Universidad del Biobío
Facultad de Ciencias Empresariales
Escuela de Ingeniería Civil en Informática

Estudio y comparación de algoritmos para el reconocimiento visual y clasificación de hojas de árboles nativos de la región del Biobío

Proyecto de título para optar al Título de Ingeniero Civil en Informática

Alumno: Cesar Andrés Navarrete Ulloa

Profesor Guía: Pedro G. Campos
Profesor Informante: Roberto Mercado

Concepción - Chile
15 de Enero, 2019

Índice General

Índice de Tablas

Índice de Figuras

Resumen

Este proyecto de título es presentado para dar conformidad a los requisitos exigidos por la Universidad del Bío-Bío en el proceso de titulación para la carrera de Ingeniería Civil en Informática. El proyecto titulado como "Estudio y comparación de algoritmos para el reconocimiento visual y clasificación de hojas de árboles nativos de la región del Biobío" tiene como objetivo comparar múltiples algoritmos de reconocimiento y clasificación de hojas de árboles.

Con el fin de comparar estos algoritmos se realizaron implementaciones propias de estos en el lenguaje de programación Python. A partir de los algoritmos implementados se desarrollaron 2 algoritmos modificados para comprobar cómo afectan ciertas características de una hoja de árbol (forma o textura de la hoja), a la tarea de clasificación de hojas.

La comparación de estos algoritmos se realizó en base a un protocolo de prueba definido, en donde se describen 3 situaciones de pruebas. Las tablas de resultados obtenidas a partir de las pruebas contienen las métricas de clasificación básicas de presicion, recall, f1- score y accuracy. Es en base a estos resultados que se desarrolla el análisis y las conclusiones de este proyecto de título.

El desarrollo de este proyecto de título demuestra la importancia de las características de textura para clasificación de hojas de árboles y como esta mejora las métricas de clasificación. Además se señala la importancia de realizar validación cruzada para el entrenamiento de modelos de clasificación y como no realizarla puede llevar a conclusiones erradas. Por último se señala la relevancia de la calidad de las características extraídas a partir de una hoja de árbol para la construcción de un modelo de clasificación.

Abstract

The following project was done in order to comply with the requirements demanded by the "Universidad del Bío-bío" to obtain the degree for the career "Ingeniería Civil en Informática". The project is entitled "Estudio y comparación de algoritmos para el reconocimiento visual y clasificación de hojas de árboles nativos de la región del Biobío" and it's main objective is to compare multiple leaf recognition and classification algortithms.

To compare these algorithms it was necessary to implement them in the programming language "Python". By modifying the implemented algorithms it was possible to develop 2 new modified algorithms which were useful to understand how certain leaf features affect the process of leaf classification.

The comparison between algorithms was done in accordance with a defined test protocol, in which 3 test situations are defined. The result tables obtained after testing contain the following basic classification metrics: precision, recall, f1-score and accuracy. These results were the basis for the analysis section and the project conclusion.

This project proves how important texture features are in the task of tree leaf classification and how using these features can help to improve the classification metrics. This project also highlights the importance of the process of cross validation to train classification models and how ignoring it can lead to mistaken conclusions. Lastly the project shows the relevance of choosing quality features to train a tree leaf classification model.

Capítulo 1: Introducción

1.1 Motivación

Los arboles existen alrededor de todo el mundo y juegan un rol muy importante en el medio ambiente. Algunos árboles tienen propiedades medicinales en beneficio de los seres humanos (Tilburt & Kaptchuk, 2008), otros son grandes fuentes de energía alternativa en forma de biocombustibles y otros son utilizados por la industria maderera para ser vendidos como material de construcción, para calefacción, etc. (Placencia & Bañados, 2015). La rápida distinción de especies de árboles es un tema de interés hoy en día ya sea para la precisa obtención de árboles beneficiosos, evitar arboles dañinos en su totalidad o la identificación y clasificación de árboles utilizados por la industria. Es por esto que resulta relevante el estudio de algoritmos para el reconocimiento visual preciso de árboles a partir de algunas(s) de sus partes, tales como sus hojas.

Diversos autores han utilizado variados métodos para lograr el reconocimiento de hojas de árboles, ya sea a través del reconocimiento de características propias de las hojas con un enfoque en la forma de ésta (Chaki & Parekh, 2011) o través de la textura única que cada especie posee. Algunos autores incluso combinan estas propiedades (Chaki, Parekh, & Bhattacharya, 2015) y utilizan la forma y textura para logar un reconocimiento más preciso.

La razón que impulsa esta propuesta de Proyecto de Título, es el interés que se tiene por el reconocimiento de especies y los algoritmos que permiten realizar este reconocimiento. Además es una verdad que en la actualidad reconocimiento visual es un tema de interés tanto para la persona común, por ejemplo utilizando aplicaciones de celular con OCR, como para los investigadores.

Es un hecho de que a pesar que vivimos rodeados de árboles y plantas, las personas más jóvenes no tienen idea a que especies pertenecen. Siendo solo las generaciones más antiguas las que aun guardan este conocimiento. En nuestra región del Biobío, una región con una vegetación tan variada como la nuestra, donde existen una gran variedad de especies de árboles y plantas, sean medicinales, frutales, o con un inmenso valor cultural, en un futuro la implementación de algún algoritmo efectivo de identificación de especies en alguna aplicación educacional podría acercar a la persona común a este patrimonio cultural generalmente ignorado.

Dicho esto se propone en este Proyecto de Título, la investigación de algoritmos para el reconocimiento visual de especies de árboles nativos de la región del Biobío, a través de la lectura de trabajos relevantes, con el fin de implementar aproximaciones de estos algoritmos y realizar comparaciones entre ellos, para poder determinar si existe la posibilidad de realizar alguna mejora de estos.

Como agregado y a pesar de no ser el foco principal del proyecto, cabe destacar que nuestra región, la región del Biobío, es una región agricultora, donde se cultivan variados cereales y existe además una industria forestal activa que representa una de nuestras principales exportaciones (INE, 2017).

En la industria la efectiva obtención y clasificación de especies puede significar un ahorro de tiempo y dinero para las empresas.

1.2 Objetivos

1.2.1 Objetivo General

Comparar el rendimiento de Algoritmos de reconocimiento visual en la tarea de clasificación de árboles seleccionados en la Región del Biobío, Chile, a partir de fotografías de sus hojas.

1.2.2 Objetivos Específicos

- Determinar las características visuales de mayor relevancia en hojas de árboles seleccionados de la Región del Biobío, para poder realizar un reconocimiento de especie a partir de dichas características.
- Generar un dataset de imágenes de hojas de árboles representativos de la Región del Biobío, Chile.
- Implementar algoritmos de extracción de características y de clasificación de imágenes.
- Implementar y comparar algoritmos de reconocimiento visual de autores estudiados.

1.3 Descripción de la metodología utilizada

Determinar las características visuales de mayor relevancia en hojas de árboles, para poder realizar un reconocimiento de especie a partir de dichas características.

Para poder lograr determinar las características más relevantes en la identificación de las hojas de árboles se utilizaron como base, trabajos relacionados con el tema a investigar, que han sido publicados en revistas científicas.

Cabe destacar que basándose en el estudio de estas publicaciones en el momento de realizar este informe, se ha determinado que se puede reconocer una especie de árbol utilizando la forma de sus hojas, ya que estas tienen una forma única que corresponde a una especie determinada.

Otra característica que se ha determinado como importante para la identificación de una especie de árbol utilizando sus hojas, es la textura de estas mismas, la razón siendo que al igual que la forma, la textura es una característica única para cada especie.

Generar un dataset de imágenes de hojas de árboles representativos de la Región del Biobío, Chile.

Para entrenar y probar los modelos de clasificación de hojas de árboles generados durante este Proyecto de Título, fue necesario contar con un dataset de especímenes de hojas de árboles.

Durante el desarrollo del proyecto, para la construcción del dataset, se decidió recolectar 3 especies de hojas de árboles nativos de la región del Biobío, las cuales fueron: Drimys Winteri (Canelo Chileno), Peumus Boldus (Boldo) y Laurelia Sempervirens (Laurel Chileno).

El dataset entonces está compuesto de 300 fotografías de hojas de árboles, 100 por cada especie de árbol. Un porcentaje de las hojas recolectadas fue utilizado para la entrenamiento del modelo, mientras que el resto se utilizó para el testeo de los algoritmos de reconocimiento visual de especies.

Implementar algoritmos de extracción de características y de clasificación de imágenes

Para realizar la implementación de los algoritmos de extracción de características visuales, se utilizaron las librerías de Python “OpenCv” y “Scikit Image”.

La primera es una de las librerías Open Source para la tarea de visión por computador más populares que existen, y cuenta con múltiples métodos para la manipulación de imágenes y detección de objetos, que son muy útiles al momento de implementar los algoritmos de extracción de características visuales estudiados.

En cambio, la librería Scikit Image fue utilizada para implementar métodos no existentes en la librería OpenCv.

Para la implementación de los algoritmos de clasificación de imágenes, se utilizan las librerías de Python, Scikit-learn y Neupy.

Estas librerías permiten el entrenamiento de los modelos para la clasificación de especies de árboles utilizando los datos de entrenamiento proporcionados por los algoritmos de extracción de características. Además permiten utilizar este modelo para poder realizar la predicción de especie al analizar una hoja de árbol.

Implementar y comparar algoritmos de reconocimiento visual de autores estudiados

Utilizando el lenguaje de programación Python, se construyeron rutinas de programación para contar con implementaciones propias de algoritmos de reconocimiento visual que han sido propuestos por otros autores.

Una vez implementados los algoritmos, se realizó una comparación de éstos, además de experimentos para evaluar ajustes a estos algoritmos que puedan resultar en alguna mejora.

1.4 Herramientas a utilizar

Software a utilizar

Pycharm 2018.1.3: Entorno de desarrollo integrado (IDE) para el lenguaje de programación Python. Este IDE será utilizado para el desarrollo de las rutinas para la detección y clasificación de hojas de árboles nativos de la región del Biobío.

Librerías Utilizadas

OpenCv v.3.4.1: Librería especializada en visión por computadoras (*computer vision*). Cuenta con una amplia gama de métodos para la manipulación de imágenes, detección de objetos, entre otros.

En este proyecto es utilizada para la limpieza de las imágenes, la detección de contornos, la binarizacion de imágenes, entre otros usos.

Scikit-image v.0.14.0: Librería de Python especializada en el procesamiento de imágenes.

En este proyecto cumple la función de proveer métodos que "OpenCv" no posea. Como por ejemplo matrices de coocurrencia de niveles de gris.

Scikit-learn v.0.20.2: Librería especializada en Machine Learning. Permite implementar diversos algoritmos de Clasificación, Regresion y Clustering entre otros.

Se usa esta librería para implementar los algoritmos de Machine Learning utilizados por los autores estudiados.

Neupy: Librería de Python que permite construir e implementar diversos algoritmos de Machine Learning.

Esta librería permite implementar algoritmos no disponibles en la librería Scikit-learn.

1.5 Estructura del Informe

En el Capítulo 1 se explica la motivación de este proyecto de título, los objetivos que se quieren lograr con este, como se lograron y las herramientas utilizadas para su desarrollo. En el Capítulo 2 se explica en detalle las principales técnicas y conceptos que fueron aprendidos con el fin de realizar este proyecto. En el Capítulo 3 se presentan los diferentes enfoques que tomaron los 3 autores estudiados con el fin de realizar la tarea de reconocimiento visual de especies de árboles a partir de sus fotografías. En el capítulo 4 se explica cómo se realizó la implementación propia de estos 3 algoritmos estudiados en el lenguaje de programación Python, y además se explica la implementación de 2 algoritmos de elaboración propia. En el Capítulo 5 se presentan los resultados que fueron obtenidos al probar los algoritmos implementados utilizando el dataset de elaboración propia, además se realiza una discusión de los resultados obtenidos.

Por último la sección final de “Conclusiones” da un cierre a este proyecto de título. En esta sección se remarca los objetivos logrados y se realizan algunas conclusiones a partir de los resultados obtenidos en el capítulo 5.

Capítulo 2: Marco conceptual

En este capítulo se explican las principales técnicas y conceptos aprendidos para el desarrollo de este proyecto de título. La sección 2.1 explica los conceptos generales que se deben conocer para poder entender la tarea de reconocimiento visual de especies de hojas de árboles. La sección 2.2 explica con detalle las técnicas y conceptos necesarios para poder realizar esta tarea.

2.1 Conceptos Generales

2.1.1 Pattern Recognition

La facilidad con la que identificamos objetos, personas, plantas y animales a través de la vista, olfato, tacto, entre otros, hace que a menudo nos olvidemos acerca de la verdadera complejidad de estos actos de reconocimiento de patrones.

Pattern recognition es el estudio de cómo hacer que las maquinas puedan observar el mundo que las rodea, de aprender a distinguir patrones de interés, y tomar decisiones razonables respecto a las categorías de los patrones encontrados (Jain, Duin, & Mao, 2000).

Un patrón puede ser definido como una entidad, vagamente definida, a la que se le puede dar un nombre. Un patrón puede ser una huella digital, la letra escrita de una persona, una cara o una forma de hablar.

A partir de lo anterior es fácil pensar acerca de la utilidad que podría tener la automatización del reconocimiento de patrones, por ejemplo, formas automáticas de reconocer caras, huellas digitales, escritura manual y secuencias de ADN. Dicho esto es natural que se busque diseñar y construir maquinas que puedan reconocer patrones. Resolviendo los problemas que conlleva construir estos sistemas se gana apreciación y entendimiento de los sistemas de reconocimiento de patrones en el mundo real, particularmente en los humanos (Duda, Hart, & Stork, 2012, p. 3).

La automatización del reconocimiento de patrones es un problema que se ha podido solucionar aplicando algoritmos de Machine Learning. Alimentando a una computadora con ejemplos de un patrón que se quiere reconocer se puede "enseñar" a esta a identificar correctamente una cara, huella digital, especies de hoja, etc.

Si se quiere distinguir entre dos objetos diferentes, en otras palabras, pertenecientes a categorías diferentes, entonces al igual a lo que hacemos en el mundo real hay que comenzar a buscar las diferencias entre estos 2 objetos. Encontradas estas diferencias decimos que ambos objetos tienen modelos diferentes, descripciones diferentes, las cuales son expresadas de forma matemática. El objetivo general en el reconocimiento y clasificación de patrones es determinar a qué clase o categorías pertenecen estos modelos.

Para entender mejor el tema consideremos el siguiente ejemplo (Duda et al., 2012): Tenemos una fábrica de pescado en donde se quiere automatizar el proceso de distinguir entre una Lubina y un Salmon, se utilizarán imágenes fotográficas para capturar la información visual de los peces. Las diferencias de tamaño, forma, textura de las escamas pueden servir como características que identifican individualmente a cada pez. El sistema propuesto para resolver este problema (ver Fig. 1) tiene las siguientes funciones: Primero captura la imagen de un pez. Después, la fotografía es pre procesada para simplificar las demás operaciones sin perder información relevante. Es esta parte donde se utilizan técnicas de manipulación de imágenes como la segmentación para aislar el pescado del fondo de la imagen. Esta información del pez es enviada a un "extractor de características", calculando propiedades y características a partir de la imagen segmentada que la describen de manera adecuada. Finalmente, las características extraídas son procesadas por un clasificador que toma la decisión de a qué especie pertenece el pescado en la fotografía original.

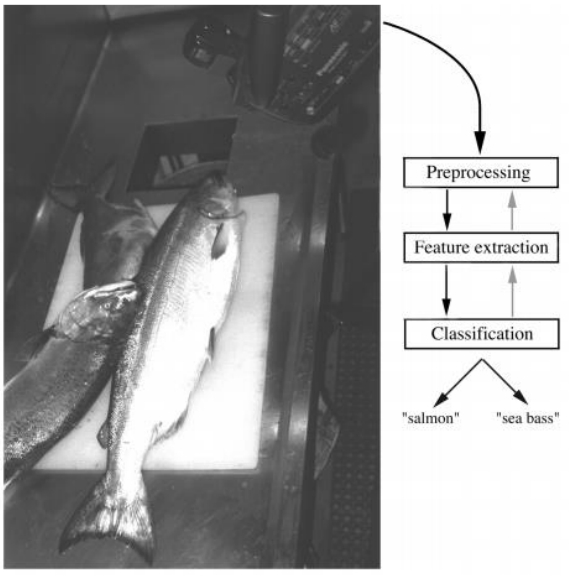


Fig. 1: Proceso de clasificación de pescados. Fuente: (Duda et al., 2012).

2.1.2 Machine Learning

Para resolver un problema en un computador es necesario contar con un algoritmo. Un algoritmo es una secuencia de instrucciones que transforman una o más entradas en una o más salidas. Por ejemplo, se puede necesitar ordenar una secuencia de números en un vector de valores, para ello se escoge un algoritmo apropiado para la tarea.

El problema que existe es que en algunos casos no contamos con un algoritmo apropiado para ordenar la información. Determinar la diferencia entre correo spam y legitimo es una tarea complicada, uno sabe que la salida es un sí/no respecto a si el

correo es spam, pero no conocemos de una forma para trasformar las entradas (email) en las salidas deseada (si/no). Para hacer más complicadas las cosas, lo que se considera como spam cambia todo el tiempo. Una forma de solucionar este problema es reuniendo datos. Se pueden recolectar mensajes los cuales ya se sabe si son spam o no, y es a partir de estos datos que queremos aprender qué actualmente se considera como "spam". A partir de la relación entre estos datos de entrada y de las salidas queremos que el computador extraiga automáticamente una función (algoritmo) para realizar esta tarea (Alpaydin, 2014, p. 1).

Recordando lo planteado en la sección anterior, se puede lograr que una computadora aprenda a reconocer patrones a través del entrenamiento de algoritmos de Machine Learning, con varios ejemplos de dichos patrones.

Una definición general de Machine Learning es: Métodos computacionales que utilizan la experiencia para mejorar su rendimiento o para realizar predicciones acertadas (Mohri, 2012, p. 1). Por experiencia se entiende, información disponible que será utilizada con el objetivo de que la computadora aprenda. Como se dijo anteriormente estos datos pueden ser datos ya clasificados por personas. La calidad y cantidad de los datos son importantes para asegurar que la computadora que realiza el aprendizaje realice predicciones acertadas.

Es posible decir que una maquina aprende cuando en respuesta a información externa, cambia su estructura, datos o programas de modo que en el futuro tenga un rendimiento mejor (Nilsson, 1996, p. 1).

Algunos de los escenarios de aprendizaje más relevantes son (Mohri, 2012):

Aprendizaje supervisado: Una maquina recibe como entrada una serie de datos ya rotulados (ejemplo spam) y usando estos ejemplos realiza predicciones respecto a información nueva. Este escenario está asociado a problemas de clasificación, ranking y regresión.

Aprendizaje no supervisado: Una maquina recibe como entrada ejemplos no rotulados y debe extraer patrones o la estructura subyacente en los datos. Como no se cuenta con ejemplos rotulados puede ser difícil tratar de medir el rendimiento de la máquina que está aprendiendo. Clustering y reducción de dimensionalidad son ejemplos de algoritmos de aprendizaje automático no supervisados.

Algunos de los problemas de aprendizaje más relevantes son (Mohri, 2012):

Clasificación: Asignar una categoría a cada elemento. Por ejemplo, se requiere clasificar una hoja de árbol dentro de un grupo de especies conocidas. Por lo general el número de categorías en este tipo de problemas es pequeño, aun así pueden existir problemas en donde el número de categorías es masivo como *Optical Character Recognition* (OCR) donde se debe categorizar correctamente un símbolo.

Regresión: Es predecir un valor real (numérico) para cada elemento. Ejemplos de regresión son predicciones en la bolsa de valores, precio de algún bien, entre otros.

Clustering: Separar elementos en regiones homogéneas. Generalmente se recurre al *Clustering* cuando se quiere analizar set de datos de gran tamaño, como por ejemplo, al analizar las redes sociales los algoritmos de *Clustering* intentan identificar comunidades en grupos masivos de personas.

Reducción de Dimensionalidad: Transformar una representación inicial de elementos en una representación de menor dimensión, conservando las propiedades importantes de la representación inicial. Un ejemplo de esto es el pre procesamiento de imágenes en tareas de *Computer Vision*.

Ranking: Ordenar elementos a partir de un criterio. Ejemplos incluyen los buscadores web, que a partir de una consulta de un usuario deben retornar los resultados más relevantes, basándose en palabras claves o búsquedas pasadas.

2.1.3 Computer Vision

Computer Vision es el campo interdisciplinario encargado de que los computadores puedan interpretar las imágenes recibidas, desde por ejemplo una cámara de video o imágenes estáticas (T. Huang, 1996, p. 1). Algunas de las disciplinas que comprende *Computer Vision* son la inteligencia artificial, la robótica, el procesamiento de señales, el reconocimiento de patrones (*pattern recognition*), teoría de control, psicología, neurociencia, entre otros.

Los seres humanos podemos percibir el mundo en tres dimensiones que nos rodea de forma fácil y sencilla. Podemos analizar una escena delante de nuestros ojos y distinguir rápidamente un objeto de otro, así como la distancia aproximada a la que estos objetos se ubican (Szeliski, 2010, p. 3). El cerebro humano divide la señal de visión en varios canales, los cuales transmiten diferentes tipos de información al cerebro, y este puede identificar con facilidad partes importantes de una imagen, mientras que puede ignorar áreas de la imagen que no tengan relación con lo que se está buscando. Hay entradas asociativas generalizadas desde sensores de control muscular y todos los demás sentidos, que permiten al cerebro recurrir a asociaciones cruzadas hechas a partir de la experiencia acumulada de vivir tantos años en el mundo. Los circuitos de retroalimentación en el cerebro se remontan a todas las etapas de procesamiento, incluyendo los sensores de hardware (los ojos), que controlan mecánicamente la iluminación a través de la Iris y regulan la recepción en la superficie de la retina (Bradski & Kaehler, 2008, p. 2,3).

En un sistema de visión artificial la información que recibe un computador no es nada más que una matriz de valores numéricos, que pueden representar las intensidades de los pixeles. A diferencia de los seres humanos, los computadores no tienen una forma innata de saber qué formas u objetos están en la imagen, no tienen la experiencia acumulada de años de vivir en el mundo para saber distinguir, por ejemplo, una persona de otra, no cuentan con las conexiones cerebrales que le permitan recordar el nombre y cara de una persona o separar el fondo y frente de una imagen. La computadora solo "ve" una matriz de valores. Visualícese una imagen donde existan peras y manzanas y se quiera contar el número de manzanas, como se indicó anteriormente, la computadora no sabe que hay en la imagen, entonces la primera tarea seria convertir la matriz de números en la percepción de que existen manzanas en la imagen (Bradski & Kaehler, 2008).

Claramente los investigadores se enfrentan a un problema difícil. Dada una representación 2D del mundo 3D no existe una forma única de reconstruir la señal 3D. Una misma imagen 2D podría representar una combinación infinita de escenas 3D. Añadiendo a esta dificultad está el hecho de que los datos extraídos desde las imágenes pueden estar corruptos, debido las condiciones propias de mundo, como la luz, las

reflexiones, los movimientos, además de otros problemas como las imperfecciones que puedan existir en la lente en la que se toman las imágenes (Bradski & Kaehler, 2008).

Psicólogos perceptuales han tratado de comprender como funciona nuestro sistema visual durante décadas, pero a pesar de que se han entendido algunos de sus principios, parece que una respuesta completa aun no será posible (Szeliski, 2010).

Computer vision es usada ampliamente en la actualidad en una gran variedad de tareas diferentes (Szeliski, 2010):

Reconocimiento Óptico de Caracteres (OCR): Las cámaras de seguridad pueden leer las placas patentes de automóviles que estén infringiendo las normas del tránsito. El reconocimiento automático de caracteres en una imagen es posible gracias a *Computer Vision*.

Inspección de Maquinarias: Se puede identificar rápidamente partes dañadas en la maquinaria, detectando los defectos de dichas partes que se aprecian en imágenes.

Construcción de modelos 3D: Construcción automática de modelos 3D, a partir de fotografías áreas.

Seguridad Automovilística: Detectar obstáculos presentes en la vía, como animales o peatones.

Match Moving: Combinar imágenes generadas por computadora (CGI) con video real, siguiendo puntos clave en el video fuente para estimar el movimiento de la cámara 3D y la forma de ambiente.

Detección de rostros: Para mejorar el foco de la cámara hacia una persona o la identificación de esta.

Modelado 3D: Convertir una o más imágenes en un modelo 3D del objeto o persona que se está fotografiando.

2.2 Reconocimiento de Plantas

Por lo menos en el contexto de este Proyecto de Título, el reconocimiento de plantas se refiere a las técnicas de manipulación de imágenes, extracción de características y el modelamiento de la información respecto a plantas, que permiten la identificación correcta de una especie de planta, a partir de una imagen digital de dicha planta. Al leer esta definición, se puede encontrar una relación con el campo de *Pattern Recognition*. Efectivamente el reconocimiento de plantas es un problema que pertenece al reconocimiento de patrones. Para reconocer especies de plantas debemos enseñarle a un computador a distinguir patrones de interés, en este caso las diferentes especies de plantas.

Para resolver el problema del reconocimiento de plantas podemos aislar los siguientes pasos (ver Fig. 2) que se deberán seguir (Chaki & Parekh, 2011; Chaki et al., 2015; Gang Wu et al., 2007; R.-X. Hu, Jia, Ling, & Huang, 2012):

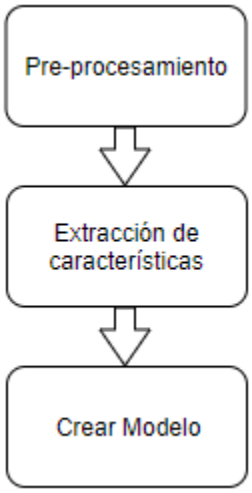


Fig. 2: Pasos generales para resolver un problema de reconocimiento de patrones (Fuente: Elaboración propia)

Pre procesamiento de imágenes

El tratamiento de la imagen que se realiza antes de extraer las características de ésta, puede realizarse para lograr varios objetivos, como eliminar el ruido y las imperfecciones de la imagen, convertir la imagen a escala de grises, separar la hoja del fondo de la imagen utilizando segmentación (Gang Wu et al., 2007) estandarizar el tamaño y forma de la hoja de planta (Chaki et al., 2015), resaltar características, entre otros.

Extracción de características

Una vez ya tratadas las imágenes, es momento de reunir las características importantes para crear el modelo de clasificación que permitirá realizar las predicciones acerca de la especie de las hojas de árboles procesadas.

Según lo investigado, las características más importantes a extraer desde una hoja de árbol son la forma, la textura y los momentos de la imagen. Se pueden utilizar estas características por separado para crear el modelo (Gang Wu et al., 2007; R.-X. Hu et al., 2012), o combinaciones de diferentes características como la forma y la textura (Chaki & Parekh, 2011; Chaki et al., 2015). Las características extraídas desde una imagen son generalmente guardadas en un vector de características.

Modelo de Clasificación

Para la creación del modelo se destina un porcentaje del conjunto de datos o casos con que se cuenta (dataset) para realizar el entrenamiento del algoritmo de clasificación, se denomina a esta porción del dataset como set de entrenamiento (training set), mientras que la porción usada para testear al modelo que se va a crear se le llama set de prueba (testing set).

El paso siguiente es el escoger un algoritmo de Machine Learning adecuado para la tarea. Reconocer la especie de una planta es un problema de clasificación, por lo que varios autores han intentado resolverlo usando redes neuronales como "perceptrones multicapa" (Chaki & Parekh, 2011) o "redes neuronales probabilísticas" (Gang Wu et al., 2007) para crear modelos que puedan realizar una predicción respecto a una hoja de árbol que no ha sido clasificada. El entrenamiento se realiza con los vectores de características obtenidas durante el proceso de extracción de características.

Habiendo completado el entrenamiento, se puede probar la eficacia del modelo usándolo sobre el conjunto de prueba para realizar predicciones acerca de la especie de cada hoja.

2.2.1 Técnicas de Pre-procesamiento

A continuación se presentan algunos de los conceptos y técnicas de pre-procesamiento más prominentes en los trabajos estudiados (Chaki & Parekh, 2011; Chaki et al., 2015; Gang Wu et al., 2007; R.-X. Hu et al., 2012) sin duda pueden existir muchos más, pero se explican a continuación aquellos que serán usados para el desarrollo de este Proyecto de Título.

2.1.1 Imagen en escala de grises

Una imagen RGB (*Red-Green-Blue*) está compuesta por 3 canales, un canal rojo (R), uno verde (G) y uno azul (B). Cada uno de estos canales puede tomar un valor desde 0 a 255. A mayor valor, mayor es la intensidad del color del canal. Las combinaciones de los valores de estos 3 canales permiten visualizar una imagen a color.

En contraste una imagen en escala de grises solo tiene un canal, donde los valores de la intensidad de los pixeles van desde 0 (Negro) a 255 (Blanco). Los demás valores entre estos 2 números corresponden a tonos de grises.

Para convertir una imagen RGB a escala de gris es necesario aplicar una fórmula que utiliza los valores de R, G y B en un pixel de una imagen a color con el fin de determinar el valor equivalente que debe tener un pixel en una imagen en escala de gris. La librería de *Computer Vision* OpenCv utiliza la siguiente formula (OpenCv, n.d.):

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

En este Proyecto de Título, para poder utilizar algunas de las funciones de manipulación de imagen de las librerías OpenCv y Scikit-Image, es necesario tener una imagen en escala de gris por lo que el primer paso es realizar la conversión de la imagen original RBG a escala de gris.

2.1.2 Thresolding

La segmentación es generalmente la primera etapa en cualquier intento de analizar e interpretar una imagen de forma automática (Efford, 2000, p. 250). La segmentación divide una imagen en distintas regiones de características similares.

Según lo indicado en (Efford, 2000) una de las técnicas de segmentación más ocupadas es el *thresholding*. El *thresholding* transforma un atributo de distintos valores en un nuevo atributo de tan solo 2 valores. Esto se logra aplicando un umbral a los datos de entrada. Los valores de entrada que estén por debajo del umbral serán remplazados por uno de los 2 valores de salida, mientras que los valores por encima del umbral tomaran el valor de salida restante.

Thresholding de imágenes es una técnica de segmentación, ya que clasifica los pixeles de una imagen en 2 categorías: Aquellos pixeles que luego de calcular alguna propiedad de la imagen caen bajo un umbral, y los pixeles iguales o superiores al umbral definido. Como solo existen 2 salidas, el *thresholding* crea una imagen binaria, donde efectivamente se separa uno o más objetos de interés del resto de la imagen.

Comúnmente se realiza *thresholding* sobre una imagen en escala de grises, donde se toma la imagen y se define un umbral, para luego aplicar la siguiente regla sobre cada pixel de la imagen:

$$g(x,y) = \begin{cases} 0, f(x,y) < T \\ 1, f(x,y) \geq T \end{cases}$$

En palabras sencillas, si el valor del pixel $f(x,y)$ en la imagen original, es menor al umbral definido entonces el valor del pixel $g(x,y)$ será 0, en cualquier otro caso su valor será de 1. Esto valores son más que nada teóricos, ya que como se trabaja con imágenes digitales en escala de grises se usan los valores 0 y 255 (donde 0 es negro y 255 es blanco, los máximos valores opuestos de esta escala).

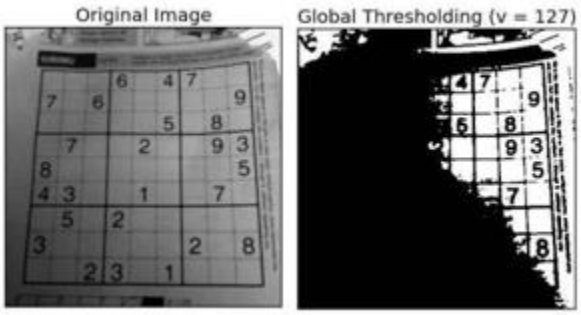


Fig. 3: Selección de umbral. Se utilizó un umbral de valor 127 para realizar Thresholding sobre la primera imagen, esto dio como resultado la segunda imagen, una imagen binaria Fuente: (OpenCV, n.d.-a).

Como se puede apreciar en la Fig. 3 es importante definir correctamente el umbral para realizar el *thresholding*, o podemos terminar con una imagen binaria que no transmita la misma información que la imagen original. Para determinar el valor correcto del umbral simplemente se puede cambiar este valor de forma manual hasta que se genere la imagen binaria ideal. El problema se presenta cuando se quiere implementar *thresholding* en alguna aplicación automática. Una solución a esto es el método de Otsu (Otsu, 1979) que calcula de forma automática un umbral global que se aplica a todos los pixeles de la imagen, si bien es conveniente, las diferencias de iluminación en una imagen entorpecen el cálculo del umbral, por lo que no siempre es la alternativa más ideal. Otro método es el *Adaptative Threshold*, que separa la imagen en varias regiones del mismo tamaño y calcula un umbral para cada una de las regiones generadas (OpenCV, n.d.-a), por esto es muy útil para realizar *thresholding* en una imagen con iluminación variada.

2.1.3 Operaciones Morfológicas

Las imágenes binarias producidas por las técnicas de segmentación como el *thresholding*, pueden contener varias imperfecciones, causadas por el ruido en la imagen, la textura, o la especificación equivocada de un umbral durante el proceso mismo de *thresholding*. La aplicación de operaciones morfológicas sobre una imagen puede lograr remover estas imperfecciones, y dejarnos con la información más importante de la imagen como la estructura y la forma de la imagen.(Efford, 2000)

A continuación se definirán algunas de las operaciones morfológicas que son relevantes para este Proyecto de Título:

Erosión

Como se puede observar en la Fig. 4 la operación de erosión remueve capas de pixeles de un objeto, haciéndolo más pequeño en el proceso. Los pixeles son erosionados desde dentro y fuera del objeto presente en la imagen, por lo que hará más grande cualquier agujero presente dentro del objeto. La erosión también eliminará imperfecciones en los

bordes de los objetos (Efford, 2000). Otro uso de la operación de erosión es la búsqueda de bordes.

Fig. 4: Efectos de la erosión. A la izquierda se muestra la imagen original, y a la derecha su imagen erosionada
Fuente:(OpenCV, n.d.-c).

Dilatación

La dilatación tiene el efecto contrario al de la erosión. Añade capas de pixeles a un objeto por lo que termina incrementando su tamaño. Los pixeles son añadidos en los bordes internos y externos de la figura (ver Fig. 5), por lo que la dilatación tendrá el efecto de cerrar los agujeros dentro de una figura, además de rellenar algunas imperfecciones en los bordes (Efford, 2000).

Fig. 5: Efectos de la dilatación. A la izquierda se muestra la imagen original, y a la derecha su imagen dilatada
Fuente: (OpenCV, n.d.-c)

Apertura Morfológica

Es simplemente una operación de erosión seguida por otra de dilatación, útil para remover el ruido presente en la imagen. La operación de erosión por si sola podría lograr el mismo resultado, pero también reduce el tamaño de la figura en la imagen. La ventaja de la apertura morfológica es que una vez realizada la erosión y el ruido sea removido, se realiza una dilatación que restaurará la figura erosionada a su tamaño original(Efford, 2000).

Fig. 6: Efectos de la apertura morfológica. A la izquierda se muestra la imagen original, y a la derecha la misma imagen tras una apertura morfológica. Fuente:(OpenCV, n.d.-c).

Cerrado Morfológico

En palabras sencillas es una dilatación seguida por una erosión. Es útil para cerrar agujeros que existan dentro de la figura. Como en la operación anterior, se puede rellenar la figura solamente utilizando la operación de dilatación, pero esto provoca que aumente de tamaño. La ventaja de realizar un cerrado morfológico, es que al realizar la dilatación se cierran agujeros dentro de la imagen y después utilizando erosión se reduce la figura a su tamaño original (Efford, 2000).

Fig. 7: Efectos del cerrado morfológico. A la izquierda se muestra la imagen original, y a la derecha la misma imagen tras un cerrado morfológico. Fuente:(OpenCV, n.d.-c).

2.2.2 Técnicas de extracción de características

A continuación se presentan algunas de las técnicas utilizadas para la extracción de características desde una imagen de una hoja de árbol, junto a sus explicaciones correspondientes.

2.2.1 Forma

La forma de una hoja es una forma común de clasificar las hojas respecto a sus contornos, lo cual tiene sentido, los seres humanos comúnmente identificamos objetos según la forma de éste (R.-X. Hu et al., 2012). Según lo estudiado, los momentos invariables de la imagen igualmente pueden ser usados para describir una imagen.

Momentos Invariantes

Para una imagen digital el momento m de un pixel $P(x, y)$ en la posición (x, y) es la multiplicación de la intensidad del pixel (su valor) por el valor de la distancia de las coordenadas, es decir, $m = x * y * P(x, y)$. Sabiendo esto el momento de una imagen es la suma de los momentos de todos lo pixeles.

El momento de orden (p, q) de una imagen $I(x\ , y)$ es:

$$m_{pq} = \sum_{x} \sum_{y} x^p \times y^q \times I(x, y)$$

$$p, q = 0,1,2 \ldots, +\infty$$

Donde p es el orden de x y q es el orden de y, por orden entiéndase la potencia que toma el componente correspondiente en la ecuación.

Los Momentos computados a partir de una imagen digital (momentos de imágenes) representan características globales acerca de la forma de tal imagen (Mukundan & Ramakrishnan, 1998, p. 1). La habilidad de representar características de la imagen, ha hecho que los momentos de imágenes sean ocupados para tareas de identificación en áreas como *Computer Vision* y robótica.

Se debe mencionar que los momentos calculados con la fórmula anterior no son invariantes cuando $I(x, y)$ cambia por rotación, traslación o escalado (Z. Huang & Leng, 2010, p. 477).

Para lograr calcular características invariantes antes de la traslación de la imagen se deben calcular momentos centrales (μ_{pq}) usando el centroide de la imagen (x_{avg}, y_{avg})

$$\mu_{pq} = \sum_x \sum_y I(x,y) \times \left(x - x_{avg}\right)^p \times \left(y - y_{avg}\right)^q$$

$$p, q = 0,1,2 \ldots, +\infty$$

$$x_{avg} = \frac{m_{10}}{m_{00}} \qquad y_{avg} = \frac{m_{01}}{m_{00}}$$

Características invariantes ante el escalado pueden lograrse a través de la normalización de los momentos centrales. Los momentos centrales normalizados son:

$$\eta_{pq} = \frac{\mu_{pq}}{m_{00}^{\frac{p+q}{2}+1}}$$

Hu (M.-K. Hu, 1962) propuso 7 características derivadas de los momentos normalizados de una imagen, que pueden utilizarse para describir dicha imagen, estas características no varían ante la rotación, el escalado y la traslación.

$I_1 = \eta_{20} + \eta_{02}$

$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$

$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$

$I_4 = (\eta_{30} - \eta_{12})^2 + (\eta_{21} - \eta_{03})^2$

$I_5 = (\eta_{30} - 3\eta_{12}) \times (\eta_{30} + \eta_{12})^2 \times [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03}) \times (\eta_{21} + \eta_{03}) \times [3(\eta_{30} + 3\eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

$I_6 = (m_{20} - m_{02}) \times [\,(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2] + 4m_{11} \times (m_{30} + m_{12}) \times (m_{21} + m_{03})$

$I_7 = (3m_{21} - m_{03}) \times (m_{21} + m_{03}) \times [3(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2] - (m_{21} - m_{03}) \times (m_{21} + m_{03}) \times [3(m_{30} + 3m_{12})^2 - (m_{21} + m_{03})^2]$

El Laplaciano

Podemos imaginar una imagen como una superficie, con una altura que representa los niveles de grises presentes en la imagen. Teniendo en cuenta esto se pueden calcular las derivadas de segundo orden. Estas miden la tasa a la que la pendiente de la superficie de niveles de gris cambia respecto a la distancia recorrida por ésta en las direcciones x e y. Estas derivadas son útiles para realizar detección de bordes en una imagen(Efford, 2000).

Como se puede observar en la Fig. 8, el gráfico superior representa los niveles de gris en una imagen dada. El gráfico del centro representa la primera derivada donde se puede observar como el cambio de intensidad observado en el gráfico superior queda expresado como un monte que crece rápidamente con este cambio, pero decrece al terminar éste. El gráfico inferior representa la segunda derivada, que permite observar dónde se encuentra el borde buscado, que corresponde al punto donde esta segunda derivada pasa por cero (*zero crossing*). Nótese la relación entre el máximo valor de la primera derivada en $x = 32$ y el punto que cruza por cero en la segunda derivada en $x = 32$.

Entonces el laplaciano de una imagen f queda expresado como una suma de las derivadas de segundo orden:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

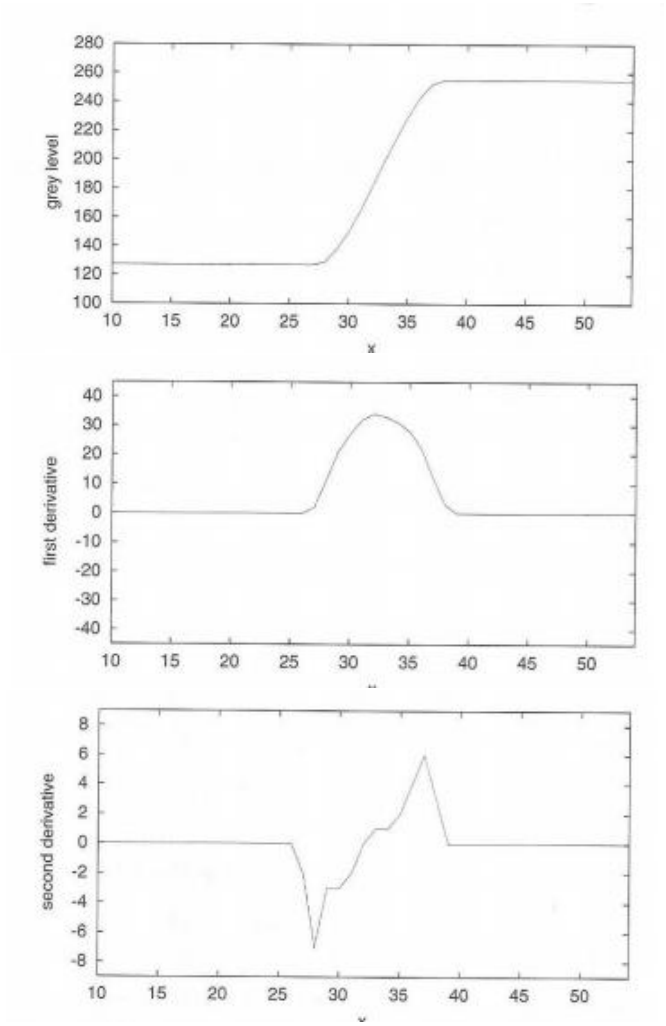


Fig. 8: Detección de bordes usando derivadas de primer y segundo orden. Fuente:(Efford, 2000).

Filtro Promedio

Se utiliza para reducir el ruido en una imagen y consiste en aplicar una máscara sobre cada pixel en una imagen en escala de grises. Esta mascara define el tamaño y forma de una región de pixeles y al ser aplicada sobre un pixel se usa para calcular la media de los valores de todos los pixeles en el área definida por la máscara teniendo al pixel que está siendo evaluado en el centro de la máscara. El valor calculado reemplaza el valor del pixel original (Rakshit, Ghosh, & Shankar, 2007, p. 890).

Para explicar de forma más clara el concepto obsérvese la Fig. 9, una máscara de tamaño 3x3, la cual es aplicada sobre un pixel en una imagen en escala de grises como se ilustra en la Fig. 10. En la imagen se puede apreciar que la máscara se está aplicando sobre un pixel con valor 140, calculando la media de todos los pixeles en el área de 3x3 con el pixel de valor 140 en el centro de la máscara. El valor final de este pixel será:

$$\frac{1}{9}[101 + 100 + 103 + 110 + 140 + 120 + 134 + 134 + 135] = 121$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Fig. 9: Mascara promedio de tamaño 3x3. Fuente:(Vernon, 1991, p. 57)

101 * 1/9	100 * 1/9	103 * 1/9	105	107	105	103	110
110 * 1/9	140 * 1/9	120 * 1/9	122	130	130	121	120
134 * 1/9	134 * 1/9	135 * 1/9	131	137	138	120	121
132	132	132	133	133	150	160	155
134	140	140	135	140	156	160	174
130	138	139	150	169	175	170	165
126	133	138	149	163	169	180	185
130	140	150	169	178	185	190	200

Fig. 10: Mascara promedio sobre una imagen en escala de grises. Fuente:(Vernon, 1991)

Detector de bordes de Canny

El detector de bordes de Canny toma en cuenta que para cada algoritmo de detección de bordes existe un intercambio entre la reducción de ruido en una imagen y la detección misma de bordes. Una buena reducción de ruido en una imagen lleva a la pérdida de información respecto a bordes en la imagen. Teniendo esto en cuenta, el algoritmo de detección de bordes de Canny da el mejor compromiso posible entre estos 2 requerimientos que se contraponen (Efford, 2000).

Un aspecto a resaltar sobre este algoritmo es la capacidad de tomar los pixeles individuales que son clasificados como parte de un borde, y ensamblarlos juntos en forma de contornos. Estos contornos son formados aplicando un umbral de histéresis a los pixeles. Esto quiere decir que existen 2 umbrales, uno superior y otro inferior. Si un pixel tiene una gradiente calculada mayor al umbral superior entonces es un pixel de un borde, si la gradiente es menor al umbral inferior se rechaza. Si la gradiente de un pixel se encuentra entre los valores del umbral inferior y superior entonces es aceptado como pixel de un borde solo si está conectado a un pixel con una gradiente mayor al umbral superior (Bradski & Kaehler, 2008). En la Fig. 11. se muestra un ejemplo de su aplicación.

Fig. 11: Detector de bordes de Canny aplicado a una imagen. A la izquierda se muestra la imagen original, y a la derecha los bordes detectados. Fuente: (OpenCV, n.d.-b)

2.2.2 Textura

La textura tiene una gran variedad de definiciones, por lo que no se intentará incurrir en una definición precisa. El lector solo debe tener en cuenta que la textura se relaciona con el tacto, una superficie puede ser rugosa, suave, con protuberancias, entre otras. Si bien una textura puede parecer rugosa desde cerca, esta misma puede parecer suave desde lejos, por lo que al momento de observar una textura se debe tener en cuenta una escala. Una textura es rugosa cuando en una superficie existe una gran diferencia entre altos y bajos, los cambios de elevación en la imagen son pronunciados (ver Fig. 12), y el espacio entre estos cambios de elevación es aproximadamente el mismo que el tamaño de la escala. Mientras que una textura puede describirse como suave cuando la diferencia entre altos y bajos en una superficie es pequeña, los cambios de elevación en la imagen son casi nulos (ver Fig. 12), y los cambios de elevación están agrupados muy cerca, en relación al tamaño de la escala usada (Hall-Beyer, 2017, p. 4). La textura en una imagen digital funciona de manera similar, excepto que los altos y bajos ya no son diferencias de elevación, si no que se ocupa la diferencia que existe entre los valores de brillo (intensidad) de los pixeles.

La textura se ha utilizado como una característica relevante a extraer desde una hoja de árbol para su posterior clasificación (Chaki et al., 2015). Según el profesor Myrka Hall-Beyer añadir algunas medidas de la textura puede mejorar la precisión de cualquier clasificador. (Hall-Beyer, 2017, p. 6)

Fig. 12: Textura rugosa a la izquierda y textura suave a la derecha Fuente: (Hall-Beyer, 2017)

Gray Level Co-Occurrence Matrix

Un método estadístico para examinar la textura que considera la relación espacial de los pixeles en una imagen es *Gray Level Co-Occurrence Matrix* (GLCM). La GLCM caracteriza la textura de una imagen calculando que tan seguido pares de pixeles con valores específicos se repiten en combinaciones específicas. A partir de la GLCM se pueden calcular varias propiedades estadísticas que proveen información acerca de la textura de una imagen (MathWorks, n.d.).

En palabras sencillas una GLCM es una tabulación del número de veces que se repiten distintas combinaciones de niveles de gris en una imagen.(Hall-Beyer, 2017)

Consideremos la Fig. 13 y los valores (niveles de gris) asociados a cada pixel:

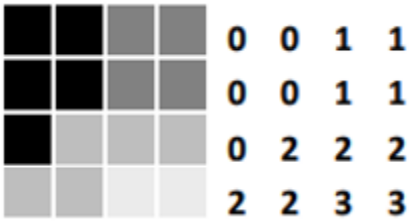


Fig. 13: Textura en escala de grises (izquierda), matriz de intensidad de los pixeles (derecha) (Fuente: (Hall-Beyer, 2017)*).*

GLCM considera la relación entre 2 pixeles a la vez, llamados pixel de referencia y pixel vecino, para este ejemplo el pixel vecino es el pixel a la derecha del pixel de referencia, esta relación puede expresarse como (1, 0), 1 pixel en la dirección de X y 0 en dirección de Y. Ya que la distancia entre el pixel de referencia y el vecino es de 1, entonces se dice que la GLCM tiene un offset de 1. Cada pixel en la imagen se vuelve un pixel de referencia, partiendo desde la esquina superior izquierda, hasta la esquina inferior derecha.(Hall-Beyer, 2017)

Teniendo esto en cuenta se construye la GLCM en donde se guardan las combinaciones de niveles de gris que se vayan encontrando. En las Fig. 14 y Fig. 15 se muestra una GLCM vacía, y sus valores finales, respectivamente.

neighbour pixel value -> ref pixel value:	0	1	2	3
0				
1				
2				
3				

Fig. 14: GLCM vacía (Imagen modificada con las combinaciones eliminadas para ilustrar mejor el proceso). Fuente: (Hall-Beyer, 2017)

Teniendo esto claro, completar la GLCM es sencillo, el primer pixel de referencia (0,0) en la matriz tiene como valor de gris 0, y su vecino el pixel (0,1) igualmente tiene el valor de 0, por lo que se cambia el valor de la casilla (0,0) en GLCM a 1, lo que indica que hasta ahora se ha encontrado una combinación de estos 2 valores en la imagen de entrada. Se itera entonces por toda la matriz, repitiendo el mismo proceso anterior, hasta obtener el número de veces que ocurre cada combinación de niveles de gris en la imagen.

neighbour pixel value -> ref pixel value:	0	1	2	3
0	2	2	1	0
1	0	2	0	0
2	0	0	3	1
3	0	0	0	1

Fig. 15: GLCM completa Fuente: (Hall-Beyer, 2017)

Reducción de Dimensionalidad

En ocasiones, los datos del dataset pueden contar con varias características. Cuando la cantidad de características se considera muy alta es recomendable reducir sus dimensiones, o encontrar una representación de menores dimensiones que conserve sus propiedades (Mohri, 2012). Algunos de los argumentos clave para realizar la reducción de dimensionalidad son:

Computacionalmente: Se quiere comprimir los datos iniciales en la etapa de pre-procesamiento para aumentar la velocidad de las operaciones subsiguientes sobre los mismos datos.

Visualización: Para visualizar los datos con el fin de realizar un análisis exploratorio mapeando los datos de entrada en 2 o 3 dimensiones.

Extracción de Características: Para generar un set de características de menor tamaño que pudiera ser más útil o efectivo.

Análisis de Componentes Principales (PCA)

La idea principal de PCA es encontrar un pequeño número de combinaciones lineales de parámetros correlacionados para describir la mayor parte de la varianza presente en el dataset en solo un pequeño número de parámetros no correlacionados (Einasto et al., 2011, p. 3). PCA trasforma los datos a un nuevo sistema de coordenadas, donde la mayor parte de la varianza para cualquier proyección de los datos se encuentra en la primera coordenada, el primer componente principal, la segunda mayor parte de la varianza, se encuentra en el segundo componente principal, etc.

Existen tantos componentes principales como existen parámetros, pero solo se necesitan de los primeros componentes para explicar la varianza.

Filtro de Gabor

Los filtros de Gabor han sido usados para varias aplicaciones como la segmentación y clasificación de textura, detección de caracteres, reconocimiento de huellas digitales, entre otros (Zheng, Zhao, & Wang, 2004, p. 139).

Una de las razones de la capacidad de discriminación de texturas de las funciones de Gabor parece ser su aptitud para modelar la respuesta de células corticales (presentes en los globos oculares), que están dedicadas a procesar señales visuales. La relación entre funciones de Gabor y el sistema visual de algunos animales ha sido un tema investigado por varios autores, como Daugman quien encontró que el comportamiento de células simples puede ser modelado por funciones de Gabor (Bianconi & Fernández, 2007, p. 1).

Las siguientes ecuaciones y sus explicaciones se presentan como están expresadas en (Chaki et al., 2015).

Un filtro de Gabor complejo se define como el producto de Kernel Gaussiano y una sinusoide compleja.

Una curva Gaussiana de 2 dimensiones g con una propagación de σ en las direcciones x e y se puede representar como:

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} * \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Una sinusoide compleja donde u representa la frecuencia espacial, θ la orientación y φ es el cambio de fase, puede expresarse como:

$$s(x, y, u, \theta, \varphi) = \exp\{j2\pi(x * ucos\theta + y * usin\theta) + \varphi\}$$

$$j = \sqrt{-1}$$

La función de Gabor compleja h es entonces:

$$h(x, y, \sigma, u, \theta, \varphi) = g(x, y, \sigma) * s(x, y, u, \theta, \varphi)$$

Se puede aplicar este filtro de Gabor h sobre una imagen en escala de grises $I(x, y)$ para así producir un par de señales complejas J.

$$J(x, y) = I(x, y) \otimes h(x, y, \sigma, u, \theta, \varphi)$$

La señal puede ser separada en su parte real (rgg) e imaginaria (igg) para ser utilizada en tareas como el análisis de textura.

$$rgg\ (x, y) = Re\ \{J(x, y)\}$$

$$igg\ (x, y) = Im\ \{J(x, y)\}$$

2.2.3 Algoritmos de Machine Learning para la creación de un Modelo de clasificación

A continuación se presenta una breve explicación acerca de algunos de los algoritmos de Machine Learning que han sido utilizados por algunos autores para el entrenamiento de modelos de clasificación con el objetivo de poder clasificar correctamente las especies de hojas de árboles.

2.3.1 Multilayered Perceptron

Antes de hablar de perceptrones multicapa, es necesario hablar sobre el perceptrón. Un perceptrón es la unidad básica de procesamiento en una red neuronal artificial. Tiene entradas que pueden ser externas, o que pueden ser salidas de otros perceptrones. Como se puede observar en la Fig. 16 los elementos $X_1, X_2, \cdots, X_m$ representan las entradas (considerando m valores de entrada), mientras que los elementos $W_{k1}, W_{k2}, \cdots, W_{km}$ representan el peso asignado a cada una de las entradas en el perceptron k, b_k es el peso que viene desde una entrada adicional, la unidad extra de *bias* o sesgo, la cuya entrada siempre es +1. Se usa este peso adicional para hacer que el modelo sea más general. Paso siguiente es calcular el producto de la suma de las entradas por su peso correspondiente.

$$v_j = \sum_{j=1}^{d} w_j x_j + b_k$$

Generalmente la amplitud normalizada de la salida de una neurona se expresa en el intervalo [0,1] en un intento de imitar como las señales son transmitidas entre las neuronas del cerebro. Para lograr esto se utiliza una *función de activación* que limita la amplitud de la salida de una neurona a un valor finito (Haykin, 2004, p. 32). Algunas de estas funciones son el sigmoide con una amplitud entre 0 y 1, y la función escalón de Heavyside o escalón unitario cuya salida es 0 o 1. Teniendo esto en cuenta se puede entonces tomar la suma ponderada obtenida en la ecuación anterior y aplicar una función de activación (representada por φ) para obtener el valor de la señal y_k.

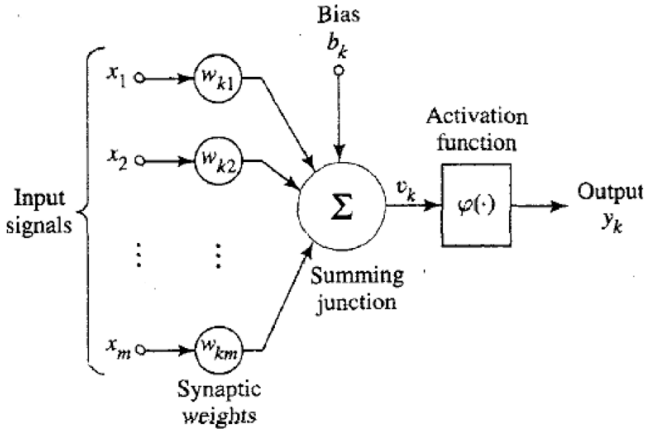


Fig. 16: Un perceptron, se aplica una función de activación a la suma ponderada de las entradas y pesos, con el fin de determinar a qué clase pertenecen las entradas. Fuente:(Haykin, 2004, p. 33)

Para poder resolver algún problema, primero se necesita aprender los pesos para que dadas una o más entradas, se puedan generar las salidas correctas. Durante el proceso de entrenamiento se usan ejemplos en las entradas de manera que se calculen cada uno de los pesos para poder generar la salida adecuada. Como se puede observar en la Fig. 17, un perceptrón puede utilizarse para implementar una función de discriminación lineal con el objetivo de separar 2 clases. En palabras sencillas, un perceptrón permite dibujar una línea para separar 2 clases de entradas, todo lo que esté sobre la línea pertenece a una clase, mientras que las entradas debajo de esa línea pertenecen a otra clase. Dicho esto, es importante recordar que el perceptrón sólo permite resolver problemas separables de forma lineal.

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

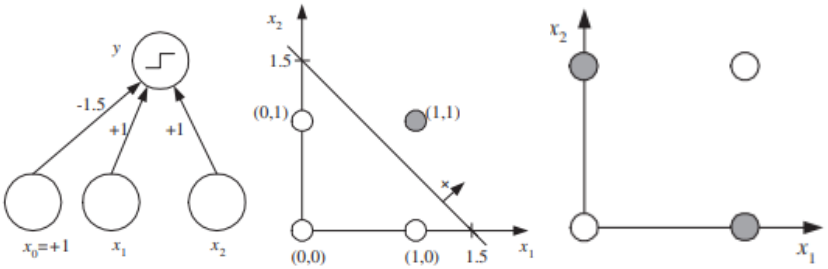


Fig. 17: En el grafico a la izquierda (problema AND) se puede observar que el problema puede ser separado fácilmente por una línea. Esto no es el caso en el grafico derecho (Problema XOR), no hay forma de dibujar una línea que separe los puntos que representan las entradas. Fuente: (Alpaydin, 2014)

Un perceptrón con una capa de pesos solo puede aproximar funciones lineales de las entradas y no puede resolver problemas como la función XOR (ver Fig. 17), que no es un problema lineal. Redes con capas escondidas entre las capas de entrada y salida no tienen esta limitación. Si este perceptrón "multicapa" es usado para la clasificación entonces puede implementar discriminantes no lineales, y si es usado para la regresión puede aproximar la función no lineal de las entradas.

En la Fig. 18 puede observarse la estructura de un perceptrón multicapa. Se pueden apreciar las 3 capas principales: de entrada, escondida y de salida. Al igual que en el perceptrón, la capa de entrada recibe los parámetros externos además del elemento extra de bias $X_0 = +1$. El elemento W_{hj} representa el peso asignado por el perceptrón Z_h a la entrada x_j de la primera capa. La suma ponderada de la capa de entrada es usada para computar la función de activación (un sigmoide en este caso) y obtener la señal de salida. Los valores de las señales obtenidas en la capa oculta serán usados como

entradas por el perceptrón en la capa de salida, en la capa oculta nuevamente se calculan pesos V_{ih} (asignados por el perceptrón de salida Y_i a la salida del perceptrón oculto Z_h) y se agrega un bias Z_0. El proceso finaliza con la salida Y_i donde nuevamente se calcula el sigmoide usando la suma ponderada de las entradas y pesos en la capa oculta. Como se puede ver en la Fig. 19 un perceptrón multicapa puede ser usado para resolver el problema no lineal de XOR.

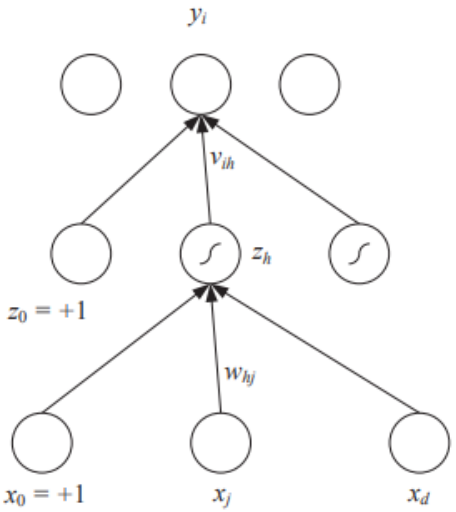


Fig. 18: Perceptron multicapa. Fuente: (Alpaydin, 2014)

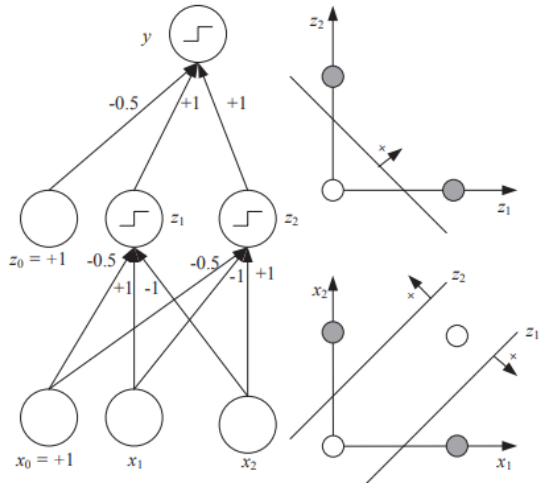


Fig. 19: Problema XOR resuelto por un perceptron multicapa. Fuente: (Alpaydin, 2014)

2.3.2 Probabilistic neuronal network

Las redes neuronales probabilísticas (PNN) son generalmente usadas para resolver problemas de clasificación. En los últimos años se han estado usando para resolver problemas relacionados con el reconocimiento de patrones. Se han empleado PNN en áreas como el diagnóstico de enfermedades, detección de resistividad para antibióticos y la evaluación de la calidad del agua (Lotfi & Benyettou, 2014, p. 979).

La arquitectura de una red neuronal probabilística puede dividirse en 4 capas, una capa de entrada, una capa de patrones, una capa de sumas, y por último una capa de salida. Como se puede observar en la Fig. 20, la primera capa contiene las unidades de entrada que distribuyen los parámetros de entrada a todas las unidades de patrones, no se realiza ningún cálculo en esta capa. La segunda capa tiene una neurona por cada elemento en el set de entrenamiento, al recibir el patrón x desde la capa de entrada, la neurona x_{ij} computa un valor de salida, usando la fórmula:

$$\varphi_{ij}(X) = \frac{1}{(2\pi)^{p/2}\sigma^d} exp\left[-\frac{(x - x_{ij})^t(x - x_{ij})}{2\sigma^2}\right]$$

$$x_{ij} = vector\ de\ la\ Neurona\ i\ que\ pertence\ a\ la\ categoria\ j$$

$$p = dimension\ del\ vector\ de\ patrones\ x$$

$$\sigma = parametro\ de\ suavizado$$

En la tercera capa existe una neurona por cada categoría en el set de entrenamiento, estas calculan la posibilidad de que el patrón x sea clasificado en la categoría C_i sumando las salidas de todas las neuronas de la segunda capa que pertenezcan a su categoría y obteniendo la media.

$$P_{ij}(x) = \frac{1}{(2\pi)^{p/2}\sigma^2}\frac{1}{Ni}\sum_{i=1}^{m} exp\left[-\frac{(x-x_{ij})^t(x-x_{ij})}{2\sigma^2}\right]$$

$$N_i = numero\ de\ muestras\ que\ pertenecen\ a\ la\ clase\ C_i$$

Por último la capa de salida toma la decisión respecto a que categoría pertenece el nuevo elemento, para calcular esto simplemente compara los resultados de la tercera capa y escoge el de mayor valor, en otras palabras escoge la categoría más probable (Mao, Tan, & Ser, 2000, p. 1009,1010).

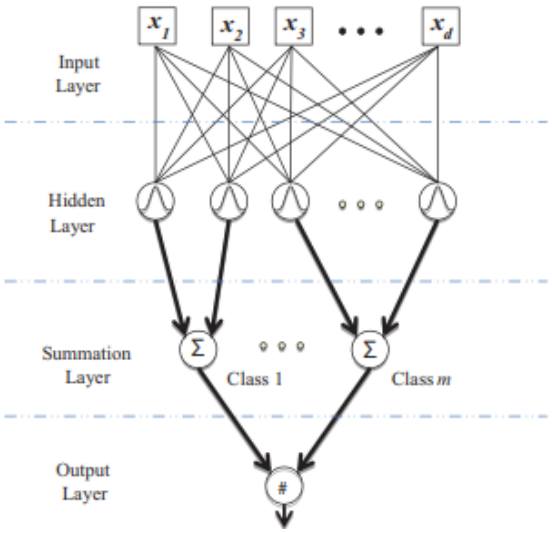


Fig. 20: Arquitectura de una red neuronal probabilística (PNN). Fuente: (Lotfi & Benyettou, 2014)

Las PNN tienen la ventaja de ser más rápidas de entrenar y de dar predicciones más precisas comparadas al Perceptrón multicapa (MLP). Además son relativamente insensibles a valores atípicos. Un problema de PNN es que es más lenta que MLP al clasificar nuevos casos y además necesita más memoria para guardar el modelo ya que necesitamos crear una neurona por cada caso presente en el set de entrenamiento (Sawant S. & Preeti S., 2015, p. 282).

2.3.3 K-Nearest Neighbours

El algoritmo de K-nearest Neighbours (K-NN), es un método no paramétrico usado para realizar clasificación y regresión. Es además un tipo aprendizaje basado en instancias o aprendizaje vago (Lazy Learning). El algoritmo de K-NN es uno de los algoritmos de *Machine Learning* más simples en comparación a los demás algoritmos.

La etapa de entrenamiento para este algoritmo solo consiste en guardar los vectores de características que existen en el set de entrenamiento, junto a sus respectivas etiquetas.

En la etapa de clasificación, se requiere clasificar un vector sin etiqueta x , esto se logra utilizando la constante k, la cual es asignada por un usuario. Utilizando esta constante se procede a buscar los k vectores de características más cercanos a p.(Parsian, 2015, Chapter 13) Una forma de determinar la distancia entre vectores es calcular la distancia Euclidiana, sean entonces los atributos para un vector de características $v = (v_1, v_2 \dots, v_n)$ y para el vector sin clasificar $x = (x_1, x_2 \dots, x\_n)$ la distancia es:

$$Distancia(v, x) = \sqrt{(v_1 - x_1)^2 + (v_2 - x_2)^2 + \cdots + (v_n - x_n)^2}$$

Una vez que se encuentran los k vectores, se procede a determinar la clase del vector p simplemente asignándole la clase a la que pertenezca la mayoría de los k vectores encontrados.

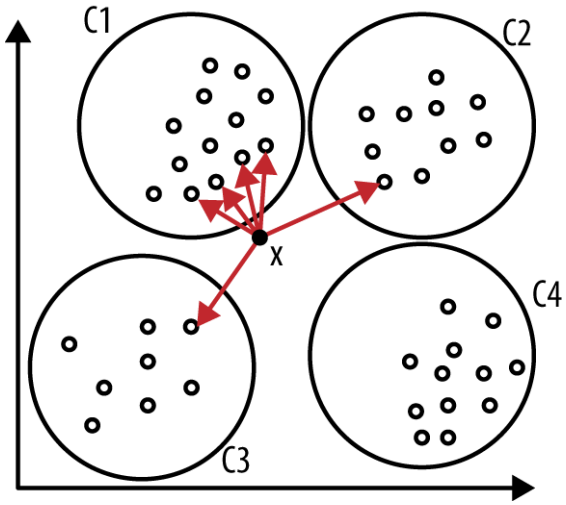


Fig. 21: Clasificación por K-NN donde K=6, de los 6 vecinos más cercanos al punto X, 4 pertenecen a C1, por lo que se clasifica X como un punto miembro de la clase C1. Fuente: (Parsian, 2015, Chapter 13)

Capítulo 3: Algoritmos Investigados

En este capítulo se presentan los 3 algoritmos que fueron estudiados con el fin de realizar este proyecto de título. Es a partir del estudio de estos que se identificaron técnicas y se aprendieron conceptos que permitieron el desarrollo de este proyecto.

Se presentan a continuacion las investigaciones que realizaron cada uno de los autores estudiados, en donde se incluye la explicación de conceptos, pasos a seguir para la realización de los algoritmos, y los resultados finales que estos obtuvieron, entre otros.

3.1 A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network

El trabajo descrito a continuación corresponde a un resumen de lo publicado por los autores en la publicación original. (Gang Wu et al., 2007)

En este trabajo se emplean Redes Neuronales Probabilísticas (PNN) con técnicas de procesado de imágenes y datos para implementar un reconocimiento automático de hojas de propósito general para la clasificación de plantas. 12 características de las hojas de árboles son extraídas y ortogonalizadas en 5 variables principales que consisten en el vector de entrada del PNN. El PNN es entrenado por 1800 hojas para clasificar 32 tipos de plantas con una precisión superior al 90%.

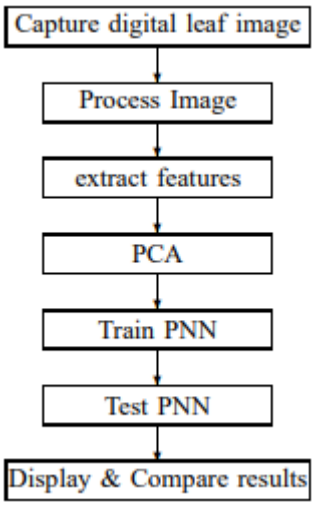


Fig. 22: Diagrama de flujo del enfoque propuesto.

3.1.1 Pre-procesamiento de la imagen

Conversión de imagen RGB a imagen binaria

Las imágenes de las hojas de árboles deben deber ser de tamaño de 800x600 pixeles. No existe una restricción respecto a la dirección de las hojas al momento de fotografiarlas.

Dicho esto, primero una imagen es convertida de RGB a escala de grises utilizando la formula Eq.1 para convertir el valor de un pixel RGB a un valor de escala de grises.

$$gray = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

R, G y B corresponden a los canales rojo, verde y azul de la imagen RGB.

Para convertir la imagen de escala de gris a una imagen binaria, se debe determinar un umbral, esto se hace creando un histograma de los valores de la imagen RGB. Se acumularon los valores de los pixeles de color R, G, y B para 3000 hojas y se dividieron por 3000 (el número de las hojas).

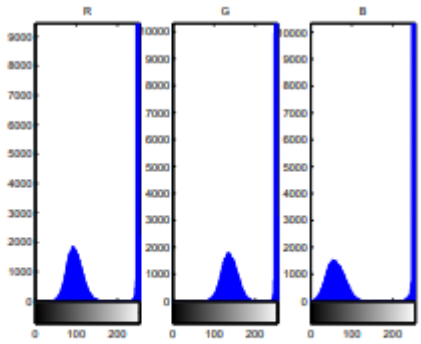


Fig. 2. RGB histogram

El punto más bajo entre 2 picos es el valor 242. Por lo que se escogió el valor 0.95 (242/255=0.949) como el nivel. La imagen de salida remplaza todos los pixeles con un valor superior al nivel definido con el valor 1, lo pixeles restantes toman el valor de 0.

Por último se aplica un filtro de promedio rectangular (*rectangular averaging filter*) de 3x3 para suavizar la imagen.

Mejora de bordes

Utilizando un filtro Laplaciano sobre la imagen se puede obtener el borde de la hoja de árbol. La imagen de salida produce una imagen negra con fondos blancos, por conveniencia en este trabajo se invierten los colores de esta imagen para obtener una imagen blanca donde se puede observar el borde de la hoja resaltado con el color negro.

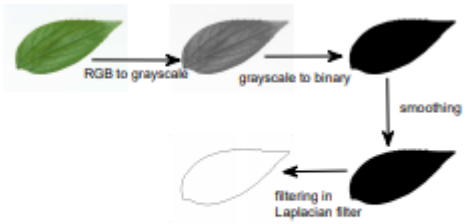


Fig. 3. A pre-processing example

3.1.2 Extracción de características

12 características morfológicas digitales comúnmente utilizadas son derivas a partir de la extracción de 5 características básicas.

Características Geométricas Básicas

1) **Diámetro: La** distancia más larga entre 2 puntos del margen de la hoja.
2) **Largo Fisiológico** (L_p): La distancia entre los puntos terminales de la vena principal. Es la única característica que debe ser indicada por una persona.
3) **Ancho Fisiológico** (W_p): La distancia más larga de la línea ortogonal a L_p.

La fig. 4 ilustra la relación entre el largo fisiológico y el ancho fisiológico.

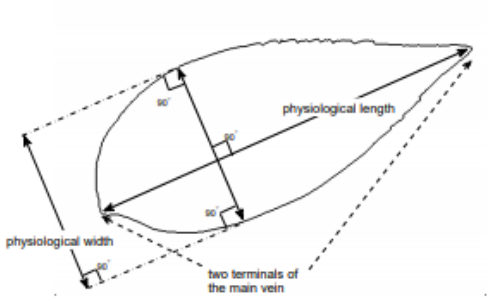


Fig. 4. Relationship between Physiological Length and Physiological Width

4) **Área de la hoja** (A): La cantidad de pixeles de valor 1 en la imagen de la hoja suavizada.
5) **Perímetro de la hoja** (P): El número de pixeles de valor 1 en el margen de la hoja.

Características morfológicas digitales

A partir de las 5 características calculadas anteriormente se pueden calcular 12 características morfológicas digitales usadas para el reconocimiento de hojas.

1) **Factor de suavidad:** Se usa el efecto del ruido en la imagen para describir la suavidad de la imagen de la hoja de árbol. El factor de suavidad está definido

como la proporción entre el área de la imagen de la hoja de árbol suavizada por un rectangular averaging filter de 5x5 y una suavizada por un *rectangular averaging filter* de 2x2.

2) **Relación de aspecto:** La proporción entre L_p y W_p, por lo tanto es igual a $\frac{L_p}{W_p}$.
3) **Factor de Forma:** Se usa para describir la diferencia entre una hoja y un círculo. Se define como $4\pi A/P^2$
4) **Rectangularidad:** Describe la similitud entre una hoja y un rectángulo. Se calcula como $L_p \times W_p/A$
5) **Factor de estrechez:** La proporción entre el diámetro D y el largo fisiológico. D/L_p
6) **Relación Perimetral de diámetro:** Proporción entre le perímetro y el diámetro. Es equivalente a $\frac{P}{D}$
7) **Relación perimetral de longitud fisiológica y anchura fisiológica:** Proporción entre el perímetro de la hoja y la suma de W_p y L_p, se calcula como $P/(L_p + W_p)$
8) **Características de las venas:** Se realiza apertura morfológica sobre la imagen de escala de gris, utilizando un elemento estructural con forma de disco plano de tamaños 1, 2, 3 y 4. El resultado de la operación parecen las venas de la hoja. Las áreas de los pixeles que quedan son denominadas A_{v1}, A_{v2}, A_{v3} y A_{v4}. Las últimas 5 características se obtienen calculando $\frac{A_{v1}}{A}, \frac{A_{v2}}{A}, \frac{A_{v3}}{A}, \frac{A_{v4}}{A}$ y $\frac{A_{v4}}{A_{v1}}$.

3.1.3 Esquema propuesto

Análisis de componentes principales (PCA)

Para reducir el tamaño del vector de entrada a la red neuronal, se usa PCA para ortogonalizar las 12 características. PCA transforma los datos a un nuevo sistema de coordenadas, tal que la varianza más grande de cualquier proyección de los datos este en la primera coordenada, la segunda varianza más grande este en la segundo coordenada, ect. Cada coordenada se le llama un componente principal.

Para este paper la contribución de los 5 componentes principales es del 93,6%. Para tener un equilibrio entre precisión y complejidad computacional se adoptan 5 componentes principales.

Resultados Experimentales

Para cada especie de plantas, 10 hojas del set de testeo son utilizadas para testear la precisión del algoritmo. El número de imágenes usadas en el entrenamiento y el número de hojas que fueron detectadas incorrectamente son listadas por especie en la Tabla a. La precisión promedio es de 90,312%.

Scientific Name(in Latin)	Common Name	training samples	number of incorrect recognition
Phyllostachys edulis (Carr.) Houz.	pubescent bamboo	58	0
Aesculus chinensis	Chinese horse chestnut	63	0
Berberis anhweiensis Ahrendt	Anhui Barberry	58	0
Cercis chinensis	Chinese redbud	72	1
Indigofera tinctoria L.	true indigo	72	0
Acer Dalmatum	Japanese maple	53	0
Phoebe zhennan S. Lee & F.N. Wei	Nanmu	60	1
Kalopanax septemlobus (Thunb. ex A.Murr.) Koidz	castor aralia	51	0
Cinnamomum japonicum Siebold ex Nakai	Japan Cinnamon	51	2
Koelreuteria paniculata Laxm.	goldenrain tree	57	0
Ilex macrocarpa	holly	50	0
Pittosporum tobira (Thunb.) Ait. f.	Japanese cheesewood	61	1
Chimonanthus praecox L.	wintersweet	51	2
Cinnamomum camphora (L.) J. Presl	camphortree	61	3
Viburnum awabuki	Japanese Viburnum	58	2
Osmanthus fragrans Lour.	sweet osmanthus	55	5
Cedrus deodara (Roxb.) G. Don	deodar	65	3
Ginkgo biloba L.	ginkgo, maidenhair tree	57	0
Lagerstroemia indica (L.) Pers.	Crepe myrtle	57	0
Nerium oleander L.	oleander	61	0
Podocarpus macrophyllus (Thunb.) Sweet	yew plum pine	60	0
Prunus ×yedoensis Matsumura	Japanese Flowering Cherry	50	0
Ligustrum lucidum Ait. f.	Chinese Privet	52	1
Tonna sinensis M. Roem.	Chinese Toon	58	1
Prunus persica (L.) Batsch	peach	50	2
Manglietia fordiana Oliv.	Ford Woodlotus	50	3
Acer buergerianum Miq.	trident maple	50	1
Mahonia bealei (Fortune) Carr.	Beale's barberry	50	0
Magnolia grandiflora L.	southern magnolia	50	0
Populus ×canadensis Moench	Carolina poplar	58	3
Liriodendron chinense (Hemsl.) Sarg.	Chinese tulip tree	50	0
Citrus reticulata Blanco	tangerine	51	0

Tabla a: Detalles respecto a las especies en el dataset (primera y segunda columna), el número de imágenes a entrenar por especie (tercera columna) y el número de imágenes identificadas incorrectamente (cuarta columna).

3.2 Plant Leaf Recognition using Shape based Features and Neural Network classifiers

El trabajo descrito a continuación corresponde a un resumen de lo publicado por los autores en la publicación original. (Chaki & Parekh, 2011)

Se propone un enfoque para la detección automática de 3 especies de plantas analizando sus formas que son obtenidas desde una colección de hojas, usando características basadas en los momentos invariables y los acercamientos basados en el modelo Centroide-Radio, usando varios tipos de clasificadores basados redes neuronales.

3.2.1 Extracción de Características

Momentos invariables

M-K Hu (M.-K. Hu, 1962) propuso 7 características que pueden ser calculadas a partir de los momentos, que pueden ser usadas para describir formas, estas características son invariantes a la rotación, traslación y el escalado.

Modelo Centroide-Radio

Propone un modelo centroide-radio para estimar las formas de los objetos en las imágenes. Una forma es definida como un área negra sobre un fondo blanco. Cada uno de los pixeles tiene una coordenada (x, y). El centroide de una forma está ubicada en la posición (C_x, C_y) que representa respectivamente la media de las coordenadas x e y correspondiente a todos los pixeles negros. El contorno de una forma corresponde a una serie de puntos de contorno, y un punto de contorno es un punto negro que tiene como vecino a un pixel blanco. Se puede definir un radio como una línea recta que va desde el centroide hasta un punto de contorno. Conociendo esto, en el modelo Centroide-Radio las distancias Euclidianas de los radios son capturadas en intervalos regulares como descriptores de forma.

Sea θ un intervalo regular (medido en grados) ubicado entre los radios (ver Fig. 23). El número de intervalos está dado por $k = \frac{360}{\theta}$. Una vez obtenidas las distancias de los radios, estas son normalizadas dividiéndolas por el radio de mayor tamaño obtenido.

Suponiendo que los intervalos se toman en sentido del reloj partiendo en el eje x. Entonces el descriptor de la forma puede ser representado por el vector:

$$S = \{r_0, r_\theta, r_{2\theta}, \dots, r_{(k-1)\theta}\}$$

Con el número suficiente de radios formas distintas pueden ser diferenciadas.

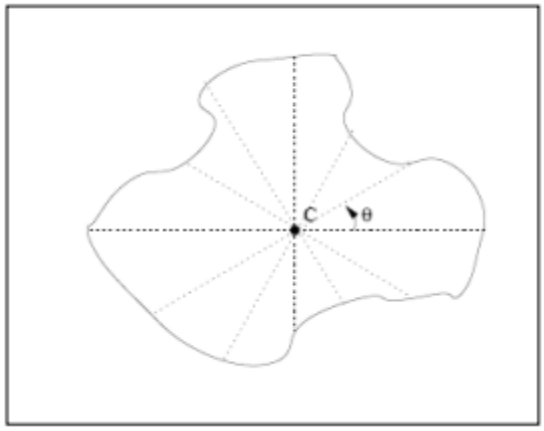


Fig. 23: Distancias centroide-radio obtenidas en intervalos de θ grados

3.2.2 Experimentación y resultados

La experimentación se realiza usando 180 hojas de la base de datos de Plantscan. El dataset es dividido en 3 clases (ver Fig. 24) A (Pittosporum Tobira), B (Betula Pendula) y C (Cercis Siliquastrum) cada uno formado por 60 imágenes. Las imágenes tienen las dimensiones 350x350 y estan en formato JPG. Un total de 90 imágenes forman el set de entrenamiento (T), mientras que las 90 imágenes restantes forman el set de testeo (S).

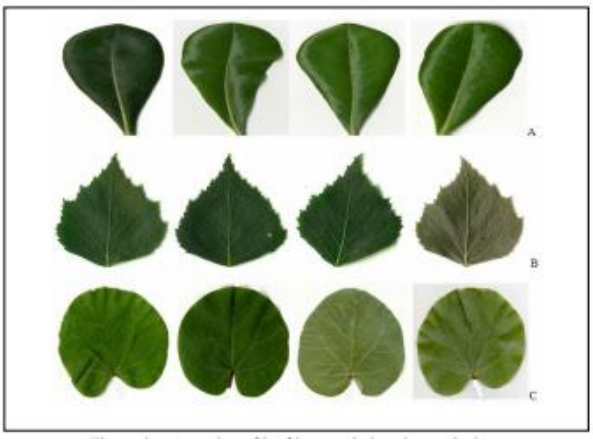


Fig. 24: Algunas de las hojas correspondientes a las 3 clases de hojas de árboles en el dataset.

Para computar las tasas de reconocimiento, comparaciones entre el set de entrenamiento y set de testeo se hacen usando Perceptrones Multicapa. Una tabla con leyendas usadas en este trabajo son mostradas en la Tabla b.

Legend	Meaning	Legend	Meaning
AT	Class-A Training set	AS	Class-A Testing set
BT	Class-B Training set	BS	Class-B Testing set
CT	Class-C Training set	CS	Class-C Testing set

Tabla b: Leyendas

Representaciones de momentos invariantes (M-I).

Los primeros 4 momentos centrales normalizados M1, M2, M3 y M4 son computados. Estas características fueron alimentadas en forma individual o en combinaciones a un clasificador basado en redes neuronales para estudiar cuál de estas configuraciones arroja los mejores resultados.

Características Individuales:

Las características M1, M2, M3 y M4 son usadas de forma individual para la etapa de entrenamiento y testeo. Los resultados del procedimiento están en la Tabla c. La primera columna indica la característica usada, la segunda muestra la configuración de la red neuronal, 1-3-3 indica: 1 unidad de entrada para la característica única, 3 unidades en la *hidden layer* y 3 unidades en la capa de salida, 1 unidad por cada clase a distinguir.

La tercera, cuarta y quinta columna indican el porcentaje de reconocimiento correcto para las clases A, B y C, además del número de imágenes correctamente identificadas para cada clase, la sexta columna muestra la precisión total respecto a las 3 clases y la última columna indica el error cuadrático medio obtenido durante el entrenamiento de la red neuronal.

F	NNC	A	B	C	O	MSE
M_1	1-3-3	70 (21/30)	96.7 (29/30)	100 (30/30)	88.9 (80/90)	0.06
M_2	1-3-3	73.3	86.7	20	60	0.17
M_3	1-3-3	10	100	70	60	0.15
M_4	1-3-3	86.7	46.7	50	61.1	0.16

Tabla c: Resultados usando características M-I individuales.

Como se indica en la imagen la característica individual M1 da los mejores resultados con un 88,9% de las imágenes correctamente identificadas.

Características Combinadas

Para mejorar los resultados obtenidos utilizando las características individuales, se usan características combinadas en un espacio de 2 dimensiones de forma M1-M2, M1-M3, M1-4. M1 es usada en todas la combinaciones ya que se demostró que de todas la características da los mejores resultados. Los resultados pueden observarse en la Tabla d.

F	NNC	A	B	C	O	MSE
M_1-M_2	2-3-3	90	100	93.3	94.4	0.058
M_1-M_3	2-3-3	90 (27/30)	96.6 (29/30)	100 (27/30)	95.5 (86/90)	0.053
M_1-M_4	2-3-3	86.6	96.6	93.3	92.2	0.068

Tabla d: Resultados obtenidos combinando 2 características M-I.

La tabla anterior indica que la combinación de 2 características que proporciona los mejores resultados es M1-M3 con un 95.5% de imágenes correctamente identificadas.

Por último se realiza una combinación de características en un espacio de 3 dimensiones: M1-M2-M3, M1-M2-M4, M1-M3-M4. Como puede observarse en la Tabla e. La combinación de características M1-M3-M4 entrega los mejores resultados en esta instancia, con un 93.3% de imágenes correctamente identificadas.

F	NNC	A	B	C	O	MSE
M_1-M_2-M_3	3-30-3	83.3	100	93.3	92.2	0.028
M_1-M_2-M_4	3-30-3	90	86.7	90	88.9	0.005
M_1-M_3-M_4	3-30-3	90 (27/30)	100 (30/30)	90 (27/30)	93.3 (84/90)	0.005

Tabla e: Resultados obtenidos combinando 3 características M-I.

Representaciones Centroide-radio

Las imágenes en el dataset son convertidas a imágenes binarias, y se utiliza el detector de bordes de Canny para identificar los contornos presentes. El centroide es calculado a partir de la media de todos los pixeles que conforman el borde.

Para cada pixel de borde se calcula el ángulo que este forma con el centroide, estos ángulos junto a los pixeles de borde usados para calcularlos son almacenados en un *array.* Desde este *array* se identifican 36 coordenadas que forman ángulos en intervalos de 10 grados, desde 0 a 359 grados. La longitud radial de estos 36 puntos son calculados usando la distancia Euclidiana. Se procede a normalizar las longitudes obtenidas al rango [0,1] dividiendo todos los valores por la longitud radial de mayor distancia, estos valores normalizados son almacenados de menor a mayor.

En la Fig. 25 se puede observar una representación visual de la imagen de una hoja, los bordes obtenidos luego del proceso de detección de bordes, la ubicación del centroide y de los pixeles de borde, y una gráfica de las distancias radiales normalizadas.

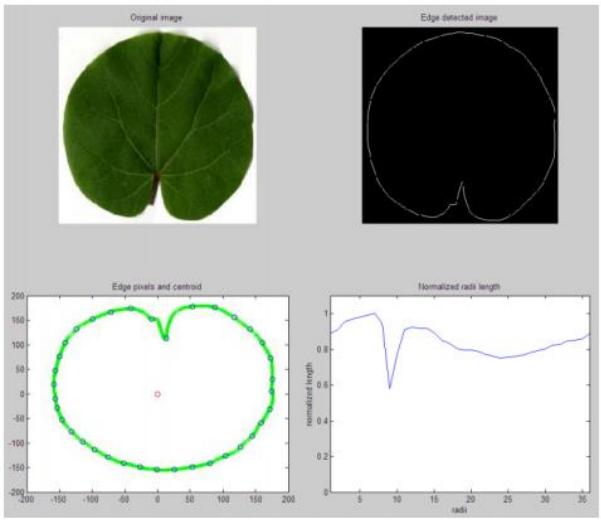


Fig. 25: Imagen de hoja (superior izquierda). Imagen de los bordes obtenidos por Canny (superior derecha). Imagen del centroide y puntos para calcular las longitudes radiales (inferior izquierda). Longitudes radiales normalizadas (inferior derecha).

Los resultados del proceso de clasificación pueden observarse en laTabla f. Como se puede apreciar la utilización del elemento de 36 longitudes radiales logra una precisión del 100%, con todas las imágenes correctamente identificadas. La convergencia se logra en 38280 epochs

F	NNC	A	B	C	O	MSE
36-element C-R vector	36-30-3	100 (30/30)	100 (30/30)	100 (30/30)	100 (90/90)	0.005

Tabla f: Resultados obtenidos usando características C-R.

Representación Hibrida

Para estudiar la precisión de un set de características hibridas se realizó una combinación de características Centroide-Radio y Momentos invariables, formando así un vector (1) de 38 elementos: 36 elementos del modelo Centroide-Radio y dos elementos (M1-M3) del modelo de momentos invariables, y un vector (2) de 39 elementos: 36 elementos del modelo Centroide-Radio y dos elementos (M1-M3-M4) del modelo de momentos invariables. Estos vectores fueron alimentados a una red neuronal y los resultados de esto pueden observarse en la Tabla g.

F	NNC	A	B	C	O	MSE
38-element hybrid vector	38-30-3	100 (30/30)	100 (30/30)	100 (30/30)	100 (90/90)	0.005
39-element hybrid vector	39-30-3	100	100	93.3	97.7	0.005

Tabla g: Resultados obtenidos usando características hibridas.

Se puede observar que la precisión total para la correcta identificación de imágenes es de 100% al usar el vector (1) de 38 elementos. A pesar de la similitud de este resultado con los resultados arrojados por el vector de 36 distancias radiales, la convergencia en este caso se logra mucho más rápido, en tan solo 20706 epochs.

Respecto a la implementación del sistema, la computación del elemento de 36 distancias radiales para 30 imágenes toma aproximadamente 35 segundos, mientras que la computación de los 4 primeros momentos para 30 imágenes toma cerca de 300 segundos, en un sistema P4 con 1GB de RAM y 2.66 GHz. La convergencia de la red neuronal para el vector híbrido de 38 elementos toma cerca de 2 minutos para un set de entrenamiento de 30 imágenes. El sistema es estable para las 3 categorías de hojas de árboles, tomando los mismos tiempos para todo el dataset.

La discriminación entre 3 clases de hojas de árboles se logró usando una variedad de acercamientos para encontrar los resultados más óptimos. El estudio revelo que para el acercamiento M-I utilizar características combinadas dan mejores resultado que usar características individuales. La precisión lograda por el método C-R es superior a la lograda por las características individuales de M-I y comparables a las características combinadas de M-I.

Se encontró que el rendimiento mejora cuando las características C-R son combinadas con características de M-I, logrando una precisión del 100% al igual que al utilizar las características de C-R, pero solo necesitando casi la mitad de los epochs.

Los resultados de las precisiones obtenidas por los distintos métodos se pueden observar en la Tabla h.

F	M-I I	M-I J2	M-I J3	C-R	H
% Accuracy	88.9	95.5	93.3	100	100

Tabla h: Resultados obtenidos utilizando distintas características.

3.3 Multiscale Distance Matrix for Fast Plant Leaf Recognition

El trabajo descrito a continuación corresponde a un resumen de lo publicado por los autores en la publicación original. (R.-X. Hu et al., 2012)

En este trabajo se propone un descriptor de forma basado en contornos denominado como Multiscale Ditance matrix o MDM, este descriptor es una matriz que captura la estructura geométrica de una forma y es invariante a la rotación, traslado y el escalado.

3.3.1 Extracción de Características

MDM

Se define una forma O como un subconjunto conectado y cerrado perteneciente a R^2. Dado n puntos $p_1, p_2, p_3 \dots, p_n$ del contorno de una forma O se puede construir una matriz de distancias D de tamaño $n \times n$, donde $\underline{D_{ij}}$ es la distancia Euclidiana entre el punto p_i y el punto p_j. Esta matriz es simétrica, con todos los elementos pertenecientes a la diagonal teniendo un valor de 0.

Basándose en la matriz D, se puede computar la MDM siguiendo los siguientes pasos:

1) Cada columna de la matriz D es desplazada circularmente hacia arriba de forma que su primer elemento sea 0. Este procedimiento resulta en la matriz D_m en donde la primera fila está compuesta de ceros.
2) Para cada fila de D_m se orden sus elementos de forma ascendente, generando así la matriz D_{ms}.

3) Para D_{ms} se remueve la primera y las últimas $[\frac{n-1}{2}]$ filas, construyendo así la matriz MDM básica.

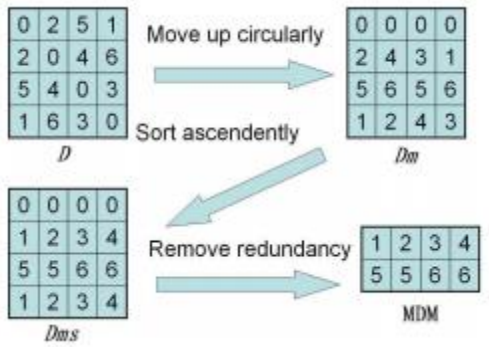


Fig. 26: Representación visual de la construcción de la MDM.

Un ejemplo de la construcción de la matriz MDM se puede observar en la Fig. 26.

Cada una de las filas de la matriz D_{ms} captura un cierto rango de propiedades geométricas de una forma. Los valores de la primera fila son todos ceros, estos valores representan la distancia entre cada punto y sí mismo. La segunda fila indica las distancias entre los puntos y los puntos siguientes a estos, lo cual captura las propiedades geométricas más finas de la forma. El resto de las filas de D_{ms} capturan propiedades geométricas de mayor grosor. Descendiendo por la matriz las filas pueden capturar propiedades geométricas cada vez más gruesas hasta llegar a la fila numero $\left[\frac{n}{2}\right]$, en donde se encuentran las propiedades geométricas de mayor grosor. Se puede observar que los valores de la segunda fila de la matriz D_{ms} son exactamente iguales a los valores de su última fila, estos valores representan la distancia entre puntos en un intervalo de 1. Por razones similares las últimas $[\frac{n-1}{2}]$ filas son redundantes y por lo tanto son removidas de la matriz D_{ms}.

En la Fig. 27, se puede observar la representación visual de algunas filas de una MDM de tamaño 32x64 creada a partir de 64 puntos extraídos a partir de la forma de la hoja. Las imágenes *a), c), d)* y *f)* ofrecen una visualización de las filas 1, 8, 16 y 32 de la MDM.

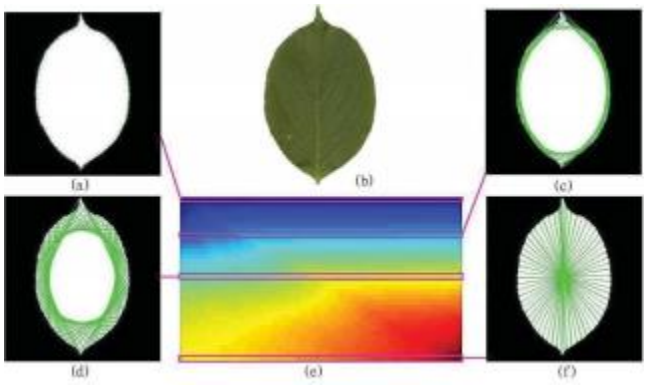


Fig. 27: b) Imagen de la hoja de árbol. e) MDM extraída desde la hoja. Las imágenes a), c), d) y e) representan las filas 1, 8, 16 y 32 de la MDM, respectivamente.

Dado el contorno de una forma en una imagen, la traslación de tal imagen no afecta las distancias entre los puntos que conforman el contorno, por lo que la MDM es invariante a la traslación. La rotación de una imagen cambia el orden de los puntos del contorno. Al computar la MDM la etapa de ordenamiento ignora el orden de los puntos, siendo así la MDM invariante a la rotación. El problema con esta etapa es que se pierde cierta información discriminativa de la matriz de distancias original. Para conservar esta información, para D_m se computan las diferencias de todos los elementos vecinos en cada fila, en otras palabras para cada fila de D_m con elementos $d1, d2, \dots d_n$ se calcula $d_2 - d_1, d_3 - d_2, d_4 - d_1 \dots, d_1 - d_n$, para luego ordenar estas diferencias de forma ascendente. Las diferencias resultantes forman un matriz de diferencias la cual es añadida al fondo de la MDM para así construir una matriz MDM-CD de doble tamaño que la MDM.

Los contornos de una forma, en dirección del reloj o contrarreloj producen columnas en un orden distinto, pero esto no afecta a la matriz D_{ms}, por lo que la MDM es invariante a la simetría bilateral de la forma.

Para lograr que MDM sea invariante al escalado, se puede crear alguna de las 5 versiones normalizadas de la MDM, estas son MDM-M, MDM-A, MDM-C, MDM-RA y MDM-CR. Las primeras 3 matrices se obtienen normalizando la MDM respecto al mayor valor en la matriz (MDM-M), el valor medio de la matriz (MDM-A), y la distancia más larga desde el centroide de la forma hasta alguno de los puntos que conforman el contorno (MDM-C). MDM-RM y MDM-RA se obtienen normalizando cada fila de la MDM por el valor máximo y el valor medio de dichas filas, respectivamente.

Luego de extraer la MDM se aplican las técnicas de reducción de dimensionalidad, DNM (Decomposed Newton Method) y MMC (Maximum Margin Criterion). Por último se usa la distancia Euclidiana y la regla de Nearest Neighbors para la clasificación.

Extensión de la MDM

La MDM presentada hasta ahora usa la distancia Euclidiana al construir la matriz D. Se puede extender la MDM utilizando otros tipos de medidas para la distancia. Una de estas es la distancia interna (inner-distance) usada en IDSC. La distancia interna entre 2 puntos es la longitud del camino más corto (al interior de la forma) que une a ambos puntos. Al construir la MDM se puede reemplazar la distancia Euclidiana por la distancia interna para crear la MDM-ID.

Otras medidas de disimilitud entre un par de puntos pueden ser usados para extender los descriptores de la MDM. Una de estas medidas es la matriz costo-distancia basada en la comparación de histogramas como se usa en [2,10]. Usando esta medida se puede computar la Multiscale Cost Matrix o (MCM).

3.3.2 Experimentos

Para validar los métodos propuestos se aplican sobre tareas de clasificación de hojas de árboles, usando 2 datasets: the Swedish Leaf dataset and the ICL Leaf dataset.

Fig. 28: Ejemplos de las hojas que forman el Swedish Leaf Dataset.

Fig. 29: Ejemplos de las hojas que forman el Clean Swedish Dataset.

A continuación se compara la capacidad de reconocimiento de los métodos propuestos, junto a otros métodos como IDSC y el descriptor de forma de Fourier.

Para cada figura se seleccionan uniformemente 128 puntos desde su contorno para construir la MDM. Estos puntos serán usados por IDSC igualmente. Antes de aplicar los métodos de reducción de dimensionalidad se debe transformar la MDM a un vector, la dimensión final del vector de características se determina experimentalmente entre 20 y 50.

Resultados experimentales usando Swedish Leaf Dataset

Swedish Leaf Dataset contiene hojas de árboles de 15 especies diferentes, 75 hojas por especie. 25 hojas son usadas para entrenamiento y 50 para el testeo del modelo. Para este dataset el método de reducción de dimensionalidad que se usa es DNM.

IDSC [10]	MDM-M	MDM-A	MDM-RM	MDM-RA	MDM-C
	92.53	92.40	91.20	91.60	92.40
94.13	MDM-CD-M	MDM-CD-A	MDM-CD-RM	MDM-CD-RA	MDM-CD-C
	93.33	92.67	92.13	92.53	93.20
FD [10]	MDM-ID-M	MDM-ID-A	MDM-ID-RM	MDM-ID-RA	MDM-ID-C
	92.67	92.67	91.60	92.13	93.07
89.60	MDM-ID-CD-M	MDM-ID-CD-A	MDM-ID-CD-RM	MDM-ID-CD-RA	MDM-ID-CD-C
	93.60	92.80	92.53	92.67	93.47

Tabla i: Tasas de reconocimiento logradas por los distintos métodos empleados sobre el Sweadish Leaf Dataset.

En la Tabla i pueden observarse los porcentajes de imágenes de hojas que fueron clasificadas correctamente, usando los distintos métodos descritos anteriormente. Primero, se puede señalar que la precisión de las versiones de MDM con la etiqueta "-CD" es superior a la de las versiones básicas. Segundo La versiones con "-RA" y "-RM" tienen un peor desempeño que la versiones con "-A", "-M" y "-C", lo que indica que normalizar la matriz en cada fila es menos efectivo que normalizar la matriz de forma global. Tercero en este caso las versiones con la etiqueta "-ID" tienen un mejor desempeño, lo que demuestra que utilizar la distancia interna tiene cierto mérito.

Como se puede ver en la Fig. 28 se debe notar que Swedish Leaf Dataset contiene imágenes de hojas junto a sus peciolos. A pesar de la información discriminativa que puedan contener, se consideró que entregaban información de poca confianza al extraerlos de la imagen.

Es por esta razón que se eliminan los peciolos de las imágenes para construir así un nuevo dataset limpio llamado "Clean Swidish Dataset" como se muestra en la Fig. 29 Sobre este dataset solo se aplicaran las versiones de MDM con la etiqueta "-CD", ya que como se demostró en el experimento anterior, estas versiones entregan los mejores resultados.

Los resultados obtenidos para este dataset se pueden observar en la Tabla j.

IDSC	MDM-CD-C	MDM-CD-M	MDM-CD-A
85.07	91.07	91.20	**91.33**
FD	MDM-ID-CD-C	MDM-ID-CD-M	MDM-ID-CD-A
83.60	89.60	90.80	90.67

Tabla j: Tasas de reconocimiento logradas por los distintos métodos empleados sobre el Clean Sweadish Leaf Dataset.

Se puede notar que el desempeño de IDSC baja drásticamente desde 94,13% a un 85.60%, lo que indica que este método se beneficiaba de la información discriminativa dada por los peciolos. El desempeño de todas las versiones de MDM baja igualmente, pero en menor medida. La versión MDM-CD-A obtiene el mejor desempeño con un 91.33% de imágenes de hojas de árboles correctamente identificadas. Por último se puede ver que las versiones de MDM con la etiqueta "-ID" tienen un peor desempeño comparadas con las versiones de MDM que usan la distancia Euclidiana.

Resultados experimentales usando ICL Leaf Dataset

Este dataset se compone de 6000 imágenes de hojas de árboles pertenecientes a 200 especies, con 30 imágenes por especie. El dataset es limpio lo que quiere decir que los peciolos han sido removidos de las hojas.

A partir de este dataset se construyen 3 subconjuntos A, B y C. Cada subconjunto está compuesto de 50 especies, con 30 hojas por cada especie. En el subconjunto A se escogieron especies que puedan ser fácilmente reconocidas por un humano. El subconjunto B tiene especies escogidas de forma aleatoria. Por último en el subconjunto C se escogieron especies de apariencia similar, pero que aún puedan ser diferenciables. Algunas de las especies escogidas para cada subconjunto se pueden observar en la Fig. 30

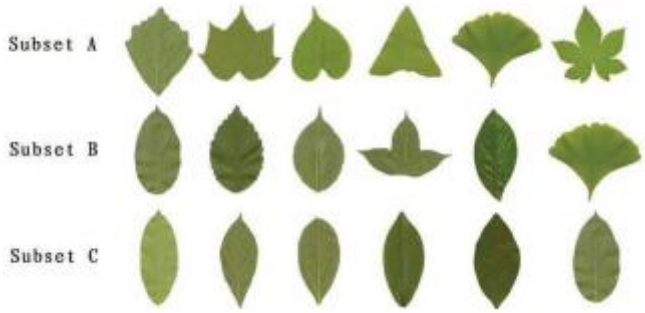


Fig. 30: Ejemplo de las hojas que conforman cada subconjunto.

Sabiendo esto es lógico esperar que el subconjunto B sea más difícil de reconocer que el conjunto A, y que le conjunto C es más difícil de reconocer que le subconjunto A y B. Se testean los métodos mencionado anteriormente en 2 situaciones. En la primera 15 imágenes de cada especie son usadas para el entrenamiento y las 15 restante para el testeo. En la segundo situación 29 imágenes de cada especie son usadas para el entrenamiento y 1 para el testeo. Estos test son repetidos 50 veces, para luego reportar las tasas de reconocimiento promedio. Se debe agregar que se utilizó MMC para realizar la reducción de dimensionalidad sobre las MDM obtenidas en este dataset.

Subset	Situations	IDSC	FD	MDM-CD-C	MDM-ID-CD-C	MCM-CD	MCM-ID-CD
A	30T15	**95.79**	92.90	95.47	94.97	95.73	94.91
	30T29	98.00	96.00	**98.20**	**98.20**	**98.20**	97.76
B	30T15	65.42	60.00	**68.75**	68.13	63.25	55.13
	30T29	70.32	66.56	**74.20**	74.08	70.44	59.64
C	30T15	63.99	59.37	73.88	**73.93**	67.27	63.95
	30T29	66.64	62.96	**80.88**	80.80	73.52	69.72

Tabla k: Tasas de reconocimiento logradas por los distintos métodos empleados sobre el ICL Plant Leaf Dataset..

Observando la Tabla k se puede ver que de todas las versiones de la MDM la versión con "-C" obtiene los mejores resultados en la mayoría de los casos. Al continuar analizando los valores de la tabla se pueden añadir las siguientes afirmaciones: Primero en el subconjunto A, la MCM entrega los mejores resultados entre los métodos descritos en este trabajo, mientras que en el subconjunto B y C, la MCM tiene la peor tasa de reconocimiento en todas las situaciones. Lo que indica que métricas de distancia diferentes tienen distintas propiedades en situaciones diferentes. Segundo a pesar que el rendimiento de los métodos con MDM disminuye al pasar del subconjunto A al B, muestran una mejora en el subconjunto C, esto demuestra el potencial de MDM para clasificar correctamente formas distintas que se diferencian por pequeñas variaciones.

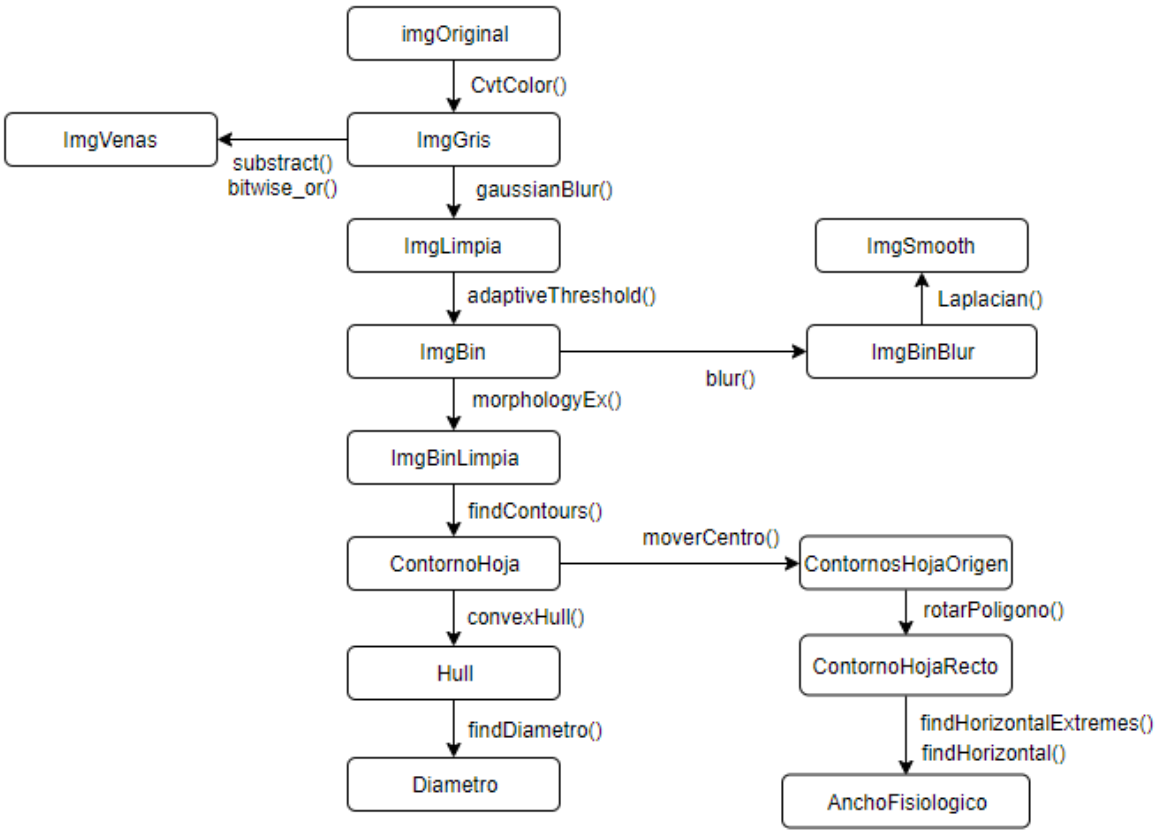
Capítulo 4: Implementaciones

En este capítulo se describe como se realizó la implementación de los 3 algoritmos estudiados, y de 2 algoritmos de elaboración propia. En las secciones 4.1, 4.2 y 4.3 se describen las modificaciones que se realizaron en algunos pasos con el fin de poder implementar los algoritmos en el lenguaje de programación Python, además se explica cada una de las funciones que fueron elaboradas, así como también sus parámetros. La sección 4.4 describe un algoritmo de elaboración propia, el cual es una modificación del algoritmo de la sección 4.2. Por último la sección 4.5 describe de forma resumida segundo algoritmo de elaboración propia basado en voto mayoritario.

4.1 A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network

4.1.1 Diagrama de Relaciones

El siguiente diagrama muestra de forma simplificada como se relacionan las funciones más relevantes que fueron utilizadas para la implementación del algoritmo. Estas funciones serán descritas de forma detallada a continuación del diagrama.



4.1.2 Etapa de Pre procesamiento

El primer paso del pre procesamiento de la imagen es convertir la imagen original RGB de entrada a una imagen en escala de grises. En el trabajo estudiado esto se realiza utilizando la función:

$$gray = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

El problema con esta función es que la conversión de los pixeles de RGB a escala de grises se realiza individualmente, esto quiere decir que esta función será llamada tantas veces como pixeles tenga la imagen original. Como es de esperar, durante la prueba de esta función, las imágenes demoraban muchos segundos en ser convertidas a escala de grises, por lo tanto se tomó la decisión de reemplazar esta función por la función de la librería "OpenCv": *cvtColor,* que a través de la función:

$cvtColor(src, dst, code, dstCn)$

Parámetros:

- **src:** Imagen de entrada.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **code:** Código para la conversión de espacio de color.
- **dstCn**: Número de canales en la imagen de salida. Si no se especifica se deriva de forma automática desde *src* y *code.*

Esta función permite transformar una imagen BGR a una imagen en escala de grises. La disminución en el tiempo de ejecución fue instantánea, las imágenes demoraban menos de un segundo en ser transformadas a escala de grises.

Para disminuir el ruido de imagen resultante se usa la función de OpenCv:

$$gaussianBlur(src, dst, ksize, sigmaX, SigmaY, borderType\ )$$

Parámetros:

- **src:** Imagen de entrada.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **ksize:** Tamaño de kernel Gaussiano a utilizar.
- **sigmaX:** Desviación estándar del kernel Gaussiano en la dirección de X.
- **sigmaY:** Desviación estándar del kernel Gaussiano en la dirección de Y. Si sigmaY es 0, entonces sigmaY es igual a sigmaX. Si ambos sigmas son 0, *sigmaX* es calculado a partir del ancho de *size,* mientras que *sigmaY* se calcula a partir de la altura de *ksize.*
- **borderType:** método de extrapolación de pixel.

Esta función aplica un filtro Gaussiano. Cabe mencionar que OpenCv carga las imágenes RGB en formato BGR, esto no tiene mayores repercusiones en la implementación de los algoritmos.

En el trabajo estudiado la conversión de la imagen en escala de grises a una imagen binaria se realizaba calculando un histograma para los pixeles rojos, verdes y azules como se explica en la sección 3.1 con el fin de determinar un umbral para realizar el proceso de Thresholding. Al igual que en paso anterior, este proceso de determinar el umbral toma mucho tiempo, ya que primero se debe calcular el número de pixeles rojos, verdes y azules que existen en todo el dataset. Otro problema que existe fue que al implementar este método se obtuvieron pésimos resultados y para varias imágenes de hojas de árboles no fue posible capturar la forma de la hoja durante la conversión a imagen binaria. La alternativa a realizar estos cálculos fue utilizar la función de "OpenCv":

$$adaptiveThreshold(src, dst, maxValue, adaptiveMethod, thresholdType, blockSize, C)$$

Parámetros:

- **src:** Imagen de entrada.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **maxValue:** Valor asignado a los pixeles sobre el umbral.
- **adaptativeMethod:** Cuál de los 2 algoritmos adaptivos se va a usar, ADAPTIVE_THRESH_MEAN_C o ADAPTIVE_THRESH_GAUSSIAN_C.
- **thresholdType:** El tipo de Thresholding, este puede ser THRESH_BINARY en donde los pixeles sobre el umbral son convertidos en negro y aquellos iguales o bajo el umbral son convertidos a blanco, o THRESH_BINARY_INV, donde ocurre lo contrario a la situación descrita anteriormente.
- **blockSize:** Tamaño de las regiones, sobre cada una de estas regiones se calculara un umbral individual.
- **C:** Constante sustraída desde la media o media ponderada.

Esta función permite dividir la imagen en varias regiones de igual tamaño y calcular un umbral para cada una de estas regiones. La imagen binaria obtenida por esta función captura de forma más precisa la forma de la hoja de árbol contenida en la imagen en escala de grises.

Es necesario mencionar que la imagen binaria obtenida contiene mucho ruido, por lo que se aplican funciones morfológicas de apertura y cerrado para limpiarla. Esto se realiza usando la función de OpenCv:

$$morphologyEx(src, op, kernel, dst, anchor, iterations, borderType, borderValue)$$

Parámetros:

- **src:** Imagen de entrada.
- **op:** Tipo de operación morfológica.
- **kernel:** Elemento estructural.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **anchor:**

- **iterations:** Número de veces que la operación es realizada.
- **borderType:** Método de extrapolación de pixel.
- **borderValue:** Valor de borde en caso de un borde constante.

La librería "OpenCv" permite obtener el contorno de una forma a partir de una imagen binaria usando la función *findContours.* Considerando que "OpenCv" provee estas facilidades, se descartan los 2 pasos mencionados al principio y se procede a extraer el contorno directamente de la imagen binaria usando la función

$$findContours(image, mode, method, contours, hierarchy, offset)$$

Parámetros:

- **image:** Imagen binaria de entrada.
- **mode:** Modo de extracción de contornos.
  - **cv2.CV_RETR_EXTERNAL:** Extraer los contornos más externos en la imagen.
  - **cv2.RETR_LIST:** Extraer todos los contornos presentes en la imagen, sin establecer relaciones jerárquicas.
  - **cv2.RETR_CCOMP:** Extraer todos los contornos en la imagen, y organizarlos en una jerarquía de 2 niveles. En el nivel superior se ubican los contornos externos (bordes) de los objetos, mientras que en el segundo nivel están los contornos internos de cada objeto.
  - **cv2.RETR_TREE:** Extraer todos los contornos en la imagen, y organizarlos en una jerarquía familiar completa.
- **method:** Método de aproximación de contornos.
  - **cv2.CHAIN_APPROX_NONE:** Se guardan todos los puntos que forman un contorno.
  - **cv2.CHAIN_APPROX_SIMPLE:** Comprime los segmentos verticales, horizontales y diagonales de un contorno, guardando solo los puntos de inicio y fin de tal segmento.
  - **cv2.CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS:** Aplica una de las formas del algoritmo de aproximación de cadenas de Teh-Chain.
- **contours:** Variable sobre la cual se guardan los contornos encontrados.
- **hierarchy:** Vector de salida opcional, contiene información adicional respecto a los contornos, como el número de estos encontrados.
- **offset:** Offset opcional usado para desplazar cada punto que forma un contorno.

4.1.3 Extracción de Características

Extracción de las características geométricas básicas

Diámetro

El contorno de una hoja de árbol está compuesto por una gran cantidad de pixeles, comparar las distancias de cada pixel con los demás pixeles tomara n^2 comparaciones para un contorno de tamaño n. Teniendo en cuenta esto primero se calcula la envolvente convexa del contorno usando la función de OpenCv:

$$convexHull(points, hull, clockwise, returnPoints)$$

Parámetros:

- **points:** Lista de puntos de 2 dimensiones.
- **hull:** La variable que contendrá a la envolvente convexa resultante.
- **clockwise:** Si su valor es *True,* entonces la envolvente se orienta en el sentido de las manecillas del reloj. En caso contrario (*False*) se orienta en contra de las manecillas del reloj.
- **returnPoints:** Parámetro en caso de una matriz. Cuando el parámetro es *True,* la función retorna los puntos de la envolvente convexa. En caso contrario devuelve el índice de los puntos que forman la envolvente convexa.

Con esta función se consigue simplificar así la cantidad de puntos a m. Se procede entonces a encontrar la distancia máxima entre 2 puntos en la envolvente convexa, realizando la misma comparación de cada pixel con los demás pixeles que fue descrita al principio, si bien esta operación tomara m^2 comparaciones, m es mucho menor que n por lo que el número de comparaciones para encontrar el diámetro es menor si se utiliza la envolvente convexa. Es claro que este no es el método más rápido u óptimo para calcular el diámetro, pero es sencillo de implementar y su tiempo de ejecución no toma una cantidad de tiempo notable.

Dicho esto se encuentra el diámetro usando la función de elaboración propia:

$$findDiametro(cnts)$$

Parámetros

- **cnts:** Lista de puntos 2D sobre la cual se buscara su diámetro.

Largo Fisiológico

El largo fisiológico se define como la distancia entre las 2 terminales de la vena principal que se encuentra presente en la hoja de árbol. Según lo señalado en el trabajo este paso debe realizarse de forma manual, con un usuario señalando las 2 terminales con la

ayuda de una interfaz gráfica. En esta implementación que se realiza para el proyecto de título se requiere que los algoritmos funcionen de forma automática sin la intervención de terceros. Como es sabido el diámetro de una hoja es la distancia más larga entre 2 puntos que se encuentran en el contorno de la hoja, debido a la forma alargada de las hojas de árboles escogidas, en la mayoría de los casos los puntos que forman el diámetro son muy cercanos a los puntos en ambos extremos de la vena principal. Es por esto que se tomó la decisión de que para este proyecto de título el Largo Fisiológico tomara el mismo valor que el diámetro de la hoja.

Ancho Fisiológico

Se puede definir el Ancho Fisiológico como la mayor distancia formada por una línea ortogonal a la línea formada por las 2 terminales de la vena principal. Para capturar el Ancho Fisiológico primero fue necesario mover el contorno de la hoja al origen del sistema de coordenadas, debido a que la siguiente función de rotación usa coordenadas polares, esto se logra con la función de elaboración propia:

$$moveraCentro(shape)$$

Parámetros:

- **shape:** Lista de puntos 2D a mover al centro del sistema de coordenadas.

Luego es necesario rotar el contorno de la hoja de árbol de manera que ambos puntos que forman el largo fisiológico quedaran alineados respecto al eje y. Esto se hace usando la función de elaboración propia:

$$rotarPoligono(poligono, angle)$$

Parámetros:

- **polígono:** Lista de puntos 2D que representa el contorno de una hoja de arbol.
- **angle:** Grados en los que será rotado el polígono respecto a su centroide.

El siguiente paso fue capturar el punto más hacia la izquierda y más hacia la derecha, $pizq$ y $pder$. Esto se realiza usando la función de elaboración propia:

$$findHorizontalExtremes(contorno\_hoja)$$

Parámetros:

- **contorno_hoja:** Lista de puntos 2D que representan el contorno de la hoja de árbol.

Una vez obtenidos los puntos, se sabe que para ambos puntos se encuentra un punto complementario que forme una línea recta con un ángulo de 0 °. Para encontrar este punto se usa la función de elaboración propia:

$$findHorizontal(poligono, pt, umbral)$$

Parámetros:

- **polígono:** Lista de puntos 2D que representa el contorno de la hoja de árbol.
- **pt:** Punto que será usado para encontrar su punto complementario, que forme una línea de 0 grados con este punto.
- **umbral:** Punto 2D que junto al parámetro *pt* es usado para calcular una distancia "umbral", un punto que forme una distancia con *pt* que sea inferior a la distancia umbral no pueden ser escogido como punto complementario. Esto se realiza para evitar que puntos que por coincidencia estén inmediatamente al lado de *pt,* sean escogidos como el punto complementario a retornar.

Se calcula la distancia de ambas líneas y la distancia mayor es escogida como el ancho fisiológico.

Área de la Hoja

En el trabajo el área de la hoja es calculada contando todos los pixeles negros de la imagen binaria suavizada. La librería "OpenCv" en cambio permite calcular el área de la hoja de árbol utilizando solo su contorno, gracias a la función:

$$contourArea(contour, oriented)$$

Parámetros:

- **contour:** Vector de puntos en 2D.
- **oriented:** Si su valor es *true* permite que la función entregue información respecto a la orientación de un contorno. Por defecto su valor es *false.*

Perímetro de la Hoja

Para calcular el perímetro de la hoja el autor simplemente cuenta el número de pixeles que conforman el borde de la hoja de árbol en la imagen que fue filtrada usando el filtro laplaciano. En cambio en esta implementación se obtiene el número de puntos que forman el contorno usando la función propia de Python *len*.

Características Morfológicas Digitales

Factor de Suavidad

Para calcular esta característica es necesario generar 2 imágenes a partir de la imagen binaria. Ambas imágenes son sometidas a un filtro medio rectangular de tamaño 5x5 y 2x2 respectivamente, con el fin de reducir el ruido en la imagen. Esta operación es realizada usando la función de la librería OpenCv:

blur(src, ksize, dst, anchor, borderType)

Parámetros:

- **src:** Imagen de entrada.
- **ksize:** Tamaño del kernel para realizar la operación.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **anchor:**
- **borderType:** Modo de borde usado para extrapolar pixeles fuera de la imagen.

Para destacar el borde de la hoja de árbol presente en las imágenes difuminadas que resultan de la aplicación del filtro medio se aplica un filtro Laplaciano sobre ambas imágenes usando la función de OpenCv:

$$Laplacian(src, ddept, dst, ksize, scale, delta, borderType)$$

Parámetros:

- **src:** Imagen de entrada.
- **ddepth:** Profundidad de la imagen de salida.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **ksize:** Tamaño del kernel para realizar la operación.
- **scale:** Valor opcional de escalado para los valore Laplacianos.
- **delta:** Valor delta opcional que se agrega a los resultados antes de guardarlos en *dst.*
- **borderType:** Método de extrapolación de pixeles.

Por último se encuentra el área de la hoja de árbol en ambas imágenes. Para esto primero es necesario capturar los bordes de la hoja destacados por la función anterior con la función de OpenCv *findContours,* y segundo se procede a calcular el área de ambas a partir de los contornos capturados usando la función de OpenCv *contourArea.* Una vez obtenidas ambas áreas simplemente se debe dividir el área de la hoja suavizada por el filtro medio rectangular de 5x5 por el área de la hoja suavizada con el filtro medio rectangular de 2x2.

La relación de aspecto, factor de forma, rectangularidad, factor de estrechez, relación perimetral de diámetro, y relación perimetral de longitud fisiológica y anchura fisiológica, fueron calculadas utilizando las formulas dadas en la sección 3.1.

Características de la Vena

Los elementos estructurales de radio 1, 2, 3 y 4 son generados en matlab, para luego ser usados para realizar las 4 aperturas morfológicas de la imagen en escala de grises. Los elementos estructurales no fueron generados en OpenCv ya que el autor originalmente genero estos elementos en matlab, y además la implementación de la función de OpenCv para generar estos elementos, *getStructuringElement* es muy distinta a la

función equivalente en matlab, creando elementos diferentes a los señalados por el autor.

Cada una de las imágenes obtenidas luego del proceso de apertura morfológica es usada para una operación de sustracción. Utilizando la función de OpenCv:

$$subtract\ (\text{src1}, \text{scr2}, \text{dst}, \text{mask}, \text{dtype})$$

Parámetros:

- **src1:** Primera imagen de entrada.
- **src2:** Segunda imagen de entrada.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **mask:** Operación opcional de mascara. Imagen binaria que especifica elementos a cambiar en la salida de la función.
- **dtype:** Profundidad opcional de la salida de la función.

Se realizan 4 operaciones de sustracción de imágenes en donde cada una de las imágenes generadas es restada de la imagen original, dando así 4 imágenes que si son analizadas con cautela se puede observar que tienen cierto parecido con las venas presentes en la hoja de árbol. Con el fin de eliminar los pixeles que no correspondan a la vena en estas nuevas imágenes, se usa procede a crear una máscara binaria de la forma de la hoja y esta es aplicada usando la función de OpenCv:

$$bitwise\_or(src, dst, mask)$$

Parámetros:

- **src:** Imagen de entrada.
- **dst:** Nombre de la variable que contendrá la imagen de salida.
- **mask:** Operación opcional de mascara. Imagen binaria que especifica elementos a cambiar en la salida de la función.

Esto se hace con el motivo de que en las imágenes solo queden presentes los pixeles de las venas dentro de la hoja. A continuación se aplica Thresholding sobre las 4 imágenes de las venas de la hoja. Por último se cuentan todos los pixeles blancos presentes en cada una de las imágenes de las venas, para obtener así $A_{v1}, A_{v2}, A_{v3}\ y\ A_{v4}$.

Se procede a calcular las características de la vena usando las formulas dadas en Sección 3.1.

Una vez extraídas todas las características de la hoja se procede a almacenarlas en un vector de características de 12 elementos.

4.1.4 Entrenamiento de la PNN

Una vez obtenidos los vectores de características para todas las imágenes del dataset, se crean 2 sets, un set de entrenamiento y un set de prueba. El set de entrenamiento estará formado por alguna proporción de los vectores de características extraídos, mientras que el set de prueba consiste en los vectores de características restantes que no serán usados para el entrenamiento del algoritmo.

Antes de entrenar el algoritmo se deben reducir las 12 características de cada vector de características obtenido a tan solo 5. Esto se realiza aplicando reducción de dimensionalidad con PCA, creando una instancia de la clase de la libreria *Sklearn, PCA:*

$$PCA(n\_components)$$

Parámetros:

- **n_components:** Numero de componentes principales a conservar.

Se procede a extraer los componentes principales desde el set de entrenamiento usando la función de la clase PCA:

$$fit(X, y)$$

Parámetros:

- **X:** Datos de entrenamiento.
- **y:** Se ignora.

A continuación se proyecta el set de entrenamiento y de prueba sobre los componentes principales extraídos en la función anterior usando la función de la clase PCA:

$$transform(X)$$

Parámetros:

- **X:** Datos que son proyectados a los componentes principales extraídos desde el set de entrenamiento.

Utilizando la librería *Neupy*, se crea una Red Neuronal Probabilística (PNN), creando una instancia de la clase *PNN* de la siguiente forma:

$$PNN(std, batch\_size, verbose)$$

Parámetros:

- **std:** Desviación estándar de la función de densidad de probabilidad. Su valor por defecto es 0,1.

- **batch_size:** Tamaño mínimo del número de ejemplos a usar. Su valor por defecto es 128.
- **verbose:** Controla la salida por la terminal. Si es *True,* se entrega información adicional por la terminal. Si es *False* se desactiva esta opción.

Se procede a entrenar el algoritmo, usando el set de entrenamiento resultante luego de aplicar la función *transform* de la clase PCA. Para esto se usa la función de la clase PNN:

$$train(X, y)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de entrenamiento.
- **y:** Lista de etiquetas correspondientes a los vectores de características en **X.**

Por último se deben realizar predicciones usando el set de prueba resultante luego de aplicar la función *transform* de PCA. La función de la clase PNN que permite realizar predicciones sobre un set de prueba usando un algoritmo entrenado previamente es la función:

$$predict(X)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de prueba.

4.1.5 Pseudocódigo

A continuación se presenta de manera simplificada como se realizó la implementación de este algoritmo.

Las líneas 3 a 7 corresponden a la etapa de pre procesamiento donde se utilizan las funciones: *CvtColor, gaussianBlur, adaptativeThreshold, morphologyEx y findContours.* Línea 11 utiliza la función *ConvexHull*. Línea 12 utiliza la función *findDiametro.* Línea 14 utiliza las funciones *moveraCentro*, *rotarPoligono, findHorizontalExtremes* y *findHorizontal.* La línea 15 utiliza la función *contourArea.* Línea 19 utiliza las funciones *blur y Laplacian.* Línea 26 utiliza las funciones *substract y bitwise_or.*

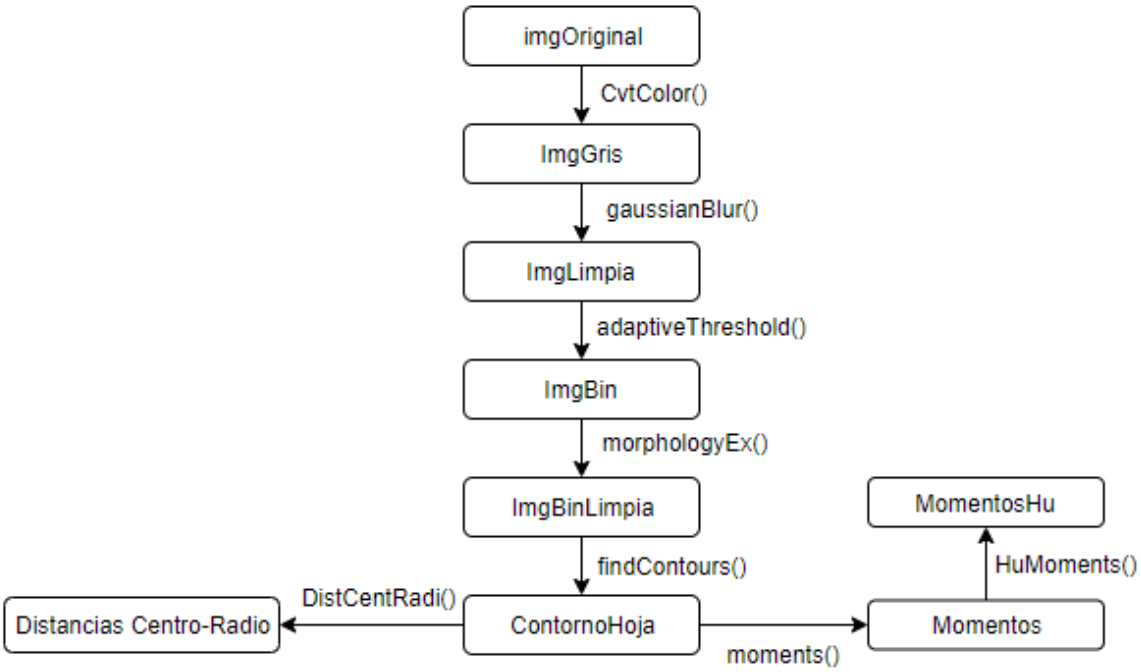
Algorithm 1 Implementacion del metodo de extraccion de Caracteristicas de Wu

```
1:  function EXTRAERCARACTERISTICAS(rutaImagen)
2:                                  ▷ Etapa de Pre procesamiento
3:      imagenBGR ← LeerImagenBGR(rutaImagen)
4:      imagenGris ← BGRAEscalaDeGris(imagenBGR)
5:      imagenGris ← FiltroGaussiano(imagenGris)
6:      imagenBin ← GrisABinario(imagenGris)
7:      contorno ← ExtraerContornos(imagenLimpia)
8:                      ▷ Etapa de Extraccion de Caracteristicas
9:
10:             ▷ Calcular caracteristicas geometicas basicas
11:     hull ← EnvolventeConvexa(contorno)
12:     diametro ← EncontrarDiametro(hull)
13:     lonFis ← diametro
14:     anchoFis ← EncontrarAnchoFis(diametro, hull)
15:     areaHoja ← CalcularArea(contorno)
16:     perimetro ← Size(contorno)
17:
18:          ▷ Calcular caracteristicas morfologicas digitales
19:     smoothFactor ← CalcularSmoothFactor(contorno)
20:     aspectRatio ← lonFis/achoFis
21:     formFactor ← (4 × π × areaHoja)/perimetro²
22:     rectangularity ← (lonFis × anchoFis)/areaHoja
23:     narrowFactor ← diametro/longFis
24:     prd ← perimetro/diametro
25:     prPlPw ← perimetro/(lonFis + anchoFis)
26:     venas ← CaractVenas(imgGris, imagenBin, contorno)
27:
28:     return smoothFactor, aspectRatio, formFactor,
              rectangularity, narrowFactor, prd, prPlPW, venas
29: end function
```

4.2 Plant Leaf Recognition using Shape based Features and Neural Network classifiers

4.2.1 Diagrama de Relaciones

El siguiente diagrama muestra de forma simplificada como se relacionan las funciones más relevantes que fueron utilizadas para la implementación del algoritmo. Estas funciones serán descritas de forma detallada a continuación del diagrama.



4.2.2 Pre Procesamiento de Imágenes

A diferencia del trabajo anterior, en esta ocasión no se explica en detalle el proceso de pre procesamiento de las imágenes, simplemente se menciona que las imágenes deben ser convertidas a imágenes binarias y que sobre estas se debe hacer uso del detector de bordes de Canny para identificar el contorno de la hoja. Teniendo esto en cuenta, se decidió usar el mismo método de pre procesamiento que el presentado en la sección 4.1. Se usa la función de OpenCv *cvtColor* para realizar la conversión de la imagen de entrada en formato BGR a escala de grises, se procede a remover el ruido de esta imagen con un filtro Gaussiano usando la función de OpenCv, *gaussianBlur*, para luego convertir esta imagen a una imagen binaria usando la función de OpenCv *adaptativeThreshold.* Nuevamente se utilizó la función de OpenCv *findCountours* para identificar y capturar el contorno por los motivos explicados en la implementación del algoritmo anterior. Se ignora implementación de Canny en el algoritmo.

4.2.3 Extracción de Características

Distancias Centro Radio

Se puede definir un radio como la distancia desde el centro de un contorno hasta algún punto de este mismo. Sabiendo esto se procede a seguir los pasos indicados en el trabajo con la tarea de encontrar las 36 distancias centro radio. Para esto se creó una función denominada:

$DistCentRadi$(polígono, t_intervalo)

- **polígono:** Contorno de una hoja de árbol.
- **t_intervalo:** El tamaño en grados de un intervalo.

Función que itera sobre todos los puntos del contorno para encontrar las distancias centro-radio. Como se puede observar en la Fig. 23 se asume en el trabajo que los ángulos se comenzaran a buscar a partir del eje x inmediatamente formando un ángulo de 0°, pero esto no es así al usar OpenCv, ya que el contorno de la hoja comienza y termina en la coordenada "y" más pequeña. Como no se puede predecir desde donde comenzara a iterar la función *DistCentRadi,* se procede a crear una lista de ángulos que contiene todos los valores que pueden generarse usando un intervalo de 10° (0°, 10°, 20°,...350°). Al iterar sobre los puntos del contorno se calcula si el ángulo generado entre el punto actual y el centroide del contorno forma algún ángulo dentro de la lista. Si es así se guarda la distancia a este punto y se elimina el valor del ángulo de la lista. La ejecución termina cuando la lista está vacía y se ha encontrado todas las distancias centro radio, o cuando se termina de iterar por el contorno.

Momentos

Como se indica en el trabajo se deben extraer el primer y tercer momentos de Hu a partir de la imagen en escala de grises. Los momentos de Hu son calculados a partir de los momentos de la imagen, por lo que estos deben ser calculados primero. Esto se logra usando la función de OpenCv,

$$moments(array, binaryImage)$$

Parámetros:

- **array:** Vector de puntos de entrada.
- **binaryImage:** Si su valor es *True,* todos los pixeles cuyo valor no sea 0 son tratados como 1.

Una vez obtenidos los momentos estos son utilizados por la función de OpenCv:

HuMoments(m, hu)

Parámetros:

- **m:** Momentos de entrada computados con la función *moments()*
- **hu:** Variable de salida que contiene los momentos invariables de Hu.

Esta función es utilizada para calcular los momentos de Hu y devuelve una lista de 7 valores de los cuales se guardan el primer y tercer valor.

Ya obtenidos las 36 distancias centro-radio y los 2 momentos de Hu, se procede a guardar estos elementos en un vector de características e 38 elementos.

4.2.4 Entrenamiento de MLP

Una vez obtenidos los vectores de características para todas las imágenes del dataset, se crean 2 sets, un set de entrenamiento y un set de prueba. El set de entrenamiento estará formado por alguna proporción de los vectores de características extraídos, mientras que el set de prueba consiste en los vectores de características restantes que no serán usados para el entrenamiento del algoritmo.

Usando la librería Sklearn, se crea una instancia de una MLP de 3 capas, creando una instancia de la clase *MLPClasifier* de la siguiente forma:

$$MLPClassifier(hidden\_layer\_sizes, max\_iter)$$

Parámetros:

Es necesario mencionar que el constructor de esta clase tiene más de 20 parámetros, pero solo los 2 parámetros descritos a continuación son relevantes para el proyecto.

- **hidden_layer_sizes:** El número de neuronas en la capa oculta.
- **max_iter:** Numero de iteraciones máximas permitidas. Si no se encuentra la solución óptima antes de *max_iter,* el algoritmo devuelve la solución encontrada hasta este punto.

Se procede a entrenar el algoritmo con el set de entrenamiento, usando la función de la clase *MLPClassifier*:

$$fit(X, y)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de entrenamiento.
- **y:** Lista de etiquetas correspondientes a los vectores de características en **X.**

Por último se realizan predicciones usando el set de prueba. Se deben remover las etiquetas de los vectores de características del set de prueba antes de realizar las predicciones.

La función de la clase *MLPClassifier,* que permite hacer predicciones sobre un set de prueba usando un algoritmo entrenado previamente, es la función:

$$predict(X)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de prueba.

4.2.5 Pseudocódigo

A continuación se presenta de manera simplificada como se realizó la implementación de este algoritmo.

Las líneas 3 a 7 corresponden a la etapa de pre procesamiento donde se utilizan las funciones: *CvtColor, gaussianBlur, adaptativeThreshold, morphologyEx y findContours.* Línea 9 utiliza la función *DistCentRad.* Línea 13 usa las funciones *Moments y HuMoments.*

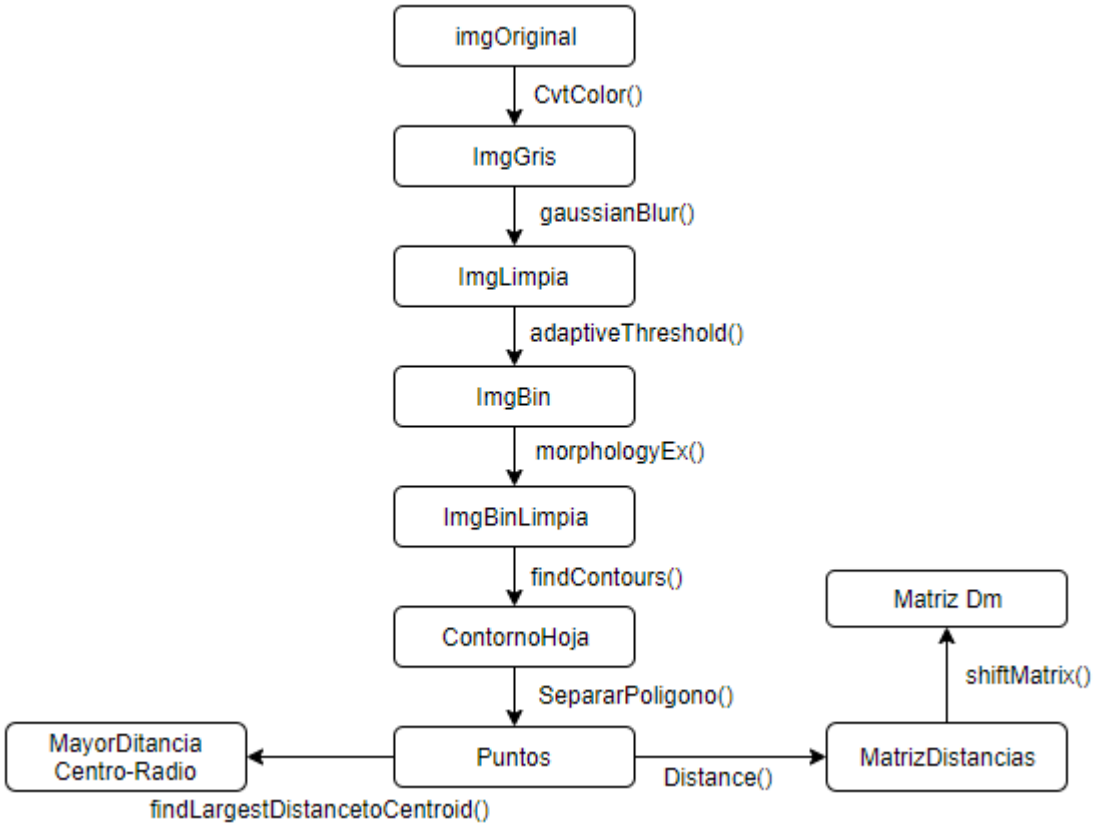
Algorithm 1 Implementacion del metodo de extraccion de Caracteristicas de Chaki

```
1:  function EXTRAERCARACTERISTICAS(rutaImagen)
2:                                  ▷ Etapa de Pre procesamiento
3:      imagenBGR ← LeerImagenBGR(rutaImagen)
4:      imagenGris ← BGRAEscalaDeGris(imagenBGR)
5:      imagenGris ← FiltroGaussiano(imagenGris)
6:      imagenBin ← GrisABinario(imagenGris)
7:      contorno ← ExtraerContornos(imagenLimpia)
8:                      ▷ Etapa de Extraccion de Caracteristicas
9:      distancias ← distanciasCentroRadio(contorno)
10:     for i = 0; i < len(distancias); i++ do
11:         distancias[i] = distancias[i]/distanciaMayor
12:     end for
13:     momentosHu ← HuMoments(imagenGris)
14:     momentosHu ← momentosHu[0], momentosHu[2]
15:     caracteristicas ← distancias, momentosHu
16:     return distancias, momentosHu
17: end function
```

4.3 Multiscale Distance Matrix for Fast Plant Leaf Recognition

4.3.1 Diagrama de Relaciones

El siguiente diagrama muestra de forma simplificada como se relacionan las funciones más relevantes que fueron utilizadas para la implementación del algoritmo. Estas funciones serán descritas de forma detallada a continuación del diagrama.



4.3.2 Pre Procesamiento de Imágenes

A diferencia de los 2 trabajos anteriores, en esta ocasión no se explica cómo se debe realizar el pre procesamiento de las imágenes. Es por esta razón que nuevamente se aplica el método de pre procesamiento utilizado en la implementación de los 2 algoritmos anteriores, método que es explicado en detalle en sección 4.1.

4.3.3 Extracción de Características

División de Puntos

Antes de calcular la MDM es necesario dividir el contorno en 128 puntos, esto se hace con la función de implementación propia:

$$separarPoligono(poligono, n\_points)$$

Parámetros:

- **polígono:** Vector de puntos que representa el contorno de la hoja de árbol.
- **n_points:** Cantidad de puntos de igual distancia entre sí, que serán extraídos desde el contorno.

Función que en primer lugar divide la cantidad de pixeles en el contorno de la hoja por 128 resultando así en la cantidad de puntos que deben ser recorridos

Matriz de distancias D

Una vez obtenidos los 128 puntos del contorno se procede a calcular la distancia Euclidiana entre cada uno de estos puntos con sus demás puntos vecinos, y a guardar estas distancias en una matriz D. Para esto se usa la función de elaboración propia:

$$Distance(p1, p2)$$

Parámetros:

- **p1:** Lista de 2 valores que representa un punto en 2D.
- **p2:** Lista de 2 valores que representa un punto en 2D.

Las distancias individuales entre cada punto hasta los demás puntos vecinos son guardadas en cada fila de la matriz D, por ejemplo la primera fila contiene la distancia del punto 1 a los demás puntos 1, 2, 3, 4, ...n.

Matriz Dm

Se crea la matriz Dm usando la función de implementación propia:

$$shiftMatrix(matrix)$$

Parámetros:

- **matrix:** Matriz de 2 dimensione sobre la cual se aplica la operación.

La cual mueve los elementos de cada columna de manera circular hacia arriba de forma que el elemento 0 de cada columna quede en el primer lugar.

Matriz Dms

La matriz Dms es creada fácilmente ordenando de menor a mayor los elementos de cada fila de la matriz Dm. Ya que una matriz es una lista de listas, donde cada una de estas listas representa una fila, se utiliza la función propia de cada lista *sort* para sortear los elementos de cada una de estas.

MDM

Se construye la MDM removiendo la primera fila de la matriz Dms y las ultimas $[\frac{n-1}{2}]$ filas.

MDM-CD

Antes de construir la MDM-CD es necesario calcular la matriz de diferencias como es señalado en la sección 3.3. Una vez calculada esta matriz es agregada al fondo de la MDM creando así la matriz MDM-CD.

MDM-CD-C

Como es señalado en el trabajo utilizar la MDM-CD-C entrego los mejores resultados a la hora de realizar la clasificación de especie, por lo que se decide utilizar esta matriz e ignorar la construcción de matrices como MDM-CD-A o MDM-CD-M que dieron peores resultados.

Para construir esta matriz simplemente se debe normalizar MDM-CD por la mayor distancia entre el centro del contorno y alguno de los 128 puntos seleccionados en un principio. Para esto se utiliza la función de elaboración propia:

$$findLargestDistancetoCentroid(points, cx, cy)$$

Parámetros:

- **points:** Lista de puntos extraídos por la función *separaPoligono().*
- **cx:** Coordenada X del centro del contorno de la hoja.
- **cy:** Coordenada Y del centro del contorno de la hoja.

4.3.4 Entrenamiento de KNN

Una vez obtenidos los vectores de características para todas las imágenes del dataset, se crean 2 sets, un set de entrenamiento y un set de prueba. El set de entrenamiento estará formado por alguna proporción de los vectores de características extraídos, mientras que el set de prueba consiste en los vectores de características restantes que no serán usados para el entrenamiento del algoritmo.

Un paso que se debe realizar antes de entrenar o probar el modelo, es reducir los 16384 componentes que forman cada vector de características a tan solo 20 componentes.

Esto se realiza aplicando el método de reducción de dimensionalidad conocido como "Análisis de Componentes Principales", que es implementado creando una instancia la clase de la librería Neupy, *PCA.* La cual es instanciada de la siguiente forma:

$$PCA(n\_components)$$

Parámetros:

- **n_components:** Numero de componentes a conservar.

Se procede a extraer los componentes principales desde el set de entrenamiento usando la función de la clase PCA:

$$fit(X, y)$$

Parámetros:

- **X:** Datos de entrenamiento.
- **y:** Se ignora.

A continuación se proyecta el set de entrenamiento y de prueba sobre los componentes principales extraídos en la función anterior usando la función de la clase PCA:

$$transform(X)$$

Parámetros:

- **X:** Datos que son proyectados a los componentes principales extraídos desde el set de entrenamiento.

El siguiente paso es crear una instancia de K-Nearest Neightbors usando la clase KNN de la librería Sklearn:

$$KNN(n\_neightbors)$$

Parámetros:

- **n_neightbors:** Numero de vecinos a considerar.

Se procede a entrenar el algoritmo, usando el set de entrenamiento resultante luego de aplicar la función *transform* de la clase PCA, usando la función de la clase KNN:

$$fit(X, y)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de entrenamiento.
- **y:** Lista de etiquetas correspondientes a los vectores de características en **X.**

El último paso entonces es realizar predicciones usando el set de prueba resultante luego de aplicarle la función *transform* de la clase PCA. Se deben remover las etiquetas de los vectores de características en este set antes de realizar las predicciones.

La función de la clase KNN que permite realizar predicciones sobre un set de prueba usando un algoritmo entrenado previamente es la función:

$$predict(X)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de prueba.

4.3.5 Pseudocódigo

A continuación se presenta de manera simplificada como se realizó la implementación de este algoritmo.

Las líneas 3 a 7 corresponden a la etapa de pre procesamiento donde se utilizan las funciones: *CvtColor, gaussianBlur, adaptativeThreshold, morphologyEx y findContours.* Línea 9 utiliza la función *SeparaPoligono.* Línea 12 utiliza la función *Distance.* Línea 15 utiliza la función *shiftMatrix.* Línea 27 utiliza la función *findLargestDistancetoCentroid.*

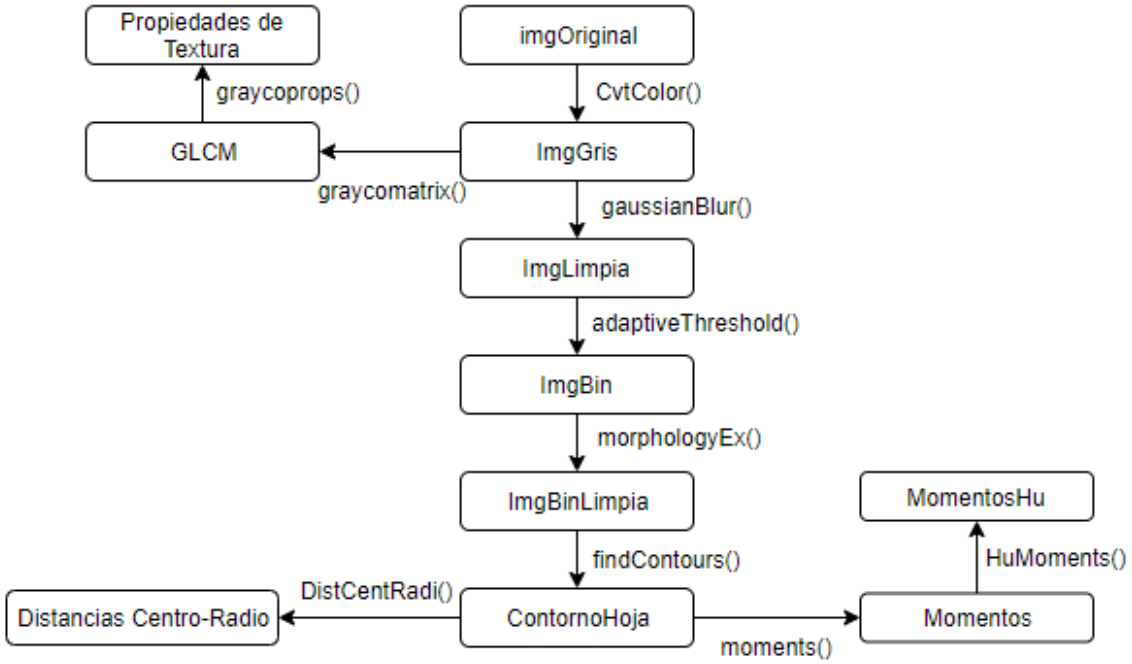
Algorithm 1 Implementacion del metodo de extraccion de Caracteristicas de Rongxiang

```
1:  function EXTRAERCARACTERISTICAS(rutaImagen)
2:                                  ▷ Etapa de Pre procesamiento
3:      imagenBGR ← LeerImagenBGR(rutaImagen)
4:      imagenGris ← BGRAEscalaDeGris(imagenBGR)
5:      imagenGris ← FiltroGaussiano(imagenGris)
6:      imagenBin ← GrisABinario(imagenGris)
7:      contorno ← ExtraerContornos(imagenLimpia)
8:                      ▷ Etapa de Extraccion de Caracteristicas
9:      puntos ← separarPoligonos(contorno)
10:     for i = 0; i < len(puntos); i++ do
11:         for j = 0; j < len(puntos); j++ do
12:             D[i][j] = Distancia(puntos[i], puntos[j])
13:         end for
14:     end for
15:     Dm ← DesplazarMatriz(D)
16:     Dms ← Dm
17:     for i = 0; i < len(puntos); i++ do
18:         Dms[i] = OrdenarAscendente(mdm[i])
19:     end for
20:     matrizDifs ← MatrizDiferencias(Dm)
21:     mdm ← Dms
22:     mdm ← RemoverFilas(Dms[0])
23:     for i = 0; i < (n-1)/2; i++ do
24:         mdm ← RemoverFilas(mdm[−1])
25:     end for
26:     mdm ← Unir(mdm, matrizDifs)
27:     distMayor ← DistMayorCentroRadio(puntos)
28:     mdm ← NormalizarMatriz(mdm, distMayor)
29:     return mdm
30: end function
```

4.4 Algoritmo Experimental de elaboración propia I

4.4.1 Diagrama de Relaciones

El siguiente diagrama muestra de forma simplificada como se relacionan las funciones más relevantes que fueron utilizadas para la implementación del algoritmo. Estas funciones serán descritas de forma detallada a continuación del diagrama.



4.4.2 Pre procesamiento de Imágenes

Nuevamente se ocupa el mismo proceso para realizar la etapa de pre procesamiento de imágenes de hojas de árboles, es por esta razón que no se explica el proceso en detalle.

4.4.3 Etapa de extracción de características

Es necesario mencionar que el algoritmo que se describirá a continuación es una modificación del algoritmo presentado por Chaki en la sección 4.2, por lo que los pasos de extracción de momentos y distancia centro-radio son exactamente las mismas.

Distancias Centro Radio

Se puede definir un radio como la distancia desde el centro de un contorno hasta algún punto de este mismo. Sabiendo esto se procede a seguir los pasos indicados en el trabajo con la tarea de encontrar las 36 distancias centro radio. Para esto se creó una función denominada:

$DistCentRadi$(polígono, t_intervalo)

- **polígono:** Contorno de una hoja de árbol.

- **t_intervalo:** El tamaño en grados de un intervalo.

Función que itera sobre todos los puntos del contorno para encontrar las distancias centro-radio. Como se puede observar en la Fig. 23 se asume en el trabajo que los ángulos se comenzaran a buscar a partir del eje x inmediatamente formando un ángulo de 0°, pero esto no es así al usar OpenCv, ya que el contorno de la hoja comienza y termina en la coordenada "y" más pequeña. Como no se puede predecir desde donde comenzara a iterar la función *DistCentRadi,* se procede a crear una lista de ángulos que contiene todos los valores que pueden generarse usando un intervalo de 10° (0°, 10°, 20°,...350°). Al iterar sobre los puntos del contorno se calcula si el ángulo generado entre el punto actual y el centroide del contorno forma algún ángulo dentro de la lista. Si es así se guarda la distancia a este punto y se elimina el valor del ángulo de la lista. La ejecución termina cuando la lista está vacía y se ha encontrado todas las distancias centro radio, o cuando se termina de iterar por el contorno.

Momentos

Como se indica en el trabajo se deben extraer el primer y tercer momentos de Hu a partir de la imagen en escala de grises. Los momentos de Hu son calculados a partir de los momentos de la imagen, por lo que estos deben ser calculados primero. Esto se logra usando la función de OpenCv,

$$moments(array, binaryImage)$$

Parámetros:

- **array:** Vector de puntos de entrada.
- **binaryImage:** Si su valor es *True,* todos los pixeles cuyo valor no sea 0 son tratados como 1.

Una vez obtenidos los momentos estos son utilizados por la función de OpenCv:

$$\text{HuMoments}(m, hu)$$

Parámetros:

- **m:** Momentos de entrada computados con la función *moments()*
- **hu:** Variable de salida que contiene los momentos invariables de Hu.

Esta función es utilizada para calcular los momentos de Hu. La función devuelve una lista de 7 valores de los cuales se guardan el primer y tercer valor.

Extracción de Textura

Para examinar las características de la textura perteneciente a las hojas de árboles contenidas en las imágenes fotográficas, es primero necesario calcular la matriz de

coocurrencias de niveles de gris (GLCM) correspondiente a la hoja que está siendo procesada. Para esto se utiliza la función de la librería *Skimage:*

$$graycomatrix(image, distances, angles, levels, symmetric, normed)$$

Parámetros:

- **image:** Imagen de entrada.
- **distances:** Distancia del pixel de referencia.
- **angles:** Lista de direcciones en radianes que tomara el pixel de referencia.
- **levels:** El número de niveles de gris a considerar, típicamente 256 para una imagen en escala de grises. Considerar que la matriz de salida es de tamaño *levels x levels.*
- **symmetric:** Si su valor es *True,* entonces la matriz de salida es simétrica.
- **normed:** Si su valor es *True,* cada una de las matrices producidas es normalizada.

Es necesario mencionar que esta función genera una GLCM por cada ángulo entregado en el parámetro *angles,* por lo que salida de esta función es una lista de GLCM.

Una vez calculada la lista de GLCM se procede a calcular sus propiedades usando la función de *Skimage:*

$$graycoprops(P, prop)$$

Parámetro:

- **P:** GLCM de entrada.
- **prop:** La propiedad de la GLCM a computar, los valores que puede tomar esta variable pueden ser 6:
  - **contrast**
  - **dissimilarity**
  - **homogeneity**
  - **energy**
  - **correlation**
  - **ASM**

Ya que se desea calcular las 6 propiedades de la GLCM y teniendo en cuenta que se construyen 4 de estas matrices, se obtienen 24 valores que representan las propiedades de la textura de la hoja de árbol.

Considerando los valores obtenidos en la extracción de momentos, distancias centro-radio y propiedades de la textura, se obtiene un vector de características de 62 valores.

4.4.4 Entrenamiento de MLP

Una vez obtenidos los vectores de características para todas las imágenes del dataset, se crean 2 sets, un set de entrenamiento y un set de prueba. El set de entrenamiento estará formado por alguna proporción de los vectores de características extraídos, mientras que el set de prueba consiste en los vectores de características restantes que no serán usados para el entrenamiento del algoritmo.

Antes de realizar el entrenamiento del modelo, es necesario aplicar una estandarización de la escala de los datos contenidos en el set de entrenamiento y el set de prueba. Para iniciar este proceso primero se debe crear una instancia de la clase StandarScaler, la cual pertenece a la librería *Sklearn:*

$$StandarScaler(copy, with\_mean, with\_std)$$

Parámetros:

- **copy:** Si su valor es *True* crea una copia de los datos y sobre esta se realiza el proceso de estandarización.
- **with_mean:** Si su valor es *True* centra los datos antes de escalarlos. Toma el valor *True* por defecto.
- **with_std:** Si su valor es *True* se escalan los datos a la unidad de varianza. Toma el valor *True* por defecto

Para computar la media y deviación estándar que se utilizara para escalar los datos, se usa la función perteneciente a la clase StandarScaler:

$$fit(X, y)$$

Parámetros:

- **X:** El set de datos que se usara para computar la media y desviación estándar.
- **y:** Parámetro ignorado.

Para realizar la estandarización de los datos del set de entrenamiento y set de prueba, se usa la función de la clase *StandarScaler:*

$$transform(X, y, copy)$$

Parámetros:

- **X:** Los datos a escalar.
- **y:** Parámetro ignorado.
- **copy:** Parámetro booleano. *True* para copiar el input X.

Usando la librería *Sklearn*, se crea una instancia de una MLP de 3 capas, creando una instancia de la clase *MLPClasifier* de la siguiente forma:

$$MLPClassifier(hidden\_layer\_sizes, max\_iter)$$

Parámetros:

Es necesario mencionar que el constructor de esta clase tiene más de 20 parámetros, pero solo los 2 parámetros descritos a continuación son relevantes para el proyecto.

- **hidden_layer_sizes:** El número de neuronas en la capa oculta.
- **max_iter:** Numero de iteraciones máximas permitidas. Si no se encuentra la solución óptima antes de *max_iter,* el algoritmo devuelve la solución encontrada hasta este punto.

Se procede a entrenar el algoritmo usando el set de entrenamiento previamente definido, utilizando la función de la clase MLPClassifier:

$$fit(X, y)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de entrenamiento.
- **y:** Lista de etiquetas correspondientes a los vectores de características en **X.**

Por último se realizan predicciones usando el set de prueba. Se deben remover las etiquetas de los vectores de características de este set antes de realizar las predicciones.

La función de la clase MLPClassifier que permite realizar predicciones sobre un set de prueba usando un algoritmo entrenado previamente es la función:

$$predict(X)$$

Parámetros:

- **X:** La lista de vectores de características que forman el set de prueba.

4.4.5 Pseudocódigo

A continuación se presenta de manera simplificada como se realizó la implementación de este algoritmo.

Las líneas 3 a 7 corresponden a la etapa de pre procesamiento donde se utilizan las funciones: *CvtColor, gaussianBlur, adaptativeThreshold, morphologyEx y findContours.* Línea 9 utiliza la función *DistCentRad.* Línea 13 usa las funciones *Moments y HuMoments.* Línea 17 utiliza la función *graycomatrix.* Líneas 18 a 23 utilizan la función *graycoprops* con distintos parámetros.

Algorithm 1 Implementacion de la modificacion al metodo de extraccion de caracteristicas propuesto por Chaki

```
1:  function EXTRAERCARACTERISTICAS(rutaImagen)
2:                                  ▷ Etapa de Pre procesamiento
3:      imagenBGR ← LeerImagenBGR(rutaImagen)
4:      imagenGris ← BGRAEscalaDeGris(imagenBGR)
5:      imagenGris ← FiltroGaussiano(imagenGris)
6:      imagenBin ← GrisABinario(imagenGris)
7:      contorno ← ExtraerContornos(imagenLimpia)
8:                      ▷ Etapa de Extraccion de Caracteristicas
9:      distancias ← distanciasCentroRadio(contorno)
10:     for i = 0; i < len(distancias); i++ do
11:         distancias[i] = distancias[i]/distanciaMayor
12:     end for
13:     momentosHu ← HuMoments(imagenGris)
14:     momentosHu ← momentosHu[0], momentosHu[2]
15:     caracteristicas ← distancias, momentosHu
16:
17:     glcm ← GLCM(imagenGris)
18:     contrast ← Contrast(glcm)
19:     dissimilarity ← Dissimilarity(glcm)
20:     homogenity ← Homogenity(glcm)
21:     asm ← Asm(glcm)
22:     energy ← Energy(glcm)
23:     correlation ← Correlation(glcm)
24:     textura ← [contrast, dissimilarity, homogenity,
        asm, energy, correlation]
25:
26:     return distancias, momentosHu, textura
27: end function
```

4.5 Algoritmo Experimental de elaboración propia II

A diferencia de los demás algoritmos presentados, no se entregara mayor detalle respecto a este algoritmo. La razón siendo que para clasificar una hoja de árbol, simplemente realiza una votación llamando a los 3 algoritmos que fueron implementados originalmente (Chaki, Hu y Wu) para que realicen una predicción de especies. La especie que es escogida por la mayor cantidad de algoritmos se denomina como la especie a la que pertenece la hoja de árbol. En caso de que exista un empate en la votación se escoge la predicción con el mayor porcentaje de probabilidad de clase. En palabras simples en este caso se escoge al algoritmo que este más seguro acerca de su predicción.

Para este algoritmo no existe una etapa de pre procesamiento, ni de extracción de características, ni de entrenamiento de un modelo, simplemente se ocupa a los algoritmos que ya fueron previamente entrenados.

Capítulo 5: Experimentación

En este capítulo se explica cómo se realizaron las pruebas con el fin de evaluar los algoritmos implementados, además se presentan los resultados de dichas pruebas en diferentes tablas. Por último se realiza una discusión a partir de los resultados obtenidos para cada situación de prueba.

5.1 Datos

Las hojas de árboles que componen el dataset que se utiliza para el desarrollo este proyecto de título fueron recolectadas de forma manual en la Universidad del Biobío sede Concepción.

El dataset está compuesto por 300 imágenes de hojas de árboles, pertenecientes a 3 especies de árboles distintas. Estas se encuentran distribuidas de la siguiente manera:

- 100 hojas de Drimys Winteri (Canelo Mapuche).
- 100 hojas de Laurelia Sempervirens (Laurel Chileno).
- 100 hojas de Peumus Boldus (Boldo).

Las fotografías están en formato JPG y tiene una dimensión de 1152 x 2048. Una muestra de las fotografías tomadas de encuentra en el Anexo sección 8.4.

5.2 Protocolo de Prueba

Para evaluar los algoritmos propuestos por los distintos autores y los de elaboración propia, se propone en primera instancia 2 situaciones:

En la primera situación se divide el dataset en 2 sets en una proporción 50/50, esto quiere decir que el 50% del dataset será utilizado para el entrenamiento del modelo, mientras que el 50% restante se toma para evaluar el modelo entrenado. La distribución del número de hojas para cada clase se hace de manera uniforme. Como se explicó anteriormente el dataset está compuesto de 300 hojas pertenecientes a 3 especies de árboles, por lo que en esta situación el set de entrenamiento estará formado por 150 hojas, 50 hojas por especie. El set de testeo estará compuesto por 150 hojas, 50 hojas por especie.

En la segunda situación, se divide el dataset en una proporción 80/20, usando el dataset propuesto esto quiere decir que 240 hojas, 80 hojas por especie, serán usadas para el entrenamiento del modelo, mientras que las 60 hojas restantes, compuestas de 20 hojas por especie, se utilizan para el testeo del modelo.

Cada algoritmo será evaluado sobre las 2 situaciones, para evaluar el rendimiento del modelo en cada situación se obtienen las siguientes métricas de rendimiento:

- Acurracy
- Presicion
- Recall

- F1-Scorre

Por ultimo para obtener métricas de clasificación más precisas se realizara una validación cruzada de 10 hojas (folds). Esto quiere decir que para cada modelo se tomara el dataset de vectores de características correspondiente y este será dividido en 10 partes, para la primera instancia de la validación cruzada se tomara la primera parte como un set de testeo, mientras que las 9 partes restantes serán usadas como el set de entrenamiento, para la segunda instancia se tomara la segunda parte como el set de testeo y las partes restantes como el set de entrenamiento, esto se repite sucesivamente hasta que todas las partes hayan sido ocupadas como el set de testeo. Para cada una de las 10 instancias se entrena un modelo utilizando las partes definidas como el set de entrenamiento. Se evalúa el modelo usando la parte del dataset asignada como el set de testeo. Por último basándose en las predicciones arrojadas en la evaluación del modelo se calculan las siguientes métricas: *precisión, recall, f1-score* y *accuracy.* Una vez obtenidas las métricas de evaluación para las 10 instancias, se obtiene la media de cada una de estas métricas.

5.3 Algoritmos utilizados

Los algoritmos que fueron utilizados para realizar las pruebas fueron:

1) A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network (**Wu**).
2) Plant Leaf Recognition using Shape based Features and Neural Network classifiers (**Chaki**).
3) Multiscale Distance Matrix for Fast Plant Leaf Recognition (**Hu**).
4) Algoritmo Experimental de elaboración propia I (**CR-MTS-TX**).
5) Algoritmo Experimental de elaboración propia II (**Voto Mayoritario**).

Las configuraciones específicas de las funciones de estos algoritmos para cada situación de prueba se encuentran en el Anexo sección 8.3.

5.4 Resultados

A continuación se presentan los resultados obtenidos por los algoritmos implementados para cada una de las situaciones de prueba. Resultados más detallados se encuentran en el Anexo sección 8.2.

5.4.1 Drimys Winteri

Situación I

	PRECISION	RECALL	F1	SUPPORT
CHAKI	**1.0**	0.90	0.95	50
HU	0.95	0.84	0.89	50
WU	0.94	0.90	0.92	50
CR-MTS-TX	**1.0**	**0.98**	**0.99**	50
VOTO MAYORITARIO	**1.0**	0.92	0.96	50

Tabla l: Tabla de resultados para la especie Drimys Winteri en la situación de prueba I.

Situación II

	PRESICION	RECALL	F1	SUPPORT
CHAKI	**1.0**	0.95	0.97	20
HU	**1.0**	**1.0**	**1.0**	20
WU	0.90	0.95	0.93	20
CR-MTS-TX	**1.0**	**1.0**	**1.0**	20
VOTO MAYORITARIO	**1.0**	**1.0**	**1.0**	20

Tabla m: Tabla de resultados para la especie Drimys Winteri en la situación de prueba II.

5.4.2 Laurelia Sempervirens

Situación I

	PRESICION	RECALL	F1	SUPPORT
CHAKI	**0.98**	**1.0**	**0.99**	50
HU	0.89	0.98	0.93	50
WU	0.96	0.96	0.96	50
CR-MTS-TX	**0.98**	**1.0**	**0.99**	50
VOTO MAYORITARIO	**0.98**	**1.0**	**0.99**	50

Tabla n: Tabla de resultados para la especie Laurelia Sempervirens en la situación de prueba I.

Situación II

	PRESICION	RECALL	F1	SUPPORT
CHAKI	0.95	**1.0**	0.98	20
HU	**1.0**	**1.0**	**1.0**	20
WU	0.95	0.90	0.92	20
CR-MTS-TX	**1.0**	**1.0**	**1.0**	20
VOTO MAYORITARIO	**1.0**	**1.0**	**1.0**	20

Tabla o: Tabla de resultados para la especie Laurelia Sempervirens en la situación de prueba II.

5.4.3 Peumus Boldus

Situación I

	PRESICION	RECALL	F1	SUPPORT
CHAKI	0.93	**1.0**	0.96	50
HU	0.94	0.96	0.95	50
WU	0.94	0.98	0.96	50
CR-MTS-TX	**1.0**	**1.0**	**1.0**	50
VOTO MAYORITARIO	0.94	**1.0**	0.97	50

Tabla p: Tabla de resultados para la especie Peumus Boldus en la situación de prueba I.

Situación II

	PRESICION	RECALL	F1	SUPPORT
CHAKI	**1.0**	**1.0**	**1.0**	20
HU	**1.0**	**1.0**	**1.0**	20
WU	**1.0**	**1.0**	**1.0**	20
CR-MTS-TX	**1.0**	**1.0**	**1.0**	20
VOTO MAYORITARIO	**1.0**	**1.0**	**1.0**	20

Tabla q: Tabla de resultados para la especie Peumus Boldus en la situación de prueba II.

5.4.4 Accuracy

Situación I

	ACCURACY
CHAKI	0.9667
HU	0.9267
WU	0.9467
CR-MTS-TX	**0.9933**
VOTO MAYORITARIO	0.9733

Tabla r: Tabla de Accuracy para la situación de prueba I.

Situación II

	ACCURACY
CHAKI	0.983
HU	**1.0**
WU	0.95
CR-MTS-TX	**1.0**
VOTO MAYORITARIO	**1.0**

Tabla s: Tabla de Accuracy para la situación de prueba II.

5.4.5 Situación III: Cross Validation (K=10)

Drimys Winteri

	PRESICION	RECALL	F1
CHAKI	0.9169	0.9091	0.9069
HU	0.9190	0.8713	0.8871
WU	0.9411	0.9578	0.9460
CR-MTS-TX	**0.9808**	**0.9888**	**0.9840**
VOTO MAYORITARIO	0.9348	0.9376	0.9310

Tabla t: Tabla de resultados para la especie Drimys Winteri en la situación de prueba III.

Laurelia Sempervirens

	PRESICION	RECALL	F1
CHAKI	0.9396	0.9524	0.9441
HU	0.8886	0.9471	0.9126
WU	0.9720	0.9636	0.9657
CR-MTS-TX	**0.9857**	**0.9774**	**0.9802**
VOTO MAYORITARIO	0.9800	0.9653	0.9705

Tabla u: Tabla de resultados para la especie Laurelia Sempervires en la situación de prueba III.

Peumus Boldus

	PRESICION	RECALL	F1
CHAKI	0.9600	0.9638	0.9588
HU	0.9732	0.9433	0.9551
WU	0.9909	0.9690	0.9788
CR-MTS-TX	**1.0**	**1.0**	**1.0**
VOTO MAYORITARIO	0.9668	0.9524	0.9575

Tabla v: Tabla de resultados para la especie Peumus Boldus en la situación de prueba III.

Accuracies

	ACCURACY
CHAKI	0.9367
HU	0.9233
WU	0.9633
CR-MTS-TX	**0.9900**
VOTO MAYORITARIO	0.9533

Tabla w: Tabla de Accuracy para la situación de prueba III.

5.5 Discusión

Teniendo en cuenta los resultados que se obtuvieron es claro que para todas las situaciones el algoritmo CR-MTS-TX obtuvo las métricas más altas. En la primera situación se alcanza un *recall* y *precision* perfectos o casi perfectos al realizar predicciones sobre las 3 especies. En la segunda situación estos resultados mejoran y se obtiene un *recall* y *precision* de 1.0 para todas las especies. Por ultimo en la validación cruzada, para "Drymis Winteri" y "Laurelia Sempevirens" se obtiene una *precision* de sobre 0.98 y un *recall* de 0.97, mientras que para "Pemus Boldus" se alcanzan resultados de 1.0 para ambas métricas. La métrica de Accuracy refleja algo similar a estos resultados, para la primera y tercera situación CR-MTS-TX obtiene la métrica más alta, mientras que para la segunda situación comparte el puesto con Hu y voto mayoritario.

El algoritmo CR-MTS-TX es una modificación del algoritmo propuesto por Chaki, con la diferencia de que se realiza un análisis de la textura de las hoja generando una GLCM y a partir de esta se extraen características de la textura. El desempeño de este algoritmo confirmaría la importancia que tiene la textura para la tarea de clasificación de hojas de árboles. Comparando estos resultados con los alcanzados por Chaki es notable la mejora en las todas métricas, sobre todo en la situación donde se hace la validación cruzada donde existe una mejora de *precision y recall* de entre 4 y 7 puntos, y de Acurracy de más de 5 puntos.

Curiosamente para cada una de las situaciones este algoritmo clasifico de forma perfecta la especie "Peumus Boldus", una explicación de esto podría ser la textura "áspera" distintiva de esta especie en contraste a la textura "lisa" de las otras 2 especies de hoja, aunque considerando que todos los algoritmos, a excepción de voto mayoritario, obtienen métricas más altas al clasificar esta especie, la razón este en la forma más ancha de esta especie, en contraste con la forma alargada de las otras 2 especies.

A diferencia de los demás algoritmos, se aplicó StandarScaler de la librería Sklearn, con el fin de estandarizar los datos, ya que no todos los datos están en la misma escala. En un momento se pensó que haber realizado este proceso pudo haber sido la causa de haber obtenido tan buenos resultados en comparación al algoritmo de Chaki original, pero al aplicar StandarScaler a Chaki, el algoritmo CR-MTS-TX sigue obteniendo mejores resultados. De todas maneras estandarizar los datos de entrenamiento y de prueba parece mejorar las métricas de clasificación.

El segundo algoritmo de "elaboración propia" que se construyo fue "voto mayoritario" el cual para realizar una predicción realiza una votación con los algoritmos de Chaki, Hu y Wu, y escoge una especie basándose en la especie de árbol que obtuvo la mayor cantidad de votos por parte de estos algoritmos.

Como se puede observar, en la primera situación este algoritmo parece obtener resultados mejores o iguales a los 3 algoritmos que lo componen. Para la segunda situación en cambio obtiene métricas de *precision* y *recall* perfectas de 1.0 para cada especie analizada, además de una *Accuracy* perfecta. En la tercera y última situación de validación cruzada es claro que, a pesar de que obtiene mejores resultados que Chaki y Hu, las métricas arrojadas por el algoritmo de Wu son superiores cuando se realiza la clasificación de hojas de "Drimys Winteri" y "Peumus Boldus", solamente para "Laurelia Sempevirens" arroja una *precision y recall* inferior a voto mayoritario y aun así la diferencia entre las métricas obtenidas para esta especie es extremadamente pequeña, además hay que recalcar que para esta situación la *Accuracy* de Wu sigue siendo superior. Habiendo dicho esto pareciera que a pesar que voto mayoritario obtiene muy buenos resultados, no existe motivo para escogerlo por sobre Wu, ya que simplemente se puede utilizar dicho algoritmo para realizar la clasificación de las hoja y así obtener mejores resultados de forma consistente en vez de invertir esfuerzo en la elaboración de este algoritmo.

El algoritmo que obtuvo los "peores" resultados comparado a los demás algoritmos es claramente Hu. En la primera situación para la especie "Drymis Winteri" se obtiene el *recall* y *f1-score* más bajos de la tabla, mientras que para "Laurelia Sempervirens" lo mismo ocurre con la métrica de *precision* y *f1-score*. En cambio para la tercera especie, "Pemus Boldus", a diferencia de los resultados anteriores las métricas logradas son altas y no muy diferentes a las obtenidas por los demás algoritmos, solo el *f1-score* permite argumentar que se obtuvieron resultados inferiores a los demás algoritmos. En la segunda situación en cambio se alcanzan resultados perfectos para las métricas de *precision* y *recall* (y por ende *f1-score* también). Para la tercera situación en donde se realiza la validación cruzada, nuevamente para la especie "Drimys Winteri" se obtiene el *recall* y *f1-score* más bajos, mientras que para "Laurelia Sempevirens" se obtiene la peor *precision* y *f1-score*. Por ultimo para la especie "Peumus Boldus" los resultados arrojados para ambas métricas son muy altos y la diferencia con los obtenidos por los demás algoritmos es muy pequeña. La métrica de *Acurracy* calculada para cada situación entrega resultados similares en donde para la primera y tercera situación se obtienen los peores resultados, pero para la segunda situación se obtiene una *Acurracy* perfecta.

Como se mencionó para la segunda situación se obtiene resultados perfectos, pero esto no quiere decir que este algoritmo sea siempre el mejor para esta situación, ya que puede existir la posibilidad de que Hu o los demás algoritmos tengan un buen desempeño específicamente con la configuración del set de entrenamiento y prueba que fue escogida. Como se puede observar en la situación de validación cruzada donde se realiza un análisis más robusto, este algoritmo obtiene las peores métricas en la mayoría de los casos, lo que elimina la ilusión de un buen rendimiento que pueda haber generado la situación 2. Es por esto que para este proyecto de titulo la situación 3 tiene

un peso mucho mayor a la hora de sacar conclusiones en comparación a las demás situaciones.

Analizando los resultados obtenidos en la validación cruzada se puede observar que el algoritmo propuesto por Wu logra los segundos mejores resultados en comparación a los demás algoritmos y logra los mejores resultados para los 3 algoritmos originales que fueron estudiados e implementados. En un principio se esperaba que obtuviera los peores resultados ya que sus vectores de características están formado por solo 12 elementos, en contraste a los 20 y 38 de Hu y Chaki. Como se mencionó, esto no fue así, y este algoritmo alcanzo métricas muy cercanas a CR-MTS-TX. La razón de esto puede ser que simplemente las características extraídas por Wu entregan mucha más información descriptiva de la hoja de árbol, que las características extraídas por Hu o Chaki. Esto podría demostrar la importancia de la calidad en la selección de características.

6 Conclusiones

Como se indica en el título del informe, este proyecto de título se realizó con la idea de estudiar y comparar algoritmos para el reconocimiento visual y la clasificación de hojas de árboles nativos de la Región del Biobío. Para esto fue necesario realizar un dataset de hojas de árboles propio, estudiar material académico relacionado con el tema, implementar y probar el rendimiento de los algoritmos de los autores estudiados y comparar los resultados obtenidos a partir de las pruebas.

Para determinar las características más relevantes de las hojas de árboles con el fin de realizar un reconocimiento de estas, se realizó un estudio de material científico relacionado con el área. Este estudio dejo como conclusión que la característica más relevante a extraer a partir de una imagen de una hoja es la forma de ésta. Sin embargo también se puede complementar la información que esta aporta extrayendo otras características, como la textura de la misma hoja.

Recolectar las 300 hojas de árboles que conforman el set de datos para este proyecto de título fue una tarea sencilla, las hojas fueron recolectadas de los árboles que se encuentran en el campus de la Universidad del Biobío, sede Concepción. A partir del material estudiado se decidió tomar las fotografías de las hojas de árboles en un ambiente controlado, con luz controlada y fondo blanco. Se consideró también la posibilidad de tomar fotografías en condiciones "normales" (no controladas), pero esto demostró no ser una buena idea por lo difícil de aislar la hoja de árbol del fondo de la fotografía. Es por esto que se decidió seguir trabando en un ambiente controlado.

La implementación de los algoritmos que realizan la extracción de características demostró ser, en un principio, una tarea un tanto complicada, ya que, en la mayoría de los casos, solo se depende de lo que expresa el autor sin poder acceder al código fuente de los algoritmos, ni a una explicación más detallada. Para realizar esta tarea se usaron las librerías de manipulación de imágenes para Python como *Skimage* y *OpenCv.*

En contraste implementar los algoritmos de Machine Learning fue tarea sencilla debido a que algoritmos usados por los autores estaban ya implementados en las librerías de Python, *Sklearn* y *Neupy.* La tarea principal entonces fue simplemente entrenar y probar los algoritmos usando las funciones ya integradas en estas librerías.

Analizando los resultados se puede apreciar la importancia de realizar validación cruzada y de no solo quedarse con los resultados obtenidos usando sets de entrenamiento y prueba estáticos. Si solo se hubiera tomado en cuenta la situación I y II, se hubieran alcanzado conclusiones incorrectas respecto al desempeño de los algoritmos. Realizando la validación cruzada se puso comprobar que algunos de los resultados arrojados por los algoritmos eran producto de la configuración específica del set de entrenamiento y testeo, lo que no demostraba correctamente la efectividad de estos algoritmos para la tarea de clasificación de hojas.

Observando los resultados obtenidos se puede concluir que los algoritmos que solo utilizan descriptores de forma para calcular características obtienen resultados inferiores a los alcanzados por el algoritmo CR-MTS-TX, que usa descriptores de forma y textura. Basándose en esta observación puede llegarse a la conclusión de que las características de textura son un aporte en la tarea de clasificación de imágenes de hojas de árboles, y su uso en conjunto a las características de forma puede mejorar en varios puntos las métricas de clasificación como *recall* y *precision* y *Accuracy.*

El algoritmo propuesto por Wu obtiene los mejores resultados entre los 3 algoritmos originalmente estudiados e implementados (Wu, Chaki, Hu). Si bien en un principio se pensó que este algoritmo obtendría los peores resultados por utilizar apenas 12 elementos para construir su vector de características, la experimentación demostró que este juicio previo estaba equivocado. Esto tampoco significa que una menor cantidad de características siempre sea mejor según lo visto en los resultados del algoritmo CR-MTS-TX que usa 62 características. Se podría argumentar entonces , según lo visto en este Proyecto de Título, que es importante escoger características de calidad que entreguen información relevante respecto al objeto que se clasifica, ya que esto podría significar no tener que manejar tantas características, lo que podría bajar el tiempo de computación durante la etapa de entrenamiento.

7 Referencias

Alpaydin, E. (2014). *Introduction to Machine Learning* (Third Edit).

Bianconi, F., & Fernández, A. (2007). Evaluation of the effects of Gabor filter parameters on texture classification. *Pattern Recognition*, *40*(12), 3325–3335.

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc."

Chaki, J., & Parekh, R. (2011). Plant leaf recognition using shape based features and neural network classifiers. *International Journal of Advanced Computer Science and Applications*, *2*(10), 41–47.

Chaki, J., Parekh, R., & Bhattacharya, S. (2015). Plant leaf recognition using texture and shape features with neural classifiers ☆. *Pattern Recognition Letters*, *58*, 61–68.

Conexión Ingenieros. (2017). *Estudio Nacional de Sueldos de Ingenieros 2017*.

Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

Efford, N. (2000). *Digital image processing: a practical introduction using java (with CD-ROM)*. Addison-Wesley Longman Publishing Co., Inc.

Einasto, M., Liivamägi, L. J., Saar, E., Einasto, J., Tempel, E., Tago, E., & Martínez, V. J. (2011). SDSS DR7 superclusters-Principal component analysis. *Astronomy & Astrophysics*, *535*, A36.

Gang Wu, S., Sheng Bao, F., You Xu, E., Wang, Y.-X., Chang, Y.-F., & Xiang, Q.-L. (2007). A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network. In *Signal Processing and Information Technology, 2007 IEEE International Symposium on* (pp. 11–16).

Hall-Beyer, M. (2017). *GLCM Texture: A Tutorial v. 3.0 March 2017*. Retrieved from https://prism.ucalgary.ca/ds2/stream/?#/documents/8f9de234-cc94-401d-b701-f08ceee6cfdf/page/1

Haykin, S. (2004). *Neural Networks: A Comprehensive Foundation*.

Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, *8*(2), 179–187.

Hu, R.-X., Jia, W., Ling, H., & Huang, D. (2012). Multiscale distance matrix for fast plant leaf recognition. *IEEE Trans. Image Processing*, *21*(11), 4667–4672.

Huang, T. (1996). Computer vision: Evolution and promise.

Huang, Z., & Leng, J. (2010). Analysis of Hu's moment invariants on image scaling and rotation. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on* (Vol. 7, pp. V7-476). IEEE.

INE. (2017). *BOLETÍN EXPORTACIONES REGIONALES*. Retrieved from http://www.inebiobio.cl/archivos/files/pdf/exportaciones/2017/6_- Boletín Exportaciones Biobío Junio 2017.pdf

Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(1), 4–37.

Lotfi, A., & Benyettou, A. (2014). A reduced probabilistic neural network for the classification of large databases. *Turkish Journal of Electrical Engineering & Computer Sciences*, *22*(4), 979–989.

Mao, K. Z., Tan, K.-C., & Ser, W. (2000). Probabilistic neural-network structure determination for pattern classification. *IEEE Transactions on Neural Networks*, *11*(4), 1009–1016.

MathWorks. (n.d.). Texture Analysis Using the Gray-Level Co-Occurrence Matrix (GLCM). Retrieved from https://www.mathworks.com/help/images/texture-analysis-using-the-gray-level-co-occurrence-matrix-glcm.html

Mohri, M. (2012). *Foundations of Machine Learning*.

Mukundan, R., & Ramakrishnan, K. R. (1998). *Moment functions in image analysis-theory and applications*. World Scientific.

Nilsson, N. J. (1996). Introduction to machine learning: An early draft of a proposed textbook. USA; Stanford University.

OpenCv. (n.d.). Color Convertions: RGB ↔ GRAY. Retrieved from https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html

OpenCV. (n.d.-a). Adaptive Thresholding. Retrieved from https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html

OpenCV. (n.d.-b). Canny Edge Detector. Retrieved from https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

OpenCV. (n.d.-c). Morphological Transformations. Retrieved from https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, *9*(1), 62–66.

Parsian, M. (2015). *Data Algorithms: Recipes for Scaling Up with Hadoop and Spark*. " O'Reilly Media, Inc."

Placencia, F., & Bañados, F. (2015). Región del Bío Bío lidera inversión en biomasa, "energía del futuro." *Diario Concepcion*. Retrieved from https://www.diarioconcepcion.cl/economia-y-negocios/2017/09/15/region-del-bio-bio-lidera-en-chile-la-pujante-industria-de-biomasa.html

Rakshit, S., Ghosh, A., & Shankar, B. U. (2007). Fast mean filtering technique (FMFT). *Pattern Recognition*, *40*(3), 890–897.

Sawant S., S., & Preeti S., T. (2015). Introduction to Probabilistic Neural Network –Used For Image Classifications. *International Journal of Advanced Research in Computer Science and Software Engineering*, *5*(4), 279–283.

Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Tilburt, J. C., & Kaptchuk, T. J. (2008). Herbal medicine research and global health: an ethical analysis. *Bulletin of the World Health Organization*, *86*(8), 594–599. Retrieved from https://www.minagri.gob.cl/region/viii-region-de-bio-bio/

Vernon, D. (1991). Machine vision-Automated visual inspection and robot vision. *NASA STI/Recon Technical Report A*, *92*.

Zheng, D., Zhao, Y., & Wang, J. (2004). Features extraction using a Gabor filter family. In *Proceedings of the sixth Lasted International conference, Signal and Image processing, Hawaii*.

8 Anexo

8.1 Estudio de Factibilidad

8.1.1 Factibilidad Técnica

Software a utilizar

Pycharm 2018.1.3: Entorno de desarrollo integrado (IDE) para el lenguaje de programación Python. Este IDE será utilizado para el desarrollo de las rutinas para la detección y clasificación de hojas de árboles nativos de la región del Biobío.

Requerimientos mínimos de Hardware

- 4GB de RAM mínimo, 8GB de RAM recomendados
- 1.5 GB de espacio libre en el disco duro + 1 GB para cache.
- Pantalla con una resolución mínima de 1024x768

Librerías Utilizadas

OpenCv 3.4.1: Librería especializada en visión por computadoras más conocido como *computer vision*. Cuenta con una amplia gama de métodos para la manipulación de imágenes, detección de objetos, entre otros.

En este proyecto es utilizada para la limpieza de las imágenes, la detección de contornos, la binarizacion de imágenes, entre otros usos.

Esta librería puede ser descargada de forma gratuita en la página oficial de OpenCv.

Scikit-image 0.14.0: Librería de Python especializada en el procesamiento de imágenes.

En este proyecto cumple la función de proveer métodos que "OpenCv" no posea. Como por ejemplo matrices de coocurrencia de niveles de gris.

Esta librería puede ser obtenida de forma gratuita en la página oficial de scikit-image.

Scikit-learn: Librería especializada en Machine Learning. Permite implementar diversos algoritmos de Clasificación, Regresion y Clustering entre otros.

Se usara esta librería para implementar los algoritmos de Machine Learning utilizados por los autores estudiados.

Esta librería puede ser descargada de forma gratuita en el Github oficial de Scikit-Learn.

Neupy: Librería de Python que permite construir e implementar diversos algortimos de Machine Learning.

Esta librería permitirá implementar algoritmos que no estén disponibles en la librería Scikit-learn.

Puede ser descargada de forma gratuita en el Github oficial de Neupy.

Hardware utilizado

Se cuenta con un equipo computacional propio para el desarrollo, ejecución y prueba del proyecto a realizar.

El equipo es un modelo Acer aspire E 15 e5-575g-548d y cuenta las siguientes características:

- 8GB de RAM
- 500GB de disco duro
- Tarjeta de video Nvidia Geoforce 940MX de 1GB de VRAM dedicada
- Procesador Intel core i5-7200U con 2.5GHz
- Pantalla con resolución de 1366 x 768

Se cuenta con un teléfono móvil Samsung Galaxy J5 que será utilizado para obtener las fotografías de hojas de árboles nativos, para construir así la base de datos de fotografías que será utilizada para el entrenamiento y prueba de los algoritmos de clasificación.

- Las fotografías tomadas para la creación del dataset son de tamaño 2048x1152 pixeles.

Conclusión Factibilidad técnica

El software requerido para el desarrollo del proyecto puede ser obtenido de forma gratuita sin mayores complicaciones, además antes de iniciar el proyecto ya se cuenta con el hardware requerido y este cumple con los requisitos mínimos del software que se va a utilizar. Se concluye entonces que como se cuenta con todas la herramientas necesarias para el desarrollo del Proyecto de Título, que el proyecto es factible técnicamente.

8.1.2 Factibilidad Económica

Recursos a Ocupar

Recursos que serán necesarios para el completo desarrollo del proyecto de título.

- PC Acer Aspire E15, 8GB RAM, 500GB Memoria, Windows10
- Teléfono celular Galaxy J5
- Horas hombre programador.

Evaluación Económica

Costos en hardware

- Celular Samsung Galaxy J5 $ 199.990
- Computador Portatil Acer E 15-575G $ 499.990

Costos en software y licencia

- "Pycharm Comunity edition" puede ser descargado de forma gratuita en la página oficial de JetBrains https://www.jetbrains.com/pycharm/download/
- La Librería "python-weka-wrapper3" puede ser descargada de forma gratuita en la página oficial de la universidad Wakaito https://www.cs.waikato.ac.nz/~ml/weka/
- OpenCV es una librería gratuita Open-Source especializada en Computer vision que puede ser obtenida de forma gratuita en su página oficial https://opencv.org/releases.html
- La librería Scikit-Image pude ser obtenida de forma gratuita en su sitio web oficial https://scikit-image.org

Costos en horas hombre (HH)

HH en desarrollo de aplicación 472 (media jornada x día)

Total de HH 472

Valor HH Ingeniero civil informático $6.800 x hora
(Conexión Ingenieros, 2017) (Calculado para 20 días hábiles por mes, y 8 horas de trabajo al día)

Costo total de horas hombre en Pesos: $ 3.209.600

Horas Hombre	**$ 3.209.600**
Costos de hardware	$ 699.980
Costos de software y licencia	$ 0
MONTO TOTAL	**$3.909.580**

Conclusión factibilidad Económica

Los valores dados son utilizados solo como referencia del costo total que tendría el proyecto en caso de ser desarrollado a pedido, ya que el proyecto al ser un Proyecto de Título no será compensado económicamente y no se pagarán horas hombre, además ya se cuenta con el computador portátil y el teléfono celular requeridos para realizar este proyecto. Por estas razones costo del proyecto es de $0 y por lo tanto en esta instancia es factible económicamente.

8.1.3 Factibilidad Operativa

El proyecto pretende realizar mejoras en la extracción de características y la clasificación de estas para el reconocimiento visual de hojas de árboles, y se desarrolla en un entorno de investigación y desarrollo de soluciones informáticas. Se espera que a futuro los algoritmos presentados en este proyecto puedan servir como una base para futuras investigaciones respecto al reconocimiento de hojas y que los investigadores puedan seguir aportando en el mejoramiento de los algoritmos aquí presentados. Por tanto potenciales usuarios serán investigadores y desarrolladores interesados en mejorar y/o aplicar las técnicas analizadas en el problema de reconocimiento visual de hojas de árboles.

Conclusión factibilidad Operativa

Se espera que el proyecto sea utilizado por investigadores interesados en los algoritmos para el reconocimiento visual de especies, o bien en la mejora de este tipo de algoritmos, por lo que se asume que no deberían tener mayor dificultad para entender este proyecto y para ejecutar las rutinas de reconocimiento. Por lo que se concluye que el proyecto es factible operativamente.

Ya que el proyecto es factible técnicamente, económicamente y operativamente, se concluye por lo tanto que este proyecto de título es factible.

8.2 Resultados Preliminares

Resultados obtenidos por los algoritmos implementados para cada una de las situaciones de prueba.

1) A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network (**Wu**).
2) Plant Leaf Recognition using Shape based Features and Neural Network classifiers (**Chaki**).
3) Multiscale Distance Matrix for Fast Plant Leaf Recognition (**Hu**).
4) Algoritmo Experimental de elaboración propia I (**CR-MTS-TX**).
5) Algoritmo Experimental de elaboración propia II (**Voto Mayoritario**).

8.2.1 Chaki

Situación I

```
MATRIZ DE CONFUSION
[[45  1  4]
 [ 0 50  0]
 [ 0  0 50]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  0.9666666666666667
                      precision    recall  f1-score   support

       drimys_winteri       1.00      0.90      0.95        50
laurelia_sempervirens       0.98      1.00      0.99        50
        peumus_boldus       0.93      1.00      0.96        50

            micro avg       0.97      0.97      0.97       150
            macro avg       0.97      0.97      0.97       150
         weighted avg       0.97      0.97      0.97       150
```

Situación II

```
MATRIZ DE CONFUSION
[[19  1  0]
 [ 0 20  0]
 [ 0  0 20]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  0.9833333333333333
                      precision    recall  f1-score   support

       drimys_winteri       1.00      0.95      0.97        20
laurelia_sempervirens       0.95      1.00      0.98        20
        peumus_boldus       1.00      1.00      1.00        20

            micro avg       0.98      0.98      0.98        60
            macro avg       0.98      0.98      0.98        60
         weighted avg       0.98      0.98      0.98        60
```

Situación III

```
CROSS-VALIDATION CHAKI

              drimys_winteri laurelia_sempervirens peumus_boldus
precision: 0.9169011544011543 0.9396031746031748 0.96
recall:    0.9091666666666667 0.9523931623931624 0.9637662337662338
f1-score:  0.9069168509284944 0.9441116138763199 0.9587931863793934

Accuracy:  0.936666666666668 ± 0.04068851871911233
```

8.2.2 Hu

Situación I

```
MATRIZ DE CONFUSION
[[42  5  3]
 [ 1 49  0]
 [ 1  1 48]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  0.9266666666666666
                      precision    recall  f1-score   support

       drimys_winteri       0.95      0.84      0.89        50
laurelia_sempervirens       0.89      0.98      0.93        50
        peumus_boldus       0.94      0.96      0.95        50

            micro avg       0.93      0.93      0.93       150
            macro avg       0.93      0.93      0.93       150
         weighted avg       0.93      0.93      0.93       150
```

Situación II

```
MATRIZ DE CONFUSION
[[20  0  0]
 [ 0 20  0]
 [ 0  0 20]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  1.0
                      precision    recall  f1-score   support

       drimys_winteri       1.00      1.00      1.00        20
laurelia_sempervirens       1.00      1.00      1.00        20
        peumus_boldus       1.00      1.00      1.00        20

            micro avg       1.00      1.00      1.00        60
            macro avg       1.00      1.00      1.00        60
         weighted avg       1.00      1.00      1.00        60
```

Situación III

```
CROSS-VALIDATION RONGXIANG

              drimys_winteri laurelia_sempervirens peumus_boldus
precision: 0.919040404040404 0.8885703185703185 0.9732167832167832
recall:    0.8713675213675213 0.9470959595959595 0.9432900432900432
f1-score:  0.8871024301822527 0.912595365418895 0.9551383719479697

Accuracy:  0.9233333333333332 ± 0.03349958540373629
```

8.2.3 Wu

Situación I

```
MATRIZ DE CONFUSION
[[45  2  3]
 [ 2 48  0]
 [ 1  0 49]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  0.9466666666666667
                      precision    recall  f1-score   support

       drimys_winteri       0.94      0.90      0.92        50
laurelia_sempervirens       0.96      0.96      0.96        50
        peumus_boldus       0.94      0.98      0.96        50

            micro avg       0.95      0.95      0.95       150
            macro avg       0.95      0.95      0.95       150
         weighted avg       0.95      0.95      0.95       150
```

Situación II

```
MATRIZ DE CONFUSION
[[19  1  0]
 [ 2 18  0]
 [ 0  0 20]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  0.95
                        precision    recall  f1-score   support

       drimys_winteri       0.90      0.95      0.93        20
laurelia_sempervirens       0.95      0.90      0.92        20
        peumus_boldus       1.00      1.00      1.00        20

            micro avg       0.95      0.95      0.95        60
            macro avg       0.95      0.95      0.95        60
         weighted avg       0.95      0.95      0.95        60
```

Situación III

```
CROSS-VALIDATION WU

              drimys_winteri laurelia_sempervirens peumus_boldus
precision: 0.9411471861471862 0.9720238095238095 0.990909090909091
recall:    0.9578199578199579 0.9635642135642136 0.968931068931069
f1-score:  0.9460465812486273 0.9656957468977929 0.978783882783883

Accuracy:  0.9633333333333333 ± 0.02333333333333333
```

8.2.4 CR-MTS-TX

Situación I

```
MATRIZ DE CONFUSION
[[49  1  0]
 [ 0 50  0]
 [ 0  0 50]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  0.9933333333333333
                      precision    recall  f1-score   support

       drimys_winteri       1.00      0.98      0.99        50
laurelia_sempervirens       0.98      1.00      0.99        50
        peumus_boldus       1.00      1.00      1.00        50

            micro avg       0.99      0.99      0.99       150
            macro avg       0.99      0.99      0.99       150
         weighted avg       0.99      0.99      0.99       150
```

Situación II

```
MATRIZ DE CONFUSION
[[20  0  0]
 [ 0 20  0]
 [ 0  0 20]]


METRICAS DE CLASIFICACION PRINCIPALES


accuracy:  1.0
                      precision    recall  f1-score   support

       drimys_winteri       1.00      1.00      1.00        20
laurelia_sempervirens       1.00      1.00      1.00        20
        peumus_boldus       1.00      1.00      1.00        20

            micro avg       1.00      1.00      1.00        60
            macro avg       1.00      1.00      1.00        60
         weighted avg       1.00      1.00      1.00        60
```

Situación III

```
CROSS-VALIDATION PNN

              drimys_winteri laurelia_sempervirens peumus_boldus
precision: 0.9585703185703187 0.9533119658119658 0.9632142857142856
recall:    0.9428535353535354 0.947936507936508 0.9832167832167833
f1-score:  0.9461056562976069 0.9471889194396935 0.971615962984384

Accuracy:  0.9566666666666667 ± 0.029999999999999992
```

8.2.5 Voto Mayoritario

Situación I

```
MATRIZ DE CONFUSION
[[46  1  3]
 [ 0 50  0]
 [ 0  0 50]]

METRICAS DE CLASIFICACION PRINCIPALES

Accuracy: 0.9733333333333334
                      precision    recall  f1-score   support

       drimys_winteri       1.00      0.92      0.96        50
laurelia_sempervirens       0.98      1.00      0.99        50
        peumus_boldus       0.94      1.00      0.97        50

            micro avg       0.97      0.97      0.97       150
            macro avg       0.97      0.97      0.97       150
         weighted avg       0.97      0.97      0.97       150
```

Situación II

```
MATRIZ DE CONFUSION
[[20  0  0]
 [ 0 20  0]
 [ 0  0 20]]

METRICAS DE CLASIFICACION PRINCIPALES

Accuracy: 1.0
                      precision    recall  f1-score   support

       drimys_winteri       1.00      1.00      1.00        20
laurelia_sempervirens       1.00      1.00      1.00        20
        peumus_boldus       1.00      1.00      1.00        20

            micro avg       1.00      1.00      1.00        60
            macro avg       1.00      1.00      1.00        60
         weighted avg       1.00      1.00      1.00        60
```

Situación III

```
              drimys_winteri laurelia_sempervirens peumus_boldus
precision: 0.9347527472527473 0.9800000000000001 0.966837606837607
recall:    0.9376179376179378 0.9652777777777779 0.9523737373737374
f1-score:  0.9309864362690451 0.9704575163398694 0.9574962957117933

Accuracy:  0.9533333333333334 ± 0.03399346342395189
```

8.3 Configuraciones para los algoritmos

En esta sección se presentan las configuraciones específicas de las funciones de los algoritmos implementados (excepto por el algoritmo de voto mayoritario) para cada una de las situaciones de prueba.

1) A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network (**Wu**).
2) Plant Leaf Recognition using Shape based Features and Neural Network classifiers (**Chaki**).
3) Multiscale Distance Matrix for Fast Plant Leaf Recognition (**Hu**).
4) Algoritmo Experimental de elaboración propia I (**CR-MTS-TX**).
5) Algoritmo Experimental de elaboración propia II (**Voto Mayoritario**).

8.3.1 Chaki

Pre procesamiento

Convertir imagen de entrada a imagen en escala de grises:

*img_gray = cvtColor(**src**=img_original,**code**=cv2.COLOR_BGR2GRAY)*

Eliminar el ruido en la imagen en escala de grises resultante:

*cv2.GaussianBlur (**src=** img_gray, **ksize=** (5, 5),**SigmaX=** 0,**dst=** img_gray)*

Convertir imagen en escala de grises a una imagen binaria:

*img_bin = cv2.adaptiveThreshold (**src=**img_gray, **maxValue=**255,**adaptativeThreshold=** cv2.ADAPTIVE_THRESH_GAUSSIAN_C, **thresholdType** = cv2.THRESH_BINARY_INV, **blockSize =** 11, **C =** 2)*

Limpieza de ruido en la imagen binaria:

*img_bin = cv2.morphologyEx (**src =** img_bin, **op =** cv2.MORPH_OPEN, **kernel** = kernel, **iterations**=1)*

*img_bin = cv2.morphologyEx (**src=** img_bin,**op =** cv2.MORPH_CLOSE, **kernel** = kernel, **iterations**=3)*

Encontrar los contornos en la imagen binaria resultante:

*im2, contornos, hierarchy = cv2.findContours(**image**=img_bin, **mode =** cv2.RETR_EXTERNAL, **method =** cv2.CHAIN_APPROX_NONE)*

Encontrar índice dentro de la lista de contornos obtenidos que corresponde al contorno de la hoja de árbol.

index_mayor = self.contornoMayorIdx(contornos)

Extracción de características

Obtener distancias centro:

*centradi = self.distCentRadi(**polígono =** contorno_hoja, **t_intervalo =** 10)*

Obtener Momentos del contorno:

*moments = cv2.moments(**array =** contorno_hoja)*

Obtener Momentos de Hu:

*huMoms = cv2.HuMoments(**m =** moments)*

Entrenamiento del modelo

Crear una instancia de la clase MLPClassifier:

*mlp = MLPClassifier(**hidden_layer_sizes**=(30), **max_iter**=1000)*

Entrenar instancia de perceptron Multicapa

*mlp.fit(**X =** trainSet, **y** = trainSet_lbl)*

Realizar predicciones sobre el set de prueba utilizando el modelo entrenado

*mlp_predictions = mlp.predict(**X =** testSet)*

8.3.2 Hu

Pre procesamiento

Convertir imagen de entrada a imagen en escala de grises:

*img_gray = cvtColor(**src**=img_original,**code**=cv2.COLOR_BGR2GRAY)*

Eliminar el ruido en la imagen en escala de grises resultante:

*cv2.GaussianBlur (**src=** img_gray, **ksize=** (5, 5),**SigmaX=** 0,**dst=** img_gray)*

Convertir imagen en escala de grises a una imagen binaria:

*img_bin = cv2.adaptiveThreshold (**src=**img_gray, **maxValue=**255,**adaptativeThreshold=** cv2.ADAPTIVE_THRESH_GAUSSIAN_C, **thresholdType** = cv2.THRESH_BINARY_INV, **blockSize =** 11, **C =** 2)*

Limpieza de ruido en la imagen binaria:

*img_bin = cv2.morphologyEx (**src =** img_bin, **op =** cv2.MORPH_OPEN, **kernel** = kernel, **iterations**=1)*

*img_bin = cv2.morphologyEx (**src=** img_bin,**op =** cv2.MORPH_CLOSE, **kernel** = kernel, **iterations**=3)*

Encontrar los contornos en la imagen binaria resultante:

*im2, contornos, hierarchy = cv2.findContours(**image**=img_bin, **mode =** cv2.RETR_EXTERNAL, **method =** cv2.CHAIN_APPROX_NONE)*

Encontrar índice dentro de la lista de contornos obtenidos que corresponde al contorno de la hoja de árbol.

index_mayor = self.contornoMayorIdx(contornos)

Extracción de Características

Encontrar 128 puntos en el contorno, que estén separados por la misma distancia

*points = self.separarPoligono(**polígono =** contorno_hoja, **n_points =** 128)*

Mover Verticalmente todas las columnas de la matriz Dm, de forma que los 0 que forman la diagonal queden en la primera fila de la matriz.

*Dm = self.shiftMatrix(**matrix =** Dm)*

Encontrar la mayor distancia centro-radio, para los 128 puntos encontrados.

*largest_dist_ctd = self.findLargestDistancetoCentroid(**point**s = points , **cx =** cx, **cy =** cy)*

Entrenamiento del modelo

Crear instancia de la clase KNeighborsClassiefier.

*knn = KNeighborsClassifier(**n_neighbors**=15)*

Crear instancia de la clase PCA, especificando que se quiere conservar 20 componentes.

*pca = PCA(**n_components**=20)*

Extraer los componentes principales a partir del set de datos de entrenamiento.

*pca.fit(**X** = trainSet)*

Proyectar el set de entrenamiento y de prueba sobre los componentes principales extraídos en la función anterior.

*trainSet = pca.transform(**X =** trainSet)*

*testSet = pca.transform(**X =** testSet)*

Entrenar la instancia de KNN usando el set de entrenamiento.

*knn.fit(**X =** trainSet, **y** = trainSet_lbl)*

Realizar predicciones sobre el set de prueba usando el algoritmo entrenado.

*knn_predictions = knn.predict(**X** = testSet)*

8.3.3 Wu

Pre procesamiento

Convertir imagen de entrada a imagen en escala de grises:

*img_gray = cvtColor(**src**=img_original,**code**=cv2.COLOR_BGR2GRAY)*

Eliminar el ruido en la imagen en escala de grises resultante:

*cv2.GaussianBlur (**src=** img_gray, **ksize=** (5, 5),**SigmaX=** 0,**dst=** img_gray)*

Convertir imagen en escala de grises a una imagen binaria:

*img_bin = cv2.adaptiveThreshold (**src=**img_gray, **maxValue=**255,**adaptativeThreshold=** cv2.ADAPTIVE_THRESH_GAUSSIAN_C, **thresholdType** = cv2.THRESH_BINARY_INV, **blockSize =** 11, **C =** 2)*

Limpieza de ruido en la imagen binaria:

*img_bin = cv2.morphologyEx (**src =** img_bin, **op =** cv2.MORPH_OPEN, **kernel** = kernel, **iterations**=1)*

*img_bin = cv2.morphologyEx (**src=** img_bin,**op =** cv2.MORPH_CLOSE, **kernel** = kernel, **iterations**=3)*

Encontrar los contornos en la imagen binaria resultante:

*im2, contornos, hierarchy = cv2.findContours(**image**=img_bin, **mode =** cv2.RETR_EXTERNAL, **method =** cv2.CHAIN_APPROX_NONE)*

Encontrar índice dentro de la lista de contornos obtenidos que corresponde al contorno de la hoja de árbol.

index_mayor = self.contornoMayorIdx(contornos)

Extracción de Características

Crear envolvente convexa

*hull = cv2.convexHull(**points =** contorno_hoja)*

Encontrar Diámetro

*points = self.findDiametro(**cnts =** hull)*

Mover el contorno de la hoja al centro del sistema de coordenadas, con el objetivo de calcular coordenadas polares

*contorno_hoja_origen = self.moveraCentro(**shape =** contorno_hoja)*

Rotar el contorno de la hoja de forma que el diámetro quede alineado con el eje Y.

*contorno_hoja_recto = self.rotarPoligono(**polígono =** contorno_hoja_origen, **angle =** comp_angle)*

Encontrar los 2 puntos más extremos respecto al eje X, estos son el punto más positivo y el más negativo.

*pizq, pder = self.findHorizontalExtremes(***contorno_hoja** = *contorno_hoja_recto)*

Encontrar los puntos complementarios para cada punto extremo.

*pizqcomp = self.findHorizontal(***polígono** = *contorno_hoja_recto, **pt** = pizq, **umbral** = pt1)*

*pdercomp = self.findHorizontal(***polígono** = *contorno_hoja_recto, **pt** = pder, **umbral** = pt1)*

Encontrar diámetro de la hoja.

diameter = self.Distance(pt1, pt2)

Encontrar Ancho Fisiológico.

psyWidth = self.Distance(pwidth[0], pwidth[1])

Encontrar área del contorno de la hoja.

*leafArea = cv2.contourArea(**contour =** contorno_hoja)*

Crear imagen 1 e identificar contorno para calcular factor de suavidad

*smooth1 = cv2.blur(**src** = img_bin, **ksize =** (2, 2))*

*smooth1 = cv2.Laplacian(**src** = smooth1, **ddepth** = cv2.CV_8UC1, **ksize**=3)*

*im2, contornos_s1, hierarchy = cv2.findContours (**image** = smooth1, **mode =** cv2.RETR_EXTERNAL, **method =** cv2.CHAIN_APPROX_NONE*

index_cnt_s1 = self.contornoMayorIdx(contornos_s1)

Crear imagen 2 e identificar contorno para calcular factor de suavidad

*smooth2 = cv2.blur(**src** = img_bin, **ksize =** (5, 5))*

smooth2 = cv2.Laplacian(smooth2, cv2.CV_8UC1, ksize=3)

*im2, contornos_s2, hierarchy = cv2.findContours(**image =** smooth2, **mode =** cv2.RETR_EXTERNAL, **method =** cv2.CHAIN_APPROX_NONE)*

index_cnt_s2 = self.contornoMayorIdx(contornos_s2)

Proceso de apertura morfológica con el fin de crear 4 imágenes que serán utilizadas con el fin de resaltar la vena de la hoja de distintas formas.

*img_gray_op1 = cv2.morphologyEx (**src** = img_gray, **op** = cv2.MORPH_OPEN, **kernel** = k1)*

*img_gray_op2 = cv2.morphologyEx (**src** = img_gray, **op** = cv2.MORPH_OPEN, **kernel** = k2)*

*img_gray_op3 = cv2.morphologyEx (**src** = img_gray, **op** = cv2.MORPH_OPEN, **kernel** = k3)*

*img_gray_op4 = cv2.morphologyEx (**src** = img_gray, **op** = cv2.MORPH_OPEN, **kernel** = k4)*

Proceso para crear 4 imágenes en donde resalte la vena.

*av1 = cv2.subtract (**src1** = img_gray, **src2** = img_gray_op1)*

*av2 = cv2.subtract (**src1** = img_gray, **src2** = img_gray_op2)*

*av3 = cv2.subtract (**src1** = img_gray, **src2** = img_gray_op3)*

*av4 = cv2.subtract (**src1** = img_gray, **src2** = img_gray_op4)*

Aplicar mascara binaria para que solo sean visibles los pixeles que se encuentran dentro del contorno de la hoja.

*av1 = cv2.bitwise_or (**src** = av1, **dst =** av1, **mask**=mask)*

*av2 = cv2.bitwise_or (**src** = av2, **dst =** av2, **mask**=mask)*

*av3 = cv2.bitwise_or (**src** = av3, **dst** = av3, **mask**=mask)*

*av4 = cv2.bitwise_or (**src =** av4, **dst =** av4, **mask**=mask)*

Convertir cada una de las imágenes en donde se exhiben las venas de la hoja a una imagen binaria.

*ret, img_bin_1 = cv2.threshold(**src** = av1, **thresh** = 0, **maxValue** = 255, **type** = cv2.THRESH_BINARY + cv2.THRESH_OTSU)*

*ret, img_bin_2 = cv2.threshold(**src** = av2, **thresh** = 0, **maxValue** = 255, **type** = cv2.THRESH_BINARY + cv2.THRESH_OTSU)*

*ret, img_bin_3 = cv2.threshold(**src** = av3, **thresh** = 0, **maxValue** = 255, **type** = cv2.THRESH_BINARY + cv2.THRESH_OTSU)*

*ret, img_bin_4 = cv2.threshold(**src** = av4, **thresh** = 0, **maxValue** = 255, **type** = cv2.THRESH_BINARY + cv2.THRESH_OTSU)*

Entrenamiento del modelo

Crear instancia de PNN.

pnn = PNN()

Crear instancia de la clase PCA, especificando que se quiere conservar 5 componentes.

*pca = PCA(**n_components**=5)*

Extraer los componentes principales a partir del set de datos de entrenamiento.

*pca.fit(**X** = trainSet)*

Proyectar el set de entrenamiento y de prueba sobre los componentes principales extraídos en la función anterior.

*trainSet = pca.transform(**X =** trainSet)*

*testSet = pca.transform(**X =** testSet)*

Entrenar la instancia de PNN usando el set de entrenamiento.

*pnn.fit(**X =** trainSet, **y** = trainSet_lbl)*

Realizar predicciones sobre el set de prueba usando el algoritmo entrenado.

*pnn_predictions = pnn.predict(**X** = testSet)*

8.3.4 CR-MTS-TX

Pre procesamiento

Convertir imagen de entrada a imagen en escala de grises:

*img_gray = cvtColor(**src**=img_original,**code**=cv2.COLOR_BGR2GRAY)*

Eliminar el ruido en la imagen en escala de grises resultante:

*cv2.GaussianBlur (**src=** img_gray, **ksize=** (5, 5),**SigmaX=** 0,**dst=** img_gray)*

Convertir imagen en escala de grises a una imagen binaria:

*img_bin = cv2.adaptiveThreshold (**src=**img_gray, **maxValue=**255,**adaptativeThreshold=** cv2.ADAPTIVE_THRESH_GAUSSIAN_C, **thresholdType** = cv2.THRESH_BINARY_INV, **blockSize =** 11, **C =** 2)*

Limpieza de ruido en la imagen binaria:

*img_bin = cv2.morphologyEx (**src =** img_bin, **op =** cv2.MORPH_OPEN, **kernel** = kernel, **iterations**=1)*

*img_bin = cv2.morphologyEx (**src=** img_bin,**op =** cv2.MORPH_CLOSE, **kernel** = kernel, **iterations**=3)*

Encontrar los contornos en la imagen binaria resultante:

*im2, contornos, hierarchy = cv2.findContours(**image**=img_bin, **mode =** cv2.RETR_EXTERNAL, **method =** cv2.CHAIN_APPROX_NONE)*

Encontrar índice dentro de la lista de contornos obtenidos que corresponde al contorno de la hoja de árbol.

index_mayor = self.contornoMayorIdx(contornos)

Extracción de características

Obtener distancias centro:

*centradi = self.distCentRadi(**polígono =** contorno_hoja, **t_intervalo =** 10)*

Obtener Momentos del contorno:

*moments = cv2.moments(**array =** contorno_hoja)*

Obtener Momentos de Hu:

*huMoms = cv2.HuMoments(**m =** moments)*

Computar GLCM

*glcm = greycomatrix(**image** = cropped_leaf, **distances =** [1], **angles** = [0, np.pi/4, np.pi/2, 3*np.pi/4], **levels** = 256, **Symmetric** = True, **Normed =** True)*

Computar propiedades a partir de la GLCM

*contrast = greycoprops(**P** = glcm, **prop** = 'contrast')*

*dissimilarity = greycoprops(**P** = glcm, **prop** = 'dissimilarity')*

*homogenity = greycoprops(**P** = glcm, **prop** = 'homogeneity')*

*asm = greycoprops(**P** = glcm, **prop** = 'ASM')*

*energy = greycoprops(**P** = glcm, **prop** = 'energy')*

*correlation = greycoprops(**P** = glcm, **prop** = 'correlation')*

Entrenamiento del modelo

Crear una instancia de la clase *StandardScaler*

scaler = StandardScaler()

Computar media y desviación estándar que se usara para escalar los datos

*scaler.fit(**X** = trainSet)*

Realizar estandarización de los datos

*trainSet = scaler.transform(**X** = trainSet)*

*testSet = scaler.transform(**X** = testSet)*

Crear una instancia de la clase MLPClassifier:

*mlp = MLPClassifier(**hidden_layer_sizes**=(30), **max_iter**=1000)*

Entrenar instancia de perceptron Multicapa

*mlp.fit(**X =** trainSet, **y** = trainSet_lbl)*

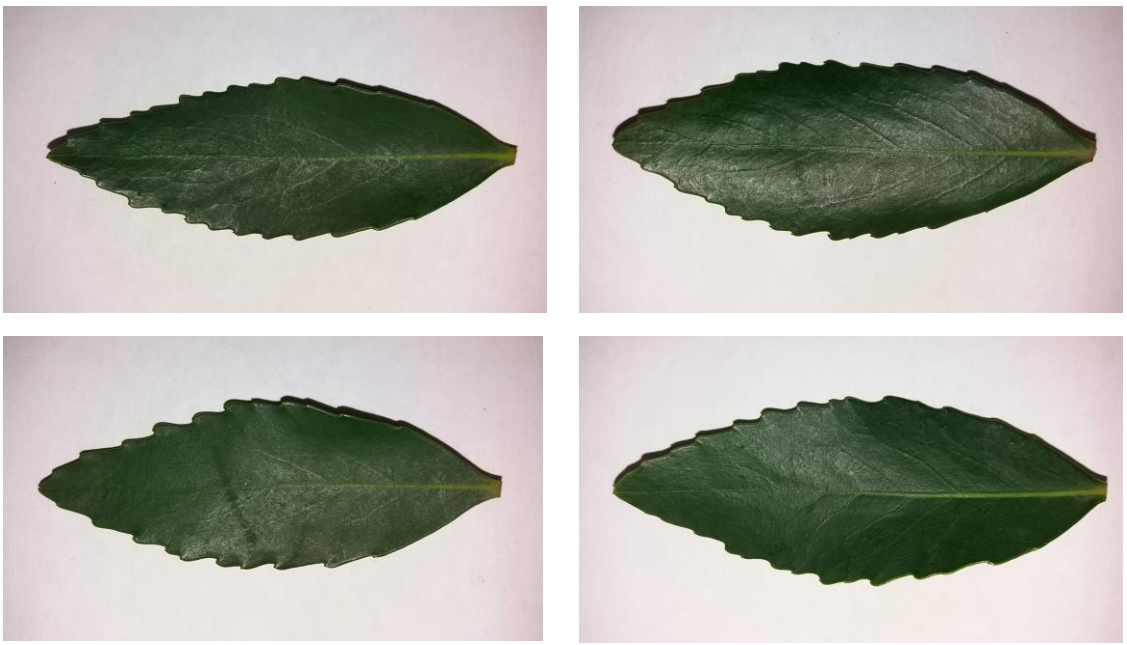
Realizar predicciones sobre el set de prueba utilizando el modelo entrenado

*mlp_predictions = mlp.predict(**X =** testSet)*

8.4 Muestra de fotografías tomadas de forma propia para la construcción del dataset

8.4.1 Drymis Winteri (Canelo Chileno)

8.4.2 Laurelia Sempervirens (Laurel Chileno)



8.4.3 Peumus Boldus (Boldo)