resultado ser contraproducente.

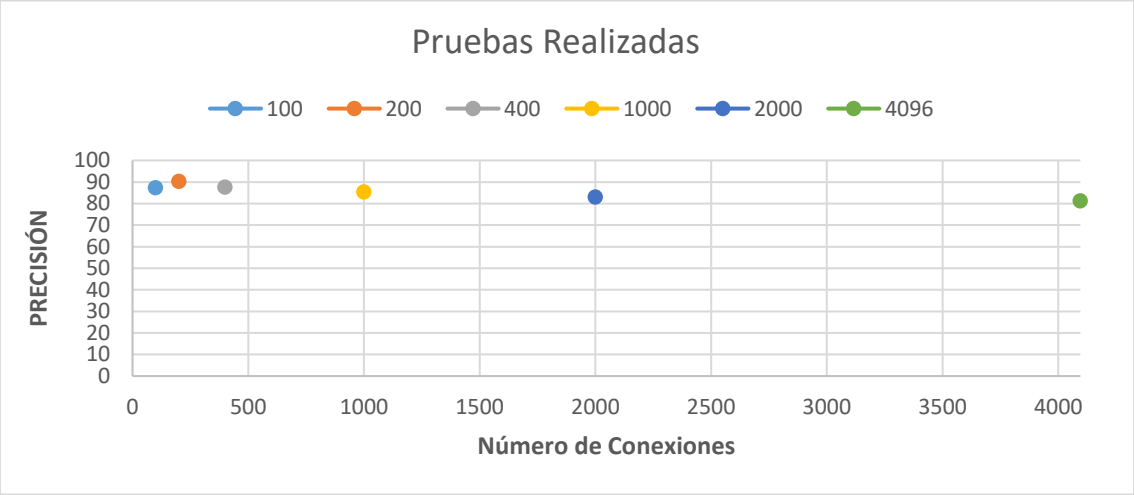


GRÁFICO 1. PRECISIÓN OBTENIDA VARIANDO LA CANTIDAD DE CONEXIONES ENTRE LAS CAPAS TOTALMENTE CONECTADA

Como última observación, la red AlexNet trabaja con tres capas de este tipo, tal y como se muestra en la Figura 30. De manera que, para referenciar cualquiera de las mismas, se hace necesario diferenciarlas utilizando el parámetro de nombre que la función ofrece con este fin.

3.5.2.2.2. Capa SoftMax

Este tipo de capa aplica una función de activación como la mostrada en el apartado ***2.2.3 CNN Pre-entrenada. AlexNet,*** ecuación [4].

Para crear una capa de este tipo se utiliza la función *SoftmaxLayer*, a la cual se podrá dar nombre a través del parámetro que ofrece de manera opcional para ello.

3.5.2.2.3. Capa de Clasificación

La última capa de la red se crea con la función ClassificationOutputLayer. Esta contiene el nombre de la función de pérdidas para el entrenamiento de la red multiclase, el tamaño de la salida, expresada como un número entero positivo correspondiente a la cantidad de clases del conjunto de datos, las etiquetas de cada una de estas clases y el nombre de la capa, si se desea especificar alguno.

Una vez definido el conjunto de capas de la nueva red, Código 5, será el momento de reentrenarla para este caso de estudio. Con este fin, Matlab ofrece una serie de opciones de entrenamiento que se pueden establecer a partir de la función trainingOptions.

```
3. %% Red pre-entrenada
4.
5.       net = alexnet;
6.       net.Layers
7.
8. %% Capas de la Red y Opciones para Re-train
9.
10.            inputlayer =
   imageInputLayer([735 735 3],'Name','input');
11.            % La Capa 1 Define las Dimensiones de Entrada
   (Modificamos)
12.      % Capas Extraídas de AlexNet
13.      layersTransfer = net.Layers(2:end-9);
14.      % Capas Dropout con Probabilidad de 50%
15.      drop6 = dropoutLayer(0.5,'Name','drop6');
16.      drop7 = dropoutLayer(0.5,'Name','drop7');
17.      % Capas Relu
18.      relu6 = reluLayer('Name','relu6');
19.      relu7 = reluLayer('Name','relu7');
20.
21.      Layers = [
22.            inputlayer
23.            layersTransfer
24.            fullyConnectedLayer(200,'name','fc1')
25.            relu6
26.            drop6
27.            fullyConnectedLayer(200,'name','fc2')
28.            relu7
29.            drop7
30.            fullyConnectedLayer(13,'name','fc3')
31.            softmaxLayer
32.            classificationLayer];
```

CÓDIGO 5. CAPAS QUE COMPONEN LA NUEVA RED

3.5.2.3. Opciones de Entrenamiento

Todas las modificaciones aplicadas sobre AlexNet implican que se haga necesario hacer un entrenamiento de la nueva red. De forma que esta pueda aprender las características que diferencian las trece clases de objetos con las que cuenta el conjunto de datos del trabajo desarrollado.

La creación de un conjunto de opciones para la formación de una red se debe realizar conforme a una serie de requisitos, y para ello, se ha utilizado la función *trainingOptions* que ofrece Matlab en la *Neural Network Toolbox*.

El problema de aprendizaje en las redes neuronales se formula en términos de la minimización de la función de error o pérdida. Esta función estará formada por dos términos, uno con el que se evalúa como se ajusta la salida de la red al conjunto de datos del que se dispone, y otro que se utiliza para evitar el sobreaprendizaje de la misma, denominado término de regularización.

El Descenso del Gradiente es el algoritmo de entrenamiento utilizado para este caso de estudio, siendo el más simple y extendido entre los métodos de entrenamiento para este tipo de redes.

El valor que tome la función de error dependerá completamente de los pesos sinápticos entre neuronas y de los bias o sesgos asociados a estas, que, como ya se ha comentado en el apartado ***2.2.1 Red Neuronal***, se denominarán como $'w'$. Por lo tanto, para construir el punto w_{i+1} a partir de w_i, este método hace uso del vector gradiente trasladando este punto en la dirección de entrenamiento d_i, donde v_i se corresponde con la tasa de aprendizaje tal que:

$$w_{i+1} = w_i - d_i = w_i - g_i v_i \,, \qquad \text{siendo } v_i > 0 \qquad [16]$$

La tasa de aprendizaje, denominada como Learning-Rate, es un parámetro configurable que indicará al optimizador hasta dónde debe mover los pesos en la dirección del gradiente, de tal forma que en cada iteración el algoritmo tratará de minimizar la función de pérdida utilizando lotes de imágenes de un tamaño que se haya establecido, siendo una época el total de iteraciones necesarias para utilizar todos los lotes que componen el set de imágenes de entrenamiento.

En el caso de que se comprobase el valor de la pérdida para cada uno de los w_i la gráfica resultante sería una $f(w_i)$ convexa como la mostrada en la Figura 31, donde existe un mínimo que se corresponde con el punto en el que la función de pérdida converge. Así pues, una vez escogido un valor inicial, entrará en juego el parámetro de tasa de aprendizaje, de tal forma que, si este se fija a un valor muy pequeño, la variación del valor del punto en la gráfica cambiará muy lentamente, y por lo tanto, el proceso de aprendizaje llevará mucho tiempo hasta alcanzar ese valor mínimo.

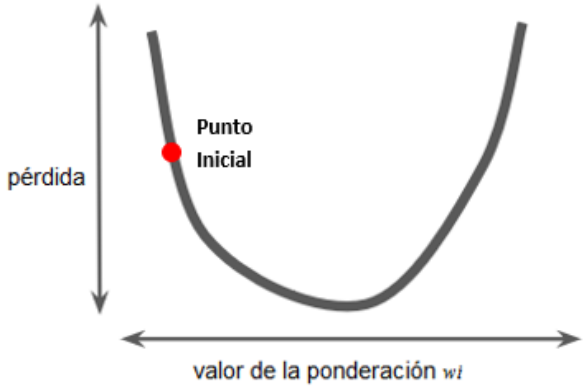


FIGURA 31. PÉRDIDA VS PESOS

En el caso contrario, si se escoge un valor demasiado grande, existirá la posibilidad de sobrepasar el punto de convergencia y que el valor resultante se encuentre rebotando de un lado al otro de la función sin llegar a converger, tal y como se muestra en la Figura 32.

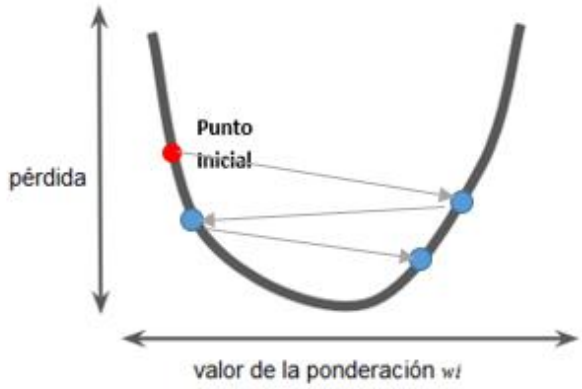


FIGURA 32. EFECTOS DE UNA TASA DE APRENDIZAJE DEMASIADO GRANDE

Por lo tanto, es importante que este parámetro se ajuste a un valor apropiado que optimice el aprendizaje de la red lo máximo posible. Para ello, se proporciona una serie de propiedades dentro de la función tratada en este apartado, siendo posible:

1. Establecer un valor inicial (parámetro *InitialLearnRate*), y actualizarlo cada cierto número de épocas (parámetro *LearnRateSchedule*) multiplicándolo por un factor fijado (parámetro *LearnRateDropFactor*).

2. Establecer como valor inicial aquel que haya resultado óptimo para el tipo de red concreta que se tenga y mantenerlo de forma constante. En cuyo caso será necesario comprobar cómo se comporta la red variando el parámetro en cuestión.

Para este caso de estudio concreto, se han obtenido los resultados expuestos en el Gráfico 2 y el Gráfico 3, siendo 10^{-4} el valor más adecuado para la red diseñada.

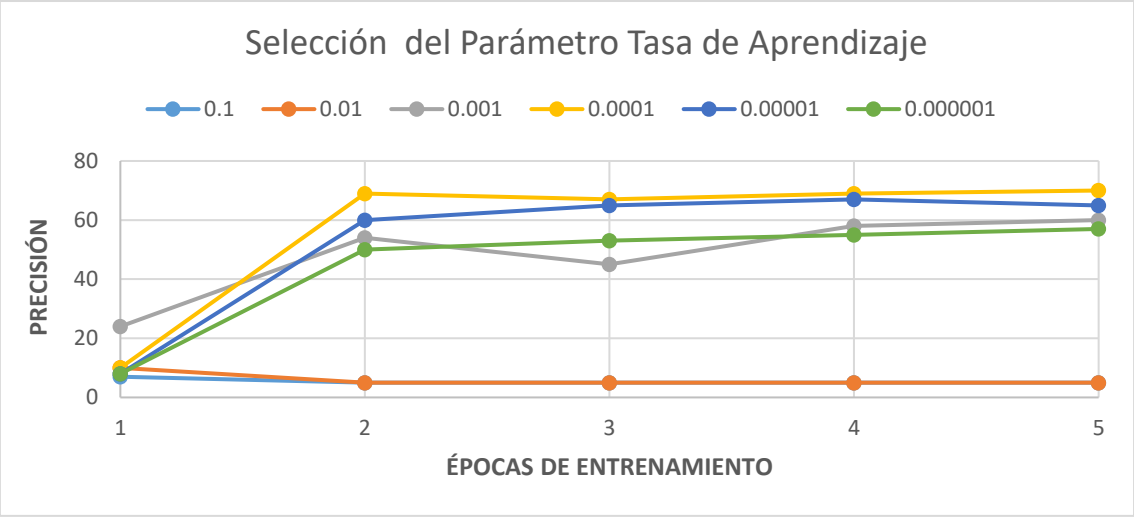


GRÁFICO 2. APRENDIZAJE DE LA RED EN FUNCIÓN DEL PARÁMETRO LEARNING-RATE. PRECISIÓN

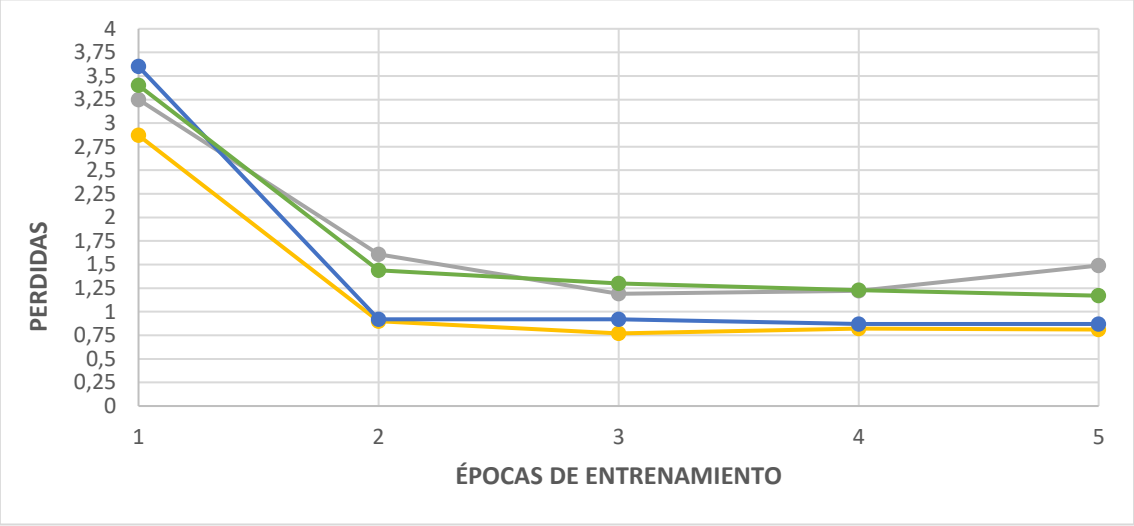


GRÁFICO 3. APRENDIZAJE DE LA RED EN FUNCIÓN DEL PARÁMETRO LEARNING-RATE. PERDIDAS

Como ya se ha comentado, en una época se evalúa el set completo de imágenes de entrenamiento por lotes de un tamaño fijado. Igual que en el caso de la tasa de aprendizaje, se tendrá un número de épocas específico con el que se obtendrán los resultados óptimos de la clasificación y con las que se completará el entrenamiento de la red. Dependiendo de los datos que se deseen evaluar y el tipo de red, este número variará considerablemente.

El objetivo que se persigue al realizar el entrenamiento de la red es que esta sea capaz de establecer correctamente en la salida correspondiente un dato de entrada que no se haya utilizado para el entrenamiento de la misma, o lo que se conoce comúnmente como la capacidad de generalización de la red.

De manera que, si se establece un número de épocas de entrenamiento demasiado grande con el fin de alcanzar una precisión suficientemente buena en el sistema, se correrá el riesgo de que se produzca un sobreajuste de la red a los datos de entrenamiento perdiendo su capacidad de generalización y, por tanto, no siendo capaz de clasificar de forma correcta una nueva imagen.

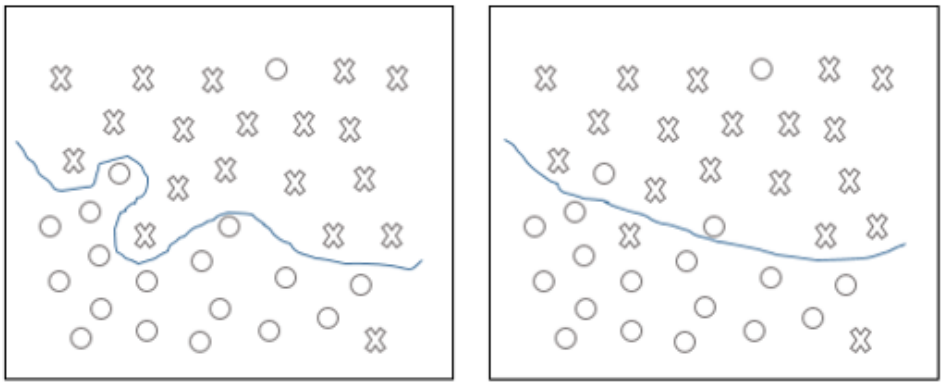


FIGURA 33. SOBRE-ENTRENAMIENTO VS BUENA CAPACIDAD DE GENERALIZACIÓN

Este fenómeno de sobreajuste se conoce como Overfitting o sobre-entrenamiento, y se podrá evitar haciendo uso de una serie de parámetros que aporta Matlab en la función tratada en este apartado, para realizar una parada temprana durante el entrenamiento de la red.

En primer lugar, es necesario realizar una división del set de imágenes como la mostrada en la Figura 34, de tal forma que quede particionado en tres subconjuntos de datos.

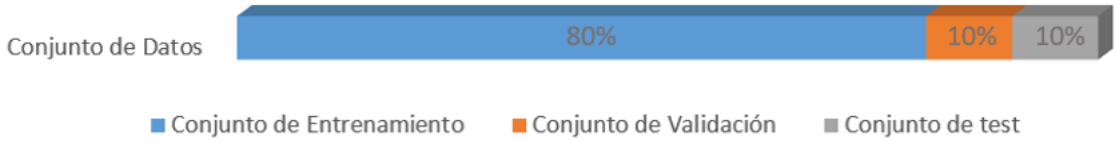


FIGURA 34. DIVISIÓN DEL CONJUNTO DE DATOS

Para saber cuándo se debe detener el entrenamiento de la red se ha utilizado el subconjunto de datos de validación, el cual equivale a un 10% del total de las imágenes. Este conjunto de muestras será evaluado cada cierto número de iteraciones durante el entrenamiento, hasta el momento en el que la perdida obtenida en el conjunto de validación comience a crecer tal y como se muestra en la Figura 35.

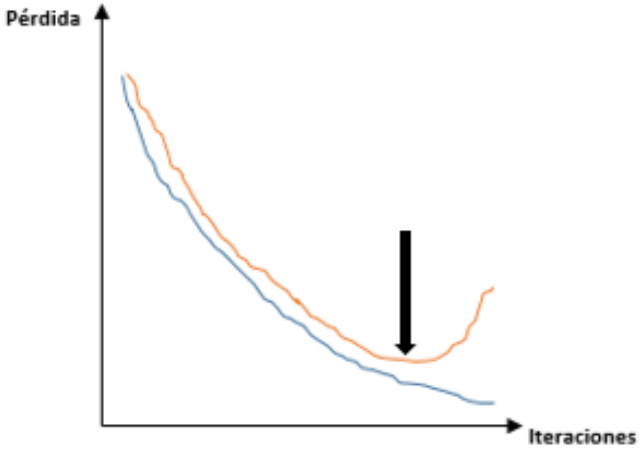


FIGURA 35. EVOLUCIÓN DEL ERROR CUANDO SE PRODUCE SOBRE ENTRENAMIENTO

El proceso de parada temprana utilizará los parámetros explicados a continuación para detener el entrenamiento en el punto en el que la red comienza a ajustarse en exceso a los datos de entrenamiento, que se corresponde con el punto señalado en la ilustración anterior.

- *MaxEpochs.* Se indicarán el número máximo de épocas de entrenamiento de la red como un número entero positivo, no siendo necesario que la red alcance dicho número.

- *ValidationData.* El subconjunto de datos de validación, que equivale en este caso a un 10% del total de las imágenes.

- *ValidationFrequency.* La frecuencia con la que se validará el entrenamiento de la red con el subconjunto de datos mencionado anteriormente. En este caso se ha decidido que esta se haga al finalizar cada época.

- *ValidationPatience.* Número de veces que la perdida en el set de validación puede ser mayor o igual a la obtenida en el punto anterior antes de realizar la parada temprana del entrenamiento, expresado como un entero positivo.

Para el caso de estudio planteado en este trabajo, ha sido necesario establecer el valor de *ValidationPatience* teniendo en cuenta que los datos tratados presentan puntos verificados en los que el error o la pérdida es superior al del punto anterior, como se muestra en la Figura 36.

Este suele establecerse en un valor de entre 10 y 100. No obstante, no hay un mecanismo para escoger un valor óptimo, si no que la elección del mismo dependerá del número de épocas que se necesite para entrenar una red específica, del estado de los datos de trabajo y de la cantidad de validaciones que se lleven a cabo por época.

Por otro lado, es necesario mencionar que será preferible excederse al establecer el valor de este parámetro. Si es demasiado grande, el valor final de pérdida y precisión que se obtenga va a suponer variaciones despreciables respecto al caso óptimo, mientras que, si se establece un valor demasiado pequeño, podría detenerse el entrenamiento siendo posible mejorar los resultados de forma considerable.

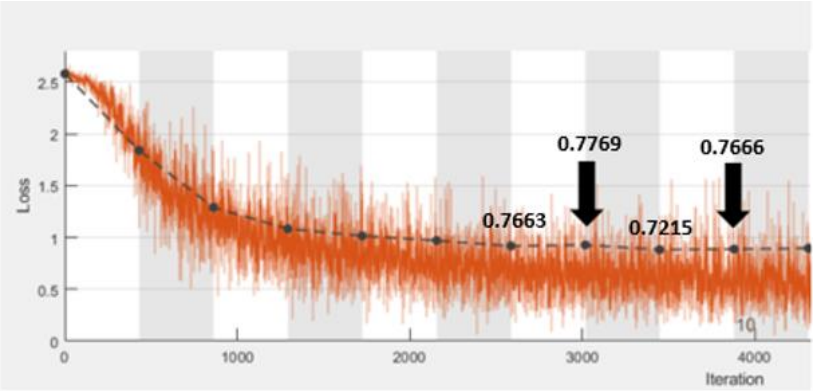


FIGURA 36. PERDIDA DURANTE EL ENTRENAMIENTO DE LA RED

La función utilizada para establecer las opciones de entrenamiento de la red quedará conformada tal y como se muestra en el fragmento de Código 6, y unido al conjunto de capas modificado y a las imágenes de entrenamiento, se procederá a realizar el re-entrenamiento de la nueva red para los datos disponibles en este trabajo.

```
1.      %% Opciones de Entrenamiento
2.      % Número de Imágenes de Entrenamiento
3.      trainImages = length(imdsPolvorasTraining.Files);
4.      % Tamaño de Lote
5.      miniBatchSize = 10;
6.      % Validaciones por Época
7.      numValidationsPerEpoch = 1;
8.      validationFrequency
          = floor(trainImages/miniBatchSize/numValidationsPerEpoch);
9.      % Clases de las Imágenes de Validación
10.       ValLabels=imdsPolvorasVal.Labels;
11.       % Opciones de Entrenamiento de la Nueva Red
12.       options = trainingOptions('sgdm',...
13.             'InitialLearnRate',1e-4,...
14.             'MaxEpochs',50, ...
15.             'MiniBatchSize',miniBatchSize,...
16.             'Momentum', 0.8,...
17.             'VerboseFrequency',validationFrequency,...
18.             'ValidationData',{imdsPolvorasVal,ValLabels},...
19.             'ValidationFrequency',validationFrequency,...
20.             'ValidationPatience',10,...
21.             'Plots','training-progress');
22.
23.       %% Re-train red para Conjunto de Sets
24.
25.       MyNet = trainNetwork(imdsPolvorasTraining,Layers,options);
26.       save ('MyNet', 'MyNet');
```

CÓDIGO 6.OPCIONES DE ENTRENAMIENTO Y NUEVA RED

Una vez se haya finalizado el proceso de entrenamiento de la nueva red, se podrá hacer uso de la misma para clasificar el conjunto de datos disponible. A continuación se va a describir la forma de realizar esta tarea.

3.6 Etapa de Clasificación. MyNet

El proceso de clasificación se llevará a cabo de la misma forma que se hacía en el apartado ***3.4 Etapa de Clasificación. AlexNet***.

Para ello, se ha usado la función *classify* cuyos parámetros de entrada se corresponden con la nueva red diseñada y el almacén de datos utilizado para administrar las imágenes de test disponibles.

La única particularidad con respecto al proceso seguido para la red pre-entrenada AlexNet, es que, en este caso, ha sido necesario reducir el tamaño de los lotes de imágenes con los que se va a trabajar, tal y como se puede apreciar en el fragmento de Código 7. Con esta modificación se pretende solventar posibles problemas de memoria que puedan surgir por el hecho de estar tratando con imágenes de dimensiones mayores, lo que conlleva a una mayor cantidad de datos con los que la red debe trabajar.

```
1. [predictedLabels,MatrixResult] =classify(MyNet,imdsPolvorasTest,
   'MiniBatchSize', miniBatchSize);
```

CÓDIGO 7. PREDICCIONES DE ALEXNET PARA NUESTROS DATOS DE TEXT

La ejecución de la función dará como argumentos de salida las etiquetas o clases a las que la red considera que pertenece esa imagen, esta vez, dentro de los 13 tipos de gránulos con los que se cuenta en el trabajo, y la matriz NxM de pesos, donde N se corresponde con el número de muestras o imágenes evaluadas, y M con el número de clases de objetos como ya se ha mencionado.

Como se mostrará en el capítulo ***4. Evaluación***, la clasificación realizada con esta nueva red presentará mejores resultados con respecto a los obtenidos con la red AlexNet. Pero, ¿qué ocurriría si no se dispone de un equipo lo suficientemente potente para desarrollar o manipular clasificadores de redes neuronales artificiales?

A continuación, se incluye un método alternativo con el que llevar a cabo la etapa de clasificación, en el que se realizará una extracción de características de las imágenes por medio de distintos métodos, para posteriormente entrenar un clasificador SVM, explicado en el apartado ***2.4. Máquina de Soporte Vectorial***.

3.7 Etapa de Extracción de Características

Con la finalidad de entrenar un clasificador, existen diversos métodos y algoritmos para realizar la extracción de características sobre un conjunto de imágenes.

En este capítulo, por un lado, se va presentar la forma de realizar este proceso haciendo uso de las CNN utilizadas en el trabajo, y por otro lado, se mostrará la alternativa de utilizar características concretas, como por ejemplo el color de los objetos en las imágenes, empleando algunos extractores con el algoritmo Bag of Features.

3.7.1 Extracción de Características con AlexNet y MyNet

Matlab ofrece la posibilidad de extraer las salidas que se producen en cada una de las capas que componen una red neuronal artificial, de tal forma que, si se desea ver las transformaciones que van sufriendo las imágenes al atravesarla, el usuario puede comprobarlo haciendo uso de la función *activations*.

Esta función se ha utilizado con la finalidad de extraer las salidas que presenta la última capa totalmente conectada de la red, características obtenidas de las clases de gránulos que se encuentran en el set de imágenes para entrenar el clasificador de máquina de soporte vectorial.

Para ello, la función requiere como parámetros: la red neuronal artificial, el conjunto de imágenes del que se desea obtener las características, y la capa de la red de la cual se desean extraer, tal y como se puede apreciar en el fragmento de Código 8.

Además, podrán especificarse una serie de parámetros opcionales, como el formato en el que establecer los valores obtenidos, o el tamaño de lote de imágenes, que supondrá un mayor uso de memoria cuanto mayor sea su valor.

El resultado de aplicar esta función sobre la última capa totalmente conectada es una matriz $M\ x\ N$, donde M se corresponde con el número de clases distintas de objetos, y N con el número de imágenes que se desean evaluar.

```
1. featureLayer = 'fc3';    % Capa de Extracción
2.
3. trainingFeatures = activations(MyNet, imdsPolvorasTraining,...
   featureLayer, 'MiniBatchSize', 32, 'OutputAs', 'columns');
4. save('trainingFeatures', 'trainingFeatures');
5.
6. testFeatures = activations(MyNet, imdsPolvorasTest,...
   featureLayer, 'MiniBatchSize', 32);
7. save('testFeatures', 'testFeatures');
```

CÓDIGO 8. CARACTERÍSTICAS EXTRAÍDAS PARA EL SET DE ENTRENAMIENTO Y DE TEST

3.7.2 Extracción de Características con Bag of Features

El algoritmo Bag of Features o bolsa de palabras, se ha utilizado para obtener un conjunto de características que representen cada clase de objetos tal y como se ha explicado en el apartado ***2.3. Algoritmo Bag of Features***.

La función disponible en Matlab aplicada para obtener el diccionario visual, conocida como *bagOfFeatures*, utiliza por defecto un tamaño de 500 como el número de palabras visuales.

No existirá un método óptimo para establecer el valor de dicho parámetro, por lo que será necesario realizar una validación experimental, probando distintos valores para el mismo, y comprobando mediante el clasificador la opción que mejores resultados obtenga bajo un conjunto de datos de test dado. No obstante, normalmente la cantidad de palabras visuales utilizada rondará valores mayores de 1000, consiguiendo obtener una representación de las imágenes más discriminativa y por lo tanto, más robusta.

En este trabajo, se han utilizado dos opciones distintas para conformar la bolsa de características. En primer lugar, se ha empleado la opción por defecto ofrecida por la función, que realiza un proceso divido en dos fases:

1. En la primera fase, extrae características SURF de todas las imágenes que forman el set de entrenamiento. Para ello, este descriptor parte de tres etapas básicas:

    I. La detección de escala.

    II. La detección de la orientación. Analizando el contenido local en una circunferencia local al punto de interés, para detectar cual es la orientación y la estructura geométrica que hay en la imagen en ese punto.

    III. La descripción. Vector que describirá las particularidades de los cambios locales encontrados en ese punto.

2. En una segunda fase, construye el vocabulario visual reduciendo el número de características utilizadas a la cantidad establecida por el parámetro k del algoritmo de agrupamiento k-means explicado en ***2.3. Algoritmo Bag of Features***.

En segundo lugar, se ha usado un algoritmo a través del cual se pueden extraer las características de color de una imagen determinada. Este extractor personalizado utiliza una función con la cual se convertirán las imágenes RGB al espacio de color L*a*b.

Este espacio de color va a permitir cuantificar fácilmente las diferencias visuales entre colores tal que:

L → Luminosidad

a → Coordenadas rojo/verde
Donde +a indica rojo y –a verde

b → Coordenadas amarillo/azul
Donde +b indica amarillo y –b azul

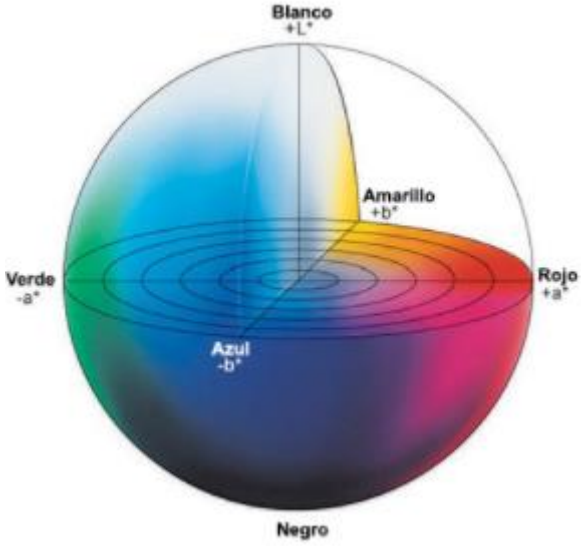


FIGURA 37. ESPACIO DE COLOR L*A*B

El proceso seguido consta de las siguientes operaciones:

1. Se comprobará si la imagen se encuentra en el espacio de color RGB, y posteriormente será convertida al espacio L*a*b utilizando la función *rgb2lab*.

2. Se tomará la imagen en bloques de un tamaño establecido, extrayendo el color promedio de cada uno de ellos y su ubicación (x y) dentro de la imagen, para posteriormente concatenarlos en un vector. De manera que, para que dos imágenes tengan características de color parecidas, se tendrán en cuenta tanto el color como la distribución espacial del mismo.

Seguidamente, y una vez se haya extraído la información de las imágenes para ambos casos (redes neuronales convolucionales y algoritmo bag of features), se explicarán los métodos empleados para entrenar el clasificador de máquina de soporte vectorial.

3.8 Etapa de Clasificación. SVM

3.8.1 Redes AlexNet y MyNet

En Matlab se dispone de la función *fitcecoc* para desarrollar modelos multiclase para máquinas de vectores u otro tipo de clasificadores. De manera que, en este caso, para obtener el clasificador SVM es necesario incluir como parámetros de esta función, la tabla de características del set de entrenamiento obtenida de la última capa totalmente conectada de la red diseñada y las etiquetas de cada uno de estos datos.

Los parámetros u opciones de mayor relevancia para desarrollar el clasificador de forma adecuada, visibles en el fragmento de Código 9, serán expuestos seguidamente:

- *'Learners'* se utiliza para seleccionar el tipo de clasificador que se desea generar, entre los cuales se encuentra el SVM como la opción establecida por defecto.

- El tipo de codificación de los datos, donde se pueden encontrar varias opciones de entre las cuales se ha seleccionado *'onevsall'*, o la comparación de cada clase con el resto. Esta supondrá entrenar un clasificador por cada clase, y ha resultado como la más adecuada para el caso de estudio.

- La estructura o dimensiones que tendrán los datos de predicción, seleccionando si esta se lleva a cabo por columnas o por filas por medio del parámetro *'ObservationsIn'*.

```
1.  % Entrenamiento de un Clasificador Multiclase SVM
2.  SVMclassifierCNN = fitcecoc(trainingFeatures, trainingLabels,
    'Learners', 'svm', 'Coding', 'onevsall', 'ObservationsIn',
    'columns');
3.  save('SVMclassifierCNN', 'SVMclassifierCNN');
```

CÓDIGO 9. ENTRENAMIENTO DEL CLASIFICADOR SVM PARA CNN

3.8.2 Algoritmo Bag of Features

La forma de entrenar un clasificador multiclase de máquina de soporte vectorial utilizando la bolsa de características obtenida a través del algoritmo Bag of Features, se puede realizar utilizando la función *trainImageCategoryClassifier* mostrada en el fragmento de Código 10, que Matlab ofrece en la Toolbox *Computer Vision System.*

Esta función requiere como parámetros de entrada el conjunto de imágenes de entrenamiento, como un almacén de datos creado a partir de la función *imageDatastore*, y la bolsa de características, especificado como un objeto que va a contener un vocabulario de los descriptores de características representativas de cada categoría de imágenes.

```
1. % Creación y entrenamiento de un clasificador SVM
2. SVMcategoryClassifierBoF =
   trainImageCategoryClassifier(imdsPolvorasTraining,
   bagOFeatures);
3. save('SVMcategoryClassifierBoF', 'SVMcategoryClassifierBoF')
```

CÓDIGO 10. ENTRENAMIENTO DEL CLASIFICADOR SVM PARA BOF

Una vez se haya completado el entrenamiento de cada uno de los clasificadores empleados en el trabajo, se comenzará con la explicación de la etapa de test, donde cada uno de los mismos serán puestos a prueba para comprobar cual resulta más eficiente sobre nuestra base de datos.

3.9 Etapa de Testeo y Predicción

En este punto entrará en juego el conjunto de datos formado por el set de test para probar la eficiencia de los clasificadores utilizados, comprobando a que clase considera que pertenece cada muestra evaluada.

Por lo tanto, como se comentó en los apartados ***3.4 Etapa de Clasificación. AlexNet*** y ***3.6 Etapa de Clasificación. MyNet***, la forma de obtener la predicción de las etiquetas para la etapa de clasificación utilizando las redes neuronales convolucionales, se ha hecho con la función *classify*.

En el caso de la clasificación con la máquina de soporte vectorial, se ha utilizado la función *predict*, *d*isponible en Matlab para obtener la predicción de la clase a la cual se considera que pertenece una nueva imagen de test. Esta función requiere como parámetros de entrada el clasificador SVM entrenado y la tabla de características de las nuevas imágenes que se desea clasificar, devolviendo un array de filas $Mx1$ en el que cada una de las mismas se corresponde con el tipo de clase predicha para cada muestra del set.

Una vez se hayan calculado las etiquetas de predicción, se tabularán los resultados en una matriz de confusión, con la que se pueda observar la precisión que tienen tanto las redes AlexNet y MyNet como el SVM para clasificar cada uno de los tipos de imágenes del problema tratado.

La toolbox *Statistics and Machine Learning* [*Ref.9*] ofrece la función *confusionmat* para obtener dicha tabla de confusión. Incluye como parámetros de entrada, por un lado, las etiquetas reales que posean las imágenes del set de test, y por otro lado, las etiquetas de predicción obtenidas anteriormente, tal y como se muestra en el fragmento de Código 11.

```
% Tabular los Resultados en una Matriz de Confusión
confMatCNN = confusionmat(testLabels, predictedLabels);
save('confMatCNN', 'confMatCNN');


% Convirtiendo la Matriz de Confusión en Porcentajes
confMatCNNPercent = (bsxfun(@rdivide,...
        confMatCNN, sum(confMatCNN,2)) * 100);
save('confMatCNNPercent', 'confMatCNNPercent');
```

CÓDIGO 11. MATRIZ DE CONFUSIÓN

La matriz resultante de aplicar esta función posee dimensiones NxN, siendo N el número de clases distintas que forman el conjunto de datos. Muestra en la diagonal principal de la misma, la cantidad de imágenes que se han clasificado correctamente, y las mal clasificadas en el resto de posiciones de la matriz según el tipo de clase con la que haya sido confundida esa muestra concreta, Tabla 2.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	27	0	0	0	0	0	0	0	0	0	0	0	4
2	0	18	10	0	0	0	0	0	0	0	3	0	0
3	0	3	25	0	0	0	0	0	0	0	3	0	0
4	0	0	0	53	0	0	0	0	0	0	0	0	0
5	0	0	0	0	43	0	0	0	0	0	0	0	0
6	0	0	0	0	0	57	0	0	0	0	0	3	0
7	0	0	0	0	0	1	47	0	0	0	0	0	0
8	0	0	0	0	0	0	0	51	0	0	0	0	0
9	0	0	0	0	0	0	0	0	31	0	0	0	0
10	0	0	0	0	0	0	0	0	0	35	0	0	0
11	0	1	0	0	0	0	1	0	0	0	52	0	0
12	0	0	0	0	0	14	1	0	0	1	0	25	0
13	23	0	0	0	0	0	0	0	0	0	0	0	8

TABLA 2. MATRIZ DE CONFUSIÓN EJEMPLO

Además, existirá la posibilidad de expresar los valores obtenidos en la matriz de confusión como porcentajes, Tabla 3, de manera que se pueda apreciar la precisión que tiene el clasificador para reconocer según qué tipo de objetos.

Una forma de hacerlo puede ser utilizando la función *bsxfun*, dentro del conjunto de funciones aritméticas que ofrece Matlab. La suma total de muestras se divide entre las que se haya clasificado en cada tipo y se multiplica dicho valor por el factor 100, fragmento de Código 11.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	93.5484	0	0	0	0	0	0	0	0	0	0	0	6.4516
2	0	74.1935	19.3548	0	0	0	0	0	0	0	6.4516	0	0
3	0	9.6774	77.4194	0	0	0	0	0	0	0	12.9032	0	0
4	0	0	0	100	0	0	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0	0
6	0	0	0	0	0	95	0	0	0	0	0	5	0
7	0	0	0	0	0	8.3333	91.6667	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0
9	0	0	0	0	0	0	0	0	100	0	0	0	0
10	0	0	0	0	0	0	0	0	0	100	0	0	0
11	0	1.8519	0	0	0	0	1.8519	0	0	0	96.2963	0	0
12	0	0	0	0	0	36.5854	4.8780	0	0	0	0	58.5366	0
13	74.1935	0	0	0	0	0	0	0	0	0	0	0	25.8065

TABLA 3. MATRIZ DE CONFUSIÓN EJEMPLO CONVERTIDA A PORCENTAJES

Por último, se ha añadido la posibilidad de almacenar aquellas muestras que se hayan clasificado erróneamente. Esto puede ser útil en caso de tener elementos defectuosos en el conjunto de datos que pudieran ser descartados para posibles actualizaciones en el sistema, o para localizar indicios o posibles características en común que hagan ver por qué su clasificación ha resultado ser incorrecta.

El proceso seguido, expuesto en el fragmento de Código 12, consiste en contabilizar el número de muestras que compone el set de test, comprobar para cada una de ellas el tipo de gránulo al que pertenecen, comparándolo con aquel que haya predicho el clasificador para esa muestra concreta, y por último, almacenar en una ruta específica las imágenes de aquellos casos en los que uno no se corresponda con el otro.

```
1. NumGranulosTest = length(imdsPolvorasTest.Files);
2. for     n=1:NumGranulosTest
3.
4.       if  predictedLabels(n) ~= testLabels(n)
5.
6.            RutaIm = char(imdsPolvorasTest.Files(n));
7.            copyfile (RutaIm,...
         'C:\Users\Noelia\Desktop\Red\Recortes\Errores1')
8.       end
9.
10.
11.      end
```

CÓDIGO 12. ALMACENAMIENTO DE IMÁGENES CLASIFICADAS DE FORMA INCORRECTA

Con la finalización de esta etapa, se tendrán varios sistemas de clasificación que serán comparados y evaluados a través de una serie de pruebas, explicadas en el capítulo ***4. Evaluación***.

Seguidamente, se va a exponer el desarrollo de una interfaz gráfica para utilizar el clasificador que haya presentado mejores resultados, facilitando al usuario final el mecanismo para poder utilizar el sistema con nuevas imágenes.

3.10 Interfaz Gráfica de Usuario

El objetivo de la realización de una GUI en este trabajo, es el de presentar una ventana de aplicación que el usuario final pueda manejar fácilmente para revisar algunos datos y para realizar clasificaciones de nuevas imágenes.

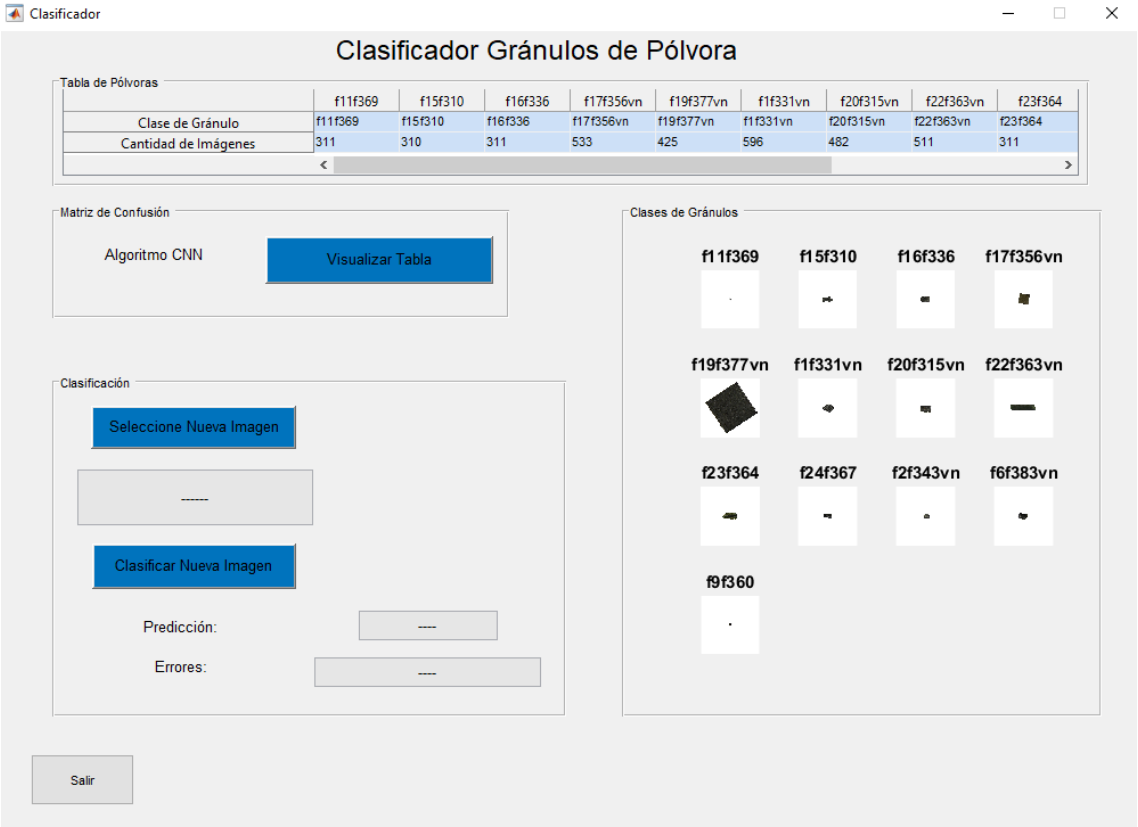


FIGURA 38. VENTANA DE LA APLICACIÓN

La interfaz gráfica diseñada tendrá la estructura expuesta en la Figura 38 y se ha dividido en cuatro bloques:

- El primer bloque, está formado por una tabla informativa donde se muestran los tipos de gránulos que se han utilizado para la clasificación, y la cantidad de imágenes que contiene cada clase de los mismos.

- El segundo bloque, informa del tipo de algoritmo utilizado a través de un cuadro de texto fijo, y ofrece la posibilidad de visualizar la matriz de confusión mostrada en la Figura 39. Para ello, se ha hecho uso de un elemento de tipo botón configurado para abrir una nueva ventana, en la que se muestra la tabla de porcentajes de predicción que se ha obtenido para el conjunto de datos de test con el que se ha evaluado el sistema.

Convolutional Neural Network (CNN):

	f11f369	f15f310	f16f336	f17f356vn	f19f377vn	f1f331vn	f20f315vn	f22f363vn	f23f364	f24f367	f2f343vn	f6f383vn
f11f369	86.5323	0	0	0	0	0	0	0	0	0	0	0
f15f310	0	78.0645	14.1935	0	0	1.9355	0.6452	0	0	1.2903	3.2258	0.6452
f16f336	0	17.2782	73.7500	0	0	1.2903	1.2500	0	0.6250	0	5.8065	0
f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0
f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0
f1f331vn	0	2	1	0	0	82.3333	2.3333	0	0	0	1.6667	10.6667
f20f315vn	0	0	0.8163	0	0	2.9167	92.9507	0	0	1.2330	0	2.0833
f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0
f23f364	0	0.6452	0	0	0	0	0.6452	0	98.7097	0	0	0
f24f367	0	1.1429	0	0	0	0	1.7143	0	0	94.2857	1.7143	1.1429
f2f343vn	0	1.8519	2.2222	0	0	0	1.1111	0	0	0.7407	92.5926	0
f6f383vn	0	0	0	0	0.9756	16.5854	8.2927	0	0	6.8293	1.4634	65.8537
f9f360	35.4839	0.6452	0	0	0	0	0	0	0	0	4.5161	0

Cerrar

FIGURA 39. TABLA DE CONFUSIÓN

- El tercer bloque, se corresponde con la visualización de una imagen aleatoria de cada clase de gránulo disponible, definido como un elemento de tipo axes dentro de un panel en el que se establece una distribución de 4x4, es decir, cuatro imágenes por fila y cuatro por columna.

- El cuarto bloque está compuesto por distintos elementos:

  - Botones
  - Cuadros de texto editable
  - Cuadros de texto estático
  - Ejes

El primer botón puede utilizarse para seleccionar el nuevo elemento que se desea clasificar. Con la función *uigetfile* se va a presentar un cuadro de diálogo de la carpeta del proyecto donde se tienen almacenadas las imágenes, incluyendo la condición de mostrar únicamente los archivos de tipo .jpg y .png para que se pueda seleccionar la nueva foto, fragmento de Código 13. El valor devuelto se corresponderá con el nombre y la ruta completa del archivo. Esta información va a quedar impresa en el primer cuadro de texto editable *edit1*, mostrando además, la imagen seleccionada.

```
1.      [LaImagenSeleccionada,rutaImagenSeleccionada] = uigetfile({
        '*.jpg'; '*.png';'Imágenes (*.jpg, *.png)'},...
2.      'Selecciona la imagen que quieres clasificar',...
3.      'MultiSelect', 'off');
4.
5.      if isfloat(LaImagenSeleccionada)
6.            string3 = '¡Selecciona Primero una Imagen!';
                % Mensaje de Error
7.            set(handles.edit3, 'String', string3);
8.            set(handles.edit1, 'String', '----');
              % Sin Mensaje de Ruta
9.            set(handles.edit2, 'String','----');
              % Sin Mensaje de Predicción
10.
11.           set(get(handles.axes3,'children'),'visible','off')
              % Ocultar Imagen
12.     else
13.           string1 = rutaImagenSeleccionada;
14.           set(handles.edit1, 'String', string1);
              % Mensaje de Ruta
15.           set(handles.edit3, 'String', '----');
              % Sin Mensaje de Error
16.     end
17.     newImage= fullfile(rutaImagenSeleccionada,...
18.     LaImagenSeleccionada);
19.     LaImagen = imread(newImage);
20.     imshow(LaImagen,'Parent', handles.axes3);
        % Se Muestra Imagen Seleccionada
```

CÓDIGO 13. PROCESO SEGUIDO EN LA OPERACIÓN DE SELECCIÓN DE NUEVA IMAGEN

En caso de cerrar la ventana sin seleccionar ningún archivo o de hacer clic en Cancelar, la función devolverá un 0 y se mostrará un mensaje de error en otro de los cuadros de texto editable incluidos en la Figura 38, concretamente el *edit3*.

La segunda parte de este bloque sirve para clasificar la imagen, de manera que accionando un segundo botón:

1. Se comprueba el número de GPUs disponibles.

    a. En caso de que este sea 0, se imprimirá un mensaje de error en la capacidad de cómputo sobre el cuadro de texto editable *edit3*.

    b. En caso contrario, se ejecutará la función desarrollada para obtener la etiqueta de predicción de la imagen seleccionada.

2. Se muestra sobre el cuadro de texto editable *edit2* la etiqueta del tipo de objeto como una cadena de caracteres.

La función con la que se resuelve la clasificación de la nueva imagen recibe el nombre *TipoGranulo*, y utilizando como parámetro de entrada la imagen, realiza la secuencia de instrucciones explicada en apartados anteriores para obtener la predicción de la clase a la que pertenece.

Por último, se establece un cuarto botón con el que se pueda salir de la ventana de aplicación, Código 14, que abre un nuevo cuadro de diálogo para comprobar si realmente se desea cerrar la aplicación.

```
1. selection = questdlg(['Desea Cerrar el
   ' get(handles.figure1,'Name') '?'],...
2. ['Desea Cerrar el ' get(handles.figure1,'Name') '...'],...
3. 'Si','No','Si');
4. if strcmp(selection,'No')
5.         return;
6. end
7. delete(handles.figure1)
```

CÓDIGO 14. CUADRO DE DIÁLOGO PARA CERRAR LA APP

El conjunto de etapas de desarrollo para completar el sistema diseñado con el entorno de programación de Matlab finalizará en este punto. En consecuencia, a partir de este capítulo se va a comenzar con la explicación de una segunda parte del diseño, con la que se ha tratado de replicar el sistema para el entorno de programación de Python.

3.11 Desarrollo del Sistema Sobre Python

A lo largo del documento se han expuesto una serie de opciones de diseño para desarrollar el sistema de clasificación. La opción final, es decir, aquella que haya resultado como la más óptima para la base de datos del trabajo, será el punto de partida para este nuevo apartado.

El primer inconveniente que se va a encontrar al tratar de reproducir el sistema utilizando este lenguaje de programación, es que no se puede importar la red diseñada en Matlab directamente a Python, ya que actualmente no se presenta compatibilidad entre este y el tipo de datos generados con Matlab.

Teniendo esto en cuenta, una opción sería realizar todo el procedimiento empleado en Matlab, es decir, preprocesamiento de las imágenes, diseño y entrenamiento de la CNN y evaluación de la misma. No obstante, y pese a que se parte del impedimento de no poder usar la red directamente, se plantea una alternativa con la que, a partir de algunos datos del sistema diseñado con Matlab, se pueda reducir la cantidad de trabajo que se debe realizar en este punto.

3.11.1 Etapa de Pre-procesamiento

La forma de afrontar el desarrollo de las distintas operaciones que forman esta fase del sistema final, es equivalente a la planteada en Matlab, ***3.3 Etapa de Pre-procesamiento***. Es decir, la estructura seguida estará formada por:

1. Una primera etapa en la que se ha realizado un redimensionado de las imágenes, rellenando de píxeles de fondo hasta la dimensión máxima de 735 localizada en el conjunto de datos.

2. Una segunda etapa en la que, utilizando la imagen transformada a escala de grises, se ha realizado una segmentación del gránulo utilizando operaciones de binarización, dilatación, rellenado, erosión y eliminación de objetos pequeños.

3. Una tercera y última etapa en la que, tras recuperar el área RGB correspondiente al gránulo segmentado, se han orientado todos los objetos de las imágenes a una posición común.

Para ello, se han empleado las bibliotecas PIL, Scipy, Numpy y Matplotlib que ofrecen una serie de subpaquetes con funciones para el tratamiento de imágenes. Asimismo, se puede encontrar documentación en línea donde se presentan una serie de funciones equivalentes en ambos lenguajes [*Ref.10*], algunas de las mismas pertenecientes a las bibliotecas mencionadas.

3.11.2 Etapa de Clasificación y Predicción. MyNet

Anteriormente, se comentó la opción de reducir el trabajo evitando el entrenamiento de la red MyNet en este nuevo lenguaje de programación. Sin embargo, cuando se desee clasificar una nueva imagen, será necesario utilizar la red. Por lo tanto, haciendo uso de la biblioteca Tensorflow, dispuesta de una gran cantidad de funciones para desarrollo de Deep Learning, se recreará la red que se ha diseñado para Matlab.

La red diseñada es una función que contiene todas las capas que la forman, configuradas y definidas convenientemente, para que ante una imagen de entrada el valor devuelto por la misma se corresponda con la etiqueta de predicción oportuna.

Esta no va a contar con una capa de entrada, por lo que el proceso de normalización de las imágenes se realiza restando a la nueva imagen de entrada, la imagen media del set de datos de entrenamiento, que se ha calculado utilizando un bucle *for* sobre este set de imágenes.

Para que la red sea equivalente a la que se entrenó en Matlab, se importarán todas las matrices de pesos y sesgos, tanto de las capas convolucionales como de las totalmente conectadas, y se cargarán según corresponda a cada una de ellas como los valores iniciales de estos parámetros, fragmento de Código 16.

En Python, la biblioteca Scipy dispone de un módulo para cargar datos generados en Matlab, fragmento de Código 15. La función disponible para cargar estos datos generará un diccionario en el que se encuentran contenidos estos arrays, y a los que hay que acceder por medio de una clave que recibe el nombre de la variable.

```
1.  weightsMat = scipy.io.loadmat(R1_Weights1.mat')
2.  biasesMat = scipy.io.loadmat(R1_Bias1.mat')
3.  weights1 = np.copy(weightsMat['R1_Weights1'])
4.  biases = np.copy(biasesMat['R1_Bias1'])
```

CÓDIGO 15. CARGA DE ARCHIVOS CON EXTENSIÓN .MAT

```
1. if weights == 'DEFAULT':    # Primer Acceso a la Red
2.
3.         # Cargo Valores de los Pesos y Sesgos Capas Conv
4.         weights1, biases1 = load_initial_weights(name='conv1')
5.         weights2, biases2 = load_initial_weights(name='conv2')
6.         weights3, biases3 = load_initial_weights(name='conv3')
7.         weights4, biases4 = load_initial_weights(name='conv4')
8.         weights5, biases5 = load_initial_weights(name='conv5')
9.         # Tabla
10.        weights = [weights1,weights2,weights3,weights4,weights5]
11.        biases =  [biases1,biases2,biases3,biases4,biases5]
```

CÓDIGO 16. CARGA DE LAS MATRICES DE PESOS Y SESGOS

En Tensorflow es posible representar cálculos computacionales como grafos, donde cada nodo del mismo, nombrados como ops, realizará cierta operación o representará cierto dato que se requiera durante la etapa de ejecución.

Por lo tanto, el proceso de programación seguido con este sistema se puede estructurar en dos fases: una fase de construcción en la que se definen todos los parámetros y objetos que se quieren emplear, y una fase de ejecución, que utiliza una sesión en la cual se iniciarán todos los parámetros con los valores que se deseen establecer para ellos, fragmento de Código 17.

```
1.      # Definición de Entradas
2.      x = tf.placeholder(tf.float32, (1,735,735,3))
3.      y = tf.placeholder(tf.float32, (1,))
4.
5.      # Definición de Funciones
6.      Modelo = MyNet(x, weights='DEFAULT')
7.      Prediccion = MyNet_Fully(y, weights='DEFAULT')
8.
9.      # Definimos Sesión
10.     sess = tf.InteractiveSession()
11.     # Inicialización de Variables
12.     init = tf.global_variables_initializer()
13.     sess.run(init)
14.
15.     # Ejecución de la Parte 1 Red con la Imagen a Clasificar
16.     MaxPool5_output = sess.run(Modelo,feed_dict={x: imgF})
17.
18.     #Formato Fortran para que la Matriz se Escriba en el Mismo
        Orden que en Matlab
19.
          Input_fully = np.reshape(MaxPool5_output,(1,112896),order='
   F')
20.
21.     # Ejecución de la Parte 2 Red con la Salida de Capa Conv5
        como Entrada
22.     Prediccion_output = sess.run(Prediccion,feed_dict={y:
        Input_fully})
23.       ClasePred = tbLabels[Features_output.argmax()]
```

CÓDIGO 17. SESIÓN TENSORFLOW

A diferencia de Matlab, va a ser necesario redimensionar la salida de la última capa convolucional antes de utilizarla sobre la primera capa totalmente conectada para que exista compatibilidad de dimensiones entre los arrays. No obstante, surge un problema ante la necesidad de cumplir con este requisito. Las funciones que Tensorflow ofrece para realizar esta operación almacenan los datos en un orden de posición distinto al seguido en Matlab, lo que se traduce en que para cada uno de los entornos de programación utilizados, se van a tener datos de salida diferentes pese a haber partido de la misma imagen de entrada y la misma red.

Este contratiempo se va a solucionar utilizando la función *reshape*, incluida dentro de la biblioteca Numpy. Con esta función se realiza un redimensionado en el que, para un array de entrada, se va a obtener un array de salida donde se puede escoger el orden del posicionamiento de los datos.

Sin embargo, durante la ejecución de la CNN se está trabajando con objetos tensores, de manera que, como se puede apreciar en las líneas 6 y 7 del fragmento de Código 17, ha sido necesario dividir el proceso en dos partes para que se pueda extraer el objeto tensor de salida de la capa convolucional como un array de valores, que será redimensionado antes de utilizarlo en la primera capa totalmente conectada.

La predicción de salida de la red obtenida para una nueva imagen se corresponderá con un array 1xN, siendo N el número de clases distintas de objetos, expuesto en la línea 22 del fragmento de Código 17.

La etiqueta del tipo de clase de gránulo al que pertenece la imagen evaluada, se va obtener haciendo uso de la función *argmax*, incluida dentro de la biblioteca de *Numpy*. Esta función devolverá la etiqueta del argumento máximo obtenido en el array de predicción extraído de la red.

A continuación, se va a exponer la forma de realizar la extracción de características de la red para entrenar el clasificador de máquina de soporte vectorial, si se desea emplear esta posibilidad.

3.11.3 Etapa Extracción de Características

Como caso alternativo, y de la misma forma que se hacía en el sistema desarrollado sobre Matlab, se puede optar por utilizar la red como extractor de características para posteriormente entrenar un clasificador SVM.

Para ello, en este caso, aparte de las matrices de pesos y sesgos que se deben importar para recrear la red, es necesario añadir los datos obtenidos como la matriz de características tanto del conjunto de datos de entrenamiento como del conjunto de test.

3.11.4 Etapa de Clasificación y Predicción. SVM

Una vez se hayan importado los datos de las características extraídas de la red diseñada en Matlab, se podrá pasar a la fase de clasificación. Para ello, la biblioteca Sklearn ofrece una serie de funciones para entrenar y validar un clasificador SVM, teniendo en cuenta los parámetros que se utilizaron en Matlab de forma que este sea equivalente en ambos lenguajes.

A continuación, se comprobará que funcione correctamente utilizando el conjunto de datos de test para realizar la predicción de las clases a las que pertenecen.

```
[[27  0  0  0  0  0  0  0  0  0  0  0  4]
 [ 0 23  5  0  0  2  0  0  0  0  1  0  0]
 [ 0  4 22  0  0  1  0  0  0  0  4  0  0]
 [ 0  0  0 53  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 41  0  0  0  2  0  0  0  0]
 [ 0  0  1  0  0 54  1  0  0  0  0  4  0]
 [ 0  0  0  0  0  1 44  0  3  0  0  0  0]
 [ 0  0  0  0  0  0  0 51  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 31  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 35  0  0  0]
 [ 0  2  5  0  0  0  2  0  0  1 44  0  0]
 [ 0  0  0  0  0  5  0  0  0  0  0 36  0]
 [22  0  0  0  0  0  0  0  0  0  0  0  9]]
```

TABLA 4. PREDICCIÓN DEL SVM SOBRE LOS DATOS DE TEST

La matriz de confusión de la **¡Error! No se encuentra el origen de la referencia.** ha sido calculada empleando el módulo *metrics* de la biblioteca mencionada anteriormente tal y como se muestra a continuación, fragmento de Código 18.

```
1. # Tabla de Confusión
2. ConfusionTb = confusion_matrix(Test_Labels, predictedLabels,
3. labels=None, sample_weight=None)
4. # Normalización a datos %
5. ConfusionTbNorm = ConfusionTb /
   ConfusionTb.astype(np.float).sum(axis=1) * 100
6. # Redondeo a 4 decimales
7. ConfusionTbNorm = np.around(ConfusionTbNorm, decimals=4)
```

CÓDIGO 18. TABLA DE RESULTADOS NORMALIZADOS

Cuando se desee clasificar una nueva imagen con el SVM desarrollado en Python, se empleará el mismo proceso que se utilizaba para el caso de la clasificación con la CNN, con la particularidad de extraer el array de valores Mx1, siendo *M* el número de clases distintas de objetos, de la última capa totalmente conectada.

Seguidamente, se detallará la forma de realizar la interfaz de usuario en este entorno de programación.

3.11.5 Interfaz Gráfica de Usuario

El diseño de la interfaz gráfica en el entorno de programación de Python, se llevará acabo con la adaptación o binding *tkinter* de la biblioteca gráfica *Tcl/Tk*. Biblioteca que se podrá encontrar disponible para distintos lenguajes de programación y que además, está considerada un estándar para el desarrollo de GUIs en Python.

El proceso seguido para diseñar la interfaz va a ser considerablemente diferente al de Matlab. No obstante, se ha tratado de replicar en la medida de lo posible.

En primer lugar, será necesario indicar que la metodología a seguir pasa por:

1. Crear una ventana principal. La interfaz se irá construyendo a partir de una primera ventana que se ha definido como:

```
1. ventana=Tk()
2. ventana.geometry('1000x700')
3. ventana.title('Clasificador')
4. ventana.mainloop()
```

CÓDIGO 19. VENTANA PRINCIPAL DE LA GUI

   El formato de la ventana podrá ser diseñado a partir de una amplia variedad de parámetros, algunos de las cuales se muestran en el Código 19, dimensiones del objeto, color, etc.

2. Opciones de diseño. Al igual que en Matlab, *Tkinter* cuenta con diversos elementos de tipo botón, texto, menú, lista, etc. Cada vez que desee añadir un elemento nuevo, será necesario indicar la ubicación del mismo, incluyendo tanto la información del componente superior en el que va a ser añadido, como la posición de este dentro del mismo. Seguidamente se mencionan algunos de los más utilizados.

I. Botones. Al contrario que en Matlab, donde al guardar el archivo .fig se creaba un archivo .mat, que contenía una serie de funciones editables a las cuales se accedía presionando el botón correspondiente. En Python es necesario indicar el proceso que se desea seguir cuando uno de estos elementos es invocado.

El parámetro encargado de llevar acabo esta asignación recibe el nombre de *command*. Este va a contener el nombre de la función que se haya creado para realizar una acción, siendo posible utilizar algunos predefinidos para ciertas operaciones. Por ejemplo, un botón en el que se desea cerrar una ventana se puede implementar de la siguiente forma, fragmento de Código 20.

```
1. BotonCerrar = Button(ventana2, text='Cerrar',command = lambda:
   salir3(ventana2),font=8)
2. BotonCerrar.place(x=20,y=350)
3.
4. def salir3(ventana2):
5.
6.       ventana2.destroy()
```

CÓDIGO 20. BOTÓN PARA CERRAR UNA VENTANA

II. Tablas. No se va disponer de un componente con el que se pueda añadir una tabla a la interfaz. Por lo tanto, una opción puede ser utilizar un elemento o widget de tipo *LabelFrame*, con el cual se podrán agrupar varios objetos relacionados, y uno de tipo *Label*, que se corresponderá con cada una de las piezas que se desea agregar a dicho conjunto.

Existirá la posibilidad de agregar una gran cantidad de opciones en el diseño manejando texto, imágenes o ambas cosas al mismo tiempo.

La ventana de la aplicación diseñada presenta la estructura mostrada en la Figura 40. En total, se ha hecho uso de cuatro ventanas, siendo estas, la ventana principal que contiene el resto de elementos, una segunda ventana donde se muestra la tabla de confusión generada con la predicción de los datos de test, una tercera ventana que abre un cuadro de dialogo para seleccionar la nueva imagen que se desea clasificar, y una cuarta ventana para salir de forma segura de la aplicación.

La ubicación de cada elemento presente en las ventanas, se ha establecido conforme a una serie de divisiones geométricas. Para ello, se han utilizado widgets de tipo *frame*, que sirven básicamente como contenedores de otros elementos, y que se pueden subdividir cuantas veces se desee en regiones más pequeñas.

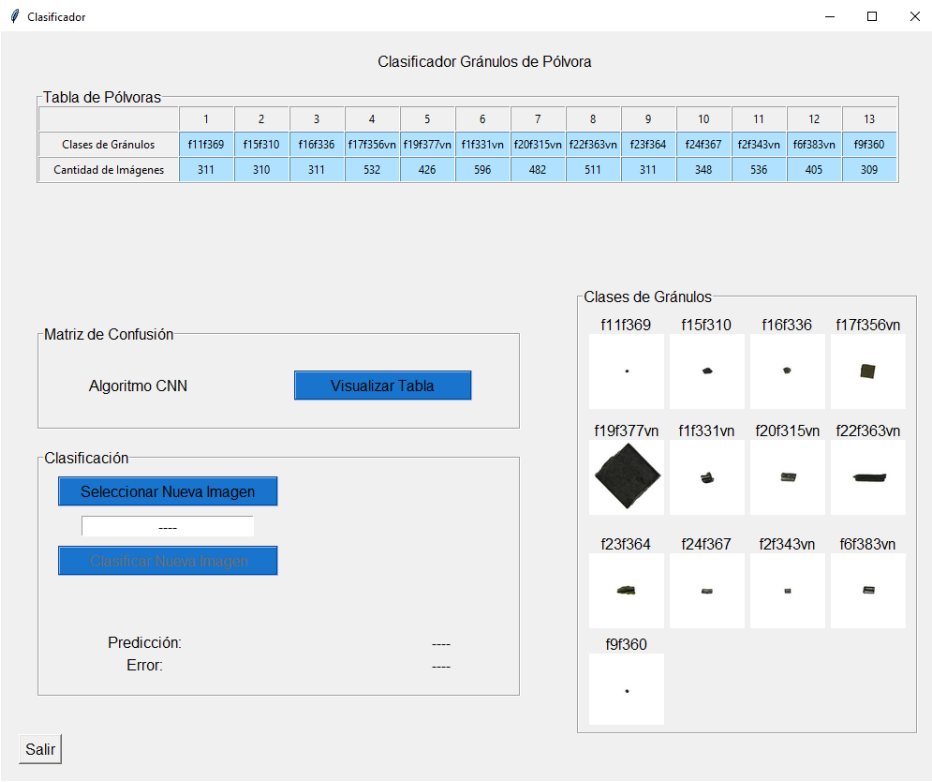


FIGURA 40. VENTANA DE LA APLICACIÓN

Finalmente, el diseño del sistema desarrollado para ambos lenguajes de programación habrá sido completado llegados a este punto. A continuación, se realizará una evaluación de los algoritmos explicados en capítulos anteriores, comparando los resultados que estos presenten en cada caso.

4. Evaluación

4.1 Resultados

En este capítulo se incluyen todos los resultados obtenidos en la evaluación de los distintos algoritmos desarrollados, pudiendo hacer una división de los ensayos realizados en distintos grupos.

4.1.1 Primer Conjunto de Pruebas

En un primer grupo se incluyen las pruebas realizadas con las imágenes de tamaño 227x227. Para ello, se ejecutará el algoritmo de la red original AlexNet y de la red MyNet para los siguientes casos:

1. Imágenes originales redimensionadas
2. Imágenes pre-procesadas sin orientación
3. Imágenes pre-procesadas con orientación

4.1.1.1. Red Original AlexNet

En cada caso se presenta la matriz de confusión y porcentaje medio de precisión para el mismo conjunto de datos de la siguiente forma:

Caso 1

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	87.0968	0	0	0	0	0	0	0	0	0	0	0	12.9032
2 f15f310	0	67.7419	12.9032	0	0	6.4516	0	0	0	6.4516	3.2258	3.2258	0
3 f16f336	0	9.6774	77.4194	0	3.2258	3.2258	3.2258	0	0	0	3.2258	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	97.6744	0	2.3256	0	0	0	0	0	0
6 f1f331vn	0	3.3333	1.6667	0	1.6667	80	5	0	0	0	0	8.3333	0
7 f20f315vn	0	0	0	0	2.0833	8.3333	81.2500	0	0	6.2500	0	2.0833	0
8 f22f363vn	0	0	0	0	0	0	1.9608	98.0392	0	0	0	0	0
9 f23f364	0	0	3.2258	0	0	0	0	0	93.5484	0	0	0	3.2258
10 f24f367	0	0	0	0	0	0	8.5714	0	0	88.5714	0	2.8571	0
11 f2f343vn	0	3.7037	9.2593	0	3.7037	0	7.4074	0	0	3.7037	64.8148	3.7037	3.7037
12 f6f383vn	0	2.4390	0	0	0	41.4634	4.8780	0	0	0	0	51.2195	0
13 f9f360	22.5806	0	0	0	0	0	0	0	0	0	0	0	77.4194

Precisión Media → 81, 9073%

Caso 2

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	90.3226	0	0	0	0	0	0	0	0	0	0	0	9.6774
2 f15f310	0	64.5161	12.9032	0	0	6.4516	3.2258	0	0	6.4516	3.2258	3.2258	0
3 f16f336	0	12.9032	70.9677	0	0	6.4516	0	0	0	0	9.6774	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	97.6744	0	0	0	0	0	0	0	2.3256
6 f1f331vn	0	3.3333	1.6667	0	1.6667	75	6.6667	0	0	0	0	11.6667	0
7 f20f315vn	0	0	0	0	0	27.0833	72.9167	0	0	0	0	0	0
8 f22f363vn	0	0	1.9608	0	0	0	0	98.0392	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	3.2258	96.7742	0	0	0	0
10 f24f367	0	0	2.8571	0	0	0	0	2.8571	0	94.2857	0	0	0
11 f2f343vn	0	1.8519	12.9630	0	1.8519	0	7.4074	0	0	1.8519	72.2222	1.8519	0
12 f6f383vn	0	0	0	0	0	34.1463	4.8780	0	0	2.4390	0	58.5366	0
13 f9f360	74.1935	0	0	0	0	0	0	0	0	0	0	0	25.8065

Precisión Media → 78,2355%

Caso 3

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	90.3226	0	0	0	0	0	0	0	0	0	0	0	9.6774
2 f15f310	0	77.4194	6.4516	0	0	9.6774	0	0	0	3.2258	3.2258	0	0
3 f16f336	0	9.6774	70.9677	0	3.2258	3.2258	0	0	0	3.2258	9.6774	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	97.6744	0	0	0	0	0	0	0	2.3256
6 f1f331vn	0	3.3333	0	0	1.6667	85	3.3333	0	0	0	0	6.6667	0
7 f20f315vn	0	0	0	0	0	6.2500	89.5833	4.1667	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	3.2258	96.7742	0	0	0	0
10 f24f367	0	5.7143	2.8571	0	0	0	2.8571	0	0	88.5714	0	0	0
11 f2f343vn	0	1.8519	1.8519	0	0	1.8519	3.7037	0	0	3.7037	87.0370	0	0
12 f6f383vn	0	0	2.4390	0	0	36.5854	2.4390	0	0	2.4390	0	56.0976	0
13 f9f360	70.9677	0	0	0	0	0	0	0	0	0	0	0	29.0323

Precisión Media → 82, 1908%

Conclusiones

En este punto, se debe tener en cuenta que, para utilizar la red AlexNet sobre nuestro conjunto de datos, la única posibilidad viable es que se realice una extracción de las características de las imágenes para entrenar el SVM. No se podrá utilizar la clasificación realizada por AlexNet directamente debido a que las etiquetas de predicción se corresponderán con objetos que fueron utilizados para entrenar esta red. Por lo tanto, no existirá similitud entre las etiquetas reales de los gránulos y las que haya predicho AlexNet.

Teniendo en cuenta esto, se puede apreciar que en algunos tipos de gránulos del conjunto original se tienen mejores resultados que en el caso de las imágenes procesadas. Esto se debe a que el recorte que presentan las imágenes originales, es decir, toda aquella información externa al gránulo, es incluido como si se tratase de características propias de estos, y por lo tanto, influyen en su clasificación. Así pues, estos resultados no se podrán tomar como datos fiables de cara a hacer una comparativa con los otros dos supuestos.

Por otro lado, se puede apreciar cierta mejora entre el caso en el que se utilizan las imágenes segmentadas sin orientar y las imágenes orientadas a una posición común horizontal. Esto se debe a que cada pixel de la imagen se corresponde con una entrada de la red, de manera que, si los gránulos de una misma clase se encuentran en posiciones distintas, afectará a la forma de realizar la clasificación de los mismos.

4.1.1.2. Red MyNet

Caso 1

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	83.8710	0	0	0	0	0	0	0	0	0	0	0	16.1290
2 f15f310	0	61.2903	22.5806	0	6.4516	3.2258	0	0	0	3.2258	3.2258	0	0
3 f16f336	0	9.6774	87.0968	0	0	0	0	0	0	0	3.2258	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	1.6667	1.6667	0	3.3333	88.3333	3.3333	0	0	0	0	1.6667	0
7 f20f315vn	2.0833	0	2.0833	0	6.2500	4.1667	81.2500	2.0833	0	2.0833	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	2.8571	0	2.8571	0	0	88.5714	0	5.7143	0
11 f2f343vn	0	5.5556	12.9630	0	7.4074	0	7.4074	0	0	0	66.6667	0	0
12 f6f383vn	0	0	0	0	9.7561	26.8293	0	0	0	0	0	63.4146	0
13 f9f360	19.3548	0	0	0	0	0	0	0	0	0	0	0	80.6452

Precisión Media → 84.7030%

Caso 2

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	93.5484	0	0	0	0	0	0	0	0	0	0	0	6.4516
2 f15f310	0	74.1935	16.1290	0	3.2258	3.2258	0	0	0	0	3.2258	0	0
3 f16f336	0	16.1290	74.1935	0	0	0	0	0	0	0	9.6774	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	3.3333	1.6667	0	1.6667	81.6667	0	1.6667	0	0	0	10	0
7 f20f315vn	0	0	0	0	2.0833	10.4167	87.5000	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	2.8571	2.8571	0	91.4286	0	2.8571	0
11 f2f343vn	0	5.5556	7.4074	0	7.4074	0	0	0	0	1.8519	74.0741	3.7037	0
12 f6f383vn	0	0	0	0	0	19.5122	0	0	0	0	0	80.4878	0
13 f9f360	70.9677	0	0	0	0	0	0	0	0	0	0	0	29.0323

Precisión Media → 83.5481%

Caso 3

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	96.7742	0	0	0	3.2258	0	0	0	0	0	0	0	0
2 f15f310	0	74.1935	12.9032	0	3.2258	6.4516	0	0	0	0	3.2258	0	0
3 f16f336	0	16.1290	70.9677	0	0	3.2258	0	0	0	0	9.6774	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	1.6667	3.3333	86.6667	0	0	0	0	0	8.3333	0
7 f20f315vn	0	0	0	0	4.1667	4.1667	91.6667	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	0	0	0	100	0	0	0
11 f2f343vn	0	3.7037	11.1111	0	5.5556	0	5.5556	0	0	1.8519	72.2222	0	0
12 f6f383vn	0	0	0	0	0	12.1951	0	0	0	2.4390	0	85.3659	0
13 f9f360	64.5161	0	0	0	0	0	0	0	0	0	0	0	35.4839

Precisión Media → 85.6416%

Conclusiones

En las tres pruebas, si estas son comparadas con los casos anteriores, se tienen unos porcentajes de precisión superiores en general. Esto es debido a que en el primer caso se está utilizando una red que fue diseñada y entrenada para clasificar objetos de 1000 clases distintas, dentro de las cuales no se encuentran los gránulos de pólvora que se desean evaluar. Sin embargo, en esta segunda propuesta, la red ha sido modificada y re-entrenada para clasificar los 13 tipos de objetos distintos con los que contamos en nuestra base de datos.

Por otro lado, y de la misma forma que ocurría para el caso de la red original, se tiene un mayor porcentaje de precisión en la prueba de las imágenes originales que en la prueba de las imágenes procesadas por el motivo expuesto anteriormente.

No obstante se ha querido contar con este primer caso para comprobar la mejora obtenida entre utilizar la red AlexNet original y la red re-entrenada.

4.1.2 Segundo Conjunto de Pruebas

El segundo grupo está compuesto por las pruebas realizadas con las imágenes de tamaño 227x227 sobre el algoritmo de Bag of Features. Para ello, este será ejecutado en distintas ocasiones variando el tamaño del vocabulario visual, y una vez se haya decidido cual resulta más adecuado para los datos tratados, se probarán distintos extractores de características.

Los casos utilizados son los siguientes:

1. Imágenes originales redimensionadas
2. Imágenes pre-procesadas sin orientación
3. Imágenes pre-procesadas con orientación

4.1.2.1. Bag Of Features- Vocabulario Visual →500

En primer lugar, se utiliza el valor por defecto del parámetro *VocabularySize* para cada uno de los casos indicados:

Caso 1

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	100	0	0	0	0	0	0	0	0	0	0	0	0
2 f15f310	0	16.1290	25.8065	9.6774	0	6.4516	0	3.2258	3.2258	3.2258	19.3548	9.6774	3.2258
3 f16f336	0	3.2258	22.5806	6.4516	0	16.1290	0	0	3.2258	0	35.4839	12.9032	0
4 f17f356vn	0	0	0	33.9623	0	0	0	66.0377	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	3.3333	8.3333	51.6667	0	15	0	20	1.6667	0	0	0	0
7 f20f315vn	0	0	0	29.1667	0	2.0833	0	68.7500	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	22.5806	9.6774	0	0	3.2258	0	35.4839	6.4516	3.2258	9.6774	6.4516	3.2258
10 f24f367	0	5.7143	0	0	0	8.5714	2.8571	0	5.7143	71.4286	5.7143	0	0
11 f2f343vn	1.8519	5.5556	7.4074	0	0	0	1.8519	0	0	7.4074	74.0741	0	1.8519
12 f6f383vn	0	14.6341	7.3171	2.4390	0	12.1951	2.4390	0	2.4390	39.0244	2.4390	17.0732	0
13 f9f360	96.7742	0	0	0	0	0	0	0	0	0	3.2258	0	0

Precisión Media → 42.8230%

Caso 2

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	100	0	0	0	0	0	0	0	0	0	0	0	0
2 f15f310	0	19.3548	16.1290	12.9032	0	9.6774	3.2258	3.2258	0	0	22.5806	12.9032	0
3 f16f336	0	6.4516	29.0323	9.6774	0	19.3548	0	0	0	0	19.3548	16.1290	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	3.3333	11.6667	55.0000	0	3.3333	5	8.3333	0	0	0	13.3333	0
7 f20f315vn	0	2.0833	0	70.8333	0	0	0	27.0833	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	9.6774	6.4516	3.2258	0	0	3.2258	41.9355	3.2258	6.4516	12.9032	12.9032	0
10 f24f367	0	2.8571	0	0	0	2.8571	0	2.8571	5.7143	57.1429	8.5714	20	0
11 f2f343vn	0	5.5556	3.7037	0	0	1.8519	1.8519	0	1.8519	11.1111	62.9630	11.1111	0
12 f6f383vn	0	0	17.0732	12.1951	0	17.0732	4.8780	0	2.4390	14.6341	0	31.7073	0
13 f9f360	100	0	0	0	0	0	0	0	0	0	0	0	0

Precisión Media → 46.6738%

Caso 3

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	100	0	0	0	0	0	0	0	0	0	0	0	0
2 f15f310	0	12.9032	12.9032	16.1290	0	9.6774	0	6.4516	3.2258	3.2258	35.4839	0	0
3 f16f336	0	0	35.4839	12.9032	0	3.2258	3.2258	6.4516	6.4516	0	25.8065	6.4516	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	8.3333	58.3333	0	20	1.6667	1.6667	10	0	0	0	0
7 f20f315vn	0	0	0	20.8333	0	0	0	79.1667	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	6.4516	3.2258	0	0	3.2258	45.1613	22.5806	3.2258	12.9032	0	3.2258
10 f24f367	0	2.8571	2.8571	0	0	0	0	34.2857	0	57.1429	2.8571	0	0
11 f2f343vn	0	5.5556	3.7037	0	0	5.5556	7.4074	0	0	3.7037	66.6667	1.8519	5.5556
12 f6f383vn	0	0	9.7561	17.0732	0	48.7805	2.4390	2.4390	7.3171	2.4390	0	9.7561	0
13 f9f360	100	0	0	0	0	0	0	0	0	0	0	0	0

Precisión Media → 48.0410%

Conclusiones

Como se puede apreciar, de igual manera que para las pruebas del primer bloque, los mejores resultados se han obtenido para el conjunto de datos orientados.

Por otro lado, se puede ver que en general estos son considerablemente más bajos que en el caso de utilizar la red CNN para la extracción de características.

4.1.2.2. Bag Of Features- Variaciones Vocabulario Visual

Teniendo en cuenta que los mejores resultados en una primera prueba han sido obtenidos para el conjunto de datos orientados, las pruebas realizadas posteriormente se han evaluado bajo dicho conjunto de datos.

Como se comentó en el punto **3.7.2 Extracción de Características con Bag of Features**, este parámetro debe ser ajustado realizando una validación experimental, de tal forma que el tamaño del vocabulario visual más adecuado para el estudio ha resultado en 4000, tal y como se puede apreciar en el Gráfico 4.

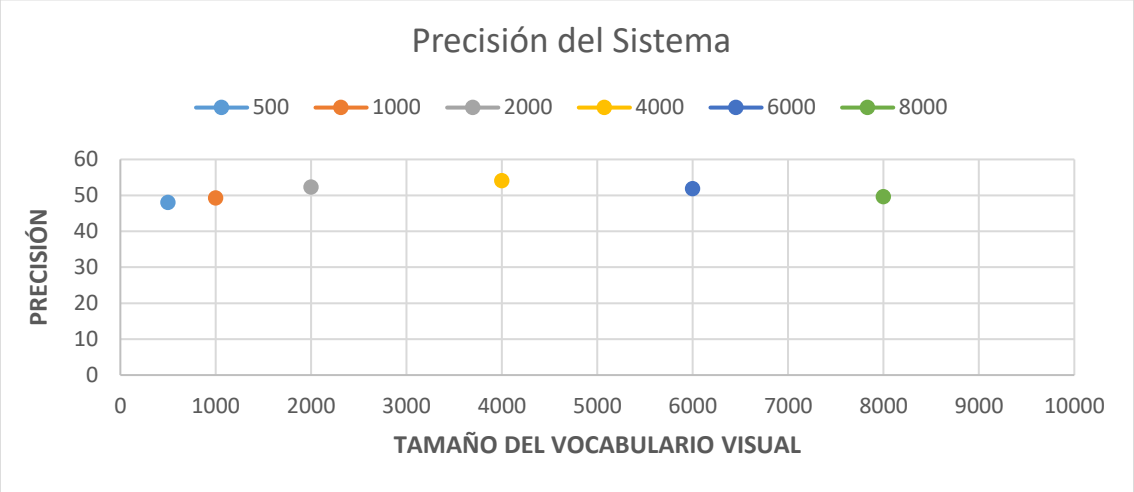


GRÁFICO 4. RESULTADOS PARA EL CONJUNTO DE DATOS ORIENTADOS VARIANDO EL TAMAÑO DEL VOCABULARIO VISUAL

Caso 3

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	100	0	0	0	0	0	0	0	0	0	0	0	0
2 f15f310	0	16.1290	16.1290	25.8065	0	0	3.2258	0	0	6.4516	29.0323	3.2258	0
3 f16f336	0	0	38.7097	9.6774	0	0	3.2258	6.4516	0	0	35.4839	6.4516	0
4 f17f356vn	0	0	0	98.1132	1.8868	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	6.6667	6.6667	48.3333	0	26.6667	3.3333	1.6667	6.6667	0	0	0	0
7 f20f315vn	0	0	0	22.9167	0	0	10.4167	66.6667	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	9.6774	6.4516	9.6774	0	0	0	32.2581	19.3548	6.4516	12.9032	0	3.2258
10 f24f367	0	2.8571	0	2.8571	0	0	0	11.4286	0	82.8571	0	0	0
11 f2f343vn	0	1.8519	1.8519	0	0	0	3.7037	0	0	1.8519	88.8889	0	1.8519
12 f6f383vn	0	7.3171	4.8780	12.1951	0	29.2683	2.4390	7.3171	2.4390	12.1951	0	21.9512	0
13 f9f360	96.7742	0	0	0	0	0	0	0	0	0	3.2258	0	0

Precisión Media → 54.0836%

4.1.2.3. Bag Of Features- Extractores

Para poder conformar la bolsa de características que se ha utilizado para entrenar el clasificador, es posible indicar al algoritmo que tipo de características o puntos de interés son más interesantes para el tipo de imágenes con las que se cuenta. En este caso se han utilizado los siguientes casos tratando con las imágenes orientadas:

1. Extracción por defecto con imágenes
2. Extracción basada en características de color

Caso 1

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	100	0	0	0	0	0	0	0	0	0	0	0	0
2 f15f310	0	16.1290	16.1290	25.8065	0	0	3.2258	0	0	6.4516	29.0323	3.2258	0
3 f16f336	0	0	38.7097	9.6774	0	0	3.2258	6.4516	0	0	35.4839	6.4516	0
4 f17f356vn	0	0	0	98.1132	1.8868	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	6.6667	6.6667	48.3333	0	26.6667	3.3333	1.6667	6.6667	0	0	0	0
7 f20f315vn	0	0	0	22.9167	0	0	10.4167	66.6667	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	9.6774	6.4516	9.6774	0	0	0	32.2581	19.3548	6.4516	12.9032	0	3.2258
10 f24f367	0	2.8571	0	2.8571	0	0	0	11.4286	0	82.8571	0	0	0
11 f2f343vn	0	1.8519	1.8519	0	0	0	3.7037	0	0	1.8519	88.8889	0	1.8519
12 f6f383vn	0	7.3171	4.8780	12.1951	0	29.2683	2.4390	7.3171	2.4390	12.1951	0	21.9512	0
13 f9f360	96.7742	0	0	0	0	0	0	0	0	0	3.2258	0	0

Precisión Media → 54.0836%

Caso 2

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	96.7742	0	0	0	0	0	0	0	0	0	0	0	3.2258
2 f15f310	0	48.3871	16.1290	0	0	3.2258	16.1290	0	0	9.6774	6.4516	0	0
3 f16f336	0	16.1290	48.3871	0	0	9.6774	9.6774	0	0	3.2258	12.9032	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	18.3333	0	53.3333	6.6667	5	11.6667	0	0	5	0
7 f20f315vn	0	0	0	16.6667	0	2.0833	58.3333	18.7500	4.1667	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	6.4516	0	0	0	38.7097	54.8387	0	0	0	0
10 f24f367	2.8571	2.8571	0	0	0	0	2.8571	0	5.7143	77.1429	0	8.5714	0
11 f2f343vn	0	5.5556	3.7037	0	0	0	1.8519	0	1.8519	1.8519	74.0741	7.4074	3.7037
12 f6f383vn	2.4390	2.4390	12.1951	0	0	14.6341	7.3171	2.4390	12.1951	7.3171	0	39.0244	0
13 f9f360	90.3226	0	0	0	0	0	0	3.2258	0	0	0	0	6.4516

Precisión Media → 65.9036%

Conclusiones

Como algunos tipos de gránulos presentan ciertos tonos de verdes y marrones, se ha tomado la opción de utilizar un extractor basado en características locales de color sobre los puntos de interés interpretados en la imagen, puesto que el objetivo es encontrar el tipo de objeto específico que esta contiene. Se puede apreciar que el uso de este método resulta ser considerablemente más preciso que el anterior, ya que el color es una característica con cierta influencia en los distintos tipos de gránulos que conforman el conjunto de datos.

No obstante, el mejor de los resultados obtenidos con este tipo de extracción queda lejos del peor de los obtenidos con la red neuronal convolucional, por lo que se tomará la opción de continuar sobre el algoritmo de extracción de la CNN.

4.1.3 Tercer Conjunto de Pruebas

Este último grupo de pruebas, parte del conjunto de imágenes de dimensiones 735x735, obtenidas en la etapa de pre-procesamiento, ***3.3 Etapa de Pre-procesamiento***. Se ha realizado un escalado de las imágenes en diferentes tamaños, comprobando cómo se comporta MyNet en cada caso.

Los casos utilizados sobre el primer grupo de imágenes son los siguientes:

1. Set de Imágenes 100x100
2. Set de Imágenes 150x150
3. Set de Imágenes 227x227
4. Set de Imágenes 350x350
5. Set de Imágenes 500x500
6. Set de Imágenes 735x735
7. Set de Imágenes 1000x1000
8. Set de Imágenes 1300x1300

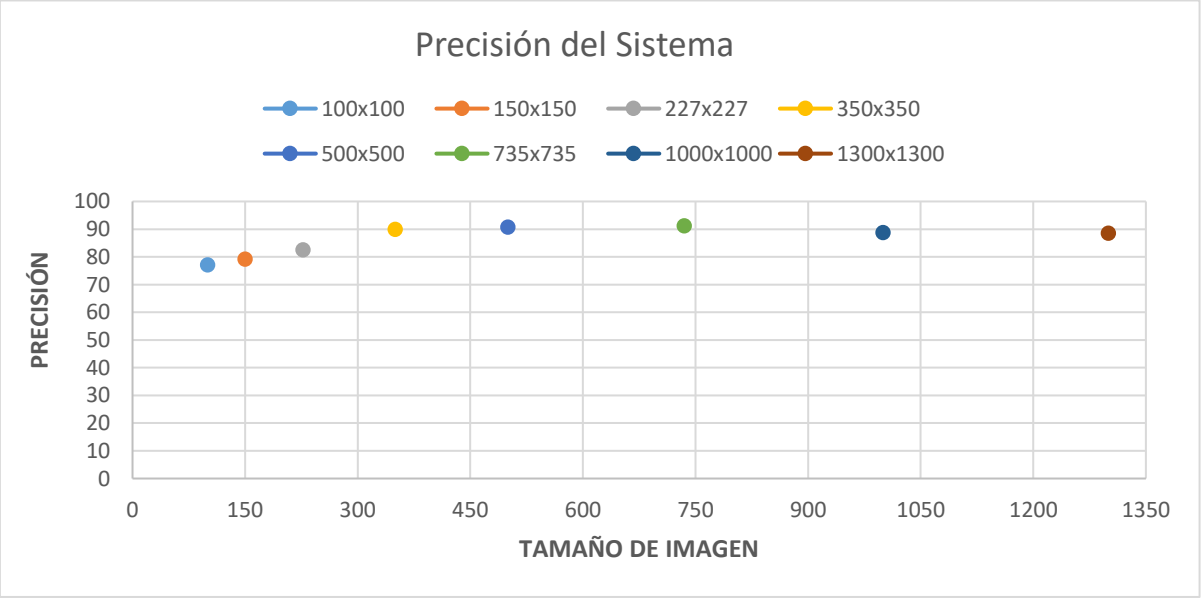


GRÁFICO 5. RESULTADOS PARA EL ENTRENAMIENTO DE LA RED BAJO DISTINTOS TAMAÑOS DE IMÁGENES

Set de Imágenes 100 x 100

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	100	0	0	0	0	0	0	0	0	0	0	0	0
2 f15f310	0	61.2903	9.6774	0	0	9.6774	6.4516	0	0	0	9.6774	3.2258	0
3 f16f336	0	12.9032	61.2903	0	0	9.6774	3.2258	0	0	0	12.9032	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	0	0	86.6667	8.3333	0	0	0	0	5	0
7 f20f315vn	0	0	0	0	0	2.0833	95.8333	2.0833	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	6.4516	93.5484	0	0	0	0
10 f24f367	0	8.5714	0	0	0	0	0	0	0	91.4286	0	0	0
11 f2f343vn	0	1.8519	16.6667	0	0	0	0	0	1.8519	7.4074	68.5185	0	3.7037
12 f6f383vn	0	2.4390	2.4390	0	0	48.7805	2.4390	0	0	2.4390	0	41.4634	0
13 f9f360	96.7742	0	0	0	0	0	0	0	0	0	0	0	3.2258

Precisión Media → 77,1742%

Set de Imágenes 150 x 150

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	100	0	0	0	0	0	0	0	0	0	0	0	0
2 f15f310	0	35.4839	19.3548	0	0	9.6774	0	0	3.2258	6.4516	19.3548	6.4516	0
3 f16f336	0	16.1290	58.0645	0	0	6.4516	6.4516	0	0	0	12.9032	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	1.6667	0	0	90	1.6667	0	0	0	0	6.6667	0
7 f20f315vn	0	0	0	2.0833	0	2.0833	95.8333	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	3.2258	93.5484	0	0	0	3.2258
10 f24f367	0	0	0	0	0	0	0	0	0	100	0	0	0
11 f2f343vn	0	1.8519	7.4074	0	0	0	0	0	0	5.5556	85.1852	0	0
12 f6f383vn	0	0	2.4390	0	0	21.9512	4.8780	0	0	2.4390	0	68.2927	0
13 f9f360	93.5484	0	0	0	0	0	0	0	0	0	3.2258	0	3.2258

Precisión Media → 79.2026%

Set de Imágenes 227 x 227

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	96.7742	0	0	0	0	0	0	0	0	0	0	0	3.2258
2	0	51.6129	38.7097	0	0	3.2258	3.2258	0	0	0	3.2258	0	0
3	0	9.6774	80.6452	0	0	0	0	0	0	0	9.6774	0	0
4	0	0	0	100	0	0	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0	0
6	0	0	3.3333	0	0	91.6667	1.6667	0	0	0	0	3.3333	0
7	0	0	0	2.0833	0	2.0833	95.8333	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0
9	0	0	0	0	0	0	0	0	100	0	0	0	0
10	0	0	0	0	0	0	0	0	0	100	0	0	0
11	0	1.8519	11.1111	0	0	0	1.8519	0	0	1.8519	83.3333	0	0
12	0	0	0	0	0	21.9512	2.4390	0	0	2.4390	0	73.1707	0
13	100	0	0	0	0	0	0	0	0	0	0	0	0

Precisión Media → 82.5413%

Set de Imágenes 350 x 350

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	83.8710	0	0	0	0	0	0	0	0	0	0	0	16.1290
2	0	80.6452	16.1290	0	0	0	0	0	3.2258	0	0	0	0
3	0	6.4516	87.0968	0	0	0	0	0	0	0	6.4516	0	0
4	0	0	0	100	0	0	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0	0
6	0	0	0	0	0	96.6667	0	0	0	0	0	3.3333	0
7	0	0	0	2.0833	0	2.0833	95.8333	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0
9	0	0	0	0	0	0	0	0	100	0	0	0	0
10	0	2.8571	0	0	0	0	0	0	0	97.1429	0	0	0
11	0	5.5556	5.5556	0	0	0	0	0	0	0	88.8889	0	0
12	0	0	0	0	0	9.7561	2.4390	0	0	0	0	87.8049	0
13	48.3871	0	0	0	0	0	0	0	0	0	0	0	51.6129

Precisión Media → 89.9663%

Set de Imágenes 500 x 500

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	87.0968	0	0	0	0	0	0	0	0	0	0	0	12.9032
2	0	83.8710	16.1290	0	0	0	0	0	0	0	0	0	0
3	0	6.4516	87.0968	0	0	0	0	0	0	0	6.4516	0	0
4	0	0	0	100	0	0	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0	0
6	0	0	0	0	0	98.3333	0	0	0	0	0	1.6667	0
7	0	0	0	2.0833	0	6.2500	91.6667	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0
9	0	0	0	0	0	0	0	0	100	0	0	0	0
10	0	0	0	0	0	0	0	0	0	100	0	0	0
11	0	1.8519	1.8519	0	0	0	1.8519	0	0	1.8519	92.5926	0	0
12	0	0	0	0	0	17.0732	2.4390	0	0	0	0	80.4878	0
13	41.9355	0	0	0	0	0	0	0	0	0	0	0	58.0645

Precisión Media → 90.7084%

Set de Imágenes 735 x 735

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	93.5484	0	0	0	0	0	0	0	0	0	0	0	6.4516
2 f15f310	0	87.0968	9.6774	0	0	3.2258	0	0	0	0	0	0	0
3 f16f336	0	6.4516	77.4194	0	0	3.2258	0	0	0	0	12.9032	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	0	0	90	0	0	0	0	0	10	0
7 f20f315vn	0	0	0	2.0833	0	6.2500	91.6667	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	0	0	0	100	0	0	0
11 f2f343vn	0	0	5.5556	0	0	0	0	0	0	1.8519	90.7407	0	1.8519
12 f6f383vn	0	0	0	0	0	9.7561	2.4390	0	0	0	0	87.8049	0
13 f9f360	32.2581	0	0	0	0	0	0	0	0	0	0	0	67.7419

Precisión Media → 91.2322%

Set de Imágenes 1000 x 1000

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	90.3226	0	0	0	0	0	0	0	0	0	0	0	9.6774
2	0	67.7419	32.2581	0	0	0	0	0	0	0	0	0	0
3	0	6.4516	80.6452	0	0	0	3.2258	0	0	0	9.6774	0	0
4	0	0	0	100	0	0	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0	0
6	0	0	0	0	0	98.3333	1.6667	0	0	0	0	0	0
7	0	0	0	2.0833	0	2.0833	95.8333	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0
9	0	0	0	0	0	0	0	0	100	0	0	0	0
10	0	0	0	0	0	0	0	0	0	100	0	0	0
11	0	3.7037	5.5556	0	0	0	0	0	0	0	90.7407	0	0
12	0	0	0	0	0	51.2195	2.4390	0	0	0	0	46.3415	0
13	16.1290	0	0	0	0	0	0	0	0	0	0	0	83.8710

Precisión Media → 88.7561%

Set de Imágenes 1300 x 1300

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	90.3226	0	0	0	0	0	0	0	0	0	0	0	9.6774
2	0	67.7419	32.2581	0	0	0	0	0	0	0	0	0	0
3	0	6.4516	83.8710	0	0	0	3.2258	0	0	0	6.4516	0	0
4	0	0	0	100	0	0	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0	0
6	0	1.6667	0	0	0	88.3333	0	0	0	0	0	10	0
7	0	0	0	0	0	2.0833	97.9167	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0
9	0	0	0	0	0	0	0	0	100	0	0	0	0
10	0	0	0	0	0	0	0	0	0	100	0	0	0
11	0	0	1.8519	0	0	1.8519	11.1111	0	0	0	85.1852	0	0
12	0	0	0	0	0	26.8293	7.3171	0	0	2.4390	0	63.4146	0
13	25.8065	0	0	0	0	0	0	0	0	0	0	0	74.1935

Precisión Media → 88.5368%

Conclusiones

Las imágenes pre-procesadas fueron redimensionadas, rellenándolas hasta la dimensión máxima de 735 que se encontró en el set de datos. Sin embargo, para el resto de casos se ha procedido escalando dichas imágenes a los valores que se presentan en el Gráfico 5, utilizando un método de interpolación que cuenta los valores de los píxeles de una cierta vecindad para obtener un promedio del valor interpolado.

Como se puede apreciar, el Gráfico 5 va a tener una forma de pendiente ascendente hasta un punto en el que esta tiende a estabilizarse. Por lo tanto, se puede concluir que en las imágenes escaladas a tamaños inferiores de 735, se está perdiendo información de los gránulos, y cuanto más se disminuye su tamaño, peores resultados se obtienen.

Con las imágenes escaladas a tamaños mayores de 735, se va a tener una mayor cantidad de píxeles, lo que puede interpretarse como mayor cantidad de información. Sin embargo, no es posible obtener más información válida y la precisión del sistema tiende a mantenerse estable en estos casos.

4.2 Mejor Resultado

El mejor resultado se ha obtenido para la clasificación sobre MyNet con el conjunto de muestras de dimensiones 735x735. Para obtener estas imágenes ha sido necesario:

1. Redimensionar todo el conjunto, rellenando cada imagen de píxeles de fondo hasta alcanzar dichas dimensiones.
2. Segmentar cada gránulo por medio de técnicas de binarización y operaciones morfológicas, dejándolo libre de información irrelevante para su clasificación.
3. Orientar todos los objetos a una posición común, evitando que las entradas de la CNN procesen la información erróneamente.

Las matrices de confusión obtenidas para cada uno de los grupos de imágenes se van a corresponder con las siguientes:

<u>Grupo de Imágenes 1</u>

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	93.5484	0	0	0	0	0	0	0	0	0	0	0	6.4516
2 f15f310	0	87.0968	9.6774	0	0	3.2258	0	0	0	0	0	0	0
3 f16f336	0	6.4516	77.4194	0	0	3.2258	0	0	0	0	12.9032	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	0	0	90	0	0	0	0	0	10	0
7 f20f315vn	0	0	0	2.0833	0	6.2500	91.6667	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	0	0	0	100	0	0	0
11 f2f343vn	0	0	5.5556	0	0	0	0	0	0	1.8519	90.7407	0	1.8519
12 f6f383vn	0	0	0	0	0	9.7561	2.4390	0	0	0	0	87.8049	0
13 f9f360	32.2581	0	0	0	0	0	0	0	0	0	0	0	67.7419

Precisión Media → 91.2322%

Grupo de Imágenes 2

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	90.3226	0	0	0	0	0	0	0	0	0	0	0	9.6774
2 f15f310	0	93.5484	6.4516	0	0	0	0	0	0	0	0	0	0
3 f16f336	0	9.6774	87.0968	0	0	0	0	0	0	0	3.2258	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	0	0	95	0	0	0	0	0	5	0
7 f20f315vn	0	0	0	0	0	0	100	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	0	0	0	100	0	0	0
11 f2f343vn	0	0	0	0	0	0	0	0	0	0	100	0	0
12 f6f383vn	0	0	0	0	0	26.8293	0	0	0	2.4390	0	70.7317	0
13 f9f360	35.4839	0	0	0	0	0	0	0	0	0	0	0	64.5161

Precisión Media → 92.4012%

Grupo de Imágenes 3

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	80.6452	0	0	0	0	0	0	0	0	0	0	0	19.3548
2 f15f310	0	87.0968	9.6774	0	0	0	0	0	0	3.2258	0	0	0
3 f16f336	0	3.2258	93.5484	0	0	0	0	0	0	0	3.2258	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	0	0	96.6667	0	0	0	0	0	3.3333	0
7 f20f315vn	0	0	0	0	0	2.0833	97.9167	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	2.8571	0	0	82.8571	2.8571	11.4286	0
11 f2f343vn	0	1.8519	0	0	0	0	1.8519	0	0	0	96.2963	0	0
12 f6f383vn	0	0	0	0	0	4.8780	0	0	0	0	0	95.1220	0
13 f9f360	0	0	0	0	0	0	0	0	0	3.2258	0	0	96.7742

Precisión Media → 94.3787%

Grupo de Imágenes 4

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	90.3226	0	0	0	0	0	0	0	0	0	0	0	9.6774
2 f15f310	0	96.7742	0	0	0	0	0	0	0	0	3.2258	0	0
3 f16f336	0	9.6774	83.8710	0	0	0	3.2258	0	0	0	3.2258	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	1.6667	0	0	0	88.3333	0	0	0	0	0	10	0
7 f20f315vn	0	0	0	0	0	0	100	0	0	0	0	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	0	0	0	100	0	0	0
11 f2f343vn	0	0	0	0	0	0	0	0	0	0	100	0	0
12 f6f383vn	0	0	0	0	0	0	0	0	0	43.9024	4.8780	51.2195	0
13 f9f360	0	0	0	0	0	0	0	0	0	0	0	0	100

Precisión Media → 93.1170%

Grupo de Imágenes 5

	1 f11f369	2 f15f310	3 f16f336	4 f17f356vn	5 f19f377vn	6 f1f331vn	7 f20f315vn	8 f22f363vn	9 f23f364	10 f24f367	11 f2f343vn	12 f6f383vn	13 f9f360
1 f11f369	90.6250	0	0	0	0	0	0	0	0	0	0	0	9.3750
2 f15f310	0	96.7742	3.2258	0	0	0	0	0	0	0	0	0	0
3 f16f336	0	3.1250	93.7500	0	0	0	3.1250	0	0	0	0	0	0
4 f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0	0
5 f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0	0
6 f1f331vn	0	0	0	0	0	33.3333	3.3333	0	0	0	5	58.3333	0
7 f20f315vn	0	0	0	0	0	2.0408	95.9184	0	0	0	2.0408	0	0
8 f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0	0
9 f23f364	0	0	0	0	0	0	0	0	100	0	0	0	0
10 f24f367	0	0	0	0	0	0	0	0	0	94.2857	0	5.7143	0
11 f2f343vn	0	0	0	0	0	0	0	0	0	0	100	0	0
12 f6f383vn	0	0	0	0	0	2.4390	0	0	0	0	0	97.5610	0
13 f9f360	3.2258	0	0	0	0	0	0	0	0	0	3.2258	0	93.5484

Precisión Media → 91.9843%

Precisión Media Total→ 92.6227%

4.3 Conclusiones

En general se cuenta con una precisión elevada en los distintos tipos de pólvoras, pudiendo ver que de forma sistemática las que más problemas dan, son aquellas que cuentan con tamaño, forma y color afines.

En el caso de las clases f9f360 y f11f369, además de la similitud de características, hay que tener en cuenta que se está tratando con objetos de dimensiones muy pequeñas, lo cual disminuye considerablemente los puntos de interés que pueden ser extraídos de los mismos.

Por otro lado, se puede apreciar que hay ciertas variaciones entre los 5 sets de imágenes que componen el conjunto total, teniendo precisiones variables para un mismo tipo de gránulos si los comparamos entre sí. Esto es así, debido a que dentro de una misma clase existen gránulos anómalos, es decir, se está tratando con objetos que no se forman siguiendo un patrón específico. Así pues, es imprescindible contar con estos casos para el desarrollo del sistema de clasificación, puesto que va a ser inevitable encontrar este tipo de irregularidades.

Por último, una posible línea de trabajo que podría resultar favorable en el sistema, sería realizar nuevamente el proceso de adquisición de los datos utilizados. De manera que, teniendo en cuenta una serie de factores durante este proceso, se consigan imágenes de mayor calidad. En el punto ***IX. Requerimientos en la Adquisición,*** se contemplan algunas posibilidades.

VII. Presupuesto

Los costes que han supuesto el trabajo desarrollado en este proyecto se puede desglosar en:

Costes Materiales

Producto	Cantidad	Coste
Ordenador	**1**	**1000€**
Sistema Operativo Windows 10	**1**	**260€**
Entorno de Desarrollo Matlab 2017b	**1**	**Licencia de un año 250€**
Microsoft Office 2013	**1**	**150€**
Sistema de Adquisición	**1**	**60€**

Coste Total	**1870 €**

Costes Personales

Actividad	Tiempo
Desarrollo con Matlab	**700 Horas**
Desarrollo con Python	**250 Horas**
Mecanografía	**100 Horas**

Englobando el trabajo en los grupos de actividades mencionados, se tiene una gráfica con el tiempo empleado para cada tarea como la mostrada seguidamente, Gráfico 6.

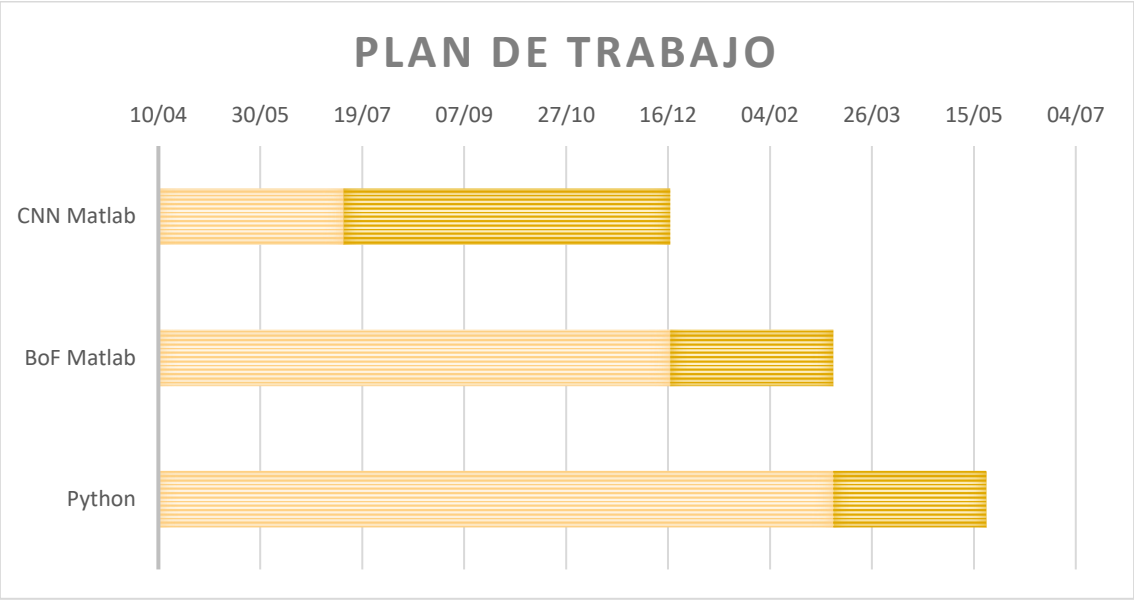


GRÁFICO 6. TEMPORIZACIÓN DE LOS TRABAJOS REALIZADOS

VIII. Pliego de Condiciones

El trabajo de este proyecto se ha realizado con una serie de elementos necesarios para su desempeño, siendo los mismos:

Requerimientos de Tipo Hardware:

- Equipo Ordenador con al menos:

  - 1 GByte de Espacio Libre de Memoria
  - 8 GByte de Memoria RAM
  - Tarjeta Gráfica con Capacidad de Computo mínima de 3.0
    Por ejemplo: NVIDIA GeForce GTX 650

Requerimientos de Tipo Software:

- Equipo con sistema operativo Windows 10
- Entorno de desarrollo Matlab versión 9.3.0.713579 (2017b) con, al menos:

  - Image Processing Toolbox versión 10.1
  - Neural Network Toolbox versión 11.0
  - Signal Processing Toolbox versión 7.5

- Lenguaje de programación Python versión 3.5 de 64 bit con, al menos:

  - Tensorflow versión 1.4.0
  - Scipy versión 1.0.0
  - Matplotlib versión 2.1.1
  - Pillow versión 4.0.0 – PIL versión 1.1.7
  - Numpy versión 1.14.0
  - Scikit-Learn – Sklearn versión 0.19.1
  - Tkinter – Tcl/Tk versión 8.6

IX. Requerimientos en la Adquisición

Como se mencionó en puntos anteriores, el proceso mediante el cual se realiza la adquisición de las imágenes será la primera pieza a tratar del sistema que se desea desarrollar. Esta resulta de vital importancia debido a que existirá una gran diferencia entre hacer una buena o una mala captación en tratamientos posteriores, reduciendo o incrementando la complejidad de los algoritmos de procesamiento.

Principalmente se tiene que tener en cuenta los elementos expuestos a continuación:

1. **Sistema de Iluminación.** Este estará compuesto por:

    I. Una fuente, que podrá ser de tipo luz LED, fluorescente, láser, luz natural, etc.

    II. Una técnica de iluminación, que en función del material del que esté compuesto el objeto o de la información que se quiera adquirir con mayor detalle, será conveniente usar una u otra, destacando, la iluminación directa, lateral, por contrate, por campo oscuro, etc.

2. **Óptica.** Donde se incluirán todos los parámetros referentes a la cámara destacando:

    I. El tipo de lente, pudiendo encontrar lentes convergentes, donde la distancia focal, que representará la distancia existente entre el centro óptico de la lente y el foco, será positiva, o lentes divergentes, donde esta será negativa.

II. El ángulo de visión, determinado por la longitud focal de la lente y el tamaño del sensor de imagen, se tratará del factor que determina la parte de la escena captada por dicho sensor.

III. La Profundidad de campo, se corresponde con la zona en la cual la imagen captada por el objetivo es nítida, dependerá de la distancia focal y de la abertura de la lente, expresada como una relación entre longitud focal y diámetro, de tal forma que a mayor abertura menor nitidez y a menor abertura mayor nitidez.

3. Sensor.

Actualmente en el mercado se puede encontrar una amplia gama de cámaras con altas prestaciones, el sensor de las cámaras está compuesto por millones de pequeños semiconductores de silicio que captan los fotones que componen la luz, siendo las dos tecnologías más populares utilizadas CCD y CMOS.

En función de la aplicación sobre la que se requiera hacer uso de este dispositivo, se dará una mayor importancia a unas prestaciones o a otras, como por ejemplo, la sensibilidad presentada ante la luz, la resolución o cantidad de píxeles que contendrá una imagen producida por el mismo, etc.

En caso de que no se disponga de los medios necesarios para obtener un sistema de captación como los que se pueden utilizar en sistemas de visión para aplicaciones industriales, se puede recurrir a métodos alternativos como los que se muestran seguidamente.

IX.I Método 1

La primera opción propuesta, está compuesta por un sistema formado por los siguientes útiles:

1. Una serie de paneles de polímero de distintos colores que sirven como fondo de las imágenes tomadas.

2. Tres estructuras de dimensiones cuadradas, distintos tamaños y distintas alturas, en función del tamaño de los objetos que se desee fotografiar y del tipo de plano en el que se requiera la imagen. Estas estructuras están compuestas de un material translucido para procurar una luz difusa y evitar sombras.

FIGURA 41. ESTRUCTURAS DEL SISTEMA

El Kit de adquisición está pensado para utilizarse con distintos dispositivos electrónicos incluyendo: smartphones, tablets o incluso cámaras de carácter más profesional.

Puede encontrarse en la web de accesorios fotográficos Modahaus [*Ref.11*] por un precio total de 54,75 $.

IX.II Método 2

Este segundo sistema portátil está formado por un cubo con las siguientes descripciones:

1. Medidas de 60x60x60 cm.

2. Cuenta con dos tableros de diodos leds de una potencia aproximada de 30W y temperatura de color de 5500K, correspondiendo a un color de luz neutro o natural.

3. Incluye tres fondos de plástico distintos, uno de color negro, uno blanco y uno naranja.

4. Su interior está fabricado de un material reflectante que evita posibles sombras.

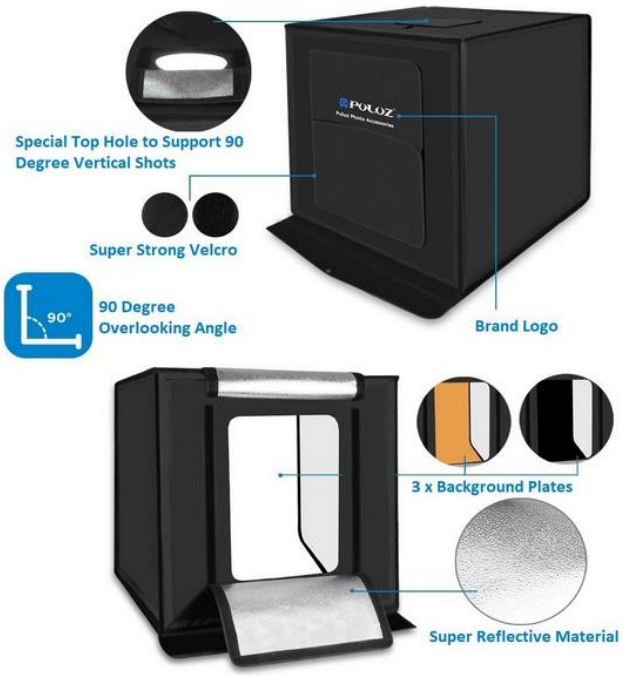


FIGURA 42. ESTRUCTURA DEL SISTEMA

El kit facilita la opción de tomar las imágenes desde distintos ángulos. Se puede encontrar en el departamento de estudios fotográficos e iluminación de la web Amazon [*Ref.12*] por un precio de 85 € aproximadamente.

IX.III Método 3

Se ofrece una tercera opción en la que se pueda escoger cada elemento del sistema por separado para que este se adecue lo mejor posibles a los datos concretos del estudio tratado.

Este sistema es considerablemente más costoso que los mencionados anteriormente, pero también supone una mayor profesionalidad en la toma de las fotografías.

FIGURA 43. COMPONENTES PARA SISTEMAS DE ADQUISICIÓN DE IMÁGENES

Todos los elementos necesarios se pueden encontrar en la sección de visión artificial de la web GrupoAlava [*Ref.13*], que posee un amplio catálogo de lentes, cámaras, sistemas de iluminación, etc.

X. Manual de Usuario

Las ventanas de aplicación diseñadas a través del entorno de programación de Matlab y de Python son visualmente equivalentes, Figura 44. Ofrecen las siguientes opciones:

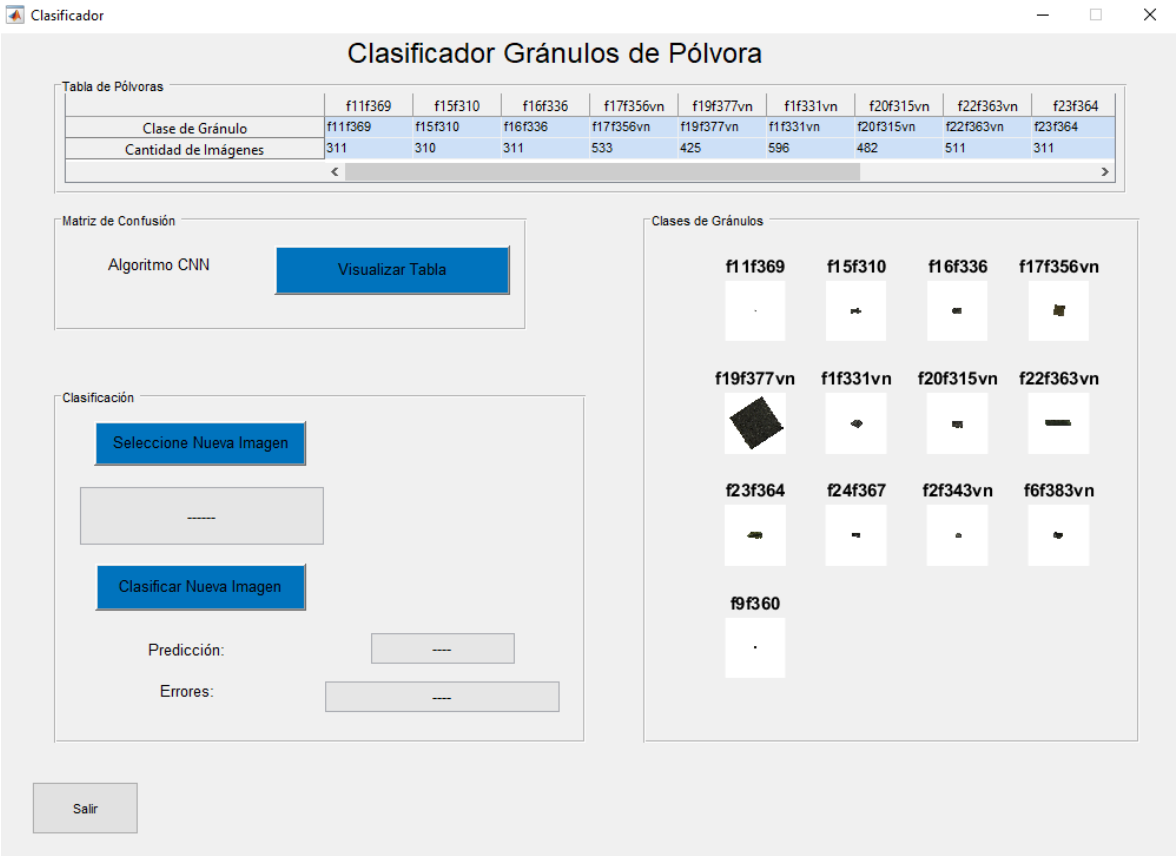
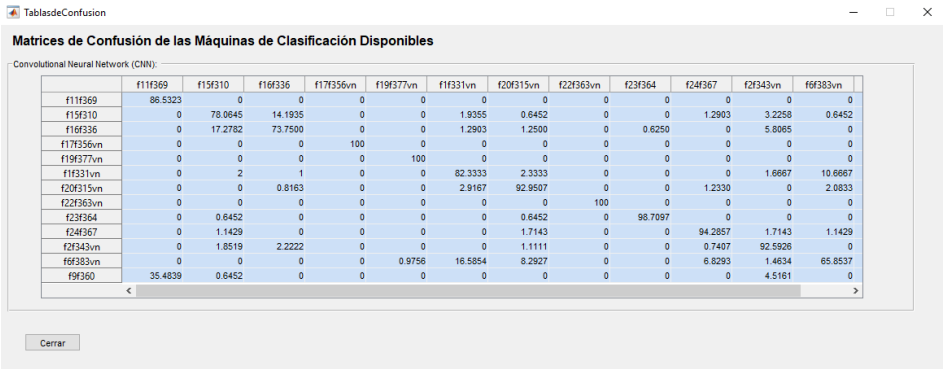


FIGURA 44. INTERFAZ GRÁFICA

1. Visualización de la tabla que contiene cada tipo de gránulo de pólvora y la cantidad de imágenes disponibles.

2. Visualización de una imagen de cada clase de gránulo junto con la etiqueta del grupo al que pertenece.

3. Visualización de la matriz de confusión, a la cual se accede presionando el botón **Visualizar tabla** dispuesto para esta acción, y con el que se abre una ventana que contiene dicha tabla, Figura 45. Para cerrar la tabla se ha incluido el botón **Cerrar.**



	f11f369	f15f310	f16f336	f17f356vn	f19f377vn	f1f331vn	f20f315vn	f22f363vn	f23f364	f24f367	f2f343vn	f6f383vn
f11f369	86.5323	0	0	0	0	0	0	0	0	0	0	0
f15f310	0	78.0645	14.1935	0	0	1.9355	0.6452	0	0	1.2903	3.2258	0.6452
f16f336	0	17.2782	73.7500	0	0	1.2903	1.2500	0	0.6250	0	5.8065	0
f17f356vn	0	0	0	100	0	0	0	0	0	0	0	0
f19f377vn	0	0	0	0	100	0	0	0	0	0	0	0
f1f331vn	0	2	1	0	0	82.3333	2.3333	0	0	0	1.6667	10.6667
f20f315vn	0	0	0.8163	0	0	2.9167	92.9507	0	0	1.2330	0	2.0833
f22f363vn	0	0	0	0	0	0	0	100	0	0	0	0
f23f364	0	0.6452	0	0	0	0	0.6452	0	98.7097	0	0	0
f24f367	0	1.1429	0	0	0	0	1.7143	0	0	94.2857	1.7143	1.1429
f2f343vn	0	1.8519	2.2222	0	0	0	1.1111	0	0	0.7407	92.5926	0
f6f383vn	0	0	0	0	0.9756	16.5854	8.2927	0	0	6.8293	1.4634	65.8537
f9f360	35.4839	0.6452	0	0	0	0	0	0	0	0	4.5161	0

FIGURA 45. VENTANA CON LA MATRIZ DE CONFUSIÓN

4. Selección de una nueva imagen. Se utiliza el botón **Seleccione Nueva Imagen,** que abre una ventana de diálogo para explorar las carpetas donde se hayan almacenado las imágenes. En caso de no seleccionar ninguna imagen se imprime el mensaje de error mostrado en la Figura 46.

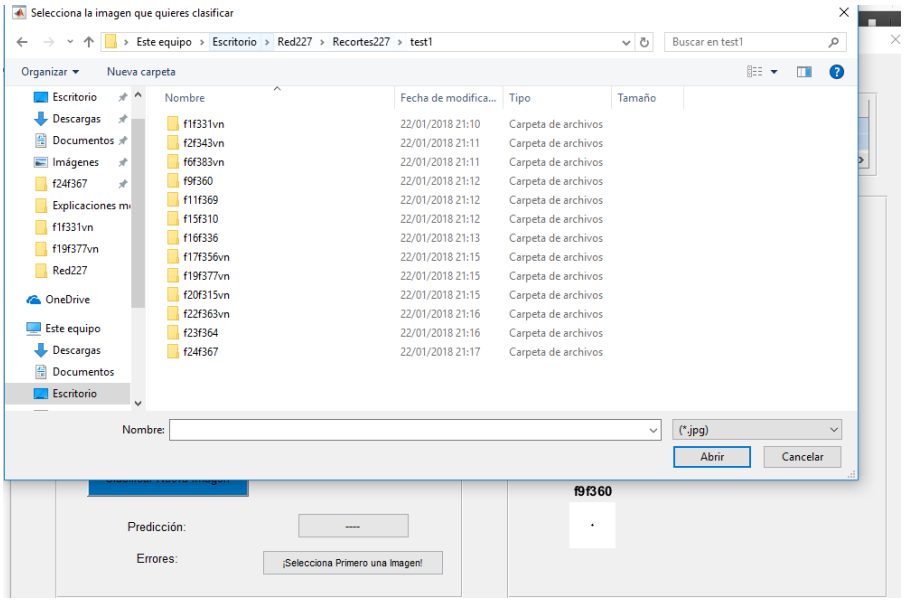


FIGURA 46. CUADRO DE DIÁLOGO Y MENSAJE DE ERROR

Una vez seleccionada la imagen, esta se visualiza junto con la ruta completa de la misma.

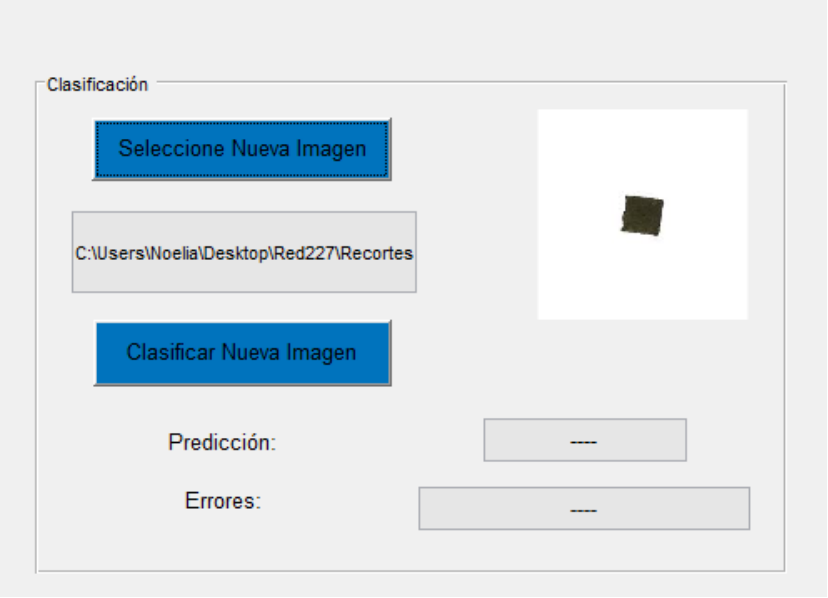


FIGURA 47. VISUALIZADO DE LA INFORMACIÓN DE LA IMAGEN SELECCIONADA

5. Clasificación de la imagen. Se acciona el botón de **Clasificar Nueva Imagen,** con el que se obtiene la etiqueta que el clasificador ha predicho para la imagen seleccionada anteriormente.

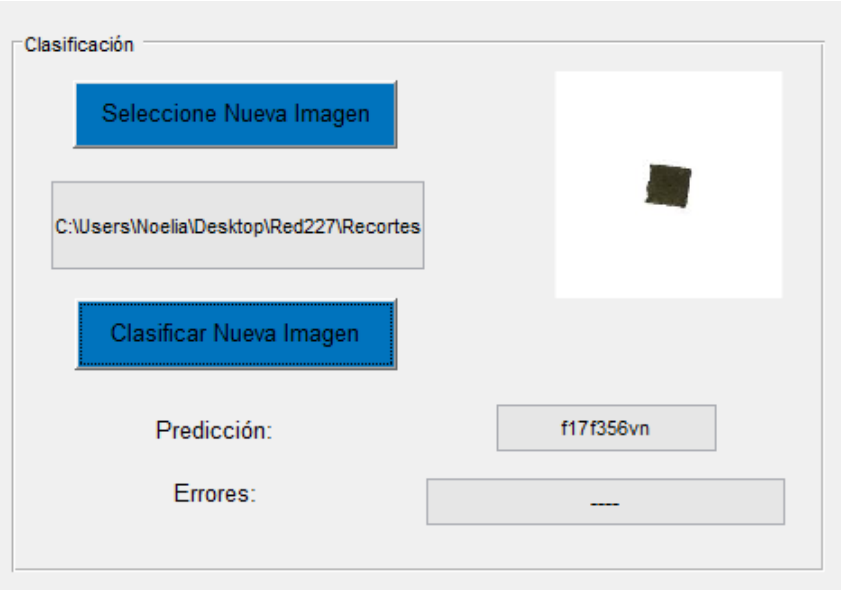


FIGURA 48. VISUALIZADO DE LA ETIQUETA DE PREDICCIÓN

6. Cierre de la ventana de aplicación presionando el botón **Salir.**

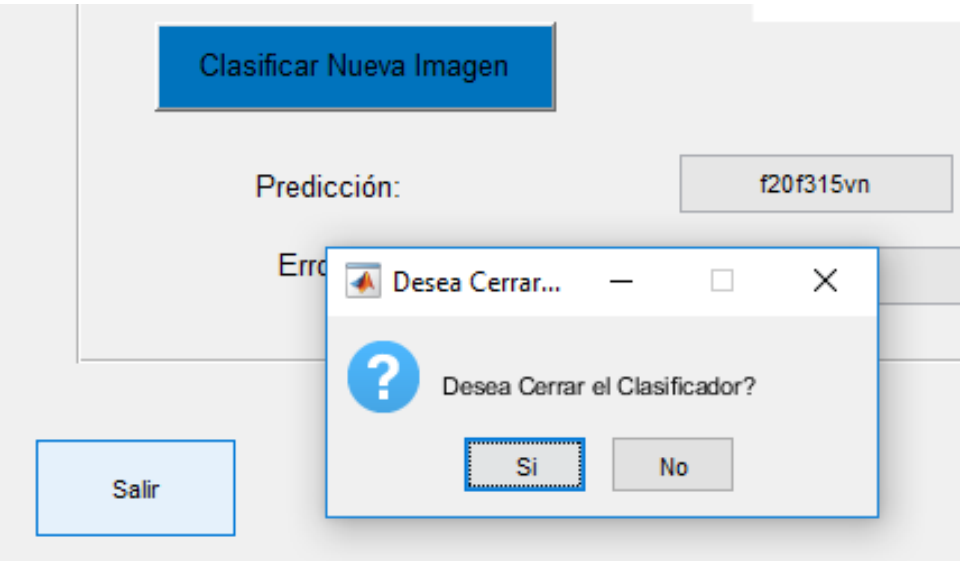


FIGURA 49. ACCIÓN DE CIERRE DE LA VENTANA

XI. Bibliografía

[*Ref.1*] Lowe, David G. "Object recognition from local scale-invariant features". Proceedings of the International Conference on Computer Vision 2. 1999.

[*Ref.2*] John Shawe-Taylor, Nello Cristianini. Kernel Methods for Pattern Analysis. Cambridge University Press. 1 Edition. June 28, 2004.

[*Ref.3*] Image Processing Toolbox

https://es.mathworks.com/products/image.html

[*Ref.4*] Rec.ITU-R BT.601-7

[*Ref.5*] Soille, P., Morphological Image Analysis: Principles and Applications, Springer-Verlag, 1999.

[*Ref.6*] Neural Network Toolbox

https://es.mathworks.com/products/neural-network.html

[*Ref.7*] Neural Network Toolbox → Classify Function

https://es.mathworks.com/help/nnet/ref/classify.html?s_tid=doc_ta

[*Ref.8*] Manuel R. Arahal, Manuel Berenguel Soria, Francisco Rodriguez Díaz. Técnicas de Predicción con Aplicaciones en Ingeniería. Universidad de Sevilla, 2006.

[*Ref.9*] Statistics and Machine Learning.

https://es.mathworks.com/products/statistics.html

[*Ref.10*] Equivalencias entre funciones de Matlab y Numpy/Scipy/Python

http://mathesaurus.sourceforge.net/matlab-numpy.html

[*Ref.11*] Sistema de Adquisición-Método 1.

https://www.modahaus.com/shop/modahaus-steady-stand-300-kit

[*Ref.12*] Sistema de Adquisición-Método 2.

https://www.amazon.es/gp/product/B074QTFRFG

[*Ref.13]* Sistema de Adquisición-Método 3.

http://www.grupoalava.com/ingenieros/productos/imagen-y-fotonica/vision-artificial

XII. Bibliografía Auxiliar

- Documentación Teórica. Asignatura Visión Artificial. Código 370009. Departamento de Electrónica de la Universidad de Alcalá. Año Académico 2017-18

- Documentación de Ayuda MathWorks. Deep Learning, Objetcs Detection and Recognition. Computer Vision System Toolbox. R2017b

- Documentación de Ayuda MathWorks. Image Processing Toolbox. R2017b

- Documentación de Ayuda MathWorks. Classification. Statistics and Machine Learning Toolbox. R2017b

- Milan Sonka, Vaclav Hlavac, Roger Boyle. Image Processing, Analysis, and Machine Vision. Fourth Edition. January 1, 2014.

- Evan Willians, Biz Stone. Medium. Towards Data Science. Machine Learning. 2012

- EliteDataScience. Overfitting in Machine Learning. September 7, 2017

- Salman Khan. KhanAcademy-Maths. Multivariable-calculus. 2006

- Raúl González Duque. Python PARA TODOS. Marzo 18, 2011.

- John V.Guttag. Introduction to Computation and Programming Using Python. Cambridge,Massachusetts; London, England: The MIT Press. Revised and Expanded Edition. August 2, 2013.