MEDICIÓN DEL RENDIMIENTO DE MÉTODOS USADOS PARA LA CLASIFICACIÓN DIGITAL DE OBJETOS CON BAJO CONTRASTE PARA LA EMPRESA JPT C&S

Universidad
del Cauca

CARLOS EDUARDO CAMPO MUELAS

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
PROGRAMA DE INGENIERÍA FÍSICA
POPAYÁN
2017

MEDICIÓN DEL RENDIMIENTO DE MÉTODOS USADOS PARA LA CLASIFICACIÓN DIGITAL DE OBJETOS CON BAJO CONTRASTE PARA LA EMPRESA JPT C&S

CARLOS EDUARDO CAMPO MUELAS.

Trabajo de grado para optar al título de ingeniero físico

Director
Ing. LEONAIRO PENCUE FIERRO

Codirector
Ing. PEDRO RIVERA

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
PROGRAMA DE INGENIERÍA FÍSICA
POPAYÁN
2017

Nota de aceptación:

El Director y los jurados han leído el presente documento, escucharon la sustentación del mismo por su autor y lo encuentran satisfactorio.

Director _____
Ing. Leonairo Pencue Fierro

Codirector _____
Ing. Pedro Rivera

Jurado _____
Ing. Carlos Guzmán

Jurado _____
Mag. Eduardo Castillo

Fecha de sustentación: Popayán, 19 de julio del 2017

Nadie sabe lo que le depara el futuro, es por esto que su potencial es infinito…

MEDICIÓN DEL RENDIMIENTO DE MÉTODOS USADOS PARA LA CLASIFICACIÓN DIGITAL DE OBJETOS CON BAJO CONTRASTE PARA LA EMPRESA JPT C&S

Universidad del Cauca

Carlos Eduardo Campo Muelas

Popayán, julio 2017

Contenido

1. Introducción

Tacto, olfato, gusto, oído y vista son los medios de interacción que desarrolló la raza humana para interactuar con su ambiente, en un entorno dinámico y caótico la correcta lectura del entorno es la clave para conseguir un objetivo establecido; la visión por computador o visión artificial como campo de investigación, pretende emular el sentido de la vista de los seres vivos: detección de objetos, aproximación de distancias, detección de bordes, detección de colores, segmentación de objetos, entre otras; son algunas de las tareas que los investigadores de esta área buscan implementar en un computador [1].

Cada una de estas tareas tiene ciertas características que las hacen especiales, sumado a que para cada una de ellas el tipo de imagen que se debe analizar puede ser diferente (color, dimensiones, escala, etc.), en visión artificial no se cuenta con un algoritmo único que solucione todas estas tareas y que funcione igual bajo todos los ambientes posibles, en contraste existen algoritmos específicos que funcionan para cierto grupo de tareas, con un especifico tipo de imágenes y que además exigen ciertos requisitos en cuanto a características de *hardware*, debido al esfuerzo computacional que ellos implican.

Un tipo especial de imágenes que se analizan, son las imágenes de bajo contraste, las cuales no cuentan con un rango muy amplio en el valor numérico de sus pixeles.

Figura 1. *Turistas tomándose una foto con un fondo falso que oculta la polución real de Hong Kong, la imagen de la polución envolviendo Hong Kong es un ejemplo de imagen de bajo contraste donde los objetos de la imagen tienen un color muy similar al de toda la imagen y los bordes de los objetos no están bien definidos* [2].

Las imágenes digitales a color son la combinación de 3 o más planos, en el caso de las imágenes en escala de grises solo se cuenta con un plano, cada plano tiene un rango numérico establecido; cuando los valores numéricos de cada pixel en la imagen son muy similares es decir están un rango muy estrecho la correcta diferenciación de los bordes de un objeto se hace más difícil (parte superior de la ***Figura 1***), por tanto la extracción de características de un objeto y su posterior clasificación son más complejos que los de una imagen normal o de alto contraste. En cuanto Las fuentes de imágenes de bajo contraste, estas pueden ser variadas: imágenes generadas con ultrasonido, imágenes generadas con rayos-X, imágenes bajo condiciones de vapores y humos o en el caso en el que la empresa JPT C&S planea trabajar que son imágenes de rocas en lodo cuyo color es similar.

El presente trabajo pretende realizar una investigación sobre los métodos de análisis de imágenes que presentan bajo contraste utilizando la metodología de análisis de rendimiento o *benchmark* Comparativo; el método de *benchmark* consiste en analizar un definido grupo de características obtenidas al procesar un mismo grupo de imágenes bajo los algoritmos de clasificación que se quieren comparar, al final se obtiene una ponderación de los métodos en base a sus resultados en el análisis. Con esta ponderación se pretende brindar a la empresa JPT C&S una herramienta para decidir cuál método es más eficiente para el trabajo posterior que van a realizar.

1.1. Sobre la empresa JPT C&S

La empresa JPT *consulting and services* diseña y desarrolla herramientas y equipos para las industrias relacionadas con petróleo y agua; la empresa JPT C&S está enfocada en ofrecer soluciones prácticas e innovadoras que ofrecen buenos estándares de calidad, orientadas a mejorar y optimizar el rendimiento y la productividad de los procesos industriales por medio de un monitoreo constante de las variables físicas que describen esos procesos.

La empresa fue establecida en 2008 en México y McAllen Texas (USA), para proveer una plataforma de servicios de consultoría y desarrollo de tecnología para la industria petrolera, sus productos se enfocaban en el registro de las variables petrofísicas de pozos petroleros. Actualmente la empresa ha adicionado dos nuevas secciones en Colombia: una sede de manufactura en Cali encargada del ensamble en masa de los productos ya desarrollados y una sección de investigación y desarrollo en Popayán encargada del diseño de nuevas herramientas y equipos, su enfoque actual es el desarrollo de: nuevos sensores para las industria petrolera, nuevas técnicas de medición y registro, transmisión de datos sin cableado, telemetría bidireccional en herramientas de registro de pozos, desarrollo de software de registro entre otras áreas relacionadas al campo petrolero y el monitoreo de variables físicas [3].

Su representante legal y cofundador el señor Juan Pablo Torne, lleva cerca de 30 años en la industria de petróleo trabajando en múltiples países para diversas empresas como: *GO international, Gearhart company, Halliburton* y actualmente JPT C&S, ocupando múltiples cargos desde: ingeniero de campo, ingeniero de operaciones, jefe de operaciones y representante legal e ingeniero jefe.

1.2. Benchmark

El *benchmark* es un procedimiento por el cual se analiza el rendimiento de un componente *hardware* o *software*, este análisis se realiza de forma comparativa con otros sistemas similares al que se quiere analizar; posterior a la elección de los sistemas a comparar, se elige una tarea o procedimiento bien definida y repetible para que los sistemas la realicen, la finalidad de este proceso es observar de forma directa las diferencias en los comportamientos (*hardware* o *software*) a medida que avanza el desarrollo de la tarea designada y las características finales cuando se termina la acción [4].

El *benchmark* tiene varias clasificaciones dependiendo del tipo de sistema que comparan en si (modelos matemáticos, *hardware*, *software*, modelos estadísticos, modelos administrativos, etc.); en el caso de los *benchmarks* de *software*, estos pueden ser divididos en 2 clases: los de comparación de bajo y alto nivel respectivamente.

- **Comparación de bajo nivel:** se mide directamente el rendimiento de componentes, como: el reloj de la CPU, el uso de la memoria *ram*, la temperatura de la CPU, el uso de la memoria cache, entre otros.
- **Comparación de Alto nivel:** se centra más a la comparación de parámetros específicos del sistema en general, como: tiempos de respuesta, tiempos de procesamiento, retardos, porcentajes de acierto, porcentajes de error, entre otros.

1.3. Visión artificial

Las imágenes digitales son uno de los más comunes y convenientes medios para transmitir información. Una sola imagen puede contener información equivalente a más de 1000 palabras. Las imágenes brindan información concisa sobre posiciones, tamaños e interrelaciones entre objetos; estas proveen información espacial de pixeles, que nosotros podemos reconocer como objetos, siluetas, bordes, patrones, etc. Los humanos somos especialmente buenos extrayendo información de imágenes debido a nuestras innatas habilidades visuales y mentales; alrededor de un 75% de la información captada por los humanos es de naturaleza visual, los objetivos de la visión artificial son emular y/o mejorar estas capacidades en computadores por medio de algoritmos que utilizan conocimientos físicos, matemáticos y estadísticos [5].

Dependiendo del trabajo que se vaya a realizar, la metodología de trabajo en visión artificial consiste en algunas etapas genéricas:

- **Conversión de información del mundo real a formato digital que pueda ser procesado por un computador**: usualmente es la creación de imágenes digitales o vídeos, usando como medio una cámara fotográfica (infrarrojos, ultravioleta, espectro visible) o algún otro medio (sensores de ultrasonido, receptores de radiación X, foto receptores) etc.

- **Procesamiento de la imagen:** con la imagen en formato digital se procede a eliminar ruido y factores que puedan afectar el análisis de la misma (filtrado), para finalmente analizarla con el algoritmo más propicio al tipo de información que se quiere extraer.

a) ***b)*** ***c)***

Figura 2. *a) Entorno que se quiere analizar, b) medio de conversión información visual a un medio digital c) equipo de procesamiento de la imagen digital. Elaboración propia.*

1.3.1. Sensores ópticos

Convierten una interacción lumínica proveniente del entorno que se está observando, a una señal eléctrica, que es posteriormente compuesta en una imagen digital que representa ese entorno. Los tipos de sensores más comunes son: CCD y CMOS, que son sensores formados en esencia por semiconductores y que están distribuidos en forma de matriz. Su función es la de acumular una carga eléctrica en cada una de las celdas de esa matriz (pixeles), así la carga eléctrica de cada pixel dependerá de la cantidad de luz que incida sobre ella. Los dos tipos son bastante parecidos pero tienen algunas características que los hacen diferentes.

Los sensores tipo CCD (*charge coupled device*) convierten las cargas de las celdas de la matriz en voltajes y entregan una señal analógica como salida, la cual posteriormente es convertida en una imagen digital por la cámara. En este tipo de sensores se hace se hace una lectura de cada uno de los valores correspondientes a cada una de las celdas. Esta información es convertida mediante un conversor AD (analógico–digital) a una secuencia digital de datos; la estructura de este tipo de sensor es muy simple, pero tiene la desventaja de requerir un circuito extra para procesar la información, lo cual se refleja en mayores precios y un aumento de tamaño en la versión final. En rango dinámico[1], los sensores CCD superan a los del tipo CMOS en un margen aproximado de dos veces, en este caso al ser el CCD menos sensible, los extremos de luz los tolera mucho mejor. En cuanto al aspecto de ruido los sensores tipo CCD al contar con un chip extra de procesado de información son menos susceptibles a ser afectados.

En los sensores tipo CMOS cada celda es independiente una de la otra, la principal diferencia con el tipo CCD es que la digitalización de los pixeles se realiza internamente en unos transistores que lleva cada celda, por lo que todo el trabajo (conversión y adaptación) se realiza dentro del sensor y no se hace necesario un chip extra; en cuanto a costos los sensores tipos CMOS son más económicos ya que no requieren un chip extra de acondicionamiento

[1]El rango dinámico es el coeficiente entre la saturación de los pixeles y el umbral por debajo del cual no captan señal.

de la señal a digital, otra de las principales ventajas de los sensores CMOS es su mayor sensibilidad a la luz, por lo que en condiciones pobres de iluminación se comportan mucho mejor, esto se debe principalmente a que los amplificadores de señal se encuentran en la propia celda, por lo que hay un menor consumo a igualdad de alimentación; otra ventaja que se logra con su procesado por celda es la velocidad de captura de imágenes logrando vídeos con una velocidad superior a 1000 fps[1] [6].

1.3.2. Imágenes digitales

Una imagen digital es la representación de un sensado óptico remoto, es típicamente compuesta de elementos de imagen (pixeles) localizados en la intersección de cada fila i, cada columna j y cada banda k de color; cada uno de estos pixeles tiene asociado un valor numérico entero dentro de un rango establecido por el esquema de color de la imagen, este valor es usualmente conocido como brillo y representa la radiación media que golpea un pequeño segmento de área de la imagen; la relación entre estos elementos de imagen y el tamaño general de la imagen define la resolución (definición o calidad) de la imagen para representar pequeños detalles.

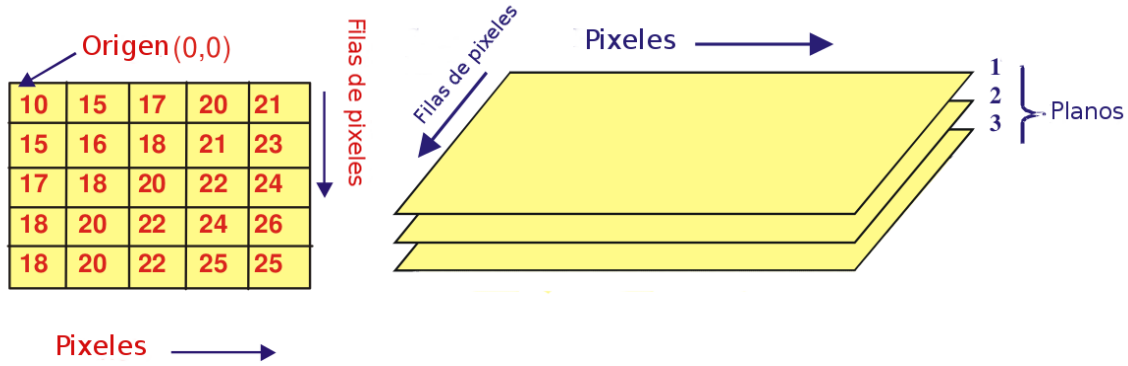


***Figura 3**. Representación de una imagen digital* [5].

Las imágenes digitales se representan como matrices de 3 dimensiones donde las primeras son definidas por las coordenadas de (x, y) de un plano y la tercera coordenada es asignada al plano de color; cada una de las imágenes en los planos son imágenes en escala de grises que al ser traslapadas una sobre otra generan el efecto de color.

a) ***b)*** ***c)*** ***d)***

Figura 4. *a) mesa vibratoria en esquema de color RGB: b) plano rojo, c)plano verde, d)plano azul* [3]. *Imagen proporcionada por la empresa JPT.*

[1]Fotogramas por segundo, es el número de imágenes que se pueden obtener al transcurso de un segundo.

1.3.3. Esquema de color

En el dominio de lo digital, puede haber cualquier número de colores primarios convencionales para formar una imagen; un canal en este caso es similar a una imagen en escala de grises sobre cualquier otro color primario convencional. Por extensión, un canal es cualquier imagen en escala de grises del mismo tamaño que la propia imagen y asociado con esta. El esquema de color hace referencia a la estructura de los canales, los rangos de sus pixeles y el color o la característica de color asociada a dichos canales; cada esquema tiene diferentes combinaciones de estos parámetros, y a su vez cada uno cuenta con ciertas ventajas y desventajas entre sí.

En el esquema de color RGB (*Red, Green, Blue*)**,** una imagen tiene tres canales: rojo, verde y azul como la imagen descompuesta en la ***Figura 4***; los canales RGB deben su creación a la emulación del funcionamiento del ojo humano, el cual también fue implementado en el despliegue de imágenes en monitores TRC (Tubo de Rayos Catódicos) bajo el mismo principio. Si la imagen RGB es de 24 bits (estándar desde 2005) cada canal tiene 8 bits, para el rojo, el verde y el azul; es decir la imagen está compuesta de 3 imágenes (una por cada canal), donde cada imagen puede almacenar pixeles con intensidades de brillo convencional entre 0 y 255. Si la imagen es de 48 bits (alta resolución), cada canal esta hecho de 16 bits.

En el esquema de color CMYK una imagen tiene 4 canales: cían, magenta, amarillo y negro. CMYK es el estándar para imprimir donde se utiliza la síntesis sustractiva de color. Una imagen CMYK de 32 bits está compuesta de canales de 4 canales de 8 bits, uno para el cían, uno para el magenta, uno para el amarillo y uno para el negro.

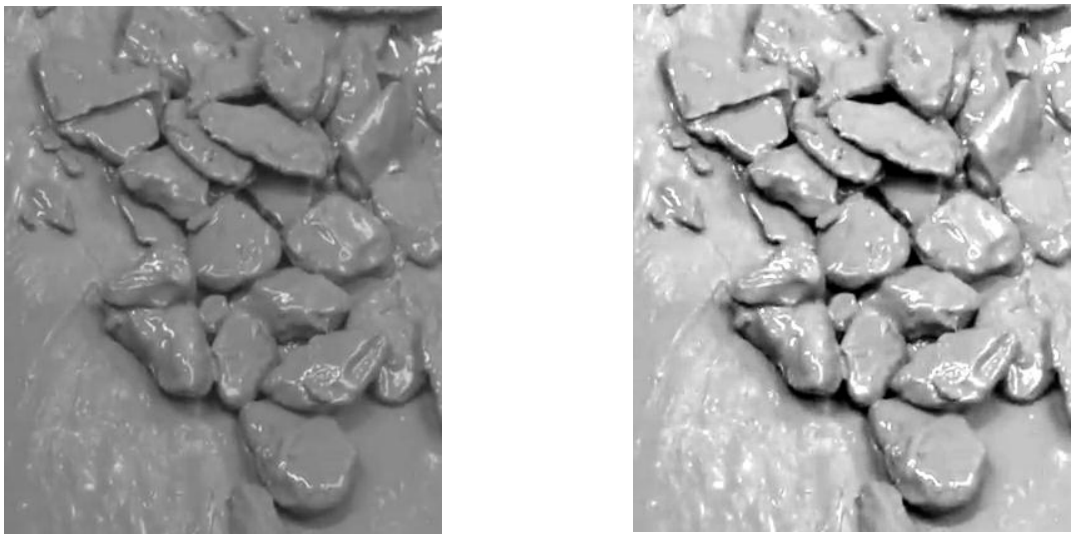
En el esquema de color HSV (*Hue, Saturation, Value*), la información se almacena en tres tipos de canales, igual que en el esquema RGB, aunque se diferencian en que el esquema HSV tiene un canal dedicado al brillo (*value*) y los otros dos transmiten la información del color. El color (*Value)* es el mismo que el canal de negro en el esquema CMYK. El esquema HSV es especialmente útil en compresión de video con pérdida, donde la pérdida de información del color es menos perceptible al ojo humano [7].

1.3.4. Procesamiento digital de imágenes

Las imágenes digitales pueden entenderse como matrices, cada uno de los puntos que componen esta matriz se llaman pixeles; estos pixeles tiene asociado un numero entero que representa la intensidad de luz que este refleja (o emite como se quiera entender); cuando se procesa una imagen estos pixeles son modificados: aumentando o disminuyendo su intensidad o en algunos casos reemplazados totalmente.

1.3.4.1. Operaciones básicas

Las operaciones básicas más usadas son las operaciones aritméticas y geométricas; las operaciones aritméticas, en esencia se basan en un aumento o disminución en el valor de la intensidad de los pixeles: lo cual se utiliza en enfoques de bordes, realce de objetos, entre otras tareas, ver ***Figura 4.***



a) ***b)***

Figura 4. *a) Imagen de rocas en escala de grises (Imagen original), b) Imagen realzada disminuyendo el valor de los pixeles cercanos a cero y aumentando el valor de los pixeles cercanos al máximo valor. Imagen proporcionada por la empresa JPT*

En cuanto a las operaciones geométricas están se basan en el fundamento de las imágenes digitales como matrices; por lo cual las imágenes digitales están sujetas a todas las operaciones matriciales que se pueden encontrar en el álgebra lineal: rotación, traslación y escalamiento, entre las más conocidas [8].

La traslación se obtiene cuando se multiplica la imagen original con la matriz de transformación ***Mt*** (ecuación 1), donde *tx* y *ty* son las coordenadas x, y, a las que se quiere desplazar la imagen respecto al centro de coordenadas.

$$\boldsymbol{Mt} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \boldsymbol{tx} \\ \mathbf{0} & \mathbf{1} & \boldsymbol{ty} \end{bmatrix} \qquad (1)$$

La rotación de una imagen se obtiene cuando esta se multiplica por la matriz de transformación ***Mr*** (ecuación 2), donde Ɵ representa el ángulo que se va a rotar la imagen:

$$\boldsymbol{Mr} = \begin{bmatrix} \boldsymbol{cos\theta} & \boldsymbol{-sin\theta} \\ \boldsymbol{sin\theta} & \boldsymbol{cos\theta} \end{bmatrix} \qquad (2)$$

El cambio en la escala de una imagen se logra cuando se reduce o aumenta ordenadamente el número de pixeles intermedios, este proceso utiliza la interpolación de un valor intermedio en el caso de disminución de tamaño y la extrapolación de nuevos valores intermedios para el aumento de dimensiones de la imagen.

1.3.4.2. Filtrado de imágenes

El filtrado de imágenes se refiere al proceso mediante el cual se eliminan o se atenúan efectos indeseados en la imagen, como saturación de brillos, puntos blancos y negros en la imagen, desvanecimiento de bordes, extracción de objetos por su color, entre otras.

a) ***b)*** ***c)*** ***d)***

Figura 5. *a) imagen de una roca redondeada (original), b) imagen rotada, c) imagen trasladado su centro de coordenadas, d) imagen con un factor de escalamiento de ½. Elaboración propia.*

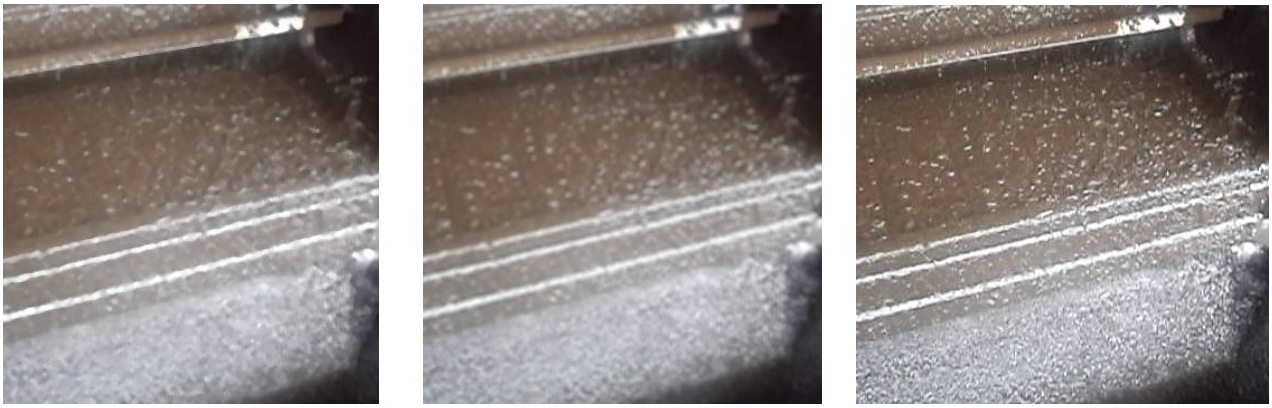
Cuando se conocen de antemano los rangos de intensidad en los que se encuentran un grupo de pixeles los cuales se quieren segmentar, o en el caso del ruido sal-pimienta[1] que se quieren eliminar, se hace uso del proceso de umbralización que consiste en la eliminación de pixeles que cumplen cierta característica (umbral de color definido, bajo o alto valor de intensidad).

a) ***b)*** ***c)***

Figura 6. *a) imagen de rocas (original), b) imagen umbralizada para valores menores a 50, c) imagen umbralizada para valores menores a 100. Elaboración propia.*

Cuando se requiere eliminar ruido en toda la imagen se procede a realizar un filtrado general, tomando pequeñas porciones de la imagen y analizándolas buscando pixeles cuyo valor no corresponda con el comportamiento promedio de los pixeles en esa pequeña porción. El análisis del comportamiento promedio de los pixeles se realiza utilizando herramientas de la estadística tal es el caso de la media, la mediana, la campana gaussiana, entre otros. Por lo general este tipo de filtrado retorna una imagen con los bordes difuminados, pero con un comportamiento continuo [1].

[1] Ruido sal-pimienta: o ruido impulsorial, es un tipo de ruido que añade pixeles con el máximo valor (blanco) y con el mínimo valor (negro) de forma aleatoria en la imagen



a) ***b)*** ***c)***

Figura 7*. a) mesa vibratoria con lodo (original), b) imagen con filtro de media, c) imagen con filtro mediana. Imagen proporcionada por la empresa JPT.*

1.3.4.3. Transformaciones morfológicas

Las transformaciones morfológicas, son operaciones sencillas basadas en la modificación de la forma de objetos en una imagen binaria[1]; estas operaciones se realizan utilizando una segunda imagen llamada *kernel* de transformación o elemento de estructuración, el cual es el encargado de recorrer la imagen a modificar y redistribuir el valor de sus pixeles, utilizando la respuesta de la interacción del *kernel* de transformación y el segmento de imagen analizado. Las transformaciones morfológicas más importantes son la dilatación y la erosión, ellas a su vez son base de otras transformaciones que se logran de la combinación sucesiva de estas dos [1].

- Dilatación: Es la operación mediante la cual los objetos en la imagen, aumentan de tamaño en sus bordes, este proceso se logra cuando la interacción del *kernel* indica que hay al menos un pixel de valor 1 en el área que interactúo con el *kernel*, si esto sucede los pixeles en esa área toman el valor de 1, aumentando de esta forma el tamaño de los bordes de los objetos.

- Erosión: es la operación inversa a la dilatación, elimina los pixeles en los bordes de los objetos, cuando los objetos son lo suficientemente pequeños (ruido) son eliminados completamente; este proceso se logra cuando la interacción con el *kernel* indica que no todos los pixeles en el área que interactúo son 1, es decir al menos un pixel tiene un valor de cero, entonces todos los pixeles toman el valor de cero.

- Apertura**:** es el resultado de aplicar una secuencia sucesiva de una erosión y luego una dilatación, como ya se había mencionado la erosión elimina pequeñas partículas en la imagen, pero también reduce los bordes de los demás objetos por tanto es necesario dilatarlo de nuevo para obtener un objeto con la misma forma que el objeto inicial pero sin las pequeñas partículas de ruido.

[1] Una imagen binaria es una imagen con un solo plano de color en donde solo hay dos posibles valores: máximo (1) y mínimo (0).

- Cierre: es el resultado de aplicar una secuencia sucesiva de una dilatación y una erosión, con el proceso de dilatación se eliminan pequeños huecos en los objetos de la imagen pero los objetos se deforman haciéndose más grandes por tanto es necesaria una erosión para volver el objeto a una forma similar al objeto inicial pero sin los pequeños huecos en los objetos de la imagen.

- Gradiente morfológico: es la resta entre el resultado de una dilatación y una erosión, por lo cual este proceso nos genera una imagen con solo los bordes de la imagen.

a) ***b)*** ***c)***

Figura 8. *a) Logo JPT binario (original) b) Logo JPT después de dilatación c) Logo JPT después de una erosión. Elaboración propia.*

a) ***b)***

Figura 9. *a) Logo JPT con ruido (putos negros) b) Logo JPT después de un proceso de apertura. Elaboración propia.*

a) ***b)***

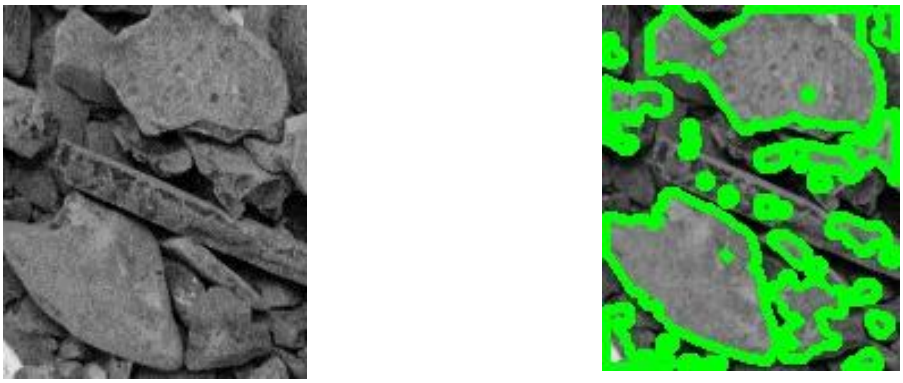
Figura 10. *a) Logo JPT con huecos blancos (ruido) b) Logo JPT después de proceso de cierre. Elaboración propia.*

a) ***b)***

Figura 11. *a) Logo JPT b) Logo JPT después del proceso gradiente morfológico (Extracción de bordes). Elaboración propia.*

1.3.4.4. Extracción de contornos

Un contorno es la silueta o bordes de grupo de pixeles que tienen un mismo valor o rango numérico, son especialmente útiles en análisis de forma para la detección y reconocimiento de objetos especialmente en imágenes binarias; los contornos a su vez son muy livianos de analizar, ya que resumen las características de todo un objeto en solo unos cuantos pixeles [9].



a) ***b)***

Figura 12. *a) Recortes obtenidos de una excavación b) contornos de la imagen resaltados en color verde. Elaboración propia.*

1.3.5. Clasificación de objetos en una imagen

La clasificación de objetos dentro de una imagen requiere por lo general una lista de los objetos en la imagen (detección) y sus respectivas posiciones en la imagen; este proceso por lo general es el resultado de un análisis de contornos u otros análisis similares, los cuales brindan información más concreta sobre las características de los objetos: forma, geometría, centros de masa, parámetros morfológicos, entre otros; con esta información numérica y la comparación de la misma en las categorías posibles, se puede clasificar cada uno de los objetos [10].

1.3.5.1. Parámetros morfológicos

Los parámetros morfológicos son combinaciones de parámetros relativos de tamaño: perímetro, área y diámetros; con la combinación de estos parámetros básicos y el uso de algunas relaciones matemáticas, se puede obtener cualidades más complejas y que brinden mayor información como: la redondez de un objeto, simetría, similitud con elipses, rugosidad, solidez y excentricidad[1]. La clasificación de objetos utilizando parámetros morfológicos funciona muy bien cuando las imágenes analizadas no tienen luces direccionales[2], ya que la presencia de ella distorsiona los valores numéricos en una dirección preferencial. Los parámetros morfológicos más importantes son: factor de forma, redondez, inverso de la relación de aspecto, relación de área, esfericidad, solidez y excentricidad [8].

[1] Excentricidad: Es un coeficiente que representa el grado de desviación de una sección cónica con respecto a una circunferencia.

[2] Luces direccionales: Luces no centralizadas, es decir la fuente de luz procede de una alguna de las esquinas de la imagen.

- Factor de forma: hace referencia a la relación entre el área y el perímetro del objeto (***ecuación 3***), buscando cuan cercano es este valor, al obtenido al analizar con la misma ecuación un círculo perfecto.

$$F = \frac{4 * \pi * A}{p^2} \qquad (3)$$

*Donde **A**=Área, **p**=perímetro del objeto.*

- Redondez: analiza las curvaturas de los bordes del objeto, buscando cuan similares son las curvas del objeto con las de un círculo perfecto (***ecuación 4***), calculando el índice de relación entre el área del objeto y la de un circulo que encierre completamente al objeto; el valor máximo que puede tomar este coeficiente es 1.

$$R = \frac{4 * A}{\pi * Dmax^2} \qquad (4)$$

*Donde **A**=Área del objeto, **Dmax**=diámetro máximo del objeto.*

- Inverso de la relación de aspecto: es un índice de cuan alargado o achatado es el objeto, comparando el menor y el mayor diámetro del objeto (***ecuación 5***).

$$IRa = \frac{Dmen}{Dmax} \qquad (5)$$

Donde **Dmen**, **Dmax** son el diámetro mínimo y máximo respectivamente, del objeto.

- Relación de área: indica cuan parecido es el objeto analizado con un rectángulo, la idea es analizar cuan cerrados son sus ángulos de una forma indirecta (***ecuación 6***).

$$Ra = \frac{A}{ARect} \qquad (6)$$

Donde **A**=Área del objeto, **ARect** es el área de un rectángulo que encierra el objeto.

- Esfericidad: indica cuan lejana es la relación entre los diámetros máximo y mínimo del objeto (***ecuación 7***), cuanto más cercano a 1 sea este valor indicara cuan cerca está el objeto a parecerse a un circulo.

$$E = \sqrt{\frac{Dmen}{Dmayor}} \qquad (7)$$

Donde **Dmen**, **Dmax** son el diámetro mínimo y máximo respectivamente, del objeto.

- Solidez: es un índice de cuan regular son los bordes de un objeto (***ecuación 8***), se calcula tomando los puntos más lejanos del objeto y formando con ellos un área que contenga todos los puntos del objeto, a esta región se le denomina envolvente convexa; al

final se calcula una relación entre el área real del objeto y el área de la envolvente convexa, para descubrir cuan sólidos son los bordes del objeto al acercarse este valor a 1.

$$\boldsymbol{S = \frac{A}{EnvCov}} \qquad (8)$$

Donde **A**=Área del objeto, ***EnvConv***=Área de la envolvente convexa.

- Excentricidad: se calcula dibujando una elipse que contenga al objeto y midiendo cuan lejano esta uno de sus focos con su eje mayor.

Para hacer una clasificación utilizando los parámetros morfológicos como medio de decisión, es necesario hacer una caracterización previa de las clases de objeto en las que un objeto puede ser clasificado; Una correcta caracterización establece una serie de rangos de los parámetros morfológicos para los cuales se puede asegurar con cierta certeza, que un objeto pertenece a una clase u otra.

1.3.5.2. Clasificación mediante estrategias computacionales

La clasificación de objetos en una imagen puede ser implementada de otras formas indirectas, tal es el caso del uso de estrategias computacionales; a la colección de estos algoritmos se le conoce como Aprendizaje automático (*machine learning*). El área del aprendizaje automático, es un área de la computación que no solamente es usada en la visión artificial, sino también en otras áreas como: el modelamiento matemático, automatización inteligente, análisis de estadísticas y probabilidades, entre otras.

En visión artificial las operaciones están incluidas y optimizados en paquetes computacionales optimizados, tal es el caso de: Opencv[1] [11], Skimage[2] [12], Numpy y Scipy[3][13], entre otros.

[1] OpenCV: OpenCV fue iniciado en Intel en 1999 por Gary Bradsky y la primera versión fue liberada en 2000. Vadium Pisarevsky trabajo con Gary Bradsky dirigiendo el equipo de OpenCv Ruso de Intel. En 2005, OpenCV fue usado en Stanley, el vehículo que gano gran el campeonato de 2005 DARPA. Después su activo desarrollo continuo bajo el soporte de Willow Garage, con Gary Bradsky y Vadium Pisarevsky liderando el proyecto. Actualmente, OpenCV brinda soporte a muchos algoritmos relacionados con Visión por Computador y Aprendizaje de máquina y continúa expandiéndose día a día. Actualmente OpenCV trabaja con una gran variedad de lenguajes de programación como C++, Python, Java, entre otros; además está disponible en diferentes plataformas incluyendo Windows, Linux, OS X, Android, iOS etc.

[2] Skimage: Skimage o Scikit-image es un paquete computacional para el procesamiento de imágenes en lenguaje de programación Python. Al igual que su lenguaje de programación base, está enfocado en una interfaz más interpretativa, estructurada y de fácil lectura.

[3] Numpy y Scipy (Numerical Python y Scientific Python): son un par de paquetes computacionales, usados en el procesamiento numérico en algoritmos desarrollados en lenguaje de programación Python, estos paquetes contienen una colección de algoritmos y convenientes funciones pre-construidas y optimizadas, permitiendo al desarrollador el uso de poderosas funciones matemáticas sin la necesidad de implementar extensos códigos que ya están embebidos en estas librerías.

La metodología de clasificación utilizando aprendizaje automático en visión artificial se resume en dos etapas: una etapa de entrenamiento y una etapa de clasificación, la etapa de clasificación utiliza los resultados de la etapa de entrenamiento previa, para clasificar nuevas imágenes; la idea fundamental es utilizando imágenes de entrenamiento ya analizadas y con todos los objetos en ellas clasificados correctamente, generar un algoritmo capaz de clasificar los objetos de una nueva imagen sin que el desarrollador le indique al algoritmo que parámetros debe utilizar para realizar dicha clasificación, de esta forma el algoritmo se asegura de elegir los mejores parámetros.

1.4. Aprendizaje automático

Aprendizaje automático es la disciplina científica que se enfoca en buscar y mejorar los métodos para programar sistemas que aprendan automáticamente y que puedan solucionar problemas utilizando experiencias anteriores; se acoge el nombre de aprendizaje aunque no es un aprendizaje como tal, en el caso del aprendizaje de maquina el aprendizaje se refiere a buscar con antelación (entrenamiento) patrones complejos de comportamiento de algunas características a la entrada que tiene el sistema, de esta forma estableciendo correctamente el patrón de comportamiento el sistema puede de forma inteligente tomar decisiones basados en los datos previamente analizados. El principal problema en el área del aprendizaje automático radica en que el conjunto de todas las posibles decisiones dadas por todas las posibles entradas es muy complejo de describir; para resolver esas adversidades el campo de aprendizaje automático busca desarrollar algoritmos que se enfoquen en un problema o en un conjunto de datos específicos; utilizando análisis estadísticos y algunos principios computacionales (dependiendo el tipo de entrada del sistema los métodos pueden variar) se caracteriza este conjunto de datos de entrada y se plantea un patrón de comportamiento, entre más selectivo y bien definido sea este conjunto de datos se puede asegurar una mayor fiabilidad en las predicciones del sistema resultante.

El área de aprendizaje automático integra el conocimiento de muchas otras metodologías, como por ejemplo: Teoría de probabilidad, lógica, optimización por combinatorias, estadística, teoría de control, patrones de búsqueda, y algunos otros menos usados; así también los métodos desarrollados en Aprendizaje automático son la base de muchas otras aplicaciones: aplicaciones de visión por computador, procesamiento de lenguaje, agrupamiento de patrones, reconocimiento de patrones, juegos, análisis de datos, sistemas expertos y robótica [14].

1.4.1. Historia

En 1946 el primer sistema computacional ENIAC fue construido, en ese tiempo la palabra computador se refería a la persona que convertía las computaciones numéricas de ENIAC a el papel, por esa razón ENIAC fue llamado una máquina de computación numérica; esta máquina era manualmente operada, una persona (computador) hacia las conexiones entre las partes de la máquina para que ENIAC hiciera las computaciones. La idea en ese tiempo era que el pensamiento y el aprendizaje humano podían ser compilados en una máquina.

En 1950 Alan Turing propuso un test para medir cuan cerca estaba una computadora a la capacidad de aprendizaje humano. El test de Turing estaba basado en la idea de que únicamente se podía determinar si una maquina podía realmente aprender si al comunicarnos con

ella no somos capaces de distinguir si se trata de una maquina o de otro humano. En ese tiempo no hubo ningún sistema que pasara el test de Turing pero se desarrollaron muchos interesantes sistemas para intentar aprobar el test.

Alrededor de 1952 Arthur Samuel (IBM) escribió el primer programa-juego de ajedrez que contenía suficiente habilidad para retar al campeón mundial de ajedrez de ese tiempo; los programas de aprendizaje que Samuel desarrollo trabajaron increíblemente bien y fueron una ayuda para medir el rendimiento de los jugadores de ajedrez. Otro increíble sistema desarrollado en los tempranos 60 fue ELIZA por Joseph Weizebaum, ELIZA simulaba ser un psicoterapeuta, usando trucos como sustitución de palabras y combinación de análisis basados en palabras claves, era capaz de dar respuestas y consejos, cuando ELIZA apareció por primera vez algunas personas que interactuaron con ella alcanzaron a confundirla con una persona; La ilusión de inteligencia trabajaba muy bien, sin embargo cuando la conversación se limitaba a hablar sobre cosas de la vida diaria, la naturaleza de ELIZA era descubierta, una fácil prueba de concepto.

Posterior a ello, muchos otros sistemas fueron desarrollados, uno de los más importantes fue el desarrollado por el grupo MYCIN (Stanford) liderado por Ted Shortliffe, los cuales demostraron el poder de los sistemas basados en reglas para la representación e inferencia del conocimiento en el dominio de la medicina, en el diagnóstico y la terapia. Este sistema fue el primero en ser denominado sistema experto. Al mismo tiempo que los sistemas expertos eran desarrollados otro tipo de aproximación al aprendizaje automático emergía, en 1957 Frank Rosenblatt invento el perceptrón en el Laboratorio de aeronáutica Cornell; el perceptrón es un clasificador linear muy sencillo pero este demostró que la combinación de un gran número de ellos en una red podía dar origen a modelos más complejos, redes neuronales.

La investigación de las redes neuronales, a pesar de ser muy prometedora, perdió fuerza después que Marvin Minsky y sus colegas demostraron que las redes neuronales no podían resolver problemas como el problema XOR[1] , el problema radicaba en que la combinación de perceptrones podía únicamente resolver problemas que son linealmente separables (existe una línea o hiperplano que separa los grupos de datos) el cual no era el caso de la función XOR, a fines de los 70's y principios de los 80's se descubre que con la adición de una capa oculta de perceptrones se podía entrenar redes neuronales para resolver cualquier tipo de problemas (incluso el XOR).

Ya cerca de los 90's el campo de aprendizaje automático se convertía en un camino muy popular que unía la ciencia de los computadores y la estadística, esta concurrencia dio resultado a un nuevo camino de pensamiento "la Inteligencia artificial"; basados en aproximaciones probabilísticas, en este nuevo sistema de pensamiento fue adicionado el concepto de incertidumbre a los modelos; todo el campo de aprendizaje automático fue reformado con la adición de esta nueva metodología, así como la característica de la necesidad de bases de datos más grandes para reducir el valor de la incertidumbre; muchos de los algoritmos actuales de aprendizaje automático se basan en ideas de esa época [15].

[1] El problema xor hace referencia a la obtención de los resultados de la función lógica OR exclusiva: un valor es falso si su par de entradas es igual de lo contrario es verdadero [15].

1.4.2. Modelos Generalizados

El área de aprendizaje automático tiene una gran cantidad de diversos modelos dependiendo el tipo de entrada de datos, el tipo de análisis que se va a realizar, entre otras características; en esta sección se da un énfasis en los modelos cuya meta es clasificar [16].

1.4.2.1. Modelos lineales generalizados

Los modelos lineales son principalmente usados en procesos de regresiones, su principal característica es que la salida del sistema es una combinación lineal de sus entradas.

$$y(x) = a_0 + a_1x_1 + a_2x2 + \ldots + a_nx_n \qquad (9)$$

Algunas de las técnicas que se utilizan en este tipo de modelos son: regresión por mínimos cuadrados, regresión por mínimos cuadrados con un factor de ajuste, entre otros.

La regresión por mínimos cuadrados consiste en buscar los coeficientes a_n de la ***ecuación 9***, componiendo la línea recta que mejor describa el comportamiento de todos los puntos, lo cual se logra escogiendo los parámetros de la línea de tal forma que la distancia entre los puntos y la recta sea mínima.

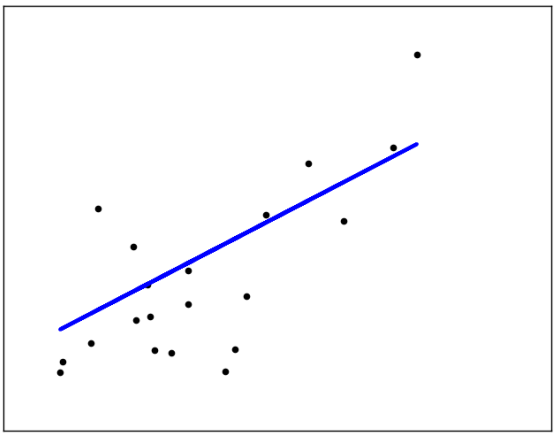


Figura 13. *Elección de la línea recta que mejor describe el comportamiento de los puntos* [16].

1.4.2.2. Máquinas de soporte vectorial

Las máquinas de soporte vectorial o SVM (*support vector machine*), son un grupo de métodos de aprendizaje supervisado, usados comúnmente en clasificación, regresión y detección de elementos fuera de un patrón; Las ventajas de las SVM son:

- efectividad en espacios de gran número de dimensiones (funciona muy bien con elementos que tienen un numero grande de características para analizar)
- Aun efectivo cuando el numero dimensiones es más grande que el número de muestras para el entrenamiento
- Usa un sub-grupo de puntos de entrenamiento en la función de decisión (denominados SVM), por lo tanto es muy eficiente en cuanto a consumo de recursos computacionales.

- Es versátil, la función de decisión acepta como entrada diferentes tipos de *kernels* lógicos, es decir diferentes tipos de análisis pueden ser realizados con la misma función.

Entre las desventajas de este tipo de método tenemos:

- si el número de formas dentro de la imagen es mucho mayor que el número de muestras, los resultados que ofrece el método de SVM es muy pobre.
- SVM no provee directamente una probabilidad para una predicción, este las estima usando las probabilidades de todas las categorías (en el caso de una clasificación).

Una forma de visualizar el método de clasificación de los SVM, es usando un grupo de objetos que tenga solo dos propiedades o características asociadas y después se gráfica en un plano los puntos donde coinciden esos valores; este proceso se realiza para cada uno de las muestras de entrenamiento previamente clasificadas de forma manual, posterior a realizar este proceso se deberían ver acumulaciones de puntos en diversas zonas del plano referentes a cada una de los tipos de objeto; ahí es donde empieza a trabajar los SVM, con la información de la posición de cada acumulación se ubican líneas (vectores) que las separen de la mejor manera, lo más equitativamente posible; este graficado equivale a la parte del entrenamiento en un algoritmo; por tanto la única información que se necesita para hacer una predicción posterior son la posición de esas líneas (SVM) que se dibujaron para separar las categorías. Así si se dibuja el nuevo punto que se desconoce su categoría y dependiendo de la posición en el plano en la que se ubique, se podrá hacer una predicción, tomando como referencia su posición con respecto a los SVM ya marcados.

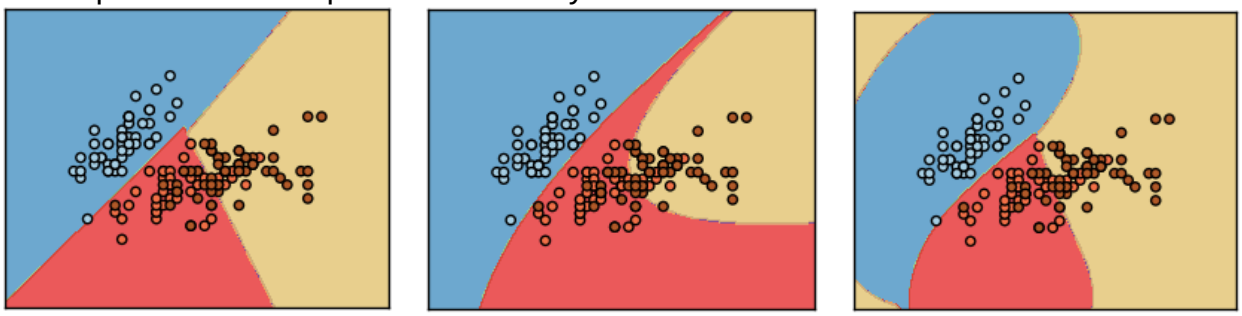


Figura 14. *a) Clasificación usando un kernel lineal, b) clasificación usando un kernel exponencial, c) clasificación usando un kernel polinomial* [17].

Existes diversas formas de dibujar la línea (SVM) y para hacerlo se hace uso de los denominados *kernels*, cada uno de ellos genera un patrón de comportamiento como el que se puede observar en la ***Figura 14***; dependiendo del tipo de clasificador que se necesite la elección del *kernel* puede variar.

El método de los SVM de forma matemáticamente estricta, no construye un híper-plano sino un conjunto de híper-planos en un espacio n-dimensional, donde cada uno de estos planos se usara para clasificar, hacer regresión u otras tareas. Intuitivamente, se puede deducir que una buena separación es requerida por los híper-planos la cual es la distancia más larga entre los puntos de entrenamiento de cada clase (función margen), la distancia entre este "margen" y el híper-plano (SVM) será el valor de incertidumbre [18].

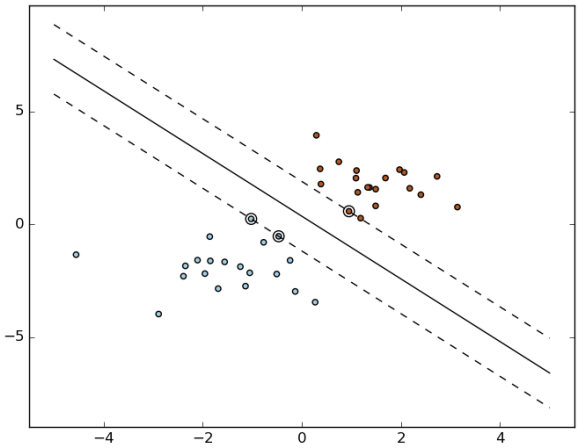


Figura 15. *Elección de los vectores que separan dos tipos de valores, línea punteada márgenes, línea solida SVM* [17].

1.4.2.3. Gradiente estocástico descendente (SGD)

El SGD (*Stochastic Gradient Descend*) es un simple pero eficiente método de aproximación al aprendizaje de los clasificadores lineales, aumentando la posibilidad de que los resultados converjan a uno de los tipos de elementos posibles, el método de SGD ha rondado por los algoritmos del campo de aprendizaje automático desde bastante tiempo atrás, pero últimamente ha aumentado su atención debido a sus características frente a grandes cantidades de datos.

El método de SGD ha sido satisfactoriamente aplicado para solucionar problemas con el manejo de grandes cantidades de variables y a problemas de aprendizaje automático en clasificación textual y procesamiento de expresiones del lenguaje natural; dada la naturaleza de la estructura de este método y su acepción de datos dispersos, los clasificadores con este tipo de método como base pueden resolver problemas con más de 10^5 muestras de entrenamiento y 10^5 tipos de formas en las muestras [19].

Las ventajas del SGD son:

- eficiencia
- fácil implementación, los parámetros y estructura del método son fácilmente modificables para cada tipo de clasificación.

Las desventajas del SGD incluyen:

- requieren un número de híper-parámetros extra, los cuales cambian el comportamiento del modelo de forma drástica, como el coeficiente de regularización, el número de iteraciones[1], entre otros.
- Es muy sensible a cambios de escala, es decir una misma figura o comportamiento pero amplificada o disminuida puede desfasar de la predicción hecha con la figura original.

[1] Iteración: repeticiones de una acción.

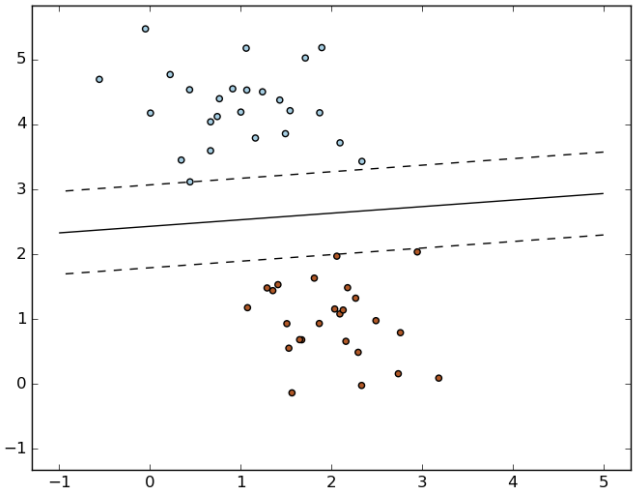


Figura 16. *Separación de dos clases de elementos, utilizando SGD* [19].

A diferencia de la clasificación con SVM, la clasificación con SGD se basa en separación utilizando rutinas de aprendizaje con comportamiento de gradiente estocástico[1] descendente, funciones de pérdida y penalizaciones. Las funciones de perdida más comunes son: margen suave (utiliza SVM para calcular los bordes, ***Figura 16***), suavizado bisagra, por regresión logística y la combinación de todas las anteriores. Los primeros dos tipos de función son funciones básicas, establecen únicamente el valor de los parámetros si se presenta una violación del valor constante de los márgenes, lo cual puede dar lugar a separaciones de muy pocas concentraciones; La ***Figura 17*** muestra un esquema de clasificación para 3 tipos de pétalos, e ilustra las separaciones implementadas con SGD.

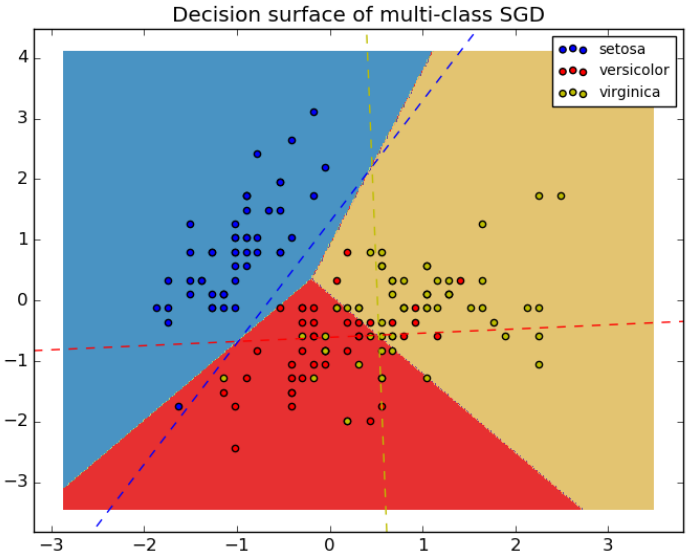


Figura 17. *Esquema de clasificación de pétalos de 3 tipos de flores utilizando SGD* [19].

[1] Estocástico: sistema cuyo comportamiento es no determinista, en la medida que el subsiguiente estado del sistema está determinado por acciones predecible del proceso como de elementos aleatorios.

Dado un grupo de muestras de entrenamiento representadas por la ***ecuación 10***

$$(\boldsymbol{x_1},\ \boldsymbol{y_1}),\ldots,(\boldsymbol{x_n},\boldsymbol{n}) \qquad \boldsymbol{x_i}\ \in\ \boldsymbol{R^n} \qquad (10)$$

Y una clasificación de dos opciones está representada por la ***ecuación 11***

$$\boldsymbol{y_i}\ \in\ \{-\mathbf{1},\mathbf{1}\} \qquad (11)$$

La meta es que el sistema aprenda a clasificar los elementos, dado la función de ponderamiento lineal de la ***ecuación 12***

$$f(x) = w^T x + b \qquad donde\ \ w \in \boldsymbol{R}^m, \qquad b\ \in \boldsymbol{R} \qquad (12)$$

Para realizar una predicción simplemente se mira el signo de *f(x);* una elección común para la elección de los parámetros del modelo es minimizar el valor del error de entrenamiento de la regularización dado por la ***ecuación 13***

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w) \qquad (13)$$

Donde la función L es la función de perdida que mide el modelo con respecto a cuanto se aleja la predicción del valor real, R es un término de regularización (penalización) que restringe el modelo cuando este se torna complejo; α > 0 es híper-parámetro no negativo.

Como se había mencionado antes la función L se puede elegir según las características que el clasificador, que se va a diseñar necesite (ver ***Figura 18***).

- *Hinge*: márgenes suaves, usando SVM para detectar los límites de las acumulaciones.
- *Log*: utiliza regresión logística para detectar los límites de las acumulaciones.
- *Least-squares*: utiliza una regresión enfatizando los límites exteriores de las acumulaciones.
- *Epsilon-insensitive*: bordes suaves, utilizando SVR, una regresión tomando como base los vectores de soporte que indican los límites de las acumulaciones.

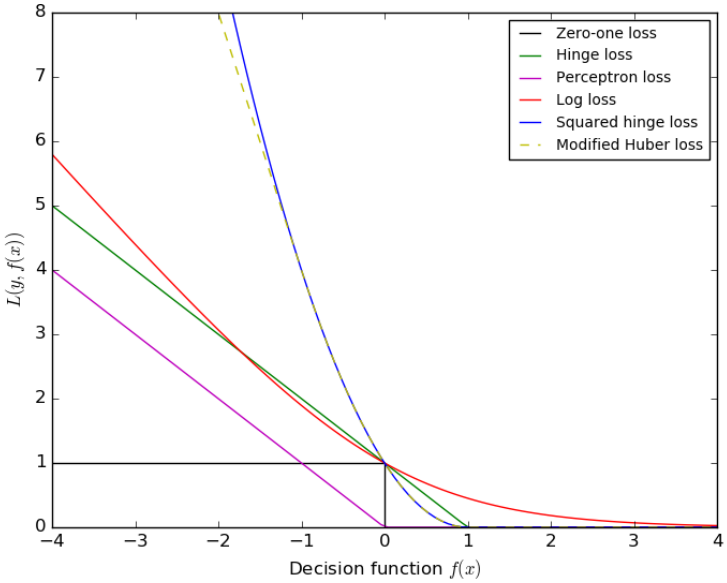


Figura 18. *Gráfica de la función de decisión vs los diferentes modelos de la función de perdida* [19].

1.4.2.4. Vecinos cercanos

El principio base detrás de *Nearest Neighbors* (NN), es encontrar un predefinido número de muestras de entrenamiento que encierren un nuevo punto y predecir la clase de este utilizando las clases de los puntos; el número de estas muestras, usualmente está definidos por el usuario, dado a que este valor controla directamente la sensibilidad y estabilidad de las predicciones, en su defecto el modelo utiliza una densidad de puntos de todas las muestras de entrenamiento y establece un límite con ese valor.

A pesar de su simplicidad NN ha sido satisfactoriamente usado en un gran número de problemas de clasificación y regresión, incluyendo detección de caracteres escritos a mano, análisis de imágenes satelitales, entre otras tantas; siendo un método no paramétrico (no requiere parámetros adicionales) tiene alta confiabilidad en problemas de clasificación donde las áreas de una clase no están bien definidas por simples líneas [20].

Los métodos basados en NN son un tipo de aprendizaje no generalizado o aprendizaje basado en instancias, el cual no se centra en generar un modelo interno genérico para usar como comparación, simplemente guarda algunas instancias del entrenamiento; la clasificación es computada de una simple votación por mayoría de los vecinos más próximos para cada punto que solicite una predicción (Ver ***Figura 19***).

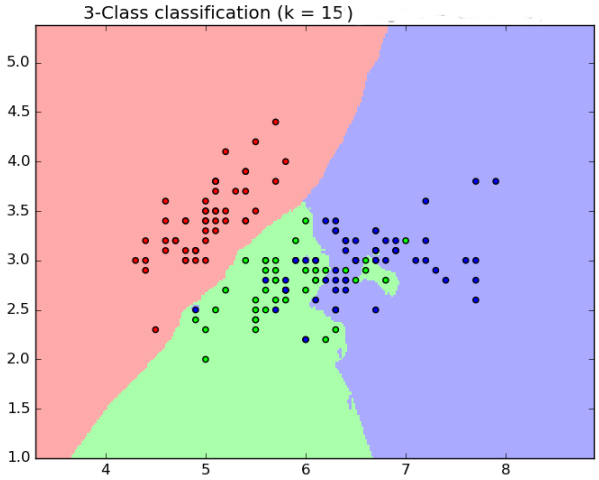


Figura 19. *Separación de muestras por sus respectivas clases (enfatizadas en sus colores)* [20].

1.4.2.5. Naive bayes

Son métodos de aprendizaje no supervisado los cuales se basan en aplicar el teorema de Bayes con la suposición de Naive, la cual indica que en un grupo de muestras, cada par de muestras que se elijan son independientes y no se relacionan directamente.

Dada una clase representada por la letra *y,* y un vector de forma independiente x_1 hasta x_n, el teorema de Bayes establece las relaciones de la ***ecuación 14***:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)} \qquad (14)$$

Usando la suposición de Naive de independencia entre formas

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y), \qquad (15)$$

Para todo *i* esta relación es simplificada a

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)} \qquad (16)$$

Donde la entrada es constante

$$P(x_1, \ldots, x_n) \qquad (17)$$

Entonces se puede utilizar la siguiente regla de clasificación

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \qquad (18)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y), \qquad (19)$$

Para completar el análisis, se realiza una estimación MAP (*Maximum A Posteriori*) para estimar

$$P(y) \quad P(x_i \mid y) \qquad (20)$$

Los diferentes clasificadores basados en Naive Bayes difieren únicamente de la suposición que ellos hacen sobre la función de distribución

$$P(x_i \mid y) \qquad (21)$$

Por el contrario a su aparente sobre simplificadas suposiciones los clasificadores basados en Naive Bayes han trabajado muy bien en problemas de la vida real, como la clasificación de documentos y filtros de *spam* en los *e-mails*; ellos requieren una pequeña cantidad de información como datos de entrenamiento para estimar los parámetros necesarios de todo el sistema (problemas) [21].

A continuación se ilustra los 2 tipos de probabilidades que se pueden implementar en los clasificadores Naive Bayes: Los NB basados en la función de distribución Gaussiana ***(ecuación 22)***

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \qquad (22)$$

Y los NB basados en la función de distribución de Bernoulli ***(ecuación 23)***

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i) \qquad (23)$$

1.4.2.6. Arboles de decisión

Son un método no paramétrico de aprendizaje supervisado, usado principalmente en clasificadores y regresiones (ver ***Figura 20***), la meta de este tipo de sistemas es crear un modelo que prediga el valor de una variable objetivo utilizando unas simples reglas de decisión inferidas en un entrenamiento [22].

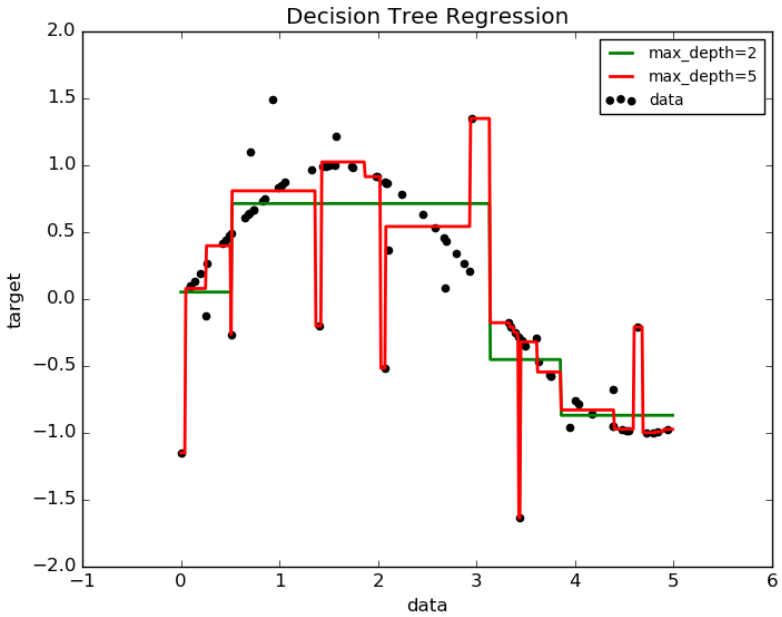


Figura 20. *Árbol de decisión utilizado en regresión de valores de una función sinusoidal con ruido a diferentes niveles de aprendizaje* [22].

Las ventajas de los árboles de decisión son:

- simples de entender e interpretar, los arboles de decisión pueden ser visualizados.
- Requieren pocos datos de preparación. Otras técnicas por lo general requieren una normalización de los datos, la creación de variables temporales, y los valores en blanco removidos.
- El costo computacional de usar el árbol de decisión para predecir valores es logarítmico con respecto al número de puntos usados para el entrenamiento.
- Capaz de manejar datos numéricos, como por categorías.
- Capaz de manejar problemas de múltiples salidas.
- Usa un modelo de caja blanca donde el desarrollador puede observar como se establece el modelo, a diferencia de las redes neuronales (caja negra) que es imposible determinar la razón de sus configuraciones.

Las desventajas de los árboles de decisión, incluyen:

- en la etapa de aprendizaje se puede dar lugar a arboles de decisión demasiado complejos que no representan correctamente el comportamiento del sistema, a este error se le conoce como sobreajuste.
- Pueden ser inestables a pequeñas variaciones.
- Tienen conceptos que son difíciles de comprender dado a que los arboles de decisión no los explican fácilmente, como el problema del XOR, problemas de multiplexores.

A continuación se ilustra el resultado de un entrenamiento utilizando arboles de decisión para la clasificación de flores utilizando la longitud de sus pétalos ***Figura 21, Figura 22***.

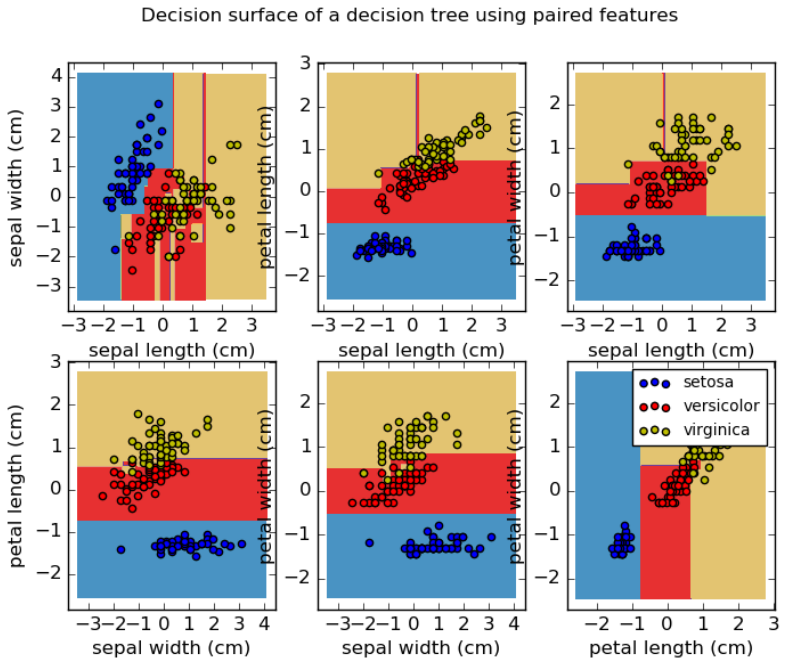


Figura 21. *Clasificador de plantas por las dimensiones de sus pétalos usando un árbol de decisión* [22].

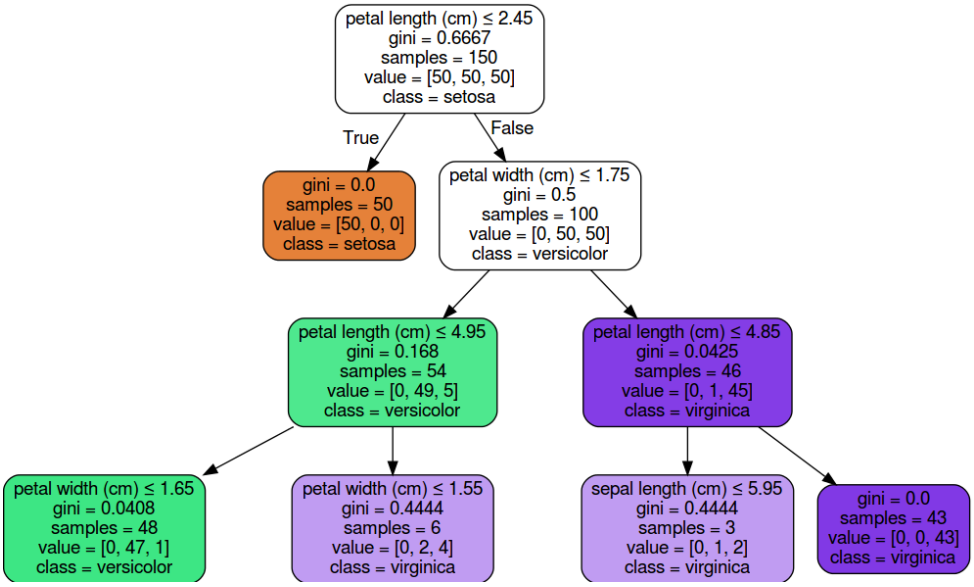


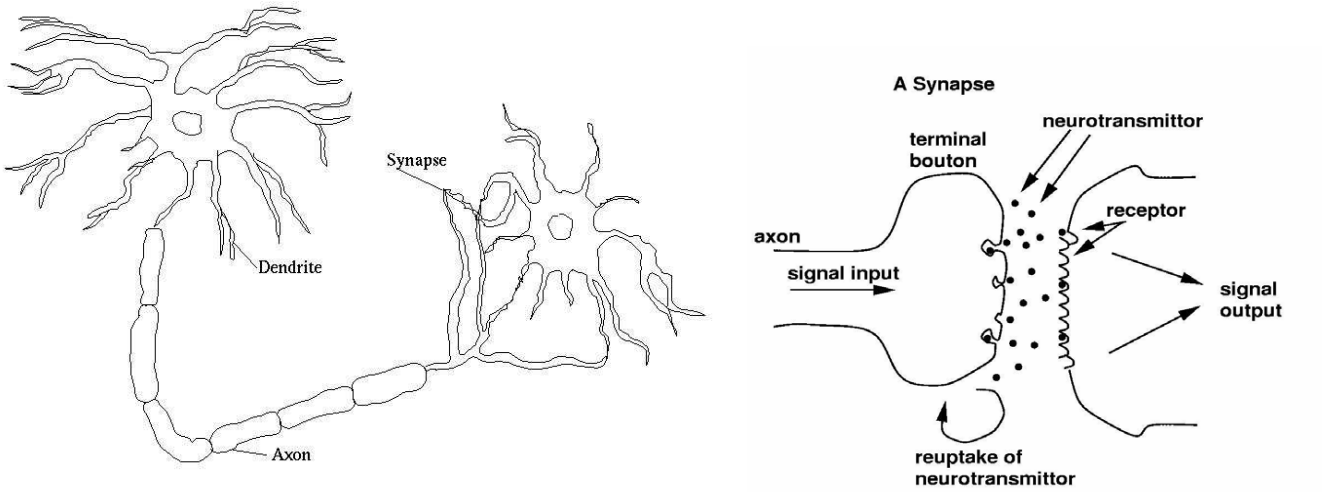
Figura 22. *Árbol de decisión obtenido como resultado de entrenamiento de clasificador de plantas con la longitud de sus pétalos como entrada* [22].

1.4.2.7. Redes neuronales

Las Redes Neuronales (NN: *Neural Networks*) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros; el primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de actividad nerviosa, este modelo era un modelo binario, donde cada neurona tenía un escalón o umbral prefijado. Las redes neuronales son modelos que se basan o están inspirados en sistemas biológicos, como el caso del primer modelo; aunque últimamente estos modelos se han convertido en entes netamente artificiales, enfocados principalmente a una tarea en especial, la cual decide su estructura y requerimientos.

Para entender la estructura de las redes neuronales es preciso, entender el sistema biológico que fue base para ello, el cerebro humano; el cerebro humano contiene más de cien mil millones (10x11) de neuronas y 10x14 sinapsis en el sistema nervioso. Los estudios realizados sobre la anatomía del cerebro humano concluyen que hay, en general, más de 1000 sinapsis por término medio a la entrada y a la salida de cada neurona.
Aunque el tiempo de conmutación de las neuronas biológicas (unos pocos mili segundos) es casi un millón de veces menor que en los actuales componentes de las computadoras, las neuronas naturales (biológicas) tienen una conectividad miles de veces superior a la de las artificiales.

El objetivo principal de las redes neuronales de tipo biológico es desarrollar operaciones de síntesis y procesamiento de información, relacionadas con los sistemas biológicos [23].



a) ***b)***

Figura 23. *a) Estructura de una neurona, b) esquema de la sinapsis neuronal* [23].

Las neuronas y las conexiones entre ellas (sinapsis) constituyen la clave para el procesado de la información. La mayor parte de las neuronas poseen una estructura de árbol, llamada dendritas, que reciben las señales de entrada procedentes de otras neuronas a través de las sinapsis. Una neurona consta de tres partes (ver ***Figura 23.a***):

1. El cuerpo de la neurona
2. Las dendritas, que reciben las entradas
3. El axón, que lleva la salida de la neurona a las dendritas de otras neuronas.

La forma completa en la que dos neuronas se relacionan no es totalmente conocida, y depende además del tipo particular de cada neurona. En general, una neurona envía su salida a otras por su axón, y éste lleva la información por medio de diferencias de potencial eléctrico (ver ***Figura 23.b***).

Este proceso es a menudo modelado como una regla de propagación representada por una función u(t). La neurona recoge las señales por su sinapsis sumando todas las influencias excitadoras e inhibidoras, si las influencias excitadoras positivas dominan, entonces la neurona produce una señal positiva y manda este mensaje a otras neuronas por sus sinapsis de salida; en este sentido, la neurona actúa como una simple función escalón f(t).

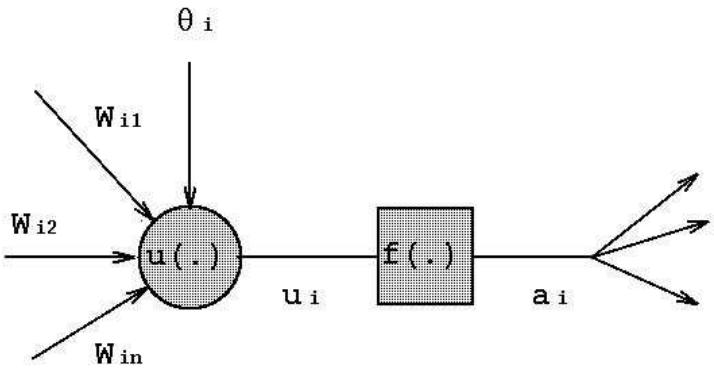


Figura 24. *Esquema gráfico del comportamiento matemático que representa a una neurona* [23].

Las redes neuronales aplicadas están, en general, inspiradas en las redes neuronales biológicas, aunque poseen otras funcionalidades y estructuras de conexión distintas a las vistas desde la perspectiva biológica. Las características principales de las redes neuronales artificiales son las siguientes:

1. Auto-Organización y Adaptabilidad: utilizan algoritmos de aprendizaje adaptativo y auto-organización, por lo que ofrecen mejores posibilidades de procesado robusto y adaptativo.
2. Procesado no Lineal: aumenta la capacidad de la red para aproximar funciones, clasificar patrones y aumenta su inmunidad frente al ruido.
3. Procesado Paralelo: normalmente se usa un gran número de nodos de procesado, con alto nivel de ínter conectividad.

El elemento básico de computación (modelo de neurona) se le llama habitualmente nodo o unidad. Recibe un input desde otras unidades o de una fuente externa de datos, cada input tiene un peso asociado w, que se va modificando en el llamado proceso de aprendizaje (ver ***ecuación 24***). Cada unidad aplica una función dada f de la suma de los inputs ponderadas mediante los pesos.

$$y_i = \sum_j w_{ij} y_j \qquad (24)$$

Por lo general se pueden definir 2 fases o etapas en la creación de un sistema de redes neuronales [23]:

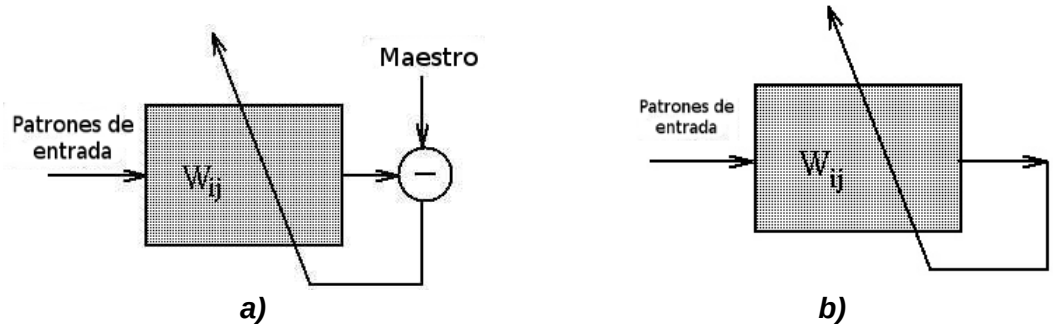
- Fase de entrenamiento: se usa un conjunto de datos o patrones de entrenamiento para determinar los pesos (parámetros) que definen el modelo de red neuronal. Se calculan de manera iterativa, de acuerdo con los valores de los valores de entrenamiento, con el objeto de minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada.

- Fase de Prueba: en la fase anterior, el modelo puede que se ajuste demasiado a las particularidades presentes en los patrones de entrenamiento, perdiendo su habilidad de generalizar su aprendizaje a casos nuevos (sobre ajuste). Para evitar el problema del sobre ajuste, es aconsejable utilizar un segundo grupo de datos diferentes a los de entrenamiento, el grupo de validación, que permita controlar el proceso de aprendizaje. Normalmente, los pesos óptimos se obtienen optimizando (minimizando) alguna función de energía; por ejemplo, un criterio muy utilizado en el llamado entrenamiento supervisado, es minimizar el error cuadrático medio entre el valor de salida y el valor real esperado.

Las redes neuronales pueden ser de dos tipos de aprendizaje supervisado y no supervisado, las redes neuronales de entrenamiento supervisado son las más populares. Los datos para el entrenamiento están constituidos por varios pares de patrones de entrenamiento de entrada y de salida (ver ***Figura 25.a***). El hecho de conocer la salida implica que el entrenamiento se beneficia de la supervisión de un "maestro" (usuario). Dado un nuevo patrón de entrenamiento, en la etapa (m + 1)- ésima, los pesos se adaptan bajo la forma de la ***ecuación 25***:

$$w_{ij}^{m+1} = w_{ij}^{m} + \Delta w_{ij}^{m} \qquad (25)$$

Para los modelos de entrenamiento no supervisado, el conjunto de datos de entrenamiento consiste sólo en los patrones de entrada. Por lo tanto, la red es entrenada sin el beneficio de un maestro [23]. La red aprende a adaptarse basada en las experiencias recogidas de los patrones de entrenamiento anteriores (ver ***Figura 25.b***).



a) ***b)***

Figura 25. *a) Esquemático del sistema de aprendizaje de una red neuronal supervisada b) esquemático del sistema de aprendizaje de una red neuronal no supervisada* [23].

1.5. Métricas de evaluación

Las métricas de evaluación son parámetros que se calculan para medir la calidad o el rendimiento de un grupo de algoritmos de tal forma que puedan ser ponderados por orden de importancia. En el caso de los algoritmos clasificadores, las métricas de evaluación permiten analizar una correcta clasificación además de detectar comportamientos anómalos [24].

Para calcular estas métricas se requiere el conteo de:

- **Positivos verdaderos (tp):** cada una de las muestras de una clase que fueron correctamente clasificadas.
- **Falsos positivos (fp):** número de muestras a las cuales los algoritmos les asignaron una clase pero no pertenecen a ella.
- **Negativos verdaderos (tn):** cada una de las muestras que no pertenecen a una clase y el algoritmo clasificador las detecto como tal.
- **Falsos negativos (fn):** número de muestras que pertenecen a una clase pero que el algoritmo clasificador predijo que no.

Las métricas más importantes son: Porcentaje de acierto, Precisión y recuperación o sensibilidad [25]:

- **Porcentaje de acierto:** es un valor que relaciona el número total de predicciones de una clase con el número total de muestras de esa clase, como se indica en la ***ecuación 26.***

$$\%\boldsymbol{Ac} = \frac{\boldsymbol{tp}+\boldsymbol{fp}}{\boldsymbol{Total\ muestras}} \qquad (26)$$

- **Precisión:** es la relación entre las clasificaciones correctas generadas por un clasificador y el total de las predicciones de una clase, permite analizar cuan fiable son las predicciones de un clasificador, su rango es de 0 a 1(ver ***ecuación 27***).

$$\boldsymbol{Pr} = \frac{\boldsymbol{tp}}{\boldsymbol{tp}+\boldsymbol{fp}} \qquad (27)$$

- **Sensibilidad:** es la relación entre las clasificaciones correctas y la totalidad de muestras de una clase, permite analizar dado un set de muestras cuantas de ellas son percibidas por los clasificadores, es decir cuan sensible es un clasificador con respecto a una clase (ver ***ecuación 28***).

$$\boldsymbol{S} = \frac{\boldsymbol{tp}}{\boldsymbol{tp}+\boldsymbol{fn}} \qquad (28)$$

2. Metodología de desarrollo

Una etapa muy importante en el desarrollo de un nuevo producto tecnológico es una correcta investigación de los caminos que se pueden seguir en su construcción; esta investigación tiene como objetivo cuantificar y analizar las características que el producto puede llegar a tener si se decide implementar un proceso u otro; la empresa JPT C&S con varios años en el desarrollo de nuevas herramientas y equipos, consiente en que un análisis previo antes del desarrollo de cualquier equipo es la clave para garantizar no solo la calidad del producto si no también un ahorro en tiempo y costos, plantea un estudio para la selección de un algoritmo de clasificación de piedras en una imagen obtenida en el proceso de extracción de los residuos sólidos de una operación de taladrado de un pozo petrolero, este estudio es la investigación previa de un nuevo proyecto en el cual trabaja la empresa JPT C&S para inferir el estado de un pozo que se está perforando.

El presente estudio fue planteado de tal forma que se utilizaran imágenes similares a las que podrían presentarse en el proceso real, creando imágenes a partir de muestras de piedras obtenidas de un proceso de taladrado facilitadas por la empresa; estas imágenes fueron de dos tipos: con piedras muy bien espaciadas (ver ***Figura 26.a***) y con acumulaciones de piedras (ver ***Figura 26.b***), las cuales representan los escenarios de un proceso de taladrado lento con un flujo de residuos sólidos regular y un proceso de taladrado rápido con gran flujo de partículas.

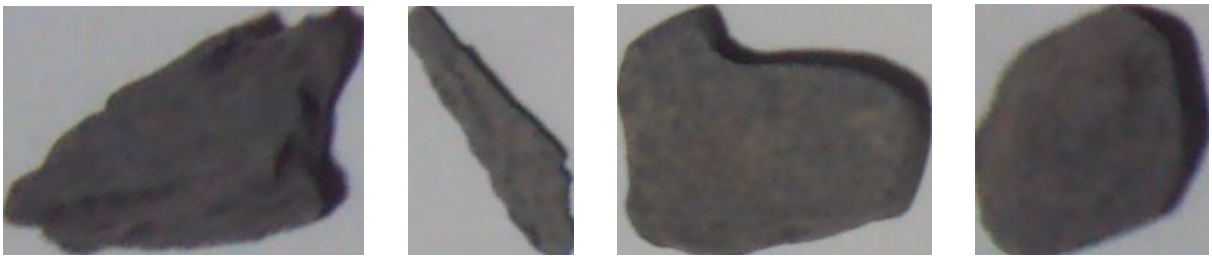
a) ***b)***

Figura 26. *a) Imagen de prueba con partículas bien espaciadas, b) imagen de prueba con partículas en acumulaciones. Elaboración propia.*

Además de elegir los tipos de imágenes similares a los que podrían presentarse en el caso real, también se eligió utilizar la misma clasificación que la empresa requiere para hacer la inferencia del estado del proceso de taladrado del pozo, es decir 4 clases de rocas: rocas angulares, rocas astillosas, rocas blocosas y rocas redondeadas ver ***Figura 27***.

- **Rocas angulares:** se caracterizan por ser rocas muy planas con un gran número de salientes y ángulos pronunciados en sus esquinas.
- **Rocas astillosas:** son rocas muy delgadas, cuya superficie está casi descrita con dos puntas muy pronunciadas y un cuerpo liso en su intermedio.

- **Rocas blocosas:** son rocas que tienen como preferencia caras perpendiculares lisas y con salientes en ángulos cercanos a los 90°.
- **Rocas redondeadas:** son rocas muy circulares (sin llegar al círculo perfecto) que se caracterizan por una superficie suave con pocas deformaciones.



a) ***b)*** ***c)*** ***d)***

Figura 27. *Rocas proporcionadas por la empresa JPT C&S: a) roca angular, b) roca astillosa, c) roca blocosa, d) roca redondeada. Elaboración propia.*

Un algoritmo de clasificación es un procedimiento para seleccionar una hipótesis de un grupo de posibilidades basándose en si las propiedades encajan mejor en un grupo de características u otro; desde el auge de las computadoras hasta ahora se han creado muchos y diversos algoritmos de clasificación, y cada uno de ellos se comporta diferente ante la naturaleza del tipo de clasificación en la que se les emplee. El tipo de clasificación que la empresa JPT C&S planea implementar es la clasificación de objetos en una imagen, con variaciones en algunas características como: la cantidad de objetos en la imagen, el espaciado entre los objetos y además una característica especial: las imágenes son de bajo contraste; esta última característica se debe a que en el proceso de taladrado se inyecta un fluido (lodo) para que ayude en el proceso: removiendo las piedrecillas producto del taladrado, evitando que el pozo colapse, y enfriando y limpiando el taladro [26], cuando las piedras son extraídas tienen el mismo color del lodo por lo cual en el momento de la captura de la imagen las partículas no pueden ser muy bien definidas (bajo contraste).

La clasificación de objetos en imágenes con bajo contraste es un complejo problema que no tiene un camino pre-establecido para solucionarse, por ello siempre es una buena práctica implementar diversos algoritmos y elegir el que ofrezca mejores resultados; para seleccionar los algoritmos de clasificación que se estudiaron en este trabajo se utilizaron como base investigaciones del rendimiento de algunos de los métodos de clasificación más populares; cada una de estas investigaciones revela comportamientos ligeramente distintos de los clasificadores y características que pueden ser útiles en el caso de la clasificación en imágenes de bajo contraste; tenemos en primera instancia el clasificador naive bayes que es muy rápido tanto en su etapa de entrenamiento como en su clasificación, su precisión es buena y mejora cuando el grupo de muestras de entrenamiento aumenta [27], con características similares también se resalta el clasificador basado en gradiente estocástico descendente (SGD), que es un clasificador lineal generalizado que ejecuta SGD como método de reducción de error, su ejecución es muy rápida y el uso de recursos computacionales es poco con respecto a otros algoritmos, su precisión varía en un rango de acierto que depende de la fase de entrenamiento [28]. Otro clasificador que se utiliza con mucha frecuencia es el clasificador basado en arboles de decisión, que se caracteriza por obtener buenos modelos ante problemas de multi-clasificación e incluso con una gran cantidad de datos de entrenamiento [29]; otro clasificador que según los estudios no disminuye su rendimiento ante grandes cantidades de entrenamiento ni ante cambios de escala en las imágenes, es el clasificador basado

en vecinos cercanos [30], además dado a que no se entrena un modelo como tal tiende a dar mejores resultados ante casos donde las clases no sean linealmente separables [27]. Y finalmente los clasificadores que obtuvieron mejor rendimiento en las investigaciones [25][26][27][28][29] las máquinas de soporte vectorial (SVM) y redes neuronales, en las investigaciones se resalta su robustez ante el ruido en los datos y ante anomalías aleatorias [27], referente al clasificador SVM estas características se deben su fundamento de garantizar siempre la mayor separación de las clases [30].

Utilizando las investigaciones mencionadas se seleccionaron 6 tipos de clasificadores de las listas que estos estudios plantean, utilizando como métrica de selección la exactitud y la susceptibilidad a cambios de escala en los vectores de entrenamiento[1], además se ha agregado un clasificador extra ya analizado por la empresa Ecopetrol [8] copartidaria del proyecto en el cual planea trabajar la empresa JPT C&S; dando un total de 7 clasificadores a los cuales se les realizó el estudio comparativo para analizar sus características, los tipos de clasificadores seleccionados fueron:

- clasificador basado en análisis de factores de forma
- clasificador basado en máquinas de soporte vectorial
- clasificador basados en SGD
- clasificador basado en un análisis de vecinos cercanos (NN)
- clasificador basado en Naive Bayes
- clasificador utilizando un árbol de decisión
- clasificador utilizando una red neuronal

En cuanto a las métricas de comparación de los algoritmos clasificadores, estas se seleccionaron de tal forma que la comparación no se diera solo por inferencia numérica sino también utilizando métricas que la empresa JPT C&S pudiera ponderar en un rango de importancia, como: la preferencia por la correcta detección: que es muy importante dado a que a mayor número de partículas detectadas mayor va a ser la cantidad de partículas a las cuales se les pueda clasificar, la sensibilidad de los clasificadores a ciertas clases: ya que esta métrica permite analizar cuan sensible es un clasificador a objetos de ciertas clases que pueden ser de mayor relevancia y el uso de los recursos computacionales: lo cual permite elegir el hardware correcto para cada clasificador; en base a ese análisis las métricas de comparación que se utilizaron fueron:

- porcentaje de detección de los objetos en la imagen
- Sensibilidad en cada clase de objeto
- promedio del tiempo requerido para completar entrenamiento
- promedio del tiempo requerido para hacer una clasificación
- uso de recursos computacionales

El estudio inició con la creación de las imágenes de entrenamiento y prueba, después se desarrollaron: los algoritmos de entrenamiento, los algoritmos clasificadores y los algoritmos de comparación; finalmente se procedió sucesivamente con cada uno de los clasificadores a: entrenar el algoritmo de clasificación y clasificar los objetos en las imágenes de prueba, midiendo en cada una de estas etapas las métricas de comparación elegidas.

[1] Vectores de entrenamiento: son vectores con las características que describen una muestra de entrenamiento.

Todos los algoritmos de: entrenamiento, clasificación, medición de rendimiento y de testeo, se diseñaron en lenguaje de programación Python3.5 y se utilizaron paquetes computacionales externos como scipy, numpy, scikit-learn, y OpenCV para algunas funciones extra. Las características de *hardware* del equipo en el que se realizaron las pruebas fueron: un *NoteBook Samsung* modelo NP355E4C con: 4 Gb de memoria RAM DDR3, procesador doble nucleo AMD E1-1200 APU with Radeon(tm) *HD Graphics*, tarjeta de gráficos Gallium 0.4 on AMD PALM (DRM 2.43.0, LLVM 3.8.0), Sistema Operativo base Ubuntu 16.04 LTS 64-bit.

2.1. Creación de las imágenes de entrenamiento y prueba

Las imágenes utilizadas en el estudio fueron creadas a partir de muestras de piedras de un proceso de taladrado, pertenecientes a la empresa JPT C&S; las imágenes se capturaron con una cámara SONY DSC-W180 de 12 megapíxeles con una iluminación controlada de dirección vertical; el procedimiento de construcción de las imágenes vario ligeramente dependiendo de si las imágenes eran: de entrenamiento, de prueba con objetos espaciados y de prueba con acumulaciones de partículas:

- **Imágenes de entrenamiento:** las imágenes de entrenamiento son imágenes con fondo gris y una sola partícula en su centro (ver ***Figura 27***), para su construcción primero se seleccionaron las muestras, teniendo en cuenta que el número de muestras por cada clase debe ser igual para que no se generen preferencias en los clasificadores; el número máximo de muestras por clase se redujo a 45 en concordancia con la cantidad de partículas de clase redondeado las cuales eran muy escasas en el set de muestras de la empresa JPT C&S; posterior a esto, se procedió a capturar las imágenes de las partículas en un fondo blanco en una determinada cantidad por cada clase, variando las posiciones y ángulos de las partículas (ver ***Figura 28***).

La separación y generación de imágenes individuales para cada una de las muestras en las imágenes, se realizó por medio del ***algoritmo 1*** de la sección de apéndices, el cual recibe una imagen con múltiples objetos y a partir de esta: ubica cada uno de los objetos en la imagen, recorta cada muestra de la imagen original y las guarda con extensión *.png*.

- **Imágenes de prueba con objetos bien espaciados:** estas imágenes contienen un numero aleatorio de partículas de clases aleatorias, para su generación se procedió a seleccionar una cierta cantidad de muestras de piedras tratando de conseguir un número similar de cada clase, posteriormente de forma aleatoria se eligieron de ese grupo un numero aleatorio de partículas las cuales se ubicaron en posiciones y ángulos aleatorios, para finalmente capturar la imagen (ver ***Figura 26.a***). se tomaron 40 imágenes de prueba con objetos bien espaciados, utilizando piedras diferentes en cada captura (5 por cada clase).

- **Imágenes de prueba con objetos en acumulaciones:** a diferencia de las imágenes con objetos bien espaciados, las imágenes con acumulaciones de partículas se crearon adicionando en primera instancia un fondo de piedras mucho más pequeñas en comparación con las piedras objetivo (ver ***Figura 26.b***); estas piedras objetivo se ubicaron en el fondo de piedrecillas, en posiciones y ángulos aleatorios, tratando de mantener un número similar de partículas de cada clase en el total de imágenes. Al igual que

las imágenes con objetos bien espaciados se tomaron 40 imágenes como set de prueba, utilizando piedras diferentes en cada captura (4 por cada clase).

Figura 28. *Imagen con 15 partículas de clase astilloso bien espaciadas. Elaboración propia.*

2.2. Desarrollo de los algoritmos de entrenamiento

El proceso de entrenamiento de un clasificador basado en aprendizaje automático es la etapa mediante la cual se modifican los valores internos del clasificador minimizando el error entre sus predicciones y los valores reales, a diferencia de estos clasificadores el clasificador basado en factor de forma opera diferente; en el clasificador basado en factor de forma se calibran los rangos de los descriptores de forma y se elige una estructura de decisión para separar una nueva partícula en una clase u otra (ver ***Algoritmo 3*** de la sección de apéndices).

2.2.1. Algoritmo de entrenamiento para el clasificador basado en factores de forma

El proceso del ***Algoritmo 3*** inicia con una detección de la partícula en la imagen de entrenamiento; utilizando el método de umbralización mencionado en la sección ***1.3.4.2*** filtrado, se transforma la imagen en binaria y se halla el máximo contorno el cual representa la forma de la partícula; posteriormente se calcula los valores de área, perímetro, área del cuadrado que encierra la partícula y área convexa de la partícula, con estos parámetros se procede a calcular el valor de los siguientes descriptores de forma: inverso del factor de forma, relación de área y solidez, mencionados en la sección ***1.3.5.1*** parámetros morfológicos, finalmente se almacena los valores de los descriptores de forma y se procede con la siguiente imagen de entrenamiento. Cuando se ha calculado el valor de los descriptores de forma de todas las imágenes de entrenamiento, el algoritmo genera los rangos normales de estos para cada clase, y la estructura de decisión.

2.2.2. Algoritmo de entrenamiento para los clasificadores basados en SVM, SGD, Naive Bayes, Vecinos cercanos y árboles de decisión

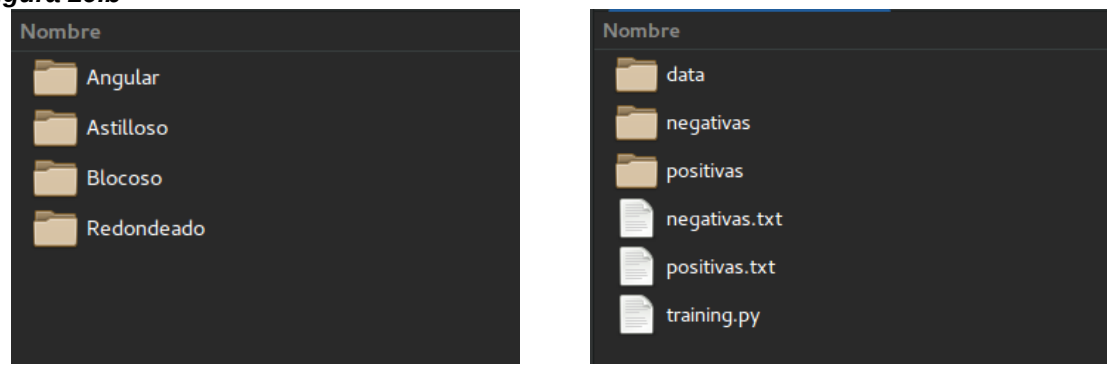
Estos algoritmos de aprendizaje automático utilizan la librería scikitlearn de aprendizaje automático como base para su implementación, por lo cual su estructura es similar exceptuan-

do el uso de algunos híper-parámetros referentes a cada clasificador, ver los ***algoritmos 5, 6, 7, 8 y 9*** de la sección de apéndices.

Para mejorar el rendimiento de estos clasificadores primero se debe procesar las imágenes de entrenamiento con HOG[1] (***Histogram Oriented to Gradient***), este procesamiento genera un vector de características mucho más pequeño que la imagen original el cual conserva la información de la forma de la partícula, este procesamiento ayuda a acelerar la etapa de entrenamiento y a reducir el uso de recursos computacionales ***(ver Algoritmo 4***). Con las imágenes transformadas en los vectores de características por el ***Algoritmo 4***, se procede a generar un modelo de predicción ajustando sus valores con la comparación del resultado de sus predicciones y la clase real de las partículas de entrenamiento.

2.2.3. Algoritmo de entrenamiento para el clasificador basado en redes neuronales

El clasificador basado en redes neuronales utiliza la librería de procesamiento de imágenes OpenCV para la generación de los modelos de clasificación; esta librería cuenta con funciones especializadas para la generación de los modelos de las redes neuronales (ver ***Algoritmo 10***). Para su implementación primero se creó una estructura de carpetas similar al de la ***Figura 29.a*** y ***Figura 29.b***



a) ***b)***

Figura 29. a) *Estructura de carpeta principal,* ***b)*** *Estructura de carpeta para cada clase. Elaboración propia.*

En la carpeta principal se crearon cuatro subcarpetas correspondientes a las cuatro clases de piedras del estudio (***Figura 29.a***), en cada una de estas se crearon tres subcarpetas y dos archivos extra (***Figura 29.b***): la carpeta "data" tiene como función almacenar el modelo generado por el proceso de entrenamiento, en la carpeta "positivas" se ubican las muestras de la clase para la cual se va crear el modelo, en la carpeta "negativas" se ubican las imágenes del resto de clases, en el archivo "negativas.txt" se escriben los nombres de las imágenes en la carpeta "negativas" y en el archivo "positivas.txt" se escriben los nombres de las imágenes en la carpeta "positivas" junto con el tamaño y la posición del centro de la imagen; posterior a esta re-ubicación de archivos, se procede a utilizar los algoritmos de OpenCV para generar los modelos, (ver ***Algoritmo 10***).

[1] El análisis HOG (*Histogram Oriented to Gradient*) primero aplica filtros Sobel en las direcciones X, Y, para realzar los bordes del objeto en la imagen, posterior a ello, se calcula la dirección y magnitud del gradiente de cada pixel, ese gradiente es cuantizado a un valor entero entre 1 y 16, luego se parte la imagen original en pequeñas celdas del mismo tamaño, por lo general cuatro celdas, y para cada una de las celdas se calcula el histograma de las magnitudes de sus pixeles y se forma un vector con la unión de cada uno de los histogramas de las celdas [31]

2.3. Desarrollo de los algoritmos de clasificación

Antes de iniciar el proceso de clasificación se deben implementar unas etapas previas a las imágenes; primero se inicia con un pre procesamiento (filtrado) de la imagen, posteriormente se seleccionan todos los contornos en esta (ver ***sección 1.3.4.4***) y se segmentan todas las posibles partículas para ser analizadas una a una por los clasificadores.

2.3.1. Clasificador basado en análisis de factores de forma

El clasificador basado en la comparación directa de parámetros morfológicos, es un clasificador bastante sensible a cambios direccionales de iluminación, es decir la exactitud de sus clasificaciones puede bajar si la iluminación tiene una dirección preferencial diferente a la orientación de la iluminación bajo la cual se capturaron las imágenes de entrenamiento.

El clasificador se basa únicamente en la discriminación por los valores de los parámetros morfológicos ya mencionados en la etapa de desarrollo del algoritmo de entrenamiento: inverso de la relación de aspecto, relación de área y solidez.

La estructura del algoritmo clasificador, utilizando discriminación por factores de forma es la de la ***Figura 30*** y su implementación en código es el ***Algoritmo 11*** de la sección de apéndices.

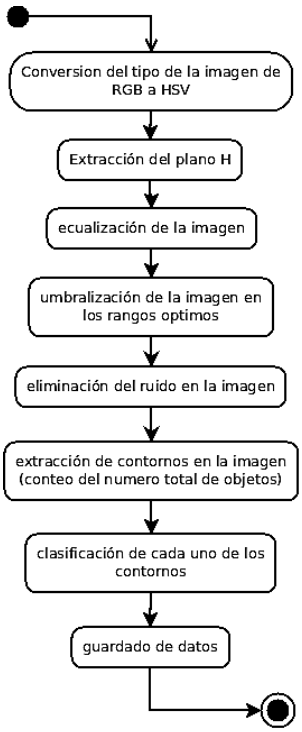


Figura 30. *Diagrama de flujo del funcionamiento del clasificador basado en discriminación por parámetros morfológicos. Elaboración propia.*

Algunas aclaraciones importantes para este tipo de clasificador es que el éxito en la clasificación depende directamente en el éxito de la umbralización, ya que es el proceso mediante el cual se evidencia la diferencia entre el fondo de la imagen y una partícula, si la umbralización es errada la forma de la partícula se puede ver comprometida.

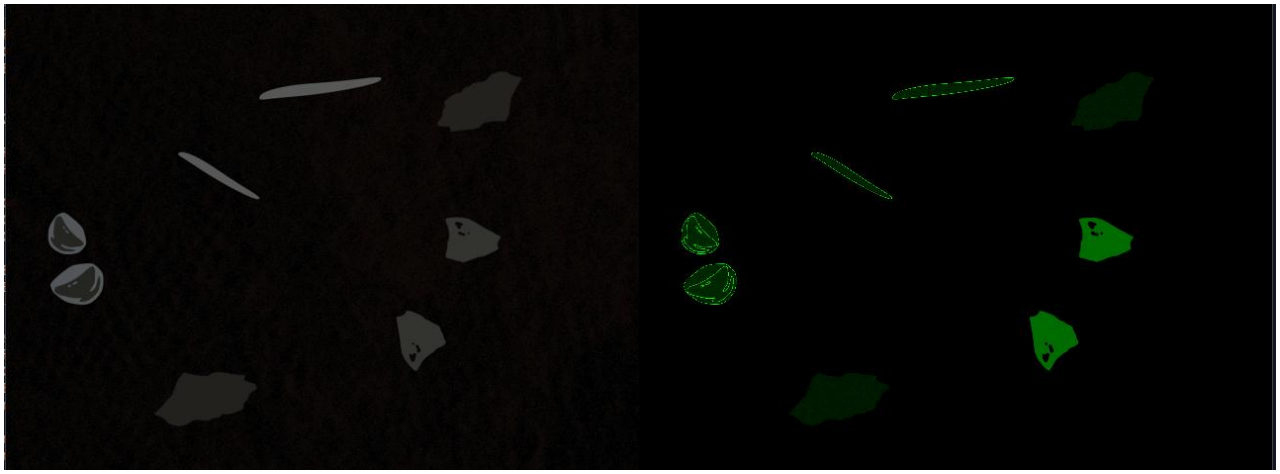


Figura 31. *Imagen simulada de un grupo de partículas (izquierda) con iluminación perpendicular y su correspondiente umbralización (derecha) resaltada en un color verde. Elaboración propia.*

Como se puede observar en la imagen real con iluminación direccional desde abajo (***Figura 32***), la umbralización no representa las formas correctas de los objetos en la imagen, a diferencia de la ***Figura 31*** en la cual las formas de las partículas están bien definidas, esto es debido a que el valor de los pixeles en la partícula es exactamente el mismo que el del fondo, lo cual ocasiona que en el momento de la eliminación del fondo (a negro) se elimine también las partes de las partículas que no tienen un borde bien definido.

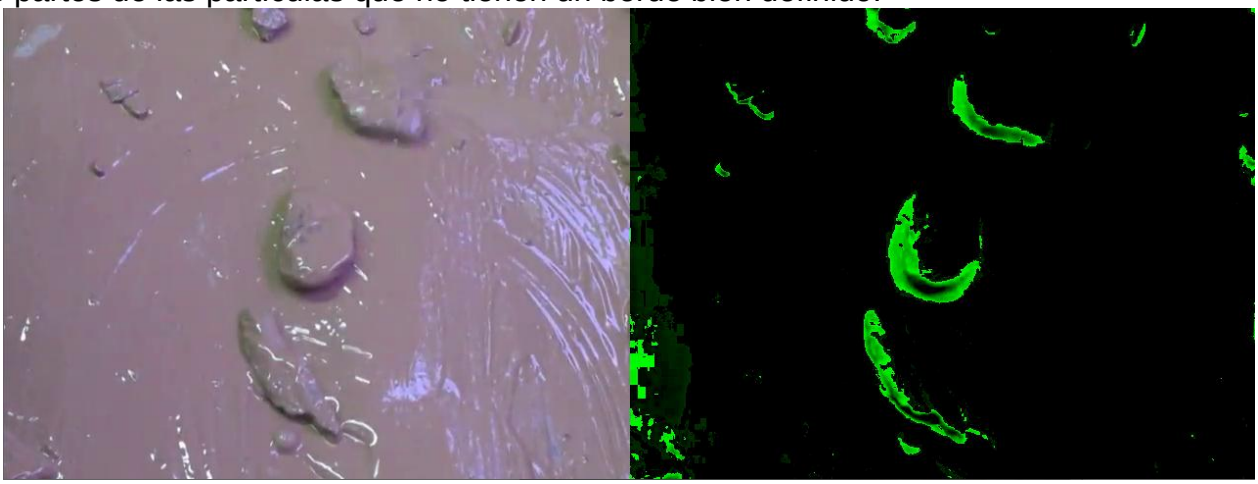


Figura 32. *Imagen real de un grupo de partículas (izquierda) con iluminación direccional desde abajo y su correspondiente umbralización (derecha) resaltada en un color verde. Elaboración propia.*

En estas situaciones el algoritmo clasificador es incapaz de definir que las partículas están mal seleccionadas y por tanto analizara las partículas (en verde) tal cual se le presenten, generando una clasificación errada. Como solución a este problema se deben garantizar siempre luces perpendiculares o utilizar el método en combinación con un método de aprendizaje de máquina, los cuales al igual que nuestro cerebro pueden completar la forma de una partícula recortada visualmente basándose en experiencias previas de cómo es una partícula.

2.3.2. Clasificadores basados en SVM, SGD, NN, NB y árboles de decisión

A diferencia de los clasificadores basados en discriminación por valores morfológicos, los clasificadores que se basan en auto aprendizaje: máquinas de soporte vectorial (SVM), SGD (*Stochastic Gradient Descend*), número de vecinos cercanos (NN), función Naive Bayes (NB) y árboles de decisión; solo requieren una etapa de entrenamiento con las imágenes de cada clase, de esta forma el mismo algoritmo elige cuales son las características que le servirán para discriminar un objeto. Una desventaja de este tipo de clasificadores es que trabaja solo con un elemento a la vez, es decir es incapaz de reconocer que hay múltiples elementos en una imagen, por tanto requiere un algoritmo preliminar que ubique las posiciones de cada objeto y luego le envié la porción de imagen en donde se encuentra cada elemento al clasificador.

La estructura de este tipo de clasificador en su etapa de entrenamiento se puede observar en la ***Figura 33***, el pre-procesamiento que se le hace a las imágenes es una conversión del esquema de color, el cual por defecto es RGB, se convierte al esquema de color HSV, el cual tiene mejores cualidades para realizar una umbralización más selectiva; otro cambio que reciben las imágenes es un análisis HOG con lo cual se obtiene un vector de caracteristicas más pequeño listo para el entrenamiento del sistema o para el clasificador final, ver los ***Algoritmos 12, 13, 14, 15, 16*** de la sección de apéndices correspondientes a los clasificadores SVM, SGD, vecinos cercanos, naive bayes y árboles de decisión.

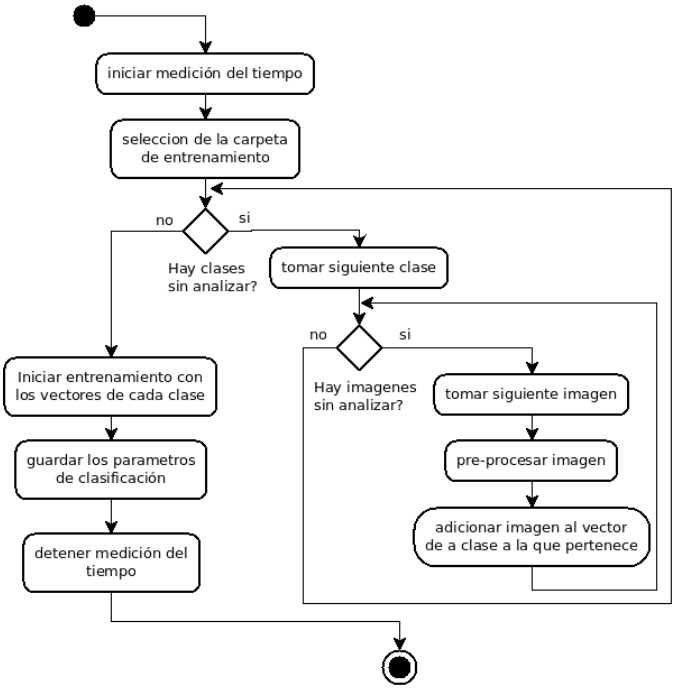


Figura 33. *Diagrama de flujo de la estructura de la fase de entrenamiento de los clasificadores basados en aprendizaje automático. Elaboración propia.*

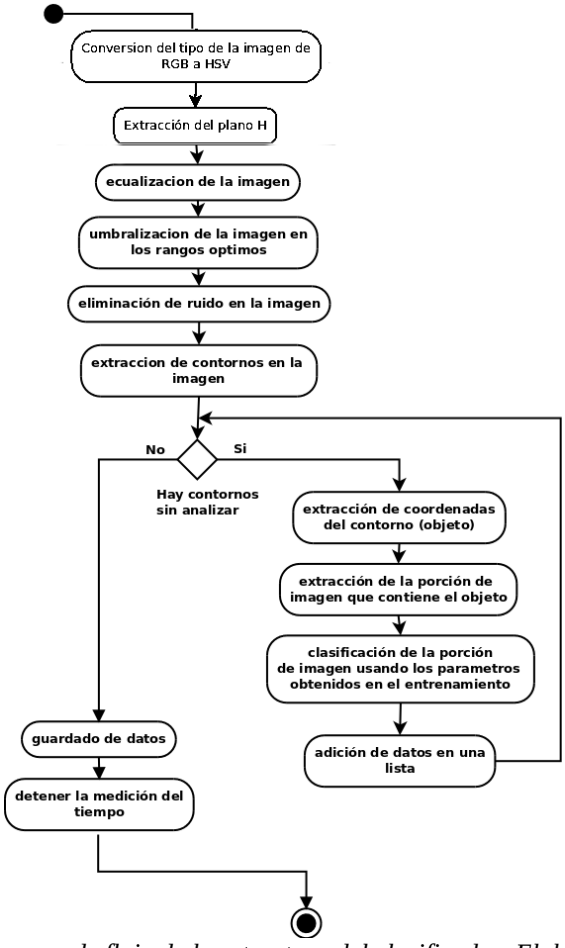


Figura 34. *Diagrama de flujo de la estructura del clasificador. Elaboración propia.*

2.3.3. Clasificador basado en red neuronal

El clasificador basado en una red neuronal, como ya se ha descrito permite la detección y clasificación de múltiples objetos en una imagen, por tanto presenta condiciones bastante favorables en comparación a los otros clasificadores analizados, el único inconveniente que presenta este método es que su estructura es del tipo *black-box* en la cual es imposible saber cuáles fueron las características que el algoritmo eligió para la clasificación, ni tampoco los valores de esos parámetros; por tanto la fase de entrenamiento es una etapa vital que garantizara el éxito o el fracaso del clasificador final.

Algunos de los problemas que se pueden presentar en la fase de entrenamiento, son básicamente: sobre-muestreo, mala elección de las muestras, sobre-entrenamiento y baja cantidad de muestras.

- Sobre-muestreo: se alimenta o se provee el sistema con demasiadas muestras y el sistema expande demasiado los rangos de cada clase de objeto, lo cual da lugar a translapamientos entre las clases (se generan dos o más posibles clasificaciones para un mismo objeto); es decir el sistema sea torna poco selectivo.
- Pocas muestras: en esta situación el sistema no alcanza a establecer rangos bien definidos y se genera un clasificador muy disperso, que puede dar lugar a los mismos problemas que el sobre-muestreo: translapamientos y poca selectividad; dada la natu-

raleza de estos problemas la solución, es hallar de forma experimental el número de muestras que genere el mejor comportamiento del clasificador.

- Mala elección de muestras: se da cuando se eligen imágenes de objetos con características parecidas (entre clases), o en el caso extremo que es cuando se alimenta el sistema con imágenes que no contienen toda la variedad de posibilidades de cada clase, lo que da lugar a sistemas con translapamientos o sistemas que no detectan la totalidad de las partículas en la imagen; la solución a este problema es bastante subjetivo ya que no se puede saber exactamente cual factor influyo en el error del entrenamiento (por la naturaleza de *black-box* de las redes neuronales), por tanto es recomendable elegir las muestras de una forma muy selectiva tratando de evitar las dos circunstancias anteriormente mencionadas.

- Sobre-entrenamiento: las redes neuronales tienen ciertos valores para configurar la calidad de entrenamiento: profundidad, mínimo rango de error, máximo rango de falsa alarma, entre otros; con los cuales se le indica a la red cuando debe considerar que el entrenamiento está completo. Cuando se le indica a la red valores muy estrictos, la red se enfoca demasiado en el grupo de muestras, es decir se vuelve demasiado estricta a que los valores deben ser iguales a los de las muestras y se vuelve incapaz de clasificar nuevos objetos que no compartan esas características, en pocas palabras se vuelve insensible a nuevos objetos; la clave para resolver este problema es la correcta determinación de estos parámetros o la determinación de los mismos a base de experimentación.

El algoritmo de clasificación basado en redes neuronales implementa funciones de la librería OpenCV, primero se pre procesa la imagen para suavizar los objetos en la imagen y después se utilizan las funciones optimizadas de OpenCV (ver ***AIgoritmo 17***).

2.4. Desarrollo de los algoritmos de comparación

Los algoritmos de comparación son los algoritmos encargados de comparar las mediciones de las métricas obtenidas por cada uno de los clasificadores en sus etapas de entrenamiento y prueba; esta comparación se dio en dos formas: una comparación numérica y una comparación gráfica; la comparación numérica compara las predicciones hechas por los clasificadores almacenadas en un archivo de texto con los valores reales de las imágenes de prueba que han sido realizados de forma manual y estima su exactitud (ver ***AIgoritmo 18***); la comparación grafica utiliza los resultados de la primera comparación, adiciona los valores de tiempos de procesos (entrenamiento y clasificación) y uso de recursos computacionales y los grafica en un diagrama de 5 puntas que permite inferir un comportamiento general de todo el clasificador (ver ***AIgoritmo 19***).

2.5. Implementación del estudio comparativo

Finalizado el desarrollo de todos los algoritmos el paso final del estudio consistió en la repetición de las etapas de entrenamiento y prueba, midiendo sus métricas de comparación. Las repeticiones se hicieron 4 veces, numero en el cual se podía garantizar una tendencia en el comportamiento de todas las métricas. El procedimiento en cada repetición que se siguió fue:

- Se desordena todas las imágenes (entrenamiento, prueba con objetos bien espaciados y de prueba con acumulaciones de partícula) cambiando el nombre de estas en sus respectivas carpetas.

- Se procede a generar los modelos de entrenamiento para cada uno de los clasificadores, midiendo los tiempos de entrenamiento y el uso de recursos computacionales.

- Se analizan las imágenes de prueba con objetos bien espaciados con cada uno de los clasificadores, midiendo los tiempos empleados en la clasificación de cada una de las partículas y guardando los resultados de la clasificación en un archivo temporal.

- Se analizan las imágenes de prueba con acumulaciones de partículas con cada uno de los clasificadores, midiendo los tiempos empleados en la clasificación de cada una de las partículas y guardando los resultados de la clasificación en un archivo temporal.

- Se comparan las clasificaciones hechas por cada uno de los clasificadores con los valores reales y se calcula la sensibilidad en cada una de las clases usando la matriz de confusión para elementos bien espaciados (***tabla 20*** de la sección de apéndices) y también con la matriz de confusión para imágenes con acumulaciones de objetos (***tabla 21*** de la sección de apéndices), con el ***Algoritmo 18*** y se guardan los resultados en las tablas ***22 y 23*** de las métricas de comparación.

Cuando se terminan las repeticiones se crean los diagramas de cuatro puntas con el ***Algoritmo 19*** para su posterior análisis.

3. Resultados

A continuación se ilustra los resultados obtenidos en la investigación; los resultados se dividieron en 2 partes: para imágenes bien espaciadas ***Figura 26.a*** y para imágenes con agrupaciones de partículas ***Figura 26.b***.

Los clasificadores elegidos fueron los que se enuncian en la parte de abajo, para una mayor uniformidad, en este capítulo se nombra cada clasificador con una letra.

Clasificadores:

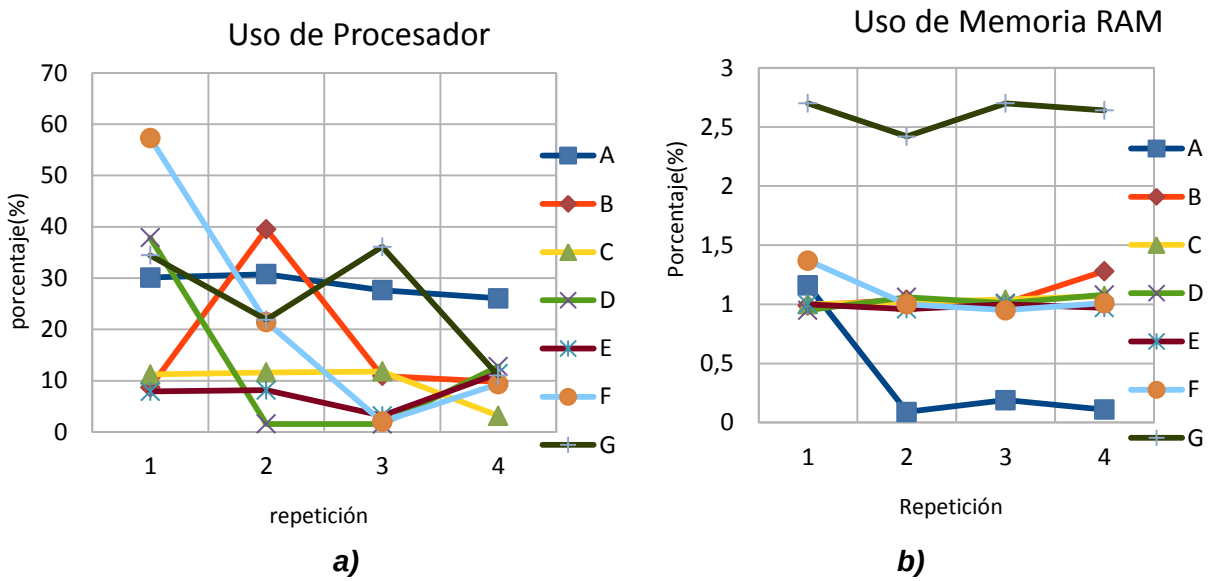
(A) clasificador basado en análisis de factores de forma
(B) clasificador basado en máquinas de soporte vectorial
(C) clasificador basados en SGD
(D) clasificador basado en un análisis de vecinos cercanos (NN)
(E) clasificador basado en Naive Bayes
(F) clasificador utilizando un árbol de decisión
(G) clasificador utilizando una red neuronal

La importancia del análisis del consumo de recursos computacionales de un nuevo *software* antes de implementarlo de manera productiva, radica en que permite elegir las cualidades del *hardware* que lo va a soportar; de esta forma se garantiza una correcta funcionalidad con un *hardware* que no se exceda en precio ni recursos computacionales. En cuanto al análisis enfocado a las características del *software,* esta brinda la posibilidad de comparar cuantitativa y cualitativamente las diversas posibilidades y opciones, de esta manera se puede escoger el que mejor se adapte a la situación que se vaya a implementar, garantizando criterios de acierto, tiempos y tiempos de operación [32].

3.1. Fase de entrenamiento

La fase de entrenamiento es la etapa más importante en un proceso de clasificación, en visión artificial en esta fase se definen los rangos de los parámetros que definirán si la imagen de una partícula pertenece a una clase u otra; por ello se debe garantizar que en esta etapa no se sobre esfuerce el equipo sino que mantenga un comportamiento estable, esto garantiza unos resultados estables en los modelos de aprendizaje generados. Las ***Figura 36.a*** y ***Figura 36.b*** correspondientes a la ***tabla 2*** y ***tabla 3*** de los apéndices, representan el consumo de recursos computacionales del equipo en el que se ejecutaron todos los procesos del estudio.

El uso del procesador indica dos parámetros del proceso realizado (entrenamiento): la carga computacional que se procesó y la estabilidad del proceso. La gráfica evidencia 3 clasificadores estables A, C y E, de los cuales C y E presentan un uso bajo y casi similar de procesador.



a) ***b)***

Figura 36. *Uso del a) procesador y b) memoria RAM del computador indicado en la metodología de desarrollo. Elaboración propia.*

Estos resultados se deben a que el clasificador A es el clasificador basado en discriminación por factores de forma y entre los demás clasificadores tiene un entrenamiento especial, ya que este no genera un archivo de entrenamiento, este genera unos cuantos rangos numéricos para las características de cada clase, de esta forma cuando las características de una partícula entran en un rango queda definida su clasificación; lo cual se ve representado en la estabilidad ya que a diferencia de los métodos de auto-entrenamiento este siempre realiza las mismas operaciones.

Por otro lado, los clasificadores C Y E: SGD y Naive Bayes son métodos pertenecientes al área de aprendizaje automático pero que utilizan procesamiento estadístico de métodos lineales, por tanto su bajo uso del procesador indica que logran un entrenamiento en casi el mismo tiempo (ver ***tabla 1 Tiempos de entrenamiento*** de los apéndices) ejecutando una cantidad de procesos semejante.

Esto referido al porcentaje de uso del procesador, en cuanto al uso de la memoria RAM, en la etapa de entrenamiento es un parámetro bastante estable, ya que los únicos valores que se almacenan temporalmente son las imágenes de entrenamiento, además los algoritmos no generan activamente nuevas variables; el uso de la memoria RAM en todos los clasificadores es relativamente bajo, esta cualidad permite observar otro parámetro, y es la compresión que se le hace a las imágenes antes de procesarlas. El clasificador A es el que menos consume memoria RAM debido a su estructura especial de entrenamiento (no almacena todas las imágenes para entrenarse, almacena unos cuantos valores numéricos y en base a ellos se realiza la clasificación), los clasificadores B, C, D, E, F son algoritmos que utilizan la misma compresión, lo cual se ve reflejado en la ***Figura 36.b*** y además comparten la característica de no poder procesar imágenes con múltiples partículas (requieren un algoritmo extra de seccionado); en contraste a estos el clasificador G, basado en redes neuronales no requiere algoritmos extras para identificar múltiples partículas en una imagen, aunque su método de compresión es dos veces inferior al de los otros métodos basados en aprendizaje automático y además su arquitectura de aprendizaje obliga a tiempos mayores de entrenamiento (ver ***tabla 1 Tiempos de entrenamiento*** de los apéndices).

3.2. Clasificación en imágenes con múltiples objetos bien separados

Las imágenes con múltiples objetos bien espaciados presentan buenas características que permiten una mejor detección de las partículas en estas: como la correcta definición de los bordes de las partículas; lo cual mejora las probabilidades para una correcta clasificación (ver ***Figura 37*** y ***Figura 38***).

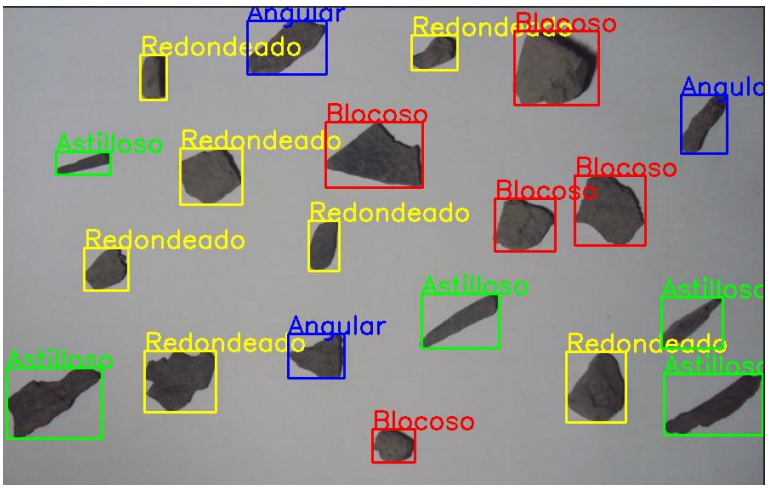


Figura 37. *Imagen resultado de una clasificación usando máquinas de soporte vectorial en imágenes con objetos bien espaciados. Elaboración propia.*

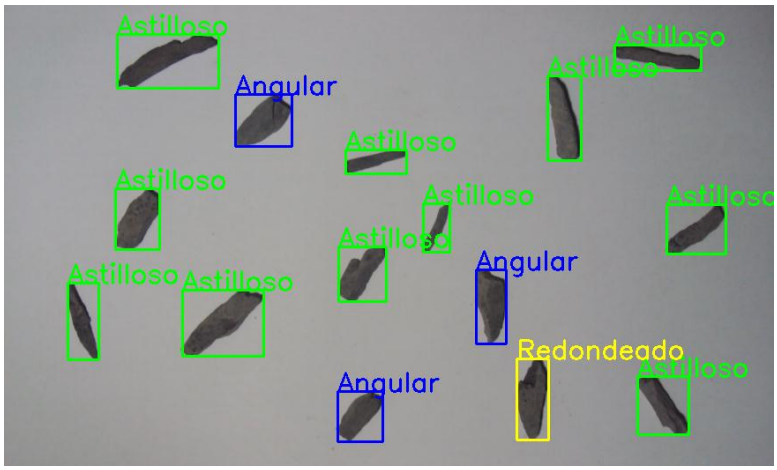
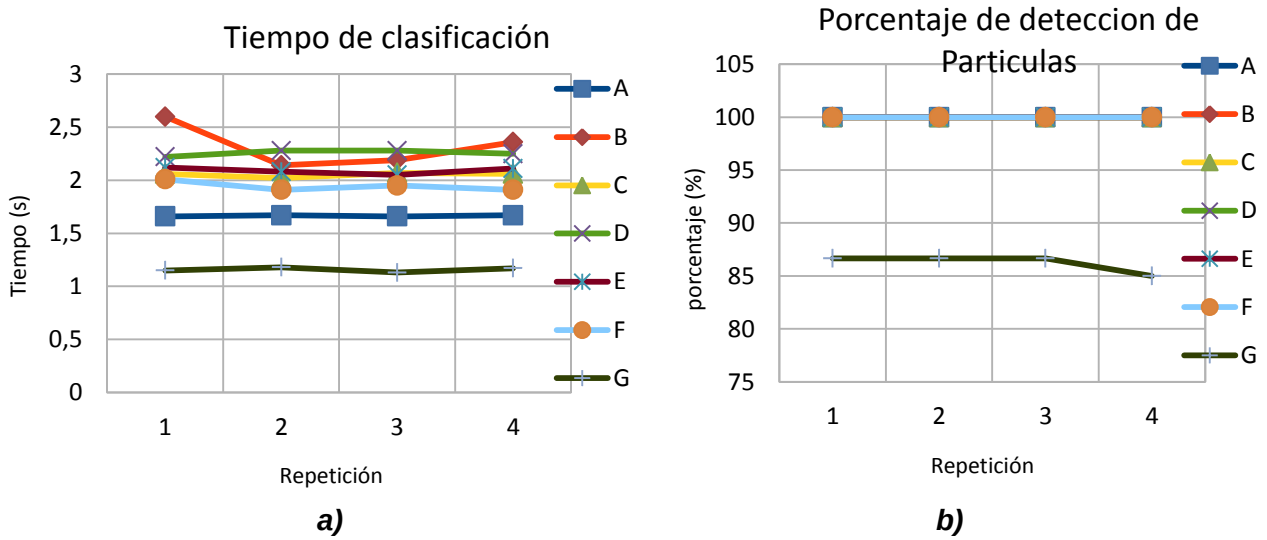


Figura 38. *Imagen resultado de una clasificación usando el método de vecinos cercanos en imágenes con objetos bien espaciados, todas las partículas son de clase astilloso (77,4% Acierto). Elaboración propia.*

Los aspectos más importantes en el estudio de este tipo de imágenes, fueron: el tiempo promedio de clasificación, el porcentaje de detección de partículas y la sensibilidad para cada una de las clases; en cuanto al uso de recursos computacionales, los resultados fueron similares a los obtenidos en la fase de entrenamiento (***Figura 36***), desfasando en que los únicos clasificadores estables en el caso de la clasificación fueron el clasificador C y E, y además que el uso de la memoria RAM fue similar en todos los clasificadores, alrededor de 1.1% (ver ***tabla 10*** **Porcentaje de uso del procesador en la etapa de clasificación para imágenes con objetos bien espaciados** y ***tabla 11 Uso de memoria Ram en la etapa de clasificación de imágenes con objetos bien espaciados*** de la sección de apéndices).

El tiempo promedio de clasificación es un parámetro que indica cuanto tiempo le toma al algoritmo determinar la clase de una partícula, es un valor muy importante ya que limita el número de imágenes que se pueden procesar en un determinado tiempo (ver ***Figura 39.a*** referente a la ***tabla 4*** de los apéndices).

La ***Figura 39.a*** muestra que los clasificadores G y A: redes neuronales y discriminación por factor de forma, superan en velocidad a los demás clasificadores, lo cual los hace prometedores para situaciones en donde se necesite procesar constantemente imágenes en un lapso de tiempo pequeño (análisis de imágenes de una línea de producción continua).



a) ***b)***

Figura 39. *a) tiempo de clasificación, b) porcentaje de detección de imágenes con partículas bien espaciadas, en donde los clasificadores: A, B, C, D, E y F lograron una detección perfecta. Elaboración propia.*

Estos resultados de los tiempos de clasificación sirven como un análisis complementario para otro factor de comparación importante en la evaluación de un clasificador: el porcentaje de acierto en la detección de partículas; la ***Figura 39.b*** referente a la ***tabla 5*** de los apéndices, muestra una detección perfecta para todos los clasificadores exceptuando al clasificador G basado en redes neuronales, este comportamiento es debido a que todos los clasificadores exceptuando al basado en redes neuronales cuenta con un algoritmo extra de seccionado, el cual con una buena definición de los bordes de las partículas (generado por el tipo de imagen) genera muy buenos resultados.

Sin embargo el porcentaje de detección por sí solo no es un factor determinante que evidencie la exactitud de los clasificadores, ya que este parámetro solo se enfoca en la correcta detección de las partículas en la imagen, para poder visualizar la exactitud de los clasificadores es necesario comparar la sensibilidad a cada una de las clases de cada clasificador, ya que la clasificación de algunos tipos de roca presentan una mayor compatibilidad con algunos algoritmos que con otros (ver ***Figura 40*** referente a las ***tablas: 6, 7, 8, 9 de sensibilidad a cada una de las clases de rocas*** de los apéndices).

Las gráficas de la sensibilidad en la clasificación permiten hacer comparaciones entre los clasificadores de 3 formas: comparando la estabilidad en las 4 repeticiones, comparando la estabilidad en las 4 clases y obviamente comparando el nivel de la sensibilidad. En cuanto a la estabilidad en las 4 repeticiones, tiene sentido en el hecho que siempre se procesaron el mismo grupo de imágenes, por tanto los valores en las 4 repeticiones debería ser el mismo

valor o estar en un rango cercano, como en el caso de los clasificadores A y B, en el caso de los clasificadores que no presentan esta estabilidad indican que sus clasificaciones están muy ligadas a la estabilidad del entrenamiento, en cuyo caso de ser implementados requerirán la búsqueda de los parámetros que garantizan el mejor entrenamiento y por tanto su mayor rendimiento en la clasificación.

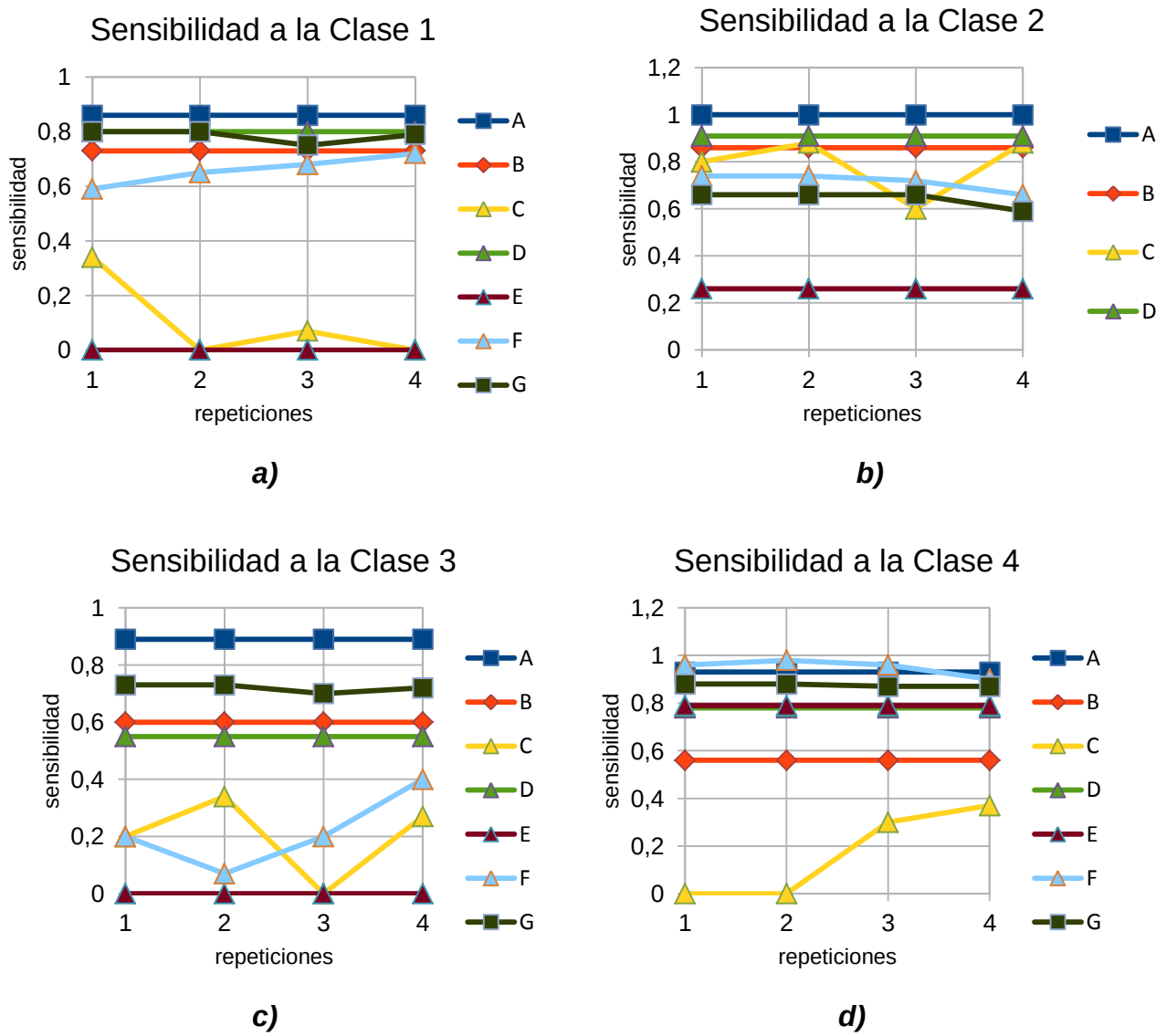


Figura 40. *a) Sensibilidad a partículas de la clase 1 (Angular), b) Sensibilidad a partículas de la clase 2 (Astilloso), c) Sensibilidad a partículas de la clase 3 (Blocoso), d) Sensibilidad a partículas de la clase 4 (Redondeado) en imágenes con partículas bien espaciadas. Elaboración propia.*

En el caso de la comparación de la estabilidad en la sensibilidad para las 4 clases y la comparación por nivel de sensibilidad, la gráfica permite observar 3 clasificadores estables y con buen índice de sensibilidad en las 4 clases de rocas, el clasificador A (0,9 de 200 posibles partículas), B (0,6 de 200 posibles partículas) y G (0,6 de 200 posibles partículas): clasificador basado en factor de forma, clasificador basado en máquinas de soporte vectorial y el clasificador basado en redes neuronales; esta estabilidad en los porcentajes de las 4 clases indica que estos algoritmos de clasificación no tienen preferencias por alguna clase en particular y por tanto los resultados de sus clasificaciones no se ven afectados por errores aleatorios presentes en el proceso, además de estos parámetros se resaltan la precisión de sus clasifi-

caciones con: 92% para el clasificador A, 62% para el clasificador B, y 75% para el clasificador G.

3.3. Clasificación en imágenes con múltiples objetos en acumulaciones de partículas

Las imágenes con acumulaciones de partículas, presentan características que hacen difícil la correcta detección y clasificación de las partículas en estas imágenes, la mayoría de problemas se deben a que en las imágenes las partículas se traslapan unas con otras, ocasionando que la forma de las partículas en la imagen se altere, aparezcan siluetas que puedan ser confundidas como partículas o una partícula pueda ser tomada como dos (ver ***Figura 41***).

Figura 41. *Errores en las imágenes de acumulaciones: mala definición de bordes (verde) y translapamientos (rojo). Elaboración propia.*

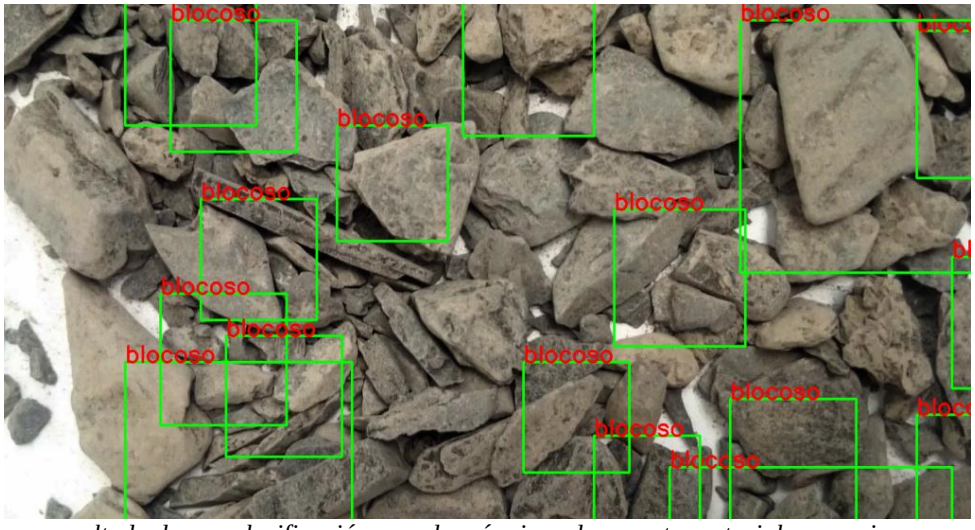
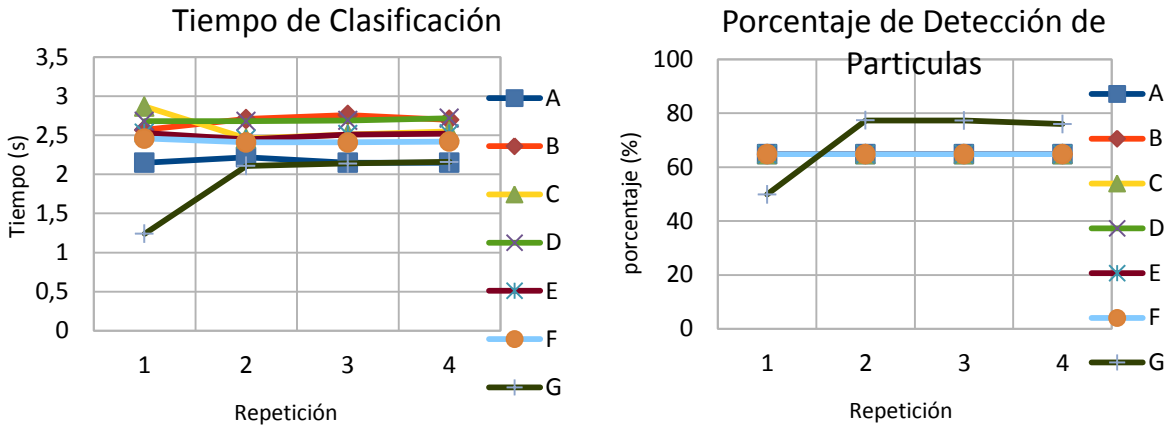


Figura 42. *Imagen resultado de una clasificación usando máquinas de soporte vectorial en una imagen con acumulación de partículas para detectar partículas de clase blocoso (%0,36 Error). Elaboración propia.*

Además de la dificultad en la clasificación y detección de partículas en este tipo de imágenes, otro factor que se ve afectado es el tiempo de clasificación (ver ***Figura 43.a*** referente a la ***tabla***

12), debido a que el sistema tiene más partículas que analizar y descartar en el caso de las falsas detecciones.

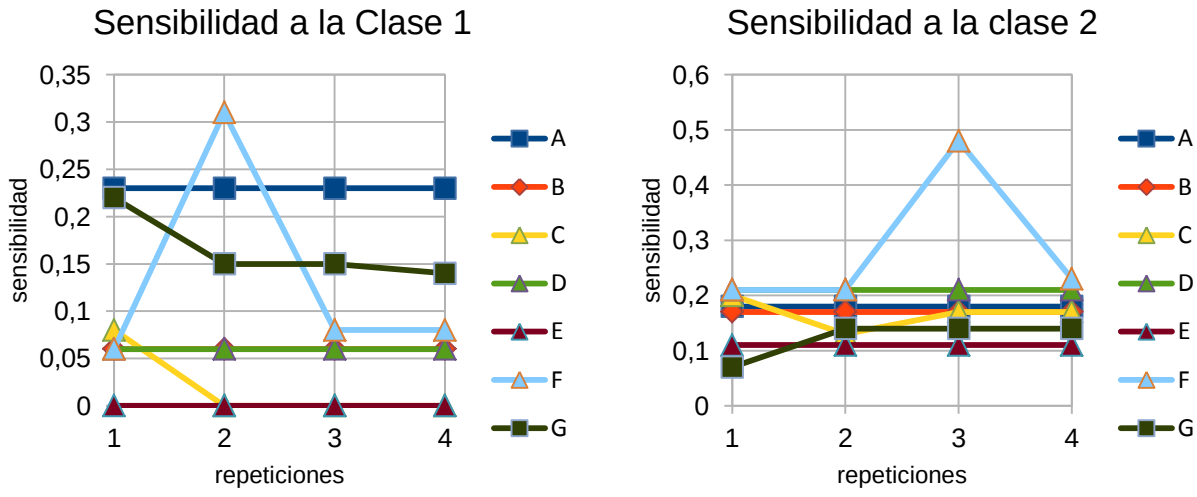


a) ***b)***

Figura 43. *a) tiempo de clasificación, b) porcentaje de detección de imágenes con partículas en acumulaciones en donde los clasificadores: A, B, C, D, E y F lograron el mismo porcentaje (64,79%). Elaboración propia.*

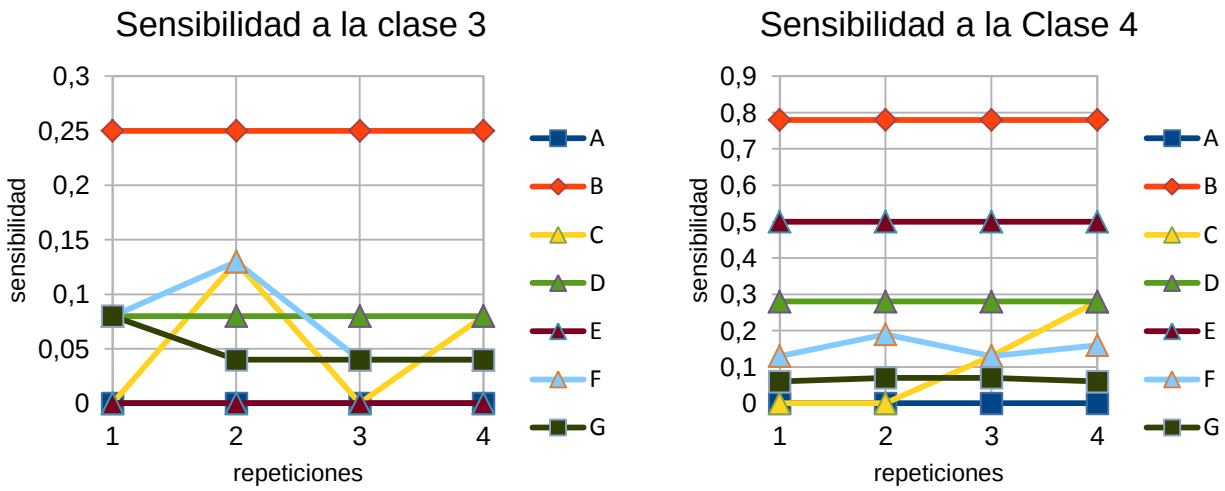
En comparación con el tiempo de clasificación de las imágenes con partículas bien espaciadas, el tiempo de clasificación en acumulaciones presenta un aumento de unas cuantas fracciones de segundo, el cual aunque en procesamiento de imágenes no representaría problemas si sería muy notable en el caso de procesamiento de video (situación de uso para la empresa JPT C&S).

En cuanto a la ***Figura 43.b*** (referente a la ***tabla 12***) del porcentaje de detección, como era esperado hubo una caída considerable frente a este tipo de imágenes; además la situación de jerarquía en detección (100%) de los algoritmos de clasificación que utilizaban un algoritmo extra de seccionado, frente a las redes neuronales se invirtió, representado en una diferencia de casi 20%.



a) ***b)***

Figura 44. *a) Sensibilidad a partículas de la clase 1 (Angular), b) Sensibilidad a partículas de la clase 2 (Astilloso), en imágenes con acumulaciones de partículas. Elaboración propia.*



a) ***b)***

Figura 45. *a) Sensibilidad a partículas de la clase 3 (Blocoso), d) Sensibilidad a partículas de la clase 4 (Redondeado) en imágenes con acumulaciones de partículas. Elaboración propia.*

La disminución del porcentaje de detección también disminuye considerablemente el porcentaje de clasificación ya que solo se pueden procesar aquellos objetos que fueron detectados, sin mencionar que en estas detecciones también se encuentran falsos positivos (ver ***Figura 44*** y ***Figura 45*** referente a las ***tablas: 14, 15, 16, 17 de sensibilidad a las clases de rocas en imágenes con acumulaciones de partículas*** de la sección de apéndices); en las ***Figuras 44*** y ***45*** se puede observar claramente la disminución en la sensibilidad de las clasificaciones, con umbrales generalizados de menos del (0,3 de 160 posibles); también muestra la preferencia de los clasificadores frente a unos tipos de roca que otros, apuntando a una falta de homogeneidad generalizada debida a la naturaleza de las imágenes con acumulaciones de partículas.

3.4. Diagramas de 5 puntas de las características de los clasificadores para comparación

Los diagramas comparativos de 5 puntas se realizan con el objetivo de analizar visualmente las características de cada clasificador como un conjunto de sus propiedades; en los 5 lados de un pentágono se ubican 5 características importantes de un clasificador y se toma como limite el valor máximo de cada característica reportada entre la comparación de todos ellos. El color azul indica cuan bueno es el valor de una característica en comparación con los demás valores y el rojo implica exactamente lo contrario.

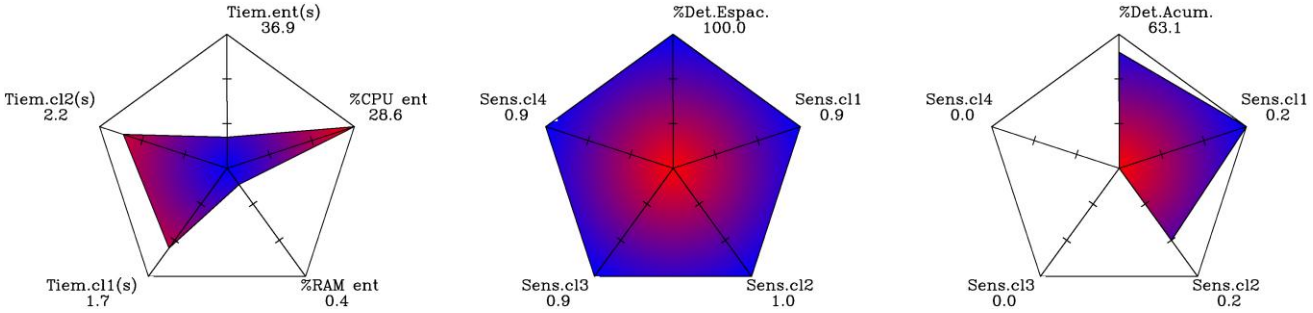
Las características de los clasificadores se distribuyeron de la siguiente manera:

- **Primer diagrama:** Dedicado a las características reportadas en la etapa de entrenamiento, Tiempo de entrenamiento, porcentaje de uso de CPU, porcentaje de uso de memoria RAM y a los tiempos promedios de clasificación para una imagen en los casos (cl1) Imagen con partículas bien espaciadas y (cl2) Imagen con acumulación de partículas

- **Segundo diagrama:** Dedicado al análisis de imágenes con objetos bien espaciados, las características analizadas fueron: el porcentajes de acierto en la detección de par-

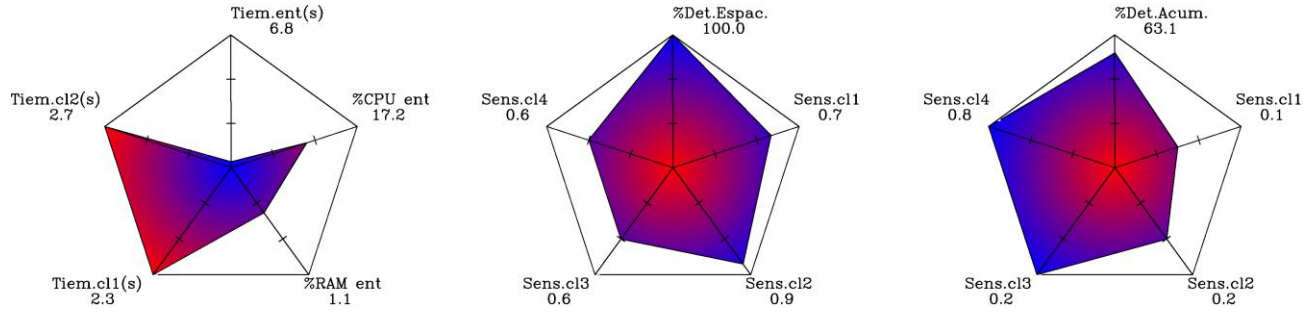
tículas y sensibilidad en la clasificación de cada una de las clases de roca; en donde: cl1: rocas clase 1 (angulares), cl2: rocas clase 2(astillosas), rocas clase 3(blocosas), rocas clase 4(redondeadas).

- **Tercer diagrama:** Dedicado al análisis de imágenes con acumulaciones de partículas, las características analizadas fueron: los porcentajes de acierto en la detección de partículas y sensibilidad en la clasificación de cada una de las clases de roca; en donde: cl1: rocas clase 1 (angulares), cl2: rocas clase 2(astillosas), rocas clase 3(blocosas), rocas clase 4(redondeadas).



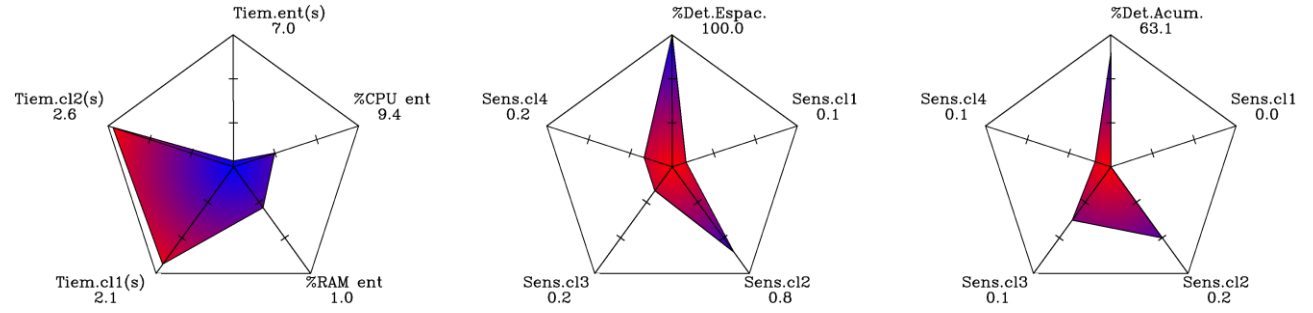
a) ***b)*** ***c)***

Figura 46. *Diagramas de comparación para el clasificador basado en análisis de factores de forma: a) etapa de entrenamiento, b) etapa de prueba con objetos bien espaciados, c) etapa de prueba con acumulaciones de partículas. Elaboración propia.*



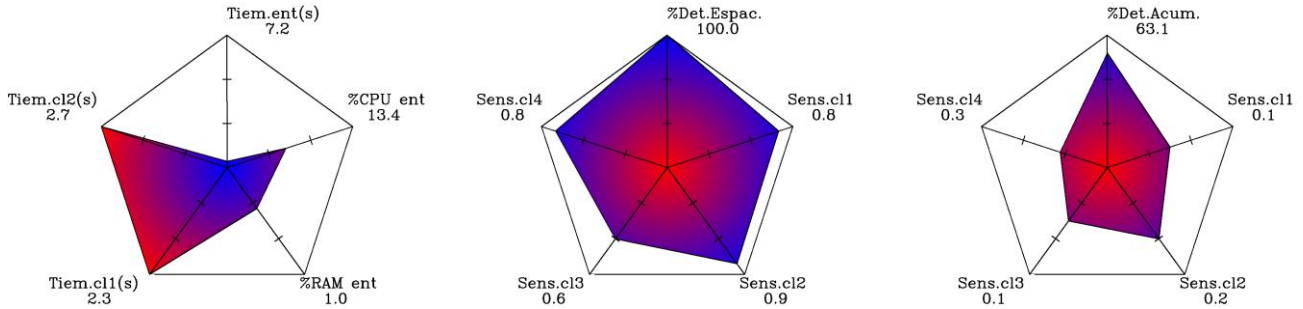
a) ***b)*** ***c)***

Figura 47. *Diagramas de comparación para el clasificador basado en máquinas de soporte vectorial: a) etapa de entrenamiento, b) etapa de prueba con objetos bien espaciados, c) etapa de prueba con acumulaciones de partículas. Elaboración propia.*



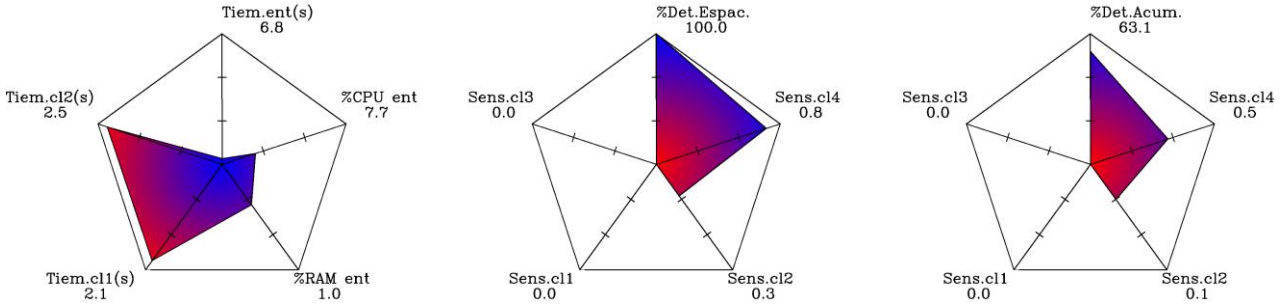
a) ***b)*** ***c)***

Figura 48. *Diagramas de comparación para el clasificador basado en gradiente estocástico descendente: a) etapa de entrenamiento, b) etapa de prueba con objetos bien espaciados, c) etapa de prueba con acumulaciones de partículas. Elaboración propia.*



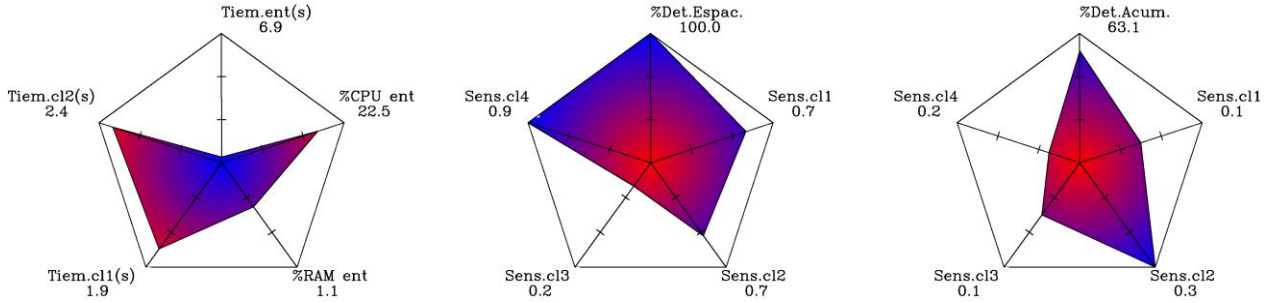
a) ***b)*** ***c)***

Figura 49. *Diagramas de comparación para el clasificador basado en análisis de vecinos cercanos: a) etapa de entrenamiento, b) etapa de prueba con objetos bien espaciados, c) etapa de prueba con acumulaciones de partículas. Elaboración propia.*



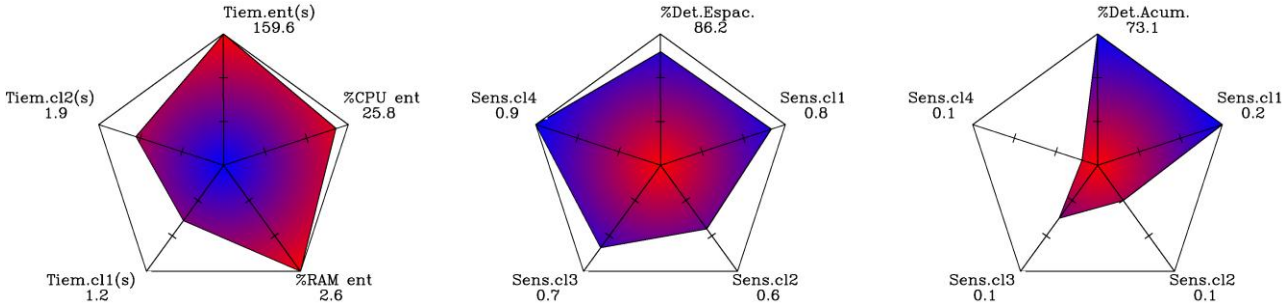
a) ***b)*** ***c)***

Figura 50. *Diagramas de comparación para el clasificador basado en naive bayes: a) etapa de entrenamiento, b) etapa de prueba con objetos bien espaciados, c) etapa de prueba con acumulaciones de partículas. Elaboración propia.*



a) ***b)*** ***c)***

Figura 50. *Diagramas de comparación para el clasificador basado en arboles de decisión: a) etapa de entrenamiento, b) etapa de prueba con objetos bien espaciados, c) etapa de prueba con acumulaciones de partículas. Elaboración propia.*



a) ***b)*** ***c)***

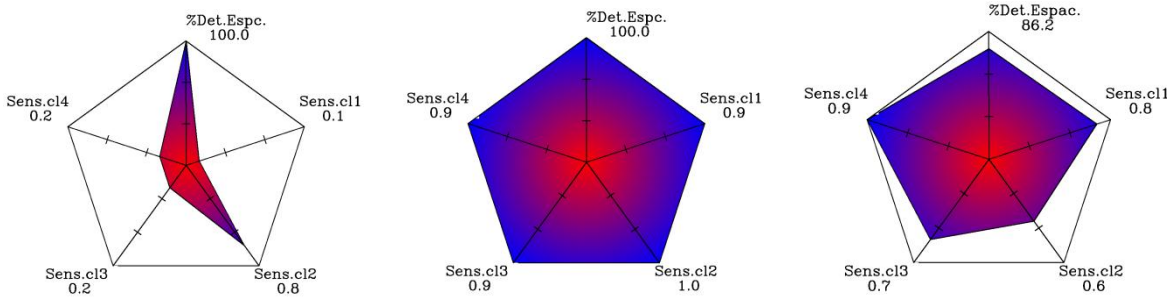
Figura 51. *Diagramas de comparación para el clasificador utilizando redes neuronales: a) etapa de entrenamiento, b) etapa de prueba con objetos bien espaciados, c) etapa de prueba con acumulaciones de partículas. Elaboración propia.*

4. Análisis de resultados

La investigación se enfocó en los 2 tipos de imágenes propuestos por la empresa JPT C&S: en imágenes con objetos bien espaciados y en acumulaciones de partículas. Las estadísticas de los valores referentes al tiempo de clasificación y uso de recursos computacionales, en todos los métodos estudiados son muy similares o en su defecto muy bajos en comparación a otras características, como para influir en la elección del mejor clasificador.

4.1. Clasificación de imágenes con partículas bien espaciadas

La clasificación de imágenes con partículas bien espaciadas, garantizan una mejor diferenciación entre partículas, una buena definición de los límites de cada partícula y por tanto una mejor apreciación de las características particulares de cada tipo de partícula. Entre los 7 clasificadores que se analizaron 6 de ellos necesitan un algoritmo extra de segmentación[1], el cual se encarga de elegir todas las porciones de imagen que pueden ser partículas y suministrarlas al clasificador como tal, el único clasificador que no utiliza este método es el clasificador basado en redes neuronales el cual de forma interna hace el trabajo de segmentación; como muestran las gráficas la detección de partículas fue de un 100% a excepción del clasificador basado en redes neuronales que logro un porcentaje de detección de 86,2%, en ese aspecto los mejores clasificadores fueron los que incluyen el algoritmo extra de segmentación; ahora bien la sensibilidad en la clasificación no fue tan equilibrado como la etapa de detección: algunos de los clasificadores se decantaron en solo algunas clases dando buenos resultados en ellas pero dando pésimos resultados en las otras clases, un ejemplo claro es el clasificador basado en gradiente estocástico descendente (ver ***Figura 51.a***).



a) ***b)*** ***c)***

Figura 51. *Diagrama comparativo de 5 puntas del a) clasificador basado en SGD y b) clasificador basado en discriminación por factor de forma en imágenes c) clasificador basado en redes neuronales, con partículas bien espaciadas (porcentaje de detección y sensibilidad en la clasificación para las 4 clases). Elaboración propia.*

De los clasificadores más acertados fueron: El clasificador basado en discriminación por factores de forma (ver ***Figura 51.b***) que dio los mejores resultados en casi todos los aspectos y el clasificador basado en redes neuronales (ver ***Figura 51.c***) que aunque no fue superior en ninguno de los aspectos tuvo valores muy equilibrados y la sensibilidad para cada clase no bajo del 0,6 (120 buenas detecciones) con precisión del 75%.

[1] Segmentación: La segmentación en el campo de la visión artificial es el proceso de dividir una imagen digital en varias partes u objetos.

4.2. Clasificación en acumulaciones de partículas

Las imágenes de acumulaciones de partículas tienen varias características que dificultan su estudio: el valor numérico de sus pixeles es muy similar, los bordes de las partículas en la imagen no están bien definidos o están ocultos por otras partículas, la iluminación puede generar efectos de desvanecimiento de bordes, entre otro; estos problemas en su gran mayoría afectan a los clasificadores que requieren el algoritmo extra de segmentación, ya que esta segmentación se basa en agrupación de pixeles de valor similar, logrando valores de detección de 64,8% en comparación con el 70,1% del clasificador basado en redes neuronales. La clasificación como tal presenta mucho menos exactitud que la etapa de detección, influido por la naturaleza del tipo de imagen que se está analizando; también, la sensibilidad para cada clase es muy inferior en comparación a los resultados obtenidos en el análisis de imágenes con objetos bien separados, este comportamiento también genera que los métodos de clasificación más equilibrados sean: el clasificador basado en SVM y el clasificador basado en arboles de decisión.

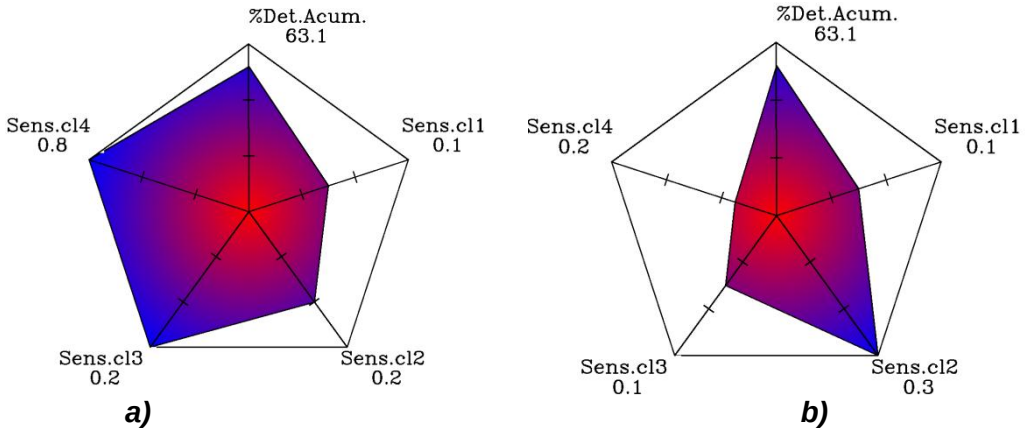


Figura 52. *Diagrama comparativo de 5 puntas del a) clasificador basado en máquinas de soporte vectorial y b) clasificador basado en arboles de decisión en imágenes con acumulaciones de objetos. Elaboración propia.*

4.3. Etapa de entrenamiento

La fase de entrenamiento se ejecuta una sola vez y es siempre la primera etapa en el desarrollo de un algoritmo clasificador, esta etapa por lo general consume más recursos computacionales y emplea más tiempo que las etapas de clasificación como tal; de los clasificadores que se estudiaron en el presente benchmarking, todos tuvieron valores de uso de recursos computacionales y tiempo de entrenamiento ligeramente similar en cuanto al rango de sus valores y a su poco nivel crítico, a excepción del clasificador basado en redes neuronales el cual consumió: una gran cantidad de tiempo en comparación con los otros métodos (2,65 minutos en comparación con los segundos requeridos por los otros métodos) y una considerable cantidad de uso del procesador (superior al 25%), aunque de no muy crítico como para influir en la elección de un clasificador.

5. Recomendaciones para la empresa JPT C&S

Usando como referencia el análisis de los clasificadores del numeral anterior, se hace prudente hacer las siguientes recomendaciones para la elección del clasificador que la empresa JPT C&S planea implementar:

- **Definición del correcto tipo de imagen que el clasificador deberá procesar:** como se demostró en el estudio comparativo, el comportamiento y respuesta de los clasificadores cambia drásticamente dependiendo el espaciamiento de los objetos en la imagen, por ello se debe definir con anterioridad las características exactas o más aproximadas de las imágenes a analizar; de esta forma se garantiza una correcta elección del clasificador con un buen rango de acierto y sensibilidad.

- **Elección del clasificador**
  Dependiendo del tipo de imagen que la empresa va a implementar se recomienda: Un clasificador basado en discriminación por factores de forma (ver ***Algoritmo 11*** de la sección de apéndices) en el caso de objetos bien espaciados con iluminación controlada y un clasificador basado en máquinas de soporte vectorial (ver ***Algoritmo 12*** de la sección de apéndices) en el caso de acumulaciones de partículas con condiciones de iluminación no controlada. En cuanto a las características de los clasificadores: el clasificador por factores de forma solo requiere el uso de los siguientes descriptores de forma: factor de forma inverso, relación de áreas y solidez de los objetos para una correcta clasificación; el clasificador basado en SVM requiere como imágenes de entrenamiento imágenes del plano H (HSV) para una mejor umbralización y una estructura basada en un *kernel* lineal sin regularización (C=1).

- **ajustes al clasificador**
  Los algoritmos clasificadores recomendados a la empresa JPT C&S para su implementación, por lo general bajo un mismo grupo de imágenes de entrenamiento retornan los mismos patrones de clasificación, es decir su clasificación solo está basada en los parámetros obtenidos en la fase de entrenamiento, por tanto si se requiere ajustar los valores de salida del clasificador el único medio posible es cambiar la elección de imágenes de entrenamiento, utilizar nuevas imágenes o en su defecto reemplazar las que están generando valores errados en la clasificación y repetir la fase de entrenamiento.

- **Aumentar la probabilidad de acierto**
  Como se mencionó en el ítem anterior mejorar los resultados de clasificación sin modificar las imágenes de entrenamiento es imposible, pero con el fin de obtener mejores resultados se puede hacer una combinación de dos o más clasificadores, es decir el bajo porcentaje de clasificación para un tipo particular de objeto, puede ser compensado si un segundo o tercer algoritmo clasificador con buenas estadísticas de clasificación para esa clase es adicionado. Como recomendación se propone utilizar una combinación de un clasificador basado en redes neuronales y el respectivo recomendado para cada tipo de imagen. Es decir para imágenes con objetos bien espaciados una combinación de: un clasificador por factor de forma como clasificador principal y un clasificador de respaldo basado en redes neuronales, para imágenes con acumulaciones de partículas se recomienda un clasificador basado en redes neuronales para

detectar la posición de las partículas y luego con un clasificador basado en SVM clasificar correctamente la partícula.

6. Conclusiones

El desarrollo del estudio comparativo de los algoritmos clasificadores de imágenes de bajo contraste, constó de varias etapas hasta su finalización: el estudio de las imágenes de bajo contraste, la elección de los métodos de clasificación, la implementación en código de cada uno de los algoritmos clasificadores y su comparación de las estadísticas de resultados; cada etapa en conjunto con los objetivos propuestos dieron lugar a las siguientes conclusiones.

- Las imágenes de bajo contraste son esencialmente imágenes cuyo valor numérico de cada pixel se encuentra entre un pequeño rango, lo cual dificulta la correcta comparación y extracción de bordes de los objetos en la imagen (acumulaciones), esta característica limita el número de algoritmos clasificadores que se pueden emplear y le da una mayor preferencia a los algoritmos clasificadores que utilizan el aprendizaje automático como base sobre los algoritmos basados en factor de forma que tienden a tener clases preferenciales.

- La preferencia de los clasificadores basados en aprendizaje automático en imágenes de bajo contraste, se debe a que estos métodos no requieren que la información de una imagen en clasificación y una imagen de entrenamiento coincidan en un 100%, por el contrario los métodos que utilizan aprendizaje automático buscan las mejores características que puedan describir a las imágenes de entrenamiento y luego en el proceso de clasificación buscan esas características en los objetos detectados para dar un veredicto de la clase con la cual se identifica más una partícula.

- El algoritmo de clasificación en la etapa de entrenamiento, con más consumo de recursos registrado (25,8%) y más tiempo de clasificación (2,65 minutos) fue el clasificador con base en redes neuronales, este alto consumo se relaciona con el tipo (*back-propagation*) de aprendizaje de la red neuronal, ya que este tiene como condicional interno el garantizar un alto nivel de acierto, obtenido de múltiples repeticiones de entrenamiento.

- El clasificador más equilibrado para imágenes con objetos bien espaciados es el basado en discriminación por factor de forma; cuando se garantiza una buena diferenciación entre los contornos de las partículas y el fondo de la imagen un clasificador que opere bajo este principio siempre obtendrá mejores estadísticas que los métodos basados en aprendizaje automático.

- La precisión de los clasificadores en imágenes con objetos bien espaciados es alta, en un rango desde el 92% (clasificador basado en factores de forma) hasta un 65% (Clasificador basado en arboles de decisión) exceptuando los clasificadores basados en gradiente estocástico descendente y naive bayes cuya precisión es menor del 37%.

- La precisión de los clasificadores en imágenes con acumulaciones de objetos es muy variable, selectiva con algunas clases y relativamente baja (78% para la clase angular del clasificador basado en factores de forma, 51% para la clase redondeado con el clasificador basado en máquinas de soporte vectorial y menores de 20% para las demás clases y clasificadores), estos resultados se deben a que la clasificación se ve in-

fluenciada por las pequeñas partículas del fondo las cuales alteran la percepción de la forma real de los objetivos de la clasificación.

- La detección y clasificación de imágenes con acumulaciones de partículas tendrá un mayor porcentaje de detección y sensibilidad si el clasificador se basa en aprendizaje de automático, entre mayor sea su autonomía más acertada será su exactitud; los clasificadores más acertados en orden jerárquico son: SVM, Arboles de decisión y redes neuronales.

- El clasificador más equilibrado en ambos tipos de presentación de los objetos es el basado en máquinas de soporte vectorial; en resumen sus mejores características son: mayor autonomía en la elección del mejor camino en la etapa de entrenamiento, mayor equilibrio en la clasificación de cada clase (no se presentan muchas preferencias) y uso moderado de recursos.

7. Referencias

[1] C. Avil, "' Detección y clasificación de objetos dentro de un salón de clases empleando técnicas de procesamiento digital de imágenes ,'" 2008.

[2] "14 FOTOS DE LA CONTAMINACIÓN EN CHINA QUE DEJAN SIN ALIENTO |." [Online]. Available: https://elmicrolector.org/2015/04/10/14-fotos-de-la-contaminacion-en-china-que-dejan-sin-aliento/. [Accessed: 16-May-2017].

[3] "About Us – JPT Consulting & Services." [Online]. Available: http://www.jpt-la.com/about-us/. [Accessed: 16-May-2017].

[4] X. Zhang and C. Science, "Application-Specific Benchmarking," 2001.

[5] M. Kumar, "Digital image processing," pp. 81–102.

[6] "Sensores con tecnología CCD vs CMOS." [Online]. Available: https://www.xatakafoto.com/camaras/sensores-con-tecnologia-ccd-vs-cmos. [Accessed: 11-May-2017].

[7] D. J. Bora, A. K. Gupta, and F. A. Khan, "Comparing the Performance of L*A*B* and HSV Color Spaces with Respect to Color Image Segmentation," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 5, no. 2, pp. 192–203, 2015.

[8] E. Archila, "Desarrollo investigativo que permita el diseño de un sistema automatizado para la caracterización geométrica (formas y tamaños) de cavings durante la perforación de pozos," 2015.

[9] J. S. Cuenca, "RECONOCIMIENTO DE OBJETOS POR DESCRIPTORES DE FORMA," 2008.

[10] G. Sánchez, "MÉTODOS Y TÉCNICAS DE RECONOCIMIENTO DE ROSTROS EN IMÁGENES DIGITALES BIDIMENSIONALES Jorge Rafael Valvert Gamboa Asesorado por el Ing. Guillermo Sánchez Barrios," 2006.

[11] "Introduction to OpenCV-Python Tutorials — OpenCV-Python Tutorials 1 documentation." [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html. [Accessed: 11-May-2017].

[12] "User Guide — skimage v0.13.0 docs." [Online]. Available: http://scikit-image.org/docs/stable/user_guide.html. [Accessed: 11-May-2017].

[13] "Introduction — SciPy v0.19.0 Reference Guide." [Online]. Available: https://docs.scipy.org/doc/scipy/reference/tutorial/general.html. [Accessed: 11-May-2017].

[14] P. Harrington, *Machine Learning in Action*. .

[15] "What is Machine Learning? | Machine Learning." [Online]. Available: http://www.mlplatform.nl/what-is-machine-learning/. [Accessed: 11-May-2017].

[16] "1.1. Generalized Linear Models — scikit-learn 0.19.dev0 documentation." [Online]. Available: http://scikit-learn.org/dev/modules/linear_model.html. [Accessed: 16-May-2017].

[17] "1.4. Support Vector Machines — scikit-learn 0.18.1 documentation." [Online]. Available: http://scikit-learn.org/stable/modules/svm.html. [Accessed: 11-May-2017].

[18] S. R. Gunn, "Support Vector Machines for Classification and Regression by," no. May, 1998.

[19] "1.5. Stochastic Gradient Descent — scikit-learn 0.18.1 documentation." [Online]. Available: http://scikit-learn.org/stable/modules/sgd.html. [Accessed: 11-May-2017].

[20] "1.6. Nearest Neighbors — scikit-learn 0.18.1 documentation." [Online]. Available: http://scikit-learn.org/stable/modules/neighbors.html. [Accessed: 11-May-2017].

[21] “1.9. Naive Bayes — scikit-learn 0.18.1 documentation.” [Online]. Available: http://scikit-learn.org/stable/modules/naive_bayes.html. [Accessed: 11-May-2017].
[22] “1.10. Decision Trees — scikit-learn 0.18.1 documentation.” [Online]. Available: http://scikit-learn.org/stable/modules/tree.html. [Accessed: 11-May-2017].
[23] J. M. Marín Diazaraque, “Introducción a las redes neuronales aplicadas,” *Man. Data Min.*, pp. 1–31, 2007.
[24] B. M. Sarwar, “Evaluation Metrics,” pp. 1–87, 2001.
[25] L. Torgo and R. Ribeiro, “Precision and recall for regression,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5808 LNAI, pp. 332–346, 2009.
[26] J. Melorose, R. Perroy, and S. Careas, “Water well drilling methods,” *Statew. Agric. L. Use Baseline 2015*, vol. 1, no. 4, 2015.
[27] D. R. Amancio *et al.*, “A systematic comparison of supervised classifiers.,” *PLoS One*, vol. 9, no. 4, p. e94137, 2014.
[28] R. Entezari-maleki, A. Rezaei, and B. Minaei-bidgoli, “Comparison of Classification Methods Based on the Type of Attributes and Sample Size.”
[29] M. Y. Kiang, “A comparative assessment of classification methods,” vol. 35, pp. 441–454, 2003.
[30] R. Caruana and A. Niculescu-Mizil, “An empirical comparison of supervised learning algorithms,” *Proc. 23rd Int. Conf. Mach. Learn.*, vol. C, no. 1, pp. 161–168, 2006.
[31] A. A. G. LÓPEZ, “RECONOCIMIENTO DE OBJETOS INMERSOS EN IMÁGENES ESTÁTICAS MEDIANTE EL ALGORITMO HOG Y RNA-MLP,” 2015.
[32] “Understanding the Purpose and Use of Benchmarking.” [Online]. Available: https://www.isixsigma.com/methodology/benchmarking/understanding-purpose-and-use-benchmarking/. [Accessed: 16-May-2017].

Apéndices

Tablas de resultados

En el presente trabajo se anexa las tablas obtenidas de la investigación del rendimiento de los métodos de clasificación en imágenes de bajo contraste, con partículas bien espaciadas y partículas en acumulaciones, para que sirvan como base en una futura elección de un clasificador, dependiendo de la situación en la que se planee implementar y las características que este deba cumplir. Cada clasificador se representa con una letra y ***rep*** hace referencia al número de la repetición del proceso.

Clasificadores:

(A) clasificador basado en análisis de factores de forma
(B) clasificador basado en máquinas de soporte vectorial
(C) clasificador basados en SGD
(D) clasificador basado en un análisis de vecinos cercanos (NN)
(E) clasificador basado en Naive Bayes
(F) clasificador utilizando un árbol de decisión
(G) clasificador utilizando una red neuronal

Tabla 1 *Tiempos de entrenamiento en la fase de entrenamiento*

Clas.	rep 1 (s)	rep 2 (s)	rep 3 (s)	rep 4 (s)
A	36,91	36,86	36,9	36,84
B	6,32	6,65	7,14	7,03
C	6,52	7,07	7,13	7,15
D	6,95	7,47	7,23	7,04
E	6,56	6,81	7,19	6,77
F	7,35	6,79	6,64	6,88
G	152,3	166,31	157,28	162,59

Tabla 2 *Uso de procesador en la etapa de entrenamiento*

Clas.	rep1 (%)	rep2 (%)	rep 3 (%)	rep 4 (%)
A	30,1	30,77	27,64	26,06
B	8,72	39,54	10,93	9,79
C	11,19	11,6	11,79	3,16
D	37,88	1,57	1,53	12,76
E	7,9	8,14	3,16	11,4
F	57,3	21,43	2,01	9,32
G	34,45	21,87	36,09	10,98

Tabla 3 *Uso de memoria Ram en la etapa de entrenamiento*

Clas.	rep 1 (%)	rep 2 (%)	rep 3 (%)	rep 4 (%)
A	1,16	0,09	0,19	0,11
B	0,99	1,04	1,01	1,28
C	1,0	1,02	1,04	1,07
D	0,95	1,06	1,01	1,08
E	1,0	0,96	1,0	0,97
F	1,37	1,0	0,95	1,01
G	2,7	2,42	2,7	2,64

Tabla 4 *Tiempo promedio de clasificación en imágenes con partículas bien espaciadas*

Clas.	rep 1 (s)	rep 2 (s)	rep 3 (s)	rep 4 (s)
A	1,66	1,67	1,66	1,67
B	2,6	2,14	2,19	2,36
C	2,06	2,02	2,07	2,06
D	2,22	2,28	2,28	2,25
E	2,12	2,08	2,05	2,11
F	2,01	1,91	1,95	1,91
G	1,15	1,18	1,13	1,17

Tabla 5 *Porcentaje acierto en la detección de partículas en imágenes con partículas bien espaciadas*

Clas.	rep 1 (%)	rep 2 (%)	rep 3 (%)	rep 4 (%)
A	100	100	100	100
B	100	100	100	100
C	100	100	100	100
D	100	100	100	100
E	100	100	100	100
F	100	100	100	100
G	86,66	86,66	86,66	85,0

Tabla 6 *Sensibilidad a partículas de la clase 1 en imágenes con partículas bien espaciadas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	0,86	0,86	0,86	0,86
B	0,73	0,73	0,73	0,73
C	0,34	0,0	0,07	0,0
D	0,8	0,8	0,8	0,8
E	0,0	0,0	0,0	0,0
F	0,59	0,65	0,68	0,72
G	0,8	0,8	0,75	0,79

Tabla 7 *Sensibilidad a partículas de la clase 2 en imágenes con partículas bien espaciadas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	1	1	1	1
B	0,86	0,86	0,86	0,86
C	0,8	0,88	0,6	0,88
D	0,91	0,91	0,91	0,91
E	0,26	0,26	0,26	0,26
F	0,74	0,74	0,72	0,66
G	0,66	0,66	0,66	0,59

Tabla 8 *Sensibilidad a partículas de la clase 3 en imágenes con partículas bien espaciadas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	0,89	0,89	0,89	0,89
B	0,6	0,6	0,6	0,6
C	0,2	0,34	0,0	0,27
D	0,55	0,55	0,55	0,55
E	0,0	0,0	0,0	0,0
F	0,2	0,07	0,2	0,4
G	0,73	0,73	0,7	0,72

Tabla 9 *Sensibilidad a partículas de la clase 4 en imágenes con partículas bien espaciadas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	0,93	0,93	0,93	0,93
B	0,56	0,56	0,56	0,56
C	0,0	0,0	0,3	0,37
D	0,78	0,78	0,78	0,78
E	0,79	0,79	0,79	0,79
F	0,96	0,98	0,96	0,9
G	0,88	0,88	0,87	0,87

Tabla 10 *Porcentaje de uso del procesador en la etapa de clasificación de imágenes con objetos bien espaciados*

Clas.	rep 1 (%)	rep 2 (%)	rep 3 (%)	rep 4 (%)
A	8,33	36,45	13,27	11,47
B	10,98	36,33	10,84	9,83
C	9,65	4,82	7,9	8,2
D	17,63	9,92	8,81	1,64
E	16,01	7,88	12,28	13,77
F	19,58	36,66	11,66	31,15
G	12,22	15,59	42,15	15,25

Tabla 11 *Uso de memoria Ram en la etapa de clasificación de imágenes con objetos bien espaciados*

Clas.	rep 1 (%)	rep 2 (%)	rep 3 (%)	rep 4 (%)
A	1,16	1,13	0,98	0,97
B	1,14	0,93	0,96	1,06
C	1,0	1,4	1,04	1,03
D	1,09	1,03	0,88	0,92
E	1,07	1,15	0,96	1,11
F	1,0	1,0	1,21	1,03
G	1,17	1,1	1,14	1,24

Tabla 12 *Tiempo promedio de clasificación en imágenes con acumulación de partículas*

Clas.	rep 1(s)	rep 2 (s)	rep 3 (s)	rep 4 (s)
A	2,15	2,22	2,15	2,15
B	2,57	2,71	2,76	2,7
C	2,87	2,46	2,51	2,55
D	2,68	2,68	2,69	2,72
E	2,53	2,45	2,51	2,52
F	2,46	2,41	2,41	2,42
G	1,24	2,11	2,14	2,16

Tabla 13 *Porcentaje acierto en la detección de partículas en imágenes con acumulación de partículas*

Clas.	rep 1 (%)	rep 2 (%)	rep 3 (%)	rep 4 (%)
A	64,79	64,79	64,79	64,79
B	64,79	64,79	64,79	64,79
C	64,79	64,79	64,79	64,79
D	64,79	64,79	64,79	64,79
E	64,79	64,79	64,79	64,79
F	64,79	64,79	64,79	64,79
G	49,85	77,37	77,31	76,01

Tabla 14 *Sensibilidad a partículas de la clase 1 en imágenes con acumulación de partículas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	0,23	0,23	0,23	0,23
B	0,06	0,06	0,06	0,06
C	0,08	0,0	0,0	0,0
D	0,06	0,06	0,06	0,06
E	0,0	0,0	0,0	0,0
F	0,06	0,31	0,08	0,08
G	0,22	0,15	0,15	0,14

Tabla 15 *Sensibilidad a partículas de la clase 2 en imágenes con acumulación de partículas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	0,18	0,18	0,18	0,18
B	0,17	0,17	0,17	0,17
C	0,2	0,13	0,17	0,17
D	0,21	0,21	0,21	0,21
E	0,11	0,11	0,11	0,11
F	0,21	0,21	0,48	0,23
G	0,07	0,14	0,14	0,14

Tabla 16 *Sensibilidad a partículas de la clase 3 en imágenes con acumulación de partículas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	0,0	0,0	0,0	0,0
B	0,25	0,25	0,25	0,25
C	0,0	0,13	0,0	0,08
D	0,08	0,08	0,08	0,08
E	0,0	0,0	0,0	0,0
F	0,08	0,13	0,04	0,04
G	0,08	0,04	0,04	0,04

Tabla 17 *Sensibilidad a partículas de la clase 4 en imágenes con acumulación de partículas*

Clas.	rep 1	rep 2	rep 3	rep 4
A	0,0	0,0	0,0	0,0
B	0,78	0,78	0,78	0,78
C	0,0	0,0	0,13	0,28
D	0,28	0,28	0,28	0,28
E	0,5	0,5	0,5	0,5
F	0,13	0,19	0,13	0,16
G	0,06	0,07	0,07	0,06

Tabla 18 *Porcentaje de uso del procesador en la etapa de clasificación en imágenes con acumulación de partículas*

Clas.	rep 1 (%)	rep 2 (%)	rep 3 (%)	rep 4 (%)
A	7,86	62,21	8,95	12,25
B	38,22	20,1	16,84	4,86
C	3,17	3,33	12,87	1,51
D	6,42	5,05	4,81	12,71
E	6,41	8,29	12,92	7,86
F	15,39	2,97	6,74	31,9
G	11,7	10,32	24,94	27,48

Tabla 19 *Uso de memoria Ram en la etapa de clasificación en imágenes con acumulación de partículas*

Clas.	rep 1 (%)	rep 2 (%)	rep 3 (%)	rep 4 (%)
A	1,39	1,53	1,2	1,43
B	1,08	1,41	1,23	1,22
C	1,37	1,22	1,44	1,4
D	1,21	1,16	1,2	1,26
E	1,35	1,19	1,22	1,32
F	1,4	1,38	1,38	1,29
G	4,14	2,13	2,42	2,38

Las siguientes tablas hacen referencia a los valores necesarios para calcular el valor de la sensibilidad (rc) a cada clase (angular, astilloso, blocoso, redondeado), para cada uno de los clasificadores (A, B, C, D, E, F, G) haciendo uso del valor del conteo de: positivos verdaderos (tp), positivos falsos (fp), verdaderos negativos (tn) y falsos negativos (fn); El total del número de objetos para cada clase en cada repetición fue de 200 en imágenes con objetos espaciados y 160 con objetos en acumulaciones de partículas, en las 4 repeticiones.

Tabla 20 Matriz de confusión para la etapa de clasificación en imágenes con objetos bien espaciados

REP.	CL.	CLASIFICADORES																											
		A				B				C				D				E				F				G			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	1	173	11	16		147		10	43	68			132	160	9	31		0	37	105	58	118	7	75		138			35
	2		200			17	173	10		40	160			19	181				52	58	90	51	149			40	115	13	5
	3	2	5	178	15	40	31	120	9	49	100	40	11		40	109	51	90	110	0		10	75	40	75		40	126	7
	4	14			186	20	16	52	112	44	33	123	0	29	15		156			43	157	7			193			20	153
2	1	173	11	16		147		10	43	0	61		139	160	9	31		0	37	105	58	130		70		138			35
	2		200			17	173	10		24	176			19	181				52	58	90	17	148	5	30	40	115	13	5
	3	2	5	178	15	40	31	120	9	116	17	67			40	109	51	90	110	0		78	78	13	31		40	126	7
	4	14			186	20	16	52	112		61	139	0	29	15		156			43	157			3	197			20	153
3	1	173	11	16		147		10	43	13	152	20	15	160	9	31		0	37	105	58	136		64		129		44	
	2		200			17	173	10		80	120			19	181				52	58	90		145		55	21	115		37
	3	2	5	178	15	40	31	120	9		64	0	136		40	109	51	90	110	0		50	72	40	38		44	121	8
	4	14			186	20	16	52	112	8		132	60	29	15		156			43	157			7	193			23	150
4	1	173	11	16		147		10	43	0		75	125	160	9	31		0	37	105	58	145			55	136	11	26	
	2		200			17	173	10		24	176			19	181				52	58	90		132	68		34	102	37	
	3	2	5	178	15	40	31	120	9	23	124	53			40	109	51	90	110	0		54	66	80			30	124	19
	4	14			186	20	16	52	112	77		49	74	29	15		156			43	157	12		8	180		15	8	150

Tabla 21 Matriz de confusión para la etapa de clasificación en imágenes con acumulaciones de objetos

REP.	CL.	CLASIFICADORES																											
		A				B				C				D				E				F				G			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	1	23		8	70	6	70	25		8	93			6	91	4		0		16	85	6	90	5		25	87		
	2	6	18	77		84	17			81	20			80	21			86	11	4		80	21			100	8	4	
	3		20	0	81			25	76			0	101		8	8	85	15	86	0				8	93		3	9	100
	4		10	91	0			22	79			101	0			73	28			51	50			88	13		5	100	7
2	1	23		8	70	6	70	25		0	101			6	91	4		0		16	85	31	70			17	95		
	2	6	18	77		84	17			88	13			80	21			86	11	4		80	21			96	16		
	3		20	0	81			25	76			13	88		8	8	85	15	86	0			10	13	78		6	5	101
	4		10	91	0			22	79			101	0			73	28			51	50			82	19		3	101	8
3	1	23		8	70	6	70	25		0	101			6	91	4		0		16	85	8	82	11		17	95		
	2	6	18	77		84	17			84	17			80	21			86	11	4		53	48			96	16		
	3		20	0	81			25	76			0	101		8	8	85	15	86	0			14	4	83		6	5	101
	4		10	91	0			22	79			88	13			73	28			51	50	5		83	13		3	101	8
4	1	23		8	70	6	70	25		0	86	15		6	91	4		0		16	85	8	4	89		16	96		
	2	6	18	77		84	17			84	17			80	21			86	11	4		78	23			96	16		
	3		20	0	81			25	76		6	8	87		8	8	85	15	86	0			9	4	88		6	5	101
	4		10	91	0			22	79			73	28			73	28			51	50		85		16	4		101	7

Tabla 22 Calculo de métricas de comparación de la matriz de confusión para la etapa de clasificación en imágenes con objetos bien espaciados

	rep.		1				2				3				4			
	clase		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
CL.	A	tp	173	200	178	186	173	200	178	186	173	200	178	186	173	200	178	186
		fp	16	16	16	15	16	16	16	15	16	16	16	15	16	16	16	15
		tn	564	537	559	551	564	537	559	551	564	537	559	551	564	537	559	551
		fn	27	0	22	14	27	0	22	14	27	0	22	14	27	0	22	14
		rc	0,86	1	0,89	0,93	0,86	1	0,89	0,93	0,86	1	0,89	0,93	0,86	1	0,89	0,93
		pr	0,92	0,93	0,92	0,92	0,92	0,93	0,92	0,92	0,92	0,93	0,92	0,92	0,92	0,93	0,92	0,92
	B	tp	147	173	120	112	147	173	120	112	147	173	120	112	147	173	120	112
		fp	77	47	72	52	77	47	72	52	77	47	72	52	77	47	72	52
		tn	405	379	432	440	405	379	432	440	405	379	432	440	405	379	432	440
		fn	53	27	80	88	53	27	80	88	53	27	80	88	53	27	80	88
		rc	0,73	0,86	0,6	0,56	0,73	0,86	0,6	0,56	0,73	0,86	0,6	0,56	0,73	0,86	0,6	0,56
		pr	0,66	0,79	0,62	0,68	0,66	0,79	0,62	0,68	0,66	0,79	0,62	0,68	0,66	0,79	0,62	0,68
	C	tp	68	160	40	0	0	176	67	0	13	120	0	60	0	176	53	74
		fp	133	133	123	143	140	139	139	139	88	216	152	151	124	124	124	125
		tn	200	108	228	268	243	67	176	243	180	73	193	133	303	127	250	229
		fn	132	40	160	200	200	24	133	200	187	80	200	140	200	24	147	126
		rc	0,34	0,8	0,2	0	0	0,88	0,34	0	0,07	0,6	0	0,3	0	0,88	0,27	0,37
		pr	0,34	0,55	0,25	0	0	0,56	0,33	0	0,13	0,36	0	0,28	0	0,59	0,3	0,37
	D	tp	160	181	109	156	160	181	109	156	160	181	109	156	160	181	109	156
		fp	48	64	31	51	48	64	31	51	48	64	31	51	48	64	31	51
		tn	446	425	497	450	446	425	497	450	446	425	497	450	446	425	497	450
		fn	40	19	91	44	40	19	91	44	40	19	91	44	40	19	91	44
		rc	0,8	0,91	0,55	0,78	0,8	0,91	0,55	0,78	0,8	0,91	0,55	0,78	0,8	0,91	0,55	0,78
		pr	0,77	0,79	0,69	0,76	0,71	0,74	0,63	0,71	0,84	0,85	0,78	0,83	0,76	0,78	0,68	0,75
	E	tp	0	52	0	157	0	52	0	157	0	52	0	157	0	52	0	157
		fp	90	147	206	148	90	147	206	148	90	147	206	148	90	147	206	148
		tn	209	157	209	52	209	157	209	52	209	157	209	52	209	157	209	52
		fn	200	148	200	43	200	148	200	43	200	148	200	43	200	148	200	43
		rc	0	0,26	0	0,79	0	0,26	0	0,79	0	0,26	0	0,79	0	0,26	0	0,79
		pr	0	0,37	0	0,64	0	0,26	0	0,52	0	0,2	0	0,43	0	0,26	0	0,51
	F	tp	118	149	40	193	130	148	13	197	136	145	40	193	145	132	80	180
		fp	68	82	75	75	95	78	78	61	50	72	71	93	66	66	76	55
		tn	382	351	460	307	358	340	475	291	378	369	474	321	392	405	457	357
		fn	82	51	160	7	70	52	187	3	64	55	160	7	55	68	120	20
		rc	0,59	0,74	0,2	0,96	0,65	0,74	0,07	0,98	0,68	0,72	0,2	0,96	0,72	0,66	0,4	0,9
		pr	0,63	0,65	0,35	0,72	0,58	0,65	0,14	0,76	0,73	0,67	0,36	0,67	0,69	0,67	0,51	0,77
	G	tp	138	115	126	153	138	115	126	153	129	115	121	150	136	102	124	150
		fp	40	40	33	47	40	40	43	47	21	44	67	45	34	56	71	19
		tn	394	417	406	379	394	417	406	379	386	400	394	365	376	410	388	362
		fn	35	58	47	20	35	58	47	20	44	58	52	23	37	71	49	23
		rc	0,8	0,66	0,73	0,88	0,8	0,66	0,73	0,88	0,75	0,66	0,7	0,87	0,79	0,59	0,72	0,87
		pr	0,78	0,74	0,79	0,77	0,78	0,74	0,75	0,77	0,86	0,72	0,64	0,77	0,8	0,65	0,64	0,89

Tabla 23 Calculo de métricas de comparación de la matriz de confusión para la etapa de clasificación en imágenes con acumulaciones de objetos

	rep.		1				2				3				4			
	clase		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
CL.	A	tp	23	18	0	0	23	18	0	0	23	18	0	0	23	18	0	0
		fp	6	30	176	151	6	30	176	151	6	30	176	151	6	30	176	151
		tn	18	23	41	41	18	23	41	41	18	23	41	41	18	23	41	41
		fn	78	83	101	101	78	83	101	101	78	83	101	101	78	83	101	101
		rc	0,23	0,18	0	0	0,23	0,18	0	0	0,23	0,18	0	0	0,23	0,18	0	0
		pr	0,79	0,38	0	0	0,79	0,38	0	0	0,79	0,38	0	0	0,79	0,38	0	0
	B	tp	6	17	25	79	6	17	25	79	6	17	25	79	6	17	25	79
		fp	84	70	47	76	84	70	47	76	84	70	47	76	84	70	47	76
		tn	121	110	102	48	121	110	102	48	121	110	102	48	121	110	102	48
		fn	95	84	76	22	95	84	76	22	95	84	76	22	95	84	76	22
		rc	0,06	0,17	0,25	0,78	0,06	0,17	0,25	0,78	0,06	0,17	0,25	0,78	0,06	0,17	0,25	0,78
		pr	0,07	0,2	0,35	0,51	0,07	0,2	0,35	0,51	0,07	0,2	0,35	0,51	0,07	0,2	0,35	0,51
	C	tp	8	20	0	0	0	13	13	0	0	17	0	13	0	17	8	28
		fp	81	93	101	101	88	101	101	88	84	101	88	101	84	92	88	87
		tn	20	8	28	28	26	13	13	26	30	13	30	17	53	36	45	25
		fn	93	81	101	101	101	88	88	101	101	84	101	88	101	84	93	73
		rc	0,08	0,2	0	0	0	0,13	0,13	0	0	0,17	0	0,13	0	0,17	0,08	0,28
		pr	0,09	0,18	0	0	0	0,11	0,11	0	0	0,14	0	0,11	0	0,16	0,08	0,24
	D	tp	6	21	8	28	6	21	8	28	6	21	8	28	6	21	8	28
		fp	80	99	77	85	80	99	77	85	80	99	77	85	80	99	77	85
		tn	57	42	55	35	57	42	55	35	57	42	55	35	57	42	55	35
		fn	95	80	93	73	95	80	93	73	95	80	93	73	95	80	93	73
		rc	0,06	0,21	0,08	0,28	0,06	0,21	0,08	0,28	0,06	0,21	0,08	0,28	0,06	0,21	0,08	0,28
		pr	0,07	0,17	0,09	0,25	0,07	0,17	0,09	0,25	0,07	0,17	0,09	0,25	0,07	0,17	0,09	0,25
	E	tp	0	11	0	50	0	11	0	50	0	11	0	50	0	11	0	50
		fp	101	86	71	85	101	86	71	85	101	86	71	85	101	86	71	85
		tn	61	50	61	11	61	50	61	11	61	50	61	11	61	50	61	11
		fn	101	90	101	51	101	90	101	51	101	90	101	51	101	90	101	51
		rc	0	0,11	0	0,5	0	0,11	0	0,5	0	0,11	0	0,5	0	0,11	0	0,5
		pr	0	0,11	0	0,37	0	0,11	0	0,37	0	0,11	0	0,37	0	0,11	0	0,37
	F	tp	6	21	8	13	31	21	13	19	8	48	4	13	8	23	4	16
		fp	80	90	93	93	80	80	82	78	58	96	94	83	78	98	89	88
		tn	42	27	40	35	53	63	71	65	65	25	69	60	43	28	47	35
		fn	95	80	93	88	70	80	88	82	93	53	97	88	93	78	97	85
		rc	0,06	0,21	0,08	0,13	0,31	0,21	0,13	0,19	0,08	0,48	0,04	0,13	0,08	0,23	0,04	0,16
		pr	0,07	0,19	0,08	0,12	0,28	0,21	0,14	0,2	0,12	0,33	0,04	0,14	0,09	0,19	0,04	0,15
	G	tp	25	8	9	7	17	16	5	8	17	16	5	8	16	16	5	7
		fp	100	95	104	100	96	104	101	101	96	104	101	101	100	102	101	101
		tn	24	41	40	42	29	30	41	38	29	30	41	38	28	28	39	37
		fn	87	104	103	105	95	96	107	104	95	96	107	104	96	96	107	105
		rc	0,22	0,07	0,08	0,06	0,15	0,14	0,04	0,07	0,15	0,14	0,04	0,07	0,14	0,14	0,04	0,06
		pr	0,2	0,08	0,08	0,07	0,15	0,13	0,05	0,07	0,15	0,13	0,05	0,07	0,14	0,14	0,05	0,06

Algoritmos

A continuación se anexan los algoritmos usados en el estudio comparativo de los clasificadores digitales, los clasificadores y los algoritmos extra que se usaron en el presente trabajo. Los algoritmos están escritos en inglés utilizando el estilo REV8 de python3 para mantener los estándares globales de programación.

Algoritmo 1: *Generación de muestras de entrenamiento a partir de imágenes con fondo blanco y objetos bien espaciados.*

```
#---------------- Start Algorithm 1 ------------------
"""With a folder path, generate square images using bigger images with a defined
number of particles"""

import cv2
import numpy as np
from statistics import mean
from matplotlib import pyplot as plt
from fileManager import listing

class gen():

    def __init__(self):
        super(gen, self).__init__()
        self.clas = "none"
        self.ind = "none"

    def smartHist(self, path):
        """Smart calculation of threshold limits, changes of concavity and
           best value to do thresholding"""

        img = cv2.imread(path)
        #resize and convert image
        shape = img.shape
        img = cv2.resize(img, (int(shape[1] / 4), int(shape[0] / 4)),
                                        interpolation=cv2.INTER_AREA)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        imgH = img[:, :, 2]
        #calculate Histogram
        histList = np.histogram(imgH, 255)
        histVals = list(histList[0])

        #remove pixels with zero insertions
        for pix in histVals:
            if pix == 0:
                histVals.remove(pix)

        #smooth surface
        nPart = int(len(histVals) / 3)
        for n in range(nPart):
            ker = histVals[3 * n: (3 * (n + 1))]
            if len(ker) == 3:
                ker = [mean(ker), mean(ker), mean(ker)]
                histVals[3 * n: (3 * (n + 1))] = ker

        #find convergences changes
        maxim = []
        for sw in range(5, 50):
            if sw % 2 == 1:
```

```
                kerS = sw
                nPart = int(len(histVals) / kerS)
                for n in range(nPart):
                    ker = histVals[kerS * n: (kerS * (n + 1))]
                    meanK = mean(ker)

                    if len(ker) == kerS:
                        if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                            if max(ker) not in maxim:
                                maxim.append(max(ker))
        maxim.sort()
        tolerance = 45
        best = histVals.index(maxim[-1]) - tolerance
        dummy, im = cv2.threshold(imgH, best, 255, cv2.THRESH_BINARY_INV)

        #erosions and dilations
        #defines a kernel to erode and dilate binary image
        kernel = np.ones((3, 3), dtype=np.uint8)

        #erodes and dilate image to eliminate holes inside particles
        for i in range(4):
            if i % 2 == 0:
                dilat = cv2.dilate(im, kernel, iterations=2)
                im = cv2.erode(dilat, kernel, iterations=1)
            else:
                ers = cv2.erode(im, kernel, iterations=2)
                im = cv2.dilate(ers, kernel, iterations=1)
        cv2.imshow("aa", im)
        cv2.waitKey(0)
        return im

    def splitImg(self, im, real):
        """find every particles in image also generate little square images for
           every detected particle"""
        allPart = 15
        real = cv2.imread(real)
        #calculate contours of image.
        #Note: im and count is not used but it's necessary to run'
        img, cont, hierarchy = cv2.findContours(im, cv2.RETR_EXTERNAL,
                                                cv2.CHAIN_APPROX_SIMPLE
                                                )
        order = []
        notOrder = []
        for c in cont:
            x, y, w, h = cv2.boundingRect(c)
            order.append(w * h)
            notOrder.append(w * h)

        order.sort()
        order.reverse()
        realPt = order[:allPart]
        realPt = [notOrder.index(pt) for pt in realPt]

        indG = 0
        for m in realPt:
            x, y, w, h = cv2.boundingRect(cont[m])
            print (x, y, w, h)
            cv2.imwrite(self.clas + self.ind + str(indG) + ".png",
                                real[y * 4: (y + h) * 4, x * 4: (x + w) * 4, :])
```

```
            indG += 1

if __name__ == "__main__":
    #look for every image in selected folder
    part = listing("Train", "JPG")
    wk = gen()
    wk.clas = "4-"
    for n in range(len(part)):
        item = str(n + 1) + "-"
        wk.ind = item
        im = wk.smartHist(part[n])
        wk.splitImg(im, part[n])
#---------------- End Algorithm 1 ------------------
```

Algoritmo 2: *Generación de listas de objetos con extensión similar*

```
#---------------- Start Algorithm 2 ------------------
"""load path name of files in a folder return a list
   requyre a kind of file"""

import re
from os import listdir

def listing(path, kind):
    #create a RE to check jpg files
    obj = re.compile("." + kind)
    elements = listdir(path)
    elements.sort()

    labels = []
    for elem in elements:
        l_obj = obj.findall(elem)
        try:
            if l_obj[0] == "." + kind:
                labels.append(path + "/" + elem)
            else:
                pass
        except:
            pass
    return  labels
#---------------- End Algorithm 2 ------------------
```

Algoritmo 3: *Algoritmo generador de los rangos de comparación para el clasificador basado en descriptores de forma*

```
#---------------- Start Algorithm 3 ------------------
"""Automatized code than look for the ranges of morphologyc values of
   any class located in a folder of samples"""

import cv2
import numpy as np
import threading
import psutil as ps
from matplotlib import pyplot as plt
from queue import Queue
from time import time, sleep
from statistics import stdev, mean
from fileManager import listing
from datetime import datetime
```

```
def smartHist(path):
    """Smart calculation of threshold limits, changes of concavity and best
       Value to do thresholding"""

    img = cv2.imread(path)
    #resize and convert image
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    imgH = img[:, :, 2]
    imgH = cv2.medianBlur(imgH, 5, 10)
    #calculate Histogram
    histList = np.histogram(imgH, 255)
    histVals = list(histList[0])
    #remove pixels with zero insertions
    for pix in histVals:
        if pix == 0:
            histVals.remove(pix)
    #smooth surface
    nPart = int(len(histVals) / 3)
    for n in range(nPart):
        ker = histVals[3 * n: (3 * (n + 1))]
        if len(ker) == 3:
            ker = [mean(ker), mean(ker), mean(ker)]
            histVals[3 * n: (3 * (n + 1))] = ker
    #find convergences changes
#    minim = []
    maxim = []
    for sw in range(5, 50):
        if sw % 2 == 1:
            kerS = sw
            nPart = int(len(histVals) / kerS)
            for n in range(nPart):
                ker = histVals[kerS * n: (kerS * (n + 1))]
                meanK = mean(ker)

                if len(ker) == kerS:
                    if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                        if max(ker) not in maxim:
                            maxim.append(max(ker))
    maxim.sort()
    tolerance = 0
    #print ([histVals.index(m) for m in maxim])
    best = histVals.index(maxim[len(maxim) - 1]) + tolerance
    dummy, im = cv2.threshold(imgH, 100, 255, cv2.THRESH_BINARY_INV)
    #erosions and dilations
    #defines a kernel to erode and dilate binary image
    kernel = np.ones((3, 3), dtype=np.uint8)




    #erodes and dilate image to eliminate holes inside particles
    for i in range(4):
        if i % 2 == 0:
            dilat = cv2.dilate(im, kernel, iterations=2)
            im = cv2.erode(dilat, kernel, iterations=1)
        else:
            ers = cv2.erode(im, kernel, iterations=2)
            im = cv2.dilate(ers, kernel, iterations=1)
```

```
    return im

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)

        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))

    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))
mnt = Queue()
mnt.put(True)
startTime = time()
perfMon = threading.Thread(target=monitor)
perfMon.start()
pth = ["samples/ang", "samples/ast", "samples/blc", "samples/red"]
for path in pth:
    images = listing(path, "png")
    Inv, Asp, ArT = [], [], []
    for i in images:
        imgH = smartHist(i)
        #calculate contours of image.
        #Note: im and count is not used but it's necessary to run'
        im, cont, hierarchy = cv2.findContours(imgH, cv2.RETR_EXTERNAL,
                                               cv2.CHAIN_APPROX_SIMPLE
                                               )
        order = []
        for c in cont:
            x, y, w, h = cv2.boundingRect(c)
            order.append(w * h)
        ind = order.index(max(order))
        #calculate parameters
        c = cont[ind]
        areaT = np.around(cv2.contourArea(c), 2)

        #values of any ellipse that enclose the contour
        center, Diameters, angle = cv2.fitEllipse(c)
        diamMin = np.around(Diameters[0], 2)
        diamMax = np.around(Diameters[1], 2)
        #splintery
        InvFormFactor = diamMin / float(diamMax)
        Inv.append(InvFormFactor)
        ar = []
```

```
        #calculate the defects and distant points of contour
        hull = cv2.convexHull(c, returnPoints=False)
        defects = cv2.convexityDefects(c, hull)
        #create a list of relevant defects
        for i in range(defects.shape[0]):
            s, e, f, d = defects[i, 0]
            start = tuple(c[s][0])
            end = tuple(c[e][0])
            far = tuple(c[f][0])
            #Roundess (look for soft or contours)
            #calculate area of a triangle of inner defects (Heron)
            s1 = (((start[0] - end[0]) ** 2) +
                  ((start[1] - end[1]) ** 2)) ** (0.5)
            s2 = (((end[0] - far[0]) ** 2) +
                  ((end[1] - far[1]) ** 2)) ** (0.5)
            s3 = ((
                  (far[0] - start[0]) ** 2) +
                  ((far[1] - start[1]) ** 2)) ** (0.5)
            #calculate semipermtr and area of inner defect triangle
            sem = (s1 + s2 + s3) / float(2)
            areaTr = (sem * (sem - s1) * (sem - s2) * (sem - s3)
                      ) ** (0.5)
            ar.append(areaTr / areaT)
        #Blocker
        #check for area of minim square than wrap contourn
        rect = cv2.minAreaRect(c)
        box = cv2.boxPoints(rect)
        box = np.int0(box)
        areaSqr = np.around(cv2.contourArea(box), 2)
        #check for blocker, using realtion of aspect with a square
        Aspect = areaT / areaSqr
        Asp.append(Aspect)
        #discrimine with area of inner triangles and aspect
        meanAreaTr = sum(ar) / len(ar)
        ArT.append(meanAreaTr)

    print ("\n\n" + path)
    print ("Limits Inv: ", min(Inv), max(Inv))
    print ("Limits Asp: ", min(Asp), max(Asp))
    print ("Limits ArT: ", min(ArT), max(ArT))

endTime = time()
mnt.put(False)
#print elapsed time
elap = round(endTime - startTime, 2)
print (elap)
#---------------- End Algorithm 3 ------------------
```

Algoritmo 4: *hog_processing.py: algoritmo que aplica un procesamiento de histogramas basados en gradientes orientados*

```
#---------------- Start Algorithm 4 ------------------
import cv2
import numpy as np
from numpy.linalg import norm

class Hog_way():
```

```
    def preprocess_hog(self, im_data):
        """receive a list of images of training and every image is analized
           With the structure Histogram Oriented Gradients, remeber use
           images with size 200x200 px, to separate image in equal 4 parts of
           100x100"""
        samples = []
        for img in im_data:
            smp_img = self.hog_single(img)
            samples = samples + smp_img
        return np.float32(samples)

    def hog_single(self, img):
        """to use with image than is going to test"""
        #calculate shape
        shape = img.shape
        h = int(shape[1] / 2)
        w = int(shape[0] / 2)
        samples = []
        gx = cv2.Sobel(img, cv2.CV_32F, 1, 0)
        gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)
        mag, ang = cv2.cartToPolar(gx, gy)
        bin_n = 16
        binary = np.int32(bin_n * ang / (2 * np.pi))
        bin_cells = (binary[:w, :h],
                     binary[w:, :h],
                     binary[:w, h:],
                     binary[w:, h:])
        mag_cells = (mag[:w, :h],
                     mag[w:, :h],
                     mag[:w, h:],
                     mag[w:, h:])
        hists = [np.bincount(b.ravel(), m.ravel(), bin_n)
                 for b, m in zip(bin_cells, mag_cells)]
        hist = np.hstack(hists)
        # transform to Hellinger kernel
        eps = 1e-7
        hist /= hist.sum() + eps
        hist = np.sqrt(hist)
        hist /= norm(hist) + eps
        samples.append(hist)
        #print (samples[0])
        return samples

if __name__ == "__main__":
    im = cv2.imread("test.png")
    Hog_way().hog_single(im)
#---------------- End Algorithm 4 ------------------
```

Algoritmo 5: *class_svm.py: clase utilizada para el entrenamiento de clasificador basado en SVM y para la implementación del clasificador*

```
#---------------- Start Algorithm 5 ------------------
import cv2
import numpy as np
from file_im import fileIm
from hog_processing import Hog_way
import threading
import psutil as ps
from queue import Queue
```

```
from datetime import datetime
from time import time, sleep
from sklearn import svm
from sklearn.externals import joblib
from statistics import mean

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)

        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))
    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))
mnt = Queue()
mnt.put(True)

class svm_classf():
    """classifier based in support vectors machine, than use lines or vectors
       to separe objects in a range of parameters"""
    def trainSVM(self):
        """load images of trainig, train model with all images"""
        #load list of images and its respective label
        imgs, labels = fileIm().check("data")
        labels = np.array(labels)
        #create a list of hist samples with hog structure
        samples = Hog_way().preprocess_hog(imgs)
        #create a set data
        model = svm.SVC(kernel='linear')
        model.fit(samples, labels)
        joblib.dump(model, 'models/svm/cuttings.pkl')


    def predictSVM(self, img):
        #self.trainSVM()
        #load sample
        sample = Hog_way().hog_single(img)
        sample = np.float32(sample)
        #load model
        model = joblib.load('models/svm/cuttings.pkl')
        pred = model.predict(sample)
        return pred

if __name__ == "__main__":
```

```
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    resp = svm_classf().trainSVM()
    end = time()
    mnt.put(False)
    print (end - init)
#---------------- End Algorithm 5 -----------------
```

Algoritmo 6: *class_sgd.py: clase utilizada para el entrenamiento de clasificador basado en SGD y para la implementación del clasificador*

```
#---------------- Start Algorithm 6 -----------------
import cv2
import numpy as np
from file_im import fileIm
from hog_processing import Hog_way
import threading
import psutil as ps
from queue import Queue
from datetime import datetime
from time import time, sleep
from sklearn import linear_model
from sklearn.externals import joblib
from statistics import mean

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)
        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))



    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))
mnt = Queue()
mnt.put(True)

class sgd_classf():
    """classifier based in stochastic gradient descend"""
    def trainSGD(self):
        """load images of trainig, train model with all images"""
        #load list of images and its respective label
        imgs, labels = fileIm().check("data")
```

```
        labels = np.array(labels)
        #create a list of hist samples with hog structure
        samples = Hog_way().preprocess_hog(imgs)
        #create a set data
        model = linear_model.SGDClassifier(loss="modified_huber",
                                           penalty="elasticnet")
        model.fit(samples, labels)
        joblib.dump(model, 'models/sgd/cuttings.pkl')

    def predictSGD(self, img):
        #self.trainSGD()
        #load sample
        sample = Hog_way().hog_single(img)
        sample = np.float32(sample)
        #load model
        model = joblib.load('models/svm/cuttings.pkl')
        pred = model.predict(sample)
        return pred

if __name__ == "__main__":
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    resp = sgd_classf().trainSGD()
    end = time()
    mnt.put(False)
    print (end - init)
#---------------- End Algorithm 6 ------------------
```

Algoritmo 7: *class_nb.py: clase utilizada para el entrenamiento de clasificador basado en Nave Bayes y para la implementación del clasificador*

```
#---------------- Start Algorithm 7 ------------------
import cv2
import numpy as np
from file_im import fileIm
from hog_processing import Hog_way
import threading
import psutil as ps
from queue import Queue
from datetime import datetime
from time import time, sleep
from sklearn.naive_bayes import GaussianNB
from sklearn.externals import joblib
from statistics import mean
def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)
        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
```

```
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))
    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))
mnt = Queue()
mnt.put(True)

class nb_classf():
    """classifier based in naive bayes"""
    def trainNB(self):
        """load images of trainig, train model with all images"""
        #load list of images and its respective label
        imgs, labels = fileIm().check("data")
        labels = np.array(labels)
        #create a list of hist samples with hog structure
        samples = Hog_way().preprocess_hog(imgs)
        #create a set data
        model = GaussianNB()
        model.fit(samples, labels)
        joblib.dump(model, 'models/nb/cuttings.pkl')

    def predictNB(self, img):
        #self.trainNB()
        #load sample
        sample = Hog_way().hog_single(img)
        sample = np.float32(sample)
        #load model
        model = joblib.load('models/svm/cuttings.pkl')
        pred = model.predict(sample)
        return pred

if __name__ == "__main__":
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    resp = nb_classf().trainNB()
    end = time()
    mnt.put(False)
    print (end - init)
#---------------- End Algorithm 7 -----------------
```

Algoritmo 8: *class_vc.py: clase utilizada para el entrenamiento de clasificador basado en Vecinos cercanos y para la implementación del clasificador*

```
#---------------- Start Algorithm 8 -----------------
import cv2
import numpy as np
from file_im import fileIm
from hog_processing import Hog_way
import threading
import psutil as ps
from queue import Queue
from datetime import datetime
from time import time, sleep
from sklearn import neighbors
from sklearn.externals import joblib
from statistics import mean
```

```
def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)
        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))
    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))
mnt = Queue()
mnt.put(True)

class vc_classf():
    """classifier based in nearest neighbors"""

    def trainVC(self):
        """load images of trainig, train model with all images"""
        #load list of images and its respective label
        imgs, labels = fileIm().check("data")
        labels = np.array(labels)
        #create a list of hist samples with hog structure
        samples = Hog_way().preprocess_hog(imgs)
        #create a set data
        model = neighbors.KNeighborsClassifier()
        model.fit(samples, labels)
        joblib.dump(model, 'models/vc/cuttings.pkl')



    def predictVC(self, img):
        #self.trainVC()
        #load sample
        sample = Hog_way().hog_single(img)
        sample = np.float32(sample)
        #load model
        model = joblib.load('models/vc/cuttings.pkl')
        pred = model.predict(sample)
        return pred

if __name__ == "__main__":
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    resp = vc_classf().trainVC()
    end = time()
```

```
    mnt.put(False)
    print (end - init)
#---------------- End Algorithm 8 ------------------
```

Algoritmo 9: *class_ad.py: clase utilizada para el entrenamiento de clasificador basado en Arboles de decisión y para la implementación del clasificador*

```
#---------------- Start Algorithm 9 ------------------
import cv2
import numpy as np
from file_im import fileIm
from hog_processing import Hog_way
import threading
import psutil as ps
from queue import Queue
from datetime import datetime
from time import time, sleep
from sklearn import tree
from sklearn.externals import joblib
from statistics import mean

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)
        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))

    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))
mnt = Queue()
mnt.put(True)

class ad_classf():
    """classifier based in decision trees"""
    def trainAD(self):
        """load images of trainig, train model with all images"""
        #load list of images and its respective label
        imgs, labels = fileIm().check("data")
        labels = np.array(labels)
        #create a list of hist samples with hog structure
        samples = Hog_way().preprocess_hog(imgs)
        #create a set data
        model = tree.DecisionTreeClassifier()
        model.fit(samples, labels)
        joblib.dump(model, 'models/ad/cuttings.pkl')
```

```
    def predictAD(self, img):
        #self.trainAD()
        #load sample
        sample = Hog_way().hog_single(img)
        sample = np.float32(sample)
        #load model
        model = joblib.load('models/svm/cuttings.pkl')
        pred = model.predict(sample)
        return pred

if __name__ == "__main__":
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    resp = ad_classf().trainAD()
    end = time()
    mnt.put(False)
    print (end - init)
#---------------- End Algorithm 9 ------------------
```

Algoritmo 10: *Algoritmo de entrenamiento del clasificador basado en redes neuronales*

```
#---------------- Start Algorithm 10 ------------------
import os
import threading
import psutil as ps
from queue import Queue
from datetime import datetime
from time import time, sleep
from statistics import mean

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)

        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))
    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))
mnt = Queue()
mnt.put(True)
init = time()
perfMon = threading.Thread(target=monitor)
perfMon.start()
```

```
#-------------------------------------------------------
os.system("cd Angular/data/cascade && touch cascade.xml")
os.system("cd Angular && opencv_createsamples -info positi\
           vas.txt -vec data/muestra.vec -num 45 -w 20 -h 20")
os.system("cd Angular && opencv_traincascade -data data/ca\
           scade -vec data/muestra.vec -bg negativas.txt -\
               numPos 45 -numNeg 135 -numStages 5 -stageType B\
               OOST -featureType HAAR -w 20 -h 20 -minHitRate \
               0.995 maxDepth 1 -maxFalseAlarmRate 0.1")
#-------------------------------------------------------
#-------------------------------------------------------
os.system("cd Astilloso/data/cascade && touch cascade.xml")
os.system("cd Astilloso && opencv_createsamples -info positi\
           vas.txt -vec data/muestra.vec -num 45 -w 20 -h 20")
os.system("cd Astilloso && opencv_traincascade -data data/ca\
           scade -vec data/muestra.vec -bg negativas.txt -nu\
               mPos 45 -numNeg 135 -numStages 5 -stageType BOOST\
               -featureType HAAR -w 20 -h 20 -minHitRate 0.995 m\
               axDepth 1 -maxFalseAlarmRate 0.1")
#-------------------------------------------------------
#-------------------------------------------------------
os.system("cd Blocoso/data/cascade && touch cascade.xml")
os.system("cd Blocoso && opencv_createsamples -info positi\
           vas.txt -vec data/muestra.vec -num 45 -w 20 -h 20")
os.system("cd Blocoso && opencv_traincascade -data data/ca\
           scade -vec data/muestra.vec -bg negativas.txt -\
               numPos 45 -numNeg 135 -numStages 5 -stageType B\
               OOST -featureType HAAR -w 20 -h 20 -minHitRate \
               0.995 maxDepth 1 -maxFalseAlarmRate 0.1")
#-------------------------------------------------------
#-------------------------------------------------------
os.system("cd Redondeado/data/cascade && touch cascade.xml")
os.system("cd Redondeado && opencv_createsamples -info positi\
           vas.txt -vec data/muestra.vec -num 45 -w 20 -h 20")
os.system("cd Redondeado && opencv_traincascade -data data/ca\
           scade -vec data/muestra.vec -bg negativas.txt -num\
               Pos 45 -numNeg 135 -numStages 5 -stageType BOOST -\
               featureType HAAR -w 20 -h 20 -minHitRate 0.995 max\
               Depth 1 -maxFalseAlarmRate 0.1")
#-------------------------------------------------------
end = time()
mnt.put(False)
print (end - init)
#---------------- End Algorithm 10 ------------------
```

Algoritmo 11: *clasificador basado en factor de forma*

```
#---------------- Start Algorithm 11 ------------------
"""here is located all the code to do particle recognition. It has a test mode
than shows real interactions between script and video. so show a processed
image in real time"""
import os
import cv2
import numpy as np
import psutil as ps
import threading
import json
from datetime import datetime
from time import sleep, time
from queue import Queue
```

```
from statistics import mean
from __init__ import file_son
from fileManager import listing
from class_svm import svm_classf

mnt = Queue()
mnt.put(True)

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)
        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))

    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))

def smartHist(path):
    """Smart calculation of umbral limits, changes of concavity and best value
       to do thresholding"""
    img = cv2.imread(path)
    #resize and convert image
    shape = img.shape
    img = cv2.resize(img, (int(shape[1] / 4), int(shape[0] / 4)),
                                       interpolation=cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    imgH = img[:, :, 2]
    #calculate Histogram
    histList = np.histogram(imgH, 255)
    histVals = list(histList[0])
    #remove pixels with zero insertions
    for pix in histVals:
        if pix == 0:
            histVals.remove(pix)
    #smooth surface
    nPart = int(len(histVals) / 3)
    for n in range(nPart):
        ker = histVals[3 * n: (3 * (n + 1))]
        if len(ker) == 3:
            ker = [mean(ker), mean(ker), mean(ker)]
            histVals[3 * n: (3 * (n + 1))] = ker
    #find convergences changes
#     minim = []
    maxim = []
```

```
    for sw in range(5, 50):
        if sw % 2 == 1:
            kerS = sw
            nPart = int(len(histVals) / kerS)
            for n in range(nPart):
                ker = histVals[kerS * n: (kerS * (n + 1))]
                meanK = mean(ker)

                if len(ker) == kerS:
                    if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                        if max(ker) not in maxim:
                            maxim.append(max(ker))
    maxim.sort()
    tolerance = 35
    best = histVals.index(maxim[-1]) - tolerance
    dummy, im = cv2.threshold(imgH, best, 255, cv2.THRESH_BINARY_INV)
    #erosions and dilations
    #defines a kernel to erode and dilate binary image
    kernel = np.ones((3, 3), dtype=np.uint8)
    #erodes and dilate image to eliminate holes inside particles
    for i in range(4):
        if i % 2 == 0:
            dilat = cv2.dilate(im, kernel, iterations=2)
            im = cv2.erode(dilat, kernel, iterations=1)
        else:
            ers = cv2.erode(im, kernel, iterations=2)
            im = cv2.dilate(ers, kernel, iterations=1)
    return im

class graph_analyse():
    def __init__(self):
        #define parameters of cameras
        son = file_son()
        config = son.read_son("config.json")

        self.weigth = config["weigth"]
        self.heigth = config["heigth"]
        self.fps = config["fps"]
        self.test = False
        self.window = np.zeros((100, 100, 3), np.uint8)

    def image_parser(self, img_file, **kwargs):
        """parse image in particles,
        requires a image file(absolute path as string)
        and returns an array with the information of particles"""

        imageH = smartHist(img_file)
        #Note: im and count is not used but it's necessary to run'
        im, cont, hierarchy = cv2.findContours(imageH, cv2.RETR_EXTERNAL,
                                               cv2.CHAIN_APPROX_SIMPLE
                                               )
        image = cv2.imread(img_file)
        shape = image.shape
        image = cv2.resize(image, (int(shape[1] / 4), int(shape[0] / 4)),
                                    interpolation=cv2.INTER_AREA)
        return cont, image

    def particle_recognizer(self, contour, image, **kwargs):
        """recognize particles in the array of countours,
```

```
        requieres a array with the information of contours, and returns the
        a list with the category of every particle in the contour,
        for more information watch categories in documentation"""
        #list to add information of contour and list areas
        information = []
        areas = []
        #analyzes the contours
        for c in contour:
            #check the dimentions of a square than enclosed the contour
            x, y, w, h = cv2.boundingRect(c)
            #remove contour with small dimentions
            if (w * h) > 800:
                img = image[y: y + h, x: x + w,  :]
                result = svm_classf().predictSVM(img)[0]
                if result == 1:
                    cav = "Angular"
                    color = (255, 0, 0)
                if result == 2:
                    cav = "Astilloso"
                    color = (0, 255, 0)
                if result == 3:
                    cav = "Blocoso"
                    color = (0, 0, 255)
                if result == 4:
                    cav = "Redondeado"
                    color = (0, 255, 255)
                information.append(cav)
                font = cv2.FONT_HERSHEY_SIMPLEX
                cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
                cv2.putText(image, cav, (x, y), font, 1, color, 2)
        dictInfo = {'categories': information, 'areas': areas}
        return dictInfo, image

if __name__ == "__main__":
    ga = graph_analyse()
    #to save results uncomment below
    #part = listing("../Test/samples", "JPG")
    #reg = open("results", "w")
    #for p in part:
        #cont, image = ga.image_parser(p)
        #dictInfo, img show = ga.particle recognizer(cont, image)
        #reg.write(json.dumps(dictInfo) + "\n")
    #reg.close()
    #end = time()
    #mnt.put(False)
    #print (end - init)

    cont, image = ga.image_parser("pb.jpg")
    dictInfo, img_show = ga.particle_recognizer(cont, image)
#---------------- End Algorithm 11 ------------------
```

Algoritmo 12: *Clasificador basado Maquinas de soporte vectorial*

```
#---------------- Start Algorithm 12 ------------------
"""here is located all the code to do particle recognition. It has a test mode
than shows real interactions between script and video. so show a processed
image in real time"""
import os
import cv2
import numpy as np
```

```
import psutil as ps
import threading
import json
from datetime import datetime
from time import sleep, time
from queue import Queue
from statistics import mean
from __init__ import file_son
from fileManager import listing
from class_svm import svm_classf

mnt = Queue()
mnt.put(True)

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)
        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))

    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))

def smartHist(path):
    """Smart calculation of umbral limits, changes of concavity and best value
       to do thresholding"""
    img = cv2.imread(path)
    #resize and convert image
    shape = img.shape
    img = cv2.resize(img, (int(shape[1] / 4), int(shape[0] / 4)),
                                       interpolation=cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    imgH = img[:, :, 2]
    #calculate Histogram
    histList = np.histogram(imgH, 255)
    histVals = list(histList[0])
    #remove pixels with zero insertions
    for pix in histVals:
        if pix == 0:
            histVals.remove(pix)
    #smooth surface
    nPart = int(len(histVals) / 3)
    for n in range(nPart):
        ker = histVals[3 * n: (3 * (n + 1))]
```

```
            if len(ker) == 3:
                ker = [mean(ker), mean(ker), mean(ker)]
                histVals[3 * n: (3 * (n + 1))] = ker
        #find convergences changes
        maxim = []
        for sw in range(5, 50):
            if sw % 2 == 1:
                kerS = sw
                nPart = int(len(histVals) / kerS)
                for n in range(nPart):
                    ker = histVals[kerS * n: (kerS * (n + 1))]
                    meanK = mean(ker)

                    if len(ker) == kerS:
                        if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                            if max(ker) not in maxim:
                                maxim.append(max(ker))
        maxim.sort()
        tolerance = 35
        best = histVals.index(maxim[-1]) - tolerance
        dummy, im = cv2.threshold(imgH, best, 255, cv2.THRESH_BINARY_INV)
        #erosions and dilations
        #defines a kernel to erode and dilate binary image
        kernel = np.ones((3, 3), dtype=np.uint8)
        #erodes and dilate image to eliminate holes inside particles
        for i in range(4):
            if i % 2 == 0:
                dilat = cv2.dilate(im, kernel, iterations=2)
                im = cv2.erode(dilat, kernel, iterations=1)
            else:
                ers = cv2.erode(im, kernel, iterations=2)
                im = cv2.dilate(ers, kernel, iterations=1)

        return im


    class graph_analyse():
        def __init__(self):
            #define parameters of cameras
            son = file_son()
            config = son.read_son("config.json")

            self.weigth = config["weigth"]
            self.heigth = config["heigth"]
            self.fps = config["fps"]
            self.test = False
            self.window = np.zeros((100, 100, 3), np.uint8)

        def image_parser(self, img_file, **kwargs):
            """parse image in particles,
            requires a image file(absolute path as string)
            and returns an array with the information of particles"""

            imageH = smartHist(img_file)
            #Note: im and count is not used but it's necessary to run'
            im, cont, hierarchy = cv2.findContours(imageH, cv2.RETR_EXTERNAL,
                                                   cv2.CHAIN_APPROX_SIMPLE
                                                   )
            image = cv2.imread(img_file)
```

```
        shape = image.shape
        image = cv2.resize(image, (int(shape[1] / 4), int(shape[0] / 4)),
                                    interpolation=cv2.INTER_AREA)
        return cont, image

    def particle_recognizer(self, contour, image, **kwargs):
        """recognize particles in the array of countours,
        requieres a array with the information of contours, and returns the
        a list with the category of every particle in the contour,
        for more information watch categories in documentation"""
        #list to add information of contour and list areas
        information = []
        areas = []
        #analyzes the contours
        for c in contour:
            #check the dimentions of a square than enclosed the contour
            x, y, w, h = cv2.boundingRect(c)
            #remove contour with small dimentions
            if (w * h) > 800:
                img = image[y: y + h, x: x + w,  :]
                result = svm_classf().predictSVM(img)[0]
                if result == 1:
                    cav = "Angular"
                    color = (255, 0, 0)
                if result == 2:
                    cav = "Astilloso"
                    color = (0, 255, 0)
                if result == 3:
                    cav = "Blocoso"
                    color = (0, 0, 255)
                if result == 4:
                    cav = "Redondeado"
                    color = (0, 255, 255)
                information.append(cav)
                font = cv2.FONT_HERSHEY_SIMPLEX
                cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
                cv2.putText(image, cav, (x, y), font, 1, color, 2)
        dictInfo = {'categories': information, 'areas': areas}
        return dictInfo, image

if __name__ == "__main__":
    ga = graph_analyse()
    #to save results uncomment below
    #part = listing("../Test/samples", "JPG")
    #reg = open("results", "w")
    #for p in part:
        #cont, image = ga.image_parser(p)
        #dictInfo, img_show = ga.particle_recognizer(cont, image)
        #reg.write(json.dumps(dictInfo) + "\n")
    #reg.close()
    #end = time()
    #mnt.put(False)
    #print (end - init)

    cont, image = ga.image_parser("pb.jpg")
    dictInfo, img_show = ga.particle_recognizer(cont, image)
    cv2.imshow("hola", img_show)
    cv2.waitKey(0)
#---------------- End Algorithm 12 -----------------
```

Algoritmo 13: *Clasificador basado en Gradiente estocástico descendente*

```
#---------------- Start Algorithm 13 ------------------
"""
here is located all the code to do particle recognition. It has a test mode
than shows real interactions between script and video. so show a processed
image in real time
"""
import os
import cv2
import numpy as np
import psutil as ps
import threading
import json
from datetime import datetime
from time import sleep, time
from queue import Queue
from statistics import mean
from __init__ import file_son
from fileManager import listing
from class_sgd import sgd_classf

mnt = Queue()
mnt.put(True)

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []

    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)

        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))
    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))

def smartHist(path):
    """Smart calculation of umbral limits, changes of concavity and best value
       to do thresholding"""
    img = cv2.imread(path)
    #resize and convert image
    shape = img.shape
    img = cv2.resize(img, (int(shape[1] / 4), int(shape[0] / 4)),
                                       interpolation=cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

```
        imgH = img[:, :, 2]
        #calculate Histogram
        histList = np.histogram(imgH, 255)
        histVals = list(histList[0])
        #remove pixels with zero insertions
        for pix in histVals:
            if pix == 0:
                histVals.remove(pix)
        #smooth surface
        nPart = int(len(histVals) / 3)
        for n in range(nPart):
            ker = histVals[3 * n: (3 * (n + 1))]
            if len(ker) == 3:
                ker = [mean(ker), mean(ker), mean(ker)]
                histVals[3 * n: (3 * (n + 1))] = ker
        #find convergences changes
        maxim = []
        for sw in range(5, 50):
            if sw % 2 == 1:
                kerS = sw
                nPart = int(len(histVals) / kerS)
                for n in range(nPart):
                    ker = histVals[kerS * n: (kerS * (n + 1))]
                    meanK = mean(ker)

                    if len(ker) == kerS:
                        if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                            if max(ker) not in maxim:
                                maxim.append(max(ker))
        maxim.sort()
        tolerance = 35
        best = histVals.index(maxim[-1]) - tolerance
        dummy, im = cv2.threshold(imgH, best, 255, cv2.THRESH_BINARY_INV)


        #erosions and dilations
        #defines a kernel to erode and dilate binary image
        kernel = np.ones((3, 3), dtype=np.uint8)
        #erodes and dilate image to eliminate holes inside particles
        for i in range(4):
            if i % 2 == 0:
                dilat = cv2.dilate(im, kernel, iterations=2)
                im = cv2.erode(dilat, kernel, iterations=1)
            else:
                ers = cv2.erode(im, kernel, iterations=2)
                im = cv2.dilate(ers, kernel, iterations=1)
        return im

class graph_analyse():
    def __init__(self):
        #define parameters of cameras
        son = file_son()
        config = son.read_son("config.json")
        self.weigth = config["weigth"]
        self.heigth = config["heigth"]
        self.fps = config["fps"]
        self.test = False
        self.window = np.zeros((100, 100, 3), np.uint8)
```

```
    def image_parser(self, img_file, **kwargs):
        """parse image in particles,
        requires a image file(absolute path as string)
        and returns an array with the information of particles"""
        imageH = smartHist(img_file)
        #Note: im and count is not used but it's necessary to run'
        im, cont, hierarchy = cv2.findContours(imageH, cv2.RETR_EXTERNAL,
                                               cv2.CHAIN_APPROX_SIMPLE
                                               )
        image = cv2.imread(img_file)
        shape = image.shape
        image = cv2.resize(image, (int(shape[1] / 4), int(shape[0] / 4)),
                                    interpolation=cv2.INTER_AREA)
        return cont, image

    def particle_recognizer(self, contour, image, **kwargs):
        """recognize particles in the array of countours,
        requieres a array with the information of contours, and returns the
        a list with the category of every particle in the contour,
        for more information watch categories in documentation"""
        #list to add information of contour and list areas
        information = []
        areas = []
        #analyzes the contours
        for c in contour:
            #check the dimentions of a square than enclosed the contour
            x, y, w, h = cv2.boundingRect(c)
            #remove contour with small dimentions
            if (w * h) > 800:
                img = image[y: y + h, x: x + w,  :]
                result = sgd_classf().predictSGD(img)[0]
                if result == 1:
                    cav = "Angular"
                if result == 2:
                    cav = "Astilloso"
                if result == 3:
                    cav = "Blocoso"
                if result == 4:
                    cav = "Redondeado"
                information.append(cav)
        dictInfo = {'categories': information, 'areas': areas}
        return dictInfo, image

if __name__ == "__main__":

    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    ga = graph_analyse()
    part = listing("../Test/samples", "JPG")
    reg = open("results", "w")
    for p in part:
        cont, image = ga.image_parser(p)
        dictInfo, img_show = ga.particle_recognizer(cont, image)
        reg.write(json.dumps(dictInfo) + "\n")
    reg.close()
    end = time()
    mnt.put(False)
    print (end - init)
```

```
#---------------- End Algorithm 13 ------------------
```

Algoritmo 14: *Clasificador basado en vecinos cercanos*

```
#---------------- Start Algorithm 14 ------------------
"""here is located all the code to do particle recognition. It has a test mode
than shows real interactions between script and video. so show a processed
image in real time"""
import os
import cv2
import numpy as np
import psutil as ps
import threading
import json
from datetime import datetime
from time import sleep, time
from queue import Queue
from statistics import mean
from __init__ import file_son
from fileManager import listing
from class_vc import vc_classf

mnt = Queue()
mnt.put(True)

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)

        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))

    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))

def smartHist(path):
    """Smart calculation of umbral limits, changes of concavity and best value
       to do thresholding"""
    img = cv2.imread(path)
    #resize and convert image
    shape = img.shape
    img = cv2.resize(img, (int(shape[1] / 4), int(shape[0] / 4)),
                                       interpolation=cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    imgH = img[:, :, 2]
```

```
    #calculate Histogram
    histList = np.histogram(imgH, 255)
    histVals = list(histList[0])
    #remove pixels with zero insertions
    for pix in histVals:
        if pix == 0:
            histVals.remove(pix)
    #smooth surface
    nPart = int(len(histVals) / 3)
    for n in range(nPart):
        ker = histVals[3 * n: (3 * (n + 1))]
        if len(ker) == 3:
            ker = [mean(ker), mean(ker), mean(ker)]
            histVals[3 * n: (3 * (n + 1))] = ker
    #find convergences changes
    maxim = []
    for sw in range(5, 50):
        if sw % 2 == 1:
            kerS = sw
            nPart = int(len(histVals) / kerS)
            for n in range(nPart):
                ker = histVals[kerS * n: (kerS * (n + 1))]
                meanK = mean(ker)

                if len(ker) == kerS:
                    if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                        if max(ker) not in maxim:
                            maxim.append(max(ker))
    maxim.sort()
    tolerance = 35
    best = histVals.index(maxim[-1]) - tolerance
    dummy, im = cv2.threshold(imgH, best, 255, cv2.THRESH_BINARY_INV)
    #erosions and dilations
    #defines a kernel to erode and dilate binary image
    kernel = np.ones((3, 3), dtype=np.uint8)
    #erodes and dilate image to eliminate holes inside particles
    for i in range(4):
        if i % 2 == 0:
            dilat = cv2.dilate(im, kernel, iterations=2)
            im = cv2.erode(dilat, kernel, iterations=1)
        else:
            ers = cv2.erode(im, kernel, iterations=2)
            im = cv2.dilate(ers, kernel, iterations=1)
    return im

class graph_analyse():
    def __init__(self):
        #define parameters of cameras
        son = file_son()
        config = son.read_son("config.json")
        self.weigth = config["weigth"]
        self.heigth = config["heigth"]
        self.fps = config["fps"]
        self.test = False
        self.window = np.zeros((100, 100, 3), np.uint8)

    def image_parser(self, img_file, **kwargs):
        """parse image in particles,
        requires a image file(absolute path as string)
```

```
        and returns an array with the information of particles"""
        imageH = smartHist(img_file)
        #Note: im and count is not used but it's necessary to run'
        im, cont, hierarchy = cv2.findContours(imageH, cv2.RETR_EXTERNAL,
                                               cv2.CHAIN_APPROX_SIMPLE
                                               )
        image = cv2.imread(img_file)
        shape = image.shape
        image = cv2.resize(image, (int(shape[1] / 4), int(shape[0] / 4)),
                                    interpolation=cv2.INTER_AREA)
        return cont, image

    def particle_recognizer(self, contour, image, **kwargs):
        """recognize particles in the array of countours,
        requieres a array with the information of contours, and returns the
        a list with the category of every particle in the contour,
        for more information watch categories in documentation"""

        #list to add information of contour and list areas
        information = []
        areas = []
        #analyzes the contours
        for c in contour:
            #check the dimentions of a square than enclosed the contour
            x, y, w, h = cv2.boundingRect(c)
            #remove contour with small dimentions
            if (w * h) > 800:
                img = image[y: y + h, x: x + w,  :]
                result = vc_classf().predictVC(img)[0]
                if result == 1:
                    cav = "Angular"
                if result == 2:
                    cav = "Astilloso"
                if result == 3:
                    cav = "Blocoso"
                if result == 4:
                    cav = "Redondeado"
                information.append(cav)
        dictInfo = {'categories': information, 'areas': areas}
        return dictInfo, image

if __name__ == "__main__":
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    ga = graph_analyse()
    part = listing("../Test/samples", "JPG")
    reg = open("results", "w")
    for p in part:
        cont, image = ga.image_parser(p)
        dictInfo, img_show = ga.particle_recognizer(cont, image)
        reg.write(json.dumps(dictInfo) + "\n")
    reg.close()
    end = time()
    mnt.put(False)
    print (end - init)
#---------------- End Algorithm 14 ------------------
```

Algoritmo 15: *Clasificador basado en naive bayes*

```
#---------------- Start Algorithm 15 ------------------
"""here is located all the code to do particle recognition. It has a test mode
than shows real interactions between script and video. so show a processed
image in real time"""
import os
import cv2
import numpy as np
import psutil as ps
import threading
import json
from datetime import datetime
from time import sleep, time
from queue import Queue
from statistics import mean
from __init__ import file_son
from fileManager import listing
from class_nb import nb_classf

mnt = Queue()
mnt.put(True)

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []

    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)

        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))

    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))

def smartHist(path):
    """Smart calculation of umbral limits, changes of concavity and best value
       to do thresholding"""
    img = cv2.imread(path)
    #resize and convert image
    shape = img.shape
    img = cv2.resize(img, (int(shape[1] / 4), int(shape[0] / 4)),
                                       interpolation=cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    imgH = img[:, :, 2]
    #calculate Histogram
    histList = np.histogram(imgH, 255)
```

```
    histVals = list(histList[0])
    #remove pixels with zero insertions
    for pix in histVals:
        if pix == 0:
            histVals.remove(pix)
    #smooth surface
    nPart = int(len(histVals) / 3)
    for n in range(nPart):
        ker = histVals[3 * n: (3 * (n + 1))]
        if len(ker) == 3:
            ker = [mean(ker), mean(ker), mean(ker)]
            histVals[3 * n: (3 * (n + 1))] = ker
    #find convergences changes
    maxim = []
    for sw in range(5, 50):
        if sw % 2 == 1:
            kerS = sw
            nPart = int(len(histVals) / kerS)
            for n in range(nPart):
                ker = histVals[kerS * n: (kerS * (n + 1))]
                meanK = mean(ker)

                if len(ker) == kerS:
                    if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                        if max(ker) not in maxim:
                            maxim.append(max(ker))
    maxim.sort()
    tolerance = 35
    best = histVals.index(maxim[-1]) - tolerance
    dummy, im = cv2.threshold(imgH, best, 255, cv2.THRESH_BINARY_INV)

    #erosions and dilations
    #defines a kernel to erode and dilate binary image
    kernel = np.ones((3, 3), dtype=np.uint8)
    #erodes and dilate image to eliminate holes inside particles
    for i in range(4):
        if i % 2 == 0:
            dilat = cv2.dilate(im, kernel, iterations=2)
            im = cv2.erode(dilat, kernel, iterations=1)
        else:
            ers = cv2.erode(im, kernel, iterations=2)
            im = cv2.dilate(ers, kernel, iterations=1)
    return im

class graph_analyse():
    def __init__(self):
        #define parameters of cameras
        son = file_son()
        config = son.read_son("config.json")
        self.weigth = config["weigth"]
        self.heigth = config["heigth"]
        self.fps = config["fps"]
        self.test = False
        self.window = np.zeros((100, 100, 3), np.uint8)

    def image_parser(self, img_file, **kwargs):
        """parse image in particles,
        requires a image file(absolute path as string)
        and returns an array with the information of particles"""
```

```
        imageH = smartHist(img_file)
        #Note: im and count is not used but it's necessary to run'
        im, cont, hierarchy = cv2.findContours(imageH, cv2.RETR_EXTERNAL,
                                               cv2.CHAIN_APPROX_SIMPLE
                                               )
        image = cv2.imread(img_file)
        shape = image.shape
        image = cv2.resize(image, (int(shape[1] / 4), int(shape[0] / 4)),
                                    interpolation=cv2.INTER_AREA)
        return cont, image

    def particle_recognizer(self, contour, image, **kwargs):
        """recognize particles in the array of countours,
        requieres a array with the information of contours, and returns the
        a list with the category of every particle in the contour,
        for more information watch categories in documentation"""

        #list to add information of contour and list areas
        information = []
        areas = []
        #analyzes the contours
        for c in contour:
            #check the dimentions of a square than enclosed the contour
            x, y, w, h = cv2.boundingRect(c)
            #remove contour with small dimentions
            if (w * h) > 800:
                img = image[y: y + h, x: x + w,  :]

                result = nb_classf().predictNB(img)[0]
                if result == 1:
                    cav = "Angular"
                if result == 2:
                    cav = "Astilloso"
                if result == 3:
                    cav = "Blocoso"
                if result == 4:
                    cav = "Redondeado"
                information.append(cav)
        dictInfo = {'categories': information, 'areas': areas}
        return dictInfo, image

if __name__ == "__main__":
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    ga = graph_analyse()
    part = listing("../Test/samples", "JPG")
    reg = open("results", "w")
    for p in part:
        cont, image = ga.image_parser(p)
        dictInfo, img_show = ga.particle_recognizer(cont, image)
        reg.write(json.dumps(dictInfo) + "\n")
    reg.close()
    end = time()
    mnt.put(False)
    print ((end - init)/4.0)
#---------------- End Algorithm 15 -----------------
```

Algoritmo 16: *Clasificador basado en arboles de decisión*

```
#---------------- Start Algorithm 16 ------------------
"""here is located all the code to do particle recognition. It has a test mode
than shows real interactions between script and video. so show a processed
image in real time"""
import os
import cv2
import numpy as np
import psutil as ps
import threading
import json
from datetime import datetime
from time import sleep, time
from queue import Queue
from statistics import mean
from __init__ import file_son
from fileManager import listing
from class_ad import ad_classf

mnt = Queue()
mnt.put(True)

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)

        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))
    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))

def smartHist(path):
    """Smart calculation of umbral limits, changes of concavity and best value
       to do thresholding"""
    img = cv2.imread(path)
    #resize and convert image
    shape = img.shape
    img = cv2.resize(img, (int(shape[1] / 4), int(shape[0] / 4)),
                                      interpolation=cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    imgH = img[:, :, 2]
    #calculate Histogram
    histList = np.histogram(imgH, 255)
    histVals = list(histList[0])
```

```
    #remove pixels with zero insertions
    for pix in histVals:
        if pix == 0:
            histVals.remove(pix)
    #smooth surface
    nPart = int(len(histVals) / 3)
    for n in range(nPart):
        ker = histVals[3 * n: (3 * (n + 1))]
        if len(ker) == 3:
            ker = [mean(ker), mean(ker), mean(ker)]
            histVals[3 * n: (3 * (n + 1))] = ker
    #find convergences changes
    maxim = []
    for sw in range(5, 50):
        if sw % 2 == 1:
            kerS = sw
            nPart = int(len(histVals) / kerS)
            for n in range(nPart):
                ker = histVals[kerS * n: (kerS * (n + 1))]
                meanK = mean(ker)

                if len(ker) == kerS:
                    if (ker[0] < meanK) and (ker[kerS - 1] < meanK):
                        if max(ker) not in maxim:
                            maxim.append(max(ker))
    maxim.sort()
    tolerance = 35
    best = histVals.index(maxim[-1]) - tolerance
    dummy, im = cv2.threshold(imgH, best, 255, cv2.THRESH_BINARY_INV)

    #erosions and dilations
    #defines a kernel to erode and dilate binary image
    kernel = np.ones((3, 3), dtype=np.uint8)
    #erodes and dilate image to eliminate holes inside particles
    for i in range(4):
        if i % 2 == 0:
            dilat = cv2.dilate(im, kernel, iterations=2)
            im = cv2.erode(dilat, kernel, iterations=1)
        else:
            ers = cv2.erode(im, kernel, iterations=2)
            im = cv2.dilate(ers, kernel, iterations=1)
    return im

class graph_analyse():
    def __init__(self):
        #define parameters of cameras
        son = file_son()
        config = son.read_son("config.json")
        self.weigth = config["weigth"]
        self.heigth = config["heigth"]
        self.fps = config["fps"]
        self.test = False
        self.window = np.zeros((100, 100, 3), np.uint8)

    def image_parser(self, img_file, **kwargs):
        """parse image in particles,
        requires a image file(absolute path as string)
        and returns an array with the information of particles"""
        imageH = smartHist(img_file)
```

```
        #Note: im and count is not used but it's necessary to run'
        im, cont, hierarchy = cv2.findContours(imageH, cv2.RETR_EXTERNAL,
                                               cv2.CHAIN_APPROX_SIMPLE)
        image = cv2.imread(img_file)
        shape = image.shape
        image = cv2.resize(image, (int(shape[1] / 4), int(shape[0] / 4)),
                                   interpolation=cv2.INTER_AREA)
        return cont, image

    def particle_recognizer(self, contour, image, **kwargs):
        """recognize particles in the array of countours,
        requieres a array with the information of contours, and returns the
        a list with the category of every particle in the contour,
        for more information watch categories in documentation"""
        #list to add information of contour and list areas
        information = []
        areas = []
        #analyzes the contours
        for c in contour:
            #check the dimentions of a square than enclosed the contour
            x, y, w, h = cv2.boundingRect(c)
            #remove contour with small dimentions
            if (w * h) > 800:
                img = image[y: y + h, x: x + w,  :]
                result = ad_classf().predictAD(img)[0]
                if result == 1:
                    cav = "Angular"
                if result == 2:
                    cav = "Astilloso"
                if result == 3:
                    cav = "Blocoso"
                if result == 4:
                    cav = "Redondeado"
                information.append(cav)
        dictInfo = {'categories': information, 'areas': areas}
        return dictInfo, image

if __name__ == "__main__":
    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    ga = graph_analyse()
    part = listing("../Test/samples", "JPG")
    reg = open("results", "w")
    for p in part:
        cont, image = ga.image_parser(p)
        dictInfo, img_show = ga.particle_recognizer(cont, image)
        reg.write(json.dumps(dictInfo) + "\n")
    reg.close()
    end = time()
    mnt.put(False)
    print ((end - init)/4.0)
#---------------- End Algorithm 16 -----------------
```

Algoritmo 17: *Clasificador basado en redes neuronales*

```
#---------------- Start Algorithm 17 -----------------
"""here is located all the code to do particle recognition. It has a test mode
than shows real interactions between script and video. so show a processed
image in real time"""
```

```
import os
import cv2
import numpy as np
import psutil as ps
import threading
import json
from datetime import datetime
from time import sleep, time
from queue import Queue
from statistics import mean
from fileManager import listing

mnt = Queue()
mnt.put(True)

def monitor():
    """do real time statistics of performance machine"""
    perf = open("performance", "a")
    proc, mem, swap = [], [], []
    sw = mnt.get()
    while sw:
        proc.append(ps.cpu_percent())
        mem.append(ps.virtual_memory().percent)
        swap.append(ps.swap_memory().percent)
        if not mnt.empty():
            sw = mnt.get()
        sleep(.3)
    perf.write("\n" + (100 * "#") + "\n\n")
    perf.write(str(datetime.utcnow()))
    perf.write("\n\nMaxim Values:\n")
    perf.write("processor: " + str(max(proc) - min(proc)) + " Ram ussage: " +
               str(max(mem) - min(mem)) + " Swap ussage: " +
               str(max(swap) - min(swap)))
    perf.write("\n\nMean Values:\n")
    perf.write("processor: " + str(mean(proc) - min(proc)) + " Ram ussage: " +
               str(mean(mem) - min(mem)) + " Swap ussage: " + str(mean(swap) -
               min(swap)))

def intercDealer(part, comp):
    w1, h1 = part[3], part[4]
    w2, h2 = comp[3], comp[4]
    #findind areas
    areaPart = w1 * h1
    areaComp = w2 * h2
    #first square
    ranx1Part = list(range(part[1], part[1] + part[3]))
    rany1Part = list(range(part[2], part[2] + part[4]))
    #second square
    ranx2Part = list(range(comp[1], comp[1] + comp[3]))
    rany2Part = list(range(comp[2], comp[2] + comp[4]))
    #find common points
    interX = np.intersect1d(ranx1Part, ranx2Part)
    interY = np.intersect1d(rany1Part, rany2Part)
    #find area of square common
    interX = list(interX)
    interY = list(interY)
    if (len(interX) != 0) and (len(interY) != 0):
        w = abs(interX[-1] - interX[0])
        h = abs(interY[-1] - interY[0])
```

```
            areaComm = (w + 1) * (h + 1)
            #find bigger area
            if areaPart > areaComp:
                bigger = areaPart
                smaller = areaComp
            else:
                bigger = areaComp
                smaller = areaPart
            rel = float(areaComm) / float(smaller)
            #print (areaComm, smaller, rel)
            if (rel > 0.3):
                if bigger == areaPart:
                    return comp
                else:
                    return part
            return []
        else:
            return []

    def particle_recognizer(image, mode, limit, **kwargs):
        imageProcess = cv2.imread(image)
        shape = imageProcess.shape
        imageH = cv2.resize(imageProcess, (int(shape[1] / 4), int(shape[0] / 4)),
                                                    interpolation=cv2.INTER_AREA)
        img = imageH.copy()
        imageH = cv2.cvtColor(imageH, cv2.COLOR_BGR2HSV)
        imageH = imageH[:,:,2]
        imageH = cv2.GaussianBlur(imageH, (7, 7), 5)
        if mode == "group":
            clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(4,4))
            imageH = clahe.apply(imageH)
        #list to add information of contour and list areas
        information = []
        areas = []
        #load nets
        ang_cascade = cv2.CascadeClassifier("Angular/data/cascade/cascade.xml")
        ast_cascade = cv2.CascadeClassifier("Astilloso/data/cascade/cascade.xml")
        blc_cascade = cv2.CascadeClassifier("Blocoso/data/cascade/cascade.xml")
        red_cascade = cv2.CascadeClassifier("Redondeado/data/cascade/cascade.xml")
        #image-scale factor-minimun number of neighborns
        if mode == "single":
            p_ang = ang_cascade.detectMultiScale(imageH, 1.9, 10)
            p_ast = ast_cascade.detectMultiScale(imageH, 1.9, 15)
            p_red = red_cascade.detectMultiScale(imageH, 1.9, 12)
            p_blc = blc_cascade.detectMultiScale(imageH, 1.9, 11)
        if mode == "group":
            p_ang = ang_cascade.detectMultiScale(imageH, 1.1, 1)
            p_ast = ast_cascade.detectMultiScale(imageH, 1.1, 1)
            p_red = red_cascade.detectMultiScale(imageH, 1.1, 1)
            p_blc = blc_cascade.detectMultiScale(imageH, 1.1, 1)
        #delete areas translapated
        particles = []
        for (x, y, w, h) in p_red:
            if (w * h) > limit:
                particles.append(["Redondeado", x, y, w, h])
            ##cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        for (x, y, w, h) in p_blc:
            if (w * h) > limit:
                particles.append(["Blocoso", x, y, w, h])
```

```
        ##cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)
    for (x, y, w, h) in p_ang:
        if (w * h) > limit:
            particles.append(["Angular", x, y, w, h])
        #cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
    for (x, y, w, h) in p_ast:
        if (w * h) > limit:
            particles.append(["Astilloso", x, y, w, h])
        #cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)
    #resize particles vector
    for part in particles:
        for comp in particles:
            #conditionals
            if part != comp:
                result = intercDealer(part, comp)
                try:
                    particles.remove(result)
                except:
                    pass
    for part in particles:
        if part[0] == "Angular":
            color = (255, 0, 0)
            information.append("Angular")
        if part[0] == "Astilloso":
            color = (0, 255, 0)
            information.append("Astilloso")
        if part[0] == "Blocoso":
            color = (0, 0, 255)
            information.append("Blocoso")
        if part[0] == "Redondeado":
            color = (0, 255, 255)
            information.append("Redondeado")
        #cv2.rectangle(img, (part[1], part[2]), (part[1] + part[3], part[2] +
                                                    #part[4]), color, 2)

    dictInfo = {'categories': information, 'areas': areas}
    #cv2.imshow("result", img)
    #cv2.waitKey(0)
    return dictInfo

if __name__ == "__main__":

    init = time()
    perfMon = threading.Thread(target=monitor)
    perfMon.start()
    part = listing("../Test/samples", "JPG")
    reg = open("results", "w")
    for p in part:
        dictInfo = particle_recognizer(p, "group", 2000)
        reg.write(json.dumps(dictInfo) + "\n")
    reg.close()
    end = time()
    mnt.put(False)
    print ((end - init) / 4.0)
#---------------- End Algorithm 17 ------------------
```

Algoritmo 18: *Algoritmo de estimación de la exactitud de un clasificador*

```
#---------------- Start Algorithm 18 ------------------
import json
```

```
from statistics import mean

#Load results of clasifier
results = open("results", "r+")
res = []
for ln in results:
    catInf = list(json.loads(ln)["categories"])
    totObj = len(catInf)
    blc = catInf.count("Blocoso")
    ang = catInf.count("Angular")
    ast = catInf.count("Astilloso")
    red = catInf.count("Redondeado")
    res.append([totObj, ang, ast, blc, red])
#Load real values of test
real = open("../Test/samples/realVals", "r")
n = 0
pTot, pAst, pAng, pBlc, pRed = [], [], [], [], []
for ln in real:
    vals = ln.split(" ")
    vals = [float(n) for n in vals]
    check = res[n]
    print (vals, check)
    if vals[0] == 0 and check[0] == 0:
        pTot.append(100)
    else:
        div = check[0]
        if vals[0] > check[0]:
            div = vals[0]
        pTot.append(100 - ((abs(vals[0] - check[0]) / div) * 100))
    if vals[1] == 0 and check[1] == 0:
        pAng.append(100)
    else:
        div = check[1]
        if vals[1] > check[1]:
            div = vals[1]
        pAng.append(100 - ((abs(vals[1] - check[1]) / div) * 100))
    if vals[2] == 0 and check[2] == 0:
        pAst.append(100)
    else:
        div = check[2]
        if vals[2] > check[2]:
            div = vals[2]
        pAst.append(100 - ((abs(vals[2] - check[2]) / div) * 100))
    if vals[3] == 0 and check[3] == 0:
        pBlc.append(100)
    else:
        div = check[3]
        if vals[3] > check[3]:
            div = vals[3]
        pBlc.append(100 - ((abs(vals[3] - check[3]) / div) * 100))
    if vals[4] == 0 and check[4] == 0:
        pRed.append(100)
    else:
        div = check[4]
        if vals[4] > check[4]:
            div = vals[0]
        pRed.append(100 - ((abs(vals[4] - check[4]) / div) * 100))
    n += 1
mTot = mean(pTot)
```

```
mAng = mean(pAng)
mAst = mean(pAst)
mBlc = mean(pBlc)
mRed = mean(pRed)
print ("n Tot: ", mTot)
print ("Ang: ", mAng)
print ("Ast: ", mAst)
print ("Blc: ", mBlc)
print ("Red: ", mRed)
#---------------- End Algorithm 18 -----------------
```

Algoritmo 19: *Algoritmo de generación de diagrama de 5 puntas*

```
#---------------- Start Algorithm 19 -----------------
import cv2
import numpy as np
from math import sin, cos, pi
font = cv2.FONT_HERSHEY_SIMPLEX

def calcValues(request):
    norm = (request[2] - request[1][0]) / (request[1][1] - request[1][0])
    return norm

def rotateName(name, value):
    """create a image with two words and rotate it, also return that image"""
    #create image
    size = 1.4
    width = int(len(name) * 20 * size)
    wd = int(len(value) * 20 * size)
    heigth = int(size * 30)
    hg = int(size * 1 * 30)
    mid = int(width / 2.0)
    image = np.ones((heigth + hg + 5, width, 3), np.uint8)
    image = 255 * image
    cv2.putText(image, name, (0, heigth - 3), font, size, (0, 0, 0), 2)
    cv2.putText(image, value,
                        (mid - int(wd / 2), heigth + hg),
                        font, size * 1, (0, 0, 0), 2)
    return image, heigth + hg + 5, width

def genGraph(Info, num, inverse):
    if inverse:
        pos = 2
        neg = 0
    else:
        pos = 0
        neg = 2
    tam = 1280
    cent = 640
    #draw background
    background = np.zeros((tam, tam, 3), np.uint8)
    background = 255 * background
    #draw forms
    rad = int(cent * 0.6)
    cv2.circle(background, (cent, cent), rad, (0, 0, 0), 2)
    #draw gradient
    for i in range(tam):
        i = i - cent
        for j in range(tam):
            j = j - cent
```

```
        if ((i ** 2) + (j ** 2)) < (rad ** 2):
            #calculate radius
            radius = ((i ** 2) + (j ** 2)) ** 0.5
            rel = float(radius) / rad
            background[i + cent, j + cent, pos] = 255 * rel
            background[i + cent, j + cent, neg] = 255 - (255 * rel)
distAng = (2 * pi) / 5.0
step = 0
caps = []
for n in range(5):
    #calculate point
    tol = calcValues(Info[n])
    x = cent + ((rad * tol) * sin(step))
    y = cent - ((rad * tol) * cos(step))
    caps.append([x, y])
    step = step + distAng
#draw mask
mask = np.ones((tam, tam, 3), np.uint8)
mask = 255 * mask
pts = np.array(caps, np.int32)
pts = pts.reshape((-1, 1, 2))
cv2.fillPoly(mask, [pts], (0, 0, 0))
mask = cv2.add(background, mask)
cv2.polylines(mask, [pts], True, (0, 0, 0), 2)
#draw limits
tn = int(rad / 3)
for n in range(5):
    x = int(cent + (rad * sin(step)))
    y = int(cent - (rad * cos(step)))
    #put names
    if n == 0:
        img, h, w = rotateName(Info[n][0][0], Info[n][0][1])
        mask[y - h: y, x: x + w, :] = img
    if n == 1:
        img, h, w = rotateName(Info[n][0][0], Info[n][0][1])
        mask[y - h - 15: y - 15, x - 15: x - 15 + w, :] = img
    if n == 2:
        img, h, w = rotateName(Info[n][0][0], Info[n][0][1])
        mask[y: y + h, x - 15: x - 15 + w, :] = img
    if n == 3:
        img, h, w = rotateName(Info[n][0][0], Info[n][0][1])
        mask[y: y + h, x - w + 15: x + 15, :] = img
    if n == 4:
        img, h, w = rotateName(Info[n][0][0], Info[n][0][1])
        mask[y - h - 15: y - 15, x - w + 35: x + 35, :] = img
    #draw equal lines
    stp = step + distAng
    x1 = cent + (rad * sin(stp))
    y1 = cent - (rad * cos(stp))
    cv2.line(mask, (int(x1), int(y1)), (int(x), int(y)), (0, 0, 0), 2)
    #another draws
    step = step + distAng
    cv2.line(mask, (cent, cent), (int(x), int(y)), (0, 0, 0), 2)
    #draw tiny lines
    xa = int(cent + (tn * sin(step - 0.1)))
    ya = int(cent - (tn * cos(step - 0.1)))
    xb = int(cent + (tn * sin(step + 0.1)))
    yb = int(cent - (tn * cos(step + 0.1)))
    cv2.line(mask, (xa, ya), (xb, yb), (0, 0, 0), 2)
```

```
            xa = int(cent + (2 * tn * sin(step - 0.05)))
            ya = int(cent - (2 * tn * cos(step - 0.05)))
            xb = int(cent + (2 * tn * sin(step + 0.05)))
            yb = int(cent - (2 * tn * cos(step + 0.05)))
            cv2.line(mask, (xa, ya), (xb, yb), (0, 0, 0), 2)
        #cv2.imshow("aaa", mask)
        #cv2.waitKey(0)
        cv2.imwrite("graph-" + num + ".png", mask)

if __name__ == "__main__":
    Info = [["precn", "100%"], [0, 1], 0.5],
            [["prc. Ang.", "100%"], [0, 1], 0.4],
            [["prc. Ast.", "100%"], [0, 1], 0],
            [["prc. Blc.", "100%"], [0, 1], 1],
            [["prc. Red.", "100%"], [0, 1], 0.3]]

    Info = sorted(Info, key=lambda d: d[2], reverse=True)
    genGraph(Info, "0", False)
#---------------- End Algorithm 19 ------------------
```