Clasificación de imagenes utilizando un perceptrón

Alexis Horacio López Fragoso

Universidad Autónoma de México,
Facultad de Estudios Superiores Cuautitlán,
Cuautitlán, Estado de México, México
guados123@gmail.com

Resumen Este proyecto presenta el desarrollo e implementación de un perceptrón para la clasificación de imágenes de 15 tipos diferentes de vegetales. Utilizando un conjunto de datos compuesto por 15,000 imágenes, se entrena el perceptrón para diferenciar entre distintas clases de vegetales. El modelo pasa por dos etapas fundamentales: una fase de entrenamiento, donde ajusta sus pesos para minimizar los errores de clasificación, y una fase de prueba, en la que se evalúa su capacidad de reconocimiento utilizando imágenes de verificación con las cuales no fue entrenado el perceptrón.
Como parte importante del proyecto, se implementó una interfaz física usando una tarjeta de desarrollo Arduino UNO que permitió interactuar con el entorno. Dada la naturaleza de clasificación binaria del perceptrón, se utilizan LEDs como indicadores físicos para representar las salidas: un LED se enciende si la imagen pertenece a una clase específica de vegetal, y otro LED se enciende para la clase opuesta, constando de 15 valores binarios posibles para la clasificación. Esta aplicación con un perceptrón demuestra la rentabilidad práctica del perceptrón en sistemas de clasificación de imágenes con una sola neurona artificial.

Keywords: Perceptrón, clasificación de imágenes, entrenamiento y prueba, tarjeta de desarrollo Arduino, indicadores LED, clasificación binaria, reconocimiento de imágenes

1. Introducción

El desarrollo de algoritmos para la clasificación de imágenes es fundamental en el campo del *machine learning*, especialmente en aplicaciones que involucran reconocimiento de detección de objetos y reconocimiento de imágenes. Los perceptrones, siendo la forma más simple de redes neuronales artificiales, proporcionan una base para abordar tareas de clasificación binaria. Este proyecto se enfoca en la implementación de un perceptrón único para realizar la clasificación de imágenes de vegetales, utilizando un conjunto de datos compuesto por 15,000 imágenes de diversas verduras.
El objetivo principal es diseñar y entrenar un perceptrón que pueda diferenciar entre diferentes y clasificar tipos de vegetales a partir de sus imágenes.

El perceptrón pasará por una fase de entrenamiento en donde se ajustarán sus pesos para minimizar los errores de clasificación, utilizando el *dataset* de conjunto de datos. Posteriormente, se evaluará su rendimiento en una fase de prueba con datos de validación, demostrando así su capacidad de generalización y aprendizaje.
Además de la implementación del perceptrón en software, el proyecto busca integrar este modelo con un sistema físico utilizando una tarjeta Arduino UNO. Al tratarse de una clasificación binaria, se utilizarán indicadores físicos como LEDs para representar las posibles salidas del perceptrón: activado o no activado el LED. Por ejemplo, un conjunto de LEDs se encenderán si se detecta una determinada clase de vegetal, y otro LED se encenderá para la clase opuesta. Cada vegetal tendrá un número binario determinado para definir su clasificación. Esta integración demostrará la aplicación práctica del modelo y mostrará la interacción con el entorno físico.
La realización de este proyecto permitirá explorar las capacidades y limitaciones de los perceptrones en tareas de clasificación de imágenes, entendiendo cómo pueden ser entrenados y utilizados en aplicaciones reales. También proporcionará experiencia práctica en el manejo de grandes conjuntos de datos y en la interacción entre software y hardware, sentando las bases para futuros trabajos con arquitecturas de redes neuronales más complejas.

2. Estado del Arte

El **perceptrón**, introducido por Frank Rosenblatt en 1958, es uno de los modelos más fundamentales en el campo del *aprendizaje automático* y las *redes neuronales artificiales* [1]. Es un clasificador lineal que mapea un conjunto de entradas a una salida binaria mediante una función de activación, típicamente la función escalón. Aunque es un modelo simple, el perceptrón ha sentado las bases para el desarrollo de arquitecturas más complejas y ha demostrado ser efectivo en problemas de clasificación linealmente separables [2].

2.1. Perceptrón en la Clasificación de Imágenes

La clasificación de imágenes es una tarea central en el procesamiento digital de imágenes y visión por computadora. Consiste en asignar una etiqueta o categoría a una imagen basada en su contenido visual. Tradicionalmente, esta tarea ha sido abordada utilizando *redes neuronales convolucionales* (CNNs) debido a su capacidad para capturar características espaciales y jerárquicas en los datos de imagen [**?**]. Sin embargo, en escenarios con recursos computacionales limitados o para fines educativos, el uso de un perceptrón simple puede ser una alternativa viable [3].

Para utilizar un perceptrón en la clasificación de imágenes, es necesario transformar las imágenes en vectores de características. Esto se puede lograr mediante técnicas de preprocesamiento como la escala de grises, normalización

y aplanamiento de las imágenes [4]. Aunque el perceptrón tiene limitaciones en cuanto a su capacidad para manejar problemas no linealmente separables, puede ser efectivo en conjuntos de datos donde las clases son distinguibles mediante combinaciones lineales de las características [5].

2.2. Clasificación de Imágenes utilizando Neuronas Artificiales

Las **neuronas artificiales** son unidades fundamentales en las redes neuronales que procesan y transmiten información, inspiradas en el funcionamiento de las neuronas biológicas [6]. En el contexto de la clasificación de imágenes, las neuronas artificiales se utilizan para construir modelos capaces de aprender representaciones y patrones complejos en los datos visuales.

Las redes neuronales multicapa (MLP), que consisten en capas de neuronas artificiales, permiten resolver problemas de clasificación no linealmente separables al introducir capas ocultas y funciones de activación no lineales como la sigmoide o ReLU [7]. Esto amplía significativamente la capacidad de los modelos para capturar relaciones complejas en los datos de imagen.

En aplicaciones de clasificación de imágenes, las neuronas artificiales procesan las entradas (por ejemplo, los píxeles de una imagen) y generan una salida que indica la probabilidad de pertenencia a una clase específica [8]. El entrenamiento de estas redes se realiza mediante algoritmos como el descenso de gradiente y retropropagación del error, ajustando los pesos sinápticos para minimizar una función de pérdida [3].

Las *redes neuronales convolucionales* (CNNs) son una arquitectura especializada que utiliza neuronas artificiales con conexiones locales y compartición de pesos para aprovechar la estructura espacial de las imágenes [9]. Las CNNs han logrado avances significativos en la clasificación de imágenes, superando ampliamente a los métodos tradicionales y a las redes totalmente conectadas en tareas como el reconocimiento de objetos y la detección de características.

Sin embargo, para problemas más simples o cuando se trabaja con recursos computacionales limitados, modelos más básicos basados en neuronas artificiales, como el perceptrón o MLP con una capa oculta, pueden ser suficientes [10]. La clave está en realizar un preprocesamiento adecuado de las imágenes y extraer características relevantes que puedan ser procesadas efectivamente por las neuronas.

2.3. Limitaciones y Desafíos

Una de las principales limitaciones del perceptrón y de las redes neuronales simples es su incapacidad para capturar relaciones altamente no lineales y complejas en los datos de imagen [4]. Además, al trabajar con imágenes de alta resolución, la cantidad de parámetros puede crecer rápidamente, lo que aumenta el riesgo de sobreajuste y requiere más datos para entrenar adecuadamente el modelo [3].

Otro desafío es la necesidad de un preprocesamiento exhaustivo y una ingeniería de características eficiente para extraer información relevante de las

imágenes [11]. Técnicas como la reducción de dimensionalidad, extracción de bordes, histogramas de gradientes orientados (HOG) y descriptores locales pueden ayudar a representar las imágenes de manera más manejable para las neuronas artificiales [12].

2.4. Integración con Hardware y Aplicaciones en Tiempo Real

La implementación de modelos de clasificación en hardware, como tarjetas de desarrollo o microcontroladores, es un área de interés creciente [**?**]. La simplicidad de las neuronas artificiales y de modelos como el perceptrón facilita su integración en sistemas físicos, permitiendo aplicaciones prácticas como la clasificación en tiempo real con indicadores físicos (por ejemplo, LEDs). Esto es particularmente útil en entornos industriales o agrícolas donde se requiere una respuesta rápida sin la necesidad de infraestructura computacional pesada [**?**].

El uso de hardware especializado, como FPGAs o ASICs, también permite acelerar el procesamiento y reducir el consumo de energía, lo que es beneficioso para dispositivos embebidos y aplicaciones móviles [13].

3. Conocimientos Previos

Para comprender el desarrollo y la implementación de un perceptrón para la clasificación de imágenes, es fundamental entender los conceptos básicos de las redes neuronales artificiales, el funcionamiento del perceptrón y las técnicas de preprocesamiento de imágenes.

3.1. Redes Neuronales Artificiales

Las **redes neuronales artificiales** (RNA) son modelos computacionales inspirados en la estructura y funcionamiento del cerebro humano [5]. Están compuestas por unidades de procesamiento simples llamadas *neuronas*, que están interconectadas y trabajan en conjunto para resolver problemas complejos.

Cada neurona procesa una entrada y produce una salida utilizando una función de activación. La neurona j en una capa recibe las entradas x_i y calcula una salida y_j mediante:

$$y_j = \phi\left(\sum_{i=1}^{n} w_{ji}x_i + b_j\right) \tag{1}$$

donde w_{ji} son los pesos sinápticos que conectan la entrada i con la neurona j, b_j es el sesgo asociado a la neurona j, y ϕ es la función de activación [3].

3.2. El Perceptrón

El **perceptrón** es el tipo más simple de neurona artificial y sirve como bloque básico para redes neuronales más complejas [1]. Es un clasificador lineal que toma una entrada $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ y produce una salida binaria y mediante:

$$y = \begin{cases} 1 & \text{si } \mathbf{w}^T\mathbf{x} + b > 0 \\ 0 & \text{en caso contrario} \end{cases} \tag{2}$$

donde $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$ es el vector de pesos sinápticos y b es el sesgo [5].

El perceptrón utiliza una función de activación escalón (*Heaviside step function*):

$$\phi(u) = \begin{cases} 1 & \text{si } u > 0 \\ 0 & \text{en caso contrario} \end{cases} \tag{3}$$

Regla de Aprendizaje del Perceptrón El entrenamiento del perceptrón implica ajustar los pesos $\mathbf{w}$ y el sesgo b para minimizar los errores de clasificación en un conjunto de datos de entrenamiento [1]. La regla de aprendizaje del perceptrón actualiza los pesos de la siguiente manera:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta\mathbf{w} \tag{4}$$

$$\Delta\mathbf{w} = \lambda(d - y)\mathbf{x} \tag{5}$$

$$b^{(t+1)} = b^{(t)} + \lambda(d - y) \tag{6}$$

donde λ es la tasa de aprendizaje, d es la salida deseada y y es la salida actual del perceptrón [5].

3.3. Funciones de Activación

En las redes neuronales modernas, se utilizan funciones de activación no lineales para permitir que la red aprenda relaciones complejas en los datos [4]. Algunas funciones de activación comunes incluyen:

Función Sigmoide

$$\phi(u) = \frac{1}{1 + e^{-u}} \tag{7}$$

Función ReLU (Rectified Linear Unit)

$$\phi(u) = \text{máx}(0, u) \tag{8}$$

3.4. Preprocesamiento de Imágenes

Para utilizar imágenes como entradas para un perceptrón, es necesario preprocesarlas y convertirlas en vectores de características [11].

Conversión a Escala de Grises Las imágenes en color se convierten a escala de grises para reducir la dimensionalidad y simplificar el procesamiento:

$$I_{\text{gris}} = 0{,}2989R + 0{,}5870G + 0{,}1140B \tag{9}$$

donde R, G y B son las componentes rojo, verde y azul de la imagen [11].

Normalización y Aplanamiento Las imágenes se normalizan para que los valores de píxeles estén en un rango estandarizado (por ejemplo, [0,1]) y luego se aplanan para convertirlas en vectores de características:

$$\mathbf{x} = \text{flatten}\left(\frac{I_{\text{gris}}}{255}\right) \tag{10}$$

3.5. Métricas de Evaluación

Para evaluar el desempeño del perceptrón en la clasificación, se utilizan métricas como la **exactitud** (*accuracy*), definida como:

$$\text{Exactitud} = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}} \tag{11}$$

Otras métricas incluyen la precisión, la exhaustividad (*recall*) y la matriz de confusión [14].

3.6. Implementación en Hardware

La implementación de modelos de aprendizaje automático en hardware requiere conocimientos de electrónica y programación de dispositivos embebidos [15]. Se utilizan tarjetas de desarrollo como Arduino o Raspberry Pi para conectar el modelo con dispositivos físicos como LEDs, permitiendo la interacción con el entorno.

4. Metodología

El desarrollo del proyecto se divide en tres etapas principales: preprocesamiento de datos, implementación del perceptrón y despliegue en hardware para la interacción física. A continuación, se describen detalladamente cada una de estas etapas.

4.1. Preprocesamiento de Datos

El conjunto de datos utilizado consta de **15,000 imágenes** de diversas verduras, recopiladas de fuentes en línea y bases de datos públicas [16]. Para preparar las imágenes para su uso en el perceptrón, se aplicaron los siguientes pasos de preprocesamiento:

Conversión a Escala de Grises, Filtrado Gaussiano y Binarización El preprocesamiento de las imágenes se realizó utilizando Python y la biblioteca OpenCV (`cv2`) [17]. Utilizando también la librería de Os (`os`) [18] para la interacción de las carpetas donde se encuentran almacenadas las imágenes. Las imágenes se procesaron de la siguiente manera:

1. **Conversión a Escala de Grises**: Cada imagen en color se convirtió a escala de grises utilizando la función `cv2.cvtColor` con el parámetro `cv2.COLOR_BGR2GRAY`.
2. **Aplicación de Filtro Gaussiano**: Se aplicó un filtro Gaussiano para suavizar la imagen y reducir el ruido, utilizando la función `cv2.GaussianBlur` con un kernel de tamaño adecuado [11].
3. **Binarización con el Método de Otsu**: Se empleó el método de Otsu para binarizar las imágenes, que selecciona un umbral óptimo automáticamente [19]. Esto se realizó mediante la función `cv2.threshold` con el parámetro `cv2.THRESH_BINARY + cv2.THRESH_OTSU`.
4. **Almacenamiento de las imágenes binarizadas**: Se almacenó en una carpeta cada una de las imágenes binarizadas, utilizando la función `imwrite` con los parámetros del path de almacenamiento y la imagen binaria.

Redimensionamiento y Aplanamiento de Imágenes Una vez preprocesadas en Python, las imágenes binarizadas se guardaron y luego se procesaron en MATLAB para realizar las siguientes tareas:

1. **Redimensionamiento**: Las imágenes fueron redimensionadas a una resolución uniforme de 64×64 píxeles utilizando la función `imresize` de MATLAB [20].
2. **Conversión a Vector Fila**: Cada imagen redimensionada se aplanó y se convirtió en un vector columna, formando un vector de características de dimensión 4096×1, donde $4096 = 64 \times 64$ es el número total de píxeles por imagen.
3. **Normalización**: Los valores de los píxeles se normalizaron para estar en el rango $[0, 1]$ mediante la división entre 255.

División del Conjunto de Datos El conjunto de datos preprocesado se dividió en conjuntos de entrenamiento y prueba en una proporción de 85 % y 15 %, respectivamente [3].

- **Conjunto de Entrenamiento**: 15,000 imágenes.
- **Conjunto de Prueba**: 3,000 imágenes.

4.2. Implementación del Perceptrón

El perceptrón se implementó en MATLAB, utilizando las funciones y herramientas disponibles para el entrenamiento de redes neuronales [20].

Entrenamiento del Modelo El entrenamiento del perceptrón se realizó en MATLAB con la siguiente metodología:

- **Establecer el número de entradas y salidas para la neurona** : Cada imagen fue redimensionada con la siguiente proporción 64 x 64, por lo que se tiene un total de 4096 entradas para el perceptrón. El número de salidas son 4, el cual es un número binario para representar las 15 clases posibles.
- **Crear el perceptrón en MATLAB** : Creamos el perceptrón mediante la siguiente función de MATLAB `$configure(net, ones(inputs, 1), ones(outputs, 1)$`. donde net es perceptrón.
- **Establecer el número de épocas** : Seleccionamos un total de 10000 épocas para el entrenamiento del perceptrón. Por medio de `net.trainParam.epochs = 10000`.
- **Entrenamiento del perceptrón** : Entrenamos el perceptrón por medio de la siguiente función `$train(net, X_train', y_train_bin')$`, con esto esperamos el tiempo suficiente para que se ajusten los pesos y los sesgos para minimizar los errores de entrenamiento.

El entrenamiento se tomó las suficientes épocas para poder corregir los errores y llegar a los pesos y sesgo correctos.

4.3. Despliegue en Hardware e Interacción Física

Para la integración del perceptrón con un dispositivo físico, se utilizó una **tarjeta de desarrollo Arduino Uno** [21]. La comunicación entre el perceptrón y el hardware se realizó utilizando el **MATLAB Support Package for Arduino Hardware** [22].

Programación del Arduino desde MATLAB Se empleó MATLAB junto con el paquete de soporte para hardware Arduino, lo que permitió programar y controlar el Arduino directamente desde MATLAB. Esto facilitó la integración del modelo de perceptrón entrenado con el Arduino.

Configuración del Sistema y Control de LEDs Se conectaron **cuatro LEDs** al Arduino Uno para representar las quince clases resultantes de la clasificación del perceptrón. El sistema funciona de la siguiente manera:

- Se ejecuta un **script en MATLAB** que solicita al usuario que introduzca una imagen de prueba ya binarizada.
- El script procesa la imagen y utiliza el perceptrón entrenado para determinar a qué clase pertenece.
- A través del paquete de soporte de MATLAB para Arduino, el script envía una señal al Arduino para encender el LED correspondiente a la clase determinada.

Interacción entre MATLAB y Arduino La comunicación entre MATLAB y el Arduino se estableció mediante el uso de funciones proporcionadas por el *MATLAB Support Package for Arduino Hardware* [22]. Esto permitió:

- Conectar MATLAB al Arduino mediante una conexión USB.
- Controlar los pines digitales del Arduino directamente desde MATLAB para encender y apagar los LEDs.

Uso de Imágenes de Validación Se realizaron pruebas con imágenes nuevas para validar el correcto funcionamiento del sistema integrado. Al introducir una imagen de prueba en el script de MATLAB, el sistema:

1. Procesa la imagen y realiza la clasificación utilizando el perceptrón entrenado.
2. Enciende el LED correspondiente en el Arduino para indicar la clase a la que pertenece la imagen.

Este proceso permitió verificar en tiempo real la precisión del modelo y la correcta interacción entre el software (MATLAB) y el hardware (ARDUINO).

4.4. Herramientas y Entornos Utilizados

- **Visual Studio Code: Python 3.8**: Para el preprocesamiento de imágenes utilizando OpenCV, Os y Numpy [23].
- **OpenCV (cv2)**: Biblioteca para procesamiento de imágenes en tiempo real [17].
- **Os (os)**: Biblioteca para manipulación de archivos del sistema operativo [18].
- **MATLAB R2023**: Para redimensionar las imágenes, convertirlas en vectores fila, entrenar el perceptrón y controlar el Arduino [20].
- **MATLAB Support Package for Arduino Hardware**: Para la comunicación y control del Arduino desde MATLAB [22].
- **Deep Learning Toolbox**: Para el entrenamiento del perceptrón [24].
- **Arduino Uno**: Tarjeta de desarrollo utilizada para la interacción física [21].
- **LEDs y resistencias**: Para la representación física de las clases clasificadas.

5. Análisis de Resultados

En esta sección se presentan los resultados obtenidos tras la implementación del perceptrón para la clasificación de imágenes de verduras, así como la interacción física mediante el Arduino Uno. Se analizan los resultados del entrenamiento del modelo, la precisión alcanzada y se muestran imágenes tanto de la ejecución del código como del funcionamiento de la interfaz física Arduino Uno.

5.1. Resultados del Preprocesamiento de Datos

El preprocesamiento de las imágenes permitió obtener datos binarizados y normalizados adecuados para el entrenamiento del perceptrón. A continuación, se muestran ejemplos de los pasos de preprocesamiento aplicados a una imagen de muestra.

Figura 1: (a) Imagen original de un brócoli

Figura 2: (b) Imagen en escala de grises

Figura 3: (c) Imagen tras filtrado Gaussiano

Figura 4: (d) Imagen binarizada con método de Otsu

El uso de la biblioteca OpenCV en Python facilitó la aplicación de filtros y técnicas de binarización, mientras que MATLAB permitió redimensionar y convertir las imágenes en vectores de características para el entrenamiento del perceptrón.

5.2. Entrenamiento y Evaluación del Perceptrón

El perceptrón se entrenó utilizando 15,000 imágenes, con 4,096 entradas correspondientes a los píxeles de cada imagen redimensionada a 64×64. El modelo fue configurado para clasificar en 15 clases distintas, representadas mediante una codificación binaria de 4 bits.

Tras 5,000 épocas de entrenamiento, el modelo alcanzó una convergencia insuficiente. A continuación, se muestra la evolución del error de entrenamiento a lo largo de las épocas.

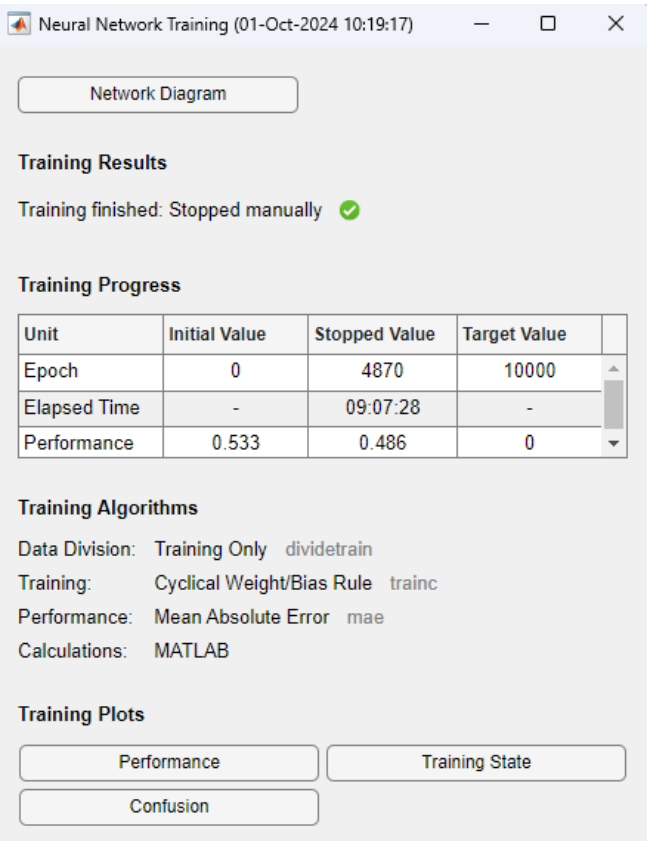


Figura 5: Evolución del error de entrenamiento del perceptrón a lo largo de 10,000 épocas.

Al evaluar el modelo con el conjunto de prueba de 3,000 imágenes, se obtuvo una precisión del **6.7 %**, lo que indica un desempeño ineficiente del perceptrón en la clasificación de las imágenes de vegetales preprocesadas.

5.3. Ejecución del Código y Clasificación de Imágenes

El script en MATLAB solicita al usuario que introduzca una imagen de prueba ya binarizada en la dirección donde estan almacenadas las imágenes. A continuación, se muestra la ejecución del código, donde se carga la imagen y se realiza la clasificación.

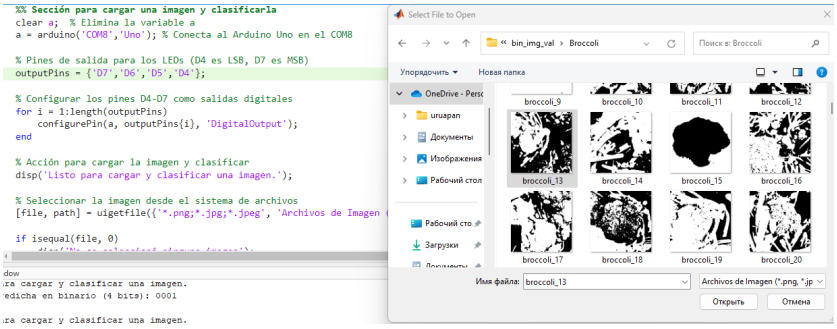


Figura 6: Ejecución del código en MATLAB para la clasificación de una imagen de prueba.

El resultado de la clasificación se muestra en una ventana de MATLAB, indicando la clase asignada a la imagen de prueba junto con la imágen introducida.

5.4. Interacción Física con Arduino

Gracias al *MATLAB Support Package for Arduino Hardware*, se estableció una comunicación directa entre MATLAB y el Arduino Uno. Se conectaron cuatro LEDs al Arduino Uno para representar las 15 clases mediante una combinación binaria de los LEDs, mediante la representación de encendido y apagado.

La siguiente figura muestra el armado del Arduino con los LEDs conectados.

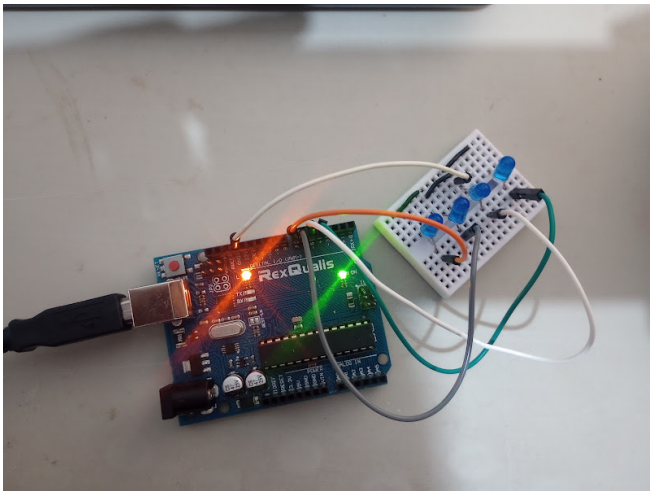


Figura 7: Uso del Arduino Uno con los cuatro LEDs conectados para representar las clases.

Tras la clasificación de la imagen, el script de MATLAB envía señales seriales al Arduino para encender los LEDs correspondientes. La siguiente figura muestra los LEDs encendidos según la clase asignada.

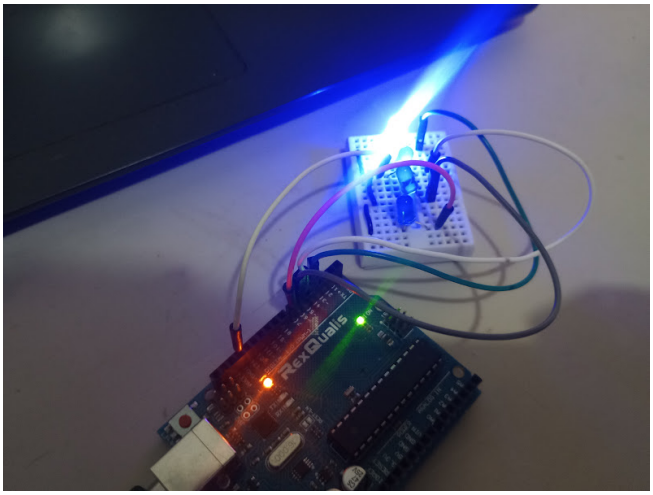


Figura 8: Visualización física de la clase clasificada mediante los LEDs en el Arduino.

Esta interacción física con el Arduino permite una representación tangible del resultado de la clasificación, facilitando su interpretación y validación de los resultados.

5.5. Rendimiento del Perceptrón

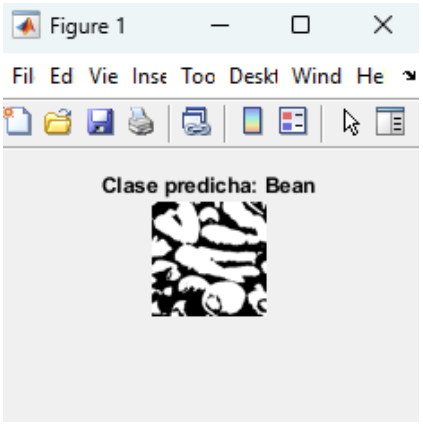
El perceptrón mostró un rendimiento deficiente en la clasificación de las imágenes de verduras, considerando la complejidad del problema y la simplicidad del modelo utilizado. Algunos puntos a destacar son:

- **Precisión del Modelo**: Con una precisión del **6.7 %** en el conjunto de prueba, el perceptrón demostró ser incapaz de generalizar datos no vistos durante el entrenamiento.
- **Tiempo de Entrenamiento**: El entrenamiento con 5,000 épocas permitió que no se logrará ajustar la neurona correctamente, aunque el tiempo de entrenamiento fue considerable largo, tomando 9 horas de entrenamiento, debido al tamaño del conjunto de datos y la dimensión de las entradas.
- **Limitaciones del Perceptrón**: Al tratarse de un modelo lineal, el perceptrón puede tener dificultades para separar clases que no son linealmente separables. Esto puede afectar la precisión en clases con características más complejas, mostrando así una gran incapacidad de clasificar imágenes con un gran tamaño de entradas.

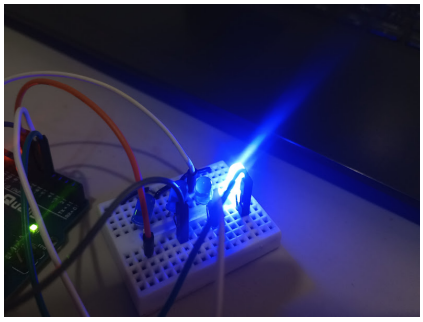
- **Interacción Hardware-Software**: La integración de MATLAB con Arduino facilitó el control de los LEDs y la visualización de los resultados, demostrando la efectividad del *MATLAB Support Package for Arduino Hardware* para este proyecto.

5.6. Casos de Prueba y Validación de las imágenes

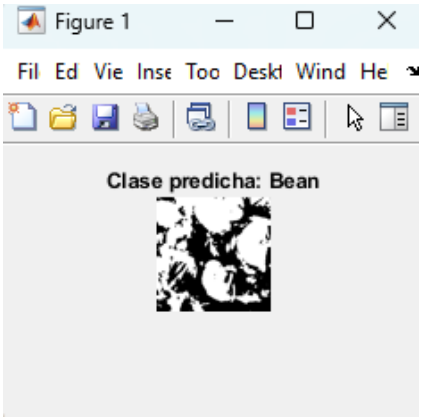
Se realizaron pruebas con diversas imágenes de diferentes clases para validar el funcionamiento del sistema. A continuación, se muestran ejemplos de imágenes de prueba y los LEDs encendidos correspondientes.



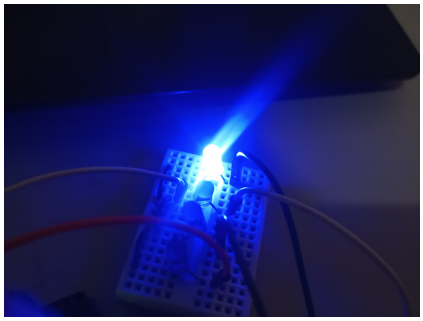
(a) Imagen de prueba 1 Zanahoria



(b) LEDs indicando la clase para prueba 1 Clasifica como si fuerán frijol



(c) Imagen de prueba 2 Lechuga



(d) LEDs indicando la clase para prueba 2 Clasifica como si fuerá frijol

Figura 9: Ejemplos de casos de prueba y los LEDs encendidos correspondientes.

Los resultados confirman que el sistema es incapaz de clasificar correctamente la mayoría de las imágenes y representar físicamente la clase asignada, mostrando en la mayoría de los casos clasificaciónes erroneas.

6. Conclusión

En este trabajo se realizo el entrenamiento de un perceptrn simple para el reconocimiento de imgenes de verduras. Utilizando el software de Matlab para el entrenamiento del perceptrn y Python con la librera de OpenCV para el preprocesamiento de las imgenes, donde se aplic/'o la conversin de blanco a negro, un filtro gaussiano para el suavizado de la imagen, y el mtodo os para la bancarizacin de la imagen. Como interfaz fsica se utiliz el hardware de Arduino Uno y 4 Leds para visualizar la validaci/'on de los datos del sistema.
El aplicar un modelo de reconocimiento de imgenes utilizando un perceptrn simple y una interfaz fsica para la visualizacin de los resultados por medio de 4 Leds. Tomo cierto grado de dificultades, donde se presento un tiempo excesivo de entrenamiento para el perceptrn, tomando 9 horas. Al ingresar 4096 entradas junto con los valores de las 15,000 imgenes de entrenamiento y al ser datos no linealmente separables, resulto una imposibilidad el lograr una correcta clasificacin de las imgenes de verduras, debido a que nunca se llegara al calculo correcto de los pesos para cada una de las entradas y para el vas de la neurona. Resultando as en 6,7 % de precisin, mostrando lo ineficiente que puede ser trabajar con un simple perceptrn para tareas complejas.
La comprobacin de las imgenes de validacin mostraba una tasa de fallo muy alta en cuestin a la clasificacin de la neurona, mostrando en la mayora de los casos una etiqueta errnea. Estos resultados obtenidos proporcionan informacin clave para escoger aplicaciones ms simples para utilizar el perceptrn y evitar implementar aplicaciones donde la gran mayora de datos de entrada son linealmente no separables. Por lo que en conclusin, se necesita hacer uso de ms neuronas y una capa oculta y una capa de salida para poder llegar a un alto grado de precisin en la clasificacin de imgenes.

A. Anexos

En esta sección se incluyen los códigos utilizados en el desarrollo del proyecto, tanto en Python como en MATLAB.

A.1. Código en Python

El siguiente código en Python se utilizó para el preprocesamiento de las imágenes, convirtiéndolas a escala de grises, aplicando un filtro Gaussiano y binarizando las imágenes utilizando el método de Otsu.

```
1 import cv2
2 import os
```

```
import numpy as np

# Lista de subcarpetas (una por cada verdura)
subfolders = ['Bean', 'Bitter_Gourd', 'Bottle_Gourd', 'Brinjal', 'Broccoli', 'Cabbage',
              'Capsicum', 'Carrot', 'Cauliflower', 'Cucumber', 'Papaya', 'Potato',
              'Pumpkin', 'Radish', 'Tomato']

# Ruta de la carpeta donde est n las im genes originales
input_root_folder = 'train_img/'
# Ruta de la carpeta donde se guardar n las im genes binarizadas
output_root_folder = 'bin_train_img/'

# Procesar cada carpeta
for subfolder in subfolders:
    # Definir la ruta de la carpeta de entrada y salida
    input_folder = os.path.join(input_root_folder, subfolder)
    output_folder = os.path.join(output_root_folder, subfolder)

    # Obtener la lista de archivos de imagen en la carpeta de entrada
    image_files = [f for f in os.listdir(input_folder) if f.endswith('.jpg') or f.endswith('.png')]

    # Procesar cada imagen en la carpeta
    for i, image_file in enumerate(image_files):
        # Leer la imagen
        image_path = os.path.join(input_folder, image_file)
        image = cv2.imread(image_path)

        # Convertir la imagen a escala de grises
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Aplicar filtro Gaussiano para eliminar ruido
        blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

        # Binarizar la imagen utilizando el m todo de Otsu
        _, binary_image = cv2.threshold(blurred_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        # Crear el nombre del archivo de salida
        output_image_path = os.path.join(output_folder, f'{subfolder.lower()}_{i+1}.png')

        # Guardar la imagen binarizada
        cv2.imwrite(output_image_path, binary_image)
```

```
44
45          # Mostrar progreso
46          print(f'Imagen {i+1}/{len(image_files)} de la carpeta
                {subfolder} procesada y guardada en {
               output_image_path}')
```

A.2. Código en MATLAB

El siguiente código en MATLAB se utilizó para redimensionar las imágenes, convertirlas en vectores fila, entrenar el perceptrón y controlar el Arduino mediante el paquete de soporte de MATLAB para el Arduino UNO.

```
1  %% Directorio actual
2  cd('C:\Users\guado\OneDrive\Documentos\python NN') %Current
       Folder Path
3
4  % 1. Define the filenames and classes
5  root_folder = 'bin_train_img';  % Root file, 15 files of 1000
        images
6  subfolders = {'Bean', 'Bitter_Gourd', 'Bottle_Gourd', '
       Brinjal', 'Broccoli', 'Cabbage','Capsicum', 'Carrot', '
       Cauliflower', 'Cucumber', 'Papaya', 'Potato', 'Pumpkin',
       'Radish', 'Tomato'};  % Nombres de las carpetas/clases
7  num_classes = numel(subfolders);  % 15 clases
8
9  % Initialize training matrices
10 X_train = [];  % To store the vectors of the images
11 y_train = [];  % To store the classes (1 a 15 para las 15
       clases)
12 image_size = 64 * 64;  % Size of the binarized images (224
       x224)
13
14 % Load images from 15 files paths
15 for class = 1:num_classes
16     train_folder = fullfile(root_folder, subfolders{class})
           % Current folder
17     image_files = dir(fullfile(train_folder, '*.png'))  %
           Find images .jpg
18     num_images = numel(image_files)  % Size of the file
19
20     % Read and convert images to vectors that they will have
           binary data
21     for i = 1:num_images
22         image_path = fullfile(train_folder, image_files(i).
               name);  % Image Root Path
23         img = imread(image_path);  % Read the image
24         img = imresize(img, [64, 64]);
25         img = double(img);
26
```

```
27          % Convert image to row vector
28          img_vector =(img(:)');  % Convertir la imagen en un
                vector fila
29
30          img_vector = img_vector / 255;
31
32
33          % Agregar el vector de la imagen a X_train
34          X_train = [X_train; img_vector];
35
36
37          % Asignar la etiqueta correspondiente (class = 1, 2,
                ..., 15)
38          y_train = [y_train; class];
39      end
40  end
41
42  %% Paso 3: Crear y configurar el Perceptr n
43  % El perceptr n debe tener tantas entradas como p xeles y
        salidas de 4 bits (16 posibles combinaciones, 15 usadas)
44  num_inputs = image_size;  % Cada imagen tiene 224x224
        p xeles (entrada)
45  num_outputs = 4;  % Salida de 4 bits
46
47  net = perceptron;  % Crear el Perceptr n
48  net = configure(net, ones(num_inputs, 1), ones(num_outputs,
        1));
49
50  % Convertir etiquetas de las clases (1-15) a formato de 4
        bits (binario)
51  y_train_bin = double(dec2bin(y_train - 1, num_outputs) - '0')
        ;  % Restar 1 para que sea de 0 a 14
52
53  % Entrenar el perceptr n
54  net.trainParam.epochs = 10000;  % N mero de  pocas para
        entrenar (puedes ajustar)
55  net = train(net, X_train', y_train_bin');  % Entrenar el
        perceptr n (trasponer X_train e y_train)
56
57  %% Paso 4: Evaluar el Perceptr n en im genes de prueba
58  % Configurar de manera similar para las im genes de prueba
59  test_folder = 'bin_img_val/';  % Ruta a la carpeta de
        im genes de prueba
60  X_test = [];
61  y_test = [];
62
63  % Leer im genes de prueba de las mismas 15 carpetas
64  for class = 1:num_classes
65      test_subfolder = fullfile(test_folder, subfolders{class})
            ;  % Carpeta de la clase actual
```

```
    test_files = dir(fullfile(test_subfolder, '*.png'));  % Buscar imágenes .png
    num_test_images = numel(test_files);

    % Leer y convertir las imágenes de prueba a vectores
    for i = 1:num_test_images
        test_image_path = fullfile(test_subfolder, test_files(i).name);  % Ruta de la imagen
        img_test = imread(test_image_path);  % Leer la imagen de prueba
        img_test = imresize(img_test, [64, 64]);
        % Convertir la imagen de prueba a vector fila
        img_test_vector = img_test(:)';  % Convertir la imagen a vector fila
        X_test = [X_test; img_test_vector];  % Agregar a X_test

        % Asignar la etiqueta de clase correspondiente
        y_test = [y_test; class];
    end
end

%% Paso 5: Predecir las clases de las imágenes de prueba
y_pred_bin = net(X_test');  % Usar el Perceptrón entrenado para predecir (4 bits)

% Convertir las predicciones de formato binario a clases (0-14)
y_pred = bin2dec(num2str(y_pred_bin')) + 1;  % Convertir de binario a decimal (1 a 15)

% Mostrar resultados
disp('Etiquetas verdaderas:');
disp(y_test);
disp('Predicciones del Perceptrón:');
disp(y_pred);

% Calcular la precisión
accuracy = sum(y_pred == y_test) / numel(y_test);
fprintf('Precisión del Perceptrón: %.2f%%\n', accuracy * 100);
```

Referencias

1. F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.
2. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.

3. C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.
4. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.
5. S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice Hall, 1994.
6. W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
7. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
8. L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.
9. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 25, pp. 1097–1105, 2012.
10. C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv preprint arXiv:1611.03530*, 2016.
11. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2008.
12. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1. IEEE, 2005, pp. 886–893.
13. V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
14. D. M. W. Powers, "Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.
15. M. Brown, *Embedded Programming with Modern C++*. Birmingham, UK: Packt Publishing Ltd, 2019.
16. M. I. Ahmed, S. M. Mamun, and A. U. Z. Asif, "Dcnn-based vegetable image classification using transfer learning: A comparative study," in *2021 5th International Conference on Computer, Communication and Signal Processing (ICCCSP)*, 2021, pp. 235–243.
17. G. Bradski, "Opencv library," 2000, available at https://opencv.org/.
18. P. library, "Miscellaneous operating system interfaces," 2024, available at https://docs.python.org/3/library/os.html.
19. N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.
20. T. M. Inc., *MATLAB Version 9.8 (R2020a)*, Natick, Massachusetts, 2020.
21. M. Banzi and M. Shiloh, *Getting Started with Arduino*, 3rd ed. Sebastopol, CA: Maker Media, Inc., 2015.
22. MathWorks, *MATLAB Support Package for Arduino Hardware*, 2020, available at https://www.mathworks.com/hardware-support/arduino-matlab.html.
23. P. S. Foundation, "Python programming language," 2020, available at https://www.python.org/.
24. MathWorks, *Deep Learning Toolbox*, 2020, available at https://www.mathworks.com/hardware-support/deep_learning.html.