COLEGIO DE POSTGRADUADOS

INSTITUCIÓN DE ENSEÑANZA E INVESTIGACIÓN EN CIENCIAS AGRÍCOLAS

CAMPUS MONTECILLO

POSTGRADO DE SOCIOECONOMÍA, ESTADÍSTICA E INFORMÁTICA

CÓMPUTO APLICADO

EVALUACIÓN DEL DESEMPEÑO DE DOS MODELOS DE REDES NEURONALES ARTIFICIALES PARA CLASIFICAR FLORES DE *Petunia* spp CON BASE EN COLOR

MARCELA CASTILLO LOAIZA

T E S I S

PRESENTADA COMO REQUISITO PARCIAL
PARA OBTENER EL GRADO DE:

MAESTRA EN CIENCIAS

MONTECILLO, TEXCOCO, EDO. DE MÉXICO

2013

La presente tesis titulada: **Evaluación del desempeño de dos modelos de redes neuronales artificiales para clasificar flores de *Petunia* spp con base en color** realizada por la alumna: **Marcela Castillo Loaiza**, bajo la dirección del Consejo Particular indicado, ha sido aprobada por el mismo y aceptada como requisito parcial para obtener el grado de:

MAESTRA EN CIENCIAS
SOCIOECONOMÍA ESTADÍSTICA E INFORMÁTICA

CÓMPUTO APLICADO

CONSEJO PARTICULAR

CONSEJERO
DR. JUAN MANUEL GONZÁLEZ CAMACHO

ASESOR
DR. JOSÉ ALFREDO CARRILLO SALAZAR

ASESOR
M.C. JUAN JOSÉ ESCOBAR AGUAYO

ASESOR
DR. JOEL AYALA DE LA VEGA

Montecillo, Texcoco, Estado de México, Noviembre de 2013.

EVALUACIÓN DEL DESEMPEÑO DE DOS MODELOS DE REDES NEURONALES ARTIFICIALES PARA CLASIFICAR FLORES DE Petunia spp CON BASE EN COLOR

Marcela Castillo Loaiza, M.C.
Colegio de Postgraduados, 2013

En esta investigación se presenta la comparación en el desempeño de dos modelos de redes neuronales artificiales, el modelo perceptrón multicapa (MLP, por sus siglas en inglés *Multilayer Perceptron*) y la red neuronal probabilística (PNN, por sus siglas en inglés *Probabilistic Neural Network*) para clasificar imágenes digitales de flores de dos variedades de *Petunia* con base en diferentes espacios de color. Las variedades son *Petunia multiflora* (varios colores) y *Petunia púrpura* (un color de flor). Los datos de entrada se obtuvieron a partir de muestras de imagen correspondientes a áreas de flor, hoja y fondo. Cada pixel de las muestras de imagen se transformaron a los espacios de color RGB, LCH y CIE Lab y se asociaron tres clases para la variedad púrpura (color de flor, hoja y fondo) y siete clases para la variedad multiflora (5 colores de flor, hoja y fondo). Se evaluaron diferentes escenarios de modelación para seleccionar las funciones de transferencia, algoritmos de entrenamiento, número de capas ocultas y neuronas del modelo MLP y el parámetro de suavizado del modelo PNN. Se utilizó como criterio de desempeño el porcentaje global de clasificación promedio de 25 particiones aleatorias del conjunto de datos y la matriz de confusión. En general los dos clasificadores presentan buenas eficiencias de clasificación, en promedio el modelo MLP obtuvo un 98.4% de clasificación correcta frente a la red PNN con un 97.1% de clasificación correcta; sin embargo, la red PNN fue superior a la red MLP, debido a su velocidad, la cual en promedio alcanza los 174.75 segundos frente a la MLP con 54589 segundos.

Palabras clave: *red neuronal probabilística, perceptrón multicapa, imágenes digitales, espacios de color.*

ASSESSEMENT OF THE PERFORMANCE OF THE ARTIFICAL NEURAL NETWORKS TO CLASSIFY Petunia spp's FLOWERS BASED ON COLOR

Marcela Castillo Loaiza, M.C.
Colegio de Postgraduados, 2013

In this research, a comparison of the performance of two artificial neural networks is presented; the multilayer perceptron model (MLP) and the probabilistic neural network (PNN) to classify digital images of two varieties of Petunia's flowers based on different color spaces. Varieties were Petunia multiflora (various colors) and Purple Petunia (one flower's color). The input data were obtained from samples corresponding to image areas of flowers, leaves and background. Each pixel of the image samples were transformed to color spaces RGB, CIE Lab and LCH and; three classes for the purple variety (color of flower, leaf and background) and seven classes for variety multiflora (5 colors flower associated leaf and background). Different modeling scenarios were evaluated to select the transfer functions, training algorithms, number of hidden layers and neurons in the MLP model and the smoothing parameter of the PNN model. The overall percentage of classification (mean of 25 random partitions of the data set) and the confusion matrix were used as a performance criterion. In general the two classifiers had good classification accuracy, in average the MLP model gave 98.4 % correct classification against PNN network with 97.1 % of correct classification; however, the PNN network was faster than the MLP model; for example in some cases, PNN had an average of 75 seconds versus 54,589 of the MLP.

Keywords*:* *probabilistic neural network, multilayer perceptron, digital images, color espaces.*

AGRADECIMIENTOS

Quiero agradecer al Consejo Nacional de Ciencia y Tecnología (CONACyT), por el apoyo económico, que hizo posible el desarrollo de mis estudios de maestría. Al Consejo Mexiquense de Ciencia y Tecnología por haber financiado este trabajo de investigación.

Quiero dar las gracias al pueblo de México, al Colegio de Postgraduados, y a mis profesores y profesoras que he tenido durante la maestría, por todo lo que me han enseñado.

Quiero agradecer a mi mamá Lia Loaiza Almaraz, a mis tíos el Pintor Hugo Loaiza Almaraz y su esposa Dra. Silvia Pimentel Aguilar, por el apoyo en momentos difíciles. Dedicada a la memoria de mis abuelos Ana Almaraz de los Ángeles † y Jaime Loaiza Vizcarra †.

Al Dr. José Alfredo Carrillo Salazar por sus ideas y opiniones, por su paciencia, y por haber hecho todo lo que estuvo en sus manos para la realización de esta tesis.

Al M.C Juan José Escobar Aguayo por toda su amabilidad y su disposición para apoyar éste trabajo en todo momento.

Al Dr. Joel Ayala de la Vega por sus útiles consejos, disponibilidad, porque siempre me apoyó para cumplir con este compromiso, por lo que he aprendido gracias a Ud.

Quiero dar las gracias al Dr. Juan Manuel González Camacho por su infinita paciencia, por su voluntad, porque siempre necesité su supervisión y siempre la tuve, por su labor de dirección sin la cual esta tesis no se hubiera podido realizar, y por sus valiosos consejos durante la maestría que me fueron siempre tan imprescindibles.

CONTENIDO

LISTA DE FIGURAS

LISTA DE CUADROS

1. INTRODUCCIÓN

1.1 ANTECEDENTES

Las redes neuronales son modelos matemáticos no paramétricos que han sido utilizadas en diversos campos de la ciencia, en particular, en la agronomía. Concretamente, para esta investigación se han explorado dichas técnicas en la clasificación de plantas ornamentales [Timmermans y Huizebosch, 1995].

Desde la década de los 40's se hicieron intentos para crear máquinas que realizaran tareas con cierta inteligencia. En 1956 nace formalmente la Inteligencia Artificial (IA). Una de sus principales ramas son las Redes Neuronales Artificiales (ANN por sus siglas en inglés Artificial Neural Networks). La primera relación entre el cerebro y la computación se debe a Alan Turing. En sus teorías, Turing probó que el cerebro humano es en esencia una máquina; sin embargo, los fundamentos de la computación neuronal se deben al neurofisiólogo Warren McCulloch, y el matemático Walter Pitts quienes en 1943 modelaron una red neuronal simple. En 1957 el psicólogo Frank Rosenblatt desarrolla el perceptrón, que es una variación del modelo de McCulloch-Pitts añadiéndole aprendizaje. De todos estos estudios surge un nuevo paradigma desarrollado por Specht, (1988) llamado Redes Neuronales Probabilísticas (PNN por sus siglas en inglés *Probabilistic Neural Network*). Existen dos áreas de la IA que son los ejes de la presente investigación de tesis. La primera se trata de las redes neuronales artificiales, las cuales son modelos simplificados de los sistemas biológicos. Por ejemplo, las sinapsis de una ANN se representan por conexiones relativamente simples, en las que se realiza una suma ponderada de las entradas para activar una función de activación o umbral; mientras que, en las redes neuronales biológicas, las sinapsis tienen miles de componentes y de procesos para la propagación de impulsos electro-químicos. Por otro lado, la segunda área es el Procesamiento Digital de Imágenes (PDI), que es un conjunto de técnicas que tienen la finalidad de efectuar operaciones sobre imágenes digitales, para obtener información a partir de ellas. La primera aplicación del PDI, surgió en 1920 gracias a la impresión de periódicos ya que una imagen tardaba en ser transmitida

analógicamente de Londres a Nueva York una semana y mediante la codificación y transmisión digital por cable submarino, ese tiempo se redujo a tres horas.

1.2 PLANTEAMIENTO DEL PROBLEMA

En este trabajo se aborda el problema de la clasificación automatizada de imágenes digitales de plantas ornamentales con base en su color, mediante la aplicación de ANN. Esta técnica es útil en el tratamiento de imágenes digitales para realizar un proceso de clasificación cuya finalidad es la obtención de patrones que permitan distinguir colores para clasificar flores. Los clasificadores no paramétricos en general presentan mejor desempeño que los clasificadores paramétricos como el análisis discriminante.

Se ha encontrado que hay una gran dificultad para diseñar la arquitectura de un modelo de red perceptrón multicapa, muchas veces, depende de la naturaleza de la aplicación y de la experiencia del investigador. En cuanto a la red neuronal probabilística, la principal dificultad es cómo determinar el parámetro de suavizado adecuado para un problema particular. Por lo que esta investigación trata de ofrecer pruebas realizadas sobre los parámetros ajustables en ambas ANN que ayudarán a entender la naturaleza de las redes en el problema de clasificación de patrones basado en el color de los pixeles en una imagen digital. Saber si las redes sirven para abordar este tipo de problemas y cuál de las dos es más eficiente en la tarea de clasificación.

1.3 JUSTIFICACIÓN

Diversos estudios muestran la importancia de aplicar ANN en la solución de problemas de aproximación de funciones y de clasificación [Parthiban y Subramanian, 2009]. En particular en la clasificación de pixeles de imágenes digitales. Es importante examinar ésta y otras técnicas de procesamiento de imágenes y redes neuronales que participen en la identificación de patrones en imágenes, y obtener más conocimiento sobre la eficacia de las técnicas de tratamiento de imágenes en conjunción con las técnicas de ANN. Lo relevante es en qué medida una técnica de clasificación es más eficiente para

distinguir entre un grupo de datos y otro. Al tener dos variedades de flores de *Petunia* spp los dos modelos de redes responderán de forma distinta, es importante reportar como las ANN responden y en qué medida una es más idónea que la otra en este caso. Esta tesis del área de computación se enfoca a la evaluación de la eficiencia de la clasificación de imágenes con base en el color mediante la aplicación de dos modelos de redes neuronales artificiales. Averiguar el grado de eficiencia para la clasificación de estos dos modelos de ANN ayudará a elegir el mejor camino, al reducir el tiempo de interpretación de la imagen y hacerlo más preciso, para investigaciones futuras.

1.4 OBJETIVO GENERAL

Comparar el desempeño en la clasificación de dos modelos de redes neuronales artificiales para clasificar imágenes digitales de flores de *Petunia* spp con base en color.

1.4.1 Objetivos específicos

1. Obtener de pixeles de imágenes digitales de flores de petunia los valores de los canales de los siguientes espacios de color: RGB, CIE Lab y LCH, para crear los conjuntos de entrenamiento, prueba y validación de las redes neuronales artificiales.
2. Implementar una red neuronal perceptrón multicapa (MLP por sus siglas en Inglés, multilayer perceptron) en la plataforma de modelación Matlab para clasificar imágenes digitales de flores de *Petunia* spp.
3. Implementar una red neuronal probabilística con funciones Gaussianas de base radial en la plataforma de modelación Matlab para clasificación de imágenes digitales de flores de *Petunia* spp.
4. Evaluar y comparar el desempeño de los modelos de red neuronal MLP y PNN mediante matrices de confusión.

1.5 HIPÓTESIS

Es posible mejorar la eficiencia de clasificación de las imágenes digitales de flores de *Petunia* con el entrenamiento supervisado de las redes neuronales artificiales.

En general, la red neuronal PNN tiene un mejor desempeño que la red neuronal MLP.

1.6 ESTRUCTURA DE LA TESIS

La presente tesis está conformada por cinco capítulos que se ilustran en el Cuadro 1.1 La parte 1 trata sobre la introducción que está constituida por los antecedentes, planteamiento del problema, justificación, objetivos e hipótesis.

Cuadro 1.1 Estructura de la tesis.

Parte 1	Parte 2	Parte 3	Parte 4	Parte 5
Introducción	**Revisión de literatura**	**Materiales y Métodos**	**Análisis de Resultados**	**Conclusiones**
Revisión general de la tesis	Explicación de las técnicas de PDI y ANN	Materiales experimentales. Descripción, diseño, codificación de modelos de ANN	Evaluación, comprobación y análisis de los modelos propuestos	Integración de la investigación

La parte 2 corresponde a la revisión de literatura dividido en dos secciones que abarcan los temas fundamentales de este trabajo el procesamiento digital de imágenes y las redes neuronales artificiales. La parte 3 trata sobre la adquisición de las imágenes digitales utilizadas en la investigación; describe los modelos ANN propuestos, y su implementación en Matlab. La parte 4 presenta los resultados obtenidos, su análisis e interpretación. Por último, la parte 5 expone las conclusiones derivadas de la investigación.

2. REVISIÓN DE LITERATURA

2.1 PROCESAMIENTO DIGITAL DE IMÁGENES

Una cámara fotográfica permite capturar una imagen y generar una señal de salida. La señal digitalizada se almacena en la memoria de la computadora, y luego se procesa para poder deducir ciertas características de la imagen mediante el empleo de algoritmos. Este proceso se conoce como PDI. En este capítulo se presentan los aspectos más relevantes de dichas técnicas.

El procesamiento digital de imágenes es un proceso de automatización de la percepción visual mediante el tratamiento de imágenes digitales. En el procesamiento de los datos de una escena para su percepción por una máquina de forma autónoma, se asocia el concepto de visión artificial. El procesamiento de imágenes contempla tanto, técnicas para mejorar la calidad de las imágenes como las relativas a la percepción de una máquina. “La visión artificial por computadora es la capacidad de la máquina para ver el mundo que le rodea, más precisamente para deducir la estructura y las propiedades del mundo tridimensional a partir de una o más imágenes bidimensionales” [Pajares y De la Cruz, 2008]. Esto es, el término visión artificial es un proceso para adquirir información visual por medio de una máquina y utilizarla para ciertas tareas. La transformación de una imagen en información digital básicamente tiene las siguientes etapas [Iñigo y Angulo, 1986]:

- Obtener una imagen captada por algún elemento fotosensible.
- Sobre dicha imagen se proyecta una retícula, que la divide en cuadros pequeños. En cada uno se mide la luminosidad.
- Con los niveles luminosos se obtiene una matriz de valores que expresan la luminosidad de cada punto. Estos niveles representan las tonalidades de gris.
- La composición de la imagen se realiza implementando sobre su matriz de cuadros los niveles de gris.

Una imagen a color se toma con una cámara con filtros para los colores rojo, verde y azul. Cada elemento básico de información comprende tres parámetros, uno para cada color. Las etapas fundamentales del procesamiento de imágenes son:

1. Adquisición de la imagen: se obtienen imágenes del objeto de interés a través de una cámara digital, luego se almacenan en la memoria de la computadora.
2. Preprocesamiento: se eliminan factores negativos, como el ruido producido durante la adquisición de la imagen, utilizando técnicas de mejora, restauración o descompresión (compresión).
3. Segmentación: se separan los objetos de interés del resto de la imagen.
4. Extracción de características: se transforma la imagen en una lista de atributos o parámetros que también se almacenan en la memoria.
5. Interpretación y reconocimiento: se usa la lista de atributos para la tarea de reconocimiento.

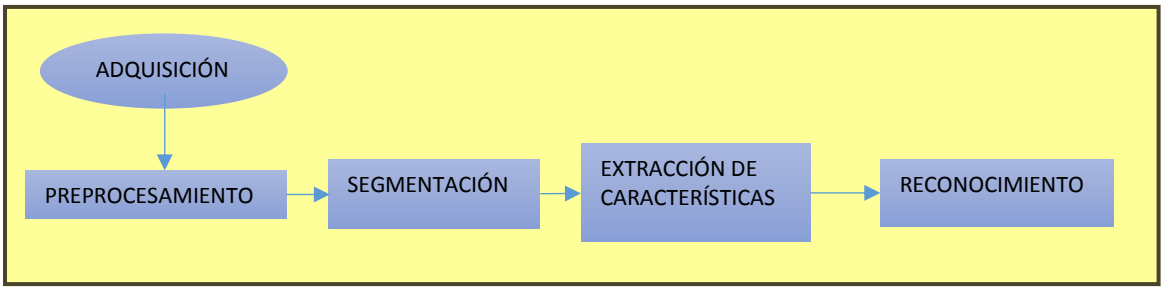


Figura 2.1 Etapas principales del procesamiento digital de imágenes.

Las cuatro áreas de aplicación de la visión artificial son: obtención de la distancia de los objetos en la escena tridimensional y estructura tridimensional; detección de objetos en movimiento; reconocimiento de patrones y formas; y reconocimiento de objetos tridimensionales [Pajares y De la Cruz, 2008].

2.1.1 Imagen digital

La imagen tratada por la computadora se presenta espacialmente digitalizada mediante una matriz de $m \times n$ elementos. Cada elemento discreto de la matriz se denomina píxel. Un píxel (acrónimo del inglés *picture element*) es la menor unidad homogénea en color más pequeña en color que forma parte de una imagen digital. Un píxel es sensible a la luz y tiene un valor definido, que es el nivel de luminosidad del punto correspondiente en la escena captada, dicho valor es la cantidad de intensidad o nivel de gris.Una imagen puede definirse como una función de intensidad bidimensional, representada por $f(x,y)$, donde x e y son las coordenadas espaciales y el valor de f en cualquier punto (x,y) es proporcional a la intensidad o nivel de gris en la imagen en ese punto [Pajares y De la Cruz, 2008]. La convención de ejes elegida se muestra en Figura 2.2.

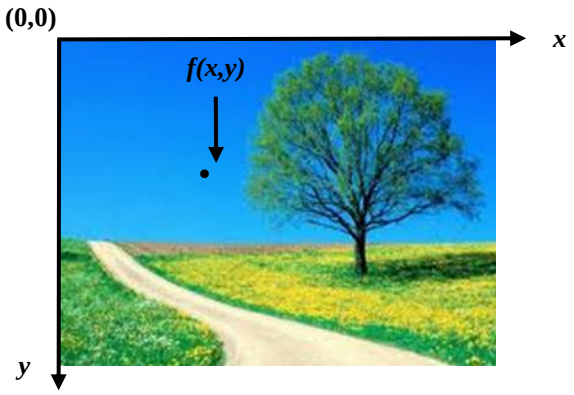


Figura 2.2 Convención de ejes utilizada para la representación de imágenes digitales.

2.1.2 Transformación de imágenes

Una imagen mediante una transformación genera una nueva imagen. El procesamiento de imágenes puede verse como la transformación de una imagen en otra imagen; mientras que el análisis de imágenes es la transformación de una imagen en algo distinto a una imagen, más bien en información para describir o decidir. El propósito del procesamiento de imágenes es hacer el análisis posterior más simple, existen transformaciones típicas.

1. Individuales: cuando se aplica una regla sobre el valor numérico de los pixeles.
2. De vecindad: donde el valor de los pixeles se modifica en función del valor de los píxeles vecinos. Las operaciones de vecindad son filtrado, suavizado y extracción de bordes.
3. Otras: aquellas que operan globalmente sobre los valores de intensidad de la imagen cuyo efecto es un realzado de la imagen original, además operaciones aritméticas y lógicas (basadas en algebra de *Boole*) y operaciones que realizan transformaciones geométricas, sin modificar los valores de intensidad.

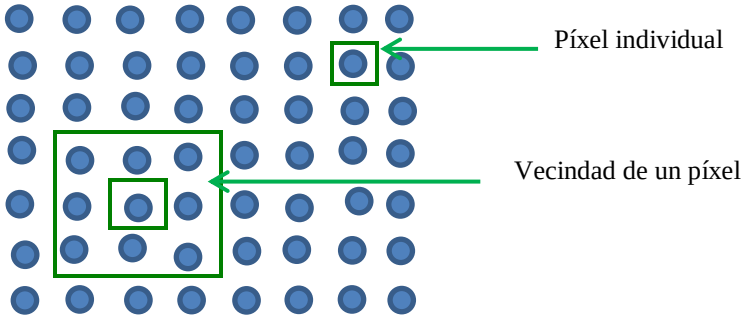


Figura 2.3 Ejemplo de operaciones: individual y de vecindad.

2.1.2.1 Transformaciones individuales

Las operaciones individuales, con operadores tales como: identidad, inverso y, umbral, entre otros, consisten en tomar una imagen y modificar cada píxel, basándose en una regla global aplicada a la imagen completa generando así una nueva imagen.

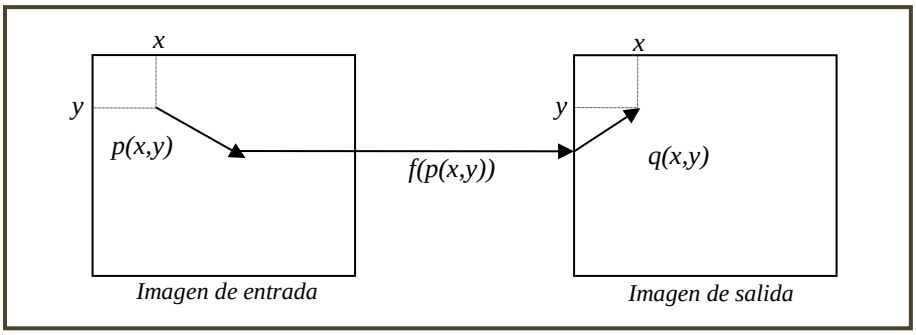


Figura 2.4 Una operación individual de imágenes.

En la Figura 2.4 un píxel se modifica mediante una operación lineal o no lineal y queda en una localización equivalente a la localización de la imagen original en la nueva imagen. El operador individual f es una transformación uno a uno y se aplica a cada píxel de la imagen o sección de la imagen. La ecuación de transformación se escribe:$q(x, y) = f(p(x, y))$.

2.1.2.2 Transformaciones de vecindad

Funcionalmente las operaciones de vecindad se dividen en: suavizado y extracción de bordes. En este trabajo se enfatiza en las operaciones de suavizado, puesto que lo que interesa del preprocesamiento es mejorar la calidad de la imagen, esto se logra mediante el suavizado; mientras que la extracción de bordes solamente se encarga de extraer los bordes de la imagen. Antes de revisar las operaciones de vecindad se destacan algunos conceptos sobre el dominio en que se esté trabajando la imagen. Las operaciones sobre imágenes pueden enfocarse según el dominio de representación usado. Se distinguen: el dominio espacial y el dominio transformado. En el dominio espacial, las operaciones utilizan los píxeles en sus posiciones espaciales dentro del plano de la imagen y los niveles de intensidad asociados. Y en el dominio transformado (espectral) se realiza algún tipo de transformación de forma que se trabaje con componentes de frecuencia (dominio de la frecuencia) u otros. Entre las transformaciones más útiles destacan la transformada de Fourier, del coseno, entre otras [Pajares y De la Cruz, 2008].

En su trabajo Guili *et al*., (2011) utilizan el concepto de tratamiento de imágenes en un dominio diferente al espacial, que es el de la imagen original, mediante la transformación de dicha imagen a un dominio diferente al de la imagen original. Esto, debido a que su tratamiento puede ser más eficaz. Asimismo, se requiere de la transformada discreta de Fourier $F(u, v)$ de la imagen $f(x, y)$ y se define:

$$F(u, v) = \delta\{f(x, y)\} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \exp\left(-2\pi i \left(\frac{ux}{M} + \frac{vy}{N}\right)\right)$$

para $u = 0,1,2,3,\ldots,M-1, \quad v = 0,1,2,3,\ldots,N-1.$

Guru *et al*. (2010) utilizan momentos de textura de color como una de las características utilizadas para la clasificación de flores; aplican una transformada de Fourier local a todos los canales de la imagen. Experimentalmente, encontraron con que los espacios de colores (SVcosH, SVsinH, V), son mejores que los otros espacios de color, y estas características se denominan momentos de textura de color.

Una imagen se convierte en otra, o bien, se convierte en una matriz de valores modificados, que no resulten en otra imagen, a través de transformaciones píxel a píxel. La operación considera el píxel que se quiere transformar junto con sus vecinos. Ejemplo, una vecindad del píxel (x, y) está formada por los ocho píxeles que le rodean:

Cuadro 2.1 Ejemplo de vecindad entre pixeles.

$(x-1, y-1)$	$(x, y-1)$	$(x+1, y-1)$
$(x-1, y)$	(x, y)	$(x+1, y)$
$(x-1, y+1)$	$(x, y+1)$	$(x+1, y+1)$

Una operación de vecindad permite transformar el valor de un píxel p en la posición (x, y) considerando los valores de los píxeles vecinos. Ejemplo, digamos que generamos un nuevo valor de un píxel que es la suma ponderada de los vecinos de dicho píxel. Para definir los valores de ponderación comúnmente se define una máscara con valores constantes. Dicha máscara se utiliza como filtro. Por ejemplo, en la máscara:

Cuadro 2.2 Mascara de vecindad.

1	2	1
0	1.2	0
-1	-2	-1

el valor del nuevo píxel viene dado por la siguiente suma ponderada

$$q(x,y) = 1 \cdot p(x-1, y-1) + 2 \cdot p(x, y-1) + 1 \cdot p(x+1, y+1) +$$

$$0 \cdot\ p(x-1,y) + 1.2 \cdot p(x,y) +\ 0 \cdot p(x+1,y) - 1 \cdot p(x-1,y+1) - 2 \cdot p(x,y+1) - 1 \cdot p(x+1,y+1) \quad \text{.....(2.1)}$$

El resultado de aplicar la Ecuación (1.1) es una especie de repujado en relieve donde se marcan los bordes respecto del resto de la imagen; debido a que la máscara aplicada es una modificación de los operadores de Sobel usados para la extracción de bordes. Mediante el mismo procedimiento de vecindad por ejemplo, se puede conseguir un mayor contraste en la imagen con el uso de alguna máscara de ponderación apropiada.

2.1.2.3 Filtrado

Dentro de las operaciones de vecindad existen unas en particular importantes, las operaciones de filtrado. Éstas eliminan un determinado rango de frecuencias de las imágenes. Las operaciones de filtrado, basan su operatividad en la convolución de la imagen utilizando el núcleo de convolución, esto significa realizar el tratamiento digital de una matriz por otra que se llama *kernel*.

El filtrado es un tipo de operación que altera el valor de un píxel en función de los valores de los píxeles que le rodean, es por ello que a este tipo de procesamiento de la imagen también se le denomina procesamiento basado en la vecindad u operación de vecindad [Pajares y De la Cruz, 2008]. En el dominio de la frecuencia la operación de filtrado se reduce a una simple multiplicación de transformadas, mientras que en el dominio espacial el filtrado resulta ser una operación de convolución. Así, dadas dos funciones imagen f y h con transformadas de Fourier F y H, el *teorema de convolución* se define:

$$[f * h][(x,y)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(m,n) f(x-m, y-n)\, dm\, dn$$

De esta manera, $f * h$ es la convolución de dos funciones imagen f y h. En el contexto de la teoría de señales (la imagen es un tipo de señal) existen dos tipos de filtros [Alegre *et al*., 2003].

- Filtros Paso Bajo

Deja pasar las bajas frecuencias (en el caso de las imágenes son frecuencias espaciales) que se refieren a cambios lentos en la intensidad; por lo tanto, las altas frecuencias no pasan, i.e. los bordes, las disminuye. El resultado es un desenfoque de los bordes. Se asocia a zonas uniformes con baja frecuencia. Elimina el ruido y suaviza, relacionando un pixel con los de su alrededor.

- Filtros Paso Alto

Dejan pasar las altas frecuencias de la señal (bordes) asociadas a cambios bruscos de intensidad en pequeños intervalos espaciales, producen un realzado de los bordes de la imagen. Se eliminan las bajas frecuencias. El análisis y justificación de dichos filtros se debe estudiar desde el punto de vista de su comportamiento en el dominio de la frecuencia. En la presente investigación este tipo de filtros no se consideran.

Esta sección se enfoca en el preprocesamiento de la imagen. De la imagen original, se obtiene otra que resulta más adecuada para el objetivo de esta tesis. Así, la imagen mejorada facilitará las operaciones del procesado de la imagen; este mejoramiento corresponde a un suavizado de la imagen. El suavizado es un filtrado de tipo paso bajo, que consiste en disminuir o eliminar las componentes de alta frecuencia (borde y ruido). Si hay un borde en la imagen se atenúa y si es ruido, el efecto deseado es que se elimine. Por lo tanto, el suavizado reduce las variaciones de intensidad entre píxeles, y se usa principalmente para eliminar ruido, i.e., eliminar píxeles con nivel muy variable de intensidad comparado con los píxeles vecinos. El suavizado también puede realizarse mediante operaciones estadísticas como: el promedio del entorno de vecindad o filtrado de la media, filtrado de la mediana, moda, máximo y mínimo. Sin embargo, con excepción del primero, aunque se les denomine filtros, no tienen las características de los filtros descritos anteriormente.

Una imagen que debe ser mejorada se dice que tiene ruido. El ruido son aquellos píxeles en la imagen que tienen valores muy diferentes a sus vecinos en regiones homogéneas. El ruido procede de funcionamientos erróneos o imperfecciones de algunos elementos en el dispositivo de captura de la imagen, tales como celdas CCD (por sus siglas en inglés, Charge Coupled Device, i.e. dispositivo de carga acoplada, que sirven para transformar la luz captada por la cámara en electricidad) con mal funcionamiento, o impurezas, tales como: partículas de polvo, suciedad, entre otras cosas, en la óptica del sistema de captura. El ruido es información no deseada que contamina la imagen. Los tipos de ruido son Gaussiano y Uniforme. El ruido llamado “sal y pimienta” es un ruido de tipo impulso, donde el valor que toma el píxel no tiene relación con el valor ideal sino que toma valores muy altos o muy bajos. Toma el valor máximo (sal) o el mínimo (pimienta). En el ruido Gaussiano, todos los píxeles de la imagen cambian su valor, de acuerdo con una distribución Gaussiana. A continuación se describen los cuatro filtros mencionados anteriormente:

- Filtrado de la media

Esta técnica se aplica sobre el dominio espacial, o sea, opera directamente sobre los píxeles de la imagen. Funciona mediante la definición de algún tipo de promediado sobre el entorno de vecindad $n \times m$ al que se le llama ventana. Calcula la media de los píxeles en la ventana,

$$Ma = \frac{1}{nm} \sum_{(x,y) \in V} f(x,y)$$

donde nm es el número de píxeles de la ventana V de dimensión n por m. Se puede implementar mediante una máscara de convolución donde todos los coeficientes de la máscara son nm.

De esta forma, el filtro de la media remplaza cada píxel por el valor medio de sus vecinos. Dicho filtro es de tipo lineal. La media es un *kernel* para filtro de tipo paso bajo

(suavizamiento), utilizados para eliminar ruido o detalles pequeños de poco interés puesto que sólo afecta a zonas con muchos cambios. A continuación se ejemplifica el efecto de suavizar la imagen antes de ser procesada. En la Figura 2.5 se muestra una imagen original, a la cual se aplica ruido sal y pimienta, después un filtro de media con máscara de 3 x 3 y por último, el mismo filtro con máscara de 9 x 9, mostrando el efecto del número de píxeles sobre una máscara.

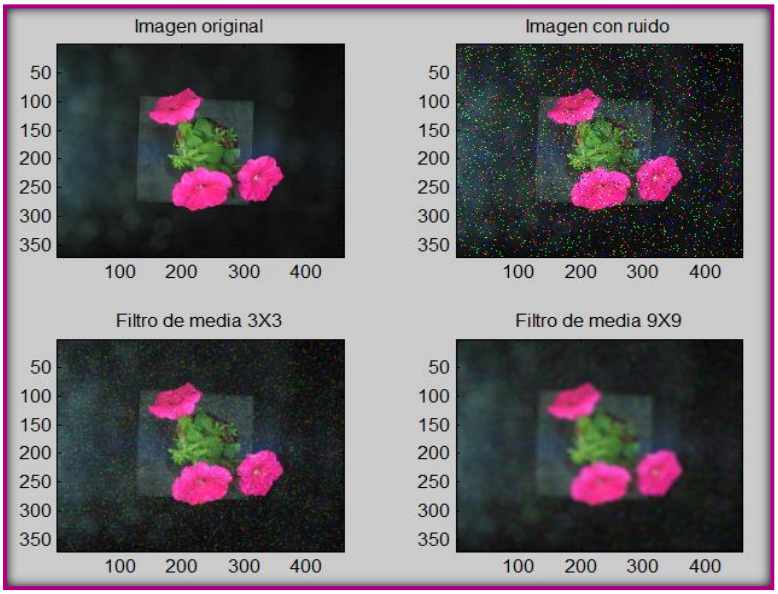


Figura 2.5 Ejemplo de eliminación de ruido, utilizando el filtro de la media, primero con una máscara de 3 x 3 píxeles y después con una máscara de 9 x 9 píxeles para visualizar su efecto.

El resultado de aplicar la media con una ventana de dimensión 3 x 3 da el mismo efecto que aplicar un filtro de paso bajo, a la imagen. Es importante definir el tamaño de la máscara. Mientras mayor sea el tamaño de la máscara se consigue una mayor reducción del ruido; sin embargo se produce una mayor difuminación de los bordes.

- Filtrado de la mediana

La mediana es un filtro no lineal. Se realiza ordenando las intensidades de los píxeles en el entorno de la vecindad y el valor medio de ese conjunto se aplica al píxel original. La mediana sirve para que los puntos con intensidades muy distintas se hagan muy parecidos a sus vecinos, esto elimina picos de intensidad que aparecen aislados en el

área de la vecindad del píxel a suavizar. Este filtro es bueno para conservar detalles de la imagen.

- Filtrado de máximos y mínimo

El filtro de máximo, ordena los valores de píxeles en el entorno de la vecindad y selecciona el mayor valor, a diferencia del mínimo que selecciona el menor. Los filtros de máximo tienden a eliminar los píxeles de ruido que presentan un valor de intensidad bajo y magnifica el ruido con valores altos, los mínimos lo hacen a la inversa.

- Filtrado de la moda

Este filtro elige dentro de la ventana de píxeles el valor más frecuente. Un problema con este tipo de filtro se presenta cuando todos los valores son diferentes, en tal caso, es posible utilizar un método aproximado de forma que todos los píxeles que no difieran entre sí, más de un determinado umbral se cuentan en conjunto. Ejemplo, si el conjunto de valores es {20, 21, 22, 23, 30, 31, 40, 80, 35}; entonces el valor más frecuente está en el intervalo [20,23], puede elegirse el valor 21.

2.1.3 Histograma de la imagen

El histograma de una imagen puede modificarse con el objeto de variar la distribución de los niveles de intensidad, manipulando así una imagen gris o de color. Generalmente se utiliza para:

- mejorar la calidad de la imagen. La técnica produce un realzado, se eliminan defectos, tales como: sombras o reflejos, además de aumentar el contraste.
- Fijar umbrales en procesos de binarización, para la extracción de regiones en las imágenes.

El histograma de la imagen es una función discreta que representa el número de píxeles en la imagen en función de los niveles de intensidad, g [Pajares *et al.,* 2004]. La probabilidad de ocurrencia de un determinado nivel g se expresa:

$$p(g) = \frac{N(g)}{M}$$

donde M es el número de píxeles de la imagen y $N(g)$ es el número de píxeles en el nivel de intensidad g.

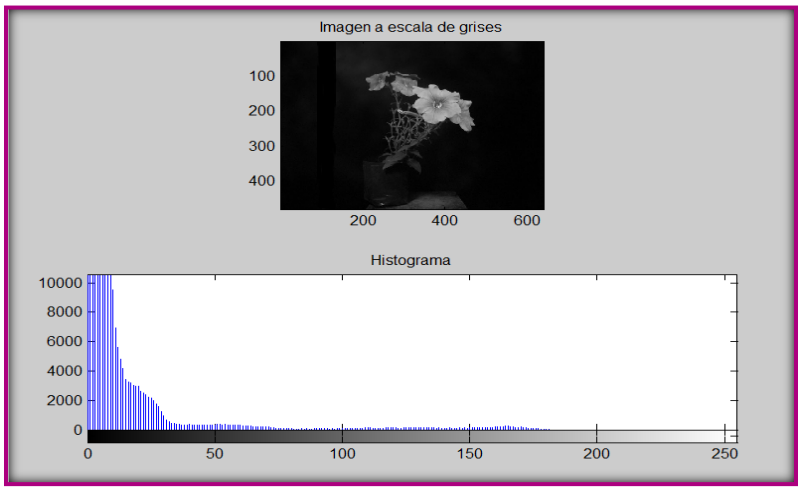


Figura 2.6 La imagen original se convirtió a escala de gris para obtener el histograma de la imagen.

Si el histograma tiene una distribución de los niveles de intensidad concentrada en una zona específica, presentará un contraste muy bajo. Por otro lado, si la distribución de los niveles es dispersa, la imagen tiene un alto contraste. El histograma con niveles concentrados de gris en la parte baja del rango corresponde a una imagen obscura; mientras que el histograma con los valores concentrados de gris en su parte alta corresponde a una imagen brillante.

2.1.4 Fundamentos del color

El color es fundamental sobre todo, para los productos agrícolas; ya que, se complementa bien con otros factores como los físicos y químicos, aspectos que son

buenos indicadores de la calidad del producto. Debido a la estructura del ojo humano, cualquier color puede especificarse en función de la combinación de tres colores primarios: rojo (R), verde (G) y azul (B). Las características utilizadas generalmente para distinguir un color de otro son: brillo, matiz y saturación.

El brillo proporciona la noción cromática de intensidad. El matiz es un atributo asociado con la longitud de onda dominante en la mezcla de longitudes de onda de la luz; que representa el color percibido por el observador. La saturación es la pureza relativa, i.e., la cantidad de luz blanca mezclada con un matiz; el grado de saturación es inversamente proporcional a la cantidad de luz blanca añadida. Las cantidades de rojo, verde y azul requeridas para forma un color particular se denominan X, Y y Z respectivamente y se definen:

$$x = \frac{X}{X+Y+Z}; \qquad y = \frac{Y}{X+Y+Z}; \qquad z = \frac{Z}{X+Y+Z}$$

donde $x + y + z = 1$.

Normalmente como cada color primario se codifica con un byte la intensidad de cada uno de los componentes se mide en la escala de 0 a 255. Por lo tanto, el rojo se obtiene con (255, 0, 0), el verde con (0, 255, 0) y el azul con (0, 0, 255), obteniendo, en cada caso un color resultante monocromático.

2.1.4.1 Espacio de color RGB

El modelo de color RGB (del inglés *Red, Green, Blue*) se representa mediante el sistema cartesiano cuyo subespacio es el cubo unitario (i.e., con lados iguales a uno). Se asigna un valor a cada uno de los colores primarios para formar un color dado; así, por ejemplo, el valor 0 significa que un color primario no interviene en la mezcla, y a medida que su valor aumenta, aporta más intensidad a la mezcla. Un píxel es en realidad un conjunto de tres puntos: rojo, verde y azul. Cada uno de ellos, brilla con una determinada intensidad. En el sistema cartesiano, cada color es un punto determinado por un vector que va del

origen a dicho punto. El origen de coordenadas representa el color negro. Cada eje está asociado a un color (rojo, verde, azul). La luminosidad del color disminuye mientras más cerca esté del origen.

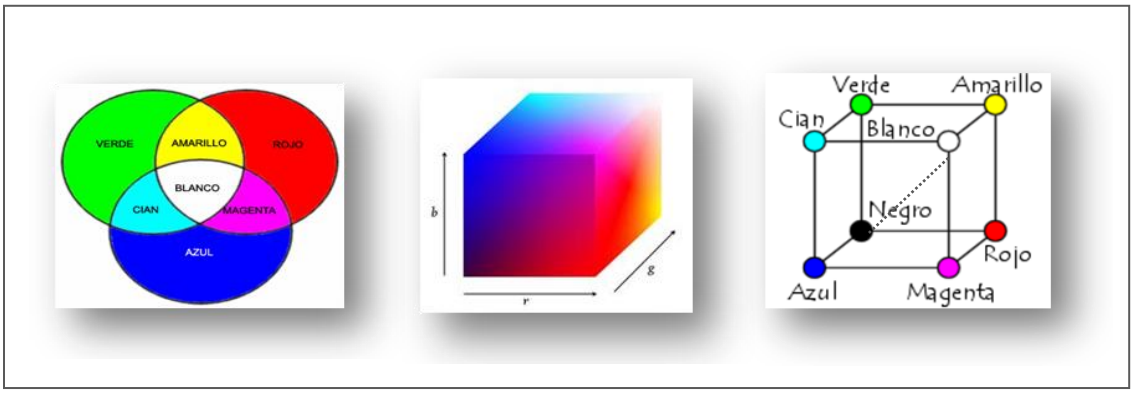


Figura 2.7 Modelización de los canales del espacio de color RGB (Fuente: imágenes Google).

Una desventaja con este modelo de color puede ser su amplitud, debido a que está constituido por 8 bits por canal, 256 niveles de gris; Una segunda desventaja está determinada por la imposibilidad de los monitores de mostrar amplios espacios de color. De forma que los colores que físicamente sea incapaz de reproducir nunca conseguirá mostrarlos.

2.1.4.2 Espacio de color CIE Lab

Este espacio de color es utilizado para describir todos los colores que puede percibir el ojo humano, permite cuantificar las diferencias de colores en una imagen, Figura 2.8. Fue desarrollado por la *Commission Internationale d'Eclairage* (Comisión Internacional de Iluminación). El modelo está construido matemáticamente para ser utilizado como un sistema de referencia para establecer coherencias entre otros sistemas de color como lo son el RGB y el CMYK.

Con el espacio de color CIE Lab es más rápido hacer correcciones eficientes de color. El hecho de que la luminosidad es completamente degradada en los canales *a* y *b* hace que sea mucho más sensible a errores. Es posible referenciar una cantidad superior de

colores en total, no solo colores que no pueden ser descritos con RGB o CMYK sino también colores que no aparecen en absoluto en el mundo real.

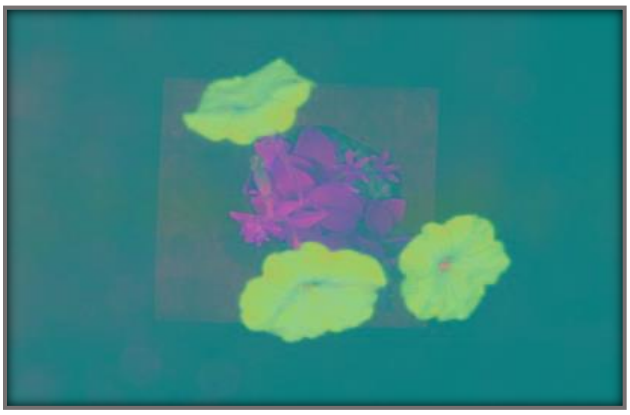


Figura 2.8 Ejemplo una imagen al aplicarle el modelo CIE Lab, se muestra la utilidad del modelo, al resaltar o diferenciar los distintos colores en la imagen.

El espacio de color CIE Lab se deriva de los valores tri-estímulos CIE XYZ, y se compone de una luminosidad *'L*' (donde *L* = 0 negro y *L* = 100 indica blanca); la capa de cromaticidad *'a'* indica que el color cae a lo largo del eje rojo-verde (valores positivos indican rojo y valores negativos indican verde), y la capa de cromaticidad *'b'* indica que el color cae a lo largo del eje azul-amarillo (valores positivos indican amarillo y valores negativos indican azul). Su enfoque consiste en elegir una muestra pequeña de la región de cada color y para calcular el color promedio de cada región de la muestra en el espacio *'ab'*, que van a utilizar estos marcadores de colores para clasificar cada píxel. Las formulas correspondientes a *L*, *a* y *b* se expresan por:

$$L = 116 \times \left(\frac{Y}{Y_0}\right)^{\frac{1}{3}} - 16$$

$$a = 500 \times \left[\left(\frac{Y}{Y_0}\right)^{\frac{1}{3}} - \left(\frac{Z}{Z_0}\right)^{\frac{1}{3}}\right]$$

$$b = 200 \times \left[\left(\frac{Y}{Y_0}\right)^{\frac{1}{3}} - \left(\frac{Z}{Z_0}\right)^{\frac{1}{3}}\right]$$

Entre las investigaciones que han utilizado este espacio se encuentran Guli *et al.* (2011) quienes eligieron *b* de CIE1976 para distinguir las características de color con el objetivo de identificar la deficiencia de nutrientes de plantas de tomates, y Mendoza *et al.* (2006) cuantificaron el color estándar de frutas y hortalizas basándose en los espacios de color sRGB, el VHS y CIE Lab. Donde esté sugirió como el mejor espacio de color para la cuantificación de los alimentos con superficies curvas.

2.1.4.3 Espacio de color LCH

El espacio de color LCH (por sus siglas en inglés *Luminance*, *Chrominance* and *Hue*) es una medida más apropiada del color que se puede obtener en el cálculo del ángulo de tono (h^o) y Chroma *C*, el cual es un índice análogo a la saturación del color o la intensidad (Zhang *et al.,* 1997). Estos pueden calcularse a partir de *a y b* de CIE Lab o, en los nuevos instrumentos Minolta se puede leer directamente. Chroma *C* representa la hipotenusa de un triángulo rectángulo creado por puntos de acceso (0, 0), (*a*, *b*), y (*a*, 0) ver Figura 2.9.

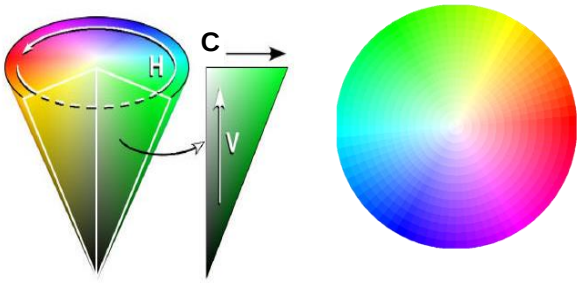


Figura 2.9 Disposición espacial de valor, hue y croma.

El ángulo *hue* se puede definir como el ángulo entre la hipotenusa y 0° en el *a* eje (azulado-verde/rojo-purpura); *h*° se calcula a partir del arco tangente de *b/a*. Sin embargo, el arco tangente, asume que los valores positivos en los primer y tercer cuadrantes y negativos en los segundo y cuarto cuadrantes. Para una interpretación útil, *h*° debería permanecer positivo entre 0° y 360°. Este espacio es una representación más intuitiva del color CIE Lab. Con *L* fijo, el color puede representarse con las coordenadas

rectangulares *a* y *b*. Pero, además también se puede hacer con las coordenadas polares *C* y *h*, definidas como:

$$C = (a^2 + b^2)^{1/2}$$

$$\theta = \left(\frac{\arctan\left(\frac{a}{b}\right)}{6.2832} \right) \times 360;$$

$$si\ a > 0\ y\ b \geq 0 \therefore h = \theta;$$

$$si\ a < 0\ y\ b \geq 0 \therefore h = 180 + \theta;$$

$$si\ a < 0\ y\ b < 0 \therefore h = 180 + \theta;$$

$$si\ a > 0\ y\ b < 0 \therefore h = 360 + \theta;$$

Los valores de *C* y *h* tienen correspondencia con croma *(chroma)* y tono o matiz *(hue)* respectivamente. El valor de *h* es el ángulo del tono, se expresa en grados, de 0º (incluido) a 360º (excluido). Si se calcula en radianes, se convierte a grados multiplicándolo por 180/*π*.

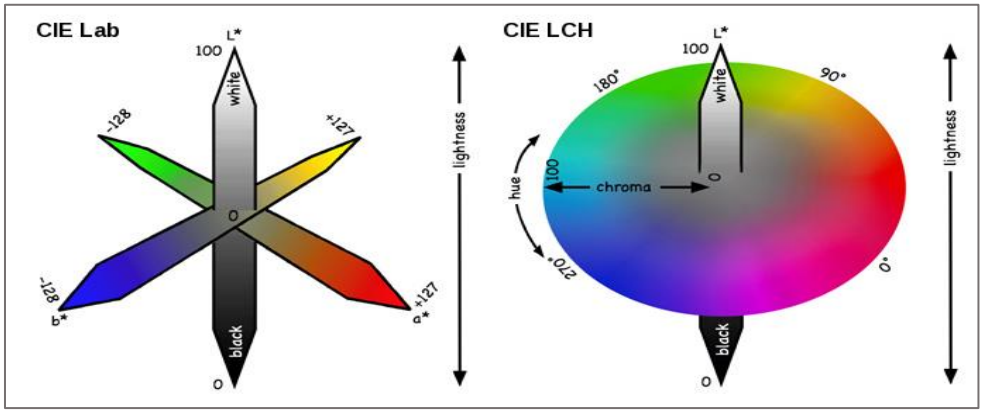


Figura 2.10 Representación gráfica de los modelos de color CIE Lab y LCH (Fuente: Imágenes Google).

2.1.5 Operaciones morfológicas

Las operaciones morfológicas se basan en la forma y geometría, sirven para simplificar las imágenes y preservar las formas principales de los objetos. Su uso en visión artificial es frecuente, para la modificación, evaluación y/o conteo de regiones. El cálculo de regiones en una imagen se facilita. Las operaciones morfológicas se aplican principalmente a imágenes binarias. Las operaciones de la morfología matemática son:

suavizar bordes de una región, separar regiones que en la segmentación se presentan unidas, o unir regiones que fueron separadas durante la segmentación.

La morfología matemática se basa en los conjuntos de puntos, la geometría integral y la topología. Se trata de modelar a la imagen a través de conjuntos de puntos de cualquier dimensión (como, el espacio Euclídeo $N - dimensional$). En el procesamiento de imágenes se utiliza el espacio Euclídeo $2D$ o E^2 (dominio para la descripción de formas planas), conjuntos con elementos pares de números enteros [Pajares *et al.*, 2004].

En el contexto de imágenes binarias: un punto es un par de enteros que son las coordenadas de la imagen digital. Una imagen binaria es un conjunto de puntos $2D$. Los puntos de los objetos de la imagen son un conjunto X, que son píxeles con valores 1s. Los puntos de conjunto complemento X^c son el fondo con valores 0s. Ejemplo,

$$\begin{bmatrix} \bullet 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 2.11 Representación de una imagen binaria.

En la Figura 2.11, el origen (destacado con un punto) tiene coordenadas (0,0) y los puntos pertenecientes al objeto tienen valor uno. Un punto x de la imagen X es un vector con respecto al origen (0,0).

2.1.5.1 Binarización

La binarización permite simplificar una imagen. Por ejemplo, si el objeto de la imagen tiene una superficie homogénea, i.e., sin cambios de color y textura, además, dentro de un fondo que produce un buen contraste, entonces la imagen se puede transformar en binaria. Que consiste en clasificar cada elemento de la imagen como un bit 1 o 0, mediante un umbral *de* intensidad. Un píxel con un nivel inferior al umbral, se hace igual a cero (negro) y uno con un nivel superior o igual al umbral, se hace igual a uno (blanco).

La imagen binaria resultante contendrá menos información, pero simplificará la aplicación de algoritmos, por lo que se podrán deducir muchas características geométricas de los objetos de la imagen, como el área, el perímetro, entre otros [Iñigo y Angulo, 1986].

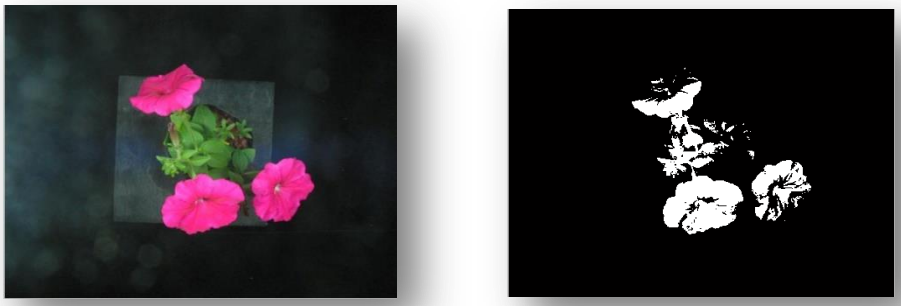


Figura 2.12 Ejemplo de una imagen original (izquierda) y una binarizada (derecha).

2.1.5.2 Transformación morfológica

La trasformación morfológica denotada por ϕ es la relación de la imagen (conjunto X) y un pequeño conjunto de puntos B, llamado elemento estructural [Pajares *et al*., 2004]. *B* se expresa con respecto a un origen local O (punto representativo o elemento director).

$$\begin{matrix} 1 & 1 & 1 \\ 1 & \bullet 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Figura 2.13 Representación de un elemento estructural.

En la Figura 2.13, el punto representativo O se marca por un punto, y el elemento estructural es el conjunto de puntos: $B = \{(-1,-1),(-1,0),(-1,1),(0,-1),(0,0),(0,1),(1,-1),(1,0),(1,1)\}$. En algunos casos no se considera al punto representativo como elemento del conjunto B y el punto donde se encuentra tiene valor cero. Ejemplos importantes de transformaciones morfológicas son la dilatación y la erosión.

La dilatación $X \oplus B$ es el conjunto de puntos de todas las posibles sumas vectoriales de pares de elementos, uno de cada conjunto X y B.

$$X \oplus B = \{\mathbf{d} \in E^2 : \mathbf{d} = \mathbf{x} + \mathbf{b}\ para\ cada\ \mathbf{x} \in X\ y\ \mathbf{b} \in B\}$$

A continuación se presenta un ejemplo:

X = {(0,1), (1,2), (2,0), (2,1), (3,0), (3,1)} y B = {(0,0), (0,1)}

$X \oplus B$ = {(0,1), (1,2), (2,0), (2,1), (3,0), (3,1), (0,2), (1,3), (2,2), (3,2)}

$$\begin{bmatrix} \bullet 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \oplus \begin{bmatrix} \bullet 1 & 1 \end{bmatrix} = \begin{bmatrix} \bullet 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 2.14 Ejemplo de una transformación morfológica de dilatación.

Una dilatación con el elemento estructural de la Figura 2.14 actúa de igual forma sobre todas las direcciones, i.e. es isotrópica y rellena o expande el objeto un píxel.

Figura 2.15 Ejemplo de una imagen original (izquierda) e imagen dilatada (derecha).

La transformación morfológica de la erosión $\otimes$ combina dos conjuntos utilizando la sustracción de vectores.

$$X \otimes B = \{\mathbf{d} \in E^2 : \mathbf{d} + \mathbf{b} \in X\ para\ cada\ \mathbf{b} \in B\}$$

En la erosión cada punto $\boldsymbol{d}$ del conjunto X es comprobado; el resultado de la erosión está dado por los puntos $\boldsymbol{d}$ para los cuales todos los posibles $\boldsymbol{d} + b$ están en X. Por ejemplo si tenemos el conjunto de puntos X y el elemento estructural B dados por:

$X = \{(0,2),(1,2),(2,0),(2,1),(2,2),(2,3),(3,2),(4,2)\}$ y $B = \{(0,0),(0,1)\}$

$X \otimes B = \{(2,0),(2,1),(2,2)\}$

$$\begin{bmatrix} \bullet 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \otimes [\bullet 1 \quad 1] = \begin{bmatrix} \bullet 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 2.16 Ejemplo de la transformación morfológica de la erosión.

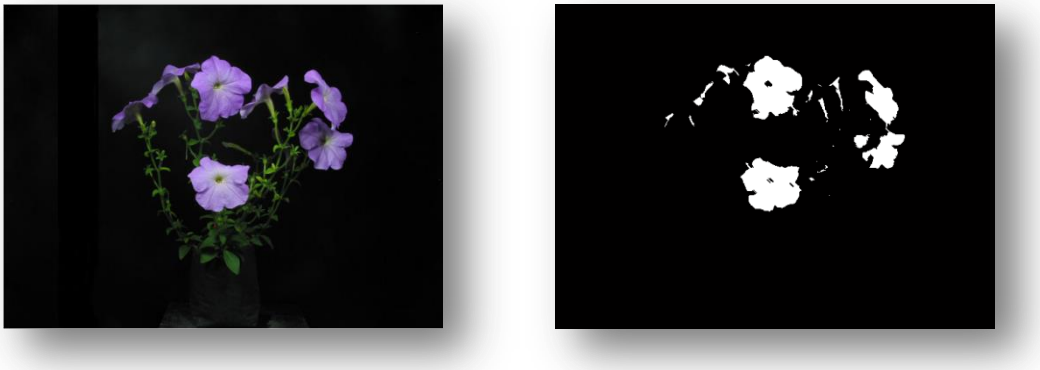


Figura 2.17 Imagen original (izquierda) e imagen erosionada (derecha).

2.1.6 Segmentación

La segmentación ocurre para obtener una clasificación sobre las regiones de color de la imagen, dicha clasificación es necesaria para el reconocimiento que será la tarea de una red neuronal artificial en una etapa posterior al procesamiento de las imágenes. La segmentación se define como “el proceso por el cual se extrae de la imagen cierta información subyacente para su posterior uso” [Pajares y De la Cruz, 2008]. Con la segmentación se obtienen las regiones que componen la imagen, asignándole a cada una de ellas una etiqueta. La segmentación está basada en dos principios esenciales: discontinuidad y similitud. Por lo cual la segmentación puede ser orientada a bordes

(discontinuidad) y orientada a regiones (similitud). Una *región* es un área de la imagen en la que sus píxeles poseen propiedades similares (de intensidad, color, etc.), mientras que un *borde* es una línea que separa dos regiones, por tanto de diferentes propiedades (discontinuidad).

2.1.6.1 Segmentación orientada a bordes

En términos formales, el borde es una propiedad local de un píxel y su vecindad inmediata. “Para detectar un borde, se recurre a conceptos de variación de la intensidad en la vecindad del píxel, que se cuantifica en forma de un vector gradiente con magnitud y dirección” [Pajares *et al*., 2004].

La segmentación orientada a bordes se basa en las diferencias entre niveles de grises: detectan la discontinuidad en los niveles de gris utilizando métodos de detección de puntos, líneas, bordes, entre otros, utilizando filtros como Sobel, Prewitt, Laplaciano [Alegre *et al*., 2003].

2.1.6.2 Segmentación orientada a regiones

“Una región es un conjunto de píxeles en la cual hay un camino entre cualquier par de píxeles y todos los píxeles de este camino pertenecen al conjunto” [Pajares *et al*., 2004]. La segmentación basada en las regiones se utiliza cuando existe similitud en los niveles de gris. Como presente trabajo se enfoca en una segmentación basada en los colores, la segmentación está orientada a regiones.

Algunas técnicas para la detección de regiones son: binarización basada en el uso de umbrales; crecimiento de regiones mediante la adición de píxeles; división de regiones y similitud de textura, color, o nivel de gris.

2.1.7 Descriptores

Para realizar la segmentación de objetos, es importante distinguir el descriptor adecuado para una determinada tarea. En general, cualquier tipo de descriptor, debe ser independiente del tamaño, orientación y posición de los objetos, pero que aporte la suficiente información sobre ellos como para poder reconocerlos. Esta fase es muy importante y va a condicionar posteriores reconocimientos e interpretaciones. Existen dos tipos de descriptores:

1. Descriptores de bordes: es la identificación de los bordes mediante el ajuste de rectas, curvas, funciones polinomiales, códigos encadenados, entre otros.
2. Descriptores de regiones: obtienen propiedades tales como color, textura, superficie, nivel medio de intensidad, entre otros.

En este caso importan los descriptores de regiones debido a que interesan las propiedades del color. En el análisis de imágenes el color es un poderoso descriptor para el ojo humano debido a que puede distinguir una amplia gama de colores comparado con los niveles de gris. Así, el color sirve para simplificación y detección de objetos en una escena. Una técnica puede ser extraer de la imagen aquellas regiones en las que predomine una determinada componente de color. Primero se elige un determinado predicado y se busca en toda la imagen los píxeles que cumplen dicho predicado. Esos píxeles se marcan en blanco y el resto en negro; así, obtenemos una imagen binaria.

En la extracción de características se pueden formar grupos significativos de colores de la imagen. Para una planta en floración los grupos son por ejemplo: el área de fondo, hojas y flores [Timmermans y Huizebosch, 1995].

2.1.7.1 Descriptores de regiones "métricas"

Dentro de los descriptores de regiones se encuentran las métricas. Estas son generalizaciones de las medidas de distancia, comúnmente la distancia Euclidiana, las cuales son:

1. Área (A): Número de píxeles contenidos dentro de su frontera.
2. Perímetro (P): Longitud de la frontera de la región. Es el número de píxeles que forman la frontera. Al usar las coordenadas de los píxeles de la frontera, se calcula:

$$P = \sum_{i} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

3. Compatibilidad (C): Es la relación entre el cuadrado del perímetro y el área de la región.

$$C = \frac{P^2}{A}$$

4. Centro de gravedad ($\bar{x}, \bar{y}$): es un punto representativo de la región.
5. Redondez: cociente entre el área y el eje mayor al cuadrado.
6. Elongación: es la relación entre la longitud y la anchura de una región.
7. Excentricidad: cociente entre la longitud de los ejes mayor y menor.
8. Compacidad: cociente entre la raíz cuadrada del área y el eje mayor.

Aitkenhead *et al.* (2003) reportan la utilización de técnicas de análisis de imágenes para discriminar mala yerba en los cultivos. Estos autores pudieron discriminar entre dos tipos de planta, un método fue el uso de características morfológicas simples para medir la forma de la hoja (P^2/área) con eficiencias entre 52 y 74%.

2.2 REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales representan un área de la inteligencia artificial que imita en forma simplificada el funcionamiento del cerebro biológico. Las ANN son modelos estudiados principalmente porque las computadoras a pesar de su velocidad de cálculo siguen un procesamiento secuencial, lo cual limita la realización de tareas que requieren un procesamiento en paralelo; tales como el reconocimiento de patrones o la generalización de futuros eventos a partir de acciones pasadas. Por lo tanto, existen tareas que solo el cerebro humano puede resolver, el cual almacena información en forma de patrones, con distintos niveles de complejidad, por ejemplo, el reconocimiento de caras desde varios ángulos. El mecanismo de las redes neuronales biológicas que se intenta emular en una computadora, puede resumirse en tres pasos:

- Las neuronas captan información que les llega en forma de impulsos procedentes de otras neuronas o receptores.
- La integran en un código de activación propio de la célula y procesan una respuesta.
- La codifican y transmiten en forma de frecuencia de impulsos a las neuronas subsiguientes.

Con el diseño de las ANN para la tarea de clasificación de las flores se pretende, identificar objetos de una imagen de flores a través de los colores de los objetos en dicha imagen. Siendo estos colores patrones a reconocer por las ANN.

2.2.1 Inspiración biológica de una red neuronal artificial

La función del sistema neuronal biológico es transmitir información mediante: células receptoras que colectan información, sistema nervioso que recibe almacena y envía información y órganos efectores que reciben e interpretan información como impulsos. Las funciones y elementos esenciales de una neurona biológica son:

1. Neurona: es una unidad estructural y funcional del sistema de comunicación neuronal, cuya función es recoger, procesar y emitir señales eléctricas; se compone de tres partes principales: las dendritas, el cuerpo de la célula o soma y el axón. Una neurona emite señales químicas para transmitir información a las distintas neuronas, a través de prolongaciones, formando redes que almacenan información.
2. Dendritas: son pequeñas ramificaciones de la neurona que reciben señales eléctricas y las propagan al cuerpo de la neurona.
3. Axón: fibra larga que tiene una cubierta llamada vaina de mielina que funciona como capa aislante, esta ramificación de salida de la neurona propaga una serie de impulsos electro-químicos desde el cuerpo de la célula hacia otras neuronas.
4. Sinapsis: zona de contacto entre las terminaciones del axón de una neurona con otras neuronas o células efectoras, pero sin llegar a fusionarse entre ellas. Es el medio por el cual se transfiere información entre neuronas; la información viaja a lo largo de los axones en breves impulsos eléctricos generados por la membrana de la neurona, ver Figura 2.18. Es un espacio líquido donde existen determinadas concentraciones de elementos ionizados, normalmente iones de sodio (Na^+) y potasio (k^+). Los iones suscitan un espacio donde se activan o inhiben el paso de impulsos eléctricos. La sinapsis puede potenciar o inhibir la señal procedente de los axones [Nason, 1993].

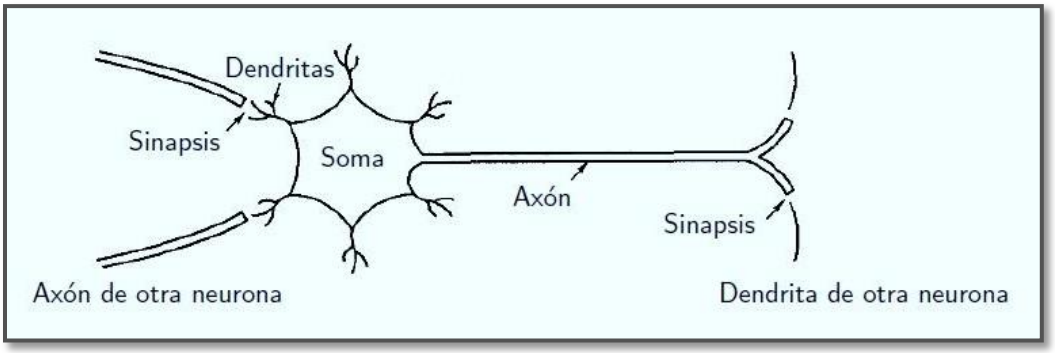


Figura 2.18 Descripción de una neurona biológica típica.

La señal eléctrica de la neurona emisora se transforma en la sinapsis en un proceso químico, creando neurotransmisores, que se transforman en señales eléctricas en la neurona receptora.

5. Redes neuronales: son el conjunto de neuronas, conectadas en forma masiva que conforman el sistema nervioso y el cerebro; el cerebro humano puede contener 10^{11} neuronas y 10^{15} interconexiones, Figura 2.19. Funcionan como una enorme malla que propaga señales electroquímicas de unas células a otras, y además modifican las concentraciones iónicas de las sinapsis, estas concentraciones hace que las neuronas actúen según su saturación, i.e., una neurona se activa si la señal recibida supera un cierto umbral, de lo contrario permanece inhibida.

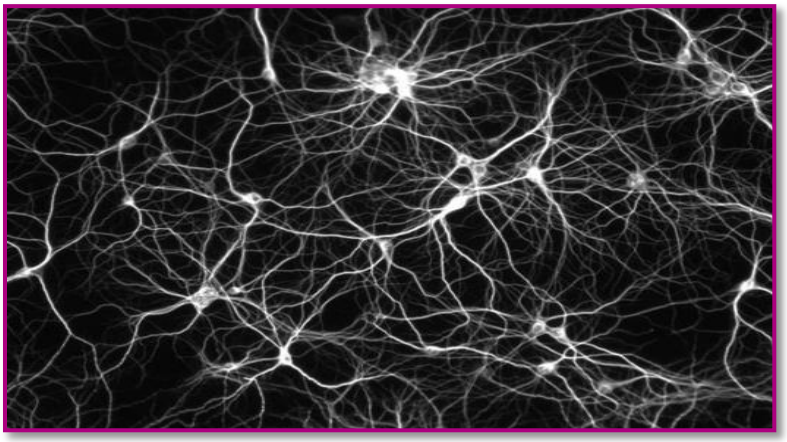


Figura 2.19 Redes neuronales biológicas (Fuente: Imágenes Google).

2.2.2 Concepto de redes neuronales artificiales

Una ANN es una técnica que procesa información inspirada en el funcionamiento del cerebro biológico. Una ANN almacena conocimiento que ha adquirido a través de un proceso de adaptación y lo utiliza con la finalidad de que el sistema adquiera la capacidad de generalización, por lo que es una técnica útil para el reconocimiento de patrones dado que de alguna forma repite o simula el mecanismo de memorización y asociación de hechos, tareas propias del cerebro humano. Se puede describir a las ANN como modelos computacionales, que imitan de manera simplificada a las redes neuronales biológicas, cuya función es extraer conocimiento para resolver problemas

como: visión, reconocimiento de patrones o control moto-sensorial. Una ANN es un procesador distribuido en paralelo y masivamente interconectado que almacena conocimiento experimental y presentan las siguientes particularidades [Sánchez y Alanís, 2006]: el conocimiento es adquirido experimentalmente y los pesos (ganancias) de interconexión (sinapsis) varían constantemente.

2.2.3 Componentes de una red neuronal artificial

Un modelo de red neuronal (Figura 2.20) consiste de los siguientes componentes:

1. Neuronas: son las unidades básicas de procesamiento de las ANN, con un nivel de activación dado. En general, una neurona recibe varias señales de entrada y envía una única señal de salida.
2. Conexiones: las señales se transmiten a través de las conexiones de la red, están juegan el papel de la sinapsis biológica y permiten la comunicación entre neuronas. Cada conexión está dirigida y tiene un peso numérico w_{ji} asociado, que determina la fuerza y el signo de la conexión. Las conexiones indican a la neurona información sobre el estado de activación de las neuronas con las que se conecta, cuando la "sinapsis" tiene un peso positivo, la señal de entrada activa a la neurona, si el peso es negativo la señal inhibe a la neurona.
3. Función de activación: ésta permite a la neurona transformar su nivel de activación actual en una señal de salida, a partir de las señales que recibe; el resultado depende de las entradas recibidas y de valores sinápticos. Para calcular el estado de activación, se obtiene la entrada total a la célula, E_i que es igual a la suma de todas las entradas ponderadas por ciertos valores. A este estado de activación se le aplica la función de activación.
4. Regla de aprendizaje: es la forma en que la red neuronal adquiere conocimiento la cual consiste en el ajuste o modificación iterativa de los pesos de las conexiones de forma que se manipulen de acuerdo a la regla de aprendizaje.

El proceso es el siguiente, una conexión con un peso $w_{j,i}$, de la neurona j a la neurona i propaga la activación x_j de j a i. Cada neurona i primero calcula una suma ponderada de sus entradas $x_1, x_2, x_3, \dots, x_j$ provenientes de un vector $\bar{X}$. Cada señal se multiplica por su peso asociado $w_{1,1}, w_{2,1}, w_{3,1}, \dots, w_{j,i}$ las cuales provienen de un vector $\bar{W}$. La sumatoria que corresponde al cuerpo de la neurona, suma todas las entradas previamente ponderadas por sus pesos de forma algebraica, produciendo la salida $E_1 = x_1 w_{1,1}, x_2 w_{2,1}, x_3 w_{3,1}, \dots, x_j w_{j,i}$, que se representa mediante:

$$E_i = \sum_{j=0}^{n} w_{j,i} x_j$$

En notación vectorial la ecuación anterior se representa como: $E = X^T W$. Luego se aplica una función de activación f a esta suma para producir la respuesta:

$$x_i = f(E_i) = f(\sum_{j=0}^{n} w_{j,i} x_j)$$

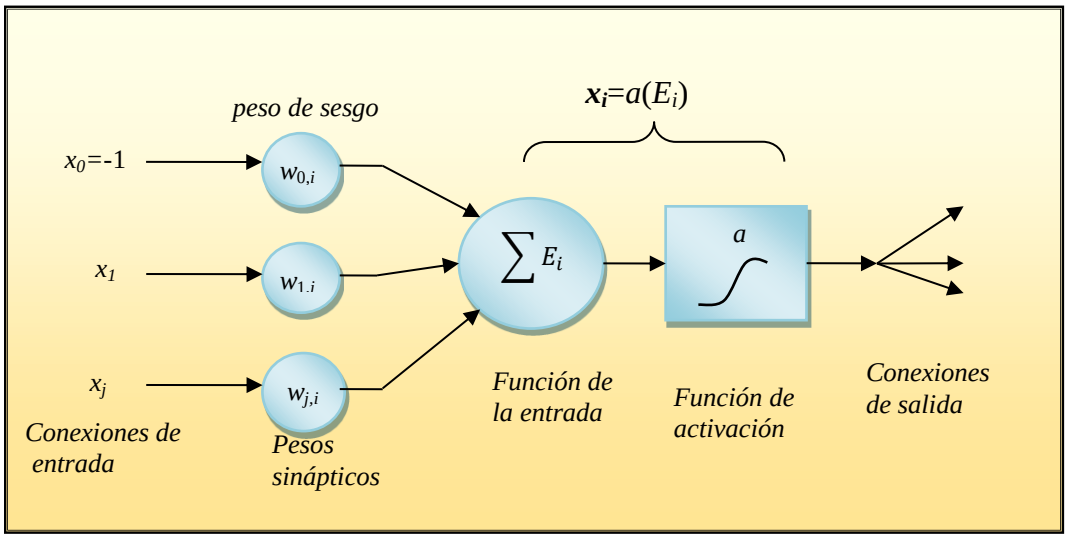


Figura 2.20 Modelo matemático de una neurona. La activación de la salida de la neurona es $x_i = a(\sum_{j=0}^{n} w_{j,i} x_j)$ donde x_j es la activación de la salida de la neurona j y $w_{j,i}$ es el peso de la conexión de la neurona j la neurona i.

En la figura, se incluye un peso de sesgo $w_{0,i}$ asociado a la entrada $x_0 = -1$[Russell y Norvig, 1996].

2.2.3.1 Función de activación

La función de activación f sirve para excitar o inhibir a la neurona dependiendo del propósito deseado. La neurona estará activa, i.e., cercana a 1, si las entradas son correctas y cercana a 0 cuando las entradas son incorrectas. La activación debe utilizar una función de tipo no lineal. Dependiendo del tipo de función f, habrá distintos modelos de red, ejemplos [Isasi y Galván, 2004]:

- Lineal: $x_i = kE_i$ con k constante.
- Umbral: $x_i = 1\ si\ E_i \geq \theta,\ x_i = 0\ si\ E_i < \theta$; siendo θ el umbral constante.
- Cualquier función: $x_i = f\ (E_i)$; siendo a una función cualquiera.

Algunas funciones de transferencia importantes se describen a continuación.

- Función purelin

La función de activación lineal se designa por *purelin* en Matlab. La salida de la función de activación lineal es igual a su entrada; presenta una pendiente unitaria, sin embargo esta puede presentar problemas con los datos, porque si alguno no es congruente puede provocar que la respuesta tienda al infinito y al no tener una acotación generaría un error en los resultados finales.

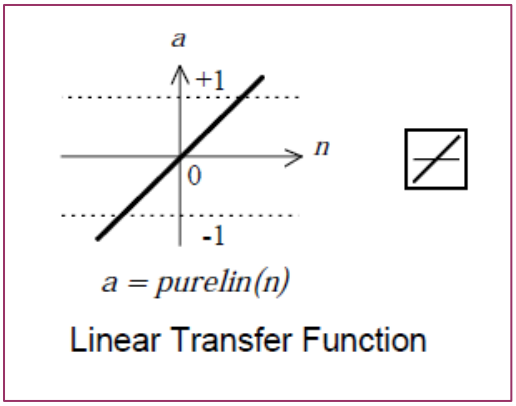


Figura 2.21 Gráfica y símbolo en Matlab de la función de activación purelin.

- Función tansig

Una red multicapa puede usar la función de activación *tansig*. Esta función matemáticamente es equivalente a *tanh(n)*. Las neuronas de salida sigmoides son a menudo usadas para reconocimiento de patrones, mientras que las neuronas de salida lineales son usadas para problemas de ajuste de función. Esta función delimita los valores de la salida entre [-1, 1], debido a que se acota los resultados, si una variable tiende al infinito dichas limitaciones frenarían ese comportamiento. La función tangente hiperbólica se define:

$$tansig(n) = \frac{2}{(1+e^{-2n})-1} \qquad \text{..... (2.2)}$$

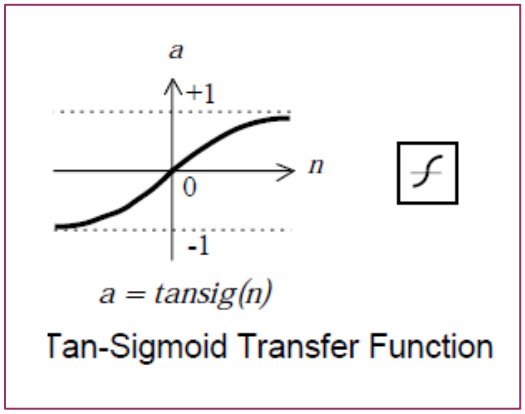


Figura 2.22 Gráfica y símbolo en Matlab de la función de activación tansig.

- Función logsig

La función de activación sigmoidea logarítmica se designa por *logsig* en Matlab. Esta función toma los valores de entrada, los cuales pueden variar entre más y menos infinito y restringe la salida a valores entre 0 y 1. La función logsig es usada para redes multicapa, como la red de propagación inversa, debido a que dicha función es diferenciable. La función sigmoidea se define como:

$$logsig(n) = \frac{1}{1+e^{-n}} \qquad \text{..... (2.3)}$$

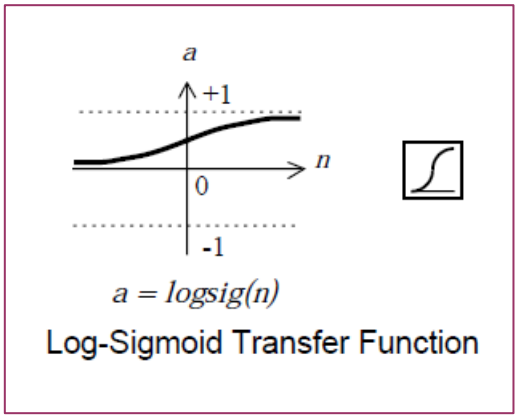


Figura 2.23 Gráfica y símbolo en Matlab de la función de activación logsig.

- Función radbas

La función de activación Gaussiana de base radial se denota por *radbas* en Matlab.La función *radbas* tiene un máximo de 1 cuando su entrada es 0. A medida que la distancia entre el vector de pesos y el vector de entrada disminuye, la salida aumenta. Por lo tanto, una neurona de base radial actúa como un detector que produce 1 cada vez que la entrada es idéntica a su vector de pesos. El umbral b permite que la sensibilidad de la neurona de base radial sea ajustada. Por ejemplo, si una neurona tiene un umbral de 0.1 ésta envía una salida de 0.5, para cualquier vector de entrada a una distancia de 8,326 (0,8326 / b) a partir del vector de pesos W. La función *radbas* Gaussiana se define como:

$$radbas(n) = e^{-n^2} \quad \text{..... (2.4)}$$

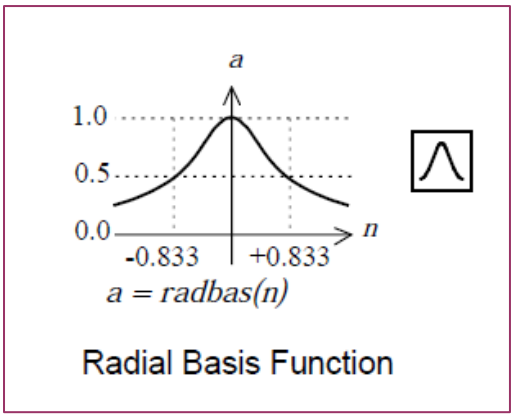


Figura 2.24 Gráfica y símbolo en Matlab de la función de activación radbas.

2.2.4 Arquitecturas de una red neuronal artificial

La arquitectura o topología de la red se determina por el número de neuronas, la disposición de las mismas, además de las conexiones existentes entre ellas [Corchado *et al.*, 2000]. Existen tres principales topologías de red: monocapa, multicapa y recurrentes. En las redes monocapa y multicapa ninguna salida de la neurona es una entrada de otra neurona del mismo nivel o de niveles anteriores o bien, de la misma neurona; por lo tanto, son redes acíclicas que no tienen uniones de retroalimentación. La red representa una función de sus entradas actuales, su estado interno es únicamente el de sus propios pesos [Russell y Norvig, 1996]. Por todo lo anterior, son llamadas redes neuronales de propagación hacia adelante. Las redes recurrentes también llamadas redes de retroalimentación o bien redes de retropropagación son cíclicas. Las salidas de las neuronas pueden ser entradas de neuronas del mismo nivel, de niveles anteriores o, de la misma neurona.

2.2.4.1 Redes neuronales multicapa con propagación hacia adelante.

Esta arquitectura posee varias capas intermedias formadas de neuronas denominadas neuronas ocultas. La señal se distribuye de una capa a la siguiente. La Figura 2.25

muestra una red multicapa que se referencia como 2-3-2 (o sea, 2 neuronas de entrada, 3 ocultas y 2 de salida). Además, como cada neurona de la red se conecta con todas las neuronas de la capa subsiguiente, se dice que la red está completamente conectada.

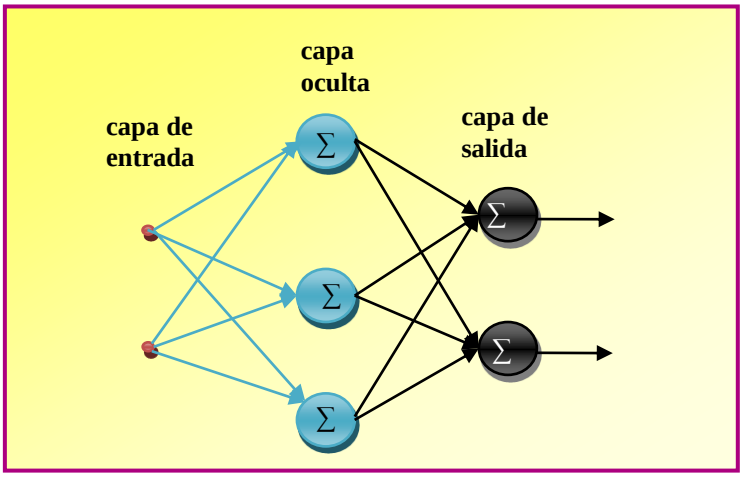


Figura 2.25 ANN multicapa con propagación hacia adelante.

2.2.5 Entrenamiento

El entrenamiento se describe como un proceso donde la configuración de una red, descrita por todos los pesos de sus conexiones, se modifica en cada iteración o etapa de entrenamiento y determina la capacidad de adaptación o aprendizaje del modelo.

Aprendizaje en el contexto de redes neuronales significa el proceso mediante el cual los parámetros de una red neuronal artificial se adaptan como consecuencia de un proceso de estimulación llevado a cabo por el entorno en el que la red opera [Corchado *et al.*, 2000]. La red tiene una configuración con pesos iniciales (que en general se designan de forma aleatoria) y con la definición de los parámetros de aprendizaje. En esta etapa empieza el entrenamiento que es el ajuste de los pesos sinápticos según una regla de aprendizaje. Las ANN utilizan un aprendizaje basado en ejemplos. De este modo, el conjunto de datos de entrenamiento debe ser: significativo (debe haber suficientes ejemplos) y representativo (el conjunto de entrenamiento debe poseer componentes diversificados). El proceso de aprendizaje se clasifica en tres paradigmas básicos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por grados o con fortalecimiento.

2.2.5.1 Aprendizaje supervisado

El aprendizaje supervisado utiliza pares de vectores de entrenamiento. Cada par se compone de valores de entrada a la red y valores de salida esperados para dichas entradas. La ANN recibe las entradas para producir las salidas cuyos resultados se comparan con los valores esperados. La diferencia entre ambos valores se propaga a las capas ocultas con el objetivo de ajustar los pesos de la red.

- Conjunto de entrenamiento

Son los datos utilizados para el proceso de aprendizaje. Los mismos pares de vectores de entrenamiento son utilizados varias veces durante el entrenamiento. El tiempo de entrenamiento queda determinado por la capacidad de la red de generar las respuestas deseadas. Cuando se entrena una ANN para memorizar datos, puede perder capacidad de generalizar, para solo imitar. Esto puede ser provocado por un sobreentrenamiento, para evitarlo la red es evaluada mediante un conjunto independiente de datos nombrado conjunto de validación.

- Conjunto de validación

Subconjunto del conjunto de datos inicial no utilizado en el entrenamiento y representativo de todo el conjunto; con este se evalúa el comportamiento de la red, estudiando en etapas distintas del entrenamiento los resultados de la red sobre el conjunto de validación. Si la red produce buenos resultados con este conjunto entonces la red ya está entrenada [Corchado *et al.*, 2000].

Sabemos que la forma de generar la acción de aprendizaje se logra principalmente con la modificación de los pesos de las conexiones, lo importante es saber cómo se modifica. Un ejemplo entra a la red y se procesa para generar una salida, la salida se compara con la salida deseada (esta es información incluida en el conjunto de aprendizaje). La diferencia entre ambas influirá en cómo se modifican los pesos. Si la diferencia es mucha

se modifican mucho los pesos, de lo contrario la modificación es mínima. Un ejemplo es cuando se quiere un clasificador de caras de diferentes personas, los ejemplos contendrán datos del individuo, la imagen de su cara, e información de la solución, digamos una etiqueta para cada persona que las diferencie. Dicha información servirá para modificar los pesos en el aprendizaje supervisado [Isasi y Galván, 2004].

2.2.5.2 Algoritmos de aprendizaje

Existen diversos algoritmos de aprendizaje. Haykin considera cinco tipos básicos de reglas de aprendizaje [Haykin, 1998]: aprendizaje basado en la corrección de error, aprendizaje basado en memoria, aprendizaje Hebbiano, aprendizaje competitivo y aprendizaje de Boltzmann. El tipo de aprendizaje depende de la forma en que cambie la configuración de la red. El tipo de aprendizaje más popular es el algoritmo básico de retropropagación del error para el entrenamiento de un perceptrón multicapa [Vesali *et al.*, 2009]. La clasificación de los tejidos de telas fue conducida por el entrenamiento en dos etapas de una ANN con el algoritmo de retropropagación [Chung *et al*., 2010]. Existen otros métodos más avanzados de aprendizaje como [Palma *et al.*, 2008]: métodos basados en mínimos cuadrados lineales, métodos de segundo orden y paso de aprendizaje adaptativo.

El primer método se basa en la minimización del error medio cuadrático (ECM). El método de segundo orden hace uso de las derivadas segundas para incrementar la velocidad de aprendizaje. Un ejemplo es el algoritmo de entrenamiento de levenberg–marquardt para entrenamiento de tamaño moderado de las redes neuronales feedforward, el cual se encuentra implementado en MATLAB [Shah *et al.*, 2009]. El paso de aprendizaje adaptativo es un método heurístico para la adaptación dinámica de la tasa de aprendizaje.

2.2.6 Primeros modelos de redes neuronales artificiales

Antes de describir los modelos de redes neuronales MLP y PNN tratados en esta tesis, se exponen brevemente los primeros modelos de redes neuronales desarrollados.

2.2.6.1 Modelo de McCulloch-Pitts

El primer modelo de una neurona, fue desarrollado por Warren MacCulloch y Walter Pitts en 1943 en el artículo "*A Logical Calculus of the Ideas Immanent in Nervous Activity".* Consistía en la modelización del funcionamiento simplificado de las neuronas del cerebro. La neurona era un dispositivo con dos estados posibles: apagado (0) y encendido (1).

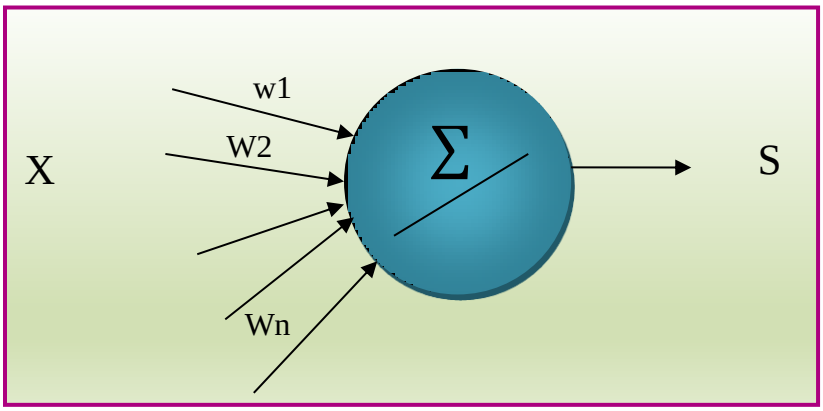


Figura 2.26 Modelo de la célula de McCulloch- Pitts.

Este modelo recibe n valores binarios, que proceden del conjunto de entradas a la red o, de las salidas de otras células, produciendo una sola salida binaria. Cada célula contiene $n + 1$ valores reales de los cuales n son los valores de los pesos de las conexiones y el otro es un valor umbral θ, que puede ser distinto para cada célula. Al recibir una entrada la célula se activará produciendo un valor 1, si la suma de las entradas multiplicadas por los pesos, supera al umbral θ, de lo contrario produce un 0.

$$S(t+1) = \begin{cases} 1, & si \sum x_i \times w_i(t) > \Theta \\ 0, & en\ otro\ caso \end{cases}$$

Según Isasi y Galván, (2004) el modelo de McCulloch-Pitts tiene la capacidad de computación universal, i.e., cualquier estructura que pueda ser programada en una computadora, puede ser modelada mediante una red de células de McCulloch-Pitts.

2.2.6.2 El Perceptrón lineal

Este modelo realiza tareas de clasificación en forma automática. Teniendo un conjunto de ejemplos de clases diferentes (llamados patrones o ejemplos de entrenamiento), el sistema determina las ecuaciones de las superficies que definen la frontera de dichas clases. La información del sistema procede de los patrones de entrenamiento y puede construir las superficies discriminantes, actuando además como un discriminador para nuevos ejemplos de entrada.

La arquitectura de este modelo contempla una estructura monocapa, con un conjunto de neuronas de entrada y una o varias neuronas de salida. Cada neurona de entrada se conecta con todas las neuronas de salida y estas conexiones son las que determinan las superficies de discriminación del sistema.

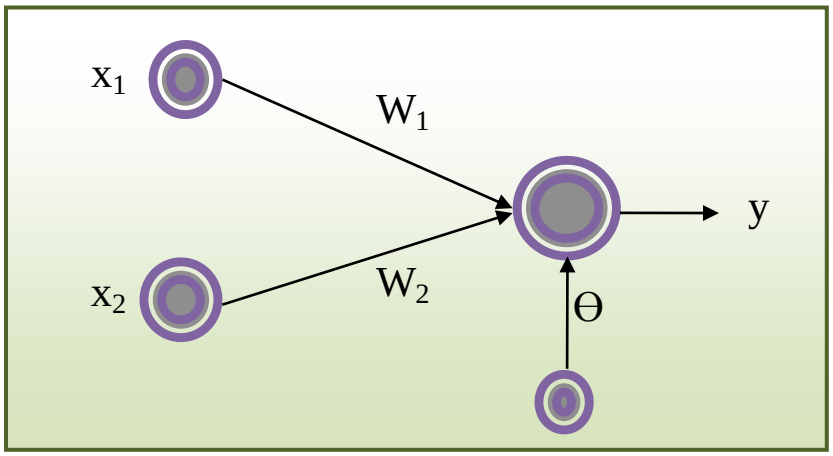


Figura 2.27 Arquitectura del perceptrón con dos entradas y una salida.

En la Figura 2.27, las entradas son x_1 y x_2, y la salida y. Los pesos son w_1 y w_2 y, el umbral θ. El umbral se utiliza para comparar la salida; todas las células tienen un umbral

específico. Para producir la salida, se calcula la suma ponderada por los pesos de todas las entradas [Isasi y Galván, 2004]:

$$y' = \sum_{i=1}^{n} w_i x_i$$

Por último, se aplica una función de salida (función escalón) al nivel de activación de la neurona. Esta función depende del umbral:

$$y = F(y', \theta)$$

$$F(s, \theta) = \begin{cases} 1, & si\ s > \Theta \\ -1, & en\ otro\ caso \end{cases}$$

Pasando el umbral θ al otro lado de la ecuación la salida es:

$$y = F(\sum_{i=1}^{n} w_i x_i + \theta)$$

donde F ya no depende de ningún parámetro:

$$F(s) = \begin{cases} 1, & si\ s > 0 \\ -1, & en\ otro\ caso \end{cases}$$

Esto es como introducir un nuevo peso θ asociado a una entrada ficticia con valor -1.

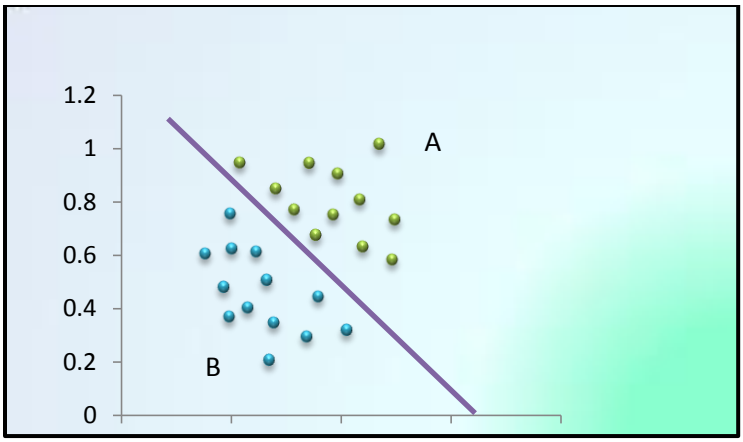


Figura 2.28 Ejemplo de clasificación lineal para dos clases.

En el perceptrón los patrones de entrenamiento pueden representarse como puntos en un espacio bidimensional ver Figura 2.28, los puntos pertenecen a una de dos clases A y B. Si las clases están separadas puede demostrarse que existe una recta discriminante de cada clase. Los valores que determinan las dos ecuaciones discriminantes $y(A)$ y $y(B)$ se obtienen a través de un proceso iterativo que, en cada etapa modifica los valores de los pesos si el patrón de entrada se clasificó incorrectamente. Los pesos sinápticos se inicializan aleatoriamente y deben ser valores pequeños, comúnmente a 1 y el peso w_0 correspondiente al umbral a 0.5 dado que afectan directamente el funcionamiento de la ANN. La regla de aprendizaje modifica los pesos por medio de incrementos o decrementos dependiendo de la clase a la cual pertenece el ejemplo en turno.

2.2.7 El Perceptrón multicapa

Dado que la mayoría de los problemas son linealmente no separables, modelos de ANN tales como el perceptrón y ADALINE (Acrónimo de ADAptive LInear NEuron) fueron descartados. Así, fue desarrollado el perceptrón multicapa o red multicapa con conexiones hacia adelante que resuelve problemas no separables. El modelo surge de la idea de combinar varios perceptrones simples. Posteriormente, surge la *regla delta generalizada* como una manera de retropropagar los errores medidos en la salida de la red hacia las neuronas ocultas. El perceptrón multicapa es en la actualidad una de las arquitecturas más utilizadas en una gran diversidad de problemas, debido a su capacidad como aproximador universal, esto quiere decir “cualquier función continua

sobre un compacto de R^n puede aproximarse con un perceptrón multicapa con al menos una capa oculta de neuronas" [Isasi y Galván, 2004].

Rasekhi *et al.*, (2010) utilizaron perceptrones multicapa (MLP) para la clasificación de maíz y mala hierba, consideraron tres tipos de maleza y definieron diez características diferenciales de la forma de las hojas para cada clase de planta. Mediante procesamiento de imágenes se extrajeron los diez rasgos y se usaron como entradas a la red. La red demostró una precisión del 98.5 % para discriminar maíz de la mala hierba. Para medir la madurez y la calidad de la sandía Shah *et al.*, (2009) capturaron la textura de la piel de la sandía mediante imágenes digitales con el objetivo de adquirir información para entrenar un perceptrón de tres capas, obteniendo así la madurez del fruto con un porcentaje de precisión de 86.5%. El color de la sandía es segmentado al mapa de color RGB que posteriormente se convirtió al espacio de color YCbCr. Después, el índice de madurez se basó en la cantidad de píxeles en cada una de las regiones segmentadas. Con estos datos se entrena la ANN que realiza la función de reconocimiento de patrones. Vesali *et al.*, (2009) descubrieron un método para obtener un índice que pronostica la densidad de humedad de las manzanas utilizando un perceptrón multicapa, cuyas entradas fueron los parámetros de textura y el análisis del índice de densidad.

2.2.7.1 Arquitectura del perceptrón multicapa

La arquitectura del perceptrón multicapa consiste una o varias capas de neuronas ocultas. Existen tres tipos de capas: de entrada, ocultas y de salida (ver Figura 2.29). La capa de entrada solo recibe señales que se propagan a la siguiente capa, la última capa actúa como salida de la red.

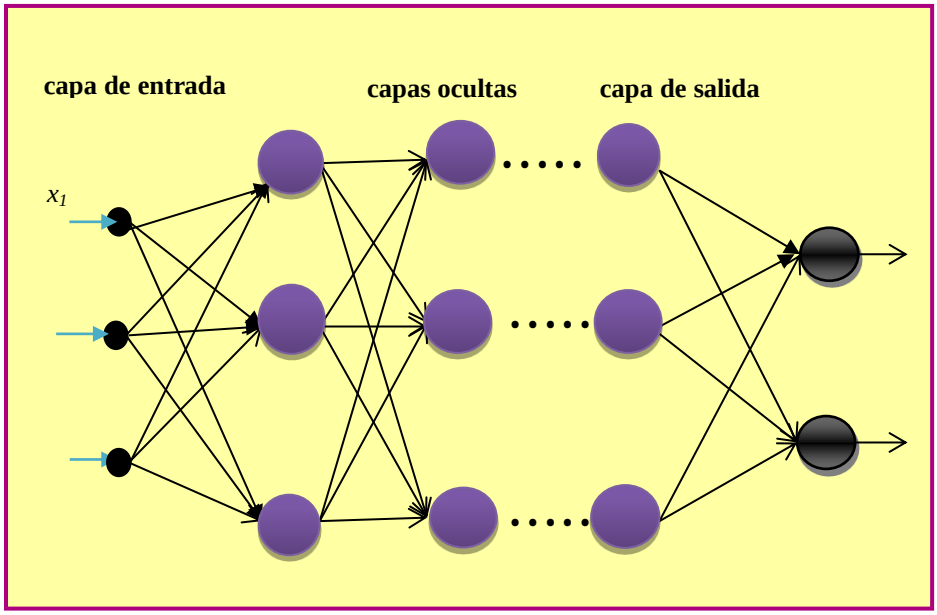


Figura 2.29 Arquitectura del perceptrón multicapa.

Las capas ocultas realizan un procesamiento no lineal, además las conexiones están dirigidas hacia adelante es por eso que se les llama redes de alimentación hacia adelante o redes *feedforward*. Dichas conexiones llevan un peso asociado que es un número real. Cada neurona se relaciona con un umbral que se trata como una conexión más con entrada igual a uno.

2.2.7.2 Propagación de las entradas

Cada neurona recibe información de sus entradas y produce una respuesta o activación que se propaga hacia las neuronas de la capa que le sigue [Isasi y Galván, 2004]. El cálculo de las activaciones se expresa como sigue.

Sea un perceptrón multicapa con C capas y n_c neuronas en la capa c, para $c = 1,2,\dots,C$. Además sea $W^c = (w_{ij}^c)$ la matriz de pesos asociada a las conexiones de la capa c a la capa $c+1$ para $c = 1,2,\dots,C-1$, donde w_{ij}^c representa el peso de la conexión de la neurona i de la capa c a la neurona j de la capa $c+1$; y sea $U^c = (u_i^c)$ el vector de umbrales de las neuronas de la capa c para $c = 2,\dots,C$. Se expresa a_i^c a la activación de

la neurona de la capa c. La activación de la capa de entrada se expresa $a_i^1 = x_i$ para $i = 1,2,\dots,n_1$ donde $X = (x_1, x_2, \dots, x_{n_1})$ representa el vector o patrón de entrada a la red. La activación de las neuronas ocultas procesa la información aplicando la función de activación f a la suma de los productos de las activaciones que recibe por sus correspondientes pesos.

$a_i^c = f(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + u_i^c)$ para $i = 1,2,\dots,n_c$ y $c = 2,3,\dots,C-1$

donde a_j^{c-1} son las activaciones de las neuronas de la capa $c-1$. Asimismo, la activación de las capas de salida es como sigue:

$y_i = a_i^C = f(\sum_{j=1}^{n_{C-1}} w_{ji}^{C-1} a_j^{C-1} + u_i^C)$ para $i = 1,2,\dots,n_c$

donde $Y = (y_1, y_2, \dots, y_{n_C})$ es el vector salida de la red.

En el caso del perceptrón multicapa las funciones de activación f más utilizadas son:

- Función Tangente hiperbólica (Ecuación 2.2) en el intervalo [-1,1].
- Función Sigmoidal (Ecuación 2.3) cuya imagen se encuentra en el intervalo [0,1].

El perceptrón multicapa define a través de sus neuronas y conexiones, una función continua no lineal del espacio $\mathbf{R}^{n_1}$ –espacio de los patrones de entrada– al espacio $\mathbf{R}^{n_C}$ –espacio de los patrones de salida–. Por lo tanto, se puede denotar por:

$$Y = F(X, W)$$

donde X es el vector de entrada de la red, Y es el vector de salida, W el conjunto de parámetros (pesos y umbrales) y F es una función continua no lineal.

2.2.7.3 Algoritmo de entrenamiento de retropropagación

La regla o algoritmo de aprendizaje del perceptrón multicapa se basa en aprendizaje supervisado; i.e., se modifican los parámetros para que la salida de la red sea lo más próxima posible a la salida deseada. Se requiere una función de error E que evalué la diferencia entre las salidas de la red y las salidas deseadas.

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n)$$

donde N es el número de patrones y $e(n)$ es el error cometido de la red para el patrón n, el cual es:

$$e(n) = \frac{1}{2}\sum_{i=1}^{n_C}(s_i(n) - y_i(n))^2 \qquad \text{..... (2.5)}$$

siendo $Y(n) = (y_1(n), \dots, y_{n_C}(n))$ y $S(n) = (s_1(n), \dots, S_{n_C}(n))$ los vectores de salida de la red y salidas deseadas para el patrón n, respectivamente.

La meta es encontrar el punto mínimo de la función error E, donde el error es próximo a cero y la salida de la red es cercana a la salida deseada. El problema de minimización es no lineal (debido a la presencia de las funciones de activación no lineales) por lo que se usan técnicas de optimización no lineales para adaptar los parámetros siguiendo una dirección de búsqueda, que para el caso es la dirección negativa del gradiente de la función E –método del descenso del gradiente–. Aplicando el método del descenso del gradiente estocástico, el cual consiste en una sucesiva minimización de los errores para cada patrón,$e(n)$, en lugar de minimizar el error total E. Cada parámetro w de la red se modifica para cada patrón de entrada n de acuerdo con la siguiente ley de aprendizaje:

$$w(n) = w(n-1) - \alpha\frac{\partial e(n)}{\partial w}$$

donde $e(n)$ es el error para el patrón n dado por la Ecuación (2.5) y α es la tasa de aprendizaje, parámetro que influye en la magnitud del desplazamiento en la superficie del error. Resultando de esto el algoritmo de retropropagación o regla delta generalizada, el término retropropagación se debe a como se implementa el método del gradiente pues el error cometido en la salida de la red se propaga hacia atrás, convirtiéndolo en un error para cada una de las neuronas ocultas.

2.2.8 Redes neuronales probabilísticas

Las redes neuronales probabilísticas resuelven problemas de clasificación. En el proceso de clasificación no existe un ajuste de pesos, asimismo, los patrones de salida son determinados mediante la comparación y el cálculo de distancias, ver arquitectura de la red en la Figura 2.30. La PNN es un típico clasificador no lineal, que usa el criterio Bayesiano de riesgo mínimo. La principal diferencia con el algoritmo de retropropagación radica en que toma menos tiempo de entrenamiento. Además siempre se puede obtener una solución óptima con el criterio Bayesiano, cuando los datos de entrenamiento son suficientes, sin tomar en cuenta la complejidad del problema de clasificación [Yu *et al.*, 2010].

Sea un vector de patrones x de dimensión m que pertenece a una de dos categorías k_1 y k_2. Dejemos $F_1(x)$ y $F_2(x)$ como las funciones de densidad de probabilidades para la clasificación de categorías k_1 y k_2 respectivamente. De la regla de Bayes de decisión discriminante, x pertenece a k_1 si:

$$\frac{F_1(x)}{F_2(x)} > \frac{L_1 P_2}{L_2 P_1} \qquad \text{..... (2.6)}$$

Por el contrario, x pertenece a k_2 si,

$$\frac{F_1(x)}{F_2(x)} < \frac{L_1 P_2}{L_2 P_1} \qquad \text{..... (2.7)}$$

donde L_1 es la función de pérdida o función de costos asociada con la clasificación errónea de los vectores como pertenecientes a la categoría k_1 cuando en realidad pertenecen a la categoría k_2, L_2 es la función de pérdida asociada con el vector de clasificación errónea como pertenecientes a la categoría k_2, cuando pertenece a la categoría k_1, P_1 es la probabilidad a priori de la ocurrencia de la categoría k_1, y P_2 es la probabilidad a priori de la ocurrencia de la categoría k_2. En muchas situaciones, las funciones de pérdida y las probabilidades a priori pueden ser consideradas iguales. Por lo tanto, la clave para usar las reglas de decisión dada por las Ecuaciones (2.6) y (2.7) es el de estimar las funciones de densidad de probabilidades de los patrones de entrenamiento [Parthiban y Subramanian, 2009].

2.2.8.1 Arquitectura de una red neuronal probabilística

Una PNN recibe una entrada del exterior en la primera capa donde se calculan las distancias desde el vector de entrada a los vectores de entrada objetivos; estas distancias se registran como un nuevo vector que indica que tan cerca se encuentra el vector de entrada del vector objetivo. La segunda capa suma las contribuciones para cada clase de entrada generando un vector de salida de probabilidades. La arquitectura de una PNN (Figura 2.30) se compone de muchas neuronas interconectadas organizadas en capas sucesivas. La capa de entrada no realiza ningún cálculo y simplemente distribuye la entrada a las neuronas en la segunda capa llamada capa patrón. Al recibir un patrón x de la capa de entrada, las x_n neuronas de la capa patrón calcula su salida mediante

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

donde d denota la dimensión del vector de patrones x, σ es el parámetro de suavizado, y x_{ij} es el vector de la neurona.

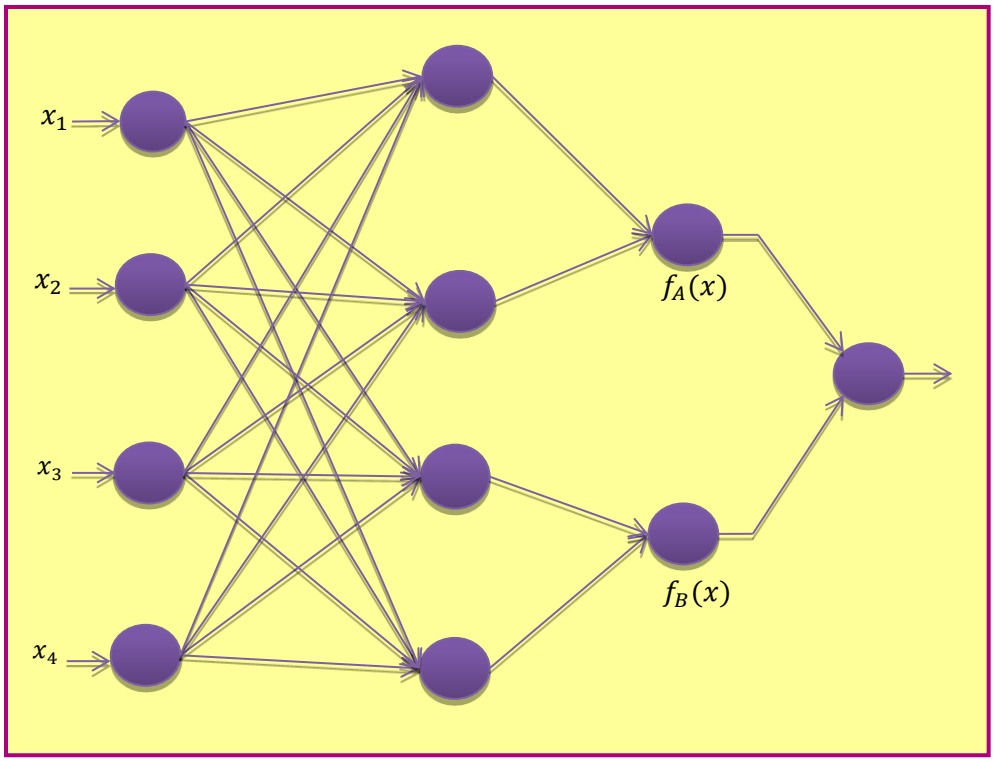


Figura 2.30 Arquitectura de una red neuronal probabilística.

Para el problema de dos clases, si las neuronas de entrada se supone que son x_1 a x_n, los patrones de entrada serán de las clases A ($Z_{A_1}\ a\ Z_{A_j}$) y B ($Z_{B_1}\ a\ Z_{B_j}$). La sumatoria de las neuronas será f_A y f_B y la neurona de salida y, a continuación, el algoritmo de entrenamiento de la red neuronal probabilística es [Parthiban y Subramanian, 2009]:

i. Para cada entrenamiento de patrones de entrada, $x(p)$, $p = 1, \ldots . P.$ Ejecutar los pasos i a iii.
ii. Crear la neurona patrón Z_p: vector de pesos para la neurona de Z_p: $W_p = x(p)$ (la neurona Z_p puede ser una neurona Z_A o bien, una neurona Z_B).
iii. Conecte la neurona patrón a la neurona sumatoria. Si $x(p)$ pertenece a la clase A, y conectar neurona patrón Z_p a la neurona sumatoria S_A. Si no, conectar la unidad patrón Z_p a la neurona sumatoria S_B.

El algoritmo de aplicación para la clasificación se da como sigue:

i. Inicializar los pesos del algoritmo de entrenamiento.

ii. Para patrón cada de entrada a ser clasificado, hacer los pasos iii-v.

iii. Neuronas patrones:

Calcular la entrada neta, $Z_{inj} = x \times W_j = x^T W_j$

Calcular la salida $Z = \exp[[Z_{inj} - 1]/\sigma^2]$

iv. Neuronas sumatorias:

Los pesos usados por las neuronas sumatorias para la clase B son

$$V_B = -P_B C_B m_A / P_A C_A m_B$$

v. Neurona de salida:

Suma las señales de f_A y f_B. El vector de entrada se clasifica de la clase A, si la entrada total a la neurona de decisión es positiva.

En la capa de salida se puede utilizar una función de activación del tipo competitiva, donde se selecciona la máxima de estas probabilidades respondiendo con un uno para esa clase y cero para las demás.

Aplicaciones utilizando los modelos de PNN, han proliferado en los últimos años. Un ejemplo es el trabajo desarrollado por Yu *et al.* (2010) sobre las interfaces cerebro máquina que son utilizadas para clasificar la gravedad de la parálisis corporal, ellos utilizan PNN para controlar prótesis de brazos, cursores de cómputo y músculos paralizados, comprendiendo la relación entre las actividades neurales y los movimientos de las extremidades. Se discretiza el valor de la presión en varios niveles como atributos de decisión de la PNN. En otra investigación se emplearon PNN con técnicas de procesamiento de imágenes y datos para implementar un reconocimiento automático de la hoja de propósito general para la clasificación de la planta. Se extraen doce características de la hoja y se ortogonalizan en cinco variables de componentes principales que consisten en el vector de entrada de la PNN. La PNN se entrenó con 1,800 hojas para clasificar 32 tipos de plantas con una precisión superior a 90% [Gang *et al.*, 2007].

3. MATERIALES Y MÉTODOS

La clasificación de imágenes digitales de dos variedades de flores de petunia (*Petunia* spp) se realizó mediante atributos extraíbles de las imágenes y la aplicación de un algoritmo de clasificación como las ANN, con el propósito de evaluar el desempeño de dos modelos de ANN. Los pasos de a seguir se resumen en la Figura 3.1.

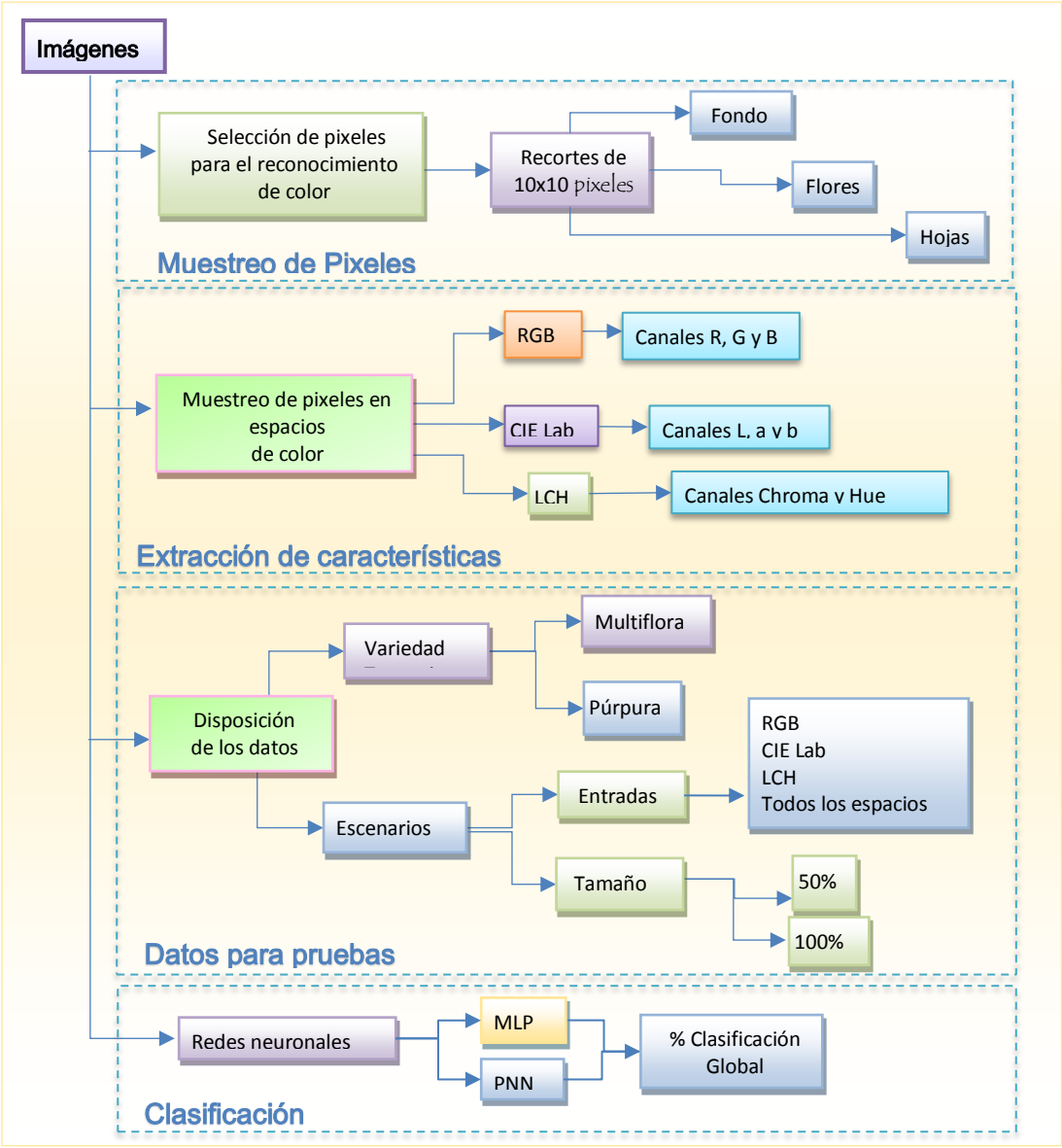


Figura 3.1 Diagrama de bloques de la metodología propuesta.

3.1 EL MATERIAL VEGETAL Y EL SITIO EXPERIMENTAL

Se cultivaron 14 plantas de petunia de los cultivares multiflora y púrpura en macetas de 14 pulgadas en un invernadero del campo experimental del Campus Montecillo, del Colegio de Postgraduados localizado en Montecillo, Estado de México, a 98° 54’ 11’’ longitud oeste y 19° 27’ 38’’ de latitud norte. El sustrato fue una mezcla 1:1 de peat moss y agrolita. Las plantas se mantuvieron a capacidad de campo y se fertilizaron con triple 18 con microelementos.

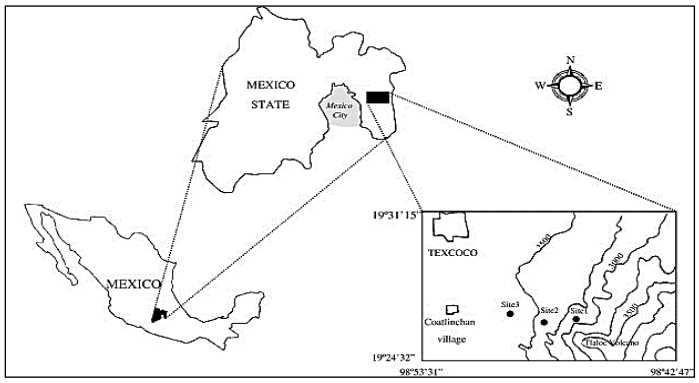


Figura 3.2 Localización del sitio experimental Campus Montecillo.

3.2 IMÁGENES DIGITALES DE LAS PLANTAS EN MACETA

Se obtuvieron 14 imágenes digitales durante la floración (49 días después de la poda) en el plano horizontal de las plantas de petunia en maceta, de modo que en cada fotografía hubiera una maceta. Dentro de la caja, la iluminación fue con dos lámparas fluorescentes colocadas en el plano horizontal; las imágenes digitales se tomaron a través de una abertura en la parte superior de la caja con una cámara digital SONY CYBER-SHOT con una abertura de diafragma de 9.0, una velocidad de obturación de 13, ISO 100, balance de blanco fluorescente, sin flash y se guardaron en formato JPEG a 2592 x 1944 pixeles y fueron comprimidas para su análisis posterior a 1556 x 1167 pixeles. A cada imagen se le aplicó un filtro de media con una matriz convolutiva de 3 x 3 para eliminar ruido.

3.3 MUESTREO DE LAS IMÁGENES DIGITALES

Con el propósito de crear un conjunto de datos de entrada a partir de las imágenes digitales de las flores de petunia, se realizó un muestreo sistemático de recuadros de imagen de tamaño 10 x 10 pixeles (Figura 3.3) de área que correspondió a flor, hoja y fondo de la imagen, seleccionando áreas contrastantes de un mismo objeto.

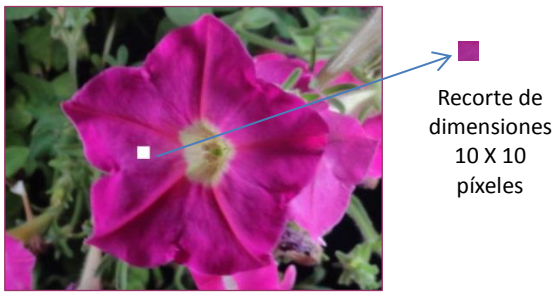


Figura 3.3 Obtención de un segmento de 10 x 10 pixeles de una imagen de flor.

Se agrupó cada área específica en clases. Se tienen 7 clases para la variedad multiflora y 3 clases para la púrpura. Un ejemplo de selección de clases se ilustra en la Figura 3.4, donde cada clase pertenece a alguna de las siguientes categorías:

- Tonalidades de flores blancas
- Tonalidades de flores rosas
- Tonalidades de flores bicolor (colores blanco y lila)
- Tonalidades de verde (hojas, pedúnculos, sépalos)
- Tonalidades de negro (fondo, que no es totalmente negro)

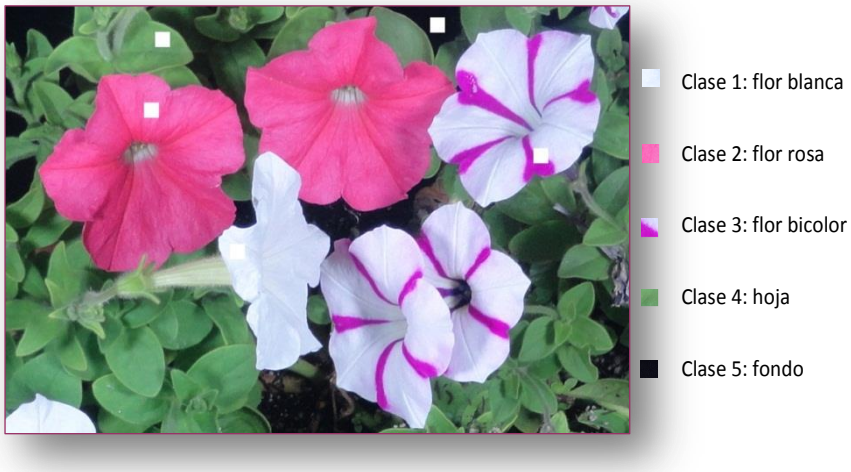


Figura 3.4 Ejemplo de selección de clases utilizando una fotografía de petunias multiflora.

Una vez identificadas las clases se forman conjuntos de datos, con tamaños que dependen de la variedad, y del número de repeticiones por objeto, finalmente se eliminan todos los pixeles redundantes. El Cuadro 3.1 muestra el número de pixeles iniciales y finales, por clase y genotipo de flor.

Cuadro 3.1 Conjunto total de datos de las variedades de petunia: 20,800 pixeles, posteriormente se reducen a 8,270 pixeles.

Multiflora			Púrpura		
Clase	**Muestras**	**Muestras reducidas**	**Clase**	**Muestras**	**Muestras reducidas**
Blanca	1200	221	Púrpura	2800	1144
Fucsia	2400	1259	Hojas	2800	1485
Rosa	400	259	Fondo	2800	141
Azul	2400	1516			
Bicolor	400	387			
Hojas	2800	1738			
fondo	2800	120			
Total	**12400**	**5500**		**8400**	**2770**

El procedimiento de muestreo es detallado en los siguientes puntos:

1. Adquisición de 14 imágenes digitales de flores de petunia.

2. En la herramienta de Microsoft Office Paint, se abre la imagen digital de las flores en maceta (una foto por maceta).

3. Se realiza una identificación visual de cada categoría: fondo, tipo de flor, hojas.
4. Se seleccionan pequeños cortes o segmentos de 10 x 10 pixeles para cada objeto en la imagen perteneciente a una categoría por medio del ratón.
5. Se eligen diversas tonalidades de un mismo objeto en la imagen.
6. De los cortes anteriores, se eligieron 4 segmentos por objeto (i.e., si en una foto hay 3 objetos, se eligen 4 segmentos de cada una, obteniendo 12 segmentos), Figura 3.5.
7. Cada segmento se guarda como una imagen nueva en formato JPEG sin compresión.
8. Se crearon dos carpetas para guardar los segmentos según la variedad de flor de la cual proceden, i.e., cada carpeta contiene los segmentos provenientes de 7 fotografías.
9. Por lo tanto, del conjunto inicial de fotografías se extrajo un total de 208 nuevas imágenes que conforman los segmentos, y cada imagen, como ya se explicó, tiene dimensiones de 10 x 10 pixeles, así, se cuenta con 20,800 (208 x 10 x 10) pixeles, donde 12,400 pertenecen a la variedad multiflora y 8,400 pertenecen a la púrpura.
10. Para cada una de las carpetas antes mencionadas, se ordenan las imágenes por colores, de esta forma serán utilizadas por un programa en Matlab que procesa pixel por pixel y los transforma en sus espacios de color; se obtendrán: una matriz donde cada hilera es el registro de los 8 canales de color de un pixel determinado; además, un vector objetivo conteniendo las clases de las cuales proceden los pixeles según su colocación en el arreglo.

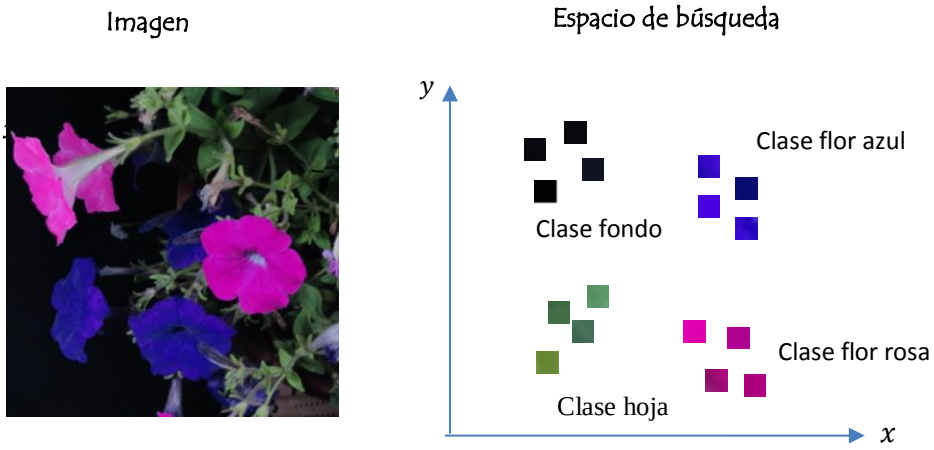


Figura 3.5 Recortes de 10 x 10 pixeles de la imagen agrupados por tonalidades del color.

3.4 EXTRACCIÓN DE VARIABLES DE COLOR

Se obtuvieron los valores de los canales R, G, y B en el rango [0,256] del modelo de color RGB de cada uno de los pixeles de las 208 imágenes. Una representación en la Figura 3.6.

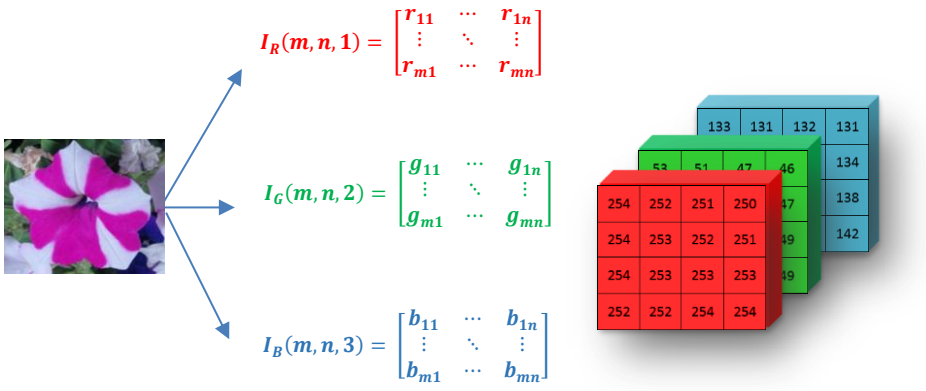


Figura 3.6 Ejemplo de una muestra de 4 x 4 pixeles que es representado en el modelo RGB por tres matrices que corresponden a sus valores en los canales: rojo, verde y azul.

Después, a partir del espacio de color RGB se hicieron las transformaciones a los espacios de color CIE Lab y LCH. Con esto, se obtuvieron ocho variables de entrada que corresponden a cada canal de los espacios de color obtenidos, una vez eliminados los canales redundantes. Cada pixel (i.e. una muestra a la red) representado por sus diferentes canales de color representa una entrada a la ANN. Esto es, cada pixel es

representado por una combinación de valores numéricos, donde algunos por su similitud representaran patrones reconocidos por una red neuronal en la etapa de clasificación. La distribución final de las muestras dependió del número de clases por variedad, número de repeticiones por categoría, y finalmente la eliminaron los pixeles repetidos, puede visualizarse en la gráfica siguiente.

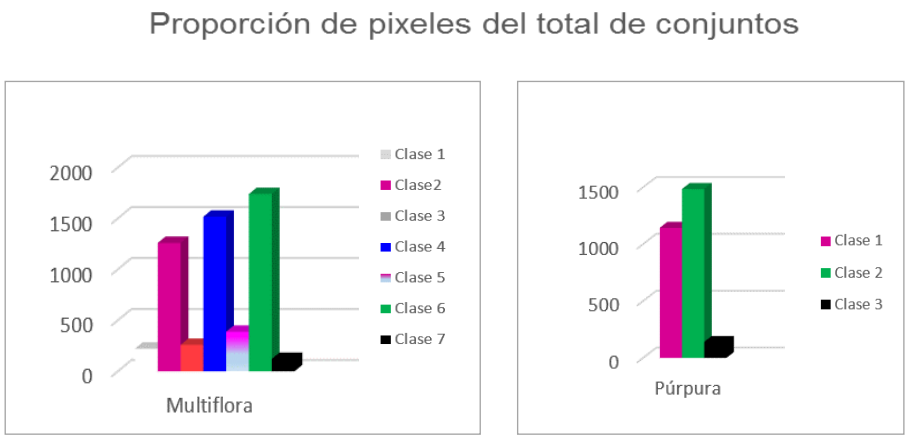


Figura 3.7 Distribución final de los pixeles que conforman la base de datos.

Se obtiene finalmente, una base de datos agrupando todas las muestras o pixeles en carpetas según el genotipo de flor del cual provienen las muestras. La base de datos queda reducida de un total de 20,800 pixeles a un total de +/- 8,270 pixeles diferentes. Por medio de un programa se vuelven a preparar los datos para obtener la matriz de entrada a la red y el vector de respuestas deseadas eliminando las columnas complementarias.

3.5 IMPLEMENTACIÓN DE REDES NEURONALES ARTIFICIALES

La base de datos de la red neuronal fue construida con 8 bandas de color (espacios de color RGB, CIE Lab y LCH) como variables independientes, y como variable respuesta, se asociaron 7 categorías de color para la variedad multiflora y 3 para la púrpura. Se evaluaron dos modelos ANN: la Red Neuronal Perceptrón Multicapa (MLP) y la Red Neuronal Probabilística (PNN). La implementación de los clasificadores de ANN se

realizó en la plataforma de modelación Matlab versión 7.10.0 (R2010a) para el sistema operativo Windows, con el *Neural Network Toolbox*.

3.5.1 Disposición de la colección de datos para probar las redes

Debido a que los datos provienen de fotografías de flores de dos variedades de petunia, la colección de datos se divide en dos carpetas Cuadro 3.1. A su vez, para cada variedad juegan otros factores importantes para realizar las pruebas, formándose así, distintos repositorios de datos.

Cuadro 3.2 Carpetas principales señalando la variedad de la que proceden.

NOMBRE DE LA CARPETA	VARIEDAD
BASEFRAG1	Multiflora
BASEFRAG2	Púrpura

Las pruebas que permitieron encontrar diferencias entre ambos modelos de ANN consistieron en: variar en el tamaño del conjunto de datos de entrada y definir diferentes configuraciones de los modelos de color. Considerando esto, cada una de las carpetas principales deben contener subcarpetas separando aún más los datos (por tamaño y espacio de color). Por ejemplo, las subcarpetas dentro de BASEFRAG1 pueden verse en el Cuadro 3.3.

Cuadro 3.3 Nombre de las subcarpetas de la carpeta BASEFRAG1.

NOMBRE DE LA SUBCARPETA
BASE1-050-LAB
BASE1-050-LCH
BASE1-050-RGB
BASE1-050-TODO
BASE1-100-LAB
BASE1-100-LCH
BASE1-100-RGB
BASE1-100-TODO

Los juegos de entradas por subcarpeta consisten en trabajar con un canal de color por separado o bien todos los canales juntos (etiquetado como TODO) y en combinación con un tamaño de población dado.

Se eligieron dos tamaños de población distintos para las pruebas: el total y la mitad (haciendo que en las distintas clases exista esa misma proporción al 50%, i.e., 50% de todos los 1's, 50% de todos los 2's y 50% de todos los 3's). Para lograrlo, se separó el total de la población en subgrupos (subgrupo I con respuestas 1, subgrupo II con respuestas 2, y subgrupo III con respuestas 3). Adicionalmente, se realizó un proceso aleatorio sobre el porcentaje elegido en cada uno de los tres subgrupos, después se reúnen formando un nuevo grupo de pixeles. Lo anterior se automatiza por medio de un programa en Matlab. Por lo que además, el programa carga en el espacio de trabajo de Matlab las carpetas y subcarpetas sucesivamente asignando la dirección, por ejemplo:

```
load ('BASEFRAG1/BASE1-050-LCH/EZ.mat');
```

Refiriéndose a la subcarpeta BASE1 dentro de la carpeta BASEFRAG1 que contiene un tamaño de 50 por ciento y modelo de color LCH y se crea el archivo *.mat* que contiene la matriz de entrada y vector objetivo con la configuración mencionada, y se asigna a la carpeta. Por ejemplo, la instrucción anterior es una combinación de variables: variedad multiflora, tamaño 50% y modelo LCH.

De esta forma, el conjunto de bases de datos contiene todas las combinaciones de variables existentes para las pruebas. En este punto, la colección de datos ha sido categorizada por las combinaciones descritas.

3.5.2 Configuración experimental para las redes neuronales

La implementación de una red neuronal específica en Matlab se encuentra en su librería de redes neuronales. Mediante este recurso podemos definir la estructura de la red, funciones de activación, regla de aprendizaje y algoritmo de entrenamiento, entre otros.

Para cada prueba, se utiliza el mismo juego de datos para ambas redes en una prueba, ya que el formato de datos de entrada es igual para el MLP y la PNN. Como es conocido, los modelos no deben ser validados con los mismos datos usados para el entrenamiento. Por lo tanto, se realizó un procedimiento de división de datos de Matlab mediante la función *dividerand* llamado Random Data División de forma que obtengamos K subconjuntos de datos seleccionados aleatoriamente.

Es importante que la colección de datos cubra el rango completo del espacio de entrada, por lo que se usa la función *mapminmax* para escalar o normalizar las entradas en el rango [-1,1]. De esta forma, la matriz de entrada XT será normalizada así:

```
[XTn, es] = mapminmax (XT);
```

Se carga en la ventana de trabajo de Matlab la matriz XTn (conjunto de entrada seleccionado previamente normalizado, antes denotado por XT) y YT (vector objetivo), contenidos en la base de datos elegida. Los siguientes pasos describen las etapas para crear, entrenar y simular las redes neuronales:

1. El conjunto de entrada se divide para las fases de entrenamiento y prueba:
    - XEn y YE Entrenamiento (80%)
    - XPn y YP Prueba (20%)
2. Se ejecuta la función *newpr* (con argumentos tansig, tansig, trainscg). Esta función es un modelo de red perceptrón multicapa.
3. Simulamos la red para el entrenamiento utilizando XEn y YE (se debe especificar la función *dividerand* con la proporciones 100, 0, 0).
4. Simulamos la red con XPn y YP como argumentos a la red para probarla.
5. Se ejecuta el modelo *newpnn*. Función que crea una red neuronal probabilística.
6. Se entrena la red con XEn y YE.
7. Se simula la red con XPn y YP.

8. Se comparan ambos modelos analizando resultados de los porcentajes de clasificación almacenados en la matriz de confusión.
9. Se repite todo el proceso para nuevas entradas, verificando los porcentajes de clasificación de las matrices de confusión.

Este procedimiento se repite con 25 particiones aleatorias de los conjuntos de datos de entrenamiento y prueba. Los resultados de cada ciclo se pasan a un cuadro con los siguientes datos: Número de la partición, modelo de ANN indicando los porcentajes de clasificación en el entrenamiento y en la prueba. De cada cuadro podemos obtener los promedios de los porcentajes de clasificación de entrenamiento y prueba globales.

3.5.3 Estrategia de clasificación del perceptrón multicapa (MLP)

En la red neuronal perceptrón multicapa se usa una red de alimentación hacia adelante "*feedforward*", con capas totalmente conectadas, probada con un algoritmo de aprendizaje de retropropagación. Para la variedad petunia multiflora se definen 7 categorías de salida y para petunia púrpura, 3 categorías. El número de categorías o clases varía según la variedad de la flor, ver Figura 3.8.

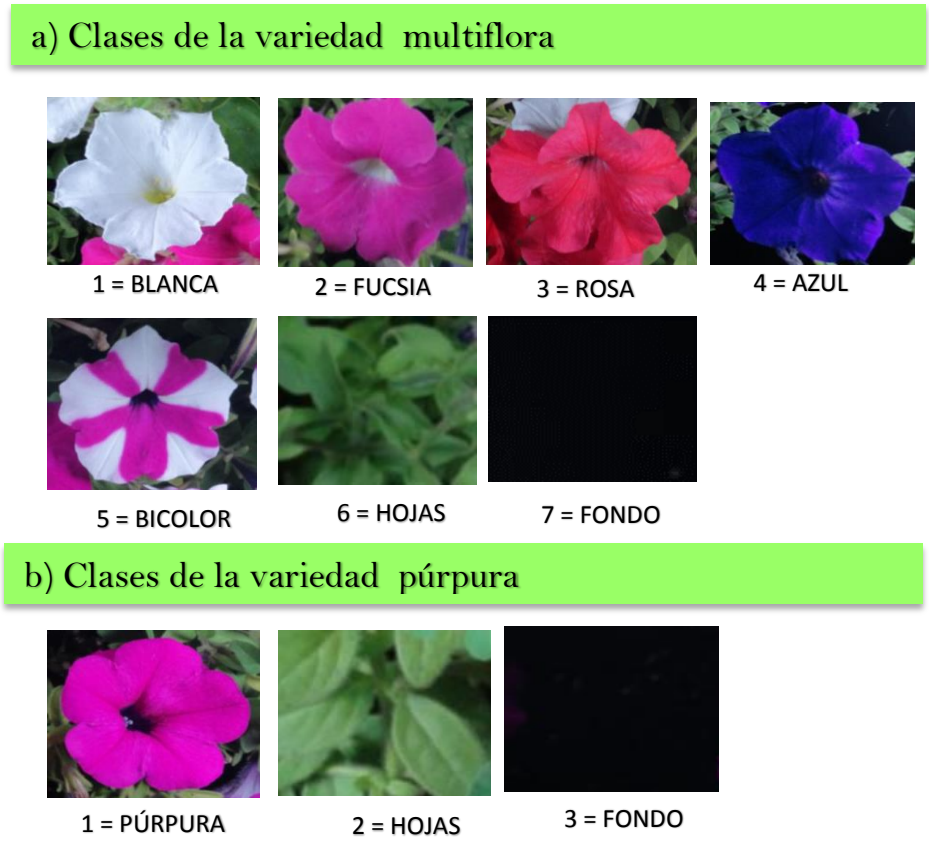


Figura 3.8 La selección de clases depende de la variedad de flor, son 7 clases para multiflora y 3 para púrpura etiquetadas con números para las redes MLP y PNN.

En la Figura 3.9 la estructura de la red para el caso de la variedad petunia púrpura debe contener las clases: 1= flor púrpura, 2 = hoja, 3 = fondo; las entradas provienen de 8 canales de color y solo hay una capa oculta con varias neuronas (color azul en la figura).

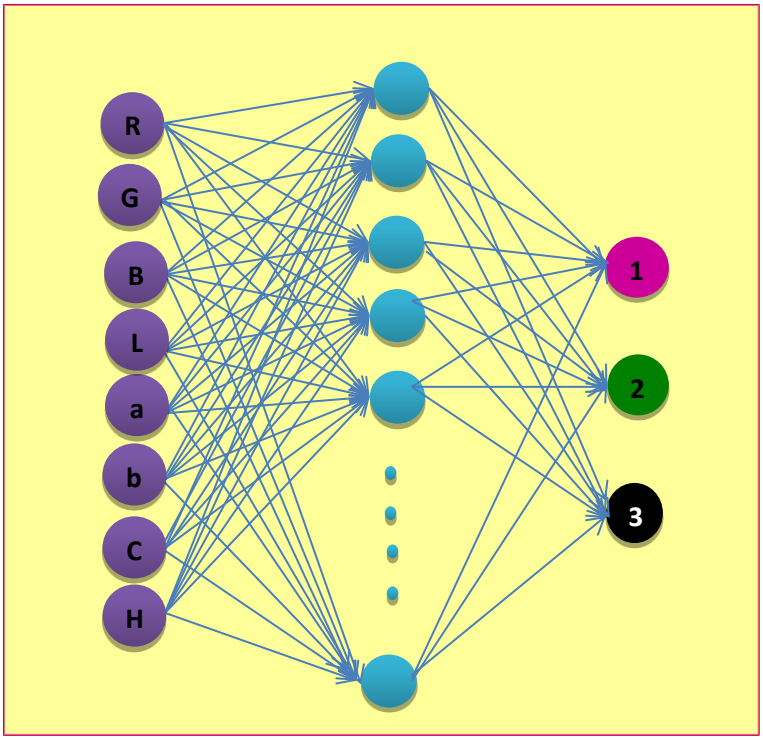


Figura 3.9 Estructura del modelo perceptrón multicapa con clases asignadas para las flores petunia purpura.

Para crear una arquitectura de red que optimice el funcionamiento del perceptrón multicapa se realizan diversas pruebas modificando los parámetros de la red:

- número de capas ocultas
- número de neuronas en cada capa
- funciones de activación o transferencia
- algoritmo de entrenamiento

3.5.3.1 Creación de la red MLP

La red neuronal debe ser creada, configurada e inicializada. En la herramienta *Neural Network Toolbox* de Matlab se tienen objetos de red donde se crea y almacena toda la información de una red neuronal; para crear la red perceptrón multicapa con conexiones hacia delante se utiliza la función *newpr* y se le asigna una variable que la identifique como *red*. Algunos subobjetos importantes del objeto *red* son: entradas, capas (se refiere

a las capas ocultas), salidas, sesgos, pesos de entrada y capa de pesos. A continuación se muestran el comando de la función para crear una red MLP:

```
red = newpr(XTn, T, num_capas, activacion);
```

La función *newpr* toma los siguientes argumentos:

1. XTn: es una matriz cuyo número de filas es igual a 8, y el número de columnas coincide con el número de entradas. Es la matriz traspuesta de la entrada de datos original. Se restringen las entradas de la red a ciertos valores mínimos y máximos lo cual debe ser evaluado en esta matriz.
2. num_capas: es un vector que indica las neuronas en cada capa de la red y el número de neuronas de salida. El tamaño del vector indica el número de capas que contiene la red.
3. activacion: es un vector de varias cadenas de caracteres que corresponde al código para las funciones de activación que tendrán todas las neuronas de una capa. Las cadenas se colocan en el mismo orden de las capas de la red, i.e. la función tiene la misma posición de la capa a la que afecta.
4. Entrenamiento: el último argumento indica el tipo de entrenamiento que utilizará la red. Si no se indica Matlab por default ejecuta el entrenamiento *trainlm*.

Opcionalmente, puede ejecutarse esta última función mediante *red**.**trainFcn* ejemplo:

```
red.trainFcn = 'trainlm';
```

La configuración de la red consiste en organizar la red de forma que sea compatible con el problema que se quiera resolver, según los datos de la muestra. Antes de entrenar la red MLP (*feedforward*), se deben inicializar pesos y sesgos. El comando de configuración automáticamente inicializa los pesos, pero si se desea reinicializarlos se hace con el comando *init*. Esta función toma un objeto de red como entrada y regresa un objeto de red con todos los pesos y sesgos inicializados. El siguiente comando permite inicializar los pesos del objeto red:

```
red = init(red);
```

3.5.3.2 Número de capas y neuronas en cada capa

Para especificar el número de capas, y número de neuronas por capa se analizaron diferentes combinaciones y seleccionaron las arquitecturas con mejor desempeño. Dado Por ejemplo para especificar una arquitectura MLP con cuatro capas ocultas, con 25, 17, 8 y 7 neuronas respectivamente el comando en MATLAB sería:

```
num_capas = [25 17 8];
```

3.5.3.3 Función de activación

Las funciones de activación calculan la salida de una capa desde la entrada de la red. El MLP puede modificar su función de transferencia de salida, puede ser una función lineal *purelin*, una tangente hiperbólica sigmoidal *tansig*, o bien una sigmoidal logarítmico *logsig*. Para asignar la función de transferencia a la capa *i* de una red, por ejemplo la función *purelin* se escribe:

```
red.layers{i}.transferFcn = 'purelin';
```

Enseguida se presentan las funciones utilizadas para probar el MLP con los algoritmos: *purelin*(n); *tansig*(n) y *logsig*(n). Se definen las funciones de activación que se utilizarán en cada capa como sigue:

```
activacion = {'tansig' 'tansig' 'tansig' 'tansig'};
```

3.5.3.4 Entrenamiento

Una vez que los pesos de la red y los sesgos son inicializados, la red está lista para el entrenamiento. Las redes fueron entrenadas para reconocer pixeles y categorizarlos según su intensidad o color. Para el entrenamiento de una MLP se usa el algoritmo

backpropagation. El proceso de entrenamiento de la red consiste en el ajuste de los valores de los pesos y sesgos de la red para optimizar el desempeño de la red, tal como se define en la función de desempeño de la red. Para obtener el error de la red en redes MLP se escribe:

```
red.performFcn
```

Para entrenar la red se ejecuta la función *train*. Previamente se utiliza la función *dividerand* con las siguientes proporciones 100, 0, 0 de entrenamiento, validación y prueba. La función *train* toma como argumentos el nombre del objeto de red *red*, la matriz de elementos de entrada normalizada *XTn* y sus salidas deseadas (en la matriz nombrada *T*).

```
[red, dome]= train (red, XTn, T);
```

La variable *dome* da información acerca del proceso de entrenamiento, mientras que la variable *red* es la red llamada luego de ser entrenada (sus pesos y sesgos se han ajustado). El comando *train* representa un entrenamiento en modo por lotes, lo que significa que todas las entradas en el conjunto de entrenamiento se aplican a la red antes de que los pesos se actualicen. Una vez entrenada la red pueden ser probadas bajo un nuevo conjunto de datos.

Para optimizar la función de desempeño se puede usar cualquier algoritmo de optimización numérica estándar, pero existen algunos que demuestran un gran desempeño en el entrenamiento de una red MLP. Estos métodos de optimización usan tanto el gradiente del desempeño de la red con respecto a los pesos de la red, como el jacobiano de los errores de la red con respecto a los pesos. El gradiente y el jacobiano son calculados usando el algoritmo backpropagation, que implica cálculos que se realizan hacia atrás a través de la red.

3.5.3.5 Algoritmos de aprendizaje

Una lista de algoritmos de aprendizaje está disponible en el *Neural Network Toolbox*, los cuales usan métodos basados en el gradiente – o Jacobiano. Para la red se realizaron pruebas con 3 de ellos: el algoritmo Levenberg–Marquardt cuya función es *trainlm*, el algoritmo *Resilient Backpropagation* con la función *trainrp* y el algoritmo *Scaled Conjugate Gradient* que se ejecuta con *trainscg*. La función de entrenamiento más rápida es generalmente *trainlm*. El método tiende a ser menos eficiente para redes más grandes (con miles de pesos), porque requiere más memoria y más tiempo de computación. Para el entrenamiento de redes grandes y de redes de reconocimiento de patrones, son ideales las funciones *trainscg* y *trainrp*. Sus requerimientos de memoria son mínimos, pero son más veloces que los algoritmos estándar del descenso del gradiente. Durante el entrenamiento, el proceso es constantemente actualizado en la ventana de entrenamiento (ver Figura 3.10). De mayor interés son

- El desempeño
- La magnitud del gradiente del desempeño
- El número de comprobaciones de validación

La magnitud del gradiente y el número de comprobaciones de validación se utilizan para terminar el entrenamiento. El gradiente se hará muy pequeño conforme el entrenamiento alcance un mínimo del entrenamiento. Si la magnitud del gradiente es menor que $1e-5$, el entrenamiento termina. Este límite se puede ajustar mediante el establecimiento del parámetro *red.trainParam.min_grad*. El número de comprobaciones de validación representa el número de iteraciones sucesivas que el desempeño de la validación no disminuye. Si este número llega a 6 el entrenamiento se detiene [Beale *et al.*, 2010]. Se puede cambiar este criterio al establecer el parámetro *red.trainParam.max_fail*. De no ser así Matlab establece el número a 6 por omisión. En este trabajo el número de comprobaciones de validación no se utiliza. La razón se debe a que en el entrenamiento de una red MLP Matlab realiza su propia división de los datos, sin embargo para este

trabajo los conjuntos de entrenamiento y prueba deben dividirse y provenir de la misma fuente para ambas redes MLP Y PNN, por lo tanto el parámetro *red.trainParam.max_fail,* el cual requiere que los datos estén divididos, se omite. Al tener un número suficientemente grande como conjunto de entrada, este criterio no es necesario para evitar el sobreentrenamiento.

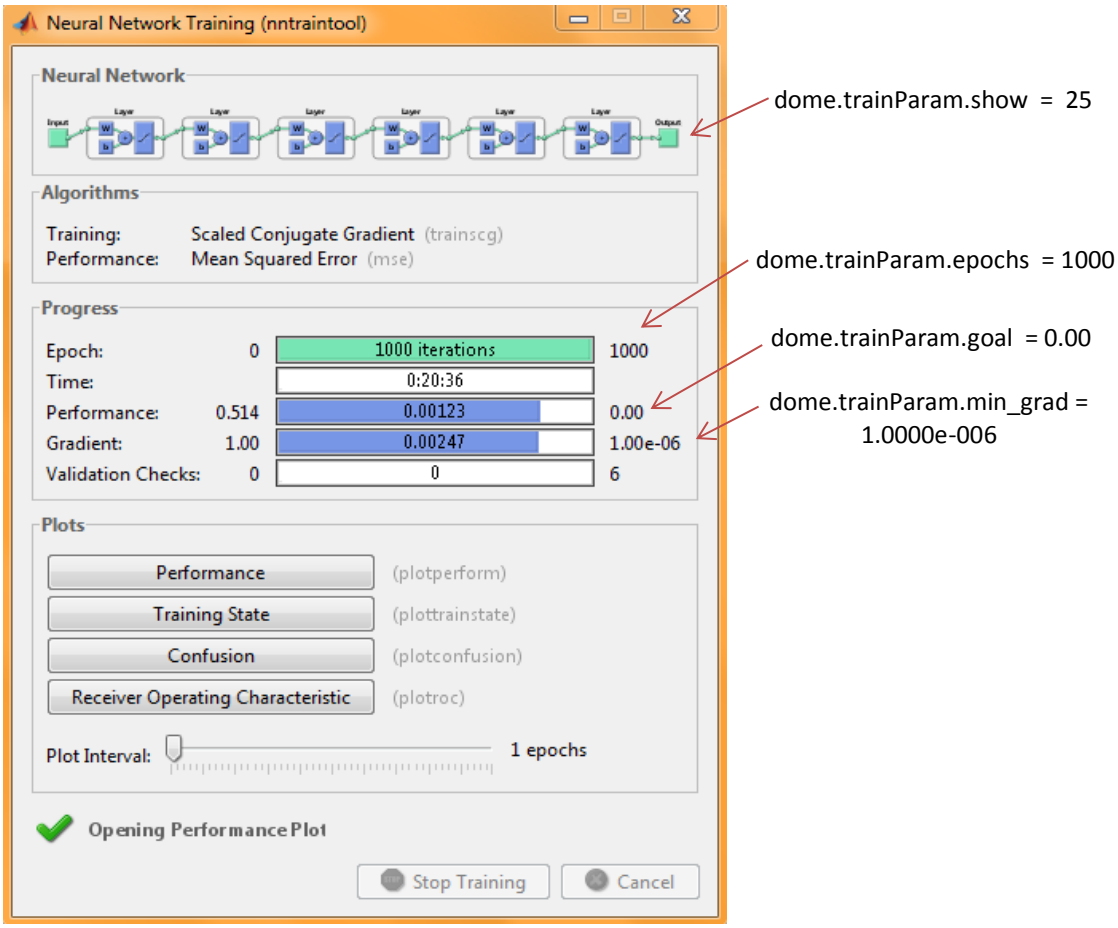


Figura 3.10 Ventana de entrenamiento que muestra funciones, progresos y gráficas usados en el entrenamiento.

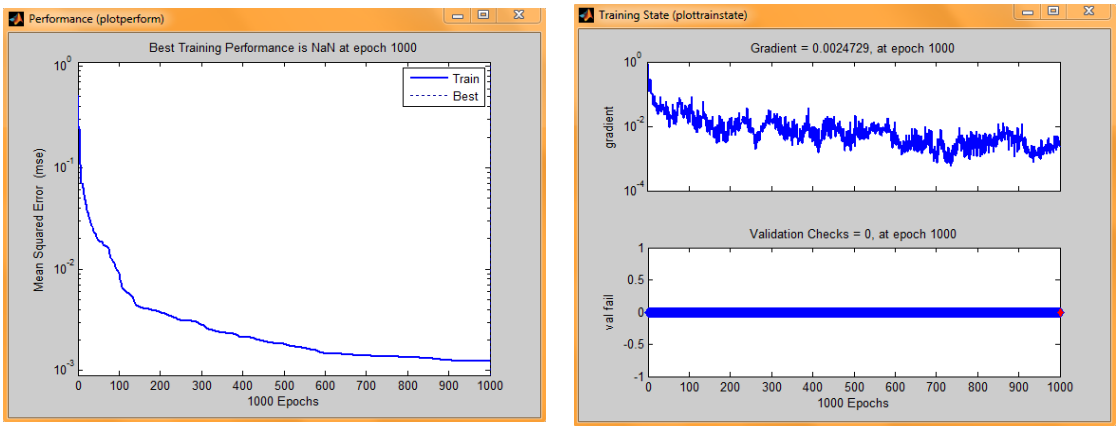
Esta ventana muestra que se determinaron cinco capas ocultas, se eligió el método de entrenamiento *Scaled Conjugate Gradient* (*trainscg*) y la función de desempeño del cuadrado medio del error (*mse*). Los criterios existentes para delimitar la red evitando el sobre entrenamiento son listados a continuación [Beale *et al.*, 2010].

1. epochs: es el máximo número de épocas de entrenamiento.

2. goal: es el valor límite que puede alcanzar la función de error de la red.
3. time: es el tiempo máximo en segundos que puede durar el entrenamiento de la red. Si el entrenamiento alcanza ese valor, finaliza.
4. min_grad: valor mínimo que debe tener el gradiente para poder parar el algoritmo.
5. max_fail: es el máximo número de iteraciones del ciclo permitido para incrementar el error de validación antes de parar el proceso.

En esta investigación serán utilizados como parámetros el *goal* y *min_grad*, debido a que son indicadores de cuanto ha aumentado el error cuadrático medio para la entrada elegida y el número de *epochs* por ser este un criterio que permite dar tiempo suficiente a la red de encontrar una solución aunque las pruebas sean extensas como por ejemplo, en este caso al hacer pruebas con varias capas ocultas.

Desde la ventana de entrenamiento, se puede acceder a cuatro graficas: desempeño, estado del entrenamiento, matriz de confusión y característica operativa del receptor ROC por sus siglas en inglés.



(a) Ventana de entrenamiento

(b) Ventana de variables de entrenamiento

Figura 3.11 Ventana de entrenamiento.

La gráfica de desempeño, Figura 3.11 (a), muestra el valor de la función de desempeño en comparación con el número de iteración. La gráfica del estado de entrenamiento,

Figura 3.11 (b), muestra el progreso de otras variables de entrenamiento, como la magnitud del gradiente y el número de comprobaciones de validación.

3.5.3.6 Simulación de la red

La ANN generará las salidas a partir de las entradas elegidas utilizando la función *sim*. La función *sim* devuelve un vector de salidas obtenidas de cada neurona de salida de la red.

Dado que se desean obtener varias simulaciones de un tiempo se introduce una matriz, cuyos vectores sean cada uno de los vectores de entrada de los que se quiere obtener su salida. Para obtener la respuesta de la red *red* dadas las entradas XTn, se escribe:

```
Yp = sim (red, XTn);
```

3.5.4 Estrategia de clasificación con una red neuronal probabilística (PNN)

La PNN es una familia de las redes neuronales de base radial, surgen a partir del estudio de clasificadores de Bayes. Una PNN es de estructura simple, ver Figura 3.12, es utilizada en problemas de clasificación [Adeli, 2009; Specht, 1988], su velocidad de aprendizaje es considerablemente mayor debido a que su ajuste de pesos no es iterativo, aunque requiere más neuronas que la red MLP. La PNN consta de cuatro capas: la capa de entrada, la capa de base radial, la capa competitiva y la capa de salida.

Capa de entrada: contiene m neuronas, cada neurona representa un canal de color.
Capa de neuronas de base radial o capa de patrones: con un número de neuronas igual al número de vectores de entrenamiento. Cada neurona calcula la gaussiana de la distancia Euclídea entre el presente vector de entrada y el vector objetivo y produce un vector cuyos elementos indican la cercanía entre dichos vectores.
Capa competitiva o sumatoria: cuyo número de neuronas es igual al número de categorías consideradas en el problema de clasificación (en la Figura 3.14 son 3). Cada

neurona suma la salida de la capa patrón para cada clase de entrada y produce con ellas un vector de salida con las probabilidades. En la salida una función de transferencia de tipo competitivo toma la máxima de estas probabilidades y produce un 1 para esa clase y cero para las demás.

Capa de salida: contiene neuronas binarias que producen la decisión de clasificación.

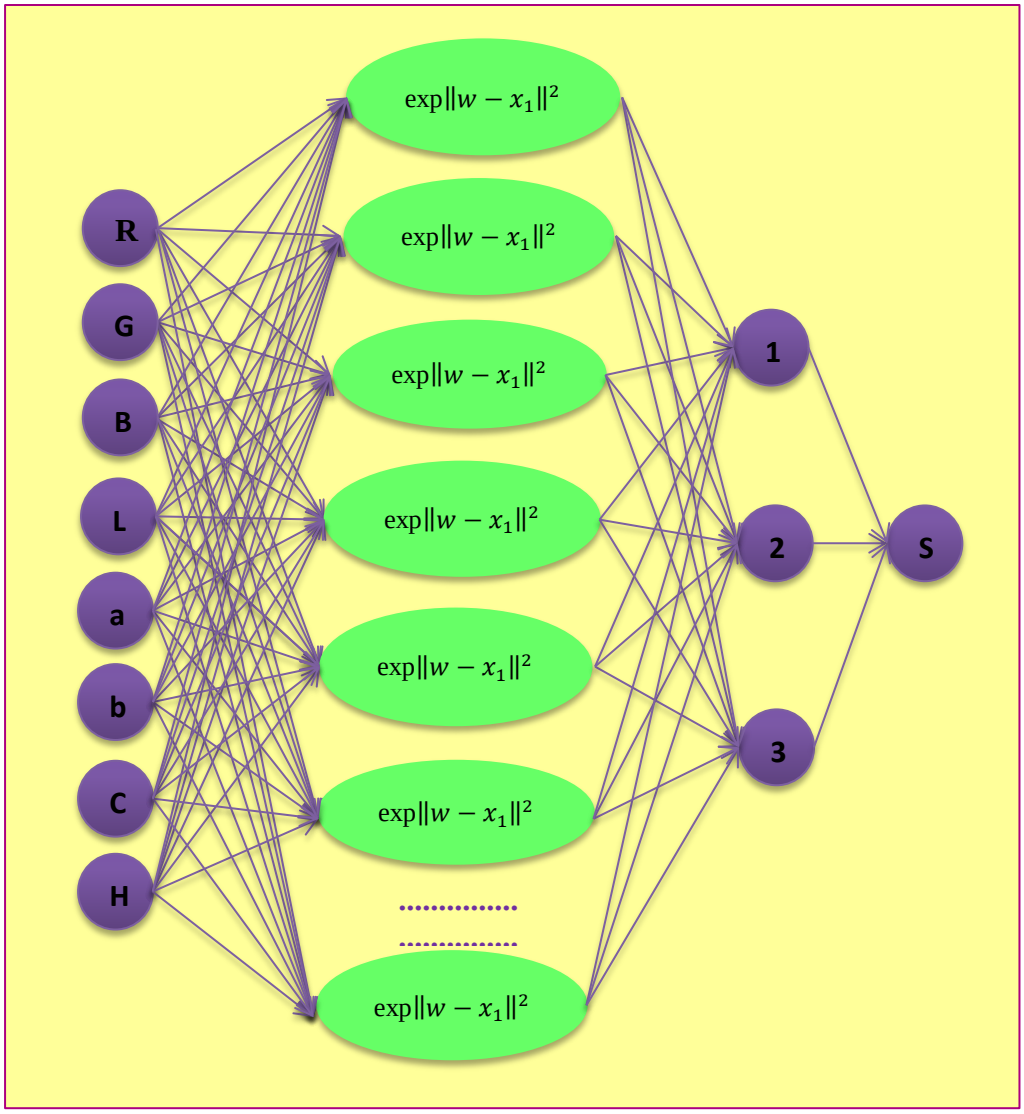


Figura 3.12 Arquitectura de la PNN para predecir colores en una imagen a partir de los valores de los canales de color.

3.5.4.1 Creación de la red

Como se mencionó anteriormente tenemos una matriz XTn (conjunto de entrada normalizado) y el correspondiente Tc (vector objetivo), convertimos los índices de las clases del objetivo Tc a vectores T:

```
T = ind2vec(Tc);
```

Establecemos un valor diferencial o *spread* de 1, ya que es una distancia típica entre los vectores de entrada.

```
spread = 1;
```

Se usa la función newpnn para crear una PNN y se le asigna una variable que la identifique como *redprob*. A continuación se muestran el comando de la función para crear una PNN:

```
redprob = newpnn(XTn,T,spread);
```

3.5.4.2 Función de activación

La PNN utiliza la función de transferencia Gaussiana o de base radial denominada radbas, su algoritmo es el siguiente:

```
radbas(n) = exp(-n^2)
```

Aquí la entrada de la red para la función de transferencia *radbas* es la distancia vectorial entre su vector de pesos W y el vector de entrada XTn, multiplicado por el sesgo *b*. La red *newpnn* crea una red de cuatro capas. La segunda capa tiene neuronas *radbas*, y calcula sus entradas ponderadas con *dist* y la entrada neta con *netprod*. La tercera capa tiene neuronas *compet*, y calcula su entrada ponderada con *dotprod* y sus entradas netas con *netsum*. Sólo la segunda capa tiene sesgos.

3.5.4.3 Simulación de la red

La ANN generará las salidas a partir de las entradas elegidas utilizando la función *sim* como en la red MLP. Por ejemplo, simulamos la red, usando la entrada XTn para asegurar que produzca las clasificaciones correctas.

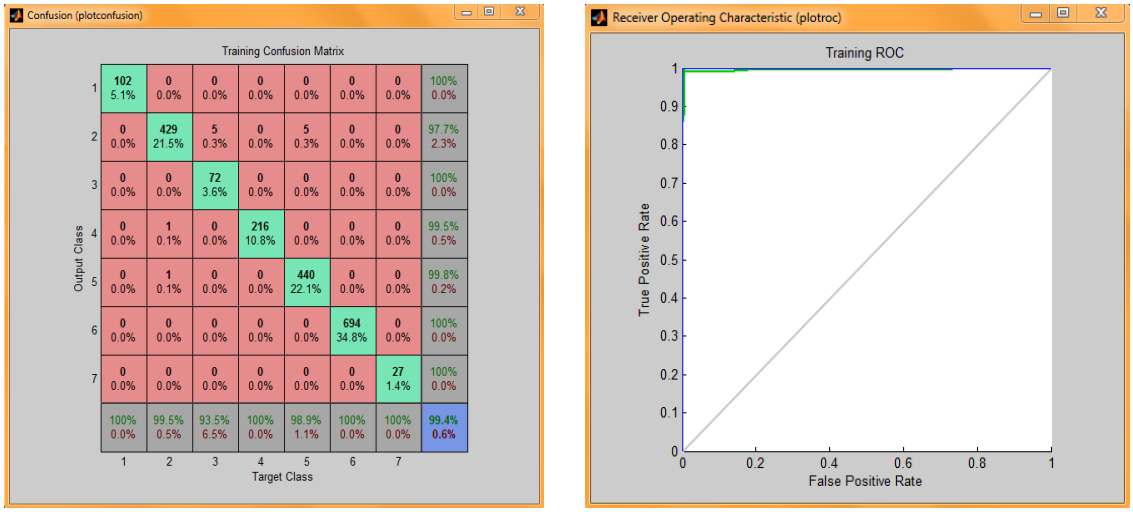
```
Y = sim(redprob,XTn);
```

Se utiliza la función *vec2ind* para convertir la salida Y a una fila de índices vectoriales Yc para hacer las clasificaciones claras.

```
Yc = vec2ind(Y);
```

3.5.4 Evaluación de las redes MLP y PNN

Con el propósito de comparar el desempeño de las ANN propuestas se utilizan dos criterios la matriz de confusión, Figura 3.13 (a), y la gráfica de la curva característica operativa del receptor (ROC) Figura 3.13 (b).



(a) Matriz de Confusion

(b) Curva ROC

Figura 3.13 Evaluación de las ANN mediante la gráfica de confusion y la curva característica operativa del receptor ROC.

La matriz de confusión sirve para visualizar la clasificación. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases. Las celdas diagonales en la tabla muestra el número de clases que fueron clasificados correctamente, y las celdas fuera de la diagonal muestran los casos mal clasificados. La celda azul en la parte inferior derecha muestra el porcentaje total de casos correctamente clasificados (en verde) y el porcentaje total de casos mal clasificados (en rojo).

En la gráfica de la curva característica operativa del receptor (ROC) las líneas de color en cada eje representan las curvas ROC para cada una de las 7 categorías. La curva ROC es una gráfica de la tasa de verdaderos positivos (sensibilidad) frente a la tasa de falsos positivos (1 - especificidad) como el umbral se varía. Una prueba perfecta mostraría puntos en la esquina superior izquierda, con una sensibilidad del 100% y un 100% de especificidad.

4. RESULTADOS Y DISCUSIÓN

En esta se sección se presentan los resultados de la comparación del modelo MLP y PNN. Se analizaron diferentes escenarios de modelación para determinar los juegos de parámetros óptimos de cada modelo y se evaluaron las eficiencias de cada modelo. Se interpretan y analizan los resultados obtenidos de las pruebas realizadas con los dos enfoques de estudio.

4.1 PRUEBAS PRELIMINARES

Para la realización de las pruebas se utilizaron los modelos MLP y PNN cuyos comandos en Matlab 2010a son *newpr* y *newpnn* respectivamente. Los conjuntos de datos contienen las variables de entrada y la variable respuesta, para cada variedad: multiflora y púrpura. Para cada variedad se crearon 8 subconjuntos de datos considerando dos tamaños 50% y 100% y cuatro subconjuntos de variables de entrada correspondientes a los modelos de color: RGB, CIE Lab, LCH y un subconjunto con todos los canales de los modelos de color anteriores (etiquetado como TODOS). Cada subconjunto de datos fue utilizado en el entrenamiento de los dos modelos estudiados para evaluar su desempeño en clasificación. En una primera aproximación, se utilizaron los parámetros por omisión de las funciones de ANN del Toolbox de Matlab. Además, para la red *newpr* se empleó una capa oculta con tres neuronas. Para la red *newpnn* se utilizó la media de las distancias Euclidianas entre pares de los vectores del conjunto de datos de entrada como valor por omisión del parámetro *spread* (ancho de banda de la función de base radial Gaussiana).

Los porcentajes de clasificación mostrados en los cuadros siguientes se refieren al promedio global de 25 corridas para las fases de entrenamiento y prueba de los dos modelos de red MLP y PNN. En las primeras 3 columnas se muestra la disposición de 8 combinaciones de tamaño y modelo de color de la variedad utilizada. Se indican por: “MLP%E” los resultados del modelo MLP para entrenamiento; “MLP%P” los resultados del modelo MLP para prueba; del mismo modo, “PNN%E” es el modelo PNN para

entrenamiento y “PNN%P” es el modelo PNN para prueba. Los conjuntos de datos referidos en el Cuadro 4.1 provienen de la carpeta de Matlab BASEFRAG1.

Cuadro 4.1 Promedio de porcentajes de clasificación de 25 corridas del algoritmo para cada conjunto de datos de la variedad Multiflora.

VARIEDAD 1: MULTIFLORA						
Conjunto	**Tamaño**	**Entrada**	MLP %E	MLP%P	PNN%E	PNN%P
1	**100**	**LAB**	90.63	90.72	**78.97**	**79.07**
2	**100**	**LCH**	84.46	84.48	58.54	58.75
3	**100**	**RGB**	83.47	83.54	76.91	77.09
4	**100**	**TODO**	**96.12**	**95.95**	78.28	78.00
5	**50**	**LAB**	91.07	91.11	**79.32**	**79.41**
6	**50**	**LCH**	85.51	85.61	58.87	59.64
7	**50**	**RGB**	85.18	85.32	76.98	76.40
8	**50**	**TODO**	**94.93**	**94.63**	78.11	78.22

Como se observa en el Cuadro 4.1 los mayores porcentajes de clasificación del modelo MLP se obtienen cuando se utilizan todas las entradas y el conjunto de datos completo; aunque el modelo de color CIE Lab genera valores altos de clasificación. Mientras que con el modelo PNN los mejores desempeños se obtienen cuando se utiliza el espacio de color CIE Lab y tamaño del conjunto de datos no presenta una diferencia significativa. Para estas pruebas con la variedad Multiflora, el modelo MLP fue superior al PNN.

El Cuadro 4.2 describe los resultados de las pruebas preliminares del análisis del conjunto de datos BASEFRAG2,

Cuadro 4.2 Promedio de porcentajes de clasificación de 25 corridas del algoritmo para cada conjunto de datos de la variedad Púrpura.

VARIEDAD 2: PÚRPURA						
Conjunto	**Tamaño**	**Entrada**	MLP%E	MLP%P	PNN%E	PNN%P
1	**100**	**LAB**	97.46	97.40	**95.00**	94.56
2	**100**	**LCH**	97.12	97.39	94.83	**95.04**
3	**100**	**RGB**	99.19	99.16	94.86	94.92
4	**100**	**TODO**	**99.79**	**99.75**	**94.97**	94.68
5	**50**	**LAB**	98.15	98.09	94.90	95.04
6	**50**	**LCH**	97.62	97.38	94.78	**95.17**
7	**50**	**RGB**	99.39	99.33	94.81	95.04
8	**50**	**TODO**	**99.61**	**99.52**	94.87	95.14

Se observa un aumento en los aciertos globales de las 4 pruebas, mayor de 90%, por otro lado, los resultados empiezan a emparejarse entre los dos modelos de red, aun así sigue siendo mayor el porcentaje en el modelo MLP. Se infiere que la clasificación mejora cuando los datos provienen de la variedad de flor púrpura al considerar menos clase (solo tres).

La información obtenida por las pruebas anteriores se desglosa en los siguientes apartados:

1) Tipos de red: MLP y PNN

Para la variedad multiflora, la red MLP es mejor que la red PNN; en el caso de la variedad púrpura, existe un aumento significativo en la clasificación para ambos modelos con respecto a la especie multiflora, sin embargo, los resultados entre redes muestra que la red MLP sigue siendo mejor que la red PNN en su tarea de clasificación.

2) Fases de modelación de la red: Entrenamiento y Prueba

Se observa que las salidas ofrecen un porcentaje de clasificación muy similar en entrenamiento y prueba en ambos modelos de red, por lo que no hay diferencias significativas entre los resultados del entrenamiento y los de la prueba, esto podría indicar que no hay sobre entrenamiento y el clasificador tiene un buen nivel de generalización Dado que los datos del conjunto de prueba proceden de la misma fuente, su construcción fue mediante el mismo procedimiento y sin ruido, es por eso que los resultados no pueden ser muy diferentes y el entrenamiento funciona correctamente.

3) Especie de la flor: Multiflora y Púrpura

La clasificación en las flores multiflora, independientemente de otras variables, es menos efectiva que en la variedad púrpura. Por ejemplo, aplicando la gran media $\bar{\bar{X}}$ para los resultados sobre la variedad púrpura y usando la red MLP para entrenamiento (columna

4, en el Cuadro 4.2), se obtiene un 98.54 por ciento de precisión, a diferencia de la multiflora (columna 4, en el Cuadro 4.1) que alcanza tan solo un 88.92% de precisión en las mismas condiciones. Con esto puede verse claramente que la variedad púrpura al tener flores del mismo color permite que una red tenga muchos patrones similares acelerando la convergencia. También puede ser una desventaja ya que se puede llegar a un sobre entrenamiento.

4) Combinación del conjunto de datos : Tamaño y Modelo de color

Considerando las combinaciones de tamaño de la población y modelo de color, los resultados no son decisivos. Para ambas variedades de petunia la clasificación más alta se obtiene con el tamaño completo 100% de la población usada y todos los canales de color “TODO”. Por otro lado, los peores resultados coincidieron para ambas variedades en el invernadero con 100% y LCH. Lo cual podría ser indicio de que un conjunto de tamaño menor no es necesariamente malo y un solo canal en este caso LCH y RGB no es muy eficiente.

4.2 PRUEBAS CON LA ARQUITECTURA DEL MODELO MLP

El modelo MLP depende del número de capas ocultas, el número de neuronas en cada capa, las funciones de activación y el algoritmo de entrenamiento.

4.2.1 Configuración del número de capas de la red MLP

Para probar la eficiencia del modelo MLP con respecto al número de capas ocultas y neuronas. Por capa, se eligieron las proporciones mostradas en el Cuadro 4.3.

Cuadro 4.3 Número de capas y neuronas en cada capa elegidas para prueba.

Capa	Número de neuronas en cada capa
2	15, 8
3	25, 17, 8
4	120, 60, 30, 15
5	154, 64, 40, 25, 16

Los porcentajes de clasificación de la red MLP en su fase de entrenamiento[1] de las pruebas preliminares (mismos que se encuentran de los Cuadros 4.1 y 4.2), se repiten en el Cuadro 4.4, indicando por colores los mejores (verde) y peores (rojo) resultados. El objetivo de esto es ver los resultados de esta red desde la entrada utilizada (tamaño y modelo de color).

Cuadro 4.4 Promedios de los porcentajes de aciertos para la red MLP en entrenamiento.

		Multiflora	Púrpura
100	LAB	90.63	97.46
100	LCH	84.46	97.12
100	RGB	83.47	99.19
100	TODO	96.12	99.79
50	LAB	91.07	98.15
50	LCH	85.51	97.64
50	RGB	85.18	99.39
50	TODO	94.93	99.61

Para las pruebas con el número de capas en la variedad púrpura, el mejor resultado fue 99.79% con la combinación 100-TODO, el peor resultado 97.12% con 100-LCH. Al variar las capas y neuronas el mejor y peor resultados siguen apareciendo en las mismas combinaciones que utilizando una capa oculta con tres neuronas (convención por default), ver Cuadro 4.5.

[1] Todos los resultados obtenidos con el conjunto de entrenamiento son coherentes con los arrojados con el conjunto de prueba.

Cuadro 4.5 Promedio de aciertos para ambas variedades en la red MLP en fase de entrenamiento, repitiendo las combinaciones donde se obtiene la mejor y peor respuesta de las pruebas por default para varios valores de capas y neuronas.

Numero de Capas	Multiflora		Púrpura	
Default	96.12¤	83.47¤¤	99.79¤	97.12¤¤
2	95.66	96.36	100	99.78
3	96.15	99.04	100	99.81
4	96.92	99.37	100	99.60
5	98.89	97.30	99.40	100

¤ La combinación de la mejor respuesta con una capa (default) se repite para toda la columna

¤¤ La combinación de la peor respuesta con una capa (default) se repite para toda la columna

En el cuadro anterior se toman como referencia las combinaciones (tamaño y modelo de color) de los mejores y peores resultados con una capa oculta para repetir el entrenamiento con varias configuraciones de capas. Nuevamente al ser los resultados muy similares, por tratarse de la variedad púrpura, es difícil ver alguna coincidencia, por lo que es mejor basarnos en los resultados con multiflora. El mejor resultado fue 96.12% con la combinación 100-TODO, el peor fue 83.47% con 100-RGB.

Como puede verse mediante los colores, la respuesta mejora mientras aumentan las capas pero no es una diferencia significativa. En cambio, la diferencia en tiempo de computación si es grande, mientras más capas, el proceso se vuelve sumamente lento. Se puede ver un ejemplo con la variedad 1, en la mejor respuesta los tiempos registrados de la última corrida son:

Cuadro 4.6 Tiempos de ejecución variando las capas en la variedad multiflora.

Número de capas	Tiempo
2	4:28
3	6:39
4	15:51
5	19:41

Como puede verse, para 5 capas son aproximadamente 20 minutos por corrida, sin embargo cada prueba se repite para 25 corridas o ciclos del algoritmo, lo que aumenta considerablemente el tiempo. Por lo tanto, el tiempo transcurrido es aceptable para 2 ó 3 capas porque mejora la respuesta y no sobrecarga recursos de computación.

4.2.2 Algoritmos de entrenamiento

Para realizar pruebas de entrenamiento con la red MLP, del Cuadro 4.4 se toman las combinaciones que dieron la mejor respuesta en multiflora: 100–TODO (100% de los datos y todos los modelos de color). Con dichas combinaciones se realizan pruebas variando el algoritmo de entrenamiento, además, debido a las pruebas de configuración de capas el número de capas ocultas utilizado será dos conteniendo 15 y 8 neuronas respectivamente. Los algoritmos de entrenamiento son métodos basados en el gradiente que son: Levenberg-Marquardt con función *trainlm*, *Resilient Backpropagation* cuya función es *trainrp* y *Scaled Conjugate Gradient* con función *trainscg* los resultados se muestra en el Cuadro 4.11.

Cuadro 4.7 Porcentajes globales de aciertos de la red MLP para varios algoritmos de entrenamiento en la variedad multiflora y combinación 100-TODO.

Algoritmo de Aprendizaje	Porcentaje de clasificación	Tiempo de la última corrida
trainscg	97.43	996s
trainlm	73.29	2398s
trainrp	**99.85**	351s

Observamos que existe una gran desventaja en ejecutar *trainlm*, una sola corrida puede durar hasta 40 minutos aproximadamente mientras que utilizando la función *trainrp* puede durar poco más de un minuto, además el porcentaje de aciertos es menor con *trainlm*.

Los resultados corroboran el hecho de que por más rápida que sea la función *trainlm* y pese a su popularidad, es cierto que es menos eficiente para redes más grandes, por lo que requiere más memoria y más tiempo de computación, mientras que la función *trainrp* es ideal para problemas de reconocimiento de patrones, utiliza poca memoria y es para redes grandes como en este caso [Beale *et al.*, 2010].

4.2.3 Funciones de activación

Para probar las funciones de activación se usaron los datos descritos en el Cuadro 4.4 con la variedad multiflora. Esta es la muestra de datos 100–TODO. La elección de la función de activación depende de la precisión y velocidad requerida, pero también del algoritmo de entrenamiento escogido. En las pruebas siguientes se omite utilizar la función *trainlm* debido a que dio los peores resultados en la pruebas de algoritmo de entrenamiento.

Cuadro 4.8 Porcentajes de aciertos de la red MLP en multiflora variando las funciones de entrenamiento y transferencia y dejando fijo el número de capas que serán dos.

Algoritmo de Entrenamiento	Función de Activación	Porcentaje
trainscg	tansig	97.43
trainscg	logsig	73.87
trainscg	purelin	75.44
trainrp	tansig	99.85
trainrp	logsig	99.91
trainrp	purelin	90.84

Todas las funciones de transferencia mejoraron sus resultados al utilizar la función *trainrp*. La función *purelin* es la menos eficiente, sin embargo, al utilizar la función *trainrp* dio el resultado más veloz con un tiempo de 1.0968e+003 segundos. La función *logsig* tuvo resultados más bajos que *tansig*. Mientras se utilice *trainscg*, *logsig* da resultados mucho más bajos que *tansig*, pero al cambiar a *trainrp*, *logsig* y *tansig* se emparejan y dan los resultados más altos. Tanto *logsig* como *purelin* dan los resultados más bajos usando *trainscg*. Estas mismas tienen los peores tiempos, para *logsig* las corridas tienen un tiempo de 5.5402e+003 segundos y *purelin* 4.3798e+003 segundos. Los resultados con *purelin* pueden deberse a que dicha función no limita adecuadamente el rango de salida de la neurona[2] y una función lineal es poco utilizada en tareas de clasificación de

[2]Nota: se debe tomar en cuenta que las pruebas realizadas involucran solo las capas ocultas, una red *newpr* debe tener como capa de salida la función no lineal *tansig*.

patrones. La funciones *logsig* y *tansig*, son las más usadas por ser diferenciables, y como se observó son similares. La función *tansig* es popular en redes multicapa y *logsig* tiene una de las derivadas más fáciles de computar.

4.3 PRUEBAS CON LOS PARÁMETROS DEL ALGORITMO PNN

Para esta sección se realizaron pruebas para mejorar la capacidad de clasificación de la red PNN. En las pruebas preliminares, las respuestas de la red *newpnn* dieron resultados más bajos que la red *newpr* tanto en la variedad multiflora como en la púrpura. Lo anterior puede visualizarse en el siguiente cuadro que muestra los porcentajes globales de aciertos para la red *newpnn* en la fase de entrenamiento, obtenidos de las pruebas iniciales.

Cuadro 4.9 Promedios de los porcentajes de aciertos para la red PNN en entrenamiento.

		Multiflora	Púrpura
100	LAB	78.97	95
100	LCH	58.54	94.83
100	RGB	76.91	94.86
100	TODO	78.28	94.97
50	LAB	79.32	94.9
50	LCH	58.87	94.78
50	RGB	76.98	94.81
50	TODO	78.11	94.87

Los aciertos son muy bajos en la variedad multiflora, en la variedad púrpura es aceptable. Por lo que se decidió repetir las pruebas del Cuadro 4.9 donde se obtuvieron los mejores y peores resultados para cada tratamiento. Las pruebas consistieron en la modificación del *spread*, parámetro que se debe ajustar para encontrar la respuesta óptima, en un intervalo de 0.01 a 5.0, ver Cuadro 4.10.

Utilizando un *spread* de 0.5 o menor el resultado es favorable, incluso en las muestras que originalmente dieron las peores respuestas. Es importante mencionar que los nuevos

resultados son concordantes con los originales en el sentido de que los conjuntos de datos con mejor respuesta también eran los de mejor respuesta en las nuevas pruebas, lo mismo ocurrió con las peores respuestas.

Cuadro 4.10 Se entrena la red modificando los valores del *spread* entre 0.01 y 5 utilizando las muestras que dan la mejor y peor respuesta de las pruebas preliminares. Se indican los promedios de aciertos de PNN.

Combinaciones donde se obtuvo la mejor respuesta										
Variedad	**Tamaño**	**Canal**	**0.01**	**0.10**	**0.20**	**0.30**	**0.50**	**1**	**3**	**5**
Multiflora	50	LAB	100	99.10	94.56	93.66	86.13	80.11	49.90	32.24
Púrpura	100	LAB	100	100	100	99.99	100	95.00	89.56	53.64
Combinaciones donde se obtuvo la peor respuesta										
Carpeta	**Tamaño**	**Canal**	**0.01**	**0.10**	**0.20**	**0.30**	**0.50**	**1**	**3**	**5**
Multiflora	100	LCH	97.84	90.77	86.93	81.01	79.42	58.45	40.48	31.63
Púrpura	50	LCH	100	100	99.92	99.92	96.86	94.78	53.95	53.49

A diferencia de la red MLP, la velocidad es mucho mayor en esta red y no se diferencia mucho por el tamaño del *spread*, por ejemplo en el tratamiento 4 con la combinación 50–LAB el tiempo de ejecución para las 25 corridas fue aproximadamente de 11 segundos para el menor y mayor valor del *spread*, i.e., 0.01 y 5. En el tratamiento 1 y combinación 100–LCH con un *spread* de 0.01 el tiempo es de 2 minutos y con un valor de 5 es de 2 minutos con 6 segundos. Por lo que el tamaño del *spread* no altera el tiempo que toma ejecutar el programa.

4.4 PRUEBAS COMPARATIVAS DE LOS ALGORITMOS MLP Y PNN

A continuación se presentan los resultados de un experimento de simulación para la comparación de los modelos MLP y PNN para determinar si alguno es superior. Se obtiene información del desempeño de las redes para dar indicios en la selección de éstas. Para la red MLP se utilizaron dos capas ocultas con 15 y 8 neuronas respectivamente, además se eligieron el algoritmo de aprendizaje elástico de retropropagación, y las funciones de activación *tansig* en las capas ocultas. En cuanto a la red PNN se utilizó un valor de 0.1 en el parámetro *spread* por dar buenos resultados

en pruebas anteriores. De estas pruebas se observa que los resultados en general aumentan emparejándose los porcentajes de aciertos en ambas redes; los valores más bajos se obtuvieron con las combinaciones 100–LCH y 50–LCH; se encontró que utilizando el 100 por ciento de los datos los resultados son prácticamente iguales utilizando cualquier combinación y variedad. Utilizando todos los modelos de color se pueden obtener los valores más altos, le sigue la combinación 50–LAB con el que se obtienen valores muy altos.

A continuación se ilustra el efecto del tamaño del conjunto de datos en las pruebas de los modelos MLP y PNN. Tomando como ejemplo los resultados en la fase de prueba en la variedad 1 para el modelo de color LCH para 20, 50 y 100 por ciento del conjunto de datos.

Cuadro 4.11 Promedios de los porcentajes de aciertos para multiflora, LCH y varios tamaños.

	100%	50%	20%
MLP	95.4	95.1	95.0
PNN	90.7	91.2	91.4

Como puede verse no hay variaciones significativas en el porcentaje de aciertos sin embargo el tiempo transcurrido empieza a reducirse a partir de 50% de clasificación, por ejemplo, para la red PNN el tiempo en segundos para los tamaños 100, 50 y 20 es 254.3, 63.4 y 13.1 respectivamente.

Respecto al número de neuronas en las capas ocultas en la red MLP se realizan nuevas comparaciones, esta vez utilizando el tamaño 50% y probando con una sola capa oculta debido a que ya se probó con varias capas anteriormente.

Cuadro 4.12 Promedios de los porcentajes de aciertos para la red MLP, primero con dos capas (15 y 8 neuronas ocultas) y luego variando el número de neuronas en una sola capa oculta a 20, 15, 10 y 5.

Variedad	Canal	[15,8]	20	15	10	5
Multiflora	LAB	99,82	99.86	99.64	99.41	96.11
Multiflora	LCH	95,78	95.06	94.36	93.78	92.28
Multiflora	RGB	99,71	99.59	99.46	98.75	95.72
Multiflora	TODO	99,73	99.81	99.74	99.41	97.36
Púrpura	LAB	100	100	100	100	98.36
Púrpura	LCH	100	99.95	99.92	99.91	99.88
Púrpura	RGB	100	100	100	100	100
Púrpura	TODO	100	100	100	100	99.99

Se reportó que el tiempo en segundo fue variable y no bajo gradualmente conforme disminuía el número de neuronas en todos los caso como se esperaría. Por lo tanto, la elección anterior de dos capas ocultas con 15 y 8 neuronas, fue tan buena elección como al usar una capa con 20 neuronas ocultas.

En general, los mejores resultados se obtuvieron utilizando todos los modelos de color TODO, sin embargo, se obtuvieron resultados casi tan aceptables utilizando los modelos de color CIE Lab y RGB.

Cuadro 4.13 Promedios de los porcentajes de aciertos y desviación estándar para las redes MLP y PNN utilizando 50% de los datos y un solo espacio de color.

Variedad	Canales	MLP%E	MLP%P	PNN%E	PNN%P
Multiflora	CIE Lab	99.8+/-0.3	99.3+/-0.4	99.1+/-0.1	98.9+/-0.5
Multiflora	LCH	95.8+/-1.6	95.1+/-1.7	90.9+/-0.2	91.2+/-1.1
Multiflora	RGB	99.7+/-0.3	99.4+/-0.4	99.0+/-0.1	98.7+/-0.4
Púrpura	CIE Lab	100+/-0.0	100+/-0.0	100+/-0.0	100+/-0.0
Púrpura	LCH	100+/-0.0	100+/-0.1	100+/-0.0	100+/-0.1
Púrpura	RGB	100+/-0.0	100+/-0.0	100+/-0.0	100+/-0.0

Tratándose de la red MLP puede examinarse el entrenamiento en cuanto a los resultados obtenidos en el error cuadrático medio y la magnitud del gradiente, ver Cuadro 4.14.

Cuadro 4.14 Subconjuntos con tamaño 50% y modelo CIE Lab para la red MLP en fase de entrenamiento.

Variedad	Gradiente	Desempeño	Época
Multiflora	8.8071e-004	3.9880e-004	1000/1000
Púrpura	1.8179e-011	2.5485e-012	14/1000

Por otro lado, a pesar de que se obtienen resultados muy parecidos, al grado de no ver una ventaja significativa de una red sobre otra, el panorama cambia al revisar los tiempos de ejecución de las simulaciones los cuales se comparan en el Cuadro 4.15 donde se muestra que la red PNN es más veloz que la MLP.

Cuadro 4.15 Tiempos de ejecución en segundos de las simulaciones para ambas redes.

Variedad	MLP	PNN
Multiflora	10950.99	66.59
Púrpura	68.03	21.81

Para la variedad púrpura se obtiene el 100% de clasificación correcta. Para la variedad multiflora, se obtienen porcentajes de clasificación muy altos pero sin llegar al 100%. En la variedad multiflora los resultados son parecidos para RGB y LAB y más bajos para LCH. Por lo tanto, se eligen las combinaciones 50-LAB y 50-LCH en la variedad 1, debido a que la primera ha dado buenos resultados y la segunda ha dado los peores resultados, ambos conjuntos son de 2750 elementos en total siendo el 20% utilizado para entrenamiento, o 550 elementos, para una mejor comparación del funcionamiento de cada modelo de red a través de gráficas de confusion.

A continuación se muestran las matrices de confusion en la fase de prueba del conjunto BASE1–050–LAB y BASE1–050–LCH Figura 4.1, de la última corrida de los algoritmos MLP y PNN. En verde se presenta la cantidad de patrones clasificados correctamente y en rojo se observan los que resultaron mal clasificados.

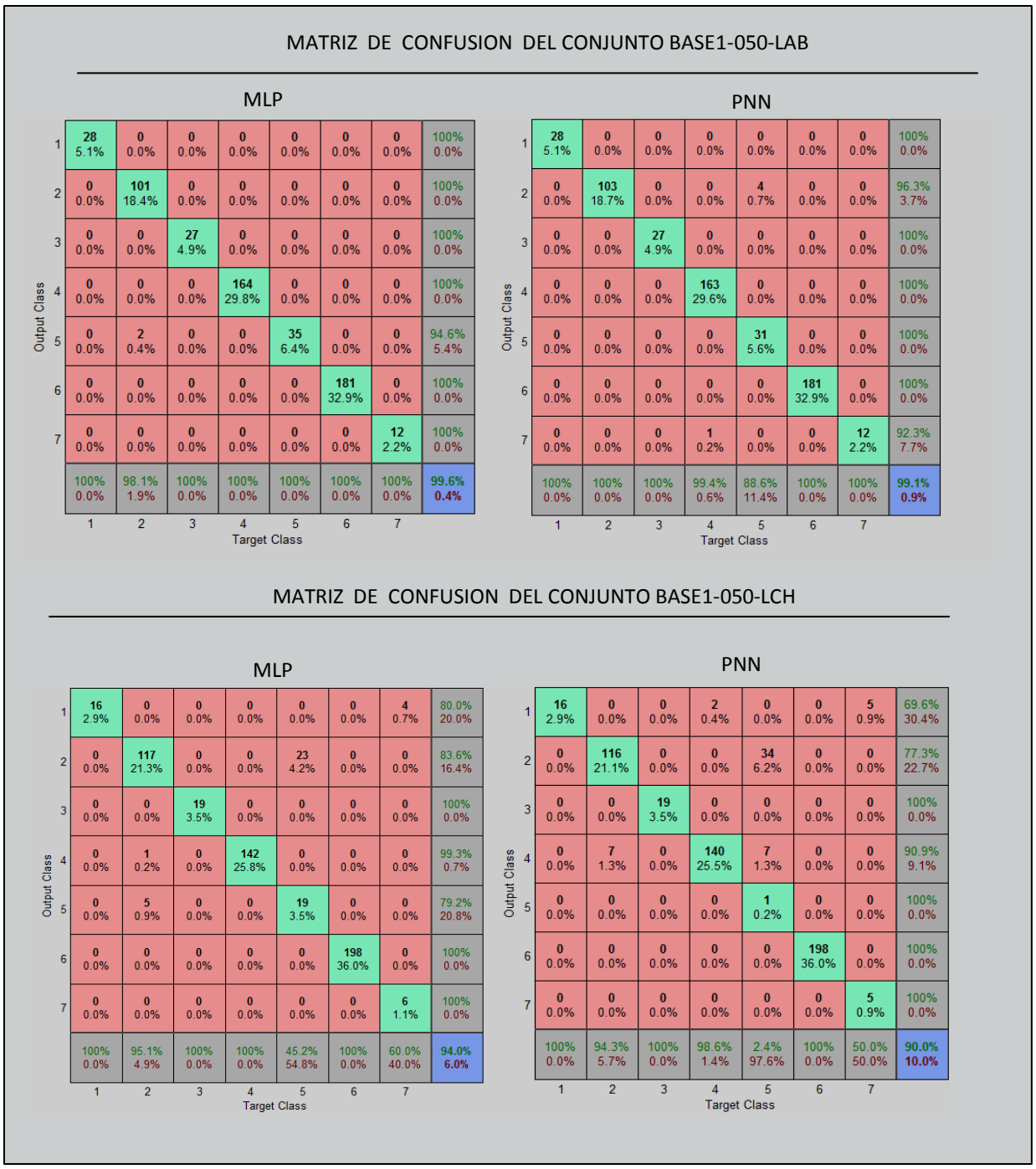


Figura 4.1 Comparación mediante matrices de confusion para los juegos de datos: BASE1–050–LAB y BASE1–050–LCH, en la prueba de las redes MLP y PNN.

Otra forma de analizar los porcentaje de clasificación realizada por la red es a través de la curva ROC obtenida a partir de información proporcionada por el diagrama confusión. Figura 4.2.

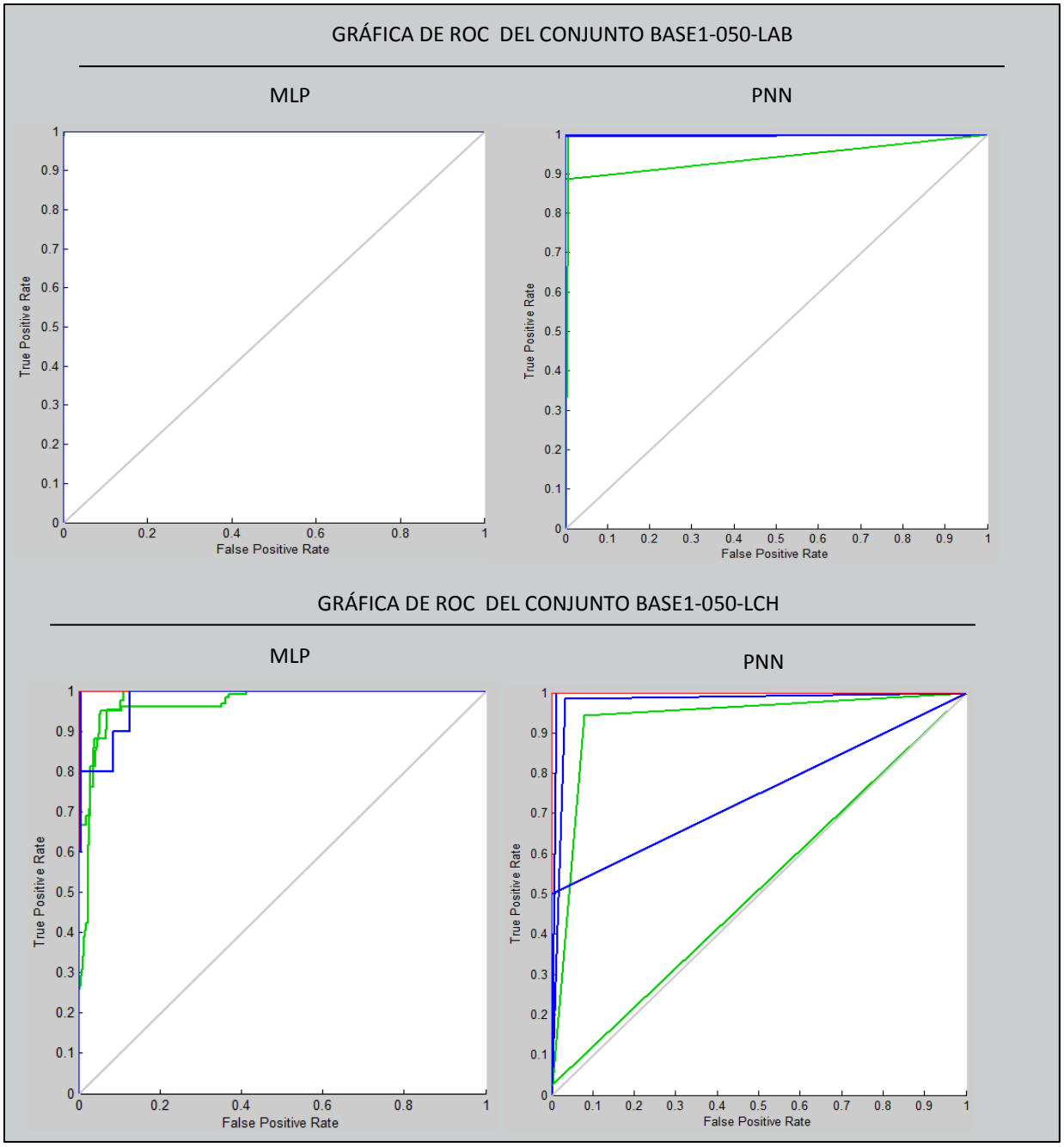


Figura 4.2 Comparación mediante la curva ROC para los juegos de datos: BASE1–050–LAB y BASE1–050–LCH, en la prueba de las redes MLP y PNN.

La gráfica de confusión, Figura 4.1, muestra el desempeño de las redes para la corrida número 25. Para el conjunto BASE1–050–LAB en la red MLP (gráfica superior izquierda) se ve claramente que de un total de 103 pixeles de la clase 5, 2 muestras fueron mal

clasificadas y 101 bien clasificadas, además las otras clase fueron completamente bien clasificadas. En cuanto a la red PNN (gráfica superior derecha) solo las clases 4 y 5 tuvieron algunos errores. Para el conjunto BASE1–050–LCH, a diferencia del conjunto anterior cuyas clasificaciones superan el 99%, la clasificación es buena en el 94% de los casos en la red MLP y en el 90% de los casos en la red PNN. La MLP y PNN coinciden al tener el mayor número de errores al clasificar la clase 5 como si fuera la clase 2; de un total de 42 elementos de la clase 5, en la MLP 23 muestras fueron clasificadas como pertenecientes a la clase 2 y en la red PNN 34 muestras fueron clasificadas como pertenecientes a la clase 2. Se deduce que para las dos redes (en especial la PNN) no logran identificar correctamente entre las clases 2 y 5 al ser clases parecidas. Aun siendo el conjunto que da los peores resultados, la clasificación es relativamente buena.

En la curva ROC, Figura 4.2, para el conjunto BASE1–050–LAB el área bajo la curva supera el valor 0.5 lo cual indica que la red es buen clasificador. Tratandose del conjunto BASE1–050–LCH la red MLP al tener un área bajo la curva superior al 0.5 presenta buen nivel de clasificación correcta, sin embargo al compararla con la red PNN para este mismo conjunto, se observa un contraste signifcativo al observar curvas alejadas de la unidad, siendo una de ellas muy cercana a 0.5.

Se observa además, que en cada gráfica ROC hay una curva perteneciente a alguna clase cerca del punto (0,1), i.e. del vértice superior izquierdo, lo cual es la situación ideal donde se tiene mucha sensibilidad y mucha especificidad. Aunque no sucede para todas las gráficas, sobre todo en el conjunto BASE1–050–LCH, tenemos en general una buena clasificación.

CONCLUSIONES

En esta investigación se aplicó el paradigma de las redes neuronales artificiales (ANN) para clasificar imágenes digitales de dos variedades de flores de *Petunia* spp con base en color. Los modelos estudiados fueron el perceptrón multicapa (MLP) y la red neuronal probabilística (PNN) con funciones de base radial Gaussianas (PNN). El propósito fue evaluar su desempeño bajo diferentes escenarios de modelación (tamaño de muestra, número de entradas con base en espacios de color, número de categorías de clasificación en función de las variedades de flor, entre otros).

Respecto a los requerimientos de construcción del modelo, PNN supera a MLP, debido a que ésta última requiere varias pruebas para elegir los parámetros que optimizan su funcionamiento (funciones de activación, número de capas y neuronas, algoritmos de entrenamiento). En cambio, las PNN tienen una construcción sencilla de implementar y sólo es necesario modificar el parámetro de suavizado *spread* de la función Gaussiana para encontrar su configuración óptima.

Los resultados indican que existen diferencias en la eficiencia de clasificación de las dos variedades de *Petunia* spp. Se obtuvo un mejor porcentaje de clasificación global en la variedad púrpura. En cuanto al tamaño y espacio de color no existen diferencias significativas. Con el escenario que utiliza 100% de los datos y como entradas, todos los espacios de color (RGB, CIE LAB y LCH) se obtienen mejores resultados; sin embargo, el escenario 2 que utiliza 50% de los datos y como entradas el espacio de color CIE Lab (tres canales de color) los porcentajes de clasificación correcta son muy similares al escenario 1; por lo cual, es más apropiado utilizar escenario 2 por razones de eficiencia computacional.

Para comparar las medidas de desempeño de los dos clasificadores propuestos, se utilizaron las gráficas de confusión y curvas ROC. Se encontró que los resultados son exactos para la variedad púrpura obteniendo generalmente 100% de clasificación correcta; mientras que en la variedad multiflora, las redes tienen casos de clasificación

incorrecta al confundir ocasionalmente algunas clases debido a la similitud entre clases con esta variedad, llegando a obtener resultados óptimos sin llegar a una clasificación exacta pero con buenos resultados para ambas redes, se obtienen los más altos los porcentajes de clasificación en la prueba en el conjunto BASE1-050-LAB con 99.3% en la red MLP y 98.9% en la red PNN. En promedio, el modelo MLP obtuvo un 98.4% de clasificación correcta en contraposición con la red PNN con un 97.1% de clasificación correcta.

Globalmente, para las imágenes digitales analizadas, los dos modelos de red neuronales presentaron buen desempeño en la clasificación de imágenes de flores de *Petunia* spp con base en color. Sin embargo se recomienda para este tipo de problemas de clasificación, el modelo PNN utilizando únicamente el espacio de color CIE Lab (tres entradas) y diferentes categorías de color por su facilidad para implementarse y rapidez en los tiempos de ejecución.

BIBLIOGRAGÍA

Adeli, H., A. Panakkat. 2009. A probabilistic neural network for earthquake magnitude prediction. ScienceDirect.22:1018-1024.

Aitkenhead, M. J., I.A. Dalgetty, C.E. Mullins, A.J.S. McDonald, N.J.C. Strachan. 2003. Weed and crop discrimination using image analysis and artificial intelligence methods. ELSEVIER, Science Direct, Computer and Electronics in Agriculture. 39:157-171.

Alegre G., E., L., Sánchez G., R. A., Fernández D. 2003. Procesamiento Digital de Imagen. Fundamentos y Prácticas con Matlab. Universidad de León, Secretariado de publicaciones y Medios Audiovisuales.

Beale M. H., M. T. Hagan, H.B. Demuth. 2010. Neural network toolbox. User's guide. Matlab version 7.951p.

Chung F., J. K., S., Chung Y., H., Cheng E., P., Kai C. 2010. Application of computer vision in theautomatic identification and classification of woven fabric weave patterns.Textile Research Journal 80(20):2144-2157.

Corchado, J. M. 2000. Redes neuronales artificiales: un enfoque práctico. Universidad de Vigo, Servicio de Publicaciones, España. 207p.

Gang W., S., F., Sheng B., E., You X., Y., Wang, Y., Chang, and Q., Xiang. 2007. A leaf recognition algorithm for plant classification using probabilistic neural network. arXiv.1:1-6.

Guili, X., Fengling Z., S., Ghafoor S., Yongqiang Y., Hanping, M. 2011. Use of leaf color images to identify nitrogen and potassium deficient tomatoes. ELSEVIER, Pattern Recognition Letters. 32:1584-1590.

Guru D., S., Y.H., Sharath K., y S., Manjunath.2010.Textural features in flower classification. ELSEVIER, Mathematical and Computer Modelling.54:1030-1036.

Haykin, S.S. 1998. Neural networks: a comprehensive foundation. 2nd Ed. New Jersey, Prentice Hall, 846p.

Iñigo M., R. y J.M., Angulo U.1986. Visión Artificial por Computador. Fundamentos, Sistemas y Aplicaciones en la Industria y la Robótica. Paraninfo, Madrid España, 292p.

Isasi V., P. e I. M., Galván L. 2004. Redes de neuronas artificiales. Un enfoque práctico. Prentice Hall Pearson Educación, S.A. Madrid, España.

Mendoza, F., P., Dejmek, and J. M., Aguilera. 2006. Calibrated color measurements of agricultural foodsusing image analysis. ELSEVIER, Science Direct, Postharvest biology and technology. 41:285-295.

Nason, A. 1993. Biología. Editorial Limusa S.A. de C.V. Grupo Noriega editores. D.F., México. 726p.

Pajares M., G., J. M., De la Cruz G., J. M., Molina P., J., Cuadrado P., A., López C. 2004. Imágenes Digitales. Procesamiento práctico con Java. Alfaomega Grupo Editor, S.A. de C.V. D.F., México. 193p.

Pajares M., G. y J.M., De la Cruz G. 2008. Visión por Computador. Imágenes Digitales y Aplicaciones. 2ª. ed. Alfaomega Grupo Editor, México. 768p.

Palma M., J. T., R., Marín M. y C., Alonso G. 2008. Inteligencia artificial: métodos, técnicas y aplicaciones. McGraw-Hill Interamericana de España. Madrid, España. 1022p.

Parthiban, L., Subramanian R. 2009. CANFIS—a computer aided diagnostic tool for cancer detection. Journal Biomedical Science and Engineering. 2:323-335.

Rasekhi, R., V. Asadi, A. Jafari.2010. Weeds and corn classification by image processing and neural network techniques. Internatıonal Journal of Natural and Engineering Sciences. 4(2): 41-46.

Russell, S. y P., Norvig. 1996. Inteligencia artificial. Un enfoque moderno. Prentice-Hall Hispanoamérica S.A. Estado de México.

Sánchez C., E. y A. Y., Alanís G. 2006. Redes neuronales. Conceptos fundamentales y aplicaciones a control automático. Prentice Hall Pearson Educación, S.A. Madrid, España.

Shah R., M. S. B., A.R., Farah Y., M. Y., Ahmad I., and K. Shazana.2009. Non-destructive watermelon ripeness determination using image processing and artificial neural network (ANN). Proceedings of world academy of science, engineering and technology.38:542-546.

Specht, D. F.1988. Probabilistic Neural Networks. Pergamon Press plc. 3:109-118.

Timmermans A., J. M., y A.A., Huizebosch.1995. Computer vision system for on-line sorting of pot plants using an artificial neural network classifier. ELSEVIER, Science Direct, Computers and Electronics in Agriculture.15:41-55.

Vesali F., M. Gharibkhani, M. H. Komarizadeh. 2009. An approach to estimate moisture content of apple with image processing method. Australian journal of crop science. 5(2):111-115.

Yu, Y., S., Zhang, H. Zhang, X., Liu, Q., Zhang, X., Zheng, J., Dai .2010. Neural decoding based on probabilistic neural network.Journal of Zhejiang University-SCIENCE B (Biomedicine & Biotechnology). 11(4):298-306.

Zhang J., S. Sokhansanj, S. Wu, R. Fang, W. Yang, and P. Winter. 1997. A transformation technique from RGB signals to the Munsell system for color analysis of tobacco leaves. ELSEVIER, Computers and Electronics in Agriculture.16:231-244.

ANEXO A CÓDIGO FUENTE

A.1 Introducción

En esta sección se presentan los programas en lenguaje de programación Matlab utilizados para crear los repositorios de datos utilizados para armar los juegos de datos para las pruebas y funciones para crear, configurar y entrenar redes neuronales en la presente investigación.
Las líneas de código precedidas del signo porciento '%' son comentarios acerca de cierta declaración, no representan instrucciones de código.

A.2 Generación de los datos

Como se utilizan 4 carpetas que contienen imágenes creadas de recortes de 10 x 10 pixeles ordenados por colores, se utilizan tres programas Generar1, Generar2 y Generar3 para procesar los pixeles, debido a que para cada carpeta con una composición determinada de una variedad cambian el número de pixeles, el número de agrupaciones de pixeles y el número de clases. A continuación se muestra a modo de ejemplo uno de los tres programas "Generar1" que utiliza la carpeta FRAG1-100 en la carpeta de Matlab, para generar el conjunto de datos: vector de entrada y vector objetivo.

```
%-----------------------------------------------------------------------
% Con este programa se generaron los espacios de color
%-----------------------------------------------------------------------
% DEL RESULTADO GUARDO E Y Z EN CARPETA BASE1
clc;
clear;
k=0;
mts = 12400;%delimita la longitud al número de pixeles de la carpeta
for m=1:124 % se repite el ciclo for para cada foto
   %leer una carpeta con imágenes de una composición:
   %variedad, separar cada pixel en los canales RGB
   mue= imread(['FRAG1-100/b' int2str(m) '.jpg']);
   R = mue( :, :,1);
   G = mue( :, :,2);
   B = mue( :, :,3);
   %cambiar a tipo de datos real o double
   R= double(R);
   G=double(G);
   B=double(B);
   %Convertir imagen RGB al modelo CIELab
   cform = makecform('srgb2lab');
   lab_flor = applycform(mue,cform);
   %convierto lab del tipo de datos uint8 a double
```

```
   do_lab_flor = double(lab_flor);
   do_lab_flor(:,:,1) = 100 * (do_lab_flor(:,:,1) / 255);
   do_lab_flor(:,:,2) = do_lab_flor(:,:,2) - 128;
   do_lab_flor(:,:,3) = do_lab_flor(:,:,3) - 128;
   L = do_lab_flor(:,:,1);
   a = do_lab_flor(:,:,2);
   b = do_lab_flor(:,:,3);
   %Convertir imagen directamente del modelo CIE Lab al modelo lch
   C = makecform('lab2lch');
   lch_flor = applycform(do_lab_flor,C);
   croma=lch_flor(:,:,2);
   hue=lch_flor(:,:,3);
   %Este for es diferente en cada programa depende del número de
   %elementos de la carpeta y el número de clases. Asigna clases
   %para cada uno de los pixeles en la imagen en turno.
  for i=1:10
     for j=1:10
         k=k+1;
         if (k <= 1200)   clase=1;
           elseif (k > 1200) && (k<= 3600) clase=2;
             elseif (k > 3600) && (k<= 4000) clase=3;
               elseif (k > 4000) && (k<= 6400) clase=4;
                 elseif (k > 6400) && (k<= 6800) clase=5;
                   elseif (k > 6800) && (k<= 9600) clase=6;
                     elseif (k > 9600) && (k<= 12400) clase=7;
         end
% Se genera una matriz T de forma que cada canal de color en
% turno se acomode en su correspondiente columna y una última
% columna con la clase dada.
         T(k,1)=R(i,j);
         T(k,2)=G(i,j);
         T(k,3)=B(i,j);
         T(k,4)=L(i,j);
         T(k,5)=a(i,j);
         T(k,6)=b(i,j);
         T(k,7)=croma(i,j);
         T(k,8)=hue(i,j);
         T(k,9)= clase;
      end
  end
end
TP= T'; % Se realiza la transpuesta de T
%Z es la última hilera la CLASE  de c/columna
% o bien, el VECTOR OBJETIVO
i=1;
for j=1:mts
    Z(i,j) = TP(9,j);
end
%E es el VECTOR DE ENTRADA tomado de la traspuesta de T
for i=1:8
    for j=1:mts
        E(i,j)=TP(i,j);
    end
end
```

A.3 Reducir datos

De los programas anteriores se guardan los vectores de entrada E y objetivo Z del espacio de trabajo y serán utilizados para ordenarlos en algunas columnas en Excel, es decir se crea una hoja Excel y añadimos columnas correspondientes a: pixel, foto, muestra, submuestra, canales de color y clases, de forma que obtengamos información acerca de la posición de los pixeles. El último paso guarda los datos de cada variedad en archivos con extensión .mat: datafrag1, datafrag2, datafrag3, datafrag4. Estos archivos son procesados en el siguiente programa para reducir registros de los datos que queden repetidos.

```
%-------------------------------------------------------------------
% Reducción de datos repetidos
%-------------------------------------------------------------------
 % Matriz RGB contiene filas ordenadas por R G B
 % Permite identificar hileras repetidas
  RGB = datafrag1;
 j=1;
 for k=1:13
     RED(j,k) = RGB(j,k);
 end
 for i = 1:12399 %número de pixeles contenidos en el archivo
   if RED(j,6) == RGB(i+1,6)
       if RED(j,7) == RGB(i+1,7)
            if RED(j,8) == RGB(i+1,8)
              continue;
            end
       end
   end
   j=j+1;
   for k=1:13
      RED(j,k) = RGB(i+1,k);
   end
 end
%Obtenemos la matriz con los datos reducidos, no repetidos en RED
 DATAOK = RED; %Hacemos DATAOK para luego quitar datos extras
 DATAOK(:,1:4)=[]; %Se eliminaron los datos extras(pixel, foto,
                   %muestra, submuestra)
 XOK= DATAOK; %Variable XOK obtener vector de entrada
 XOK(:,9)=[]; %Es E
 YOK= DATAOK; %Variable YOK (sin datos extras)
 YOK(:,1:8)= []; %Es Z
  %SE HAN ELIMINADO LAS COLUMNAS DE DATOS COMPLEMENTARIOS DEJANDO LOS
VECTORES DE ENTRADA Y OBJETIVO PARA CADA VARIEDAD
```

A.4 Generar vectores E y Z según composición y combinación

Para cada variedad los vectores de entrada y objetivo de la sección anterior se guardan en archivos .mat llamados: EZfrag1, Ezfrag2, EZfrag3, EZfrag4. El siguiente paso es crear dentro de la carpeta Matlab las carpetas generales y subcarpetas que por lo pronto estarán vacías para depositar los juegos de datos correspondientes, por medio de los programas que se presentan adelante.
El resultado del siguiente programa es obtener para cada subcarpeta una matriz de entrada y su vector objetivo, de una determinada composición: variedad-tratamiento y combinación: modelo de color y tamaño.

```
%-------------------------------------------------------------------
% Generando archivos para distribuir en subcarpetas
%-------------------------------------------------------------------
clear;
clc;
% Cadena de letras que corresponde a la dirección de la carpeta
% elegida. La dirección se modifica y se corre el programa
cadena = 'BASEFRAG2/BASE2-050-TODO/EZ.mat';
% Las variantes son porcentaje del conjunto y modelos de color
% Buscar en la cadena el número para saber de qué carpeta procede y
% saber que archivo cargar
 for l=['1','2','3','4'];
     if cadena(9) == l
         s = str2double(l);
         load(['EZfrag' int2str(s) '.mat']);
     end
 end

 % Buscar en la cadena la referencia al tamaño elegido para esa
 % carpeta

 cad=cadena(17:19);
 switch cad
        case '050'
            porciento = 0.50;
        case '100'
            porciento = 1.00;
 end
% Buscar en la cadena la posición que hace referencia al modelo
% elegido para la carpeta
modelo = cadena(21:23);
% Formar agrupaciones de elementos de entrada que coinciden en
% la clase
Set = EZ;
tam = length(EZ);
i=1;
j=1;
k=1;
l=1;
```

```
o=1;
p=1;
q=1;

   for m=1:tam
               if(Set(9,m)==1)
                         for n=1:9
                               A(n,i)=Set(n,m);
                         end
                         i = i + 1;
               elseif(Set(9,m)==2)
                         for n=1:9
                               B(n,j)=Set(n,m);
                         end
                          j = j + 1;
               elseif(Set(9,m)==3)
                         for n=1:9
                               C(n,k)=Set(n,m);
                         end
                          k = k + 1;
               elseif(Set(9,m)==4)
                         for n=1:9
                               D(n,l)=Set(n,m);
                         end
                          l = l + 1;
               elseif(Set(9,m)==5)
                         for n=1:9
                               F(n,o)=Set(n,m);
                         end
                          o = o + 1;
               elseif(Set(9,m)==6)
                         for n=1:9
                               G(n,p)=Set(n,m);
                         end
                          p = p + 1;
               elseif(Set(9,m)==7)
                         for n=1:9
                               H(n,q)=Set(n,m);
                         end
                          q = q + 1;
               end

   end

% Etiquetamos las clases en letras que son las agrupaciones
% para poder manejarlas posteriormente
  if ((cadena(9) == '1' ) || (cadena(9) == '2') )
        letras = ['A','B','C','D','F','G','H'];
  elseif ((cadena(9) == '3') || (cadena(9) == '4'))
       letras = ['A','B','C'];
  end

for dato = letras;

    switch dato
        case 'A'
```

```
            long = size(A);
        case 'B'
            long = size(B);
        case 'C'
            long = size(C);
        case 'D'
            long = size(D);
        case 'F'
            long = size(F);
        case 'G'
            long = size(G);
        case 'H'
            long = size(H);
    end
    long1=long(1,2); % va obteniendo la longitud de cada agrupación
    v=randperm(long1); %barajea los elementos por medio de subíndices
    long2 = porciento * long1;  %Longitud del grupo según el tamaño
    for i=1:9 %Vuelve a armar los grupos ya barajados
        for j=1:long2
            k=v(j);
            switch dato
                    case 'A'
                      M(i,j)=A(i,k);
                    case 'B'
                      N(i,j)=B(i,k);
                    case 'C'
                      P(i,j)=C(i,k);
                    case 'D'
                      Q(i,j)=D(i,k);
                    case 'F'
                      R(i,j)=F(i,k);
                    case 'G'
                      S(i,j)=G(i,k);
                    case 'H'
                      T(i,j)=H(i,k);

            end
        end
    end


end

% Reune nuevamente las agrupaciones en la población de entrada según
% la carpeta
if ((cadena(9) == '1' ) || (cadena(9) == '2') )
     U = [M N P Q R S T];
elseif ((cadena(9) == '3') || (cadena(9) == '4'))
       U = [M N P];
end

% Obtiene longitud de la entrada, baraja la entrada
long=size(U);
long1=long(1,2);
v=randperm(long1);
```

```
 for i=1:9
    for j=1:long1
       k=v(j);
       V(i,j)=U(i,k);
    end
 end


 % Redondea longitud para poder dividir en grupos de 60% 20% y 20%
 % en el siguiente programa

 long3=long1;
 i=1;
 if ( mod (long1,10) ~= 0)
       long1 = fix(long1/10)* 10;
 end

 %Z es de la transpuesta o sea, la última hilera la CLASE  de
 %c/columna o bien, el VECTOR OBJETIVO

for j=1:long1
    Z(i,j) = V(9,j);
end
% Finalmente vector de ENTRADA
for i=1:8
    for j=1:long1
        E(i,j)= V(i,j);
    end
end

% Elimina columnas que no correspondan al modelo elegido
switch modelo
         case 'RGB'
             E(4:8,:) = [];

         case 'LCH'
             E(1:6,:) = [];

         case 'LAB'
             E(1:3,:) = [];
             E(4:5,:) = [];
end

% Guarda en la subcarpeta correspondiente los vectores E y Z
savefile = (cadena);
save(savefile,'E','Z');
```

A.5 Juegos de datos para 25 ciclos del algoritmo

El siguiente programa sirve para crear un archivo más dentro de cada subcarpeta, conteniendo subíndices que serán utilizarlos posteriormente para determinar los juegos de datos elegidos para cada uno de los 25 ciclos del algoritmo de ANN.

```
%-----------------------------------------------------------------------
%Obtener archivos EZjuegos.mat para distribuir en subcarpetas
%-----------------------------------------------------------------------
clear;
clc;
% Se asignan a cadenas nombres de variables para posterior uso
cad1 = 'BASEFRAG1/BASE1-020-RGB/'; % esta cadena se modifica según la
% muestra o subcarpeta que se este utilizando
cad2 = 'EZ.mat';
cad3 = 'EZjuegos.mat';
cad4=[cad1 cad2];
cad5 = [cad1 cad3];
load(cad4); % Descarga la subcarpeta que se creo en el programa
anterior
p = 1;
tam = length(E); %Longitud del vector de entrada
p1 = tam*0.6;  %Longitud del 60 por ciento de la entrada
p2 = tam*0.2;  %Longitud del 20 por ciento de la entrad

% El ciclo while genera un vector de números que serán colocados en
una
% posición a la que se le asigna un 1,2 o 3 referente  a los
% subconjuntos
while (p < 26)

    %Utilizando la función Random data division (dividerand)
    %Divide los datos en tres subconjuntos los que llevan P
(entrenamiento)
    % los T (objetivos) y los Ind (índices)
    [trainP,valP,testP,trainInd,valInd,testInd] = dividerand(E);
    % Función divideind genera indices para entrenamiento, prueba
    % y validación
    [trainT,valT,testT] = divideind(Z,trainInd,valInd,testInd);

    j=1;
      %construyendo el vector de indices
      for q= 1:p1
          lin(p,j) = trainInd(1,q);
          x= lin(p,j);
          c(p,x)=1;
          j=j+1;
      end
      for q =1:p2
          lin(p,j) = valInd(1,q);
          x= lin(p,j);
```

```
            c(p,x)=2;
            j=j+1;
        end
        for q= 1:p2
            lin(p,j) = testInd(1,q);
            x= lin(p,j);
            c(p,x)=3;
            j=j+1;
        end

      p = p+1;

end
% al finar se obtiene un vector de índices que se guarda como un
% archivo nuevo ".mat" junto con el vector de entrada y objetivo en
% su correspondiente subcarpeta
juegos= c';
savefile = (cad5);
save(savefile,'E','Z','juegos');
```

A.6 Porcentajes de Aciertos de las Redes Neuronales

El siguiente programa realiza la creación, simulación y despliega en pantalla los porcentajes de clasificación de las redes neuronales *newpr* y *newpnn*.

```
%-----------------------------------------------------------------------
%Redes neuronales Perceptrón Multicapa y Red Neuronal Probabilística
%-----------------------------------------------------------------------
clear;
clc;
l=1;
tiempo = 0;
% Esta cadena se modifica según la subcarpeta elegida para
% entrenamiento
cadena = 'BASEFRAG1/BASE1-050-TODO/EZjuegos.mat';
% Usando la dirección se sabe cuántas clases se usaran en las redes
if ((cadena(9) == '1' ) || (cadena(9) == '2') )
       clas = 7;
elseif ((cadena(9) == '3') || (cadena(9) == '4'))
       clas = 3;
end

% Usando la dirección de subcarpeta sabemos el modelo de color
switch cadena(21:23)
    case 'TOD'
        bandas = 8;
    case 'RGB'
        bandas = 3;
    case 'LAB'
        bandas = 3;
```

```
    case 'LCH'
        bandas = 2;
end

s = RandStream('mcg16807', 'Seed', 192736547);
RandStream.setDefaultStream(s);
%Descargar la subcarpeta
load(cadena);
mst = length(E);  %mst es longitud de la entrada
% El ciclo while sirve para obtener 25 ciclos para entrenamiento y  %
prueba de los modelos de ANN
while (l < 26)

des(l,1) = l;
j=1;
m=1;
n=1;
% El ciclo for genera los subconjuntos de entrenamiento y prueba
% basándose en los índices creados en el programa anterior
for k = 1:mst
    if juegos(k,l) == 1
        for i = 1:bandas
            entrenaE(i,j) = E(i,k);
        end
            entrenaT(1,j) = Z(1,k);
        j = j + 1 ;
    elseif juegos(k,l) == 2
        for i = 1:bandas
            validaE(i,m) = E(i,k);
        end
            validaT(1,m) = Z(1,k);
        m = m + 1;
    else
        for i =1:bandas
            prueE(i,n)= E(i,k);
        end
            prueT(1,n) = Z(1,k);
        n = n + 1;
    end
end
% Se generaron los vectores de entrada y objetivo para entrenamiento
entrada=[entrenaE validaE];
objetivo =[entrenaT validaT];
T = ind2vec(objetivo);
T = full(T);
%RED PERCEPTRON MULTICAPA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[XTn,es] = mapminmax(entrada); % Normaliza la entrada

num_capas = [15 8];     %Elección del número de capas y neuronas
% Opcionales
%num_capas = [25 17 8];
%num_capas = [120 60 30 15];
%num_capas = [ 154 64 40 25 16];

activacion = {'tansig' 'tansig' 'tansig'}; % Elección de las
% funciones de activación en las capas ocultas
```

```
% Opcionales
% activacion = {'logsig' 'logsig' 'logsig'};
% activacion = {'purelin' 'purelin' 'purelin'};

%Creación de la red perceptron multicapa MLP usando la función newpr
red = newpr(XTn, T, num_capas, activacion);

% Diferentes elecciones del algoritmo de aprendizaje para la red newpr
red.trainFcn = 'trainrp';
% Opcionales
%red.trainFcn = 'trainlm';
%red.trainFcn = 'trainscg';
%red.trainFcn = 'trainbr';

red.divideFcn = '';  % Impide la subdivisión de datos de la función
newpr

[red, dome]= train(red,XTn,T); %ENTRENAR LA RED NEWPR
% (1) SIMULAR CON ENTRENAMIENTO Y VALIDACIÓN-----------------------
Yp = sim(red,XTn); % Simular la red newpr previamente entrenada

% Utilizamos la función confusion para obtener la suma de
coincidencias
% entre el vector objetivo y el vector de salida para entrenamiento
[c, cm, ind, per] = confusion (T, Yp);
sum = 0;
for i = 1:clas
    for j = 1:clas
        if (i==j)
            num = cm(i,j);
            sum = sum + num;
        end
    end
end
aciertos = (sum/length(objetivo)) * 100;
des(l,2) = aciertos; %Porcentaje de aciertos guardados en un vector
% (2) SIMULAR CON PRUEBA-------------------------------------------
%Da un vector con ceros excepto en los lugares correctos
T2 = ind2vec(prueT);
T2 = full(T2);
XTp =mapminmax('apply', prueE, es); % Normaliza la entrada con nuevos
datos
Yp2 =sim(red, XTp); %Simula con datos nuevos de prueba
% Utilizamos la función confusion para obtener la suma de
coincidencias
% entre el vector objetivo y el vector de salida de la prueba
[c, cm, ind, per] = confusion (T2, Yp2); %%T
sum = 0;
for i = 1:clas
    for j = 1:clas
        if (i==j)
            num = cm(i,j);
            sum = sum + num;
        end
    end
end
```

```
aciertos = (sum/length(prueT)) * 100;
des(1,3) = aciertos;
%% RED PROBABILISTICA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Utilizando la media de las distancias en los datos de entrada para
el spread
%Opcional
%M = XTn;% maize data
%M1 = (mst * 0.8)-1; %pero lo saco de E en vez de entrada
%for j = 1:M1
% distancia(j) = norm(M(:,j) - M(:,j+1));
%end
%media = mean(distancia)
%spread= media;
spread= 0.3;
% Crear la red neuronal probabilística
redprob = newpnn(XTn,T,spread);
%% (3) SIMULAR CON ENTRENAMIENTO Y VALIDACIÓN-------------------------
-----
%% Probamos la red para el vector de entrada diseñado
Yc = sim(redprob,XTn);
YcV = full(Yc);
% Utilizamos la función confusion para obtener la suma de            %
coincidencias entre el vector objetivo y el vector de salida del
% entrenamiento
[c, cm, ind, per] = confusion (T, YcV);
sum = 0;
for i = 1:clas
    for j = 1:clas
        if (i==j)
            num = cm(i,j);
            sum = sum + num;
        end
    end
end
aciertos = (sum/length(objetivo)) * 100;
des(1,4) = aciertos;
 %% (4) SIMULAR CON PRUEBA----------------------------------------
%% Normalización de la entrada
% Transformación de entradas y salidas intervalo -1,1
Yc2= mapminmax('apply', prueE, es);
Yc2=sim(redprob, Yc2);
Yc2V = full(Yc2);
% Utilizamos la función confusion para obtener la suma de
% coincidencias entre el vector objetivo y el vector de salida de la
% prueba
[c, cm, ind, per] = confusion (T2, Yc2V);
sum = 0;
for i = 1:clas
    for j = 1:clas
        if (i==j)
            num = cm(i,j);
            sum = sum + num;
        end
    end
end
aciertos = (sum/length(prueT)) * 100;
des(1,5) = aciertos;
```

```
%% Algunos objetos de red de newpr
iteraciones = dome.num_epochs
desemp = dome.perf(iteraciones + 1)
tie = dome.time(iteraciones + 1)
tiempo = tiempo + tie
gradiente = dome.gradient(iteraciones + 1)

l=l+1;
end % fin de while
% Despliega en pantalla los porcentajes de aciertos del              %
entrenamiento y prueba para los dos modelos de ANN
 A=mean(des(:,2));
 B=mean(des(:,3));
 C=mean(des(:,4));
 D=mean(des(:,5));
 F = [A B C D];
fprintf('El promedio global de aciertos: %12.5f\n', F);
```