



ugr | Universidad
de **Granada**

Extracción de características en imágenes

Práctica 1: Extracción de rasgos

Autor

Mónica Calzado Granados
monicacg1111@correo.ugr.es



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS
INFORMÁTICA Y DE TELECOMUNICACIÓN

Contents

1	Introducción	2
1.1	Camino elegido	2
1.2	Código fuente	2
2	Conjunto de datos	3
3	Clasificación usando HOG	5
3.1	Descriptor HOG	5
3.2	Lectura de imágenes	5
3.3	Entrenamiento del clasificador + Mejora 1	5
3.4	Mejora 2	8
3.5	Mejora 3	8
3.6	Ejemplos de clasificación	9
3.7	Conclusiones	11
4	Clasificación usando LBP básico	12
4.1	Descriptor LBP	12
4.2	Lectura de imágenes	12
4.3	Entrenamiento del clasificador	13
4.4	Comparación HOG+SVM y LBP+SVM	14
5	Clasificación usando LBP uniforme	15
5.1	Lectura de imágenes	15
5.2	Entrenamiento del clasificador	15
5.3	Comparación HOG+SVM, LBP+SVM, LBPU+SVM	16
6	Incluir una tercera clase	18
6.1	Clasificación con descriptor HOG	18
6.2	Clasificación con descriptor LBP	19
6.3	Clasificación con descriptor LBP uniforme	20
6.4	Comparación HOG+SVM, LBP+SVM, LBPU+SVM	21
7	Conclusiones generales	23

1 Introducción

1.1 Camino elegido

El camino que he elegido tomar es el siguiente: (1, 3, 5, 6, 7, 8).

1.2 Código fuente

En la carpeta entregada se encuentran varios ficheros .py cuya estructura es la siguiente:

- **funciones_HOG.py**: se incluyen varias funciones necesarias para leer las imágenes (tanto train como test) y calcular su descriptor HOG.
- **funciones_LBP.py**: se incluyen varias funciones necesarias para leer las imágenes (tanto train como test) y calcular su descriptor LBP.
- **funciones_LBPu.py**: se incluyen varias funciones necesarias para leer las imágenes (tanto train como test) y calcular su descriptor LBP uniforme.
- **funciones_aux.py**: contiene funciones para calcular las métricas obtenidas por el modelo y para realizar la búsqueda de hiperparámetros.
- **main.py**: es el programa principal, contiene un menú para elegir el modo de 2 clases o el de 3 clases (ejercicio 8). Llama a diversas funciones para calcular los 3 tipos de descriptores y compara el desempeño de las distintas combinaciones para el mejor clasificador SVM.

Inicialmente, ejecuté todo en un Jupyter Notebook por comodidad, por lo que también incluyó dicho archivo con el nombre **clasificador_svm.ipynb**. Este fichero contiene todo el proceso que fui siguiendo y las ejecuciones resultantes. Este archivo es más extenso, pues el **main.py** va directamente al grano. Se recomienda revisar el .ipynb en lugar de ejecutar el **main.py**, pues el código fuente tarda alrededor de 2 horas en ejecutarse.

2 Conjunto de datos

Para esta práctica, se ha seleccionado un conjunto de datos extraído de Kaggle, titulado *Plant Disease Recognition Dataset*, creado por Rashik Rahman Pritom. Este conjunto de datos se encuentra disponible en el siguiente enlace: <https://www.kaggle.com/datasets/rashikrahmanpritom/plant-disease-recognition-dataset>.

El propósito de este conjunto es proporcionar imágenes de hojas de plantas, clasificadas según el estado de salud de la planta y los tipos de enfermedades detectadas. Este dataset se puede utilizar en tareas de reconocimiento de enfermedades en plantas, para ayudar a los investigadores y profesionales a identificar problemas agrícolas.

Las clases disponibles son las siguientes:

- **Healthy:** Esta clase representa hojas de plantas saludables que no presentan signos visibles de enfermedad. Estas imágenes sirven como referencia para identificar el estado óptimo de las plantas.
- **Rust:** Representa hojas afectadas por la roya, una enfermedad causada por hongos que forma pústulas de color marrón o anaranjado en la superficie de las hojas. La roya es una enfermedad común que puede reducir significativamente el rendimiento de los cultivos.
- **Powdery:** Representa hojas afectadas por el mildiu polvoriento (*Powdery Mildew*), una enfermedad causada por hongos que se caracteriza por una capa blanca en la superficie de las hojas, afectando negativamente su crecimiento.



Healthy



Rust



Powdery

Figure 1: Ejemplos de cada clase disponible en el conjunto de datos.

No obstante, como se pide tomar 2 clases, hemos elegido las clases **Healthy** (clase 0) y **Rust** (clase 1).

Cada clase cuenta con imágenes separadas para el entrenamiento, prueba y validación, organizadas en carpetas. Aunque el conjunto completo unas 400 imágenes para cada una de las tres clases, en este trabajo, se han seleccionado 200 imágenes por clase para el entrenamiento y 50 imágenes por clase para la prueba (20% test y 80% train), con el fin de reducir los tiempos de procesamiento. Todas las imágenes han sido redimensionadas a un tamaño uniforme de 128x128 píxeles o 128x256 píxeles, dependiendo del descriptor utilizado (HOG o LBP), para garantizar consistencia en el preprocesamiento.

Este conjunto de datos ha sido elegido porque me ha parecido interesante su posible aplicación en problemas reales de clasificación multiclase en el ámbito agrícola. También por su accesibilidad pública, y su estructura clara (una carpeta para train y otra para test, con subcarpetas para cada clase).

Por último, echándole un primer vistazo al dataset, he observado que las distintas clases presentan diferencias significativas en términos de **textura**. Este aspecto es particularmente interesante, ya que, como se ha estudiado en clase, los descriptores HOG y LBP son herramientas muy útiles para capturar características relacionadas con texturas y patrones en imágenes. Por tanto, he tomado este dataset no solo por su aplicación práctica, sino porque también permite evaluar la eficacia de los descriptores estudiados en clase.

3 Clasificación usando HOG

3.1 Descriptor HOG

El descriptor *Histogram of Oriented Gradients* (HOG) es una técnica ampliamente utilizada en Visión por Computador para extraer características relevantes de imágenes. Este descriptor se basa en capturar patrones de textura y de forma, mediante el análisis de los gradientes locales de las intensidades de píxeles. HOG divide la imagen en celdas pequeñas, calcula el histograma de las orientaciones de los gradientes en cada celda, y posteriormente normaliza estos histogramas, para mejorar la invariancia a cambios de iluminación o contraste.

3.2 Lectura de imágenes

Hemos usado la función `HOGDescriptor` de la librería OpenCV para calcular las características de las imágenes. Nuestra función `load_training_data_HOG` sirve para leer y procesar las imágenes del dataset. Esta función parte de la proporcionada en Prado por el profesor, por lo que tienen mucha similitud.

La función comienza recorriendo los nombres de los archivos almacenados en carpetas separadas para cada clase, verificando que coincidan con el formato de las imágenes (.jpg). Cada imagen se redimensiona a un tamaño fijo de 128×256 píxeles, lo que nos asegura uniformidad en el cálculo del descriptor HOG. Posteriormente, se emplea la función `HOGDescriptor` para calcular el descriptor de cada imagen, que se almacena en una lista junto con la etiqueta correspondiente (1 para la clase positiva y 0 para la negativa). Hemos fijado el número de imágenes de entrenamiento a 200, para no sobrecargar el modelo.

Una característica destacable de nuestra implementación es que fijamos una semilla para garantizar la reproducibilidad entre distintas ejecuciones (que siempre obtengamos los mismos resultados si partimos de las mismas condiciones).

Finalmente, los descriptores y las etiquetas obtenidos se convierten en arrays de `numpy`, y se devuelven para su uso en las etapas posteriores de entrenamiento y evaluación del modelo.

Paralelamente, hemos definido una función similar llamada `load_testing_data_HOG`, que es idéntica a la anterior pero para leer los datos de test. Hemos fijado el número de imágenes test a 50.

3.3 Entrenamiento del clasificador + Mejora 1

Se han seleccionado dos clases del conjunto de datos: **Healthy** (hojas saludables) y **Rust** (hojas afectadas por roya). Las hojas dadas vienen representadas por la etiqueta 0 y las enfermas por la etiqueta 1.

Mediante las funciones `load_training_data_HOG()` y `load_testing_data_HOG()`, leemos y procesamos las imágenes, separando las características de las etiquetas. Posteriormente, se llama a la función `clase_imagenes(X_train, X_test, y_train, y_test)`, donde entrenamos un clasificador SVM llamando a la función `train(training_data, classes)`, utilizando un kernel lineal. Finalmente se mide la bondad del clasificador evaluándolo con el conjunto test. Hemos considerado un conjunto de medidas estándar, como el accuracy, *F1-score*, *recall* y la matriz de confusión.

En primer, la **matriz de confusión** nos proporcionará una visión general del rendimiento del modelo, al mostrar cuántas imágenes se han clasificado correctamente e incorrectamente en cada clase. La matriz se compone de las siguientes cuatro medidas, definidas en el contexto de nuestro problema de la siguiente forma:

- **TP (verdaderos positivos)**: el número de instancias Rust que han sido correctamente etiquetadas como Rust.
- **FP (falsos positivos)**: el número de instancias Healthy que han sido incorrectamente etiquetadas como Rust.
- **TN (verdaderos negativos)**: el número de instancias Healthy que han sido correctamente etiquetadas como Healthy.
- **FN (falsos negativos)**: el número de instancias Rust que han sido incorrectamente etiquetadas como Healthy.

		Clase real	
		Positiva	Negativa
Clase predicha	Positiva	Verdaderos Positivos (TP)	Falsos Positivos (FP)
	Negativa	Falsos Negativos (FN)	Verdaderos Negativos (TN)

Table 1: Matriz de confusión para un problema con dos clases

A partir de esta matriz, podemos calcular diversas métricas de evaluación del modelo:

- **Exactitud (accuracy)**: Es la proporción de instancias (tanto Rust como Healthy) que han sido etiquetadas correctamente.

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

- **Precisión (precision)**: Es la proporción de instancias Rust bien clasificadas entre todos los casos clasificados como positivos.

$$\text{Precisión} = \frac{TP}{TP + FP} \quad (2)$$

- **Sensibilidad (recall)**: También conocida como *Tasa de verdaderos positivos (TPR)*, es utilizada para conocer la proporción de instancias Rust que son etiquetadas correctamente.

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \quad (3)$$

- **F1-Score**: es una métrica muy utilizadas en problemas desbalanceados. Se construye a partir de la media armónica de la exactitud y la sensibilidad (recall).

$$F1\text{-score} = 2 \cdot \frac{\text{exactitud} \cdot \text{recall}}{\text{exactitud} + \text{recall}} \quad (4)$$

Los resultados obtenidos son los siguientes:

- **Imágenes correctamente clasificadas:** 72/100
- **Precisión total:** 72.00%

El informe detallado de clasificación proporciona las siguientes métricas:

Clase	Precision	Recall	F1-Score	Support
Negativo	0.73	0.70	0.71	50
Positivo	0.71	0.74	0.73	50
Accuracy	72.00%			
Macro Avg	0.72	0.72	0.72	100
Weighted Avg	0.72	0.72	0.72	100

La matriz de confusión obtenida se presenta en la Figura 2:

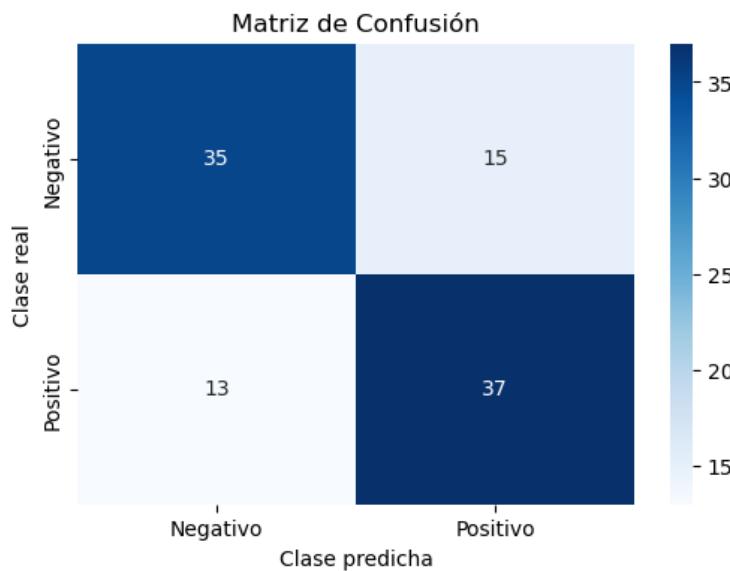


Figure 2: Matriz de confusión del clasificador HOG+SVM lineal.

Conclusiones: El clasificador ha alcanzado una precisión del 72%, con un *F1-Score* de 0.71 para la clase negativa y 0.73 para la clase positiva. A pesar de que ambas clases muestran un rendimiento equilibrado, se observan los siguientes pequeños detalles:

- La clase 0 (Healthy) tiene un recall ligeramente inferior (0.70), lo que indica que algunas imágenes negativas se clasificaron incorrectamente como positivas.
- La clase 1 (Rust) tiene una precision ligeramente menor (0.71), lo que sugiere que algunas predicciones positivas resultaron ser incorrectas.

En general, el modelo presenta un rendimiento no increíble pero aceptable, con un balance adecuado entre las dos clases. Sin embargo, podríamos reducir el error en ambas clases con ajustes adicionales, como la optimización de los parámetros del clasificador SVM, técnicas de validación cruzada, el uso de un conjunto de datos más amplio o el uso de otros descriptores.

3.4 Mejora 2

Como segunda mejora, se pide considerar distintos conjuntos de train y test. Para ello, hemos elegido **validación cruzada**. Consiste en realizar k particiones de igual tamaño del conjunto de datos, y a lo largo de k iteraciones se utilizan $k-1$ particiones para entrenar el clasificador, para luego validar con la partición sobrante. La precisión del clasificador se obtiene como la media de las k precisiones calculadas en cada iteración.

A modo de ejemplo, gracias a la función `StratifiedKFold` de `Scikit-learn`, hemos entrenado un clasificador 5 veces (`n_splits=5`) tomando distintas particiones del conjunto de entrenamiento. Este método tiene de particular que al dividir el conjunto de datos, mantiene la proporción de tamaños de cada clase. En la siguiente tabla vemos la precisión obtenida en cada pliegue:

Pliegue	Accuracy (%)
1	73.75
2	75.00
3	70.00
4	75.00
5	78.75

Table 2: Precisión obtenida en cada pliegue durante la validación cruzada.

Podemos observar que en cada pliegue se alcanza una precisión distinta, lo cual destaca la importancia de aplicar técnicas de validación para distintos conjuntos de entrenamiento.

3.5 Mejora 3

Como tercera mejora, se pide encontrar el mejor clasificador SVM, variando sus hiperparámetros. Para la **búsqueda de hiperparámetros** hemos utilizado la función `GridSearchCV` de `Scikit-learn`. Esta herramienta permite definir un grid o una cuadrícula de hiperparámetros que serán explorados exhaustivamente, realizando pruebas con todas las combinaciones posibles. Además, utiliza directamente validación cruzada para evaluar cada combinación, lo cual nos asegura que el rendimiento del modelo sea menos afectado por las peculiaridades que pueda tener una partición particular de los datos.

Hemos definido la cuadrícula de hiperparámetros de la siguiente forma:

- **C**: [0.1, 1, 10, 100], que controla la penalización de errores en el modelo.
- **kernel**: ['linear', 'rbf', 'poly'], que define la función del núcleo a utilizar (lineal, radial o polinómica).
- **gamma**: ['scale', 'auto'], que ajusta cómo se considera la influencia de cada punto de soporte en los modelos con kernel no lineal.

Se utilizó la métrica *F1-score* como métrica de evaluación principal, ya que ofrece un balance entre precisión y recall.

Después de ejecutar el proceso de búsqueda, los mejores hiperparámetros encontrados fueron los siguientes: `{'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}`. Dicho modelo ha alcanzado una puntuación *F1* de 0.7431 durante la validación cruzada.

Tomando dichos hiperparámetros y su modelo correspondiente, hemos procedido a evaluar su rendimiento para el conjunto de datos test. Se han obtenido los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
Negativo	0.75	0.72	0.73	50
Positivo	0.73	0.76	0.75	50
Accuracy	74.00%			
Macro Avg	0.74	0.74	0.74	100
Weighted Avg	0.74	0.74	0.74	100

Table 3: Informe detallado de clasificación del mejor modelo SVM+HOG.

La matriz de confusión obtenida se presenta en la Figura 4:

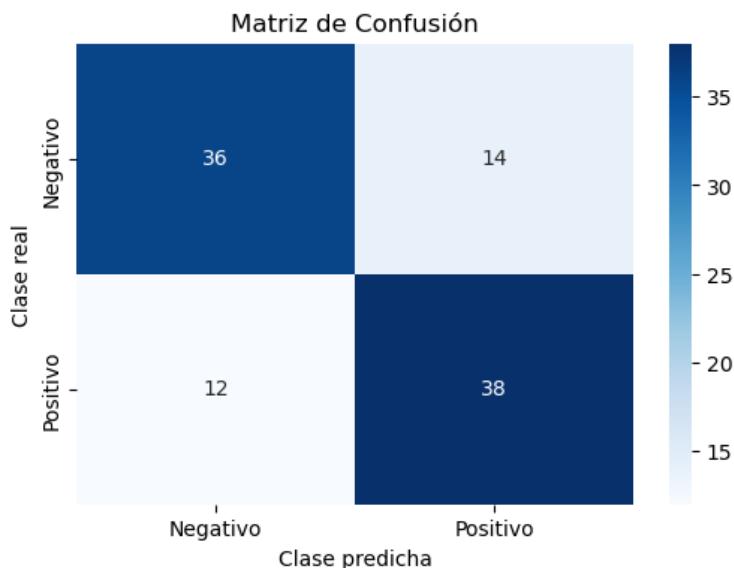


Figure 3: Matriz de confusión del mejor clasificador HOG+SVM.

El modelo muestra un rendimiento equilibrado entre ambas clases, alcanzando un *accuracy* del 74% y un *F1-score* de 0.75 para la clase positiva y 0.73 para la clase negativa. Aunque el rendimiento es aceptable, estos resultados podrían mejorarse aún más explorando otros descriptores como LBP en combinación con SVM.

3.6 Ejemplos de clasificación

Para visualizar los resultados de manera más clara, hemos tomado nuestro mejor modelo HOG y le hemos pasado 4 imágenes de test, para valorar de forma gráfica los resultados.

Las rutas de las imágenes elegidas son las siguientes, es decir, las dos primeras imágenes de tipo Healthy y las otras dos siguientes de tipo Rust:

```

1 image_paths = [
2     "data_plantas/Test/Test/Healthy/8ddd5ec1c0de38c4.jpg", #Healthy1

```

```

3 "data_plantas/Test/Test/Healthy/8def4d91382175c3.jpg", #Healthy2
4 "data_plantas/Test/Test/Rust/82add70df6ab2854.jpg", #Rust1
5 "data_plantas/Test/Test/Rust/84c3f037e53e66a1.jpg", #Rust2
6 ]

```

Los resultados obtenidos fueron los siguientes:

```

Imagen: data_plantas/Test/Test/Healthy/8ddd5ec1c0de38c4.jpg -> Clase predicha: Healthy
Imagen: data_plantas/Test/Test/Healthy/8def4d91382175c3.jpg -> Clase predicha: Rust
Imagen: data_plantas/Test/Test/Rust/82add70df6ab2854.jpg -> Clase predicha: Rust
Imagen: data_plantas/Test/Test/Rust/84c3f037e53e66a1.jpg -> Clase predicha: Healthy

```

Figure 4: Clasificación de 4 imágenes de dos tipos distintos

En la figura 5 mostramos las imágenes en cuestión, para poder analizarlo con más profundidad:



Healthy clasificada como Healthy



Healthy clasificada como Rust



Rust clasificada como Rust



Rust clasificada como Healthy

Figure 5: Ejemplos de imágenes clasificadas utilizando HOG+SVM

A continuación, discutimos posibles razones por las cuales el modelo ha clasificado incorrectamente algunas imágenes, basándonos en las características visibles de las hojas:

- **Healthy clasificada como Healthy:** En este caso, la clasificación fue correcta, ya que la hoja presenta una textura lisa y uniforme, sin patrones distintivos que puedan asociarse con enfermedades. Podría decirse que este tipo de hojas representan la clase *Healthy* de manera ideal.
- **Healthy clasificada como Rust:** Esta imagen fue incorrectamente clasificada. La confusión en este caso puede deberse a la presencia de zonas sombreadas en la hoja que generan gradientes similares a los patrones que aparecen en hojas afectadas por la enfermedad. Esto demuestra que el modelo puede ser sensible a cambios de iluminación.
- **Rust clasificada como Rust:** Este es un ejemplo de clasificación correcta. Las características visuales como las pústulas naranjas y los patrones asociados a la enfermedad han sido correctamente identificados por el modelo.
- **Rust clasificada como Healthy:** Esta imagen fue incorrectamente clasificada. En este caso, la confusión puede explicarse por la presencia de manchas poco desarrolladas en la hoja. Estas características pueden no ser lo suficientemente prominentes para que el descriptor HOG las diferencie de las texturas lisas típicas de las hojas sanas.

Los errores que hemos observado reflejan algunas limitaciones en la robustez del descriptor HOG, frente a variaciones de iluminación, orientación y textura. Aunque el modelo es eficaz para identificar patrones claros y distintivos, las imágenes que presentan características un poco más ambiguas ambigas o iluminaciones distintas, tienden a confundir al clasificador.

3.7 Conclusiones

La combinación de características HOG con un clasificador SVM ha demostrado ser medianamente efectiva para distinguir entre las clases *Healthy* y *Rust*. La precisión global del modelo y las métricas obtenidas para ambas clases reflejan un buen rendimiento. No obstante, se podrían explorar descriptores adicionales como LBP para comparar resultados y evaluar mejoras.

Algo a destacar es que el modelo entrenado es capaz de clasificar imágenes de prueba con la misma precisión que en entrenamiento, es decir, no está sobreajustando.

4 Clasificación usando LBP básico

4.1 Descriptor LBP

El descriptor *Local Binary Pattern* (LBP) es una técnica ampliamente utilizada en la extracción de características basadas en textura.

El LBP funciona asignando un valor binario a cada píxel de la imagen en función de su relación con los píxeles vecinos. Para ello:

1. Cada píxel se compara con sus P vecinos en un radio R . Si un vecino tiene un valor de intensidad mayor o igual que el píxel central, se asigna un valor de 1; de lo contrario, se asigna un valor de 0.
2. Los valores binarios resultantes se interpretan como un número decimal, lo que da lugar a un "código LBP" para el píxel central.
3. La imagen se divide en celdas o regiones, y se calcula un histograma de los códigos LBP para cada región.
4. Finalmente, los histogramas de todas las celdas se concatenan para formar el descriptor final de la imagen.

Lo anterior hace que el LBP sea muy útil para capturar información de texturas, independientemente de las variaciones de iluminación que pueda presentar la imagen.

4.2 Lectura de imágenes

Para esta práctica, hemos hecho uso de una implementación de LBP proporcionada por la librería `scikit-image`, que puede ser consultada en su documentación oficial: <https://scikit-image.org/>.

La función `load_training_data_LBP` permite leer y procesar las imágenes del dataset, calculando los descriptores LBP para cada imagen. Al igual que en el caso de HOG, esta función organiza los datos de entrenamiento a partir de carpetas separadas para cada clase y redimensiona las imágenes a un tamaño fijo de 128×128 píxeles.

El proceso comienza recorriendo los archivos en las carpetas de las clases positivas y negativas, asegurándose de que correspondan al formato de imagen definido (`.jpg`). Cada imagen es convertida a escala de grises, mediante la función `rgb2gray`, ya que el cálculo de LBP se realiza en imágenes en niveles de intensidad. Posteriormente, se calcula el patrón LBP básico mediante la función `local_binary_pattern`, configurando $P = 8$ (número de vecinos) y $R = 1$ (radio). Para representar cada imagen, se construye un histograma normalizado a partir de los valores LBP generados.

Se selecciona un subconjunto aleatorio y reproducible de imágenes (200 para entrenamiento y 50 para prueba), reduciendo la carga computacional sin comprometer la representatividad del conjunto. De nuevo fijamos una semilla fija para garantizar consistencia entre distintas ejecuciones. Los histogramas LBP se normalizan dividiendo cada valor entre la suma total más un factor de corrección (10^{-6}), lo cual permite la comparabilidad entre descriptores.

Finalmente, los descriptores y etiquetas resultantes se almacenan en listas que son convertidas a `numpy arrays` para facilitar su uso en las etapas posteriores de entrenamiento y evaluación del modelo.

Paralelamente, se ha implementado una función similar, `load_testing_data_LBP`, que realiza los mismos pasos descritos pero aplicados al conjunto de datos de prueba. Esta función también selecciona aleatoriamente 50 imágenes por clase para reducir la carga de procesamiento y permitir una evaluación rápida del modelo entrenado.

4.3 Entrenamiento del clasificador

Para encontrar el mejor modelo SVM al utilizar el descriptor LBP, se llevó de nuevo a cabo una búsqueda de hiperparámetros mediante `GridSearchCV`. Las combinaciones usadas son las mismas que antes.

La mejor combinación encontrada fue: `{'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}`

Con estos hiperparámetros, el modelo alcanzó un **F1-score** promedio de 0.7188 durante la validación cruzada. Posteriormente, hemos evaluado el modelo en el conjunto de prueba, obteniendo los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
Negativo	0.73	0.80	0.76	50
Positivo	0.78	0.70	0.74	50
Accuracy	75.00%			
Macro Avg	0.75	0.75	0.75	100
Weighted Avg	0.75	0.75	0.75	100

Table 4: Informe detallado de clasificación utilizando el mejor modelo SVM con descriptor LBP.

La matriz de confusión obtenida se presenta en la Figura 6:

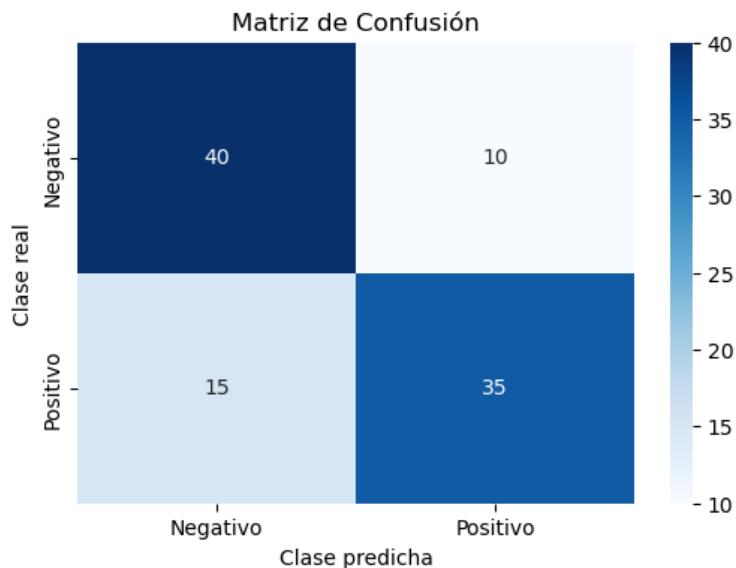


Figure 6: Matriz de confusión del mejor clasificador LBP+SVM.

En términos generales, el modelo ha mostrado un rendimiento equilibrado entre las clases positiva y negativa, con un **accuracy** del 75% y un *F1-score* de 0.76 para la clase negativa y 0.74 para la clase positiva. Estos resultados demuestran que el descriptor LBP combinado con SVM es una herramienta efectiva para tareas de clasificación basada en texturas.

4.4 Comparación HOG+SVM y LBP+SVM

En esta sección, comparamos el desempeño de los enfoques **HOG+SVM** y **LBP+SVM** utilizando los mismos conjuntos de datos y técnicas de validación. Aunque ambos descriptores son útiles para la extracción de características basadas en textura, han presentado ciertas diferencias en términos de desempeño y eficiencia computacional.

Una de las principales ventajas del descriptor LBP es su notable eficiencia en términos de tiempo de cómputo. Durante la validación cruzada, el cálculo de los descriptores HOG requirió aproximadamente **50 segundos por partición**, mientras que el cálculo de los descriptores LBP fue prácticamente instantáneo (**0 segundos**). Esta diferencia se debe a la simplicidad inherente del cálculo de LBP, que no requiere el procesamiento intensivo de gradientes y normalización que caracteriza a HOG.

En cuanto a la implementación, ambos descriptores fueron implementados utilizando bibliotecas preexistentes (**OpenCV** para HOG y **scikit-image** para LBP). Sin embargo, la simplicidad de LBP también se refleja en su configuración, ya que no requiere ajustar parámetros como el tamaño de los bloques o celdas.

En términos de desempeño, la opción **HOG+SVM** logró un accuracy del 74% en el conjunto de prueba, mientras que el enfoque **LBP+SVM** alcanzó un accuracy muy ligeramente superior, del 75%. Al analizar los *F1-scores*, HOG mostró un mejor equilibrio entre clases, mientras que LBP presentó ligeras diferencias en la precisión y el *recall* entre las clases positiva y negativa. En la siguiente tabla se muestran los resultados obtenidos:

Descriptor	Accuracy	Precision Negativo	Recall Negativo	F1-Score Negativo	Precision Positivo	Recall Positivo
HOG	0.74	0.75	0.72	0.73	0.73	0.76
LBP	0.75	0.73	0.80	0.76	0.78	0.70

Table 5: Resultados comparativos de los descriptores HOG y LBP

En conclusión, aunque el descriptor HOG ofrece un rendimiento más equilibrado entre clases, el descriptor LBP destaca por su simplicidad y rapidez computacional, lo que lo convierte en una opción más eficiente para problemas donde el tiempo de cómputo es crítico. Sin embargo, la elección del descriptor depende del problema en cuestión, y ambos enfoques han demostrado ser efectivos para esta tarea de clasificación.

5 Clasificación usando LBP uniforme

El descriptor **LBP uniforme** es una variante del descriptor LBP básico, que reduce la dimensionalidad de los descriptores generados, manteniendo su capacidad de capturar información relevante sobre la textura de las imágenes. Con LBP uniforme, solo se consideran patrones de transición entre píxeles que presentan un máximo de dos cambios de 0 a 1 o de 1 a 0 en su representación binaria, ignorando patrones más complejos. Esto simplifica el histograma resultante, y así podemos representar textura con mayor eficiencia, especialmente para imágenes con bajo contraste o ruido.

La principal diferencia entre el LBP básico y el uniforme es que el LBP básico genera descriptores con hasta 2^P bins (en este caso, 256 bins para $P = 8$), mientras que LBP uniforme reduce el número de bins a $P + 2$ (10 bins en este caso). Esta reducción permite un análisis más rápido y mejora la discriminación en problemas de clasificación donde los patrones no uniformes no aportan información significativa.

Para calcular el descriptor, hemos hecho uso de la función `local_binary_pattern` de la librería `scikit-image`, utilizando el método '`uniform`'.

En comparación con el LBP básico, el enfoque basado en LBP uniforme tiene las siguientes ventajas:

- **Reducción de dimensionalidad:** Menor número de bins en los histogramas, lo que resulta en descriptores más eficientes.
- **Robustez:** Mejoramos la discriminación en problemas de clasificación al eliminar patrones complejos que pueden no aportar información significativa.
- **Eficiencia computacional:** conseguimos menores tiempos de cómputo al trabajar con histogramas más pequeños.

5.1 Lectura de imágenes

La función `load_training_data_LBP_uniform` se encarga de leer y procesar las imágenes de entrenamiento de manera reproducible. La única diferencia con respecto al LBP básico es que mientras que en el LBP básico se generan 256 bins en el histograma, en LBP uniforme el número de bins se reduce a $P + 2 = 10$, lo que mejora la eficiencia al eliminar patrones no uniformes. Por tanto, el histograma generado tiene un tamaño más pequeño, lo que reduce el costo computacional en las etapas posteriores de entrenamiento y evaluación.

De manera similar, la función `load_testing_data_LBP_uniform` replica este proceso para las imágenes de prueba.

5.2 Entrenamiento del clasificador

Para encontrar el mejor modelo SVM utilizando el descriptor **LBP uniforme**, hemos llevado a cabo nuevamente una búsqueda de hiperparámetros mediante la herramienta `GridSearchCV`. Se utilizaron las mismas combinaciones de hiperparámetros que en los experimentos previos con los descriptores HOG y LBP.

La mejor combinación encontrada para el descriptor LBP uniforme fue: {’C’: 100, ’gamma’: ’scale’, ’kernel’: ’rbf’}

Con estos hiperparámetros, el modelo alcanzó un **F1-score** promedio de 0.7126 durante la validación cruzada. Posteriormente, evaluamos el modelo en el conjunto de prueba, obteniendo los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
Negativo	0.76	0.78	0.77	50
Positivo	0.78	0.76	0.77	50
Accuracy	77.00%			
Macro Avg	0.77	0.77	0.77	100
Weighted Avg	0.77	0.77	0.77	100

Table 6: Informe detallado de clasificación utilizando el mejor modelo SVM con descriptor LBP uniforme.

La matriz de confusión obtenida se presenta en la Figura 7:

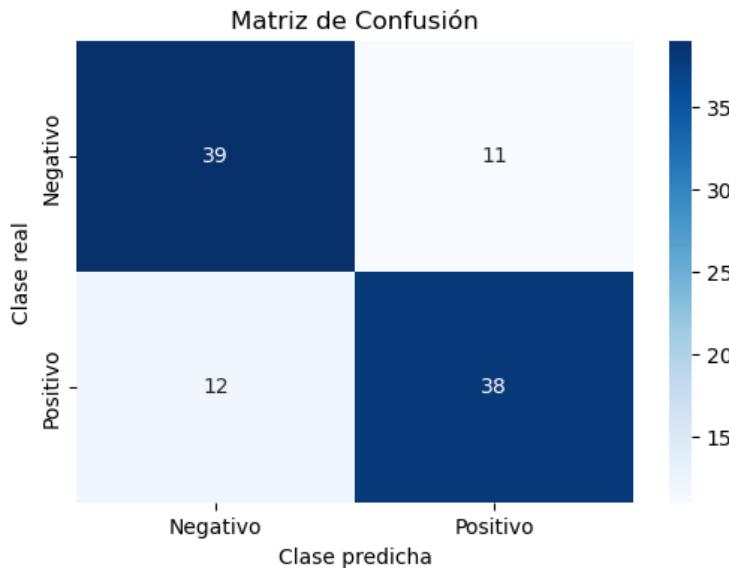


Figure 7: Matriz de confusión del mejor clasificador LBP uniforme+SVM.

El modelo entrenado con el descriptor LBP uniforme mostró un rendimiento superior respecto a los descriptores HOG y LBP básico, alcanzando un **accuracy** del 77% en el conjunto de prueba. Además, los *F1-scores* para ambas clases (positiva y negativa) fueron de 0.77, evidenciando un desempeño equilibrado entre las categorías.

5.3 Comparación HOG+SVM, LBP+SVM, LBPU+SVM

En esta sección, comparamos el desempeño de los enfoques **HOG+SVM**, **LBP+SVM** y **LBPU+SVM** utilizando los mismos conjuntos de datos y técnicas de validación.

Una de las principales ventajas de los descriptores basados en LBP es su notable eficiencia computacional en comparación con HOG. Durante la validación cruzada, el cálculo de los

descriptores HOG requirió aproximadamente **50 segundos por partición**, mientras que los descriptores LBP (básico y uniforme) se calcularon prácticamente de forma instantánea (**0 segundos**).

En cuanto a las diferencias entre LBP básico y LBP uniforme, este último reduce significativamente el número de patrones posibles, concentrándose en aquellos más relevantes. Esta simplificación no solo mantiene la rapidez de cálculo, sino que también mejora el rendimiento en problemas de clasificación al reducir el ruido en los histogramas. No obstante, como LBP básico ya tardaba muy poco, no hemos notado la diferencia.

En términos de implementación, los tres descriptores fueron construidos utilizando bibliotecas preexistentes: `OpenCV` para HOG y `scikit-image` para LBP básico y uniforme. Si bien HOG ofrece mayor personalización con parámetros como el tamaño de celdas y bloques, LBP destaca por su configuración más sencilla.

En términos de desempeño, los resultados fueron los siguientes:

- **HOG+SVM**: Alcanzó un **accuracy** del 74% y un ***F1-score*** promedio de 0.73.
- **LBP+SVM**: Mejoró ligeramente con un **accuracy** del 75% y un ***F1-score*** promedio de 0.74.
- **LBPu+SVM**: Mostró el mejor desempeño global con un **accuracy** del 77% y un ***F1-score*** promedio de 0.77.

Los resultados comparativos detallados se presentan a continuación:

Descriptor	Accuracy	Precision Negativo	Recall Negativo	F1-Score Negativo	Precision Positivo	Recall Positivo
HOG	0.74	0.75	0.72	0.73	0.73	0.76
LBP	0.75	0.73	0.80	0.76	0.78	0.70
LBPu	0.77	0.76	0.78	0.77	0.78	0.76

Table 7: Resultados comparativos de los descriptores HOG, LBP y LBP uniforme con SVM.

En conclusión, mientras que HOG ofrece un rendimiento más equilibrado entre clases, los descriptores LBP (especialmente el uniforme) destacan por su simplicidad y eficiencia computacional. El descriptor **LBPu+SVM** ha sido la mejor opción en este contexto, al combinar su alto desempeño con tiempos de cómputo mucho menores.

6 Incluir una tercera clase

Por suerte, el dataset que hemos escogido contiene 3 clases: Healthy, Rust y Powdery. Como vimos en el apartado 2, la tercera clase consiste en otra enfermedad de las hojas en la que los hongos forman una capa blanca sobre su superficie.

Para realizar este ejercicio, he redefinido las funciones para leer imágenes de forma que podamos leer más clases en lugar de solo dos. De esa forma, podemos leer las imágenes de la clase Powdery y calcular sus descriptores.

Al elegir la opción 2 en el menú de `main.py`, se ejecuta la función `ejecutar_clasificacion_3_clases()` y se realizan las clasificaciones para los 3 descriptores en las 3 clases definidas.

6.1 Clasificación con descriptor HOG

Se llama a las funciones `load_training_data_HOG_multiple()` y `load_testing_data_HOG_multiple()` para leer las imágenes y calcular sus descriptores HOG, tanto para train como para test. En el caso HOG, debido a que añadimos una clase más, el tiempo de ejecución se disparaba, por lo que hemos reducido el número de imágenes train por cada clase a 80, y a 20 para test.

Luego, directamente realizamos la búsqueda de hiperparámetros con validación cruzada, con la función `grid_search_multiple()`, que es igual que del caso de 2 clases, pero configurando la función `SVC()` para que acepte múltiples clases, mediante el parámetro `decision_function_shape='ovr'`.

Tras el proceso de búsqueda, los mejores hiperparámetros encontrados fueron: '`C`': 10, '`gamma`': 'scale', '`kernel`

Tomando dichos hiperparámetros y su modelo correspondiente, hemos procedido a evaluar su rendimiento para el conjunto de datos test. Ejecuando la función `evaluate_model_multiple()` se han obtenido los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
Healthy	0.55	0.55	0.55	20
Powdery	0.62	0.50	0.56	20
Rust	0.38	0.45	0.41	20
Accuracy	0.50 (50%)			
Macro Avg	0.52	0.50	0.50	60
Weighted Avg	0.52	0.50	0.50	60

Table 8: Informe detallado de clasificación con tres clases (Healthy, Powdery, Rust).

La matriz de confusión obtenida ahora es 3x3, pues tenemos 3 clases, como vemos en la figura 8:

Estos resultados reflejan que, aunque el modelo muestra cierta capacidad para distinguir la clase Powdery y quizás la Healthy, su rendimiento es muy limitado en términos de precisión y *recall*, especialmente para la clase **Rust**. Puede ser que al haberlo entrenado con tan pocos

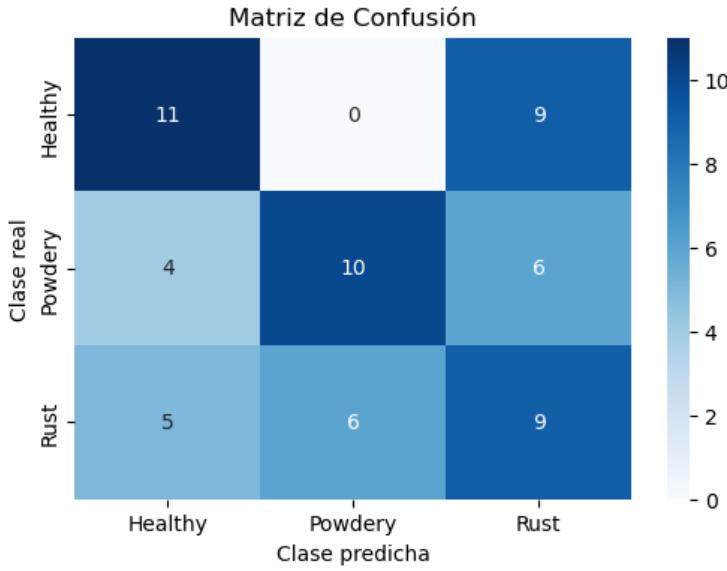


Figure 8: Matriz de confusión del mejor clasificador HOG+SVM para 3 clases.

ejemplos de entrenamiento, la eficacia del modelo se haya visto afectada. En cualquier caso, en los siguientes apartados usaremos un mayor número de ejemplos.

6.2 Clasificación con descriptor LBP

En consonancia con el apartado anterior, hemos definido nuevas funciones que importan las imágenes para las tres clases y calculan el descriptor LBP.

Tras el proceso de búsqueda, los mejores hiperparámetros encontrados fueron: {’C’: 100, ’gamma’: ’scale’, ’kernel’: ’rbf’}. Dicho modelo ha alcanzado una puntuación de F1 de 0.7632 durante la validación cruzada.

Tomando dichos hiperparámetros y su modelo correspondiente, hemos procedido a evaluar su rendimiento para el conjunto de datos de prueba. Ejecutando la función `evaluate_model_multiple` se han obtenido los siguientes resultados:

Clase	Precision	Recall	F1-Score	Support
Healthy	0.67	0.78	0.72	50
Powdery	0.93	0.74	0.82	50
Rust	0.65	0.68	0.67	50
Accuracy	0.73 (73%)			
Macro Avg	0.75	0.73	0.74	150
Weighted Avg	0.75	0.73	0.74	150

Table 9: Informe de clasificación con tres clases (Healthy, Powdery, Rust) usando descriptor LBP.

La matriz de confusión obtenida se observa en la Figura 9:

Estos resultados reflejan que el modelo basado en LBP ha mostrado un rendimiento muy superior al observado con el descriptor HOG en el caso de tres clases, logrando un accuracy

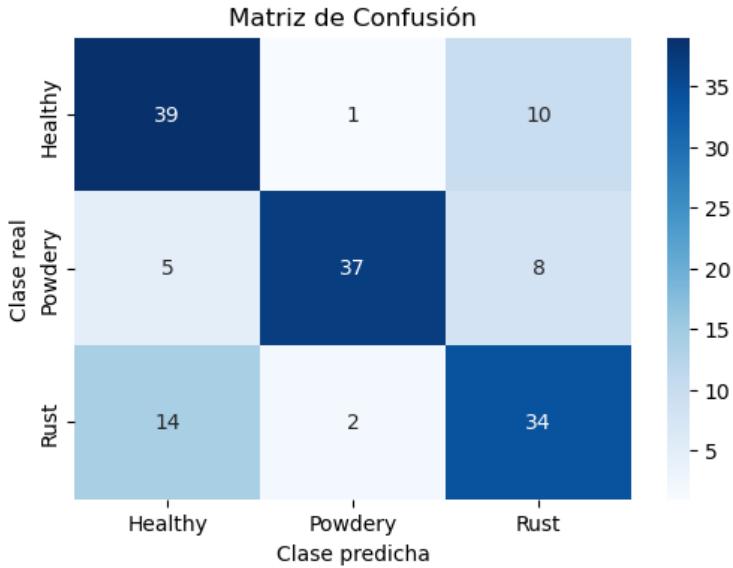


Figure 9: Matriz de confusión del mejor clasificador LBP+SVM para 3 clases.

del 73%. Destaca el desempeño del clasificador para la clase Powdery, con un *accuracy* de 0.93, un *recall* de 0.74 y un *F1-score* de 0.82. Sin embargo, la clase Rust sigue siendo un poco problemática, obteniendo valores de *precision* y *recall* de 0.65 y 0.68 respectivamente.

En conclusión, aunque el rendimiento del descriptor LBP ha mejorado notablemente, particularmente para las clases Healthy y Powdery, el modelo todavía tiene margen de mejora, especialmente en la diferenciación de la clase Rust. Este comportamiento puede deberse a unas texturas más similares entre Rust y Healthy (pues son idénticas excepto las manchas marrones), lo que dificulta la clasificación.

6.3 Clasificación con descriptor LBP uniforme

Hemos procedido de igual forma que en los anteriores casos para 3 clases. Tras realizar la búsqueda de hiperparámetros mediante la función `grid_search_multiple()`, los mejores hiperparámetros encontrados fueron: `{'C': 10, 'gamma': 'scale', 'kernel': 'poly'}`. Este modelo alcanzó una puntuación de F1 de 0.7005 durante la validación cruzada.

Para evaluar el rendimiento en el conjunto de datos de prueba, utilizamos la función `evaluate_model_multiple()`, obteniendo los siguientes resultados:

La matriz de confusión obtenida para este modelo, como se observa en la Figura 10, tiene una dimensión de 3×3 , lo que refleja las tres clases clasificadas:

Los resultados indican que el modelo presenta un rendimiento ligeramente equilibrado entre las tres clases, con la clase *Powdery* mostrando el mejor desempeño en términos de *precision* (**0.69**) y *recall* (**0.72**). En cambio, las clases *Healthy* y *Rust* tienen un desempeño inferior, especialmente en su *recall*, que se sitúa en **0.64** y **0.62**, respectivamente.

El modelo tiene un **accuracy** del 66%, que es ligeramente inferior al del LBP básico, lo

Clase	Precision	Recall	F1-Score	Support
Healthy	0.63	0.64	0.63	50
Powdery	0.69	0.72	0.71	50
Rust	0.66	0.62	0.64	50
Accuracy		0.66 (66%)		
Macro Avg	0.66	0.66	0.66	150
Weighted Avg	0.66	0.66	0.66	150

Table 10: Informe de clasificación con tres clases (Healthy, Powdery, Rust) usando descriptor LBP uniforme.

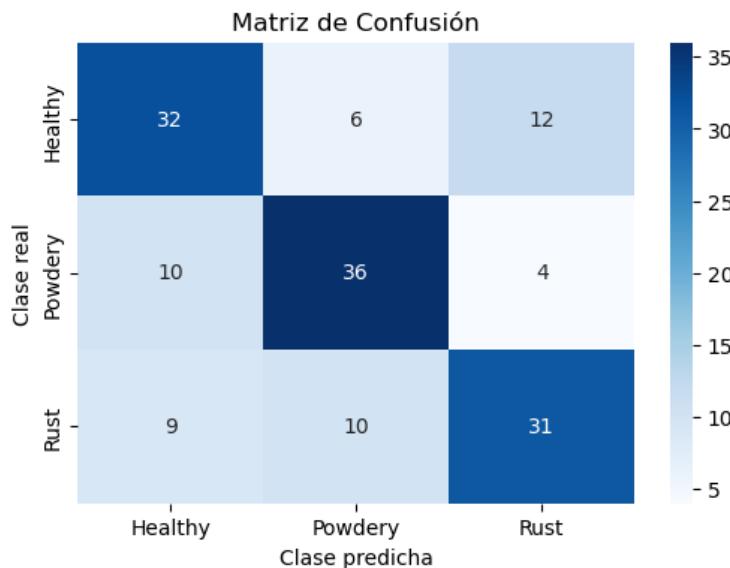


Figure 10: Matriz de confusión del mejor clasificador LBP uniforme+SVM para 3 clases.

cual se ve reflejado en una mayor cantidad de imágenes mal clasificadas para las tres clases.

6.4 Comparación HOG+SVM, LBP+SVM, LBPU+SVM

En este apartado comparamos los resultados obtenidos al aplicar los tres descriptores, **HOG**, **LBP básico** y **LBP uniforme**, junto con un clasificador SVM para la clasificación de las tres clases (*Healthy*, *Powdery* y *Rust*). Los resultados obtenidos se presentan en la siguiente tabla:

Descriptor	Accuracy	Precision Healthy	Recall Healthy	F1-Score Healthy	Precision Rust	Recall Rust	F1-Score Rust	F1-Score Powdery
HOG	0.50	0.55	0.55	0.55	0.63	0.50	0.56	0.41
LBP	0.73	0.67	0.78	0.72	0.93	0.74	0.82	0.67
LBPU	0.66	0.63	0.64	0.63	0.69	0.72	0.71	0.64

Table 11: Comparativa de resultados entre los descriptores HOG, LBP y LBP uniforme con SVM.

A partir de los datos obtenidos, se pueden extraer las siguientes conclusiones:

- **HOG+SVM**: Este enfoque muestra un desempeño muy limitado, con un accuracy de apenas el 50%. Aunque la clase *Healthy* tiene un desempeño moderado (F1-score de 0.55), el modelo tiene dificultades para distinguir entre las clases *Powdery* y *Rust*, con un *F1-score* de 0.41 para *Powdery*. Esto refleja la sensibilidad de HOG a variaciones en las texturas cuando las diferencias no son muy pronunciadas. También tenemos que tener en cuenta que el número de imágenes ha sido menor (un poco menos de la mitad), por lo que puede ser que el clasificador entrenado no tuviera suficientes datos.
- **LBP+SVM**: Este descriptor supera significativamente a HOG, logrando un accuracy de **73.33%**. La clase *Rust* muestra el mejor desempeño, con un F1-score de 0.82, gracias a la capacidad de LBP para capturar patrones de textura relevantes. Sin embargo, la clase *Powdery* presenta una ligera reducción en su recall, lo que indica cierta dificultad para distinguirla de las otras clases en algunos casos.
- **LBPu+SVM**: Aunque este enfoque no supera en rendimiento general a LBP básico, muestra un desempeño equilibrado entre las tres clases, al menos con respecto al HOG. Con un accuracy de 66%, logra resultados similares para *Rust* y *Powdery*, con F1-scores de 0.71 y 0.64, respectivamente. Su rendimiento es más consistente, aunque bastante inferior en términos globales al de LBP básico.

En términos generales, **LBP+SVM** se posiciona como la mejor combinación para este problema, al ofrecer el mejor balance entre precisión y recall, especialmente para la clase *Rust*. Sin embargo, el descriptor **LBPu+SVM** muestra ventajas al seguir estando bastante equilibrado y poco sensible a las variaciones entre clases, lo que lo hace una alternativa viable en contextos los tiempos de ejecución se disparen (que no es el caso).

7 Conclusiones generales

En este trabajo hemos explorado y comparado tres enfoques principales para la clasificación de imágenes en un conjunto de datos con dos y tres clases: HOG+SVM, LBP+SVM y LBP uniforme+SVM. Cada uno de estos enfoques ha demostrado ventajas y desventajas particulares, lo cual nos ha permitido elegir el método más adecuado en función del contexto de nuestro problema. Los errores observados han reflejado algunas limitaciones de los métodos empleados frente a variaciones de iluminación, orientación y textura. En general, hemos comprobado que las imágenes que presentan características más ambiguas o con distintas iluminaciones, pueden llegar a confundir al clasificador.

Por un lado, en la **clasificación binaria**, el descriptor LBP ha superado ligeramente a HOG en términos de precisión general, logrando un accuracy del 75% frente al 74% de HOG. Además, LBP ha demostrado una ventaja computacional muy significativa, ya que su cálculo fue prácticamente instantáneo, mientras que HOG requirió aproximadamente 50 segundos por partición.

No obstante, el descriptor LBP uniforme ha presentado el mejor desempeño global, con un accuracy del 77%, destacando como la opción más equilibrada y efectiva. Este enfoque combina la simplicidad del cálculo de LBP con una representación reducida y eficiente de las texturas. De todas formas, los resultados fueron muy similares para los tres descriptores, por lo que no podemos sacar conclusiones concluyentes de cuál es mejor, salvo por el tiempo de cómputo.

Por otro lado, en la **clasificación multiclase**, cuando se incorporó una tercera clase al problema, los resultados evidenciaron diferencias mucho más marcadas entre los descriptores:

El descriptor HOG fue el peor debido a que tuvo un 50% de accuracy. Todas las clases y en especial, la clase *Powdery* resultaban difíciles de clasificar, lo cual se refleja en las métricas tan bajas, muchas veces incluso por debajo del 0.5. El descriptor LBP fue el más efectivo con un accuracy del 73%, en comparación con su compañero LBP uniforme, que sólo obtuvo una precisión del 66%.

En conclusión, los resultados indican que **el descriptor LBP es la mejor opción** para nuestro conjunto de datos, especialmente en tareas multiclase. Sin embargo, en escenarios donde la simplicidad y la velocidad de cálculo son primordiales, el descriptor LBP uniforme puede ser una alternativa muy competitiva, especialmente en problemas donde las clases presentan diferencias más sutiles.