UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Clasificación de enfermedades de la Yuca subsahariana mediante Aprendizaje Automático

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Antonio Blasco Calafat

Tutor: Carlos David Martínez Hinarejos

Curso 2020-2021

Resumen

El objetivo principal del siguiente proyecto es la implementación de un algoritmo capaz de detectar enfermedades de la yuca subsahariana en imágenes de dicha planta mediante técnicas de aprendizaje automático.

Para ello se han estudiado diversas técnicas de *machine learning*, eligiendo Máquinas Vectores Soporte (SVM) con diversas técnicas de extracción de características y Redes Neuronales Convolucionales (CNN). Se han implementado los diferentes clasificadores, variando diferentes parámetros hasta encontrar aquél que se adapte mejor al problema siguiendo un análisis detallado de su error.

Posteriormente se ha implementado una aplicación con aquel método que obtiene la mejor clasificación, permitiendo al usuario de forma sencilla la utilización de la misma para obtener de forma rápida y eficaz la solución al problema.

Por último, se han estudiado conclusiones y posibles líneas a mejorar del proyecto, tanto del clasificador como de la aplicación desarrollada.

Palabras clave: Inteligencia artificial, *machine learning*, SVM, CNN, clasificadores.

Abstract

The main objective of the following project is the implementation of an algorithm capable of detecting sub-Saharan cassava diseases in images using machine learning techniques.

For this, various machine learning techniques have been studied, choosing Support Vector Machines (SVM) with various feature extraction techniques and Convolutional Neural Networks (CNN). The different classifiers have been implemented, varying different parameters until finding the one that best suits the problem, followed by a detailed analysis of its error.

Subsequently, an application has been implemented with the method that obtains the best classification, allowing the user to easily use it to quickly and efficiently obtain the solution to the problem.

Finally, conclusions and possible lines to improve the project have been studied, both in the classifier and in the application developed.

Keywords : artificial intelligence, machine learning, SVM, CNN, classifiers .

Resum

L'objectiu principal del següent projecte és la implementació d'un algorisme capaç de detectar malalties de la iuca subsahariana en imatges mitjançant tècniques d'aprenentatge automàtic.

Per a això s'han estudiat diverses tècniques de *machine learning*, triant Màquines Vectors Suport (SVM) amb diverses tècniques d'extracció de característiques i Xarxes Neuronals Convolucionals (CNN). S'han implementat els diferents classificadors, variant diferents paràmetres fins a trobar aquell que s'adapte millor al problema seguint una anàlisi detallada del seu error.

Posteriorment s'ha implementat una aplicació amb aquell mètode que obté la millor classificació, permetent l'usuari de forma senzilla la utilització de la mateixa per obtenir de forma ràpida i eficaç la solució al problema.

Finalment, s'han estudiat conclusions i possibles línies a millorar del projecte, tant del classificador com de l'aplicació desenvolupada.

Paraules clau: intel·ligència artificial, *machine learning*, SVM, CNN, classificadors.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Motivación

Durante miles de años los humanos hemos intentado entender cómo funciona nuestra capacidad de pensar, es decir, nuestra forma de entender, predecir y manipular con un simple puñado de materia. El campo que se trata en este proyecto, la inteligencia artificial (IA), no solo intenta comprender, sino que también se esfuerza en construir entidades pensantes.

Gracias a la inteligencia artificial se han producido una gran serie de avances en nuestra sociedad, avances que anteriormente se pensaban imposibles, como predicciones de cáncer prematuras que permiten salvar vidas, predicciones de modelos que ayudan a la inversión económica de empresas [1], etc; en definitiva, la IA sintetiza y automatiza tareas intelectuales y es, por lo tanto, potencialmente relevante para cualquier ámbito de la actividad humana.

La idea de conocer y entender cómo funciona esta disciplina, que trata de crear sistemas artificiales capaces de comportamientos que, de ser realizados por seres humanos, se diría que requieren inteligencia, es altamente atractiva.

A su vez, la agricultura a través de la historia ha cumplido un rol muy importante en el desarrollo económico del mundo. Por ello, con el transcurrir del tiempo se han desarrollado infinidad de técnicas para mejorar el desarrollo agrícola permitiendo aumentar la productividad de los cultivos y así cumplir con la demanda propuesta. Este aumento de la demanda ha provocado la proliferación de enfermedades en los cultivos; esto hace necesario contar con expertos en el campo de la agricultura, que reconozcan los patrones de las enfermedades. Sin embargo, este método manual requiere de mucho tiempo para plantaciones de gran tamaño [2].

Es por ello que la detección de enfermedades puede ser más fácil con técnicas como el *machine learning* en comparación con métodos manuales; con el procesamiento de imágenes que detecten patrones de diferentes enfermedades se pueden generar modelos de detección, y con grandes muestras se hace más precisa la detección de enfermedades.

Es gratificante ver las ventajas que ofrece esta automatización y saber que con ellas podemos ofrecer a gente que necesita del uso de estas técnicas una mejora para su día a día. Es por ello que con el trabajo realizado se busca que pueda ser útil y accesible para estas personas; por ello se buscarán formas de que el usuario pueda

interactuar con lo propuesto en el trabajo utilizando tecnologías que se han cursado durante los años del grado, junto con nuevos conocimientos que se han ido adquiriendo para la realización del trabajo. El uso de estas nuevas tecnologías, que se usan actualmente a nivel global y que resultan cambiantes en periodos breves de tiempo, implican una gran flexibilidad de adaptación y de renovación, que producen a nivel personal una gran satisfacción y un afán de conocimiento al seguir aprendiendo nuevas herramientas que mejoren de forma eficaz los algoritmos.

Con todo lo comentado anteriormente se deja claro el porqué llevar a cabo este proyecto resulta de interés para un ingeniero, y sirve como punto de partida para la elaboración de los objetivos a continuación.

1.2 Objetivos

Para lograr el objetivo principal de la creación de una pequeña aplicación capaz de integrar un clasificador que detecte las distintas enfermedades de la yuca subsahariana para el conjunto de muestras dado aplicando técnicas de Inteligencia artificial y *machine learning*, es necesario subdividirlo en una serie de objetivos que se desarrollarán a lo largo del trabajo.

- Estudio previo sobre las principales técnicas de procesamiento de imágenes en *machine learning*

- Obtención de un conjunto de datos de imágenes de la yuca subsahariana para la construcción de un clasificador

- Entrenamiento de los clasificadores Máquinas Vectores Soporte (SVM) y Redes Neuronales Convolucionales (CNN), alternando los parámetros para obtener diversos clasificadores

- Comparativa de resultados entre los clasificadores, tanto entre ellos como individualmente, para la elección del clasificador que maximice los resultados

- Creación de una aplicación de uso simple integrando el clasificador elegido anteriormente

1.3 Impacto esperado

Con la implantación de la aplicación se espera que el usuario final tenga la capacidad, de forma rápida, eficaz y simple, de reconocer qué tipo de enfermedad tiene la planta en caso de tenerla, ya que permite al agricultor una fácil reacción y sobre todo prematura ante el problema, que puede beneficiarle directamente disminuyendo el riesgo de pérdidas agrícolas y por tanto económicas.

1.4 Metodología

Se ha realizado una cronología del proyecto indicando la planificación seguida. En el primer evento de la Figura 1.1 se observa el inicio del TFG, periodo en el que tutor y el autor comprobamos su viabilidad y posibles modificaciones o ampliaciones al problema original. Posteriormente comienza la fase de estudio de las librerías y tecnologías a usar en el proyecto, con su respectiva instalación en los dispositivos. En el tercer evento empieza la implementación de diferentes versiones del trabajo hasta obtener el resultado deseado; aquí se desarrolla la parte más relacionada con la inteligencia artificial, ya que solo se intentan obtener los clasificadores finales. En el cuarto evento, tras una entrega y revisión del profesor, se modifican todos aquellos aspectos mejorables. Finalmente, en el último evento se construye la aplicación que verá el usuario más relacionado con el software. Se debe remarcar que la realización de la memoria se ha realizado conjuntamente y durante todo el proceso de construcción del TFG.

Figura 1.1. Cronología empleada en el trabajo

1.5 Estructura del documento

Este trabajo consta de los siguientes capítulos:

Capítulo 1: Introducción. Es el presente capítulo; se plantea un objetivo final, además de introducir al lector en el ámbito de la inteligencia artificial y la detección de enfermedades mediante el computador.

Capítulo 2: Estado del arte. Se mostrarán diversos estudios aplicados a la prevención de enfermedades de plantas donde se hará uso de la tecnología utilizada en el trabajo.

Capítulo 3: Se explicarán teóricamente todos los conceptos que se han utilizado para el desarrollo del proyecto, permitiendo así una comprensión del porqué de los parámetros elegidos.

Capítulo 4: Diseño de la solución. Se explicará de forma detallada los pasos que se han seguido para la obtención de la solución final, sin entrar a valorar resultados. Este diseño se divide en tres etapas, extracción de características, clasificación y valoración de resultados.

Capítulo 5: Desarrollo de la solución propuesta. Se mostrarán los resultados obtenidos habiendo variado los parámetros correspondientes. Posteriormente se realizará una comparativa de resultados.

Capítulo 6: Implantación. En este capítulo se presenta la etapa de implantación de la solución. Se mostrará a través de la interfaz del programa creado las soluciones proporcionadas por el clasificador elegido.

Capítulo 7: Conclusiones. En este capítulo se presentarán las conclusiones obtenidas de este trabajo.

Capítulo 8: Relación del trabajo desarrollado con los estudios cursados. Se realizará una comparativa de lo aplicado en el trabajo junto con lo estudiado durante el grado.

Capítulo 9: Trabajos futuros. Se realizará una autocrítica sobre qué puntos flacos tiene la aplicación y qué mejoras se pueden implementar en la misma.

Capítulo 2

Estado del arte

En este capítulo se estudiará el estado actual de la detección automática desde imágenes de enfermedades en plantas, analizando diversos métodos que se utilizan hoy en día.

Diferenciaremos entre dos grupos: aquellos métodos que requieren una extracción de características de la imagen para conocer información de la misma, con su posterior clasificador, y el grupo de redes neuronales, que realizan ambas. Por tanto, revisaremos primero los métodos de extracción de características más usuales (Sección 2.1), que se aplicarán sobre los métodos de clasificación no neuronales descritos en la Sección 2.2, para finalmente revisar la aproximación neuronal en la Sección 2.3.

2.1 Procesamiento de imágenes y clasificación

2.1.1. Histograma de gradientes orientados (HOG)

HOG [13], es un extractor de características basado principalmente en la forma de la imagen mediante el uso de los gradientes. Es una de las técnicas más empleadas en la visión por computador y preproceso de imágenes para la detección de objetos. A continuación se presentan algunos trabajos que han utilizado esta técnica.

En el trabajo descrito en [3], la estrategia utilizada es transformar las imágenes de entrada a un mismo tamaño de imagen, para posteriormente aplicar el extractor de características HOG y tras ello aplicar tres extractores diferentes: *Hu moment* (extrae la forma de la hoja), *Haralick texture* (para extraer la textura) y finalmente un histograma de colores para extraer el color.

En [4] sigue una estrategia similar a la comentada en el anterior artículo, donde se realiza un redimensionamiento de la imagen para la aplicación del extractor HOG, con la diferencia de que se aplica una transformación a escala de grises de la imagen antes de ser tratada por el extractor, debido a que el color no tiene importancia en la clasificación.

2.1.2. Histograma de colores

El histograma de colores puede ser un buen descriptor de características en aquellas imágenes en las que la variación de los colores es significativa para diferenciarlas. Es por ello que la aplicación de los histogramas se encuentra usualmente en la detección de imágenes en plantas, ya que las diferentes tonalidades de la hoja puede variar dependiendo de la enfermedad. Encontramos varios artículos que hacen uso de la técnica.

Por ejemplo, en [5], para la elaboración de un clasificador de imágenes de frutas hace uso de un histograma de colores 3D RGB. Es un histograma que representa la distribución de intensidades en los píxeles que contiene una imagen. Para la diferenciación de la característica que se quiere extraer se hace un enmascaramiento de la imagen, permitiendo aislar el objeto que se busca en la imagen y mejorar la clasificación.

En cambio, en el trabajo presentado en [6] se busca la representación de la imagen a través del nivel de iluminación del mismo. Para ello, cada imagen del conjunto de datos se transforma de los tres canales principales RGB (*red*, *green*, *blue*) a HSV (*hue*, *saturation*, *value*).

2.2 Clasificadores

Tras la extracción de características con los descriptores de las imágenes se pasa a una posterior clasificación de las mismas.

En el artículo [3] se hace uso de una gran variedad de clasificadores (K vecinos más cercanos, SVM, *Boosted Tree* y *Bagged Tree*), comparándose para el caso de estudio específico los diferentes resultados, obteniendo finalmente que *Bagged Tree* se adapta mejor con la aplicación de las técnicas de preproceso comentadas anteriormente.

En el artículo [4] se hace uso de una SVM para la clasificación de los troncos, obteniendo resultados mayores al 76% de acierto. Aquí se estudia cómo afecta cuando se varía el tamaño de ventana, donde aumenta el porcentaje de acierto a medida que aumenta el tamaño.

En el artículo [5] se opta por la construcción de un clasificador supervisado como el *Random Forest,* en el que se dividen las imágenes con un porcentaje para entrenamiento y otro de test para el clasificador, obteniendo un porcentaje de acierto en la clasificación del 100%.

Finalmente, en el artículo [6] se hace uso de una gran variedad de clasificadores (K vecinos más cercanos, SVM, *Boosted Tree* y *Bagged Tree* como en el caso del artículo [3]) siguiendo la misma estrategia, viendo cuál es aquel que se adapta mejor al caso de estudio variando sus parámetros. Finalmente, concluyen que *Bagged Tree* es el clasificador que obtiene mejores resultados.

2.3 Métodos con Redes Neuronales Convolucionales

Las redes neuronales convolucionales (convolutional neural networks, CNN o ConvNets) son una herramienta reciente de la tecnología. Son un tipo de red neuronal que surge como variación del perceptrón multicapa. Tanto el extractor de características como el clasificador pueden ser entrenables mediante la red, que realiza las funciones de ambos ajustándose automáticamente para maximizar el resultado.

En el artículo [7] se parte de tres conjuntos de datos: el primero se trata de un *dataset* donde las imágenes son capturadas con unas condiciones óptimas, mismo fondo y solo una hoja situada. En el segundo *dataset* encontramos las imágenes donde se capturan desde el mismo órgano para todas, sin embargo, ya no se encuentra ese control del anterior conjunto de datos, al no tener el mismo fondo de imagen. Finalmente, en el tercer *dataset* las imágenes se encuentran sin ningún tipo de condición previa, por lo que pueden aparecer desde diferentes posiciones y sin el mismo fondo de imagen para todas.

En el entrenamiento del conjunto de imágenes se opta por escoger una red pre-entrenada, adaptando los pesos al problema; además, se hace uso de clasificadores diferentes a las redes neuronales comparándose posteriormente con las redes neuronales convolucionales, como se muestra en la Figura 2.1.

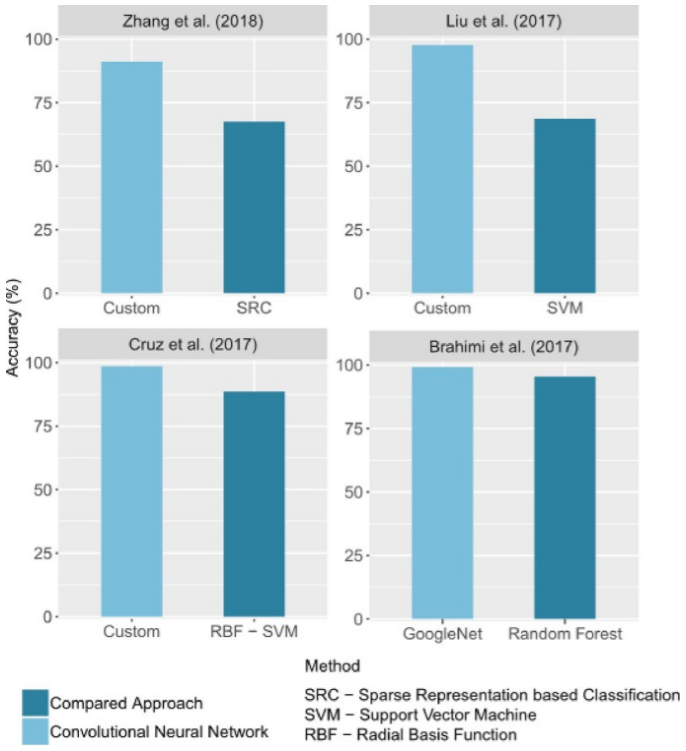

Figura 2.1. Comparación de CNN con otros clasificadores

Se observa cómo las CNN para el caso de estudio presentado obtienen mejores resultados de clasificación.

En el artículo [8] encontramos dos tipos de *datasets*, uno para el entrenamiento y otro para el test, donde el primero de ellos contiene una sola enfermedad por imagen tomada. En el primer *dataset* lo primero que realiza es clasificar el par enfermedad-cultivo y posteriormente clasificar la enfermedad independientemente del cultivo afectado. Para la realización del estudio se hace uso de diferentes redes ya pre-entrenadas, como ResNet 18 o ResNet 34, adaptando la red al caso de estudio en concreto para obtener el mejor resultado. Finalmente, se obtienen porcentajes de acierto superiores al 97%.

Capítulo 3

Fundamentos teóricos

3.1 Aprendizaje automático

El aprendizaje se refiere a un amplio aspecto de situaciones en las cuales el aprendiz incrementa su conocimiento o sus habilidades para cumplir una tarea. Para ello, aplica diferentes inferencias a una determinada información, permitiendo una representación apropiada de algún aspecto relevante.

Una metáfora habitual en el área del aprendizaje automático es considerar la resolución de problemas como un tipo de aprendizaje, que consiste en, una vez resuelto el problema, ser capaz de reconocer la situación problemática y reaccionar usando la estrategia aprendida.

Se pueden diferenciar principalmente dos fases en el área del aprendizaje automático. Por una parte, encontramos aquella en la que el sistema se encarga de seleccionar las características más relevantes de un objeto. La segunda fase, en cambio, trata de adaptar el modelo cuando las diferencias son significativas, según el resultado del cotejamiento.

Un sistema artificial puede emplear técnicas muy diversas para aprovechar la capacidad de cómputo; estas técnicas incluyen métodos matemáticos muy sofisticados, métodos de búsqueda, etc.[9], que requieren la creación de estructuras de representación del conocimiento para agilizar la identificación de hechos relevantes.

Hay distintas formas de diferenciar los diferentes sistemas de aprendizaje automático. Por una parte encontramos los paradigmas, como puede ser el aprendizaje deductivo, analítico, analógico, etc.[9] En el caso que nos ocupa nos centramos más en dividir el sistema mediante el tipo de estrategia que usan. En este caso tendremos:

Sistemas supervisados: La posición fundamental de este tipo de método es que los ejemplos proporcionados como entrada son necesarios para cumplir las metas del aprendizaje. Se dan ejemplos y se especifica de qué concepto lo son.

Sistemas no supervisados: Son diseñados para desarrollar nuevos conocimientos mediante el descubrimiento de regularidades en los datos. Estos métodos no están dirigidos por las metas.

3.2 Histograma de gradientes orientados

El histograma de gradientes orientados, HOG [13], es un extractor de características, es decir, se encarga de extraer la información útil de la imagen y desechar aquella que no lo es.

Para ello, se divide principalmente en tres pasos. El primero trata de separar la imagen del fondo, tal y como se muestra en la parte izquierda de la Figura 3.1. Este proceso consta de buscar las diferentes magnitudes de la imagen, tanto para el eje horizontal como para el vertical, aplicando para ello la siguiente ecuación:

$$m(u,v) = \sqrt{f_u(u,v)^2 + f_v(u,v)^2}$$

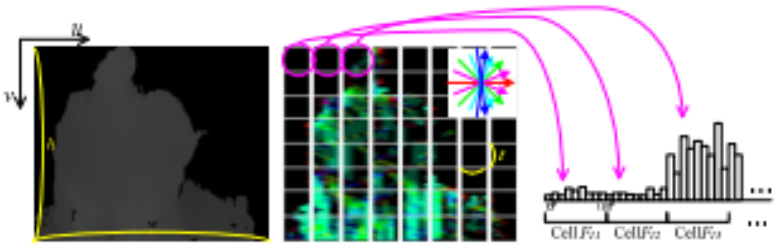

Figura 3.1. Pasos de la función HOG

En esta ecuación encontramos que la magnitud m en el punto (u,v) se obtiene como la raíz cuadrada de $f_u(u,v)$ y $f_v(u,v)$ elevados al cuadrado, donde $f_u(u,v)$ es la componente en la dirección de u (horizontal) y $f_v(u,v)$ es la componente en la dirección v (vertical).

El segundo paso consiste en, una vez extraída la diferencia entre el fondo de la imagen y el objeto, buscar la forma de determinar el objeto de la forma más precisa posible. Para ello se hace uso del vector director:

$$\Theta(u,v) = tan^{-1}\frac{f_v(u,v)}{f_u(u,v)}$$

Con la ayuda de la arcotangente para la obtención del ángulo, los vectores que haya en el mismo punto para la descripción de la imagen se diferenciarán con la ayuda

del vector director, como se puede observar en la imagen central de la Figura 3.1. El resultado de aplicar este proceso es difícil de interpretar; para ello se divide la imagen en una gran cantidad de píxeles y posteriormente se formará un histograma a partir de ellos, como se observa en la parte derecha de la Figura 3.1. Así se demuestra qué píxeles proporcionan una mayor diferenciación con el fondo de la imagen, permitiendo al programa discriminar correctamente entre ambos.

3.3 Histograma de colores

Como se ha comentado anteriormente en el caso de HOG, el histograma de colores también se va a utilizar como un extractor de características de la imagen.

Un histograma es una gráfica en donde se muestra la frecuencia con la que aparecen los distintos niveles de intensidad de una imagen. En nuestro caso específico el color, ya que según la enfermedad se observan diferentes patrones de color, por lo que se muestran los tres canales principales RGB (*red*, *green* y *blue*) donde el nivel de intensidad de cada píxel está en un rango de 0 a 255, en donde 0 es el valor negro y 255 el máximo para el canal, tal y como se muestra en la Figura 3.2. En el proceso de obtención se recorre la imagen para cada canal y se realiza el histograma correspondiente para el mismo, permitiendo así con esta información observar qué color es el predominante en la imagen y ayudando al clasificador a distinguir entre diferentes clases.

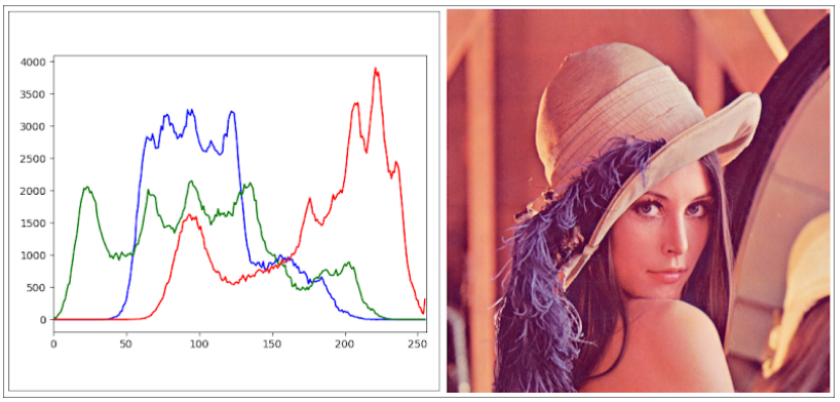

Figura 3.2. Histograma de colores a partir de una imagen

3.4 Máquinas Vectores Soporte (SVM)

El método de clasificación SVM fue desarrollado en la década de los 90. Si bien se desarrolló como un método de clasificación binaria, su aplicación se ha extendido a problemas de clasificación múltiple y regresión [11].

Las máquinas vectores soporte se fundamentan en el *Maximal Margin Classifier,* que a su vez se basa en el concepto de hiperplano.

La definición matemática del mismo es simple. En el caso bidimensional, el hiperplano se describe acorde a la ecuación de una recta:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

Dados los parámetros $\beta_0, \beta_1, \beta_2$todos los pares de valores $x(x_1, x_2)$ para los que se cumple la igualdad son puntos del hiperplano, obteniéndose un plano como el de la Figura 3.3. Esta ecuación puede generalizarse para p-dimensiones.

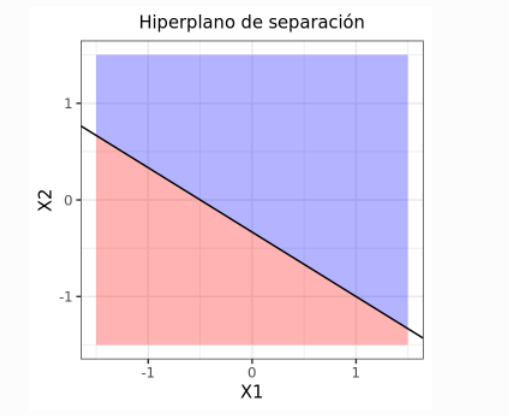

Figura 3.3. Separación de dos clases mediante un hiperplano

El cálculo del hiperplano para aquellas clases que son perfectamente separables linealmente resulta en un infinito número de casos. Por tanto, para simplificarlo a un único caso, se hace uso del *maximal margin hyperplane*, que se corresponde con aquél que se encuentra más alejado de todas las observaciones de entrenamiento. Para obtenerlo, se debe calcular la distancia perpendicular de cada observación a un determinado hiperplano. La menor de estas distancias (conocido como el margen) determina cómo de alejado está el hiperplano de las observaciones. Por tanto el *maximal margin hyperplane* se define como aquél que consigue un mayor margen, es decir, que la distancia mínima entre el hiperplano y las observaciones sea lo más grande posible. En la Figura 3.4 se muestra gráficamente cómo quedaría un plano cuando se le aplica el *maximal margin hyperplane;* se observa cómo el margen del plano se amplía hasta que éste pase por el primer punto.

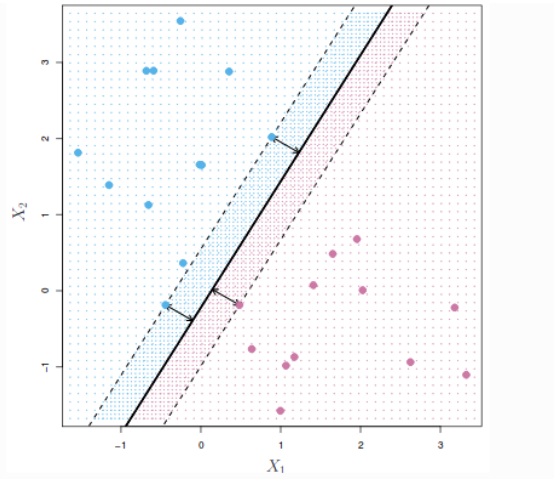

Figura 3.4. Hiperplano implementando mediante *maximal margin*

Se conoce por tanto como vectores soporte a aquellos vectores que se sitúan en un espacio p-dimensional y soportan, es decir, están a la mínima distancia posible, el *maximal margin hyperplane.* Sin embargo, en la práctica es difícil encontrar casos linealmente separables. Por ello se prefiere la implementación de un clasificador con una mayor capacidad predictiva al aplicarlo a nuevas muestras, evitando así problemas de sobreentrenamiento u *overfitting*. Esto se consigue a través de los conocidos como *soft margin classifiers*. Para lograrlo, en lugar de buscar el margen de clasificación más ancho posible, se permite que ciertas observaciones estén en el lado incorrecto del margen, tal y como se muestra en la Figura 3.5.

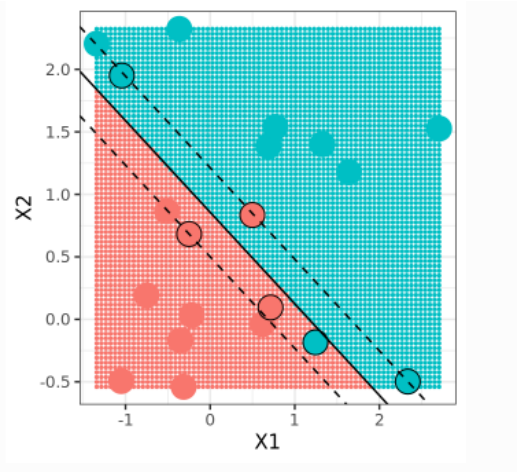

Figura 3.5. Hiperplano implementando mediante *soft margin*

Las SVM pretenden resolver el problema de optimización que observamos a continuación:

$$min\ \frac{1}{2}\theta^t\theta\ +\ C\sum_{n=1}^{N}\zeta_n$$

sujeto a las restricciones:

$$c_n(\theta^t x_n + \theta_0) >= 1 - \zeta_n, 1 <= n <= N$$

$$\zeta_n >= 0, 1 <= n <= N$$

Donde θ se trata de un vector de pesos, θ_0 del umbral, *C* de un entero, ζ_n error de margen y c_n la clase binaria correspondiente, es decir, +1 o -1.

De entre todos los parámetros descritos se destaca el parámetro *C*, ya que se trata de un parámetro de regularización, y controla el número y severidad de las violaciones del margen que se toleran en el proceso de ajuste. Si $C = \infty$, no se permite ninguna violación del margen, y por tanto es equivalente al *maximal margin classifier*. En cambio, cuanto más se aproxima a cero, menos se penalizan los errores y más observaciones pueden estar en el lado incorrecto del margen.

A diferencia de los s*oft margin classifiers* descritos anteriormente, SVM funciona principalmente si la separación de las clases es lineal. Si no lo es, su capacidad decae. La estrategia para enfrentarse a escenarios en los que la separación de los grupos no es lineal consiste en expandir las dimensiones, ya que dos clases que no son linealmente separables en dos dimensiones sí lo pueden llegar a ser en una dimensión mayor, como en el caso de la Figura 3.6.

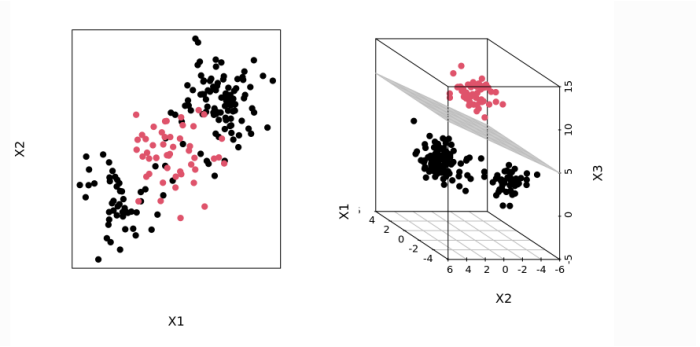

Figura 3.6. Casos separables linealmente aumentando su dimensión

Aquí es por tanto donde entran en juego los kernels [12]. Un kernel es una función que devuelve el resultado del producto escalar entre dos vectores realizado en un nuevo espacio dimensional distinto al espacio original en el que se encuentran los vectores. Para la implementación de este proyecto se hace uso de tres kernels:

Kernel lineal: es equivalente al *soft margin classifier.*

Kernel polinómico: k(*x*,*x'*) = $(x * x' + c)^d$, se generan límites de decisión no lineales; aumentando el valor de d se pueden llegar a generar problemas de *overfitting*. En la Figura 3.7 se muestra un ejemplo de un kernel polinómico de grado 3.

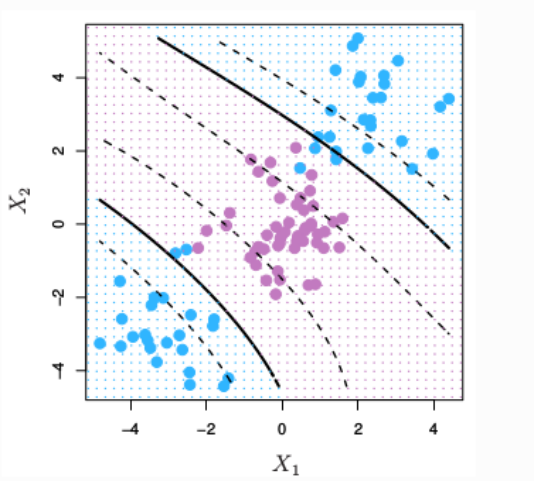

Figura 3.7. Ejemplo de kernel polinómico de grado 3

Kernel gaussiano: k(*x*,*x'*) = $exp(-\gamma||x - x'||^2)$, en este caso el valor de γ controla el comportamiento del kernel; cuando es pequeño el modelo final es equivalente a uno lineal, y a medida que aumenta su valor, también lo hace la flexibilidad del modelo. En la Figura 3.8 se muestra un ejemplo de un kernel gaussiano.

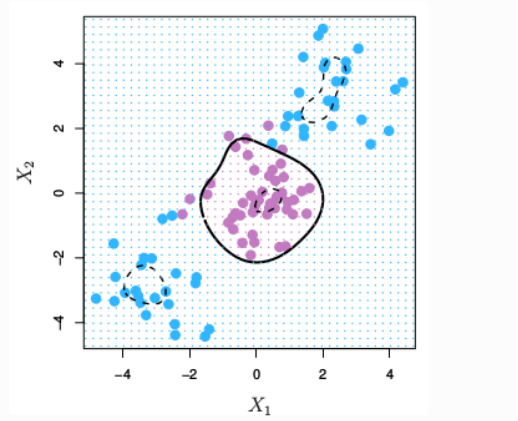

Figura 3.8. Ejemplo de kernel gaussiano

3.5 Redes neuronales

Las redes neuronales están basadas en el funcionamiento de las redes de neuronas biológicas. Las neuronas cerebrales están compuestas de dendritas, el soma y el axón. Las dendritas se encargan de captar los impulsos nerviosos que emiten otras neuronas. Estos impulsos se procesan en el soma y se transmiten a través del axón, que emite un impulso nervioso hacia las neuronas contiguas.

El modelo de una neurona está formado por:

• **Un conjunto de entradas** x_j y unos pesos sinápticos w_{ij}, con *i,j* = 1,...,*n*. Parte izquierda de la Figura 3.9.

• **Una regla de propagación** h_i definida a partir del conjunto de entradas y los

pesos sinápticos, es decir: $h_i(x_1,.... x_{n,} w_{i1},..., w_{in})$

La regla de propagación más comúnmente utilizada consiste en combinar linealmente las entradas y los pesos sinápticos, obteniéndose:

$$h_i(x_1,.... x_{n,} w_{i1},..., w_{in}) = \sum_{j=1}^{n} w_{ij} x_j$$

Suele ser habitual añadir al conjunto de pesos de la neurona un parámetro adicional, que se denomina umbral Θ_i en la Figura 3.9, el cual se acostumbra a restar al potencial post-sináptico. Es decir:

$$h_i(x_1,.... x_{n,} w_{i1},..., w_{in}) = \sum_{j=1}^{n} w_{ij} x_j - \Theta_i$$

Si hacemos que los índices *i* y *j* comiencen en 0, y denotamos por $w_{i0} = \Theta_i$

y $X_0 = -1$, podemos expresar la regla de propagación como:

$$h_i(x_1,.... x_{n,} w_{i1},..., w_{in}) = \sum_{j=0}^{n} w_{ij} x_j = \sum_{j=1}^{n} w_{ij} x_j - \Theta_i$$

- **Una función de activación**, la cual representa simultáneamente la salida de la neurona y su estado de activación. Si denotamos por y_i dicha función de activación, se tiene $y_i = f_i(h_i) = f_i(\sum_{j=0}^{n} w_{ij}x_j)$. Tienen como objetivo mantener los pesos de la neurona entre valores determinados, f( ) en la Figura 3.9. Suelen ser valores no lineales y encontramos diferentes tipos, entre ellas la sigmoide o logística, donde los pesos de salida toman valores entre 0 y 1, mientras que en ReLU o tangente hiperbólica los pesos positivos mantienen su valor y los negativos se ponen a 0 [10].

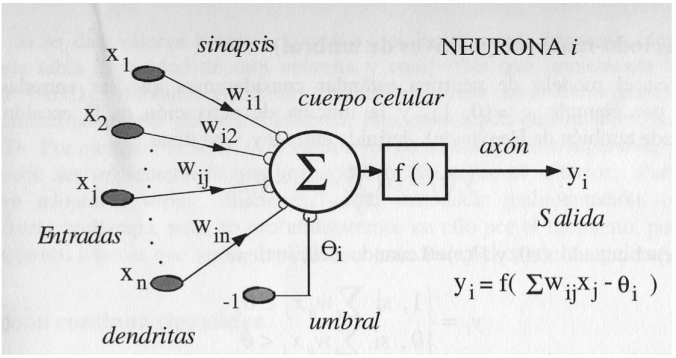

Figura 3.9. Esquema de una neurona

Una red neuronal está formada, por tanto, por diversas neuronas conectadas entre sí agrupadas mediante diferentes capas. Una capa es un conjunto de neuronas cuyas entradas provienen de una capa anterior y cuyas salidas son la entrada de una capa posterior, a las que se les asigna un peso. Cada neurona realiza el producto escalar de los pesos con las correspondientes activaciones de las neuronas de la capa anterior. Tras esto, se aplica una función de activación al resultado obtenido. Los pesos iniciales se iniciarán de forma aleatoria para no influir en el proceso de entrenamiento. En la Figura 3.10 se muestra un esquema típico completo de una red neuronal con una capa de entrada, una capa oculta y una capa de salida, en este caso binaria al haber solo una salida.

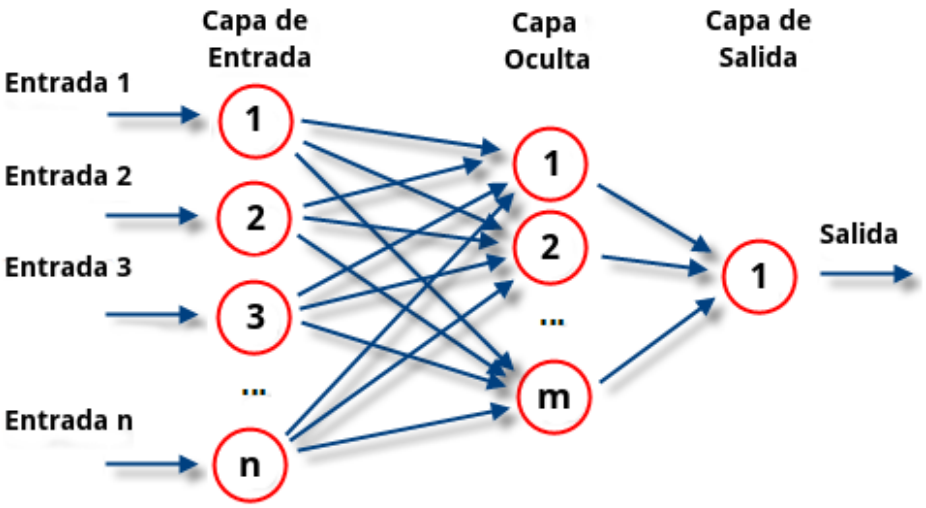

Figura 3.10. Ejemplo de una red neuronal

3.5.1 Redes convolucionales (CNN)

Las redes neuronales convolucionales son las elegidas para la implementación en el trabajo, debido a que son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes [14].

Podemos distinguir diferentes capas dentro de las redes neuronales convolucionales:

La capa de entrada en una red CNN está formada por los propios píxeles de la imagen de entrada; cabe añadir que el color de la misma también importa, ya que si encontramos una imagen a color se deben añadir los respectivos canales por píxel.

Capas convolucionales, que consisten en la aplicación de diversos kernels; se trata de unas matrices de tamaño menor al de la imagen (segunda imagen de la Figura 3.11), donde se realiza el producto escalar de los grupos de píxeles de la imagen por el kernel correspondiente, obteniendo una submatriz de salida como el caso de la tercera imagen de la Figura 3.11; a esta submatriz de salida se le aplicará una función de activación, típicamente ReLU.

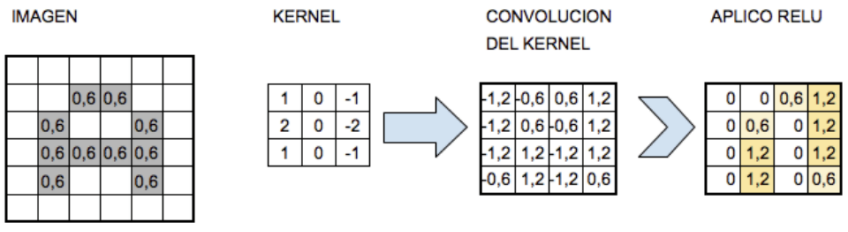

Figura 3.11. Esquema básico de una capa convolucional

Capas de muestreo o reducción, que se aplican posteriormente a la capa convolucional con el objetivo de reducir la dimensionalidad de las capas convolucionales, extrayendo las partes más importantes. *Max-pooling*[1] es la técnica más utilizada; tal y como se observa en la Figura 3.13, consiste en extraer el máximo valor entre el tamaño de celda que se elija.

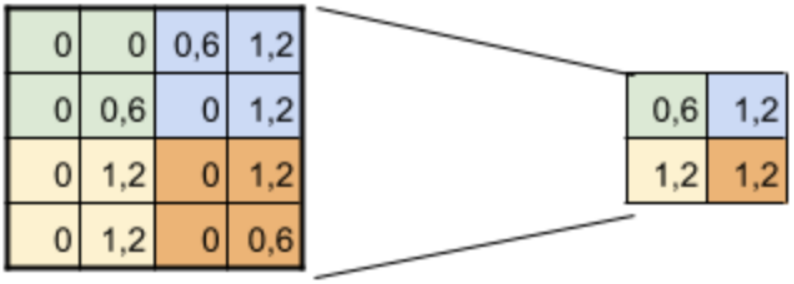

Figura 3.12. Ejemplo de *max-pooling* con tamaño celda 2x2

Capas *fully connected*, que se añaden a partir de la última capa oculta; se trata de poner en una sola dimensión la salida de la última capa oculta, formando una subred cuya capa de entrada capta todas las componentes resultados de la convolución y con una salida con el mismo número de neuronas que clases a predecir. Se le aplicará una función de activación *Softmax*, ya que se encarga de pasar una probabilidad entre 0 y 1 en las capas de salida.

Además de las diferentes capas, las redes neuronales convolucionales requieren de otros parámetros, entre los que podemos encontrar los siguientes:

- *Batch size*: define el número de muestras que se propagarán a través de la red. Por ejemplo, supongamos que se tienen 1050 muestras de entrenamiento y se configura un *batch size* igual a 100. El algoritmo toma las primeras 100 muestras (del 1 al 100) del conjunto de datos de entrenamiento y entrena la red. Luego, toma las segundas 100 muestras (de la 101 a la 200) y entrena la red nuevamente. Típicamente se toman potencias de dos como tamaños de *batch* .

- *Num epoch*: indica el número de pasadas a todo el conjunto de entrenamiento que ha completado el algoritmo de aprendizaje automático.

[1] https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

- Optimizadores: sirven para actualizar los pesos de la red y llegar al resultado más rápido.

- Técnicas de Regularización: se usan para evitar o reducir el *overfitting* o sobre ajuste.

- *Padding*: se refiere al número de píxeles de valor cero, que se le añaden a la imagen de salida para que tenga el mismo tamaño que la imagen de entrada, además, permite que información de las esquinas pase a estar en el centro, permitiendo así que aumente el nivel de importancia de las esquinas, ya que el kernel pasará más veces por valores céntricos.

- *Dropout*: esta técnica difiere de las vistas hasta el momento. Consiste en, por cada nueva entrada a la red en fase de entrenamiento, desactivar aleatoriamente un porcentaje de las neuronas en cada capa oculta, acorde a una probabilidad de descarte previamente definida. Dicha probabilidad puede ser igual para toda la red o distinta en cada capa. Lo que se consigue con esto es que ninguna neurona *memorice* parte de la entrada, que es precisamente lo que sucede cuando tenemos sobreajuste.

Capítulo 4

Diseño de la solución

Para llegar al objetivo final de la aplicación que se encarga de implementar el clasificador que indica la enfermedad de la planta es necesario realizar dos procesos principales. Primero, el diseño y entrenamiento de diferentes modelos, mediante redes neuronales y máquinas vectores soporte. Posteriormente, el desarrollo de una aplicación que se encargará de implementar este clasificador dando un resultado.

En este capítulo se detalla la estrategia utilizada, dejando los aspectos experimentales para el Capítulo 5.

4.1 Desarrollo del clasificador

En esta etapa se desarrollarán diversos clasificadores mediante las técnicas SVM y CNN, capaces de diferenciar entre las distintas enfermedades, o en caso de estar sana así indicarlo.

Se partirá de un conjunto de imágenes (*dataset*) con miles de imágenes de las plantas, y se entrenarán los clasificadores mencionados anteriormente. Posteriormente se evaluarán los resultados obtenidos con diferentes métricas, escogiendo finalmente aquél que maximice el resultado según los parámetros correspondientes.

4.1.1 *Dataset*

Un *dataset* no es más que un conjunto de datos en cualquier sistema de almacenamiento de datos estructurados. El término hace referencia a una única base de datos de origen**,** la cual se puede relacionar con otras. Cada columna del *dataset* representa una variable y cada fila corresponde a cualquier dato que estemos tratando.

Para este trabajo se parte de un *dataset* proporcionado por la web de Kaggle [2] en la que encontramos un conjunto de 21.397 imágenes, todas ellas a un tamaño de 800x600 píxeles y formato JPG, dividiéndose en cinco clases, cuatro de ellas de los diferentes tipos de enfermedades que encontramos en la yuca (clase 0: *Cassava Bacterial Blight*, clase 1: *Cassava Brown Streak Disease*, clase 2: *Cassava Green Mottle*, clase 3: *Cassava Mosaic Disease*) y la última en caso de que esté sana.

[2] https://www.kaggle.com/c/cassava-leaf-disease-classification/data

La distribución del *dataset* no es uniforme, debido a que se encuentra una gran disparidad entre imágenes para cada clase, siendo la 3 (*Cassava Mosaic Disease*) aquella que posee la mayor parte de ellas (13.158), mientras que las demás si están más equilibradas, con una media de unas 2.060 imágenes por clase. Es importante remarcar este desequilibrio entre las distintas clases, ya que afecta negativamente a la clasificación.

En la Figura 4.1 se observan diferentes tipos de imágenes que podemos encontrar. En estas imágenes, obtenidas usando un dispositivo móvil, se observa como las imágenes están hechas desde diferentes posiciones, estando todas a color, lo cual ayudará a la clasificación cuando hagamos uso del histograma de colores.

Figura 4.1. Imágenes del *dataset*

4.1.2 Extracción de características + SVM

Se partirá de la implementación de máquinas vectores soporte SVM para la clasificación de imágenes. En esta sección se explicarán los diferentes métodos de extracción de características implementados y el clasificador SVM.

4.1.2.1 Etapa extractora

Para esta etapa encontramos los dos tipos de extracción explicados anteriormente: HOG e histograma de colores.

Por una parte se implementará el extractor de características HOG; éste se implementa por medio de la librería *scikit-learn.* En la Figura 4.2 se muestra el resultado de aplicar el método HOG a una imagen extraída del *dataset*.

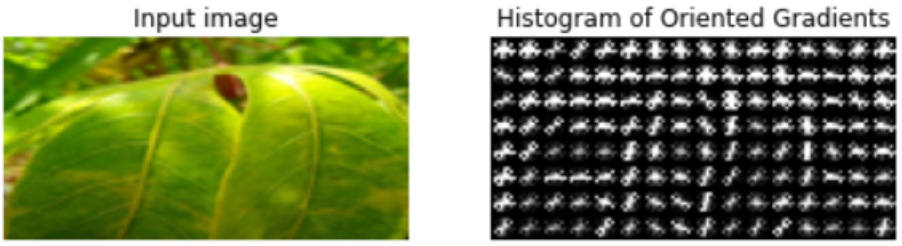

Figura 4.2. Ejemplo HOG

Por otra parte, se hará uso del histograma de colores para la implementación del segundo tipo de extracción. La implementación se realiza mediante la librería cv2[3] de python. En la Figura 3.2 se muestra el resultado de la extracción de colores RGB a través de una imagen.

4.1.2.2 Etapa clasificadora

Una vez extraídas las características de la imagen es turno de implementar un clasificador SVM que las posicione. El clasificador se implementa de nuevo con la librería *scikit-learn* de python.

El SVM se entrenará con tres kernels diferentes: lineal, gaussiano y polinómico. En el primer caso, el parámetro a variar será el parámetro *C*; en el segundo se variará tanto éste como el valor γ; finalmente, en el tercero se variarán los mismos parámetros que en el caso anterior.

Se dividirán las imágenes en una parte de entrenamiento (80%) y otra de test (20%) del total de 21397 imágenes.

Finalmente, los resultados que se obtengan de la ejecución de los siguientes clasificadores variando los correspondientes parámetros, se comparan entre ellos mismos, obteniendo aquel que maximice el resultado siguiendo las métricas de evaluación correspondientes para una posterior comparación con el modelo CNN.

4.1.3 CNN

Una vez realizados los diferentes modelos de máquinas vectores soporte, se ha optado por implementar dos redes neuronales en las que se sigue el mismo número de capas ocultas pero variando el número de conexiones entre ellas (siendo la segunda mucho más simple que la primera). Se ha mantenido para ellas el mismo tamaño de *max-pool*, así como el tamaño de *batch* y los *epochs.*

Se han dividido las imágenes, al igual que en SVM, en dos partes: una parte de entrenamiento y otra de test (17.118 imágenes de entrenamiento y 4.279 de test). La implementación de las redes se ha realizado con la librería de *tensorflow* y *keras.*

[3] https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html

4.1.4 Evaluación de resultados

Para la evaluación de los resultados se hace uso de la matriz de confusión, mostrada en la Figura 4.3. Esta matriz se obtendrá mediante la librería *sklearn-metrics.*

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 4.3. Matriz de confusión

A partir de la matriz de confusión se hará uso de *f1score* para obtener la calidad del clasificador; una vez calculado el *f1score* para cada etiqueta se obtendrán dos resultados con la ayuda de la librería *sklearn-metrics*, el *accuracy* y el *weighted-avg.* Se pretende maximizar los resultados, ya que a mayor porcentaje, mayor probabilidad de acierto tendrá el clasificador. El cómputo de las diversas medidas es el siguiente:

$$precision = \frac{TP}{TP+FP}$$

$$recall = \frac{TP}{TP+FN}$$

$$f1score = 2 \times \frac{precision \cdot recall}{precision^2+recall}$$

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Donde *TP* se refiere a verdaderos positivos, TN a verdaderos negativos, FN a falsos negativos, FP a falsos positivos.

Es importante remarcar que el parámetro de *accuracy* no es un buen medidor para aquellos modelos en los que la distribución de datos entre clases no es similar. Esto se debe a que la fórmula calcula la media de las clases, dándole el mismo peso a cada clase y, por tanto, para *datasets* en los que la distribución es desigual el valor del *accuracy* no representa la realidad. Por ello se ha optado por la métrica *weighted-avg,* equivalente al *f1score* ponderado, que explica el desequilibrio de clases calculando el promedio de métricas binarias en las que la puntuación de cada clase se pondera por su presencia en la muestra de datos reales.

En el caso de evaluación de las redes neuronales, además de la matriz de confusión comentada anteriormente, se tendrán en cuenta los valores de *accuracy* y de *loss;* una gran dispersión entre los mismos puede producir casos de *overfitting o underfitting.*

Finalmente, tras evaluar todos los datos siguiendo las métricas descritas, se compararán posteriormente SVM y CNN, obteniendo así aquel modelo que mejor se adapte al problema para implementarlo en la aplicación posteriormente.

4.2 Diseño de la aplicación

Después de conseguir el objetivo principal del trabajo, se pasa a implementar una pequeña aplicación para que se pueda ver de forma funcional el clasificador, permitiendo también al usuario que interactúe con la misma de forma simple.

Para ello se ha optado por desarrollar la aplicación en python mediante la librería de *tkinter*, librería que viene por defecto en la instalación de python. Se ha optado por la implementación de dos botones, cuyas funciones son buscar la imagen navegando a través del sistema de archivos del ordenador y un botón de clasificación que se encargue de llamar al modelo que se ha escogido para realizar la predicción de la imagen seleccionada. Finalmente, a través de un pequeño cuadro de texto se mostrará la enfermedad que posee la planta o, en caso contrario, que esté sana.

Capítulo 5

Desarrollo de la solución propuesta

En este capítulo del trabajo se presentan de forma analítica los métodos empleados: SVM y CNN. Primeramente se detallarán los parámetros usados en cada clasificación, que se irán variando constantemente para obtener diferentes soluciones. Posteriormente, se elegirá según las métricas comentadas en el Capítulo 4 el mejor valor obtenido para cada modelo. Finalmente, se realizará una comparativa entre los diferentes modelos, escogiendo así en base a los criterios correspondientes aquel que se acople mejor al objetivo final.

5.1 SVM

Para explicar los datos obtenidos en la construcción de las SVM, se han variado distintos parámetros dependiendo del tipo de kernel que se ha querido implementar, como se ha comentado anteriormente en el Capítulo 3. En el caso del kernel lineal solo se ha variado el parámetro *C,* ya que sólo depende de ese valor; en el caso del kernel gaussiano se ha optado por variar el parámetro γ; finalmente, en el kernel polinómico se han variado los mismos parámetros que en el caso anterior.

El objetivo de esta sección es buscar de forma práctica aquel parámetro que se adapte mejor al problema a resolver. Se ha de tener en cuenta, como se ha comentado en el Capítulo 3, problemas de *overfitting* que encontramos con valores altos de *C*, o para valores altos de γ*,* y problemas de *underfitting* con valores bajos de los parámetros descritos.

Por tanto, los valores elegidos para el parámetro *C* son los siguientes:

C = {0.0001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 50000}.

Mientras que para los valores para el valor γ son:

γ= {0.0000000001, 0.000000001, 0.0000001, 0.000001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000}

Finalmente, en el término independiente se ha optado por el valor por defecto de la función, es decir, 0.

5.1.1 Resultados aplicando método HOG

Primeramente se ha implementado mediante la librería *sklearn* el extractor de características HOG, obteniendo diferentes resultados variando el tipo de kernel.

5.1.1.1 Resultados del entrenamiento kernel lineal

A continuación, en la Tabla 5.1 se muestra para el kernel lineal los valores de *accuracy* y *weighted-avg* que se han obtenido para un subconjunto de los valores de *C* que se han propuesto anteriormente.

C	***accuracy***	***weighted-avg***
0.01	0.63	0.52
0.05	0.62	0.54
0.1	0.63	0.55
0.5	0.60	0.54
1	0.59	0.54

Tabla 5.1. Resultados kernel lineal utilizando HOG

El kernel lineal solo obtiene soluciones al problema para valores bajos del parámetro *C,* debido a que, como se ha comentado con anterioridad, a medida que se aumenta el valor *C* los márgenes se vuelven más restrictivos y, por tanto, hay menos tolerancia a fallos.

Para aquellos valores que el algoritmo ha encontrado solución se ha optado por mostrar, mediante la Figura 5.1, la cantidad de verdaderos positivos (*true positives*) que ha encontrado el algoritmo para cada valor del parámetro *C.* Se obtiene así una imagen clara de cuál es la calidad del clasificador.

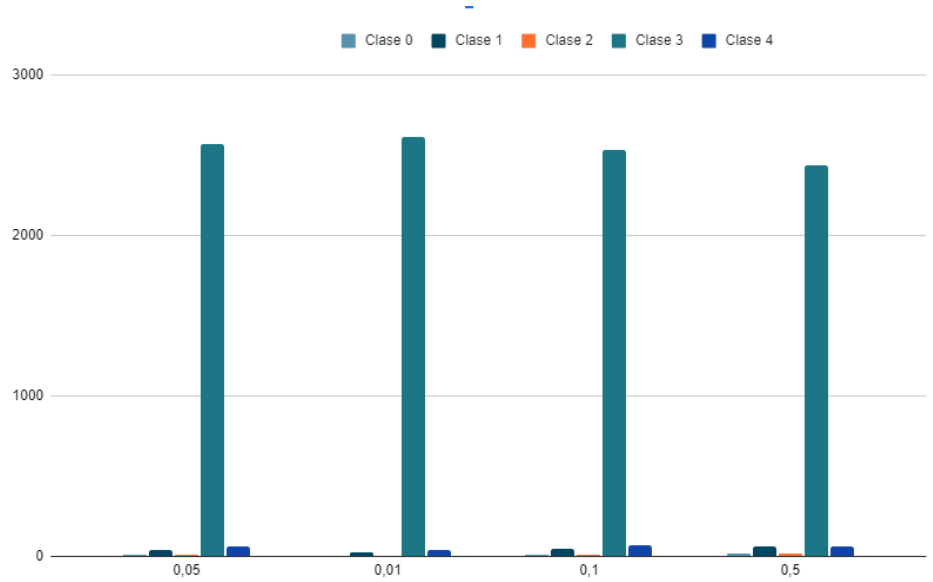

Figura 5.1. Gráfica de *true positives* kernel lineal HOG

En un primer vistazo a la Figura 5.1 podemos observar claramente como la clase 3 es aquella que posee la mayor cantidad de datos. Como se ha comentado en el Capítulo 4, la diferencia entre clases es notoria y se observa a la hora de la clasificación de las muestras en el test. Por tanto, como el valor de la clase 3 continuará siendo muy alto para el resto de clasificadores se ha optado por eliminarla de la gráfica, situando sólo el resto de clases y buscando aquella en la que se maximice el valor de *true positives* para cada clase restante.

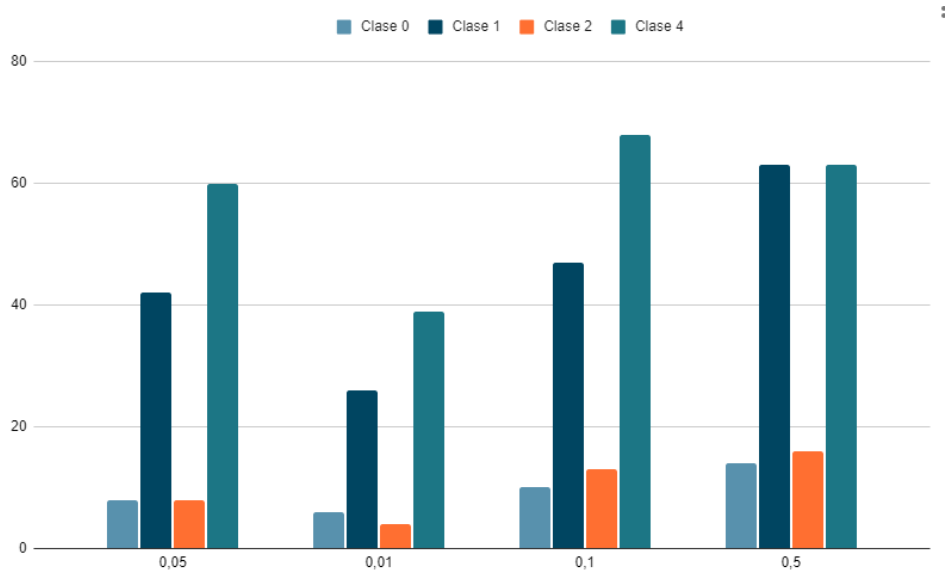

Figura 5.2. Gráfica de *true positives* sin clase 3 kernel lineal HOG

En la Figura 5.2 se observa claramente como para las clases 0 y 2 hay un reparto menor de imágenes. Teniendo en cuenta que hay 4.279 imágenes de test, puede verse que se clasifican menos de 20 para esas clases y por tanto se predecirá mal en la mayoría de casos.

5.1.1.2 Resultados del entrenamiento kernel gaussiano

El kernel gaussiano viene definido por los parámetros γ y *C*. Para la obtención de los resultados de este kernel se han ido variando los diferentes parámetros, teniendo como resultado que no hay forma posible de separar las clases. Se ha optado por tanto, por dejar el parámetro γ por defecto de la función SVM de la librería *sklearn*, donde ese γ se calcula de la siguiente forma: $\frac{1}{(n\_features * X\_var())}$, siendo $n\_features$ el número de características y $X\_var()$ la varianza del conjunto de entrenamiento.

De la experimentación se obtienen los resultados que se muestran en la Tabla 5.2.

C	***accuracy***	***weighted-avg***
1	0.62	0.49
5	0.63	0.55
10	0.63	0.55
50	0.63	0.55
100	0.63	0.55
500	0.63	0.55
1000	0.63	0.55
5000	0.63	0.55

Tabla 5.2. Resultados kernel gaussiano utilizando HOG

El kernel gaussiano, como se ha comentado en el Capítulo 3, viene dominado por el parámetro γ; como en nuestro caso no se ha podido variar porque no se han obtenido resultados, los valores de *accuracy* y *weighted-avg* apenas varían a medida que se aumenta el valor del parámetro *C*. A continuación en la Figura 5.3 se muestran los valores de *true positives* para las clases 0, 1, 2 y 4.

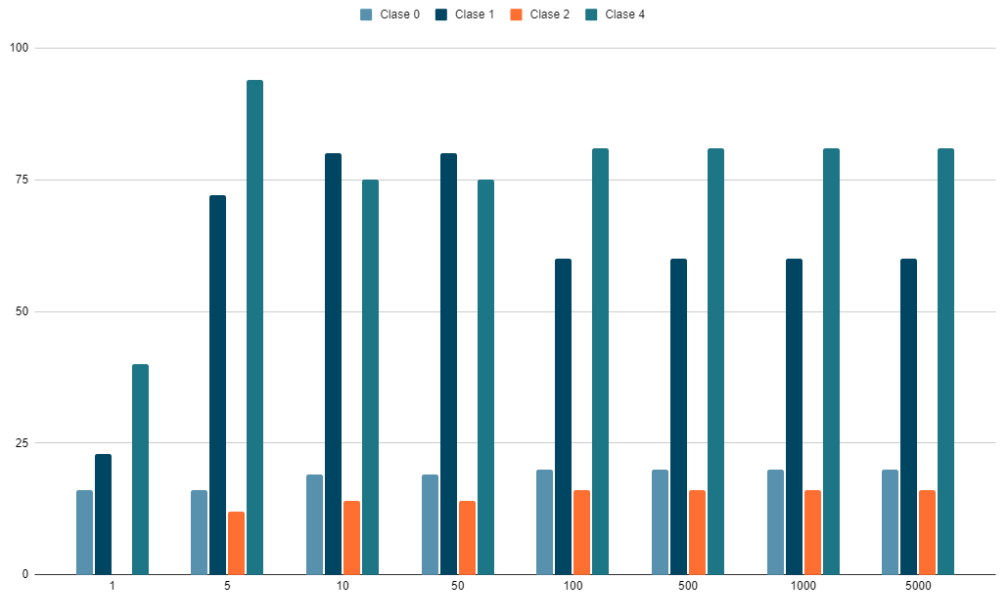

Figura 5.3. Gráfica de *true positives* sin clase 3 kernel gaussiano HOG

Respecto el gráfico de *true positives* para el kernel gaussiano, se observa cómo sigue la tendencia mostrada en la Tabla 5.2, donde se puede ver poca variabilidad de valores de acierto para cada clase.

5.1.1.3 Resultados del entrenamiento kernel polinomial

Finalmente, en el caso del kernel polinomial se ha encontrado el mismo caso que en kernel gaussiano: las variaciones del parámetro γ no han dado resultado, por lo que se ha optado por elegir el valor comentado en el caso anterior, obteniendo los resultados que se observan en la Tabla 5.3.

C	*accuracy*	*weighted-avg*
0.1	0.62	0.51
0.5	0.61	0.55
1	0.61	0.55
5	0.62	0.56
10	0.61	0.55
50	0.61	0.55
100	0.61	0.55
500	0.61	0.55
1000	0.61	0.55
5000	0.61	0.55

Tabla 5.3. Resultados kernel polinomial utilizando HOG

Observamos un caso parecido al anterior, y es la poca variabilidad encontrada en las métricas de evaluación. En la Figura 5.4 se observa la gráfica con los valores de acierto para cada clase para encontrar diferencias entre las diferentes ejecuciones.

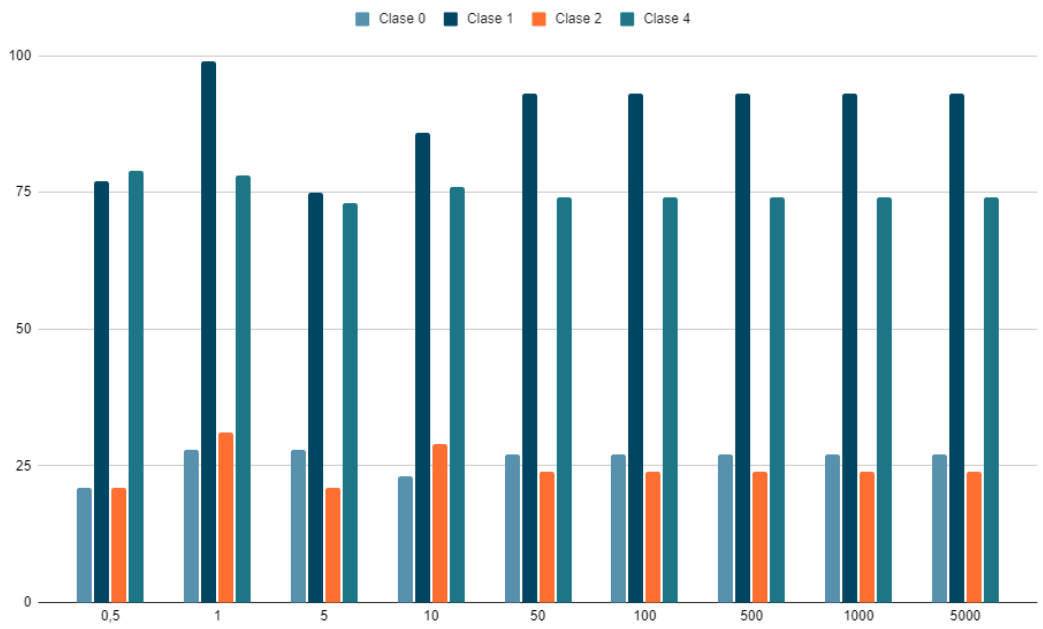

Figura 5.4. Gráfica de *true positives* sin clase 3 kernel polinómico HOG

Ocurre el mismo caso que en el kernel gaussiano: el algoritmo obtiene siempre los mismos resultados aunque se varíe el parámetro, y para la clases 0 y 2 se aciertan pocas muestras de test.

5.1.2 Resultados aplicando histograma de colores

A continuación se mostrarán los resultados obtenidos implementando el histograma de colores de la librería cv2 de python.

5.1.2.1 Resultados del entrenamiento kernel lineal

Siguiendo el mismo procedimiento que en el caso anterior se ha implementado el kernel lineal mediante la librería *sklearn,* obteniendo los resultados que se muestran en la Tabla 5.4.

C	***accuracy***	***weighted-avg***
0.1	0.64	0.51
0.5	0.61	0.48
1	0.62	0.55
5	0.62	0.56
10	0.62	0.55
50	0.62	0.55

Tabla 5.4. Resultado kernel lineal usando histograma de colores

En la Figura 5.5 se muestra la gráfica con los valores *true positive,* permitiendo ver cómo se comporta el clasificador.

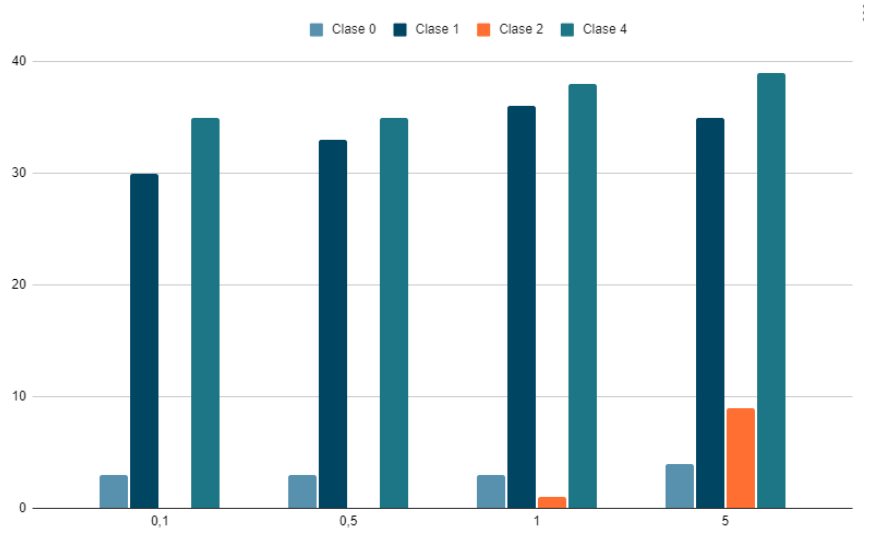

Figura 5.5. Gráfica de *true positives* sin clase 3 kernel lineal histograma de colores

Se puede observar como la implementación de este extractor de características no funciona bien para el modelo, debido a que la clase 2 no tiene valores de acierto para *C*= 0.1 y *C*= 0.5; además para valores como *C* = 1 y 5 no sobrepasa los 10 valores de acierto.

Es por ello que el clasificador no encuentra correctamente una forma lineal de separar el problema y produce demasiados errores por clase, lo que resulta aunque tenga valores de *weighted-avg* semejantes a los comentados en el extractor de característicos HOG, un problema, y por tanto no se puede utilizar como clasificador.

5.1.2.2 Resultados del entrenamiento kernel gaussiano

Para la implementación de este kernel se ha utilizado la misma metodología comentada en el caso del extractor HOG; se ha variado tanto el parámetro C como el γ; en el caso del segundo se vuelve a producir el mismo comportamiento que anteriormente. Por ello se ha optado por implementar la función por defecto obteniendo los resultados presentados en la Tabla 5.5.

C	*accuracy*	*weighted-avg*
0.5	0.63	0.50
1	0.64	0.52
5	0.64	0.54
10	0.64	0.54
50	0.66	0.58
100	0.64	0.58
500	0.64	0.58
1000	0.63	0.58
5000	0.62	0.59

Tabla 5.5. Resultado kernel gaussiano mediante histograma de colores

En este caso, a medida que se aumenta el valor del parámetro *C* se aumenta el valor de *weighted-avg*. En el problema que se está resolviendo no se produce un caso de *overfitting*, ya que no se encuentra una progresión lineal de porcentajes de predicción para las muestras, además de la difícil tarea de clasificación debido al mal reparto de imágenes por clase, que implica un grado de dificultad el entrenamiento.

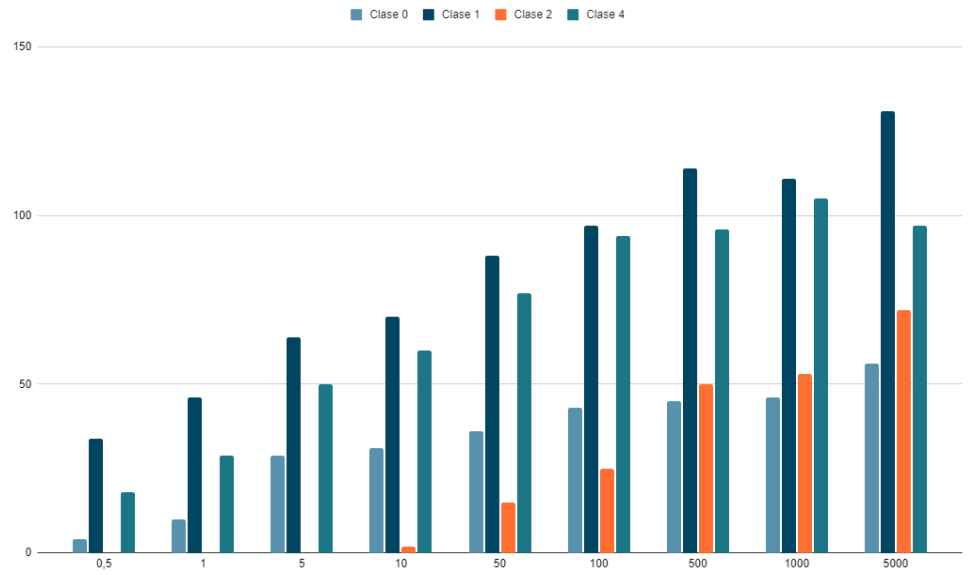

Figura 5.6. Gráfica de *true positives* sin clase 3 kernel gaussiano histograma de colores

En la Figura 5.6 se puede observar como a medida que se aumenta el parámetro *C*, también aumenta el porcentaje de muestras que se clasifican correctamente en la clase. Además, en las clases 0 y 2 se superan los 70 aciertos, mayor que cualquier otro clasificador que se ha probado anteriormente.

5.1.2.3 Resultados del entrenamiento kernel polinómico

Finalmente, y siguiendo la misma metodología que la implementada en el caso del extractor HOG, se ha implementado el kernel polinómico para el histograma de colores, obteniendo los resultados que se muestran en la Tabla 5.6.

C	***accuracy***	***weighted-avg***
0.5	0.63	0.51
1	0.63	0.52
5	0.63	0.53
10	0.65	0.56
50	0.67	0.57
100	0.65	0.57
500	0.65	0.58
1000	0.64	0.59
5000	0.64	0.59

Tabla 5.6. Resultado kernel polinómico usando histograma de colores

En este caso, a diferencia de los demás, el tiempo de ejecución sí es notorio, incrementándose cuanto más se aumenta el valor de *C*. Sigue la misma tendencia de valores de acierto que en el caso polinómico anterior; sin embargo, se puede observar en la Figura 5.7 que el número de datos que se predicen correctamente en la clase correspondiente aumenta a medida que aumenta el parámetro *C.* Por tanto, no se encuentran problemas de sobreentrenamiento en la aplicación de este algoritmo.

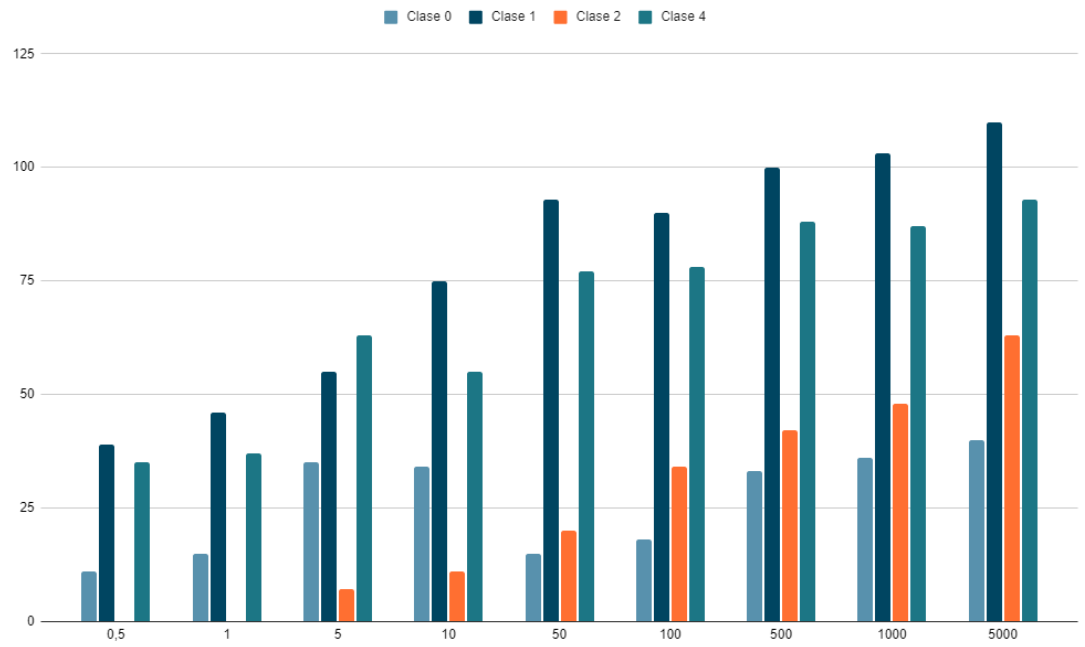

Figura 5.7. Gráfica de *true positives* sin clase 3 kernel polinómico histograma de colores

5.2 CNN

Realizados los diversos experimentos con las máquinas vectores soporte, se pasa ahora a implementar una red neuronal siguiendo la estrategia seguida descrita en el Capítulo 4.

Se mostrarán los diseños utilizados para la construcción de las CNN y se realizará un análisis para encontrar el modelo más efectivo. Posteriormente se comparará con las SVM, eligiendo así el más adecuado para su implementación.

5.2.1. Diseño de las redes

Se ha optado por diseñar dos redes siguiendo la misma estructura de capas entre las mismas, pero reduciendo el número de entradas por capa para una de las redes. Por ello se ha optado por 4 capas ocultas además de las capas de entrada y de salida.

Esto implica un cambio en el tiempo de ejecución por capa, ya que al tener menos conexiones (o al contrario, al tener más) el tiempo de ejecución disminuirá o aumentará.

5.2.1.1. Parámetros de las redes neuronales.

Los parámetros que se estudiarán para la obtención de los diferentes resultados y que se han implementado sobre el algoritmo son los siguientes:

- Función de activación: se hará uso de la función ReLU.

- Ancho de la imagen: para la anchura de la imagen se ha utilizado un tamaño de 256 píxeles en el diseño de la primera red y 300 para el diseño de la segunda.

- Alto de la imagen: en la altura de la imagen también se ha optado por un tamaño de 256 píxeles en el caso de la primera red y de 300 en el caso de la segunda.

- *Batch size*: en ambos casos se utiliza un tamaño de 64.

- *Num epoch*: el valor que se ha optado para el entrenamiento ha sido de 10 iteraciones.

- Número de mapas de características de la capa 1: es el número de *feature maps* que se ha utilizado en la primera capa; se ha diseñado la red para 64 *features* en el caso de la primera red y 16 en el caso de la segunda.

- Número de mapas de características de la capa 2: para la segunda capa se ha utilizado un valor de 128 *features* en la primera red y 32 para la segunda.

- Número de mapas de características de la capa 3: en la tercera capa se ha utilizado un valor de 128 *features* para la primera red y 64 para la segunda.

- Número de mapas de características de la capa 4: en la cuarta capa se ha vuelto a optar por un valor de 128 *features* en la primera red y 64 para la segunda.

- Número de mapas de características de la capa 5: finalmente en la última capa se ha optado por un valor de 256 *features* en la primera red y 64 en la segunda.

- Número de características de la capa *fully connected*: se ha diseñado la red con un valor de 1024 *features* en el caso de la primera red y 512 en el caso de la segunda.

- *Padding*: se hará uso del parámetro *same,* que permite que se añadan filas para que el filtro de convolución se aplique de igual manera a toda la imagen.

- *Dropout*: se usará un valor de 0.25 en el caso de la primera red neuronal, en la segunda no se hará uso de esta función.

- *Max pooling*: se implementará un tamaño de celda de 2x2 en ambos casos.

- Optimizador: se hará uso del optimizador *Adam.*

- Regularizador: se empleará la técnica de *early stopping* para reducir sobreajustes.

5.2.2. Resultados de entrenamiento

Primeramente se muestra mediante las Figuras 5.8 y 5.9 la comparativa del *accuracy* y del *loss* en el entrenamiento y en el test para las redes escogidas, permitiendo así saber si han habido problemas de *overfitting* o *underfitting.*

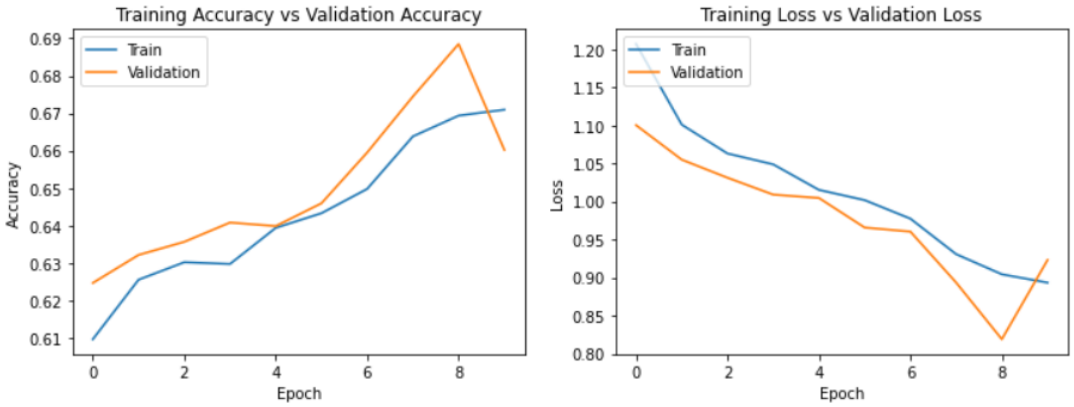

Figura 5.8. *Accuracy* y *loss* para la primera red neuronal

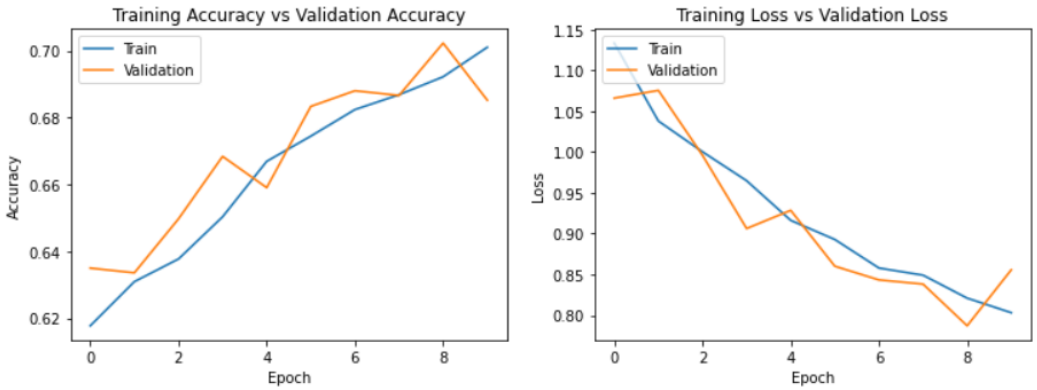

Figura 5.9. *Accuracy* y *loss* para la segunda red neuronal.

Como se observa, el valor de *loss* decrece a cada iteración y las gráficas de *accuracy* siguen una tendencia creciente hasta la iteración 8 en ambos casos, a partir de la 8 encontramos como decrece el valor del accuracy en la validación a cada valor de epoch, por lo que nos indica que se está produciendo un caso de *overfitting* a partir de esa iteración.

Posteriormente se ha obtenido en la Tabla 5.7 los valores de *weighted-avg* y *accuracy,* observando así cuál es la calidad del clasificador.

Red	***accuracy***	***weighted-avg***
1	0.41	0.41
2	0.43	0.42

Tabla 5.7. Resultado redes CNN

Finalmente, se ha obtenido el valor de *true positive*, observando en las Figura 5.11 y 5.12 el bajo valor de acierto que tienen las redes para clasificar las plantas en las diferentes clases.

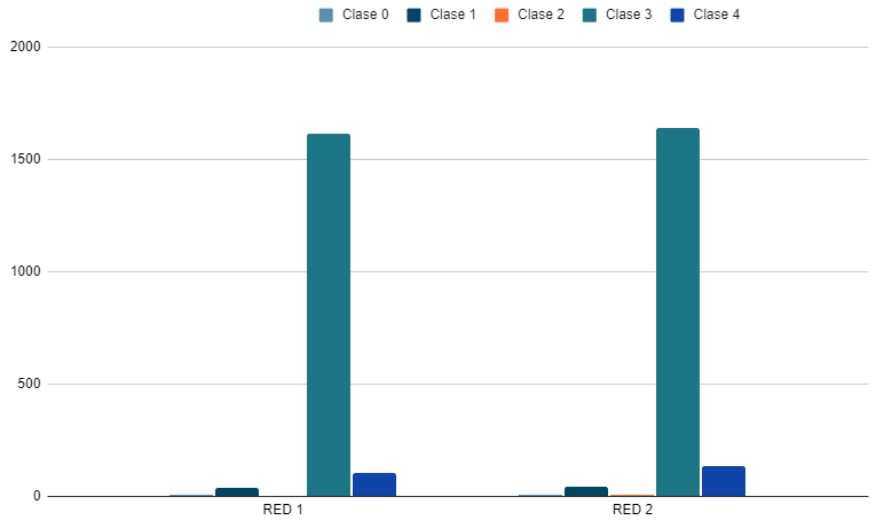

Figura 5.10 Gráfica *true positives* para las redes neuronales

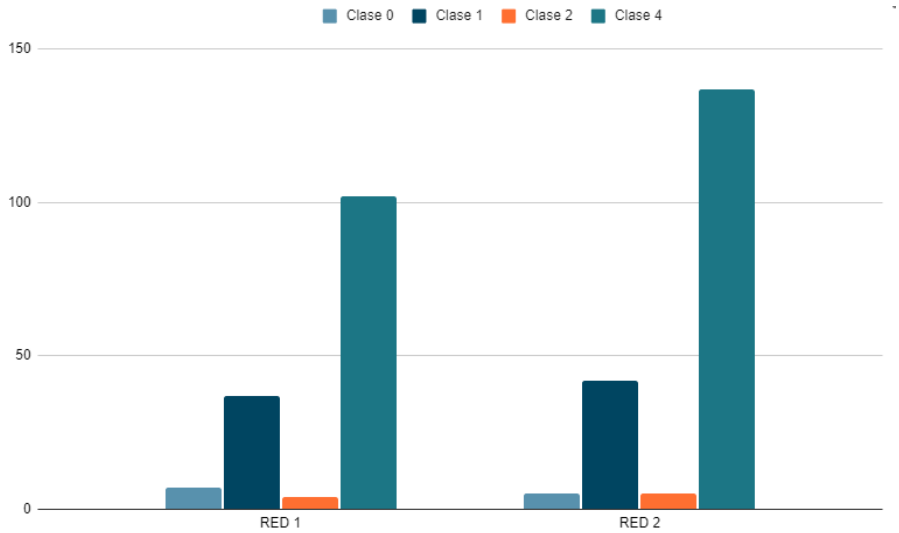

Figura 5.11 Gráfica *true positives* sin clase 3 para las redes neuronales

5.3 Valoración de resultados

A continuación, tras analizar los resultados de cada una de las implementaciones realizadas a lo largo de este capítulo, se pretende analizar detenidamente aquel

modelo que se adapte mejor al problema para poder implementarlo y obtener resultados fiables.

5.3.1 Resultados SVM

Siguiendo la métrica de evaluación que se ha comentado anteriormente en el Capítulo 4, la máquina vector soporte que mejor se ha adaptado al problema ha sido la que se ha implementado con el kernel gaussiano e histograma de colores, concretamente aquella con valor de C = 5000.

Se ha podido observar que, dependiendo del extractor de características utilizado, cada kernel se ha adaptado mejor o no al problema. En el caso del gaussiano, los valores para el extractor HOG realizan una mala clasificación; en cambio, se adapta muy bien al histograma de colores. En el caso del kernel lineal, se puede observar que el problema no es linealmente separable, ya que para aquellos casos que sí se ha producido una separabilidad encontramos muchos errores de clasificación y, por tanto, no es óptimo para ser implementado.

Finalmente, el kernel polinómico ha obtenido resultados semejantes al kernel gaussiano. Sin embargo, el principal inconveniente es el tiempo de cálculo, ya que aumenta considerablemente con el incremento del parámetro C.

5.3.2 Resultados CNN

En el caso de las redes neuronales su porcentaje de *f1score* es claramente menor al que se observa en las máquinas vectores soporte, incluso bajando del 50%, se debe a que para muestras de test nuevas al haber un caso de overfitting el clasificador no se adapta correctamente produciendo valores de clasificación bajos.

Por tanto, para el proceso de implementación se empleará la máquina vector soporte, aplicándole un kernel gaussiano y el histograma de colores como preproceso de imagen con C = 5000, ya que se obtiene un *weighted-avg* de 59% con valores de acierto por clase mayores a los modelos polinomiales, teniendo además un tiempo de ejecución menor a los mismos.

Capítulo 6

Implantación

Una vez elegido el clasificador con su extractor de características correspondiente, se procede a la implementación del mismo en una aplicación de carácter práctico, mostrando así de forma clara y sencilla el funcionamiento del clasificador.

Este capítulo se dividirá en dos apartados. Primero se mostrará el diseño de la interfaz, donde encontraremos ventana de inicio, ventana de selección de archivo y finalmente ventana donde se muestra la clasificación realizada. Posteriormente se mostrará de forma interna cuál ha sido el desarrollo de la aplicación.

6.1 Interfaz de la aplicación

La aplicación; como se ha comentado en el Capítulo 4, se ha realizado con la librería de python *tkinter;* es una librería gráfica que viene por defecto con la instalación de python para Windows.

Para la interfaz principal, como se observa en la Figura 6.1, se ha optado por la construcción de dos botones, uno para navegar entre el sistema de archivos con la finalidad de elegir la imagen que se pretende clasificar y otro para clasificarla en alguna de las enfermedades, o en ninguna de ellas en caso de estar sana, tal y como se ha ido comentando a lo largo del trabajo.

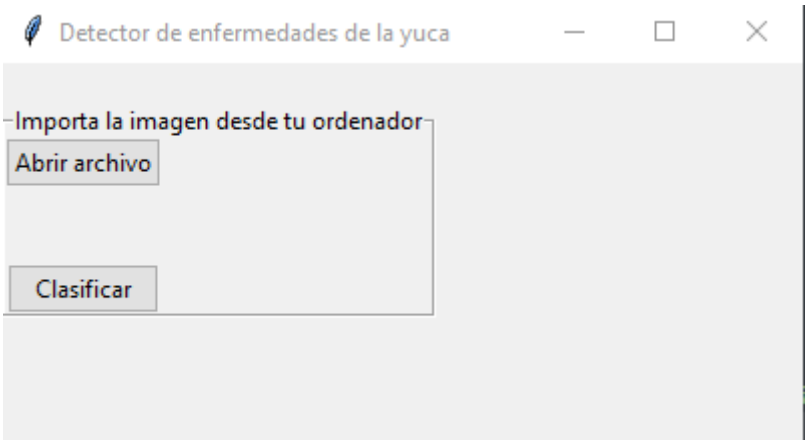

Figura 6.1. Pantalla principal de la aplicación

En la Figura 6.2 se observa cómo funciona el sistema de navegación de archivos; se ha configurado para que se elija imágenes .jpg como primera opción debido a que las imágenes de entrenamiento están en ese formato, pero en la pestaña desplegable,

como se observa, también encontramos otro tipo de formatos que son válidos para la clasificación.

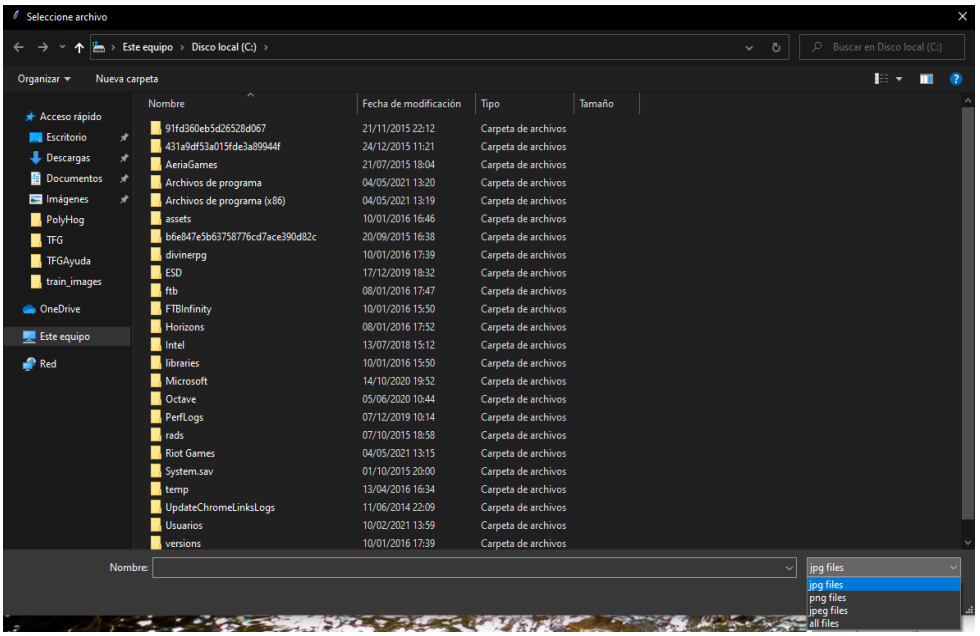

Figura 6.2. Selección de archivos de la aplicación

Posteriormente, una vez elegida la imagen ésta se mostrará en la aplicación con un tamaño reducido, permitiendo así ver de forma clara si el usuario ha elegido correctamente la imagen, como se puede ver en la Figura 6.3.

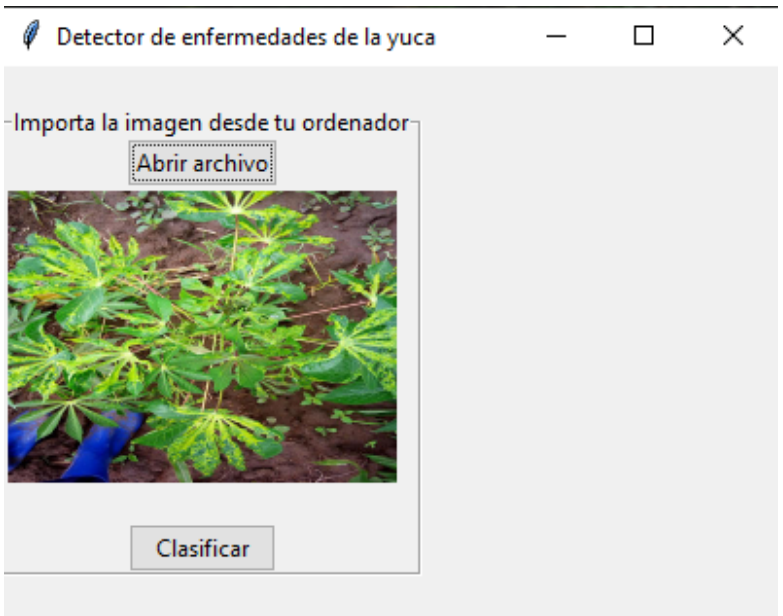

Figura 6.3. Pantalla con la imagen seleccionada

Finalmente, en la Figura 6.4 se muestra el resultado de la clasificación. En ella se ve como aparece un texto de color verde debajo de la imagen que nos indica una de las enfermedades o no, en caso de estar sana.

Figura 6.4. Pantalla con la clasificación de la imagen

6.2 Desarrollo interno de la aplicación

Para la navegación entre ficheros de la aplicación se ha optado por librería *filedialog* dentro de *tkinter,* que permite abrir y guardar archivos de forma sencilla. Como se observa en la Figura 6.5, vemos cómo se guarda en la variable *path_image* de la línea 63 la imagen seleccionada, para posteriormente a partir de la línea 71 leerla si se ha seleccionado una imagen y mostrarla en pantalla a través de un *label* como se observa en la líneas 81 y 82.

```
60
61      def abrir_archivo(self):
62          global path_image
63          path_image = filedialog.askopenfilename(initialdir = "/", title = "Seleccione archivo",
64                                                  filetypes = [("jpg files","*.jpg"),
65                                                               ("png files","*.png"),("jpeg files","*.jpeg"),
66                                                               ("all files","*.*")])
67
68          if len(path_image) > 0:
69              #global image
70              #Imagen para tratar en la red
71              image = cv2.imread(path_image)
72
73          #visualizar la imagen en el label
74          imageToShow = imutils.resize(image, width = 200)
75          imageToShow = cv2.cvtColor(imageToShow,cv2.COLOR_BGR2RGB)
76
77          im = Image.fromarray(imageToShow)
78          img = ImageTk.PhotoImage(image=im)
79
80          global lblInputImage
81          lblInputImage.configure(image=img)
82          lblInputImage.image=img
```

Figura 6.5. Código implementado para el primer botón

En el segundo botón se realiza la clasificación de la imagen, para ello como se observa en la Figura 6.6 en la línea 87 se abre el archivo que guarda los pesos de la clasificación realizada, posteriormente la imagen que se pretende clasificar se lee y se realiza la extracción de los colores con la llamada al método *quantify_image* de la línea 89, obteniendo así un array de características que se pasará al clasificador gracias al método predict que observamos en la línea 95, devolviendo en la variable status un valor que será la clase elegida por el clasificador, para posteriormente según el tipo de clase elegida variar el texto de salida que obtendrá el usuario.

```
85      def aceptar(self):
86          if len(path_image) > 0:
87              with open('modeloFinal','rb') as f:
88                  mp = joblib.load("modeloFinal.sav")
89              image = cv2.imread(path_image)
90              bins = (3,3,3)
91              features = self.quantify_image(image, bins)
92              aux = np.array(features)
93              aux = aux.reshape(1, -1)
94              status = mp.predict(aux)
95              print(status)
96              pos = status[0]
97
98              if pos == 0:
99                  texto = "La enfermedad de la planta es Cassava Bacterial Blight "
100             elif pos == 1:
101                 texto = "La enfermedad de la planta es Cassava Brown Streak Disease "
102             elif pos == 2:
103                 texto = "La enfermedad de la planta es Cassava Green Mottle "
104             elif pos == 3:
105                 texto = "La enfermedad de la planta es Cassava Mosaic Disease "
106             else:
107                 texto = "La planta no tiene ninguna enfermedad de las analizadas "
108
109         global lblSolucion
110         lblSolucion.configure(text = texto) #texto
111         lblSolucion.configure(foreground = "green", font = ("Time news Roman",16))
```

Figura 6.6. Código implementado para el segundo botón

Capítulo 7

Conclusiones

Tras el estudio de las técnicas de aprendizaje automático, la implementación del histograma de colores y HOG con las máquinas vectores soporte y la experimentación con las CNN, se puede afirmar que no se obtienen resultados óptimos para la detección de enfermedades de la yuca subsahariana.

Aunque en la implementación de las SVM se han obtenido valores similares y se ha elegido aquél que maximiza el valor, no se ha podido encontrar un modelo que sobrepase el 70% de acierto, mientras que en el estado del arte presentado en el Capítulo 2 se han encontrado valores superiores al 80%. Además, para la implementación de las CNN el valor de acierto es menor al 50%, por lo que resulta inviable y por tanto no apto para su implementación en una aplicación dirigida al usuario. Esto se debe principalmente al *dataset* utilizado para el entrenamiento, ya que la gran diferencia de imágenes entre clases dificulta el entrenamiento y lo hace menos preciso.

Por otro lado, se ha realizado satisfactoriamente la realización de la aplicación mostrando al usuario una interfaz simple y funcional, permitiendo el fácil uso de la misma junto con el mínimo tiempo de aprendizaje para dominarla.

Capítulo 8

Relación del trabajo desarrollado con los estudios cursados

Durante mi estancia en el grado he ido obteniendo una gran cantidad de destrezas que han permitido la realización de este proyecto.

8.1 Relación teórica

La relación teórica del proyecto se puede observar en el uso de las técnicas de reconocimiento automático empleadas, ya que ambas se estudian en la asignatura de Aprendizaje Automático cursada en el cuarto curso de la rama de computación. En ella se hace hincapié en cuál es la teoría detrás de la implementación de las funciones utilizadas.

8.2 Relación con las tecnologías empleadas en el grado

Respecto a las tecnologías empleadas, se hace uso de python para implementación de la aplicación y de las librerías para el uso de *machine learning*.

Aunque no se hayan utilizado estas herramientas específicamente en ninguna asignatura sí se han realizado diversas asignaturas que utilizan de estas técnicas y además han dado una base en otros lenguajes que son aplicables a cualquier otro.

SAR (Sistemas de almacenamiento y recuperación de la información), cursada en tercero en la rama optativa de Computación, da las bases para poder programar en python, ya que se hace una introducción de todas las funcionalidades del mismo.

Así mismo, en la asignatura de Aprendizaje automático (APR), aunque en la parte práctica no se utiliza python sí se hace uso de las tecnologías y parámetros que tienen las funciones, por lo que permite una rápida adaptación.

Finalmente, para la implementación de la aplicación, la asignatura de ISW (Ingeniería del software) permite tener las bases necesarias de cómo una aplicación debe funcionar. Por tanto, se ha realizado la aplicación siguiendo las bases de la asignatura.

8.3 Competencias transversales

En la realización de un proyecto se hace uso de diversas competencias transversales, se han destacado las más fundamentales.

Entre ellas encontramos **la planificación y gestión del tiempo** junto con **comunicación efectiva** como dos de las más importantes, debido a la repercusión que tiene sobre el trabajo. Es necesario saber qué tiempo se le debe repartir a cada tarea durante la realización de la misma, para así poder entregar en los plazos determinados aquello que se requiere. De igual manera, la forma de comunicar todo aquello que se quiere decir es vital en un trabajo de estas repercusiones, ya que se tiene que saber no tanto qué decir sino cómo decirlo para que se entienda de forma sencilla. Ya entrando en la parte más técnica del proyecto, el **análisis y resolución de problemas** ha sido vital debido a la importancia de obtener el mejor clasificador y, por tanto, se necesita de un gran análisis y **comprensión e integración** del problema.

Finalmente son destacables competencias como **aprendizaje permanente** y **pensamiento crítico**, ya que en la informática, y específicamente en el campo de la computación, las tecnologías evolucionan mucho y muy rápido, y es por tanto importante aprender constantemente y formarse en el campo para hacer uso de las tecnologías punteras y entender qué hay detrás de todas ellas.

Capítulo 9

Trabajos futuros

Para finalizar se exponen a continuación algunos trabajos que podrían mejorar la aplicación desarrollada.

Lo primero, cabe comentar que sería necesario trabajar más sobre el *dataset* proporcionado, mediante otras técnicas de extracción de características como la extracción de características mediante texturas, o bien la combinación de ambas, como HOG e histograma de colores. Otra forma de trabajar sobre el conjunto de datos es sobre los mismos datos en sí, ya sea tomando más fotos, para igualar el número de plantas por clase, o simplemente reducir de la que excede con mucha diferencia.

En cuanto al código se refiere, se puede mejorar de diversas formas, entre ellas mejorando el número de capas y conexiones que se ha implementado, buscando así una mayor efectividad para este caso. Otra opción de mejora sería implementar diversos clasificadores diferentes a los utilizados, como K vecinos más cercanos, entre otros.

Finalmente se podría mejorar tanto la interfaz como la funcionalidad de la aplicación, ya que se ha realizado una aplicación sencilla para que se pueda ver el uso de la aplicación y mostrar que el alumno domina diversos campos de la informática. Esto podría ayudar a la venta de la aplicación.

Por lo tanto, el objetivo principal que cabría perseguir en una futura mejora del proyecto sería trabajar sobre el conjunto de datos proporcionado y sobre la mejora de la interfaz, permitiendo así que la funcionalidad de la misma sea máxima.

Capítulo 10

Referencias

[1] Ramon López de Mántaras Badia ; Pedro Meseguer González. Inteligencia artificial . Editorial CSIC, 2017. ISBN: 978-84-00-10233-3.

[2] Andres Calderon, Harold David Hurtado Cortes, Machine learning in plant disease detection.(2019) TIA, 7 (2), pp. 55-61.

[3] Shima Ramesh Maniyath, "Plant Disease Detection Using Machine Learning," *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*, 2018, pp. 41-45, doi: 10.1109/ICDI3C.2018.00017.

[4] Guillermo Steinera, Alvaro Shapiro, Eduardo Destefanis. Implementación de un sistema de detección de troncos de árboles utilizando histogramas de gradientes orientados (HOG). Centro de Investigación en Informática para la Ingeniería (CIII) Universidad Tecnológica Nacional - Facultad Regional Córdoba. 40JAIIO, AST 2011, Página 228, ISSN: 1850-2806

[5] Diego Heras. Clasificador de imágenes de frutas basado en inteligencia artificial Fruit image classifier based on artificial intelligence. Revista Killkana Técnica. 2017, Volume 1, no 2. Página 21, doi: 10.26871/killkana_tecnica.v1i2.79

[6] Hussam Ali, Muhammad Ikram Ullah Lali, Nawaz, Muhammad Sharif, Saleem. Symptom based automated detection of citrus diseases using color histogram and textural descriptors, Computers and Electronics in Agriculture, Volume 138,2017, Pages 92-104, ISSN 0168-1699, https://doi.org/10.1016/j.compag.2017.04.008.

[7] Justine Boulent, Samuel Foucher, Jérôme Théau, Pierre-Luc St-Charles. Convolutional Neural Networks for the Automatic Identification of Plant Diseases, Frontiers in Plant Science, Volume 10, Pages 941, Year 2019, ISSN 1664-462X. https://doi.org/10.3389/fpls.2019.00941

[8] Ahmed Afifi, Abdulaziz Alhumam, Amira Abdelwahab. Convolutional Neural Network for Automatic Identification of Plant Diseases with Limited Data. Plants 2021, 10, 28, ISSN 2223-7747. https://dx.doi. org/10.3390/plants10010028

[9] Antonio Moreno, Eva Armengol, Javier Bejar, Luis Antonio Belanche, Claudio Ulises Cortes, Ricard Gavaldà, Juan Gimeno, Mario Martin, Miquel Sànchez-Marrè, Beatriz López, B. Aprendizaje automático. Edicions UPC. 1994.

[10] Pedro Larranaga, Iñaki Inza, Abdelmalik Moujahid. Tema 8. Redes neuronales. Departamento de ciencias de la computación e Inteligencia artificial. Universidad del País Vasco. 2021

[11] Alexandre Kowalczyk, Support Vector Machines Succinctly, October 23, 2017, Páginas 114, ISBN 978-1-64200-150-1

[12] Enrique Carmona. Tutorial sobre Máquinas de Vectores Soporte (SVM). Departamento de inteligencia artificial. Universidad Nacional de Educación a Distancia. (2016)

[13] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

[14] Wang, W., Yang, Y., Wang, X., Wang, W., & Li, J. (2019). Development of convolutional neural network and its application in image classification: a survey. Optical Engineering, 58(4), 040901.