Modelo de Clasificación Con Redes Neuronales Convolucionales y Optimización Lineal Entera Mixta

*Nota: Artículo basado en 0-1 MILP[1]

1st Julieth Stefanny Escobar Ramírez
Ingeniería Matemática
Universidad EAFIT
Medellín, Colombia
jsescobarr@eafit.edu.co

2nd Sara Gallego Villada
Ingeniería Matemática
Universidad EAFIT
Medellín, Colombia
sgallegov@eafit.edu.co

3rd Juan José Sánchez Sánchez
Ingeniería Matemática
Universidad EAFIT
Medellín, Colombia
jjsanchez@eafit.edu.co

***Resumen*—Colombia es un país productor de tomate, siendo el número 34 en el mundo en cuanto a producción [5]. Pero para ser exportado, se deben cumplir ciertos requisitos de calidad. Las producciones de tomate son muy sensibles a las plagas y enfermedades, las cuales se propagan con gran facilidad, por ello, se planteó el objetivo de que por medio de neuronales convolucionales, se implemente el modelo MILP para detectar tempranamente la infección de plantas de tomate, de manera que se puedan tomar acciones prontamente y prevenir la pérdida de este producto**

***Index Terms*—MILP, redes neuronales, optimización, bacterias, tomate**

I. Introduction

Las Redes neuronales profundas son una herramienta ampliamente utilizada en la actualidad, su difusión e implementación para diferentes modelos de aprendizaje profundo es cada vez mayor debido a su importancia a la hora de intervenir en procesos que requieren reconocer algún tipo de patrón. Las redes neuronales se basan en la idea de simular el funcionamiento de las redes neuronales de los seres vivos en cuanto a su capacidad para trabajar en conjunto con el fin de aprender algo, siendo la matemática y la estadística su fundamentación teórica.

Para el desarrollo de la investigación, se considerará una de las aplicaciones más básicas de las DNN, el cual es un problema de clasificación del estado de salud de las hojas en 2 categórias "Infectada" o "No infectada", por lo que puede ser abordado como un problema binario de programación lineal-entera(MILP), el cual se implementará por medio de una red neuronal con una función de activación lineal, ReLu. Cabe aclarar que se acotará un problema de clasificación que podría ser más complejo, a un problema cuya solución e implementación se ajustan a las habilidades que se buscan desarrollar durante el curso.

Identify applicable funding agency here. If none, delete this.

II. Conceptos previos

II-A. Optimización discreta

Es una rama de las matemáticas que se centra en optimizar las operaciones matemáticas. Se ocupa de hacer el mejor uso de las operaciones para que funcionen mejor cuando se presentan con un conjunto de valores discretos.

II-B. Redes neuronales profundas (DNN)

Una red neuronal profunda es una red neuronal artificial (ANN) con varias capas ocultas entre las capas de entrada y salida. El propósito principal de una red neuronal es recibir un conjunto de entradas, realizar cálculos progresivamente complejos en ellas y dar salida para resolver problemas del mundo real como la clasificación.

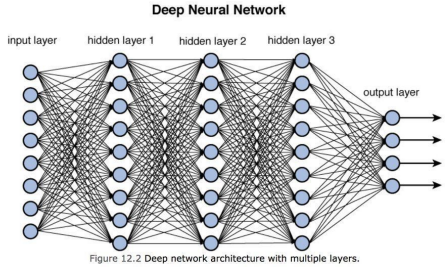


Figura 1: Red Neuronal

Las redes neuronales se utilizan ampliamente en el aprendizaje supervisado y en los problemas de aprendizaje por refuerzo.

II-C. Gradiente descendiente estocástico (SGD)

El gradiente descendente es un algoritmo muy popular utilizado en los modelos de aprendizaje automático, siendo también la base base de las redes neuronales. Es un algoritmo iterativo, que comienza desde un punto aleatorio en una función y viaja por su pendiente hasta alcanzar el punto más

bajo de la función. Este algoritmo es útil en los casos en los que no se puede encontrar los puntos óptimos al igual la pendiente de la función a 0.

La palabra 'estocástico' significa que está vinculado a un proceso con una propiedad aleatoria. Por lo tanto, en el gradiente descendente estocástico se seleccionan aleatoriamente algunas muestras en lugar de todo el conjunto de datos para cada iteración.

Algoritmo SGD(Graficente descendiente estocástico):

$$\theta_j - \alpha(\widehat{y}^i - y^i)x_j^i$$

En SGD, encontramos el gradiente de la función de costos de un solo ejemplo en cada iteración en lugar de la suma del gradiente la función de costos de todos los ejemplos. Dado que SGD es generalmente más ruidoso que el gradiente descedente típico, por lo general se necesita mayor número de iteraciones para alcanzar los mínimos, sin embargo es computancionalmente menos costoso que este último.

II-D. Funciones de activación

Son funciones matemáticas simples, las cuales transforman una entrada dada a la salida requerida. Las funciones son las encargadas de activar los neurones, es decir, cambiar de Apagado/Encendido. El neurón recibe una suma del producto de las entradas y aleatoriamente inicializa los pesos de estos con un sesgo en cada capa. A esta suma se la aplica la función de activación y se genera la salida de la función. Es necesario agregar que las funciones de activación son de carácter no lineal, porque si bien pueden parecer lineales no lo son, sino los modelos se comportarían como una regresión lineal y tendría una limitante de aprendizaje.

II-D1. Función sigmoide: Es una función matemática que tiene una curva característica en forma de "S", que transforma los valores entre el rango 0 y 1. La función sigmoide también se llama curva sigmoide o función logística. Es una de las funciones de activación no lineal más utilizadas[3].Esta función está definida por la fórmula:

$$S(t) = \frac{1}{1 + e^{-t}} \tag{1}$$

La combinación lineal en la función sigmoide resultante es:

$$S(w^t x) = \frac{1}{1 + e^{-w^t x}} \tag{2}$$

La gráfica muestra lo descrito anteriormente:

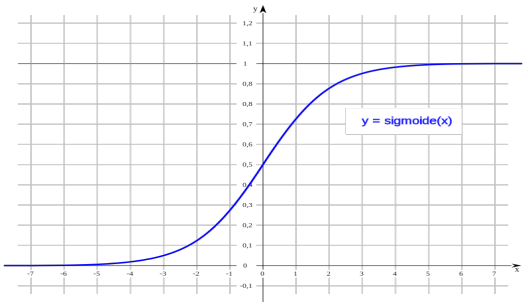


Figura 2 Regresión logística

Ahora, esta permitirá que la probabilidad esté correctamente calibrada y que arroje la probabilidad de que el modelo este entre esos valores, es decir que:

$$0 \text{ si y solo sí } (6) \ngeq 0{,}5$$

$$1 \text{ si y solo sí } (6) \geq 0{,}5$$

II-D2. ReLU: Ahora, para nuestro modelo utilizaremos la función de activación ReLU, por lo cual definiremos esta como una función no lineal.Esta actua como linear en los valores positivos, y como no lineal en los valores negativos. Estos ayuda a presevar propiedades y hacer modelos lineales más faciles de optimizar con gradiente basado en algoritmos. La rectificadora, mejor conocida como ReLU. Esta función dara un salida si la entrada es positiva, sino retornará cero. La entrada seria la suma mencionada anteriormente de los pesos.

Esta es una función muy utilizada, especialemte en las redes neuronales convulcionales. Por lo cual es la que usaremos en el modelo. Además de que es mas eficaz que la sigmoide y tanh.

La función está definida de los $\mathbb{R} \to \mathbb{R}_0^+$ como sigue:

Función por partes

$$R(x) : \begin{cases} x, & x \geq 0 \\ 0 & x < 0 \end{cases}$$

[1]

De igual manera y mejor conocidad como función máx:

tal que: $R(x) := máx(x, 0)$ Gráficamente la podemos ver como:

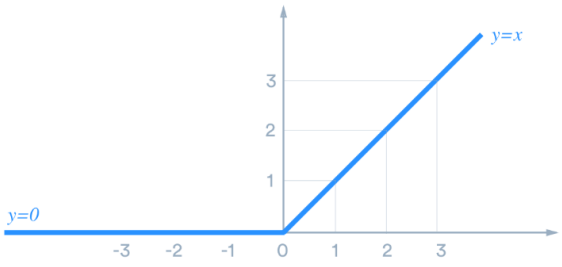


Figura 3: ReLU gráficamente[1]

II-E. Max pooling y avg pooling

Para hacer más eficientes las capas de convulciones utilizamos **máx pooling**, esta operación calcula el valor máximo para fragmentos de un mapa de caraterísticas y lo utiliza para crear un mapa de características agrupado. Es decir, es como traducir la imagen a una pequeña cantidad, de manera que no afecta significativamente a los datos. Ejemplo gráfico. Esto se hace para involucrar multiples entradas.

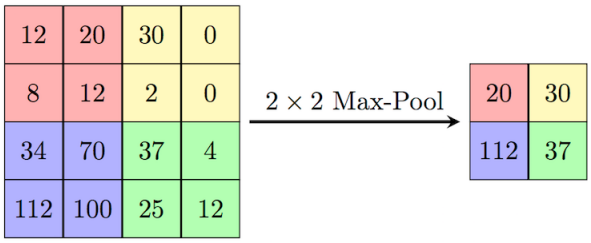


Figura 4: Máx-pool

$x = MaxPool(y_1, ..., y_{n=t}) - max\{y_1, ..., y_t\}$
Ahora tenemos por otro lado tenemos el **avgPool** La cual usaremos después de una capa convulcional, esta crea transforma la imagen en una versión más pequeña, sin embargo, esta reducción no genera cambios significativos en la imagen, y es incluso más suave que el que hace máx pooling. Esta operación calcula el valor medio de los parches de una gráfica de características, y con este crea otro gráfico pero con muestreo reducido. Ejemplo gráfico.

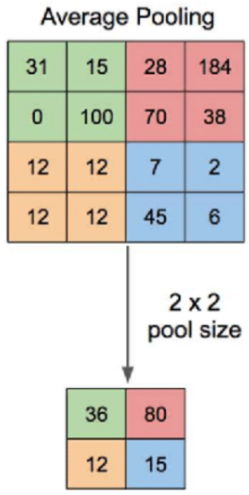


Figura 5 Avg-Pool[2]

III. Mixed Integer Linear Program Binario(Optimización lineal entera mixta)

MILP se puede ver como una extensión de la programación lineal. Intercambiando algunas variables continuas a discretas, tiene la siguiente forma:

$$\text{mín}\, c^T x + d^T y$$

$$S.A\ Ax + By \leq e$$

$$x \in \mathbb{R}^m, y \in \mathbb{Z}^n$$

Donde $y \in \mathbb{Z}^n$ es llamada las restricciones de integralidad, Este tipo de problemas pertenece a los problemas NP-hard, su región factible es: $P(A, B, e) := \{x \in \mathbb{R}^m, y \in \mathbb{Z}^n | Ax + By \leq e\}$
Si $P(A, B, e)$ está acotado, la región factible del MILP es finita. Definimos el casco entero $PI(A, B, e)$ como $PI(A, B, e) := conv(x \in \mathbb{R}^m, y \in \mathbb{Z}^n | Ax + By \leq e)$. [3]

Para implementar MILP necesitamos implementar una variable de activación z
Donde:

$$\left.\begin{aligned} z = 1 \rightarrow x \leq 0 \\ z = 0 \rightarrow s \leq 0 \\ z \in \{0, 1\} \end{aligned}\right\}$$

El uso de una variable de activación binaria z_j^k para cada $UNIT(j, k)$ conduce entonces a la siguiente formulación 0-1 formulación MILP de la DNN:
Definimos nuestro modelo con $K + 1$ capas, enumeradas desde la capa 0 que sería la ficticia corresponde a la entrada de la DNN hasta la última capa K correspondiente a los outputs. Cada capa $k \in \{0, 1, ..., K\}$ está formada por n_k nodos, enumerados de 1 hasta n_k. Se denota por $UNIT(j, k)$ el j-ésimo nodo de la capa k. Ahora, sea $x^k \in \mathbb{R}^{n_k}$ el vector de salida de la capa k entonces x_j^k es el output de la $UNIT(j, k)(j = 1, ..., n_k)$. Como la capa 0 corresponde a la entrada de la DNN, entonces x_j^0 es el j-ésimo valor de la entrada de n_0. Ahora, la formulación sería:

$$\text{min} \sum_{k=0}^{K} \sum_{j=1}^{n_k} c_j^k x_j^k + \sum_{k=1}^{K} \sum_{j=1}^{n_k} Y_j^k Z_j^k$$

$$\left.\begin{aligned} \sum_{i=1}^{n_{k-1}} w_{ij}^{k-1} x_i^{k-1} + b_j^{k-1} = x_j^k - s_k^k \\ x_j^k, s_k^k \geq 0 \\ z_j^k \in \{0, 1\} \\ z_j^k = 1 \rightarrow x_j^k \leq 0 \\ z_j^k = 0 \rightarrow s_j^k \leq 0 \end{aligned}\right\} (1)$$

Con $k = 1, ..., K., j = 1, ..., n_k$ (2)

$$\left.\begin{aligned} lb_j^0 \leq x_j^0 \leq ub_j^0,\ j = 1, ..., n_0 \\ lb_j^k \leq x_j^k \leq ub_j^k \\ \bar{lb}_j^k \leq s_j^k \leq \bar{ub}_j^k \end{aligned}\right\} (3)$$

con $k = 1, ..., K., j = 1, ..., n_k$
En la formulación anterior, todos los pesos w_{ij}^{k-1} y los sesgos b_j^k son parámetros dados (constantes);lo mismo ocurre con la función de costo c_j^k y Y_j^k , que puede definirse según el problema específico que se plantee. Las condiciones de (1) definen la salida de la ReLU para cada unidad, mientras que de (2-3) imponen límites inferiores y superiores conocidos en las variables x y s: la superior en las variables x y s: para $k = 0$, estos límites se aplican a los valores de entrada de la DNN de entrada de la DNN x_j^0y dependen de su significado físico, mientras que para $k \geq 1$ se tiene $lb_j^k = \bar{lb}_j^k = 0$ y $ub_j^k, \bar{ub}_j^k \in \mathbb{R}_+ U\{\infty\}$ [4]

IV. Aplicaciones

IV-A. Problemas de clasificación

Para este proyecto se tomó un problema de clasifiación binaria, entre hoja infectadas por y hojas no infectadas. Para ver la aplicación de esta problemática de manera más general, se considera un problema trivial de clasificación: El reconocimiento automático de los dígitos del 0-9 de una imagen 28x28. Se debe configurar la red neuronal, de tal manera que las entradas de la primera capa sean los valores de cada pixel, cada uno de dichos pixeles de la figura debe ser normalizado a una escala de grises definidas por la variable binaria[0,1], donde 0 corresponde al color blanco y 1 al negro. La red también debe tener 10 neuronas de salida, una para cada dígito.

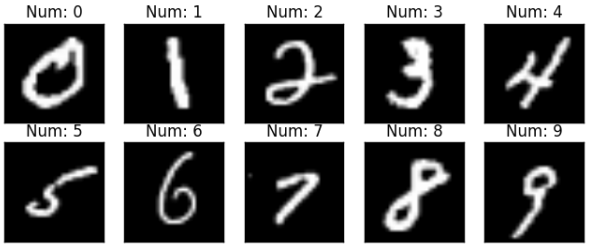


Figura 6 Clasificación de dígitos

Para entrenar la red neuronal se utiliza la base de datos "MNIST", la cual es una colección de estos dígitos que contiene más de 60.000 ejemplos y 10.000 imágenes de prueba, este conjunto de datos permitirá al algoritmo hacer el proceso de reconocimiento, para así poder clasificar las imágenes por medio de la función de activación que se defina, por ejemplo, ReLu. Finalmente se evalua la cantidad predicciones acertadas y erradas para determinar la precisión del modelo.

IV-B. Visualización de características

El 0-1 MILP permite hallar ejemplos de entrada que maximicen la activación x_j^k de cada unidad(j,k) Las redes neuronales aprenden ciertas características y conceptos de imágenes sin procesar, dichas funciones que son aprendidas pueden ser visualizadas mediante la maximización de activación. Por ejemplo en la figura(x) se refleja una entrada convencional y es evidente que mo es fácil encontrar algún tipo de patrón en la imagen.

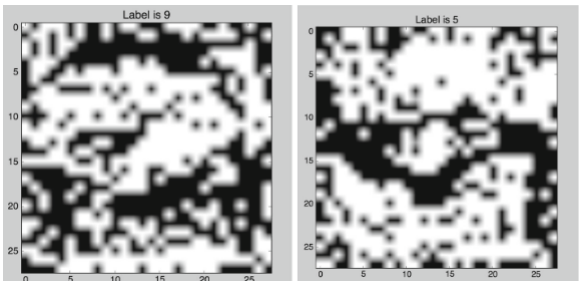


Figura 7 Ningún patrón reconocible

Tradicionalmente, la activación máxima se ha implementado por medio de algoritmos voraces, los cuales pueden no ser tan efectivos, al capturar soluciones óptimas locales. Las implementaciones fundamentadas en el gradiente a veces son incapaces de encontrar soluciones óptimas significativas con activación estrictamente mayor que 0.

IV-C. Construcción de ejemplos adversos

Consiste en encontrar soluciones óptimas que permitan identificar algunas debilidades de la red neuronal. El procedimiento consiste en modificar sutilmente una entrada para producir una salida incorrecta. Por ejemplo, se tiene determinada entrada $x^{\sim 0}$ que está clasificada correctamente como un dígito $\tilde{d}$, y se quiere generar una figura similar x que se clasifique erróneamente como $d \neq \tilde{d}$. La creación de casos adversos evidencia la vulnerabilidad de los modelos de aprendizaje automático. Por ejemplo, un automóvil que funciona de manera autónoma, choca contra otro debido a que la señal de alto que debía interpretar el vehículo fue modificada por alguien; Una persona podría identificar fácilmente que la señal le está expresando que se detenga a pesar de que haya sido alterada, pero el auto no pudo saberlo.

V. Metodología

La investigación empezó con el estado del arte, donde se definieron conceptos importantes usados durante la apliación del modelo, posteriormente se define una aplicación trivial, para dar una introducción y un antecedente al proyecto. De lo anterior, se tiene que con este proyecto se pretende hacer una clasificación de hojas, entre infectadas y no infectadas. Siendo esta la variable binaria considerada. Para esto, se tiene una base de datos, y mediante el modelo de clasificación de imagenes DNN , funciones de activación sigmoid y ReLU, se pretende lograr la mayor precisión posible.

V-A. Herramientas

Para el modelo utilizamos librerías como: Os, Pandas, Numpy y Tensor Flow. Con los cuales se hizo el modelo, las conclusiones, y gráficas para las mismas
Además, una base de datos de plantas la cual consistía en 600 datos en total.

V-B. Formulacion del modelo

Con las librerías se hizo el respectivo análisis previo. Se normalizaron las imágenes y se pusieron de tamaños iguales, se tomó 50 de epoch y un bathsize de 50. Para comprobar el modelo se hicieron diversas versiones, así que inicialmente se tomaron 3 capas de (8,8,8)ReLU, al igual que el modelo propuesto por [4] y una capa final sigmoide para hacer la clasificacion entre infectada y no infectada, pero se modificaron continuamente estos "parámetros"para obtener las conclusiones. El modelo definitivo fue 2 ReLU, con una capa final de sigmoide entre cada capa, para hacer la DNN, se utilizó Max Pooling y antes de la última capa se hizo un flaten para entregar mejores resultados.

VI. Resultado y conclusiones

En el proceso de resultados las entradas que se le dieron al modelo, retornaban con altas probabilidades que la imagen pertenecía a los tomates sano o no, sin embargo, este resultado puede ser no muy valioso debido a que se obtuvo overfitting [1] como se puede observar en el siguiente análisis del modelo

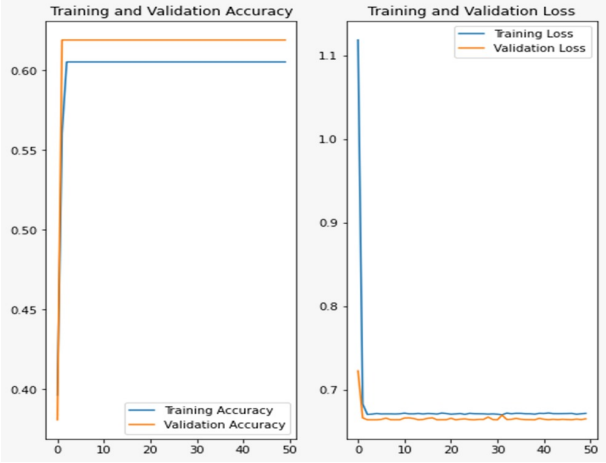
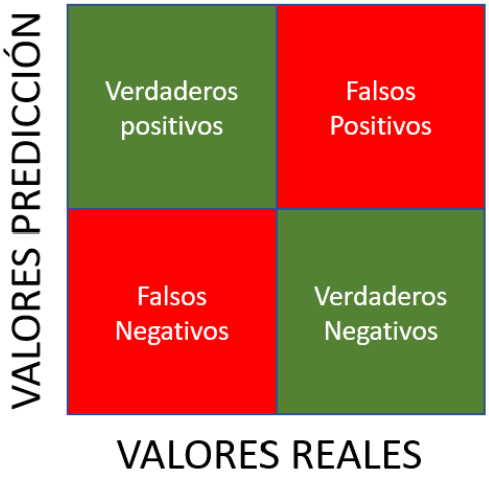


Figura 9: Overfitting

La razón de esto en el modelo implementado, es que los datos utilizados presentan demasiado ruido, es decir son muy diferenciables entre si. De igual manera, el modelo tenía cerca de 4000 entradas para cada categoría, aquí el sesgo y la varianza hacen que se produzca un modelo extraño, dado que si es súper flexible es porque está aprendiendo el ruido del conjunto de datos y hace que se sobreajuste. Sin embargo, de todos los datos la configuración que mejores resultados dio fue la final.

También se decidió hacer un análisis con matrices de confusión, la cual tiene la siguiente estructura:



[1] el modelo pierde la capacidad de generalizar, puesto que, en vez de capturar los patrones generales subyacentes en el conjunto de entrenamiento

Figura 10: Matriz de confusión

La figura anterior indica como una matriz de confusión plasma resultados, en este caso los verdaderos positivos serían las hojas sanas que en realidad estaban sanas y los falsos positivos serían las hojas enfermas que en realidad estaban enfermas.Lo enunciado anteriormente se puede observar en la siguiente imagen:

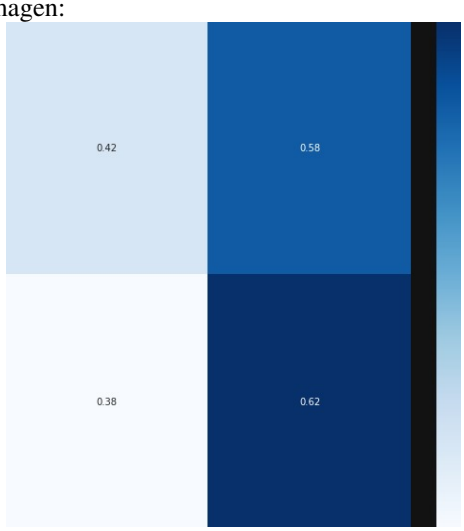


Figura 11: resultado más acertado

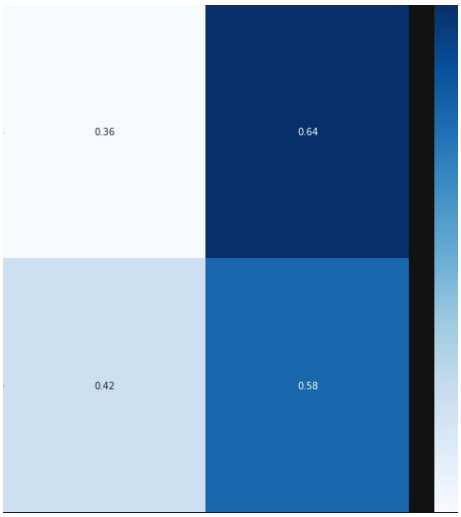


Figura 12: Resultado menos acertado

VI-A. Conclusiones:

Después de varios resultados obtenidos, se concluyó que el modelo implementado no es exacto en el proceso de clasificación de hojas. Esto se evidencia en los resultados ubicados en la diagonal principal del arreglo rectangular, donde se encuentran las evaluaciones correctas realizadas por el modelo: verdaderos positivos y negativos, revelando que el desempeño de la clasificación no fue el mejor, ya que no acertó en muchos casos, por ejemplo únicamente en el 36 %-42 % de las veces que las catalogó a las hojas como

sanas, realmente lo estaban. Además, es evidente que también hay presencia de varios falsos positivos, falsos negativos y verdaderos negativos, claramente los resultados no coinciden con los valores reales. Esta inconsistencia se podría presentar por varias razones.

- Al tratarse de un modelo de aprendizaje no supervisado, no existe ninguna etiqueta o categorización de los datos, por lo que recurre a agrupar las cosas de acuerdo con las similitudes existentes, es decir, es probable que el algoritmo pueda fallar en la detección de dichos patrones o estructuras análogas en las imágenes.

- - Para la evaluación del modelo, se implementa la matriz de confusión. Para ello, se divide el dataset en 2 partes: la primera está compuesta por el 75 % de los datos y conforman el grupo de datos de entrenamiento, el 25 % restante hace parte de los datos de prueba. Debido a que los datos son muy uniformes y se generó "overfitting", es muy probable que el algoritmo se haya ajustado únicamente a características muy especifícas de los datos, esto sumado a que seguramente la división de los datos no fue la adecuada, pudo ocasionar que la red no diferencie fácilmente una hoja sana de una enferma. Eso se evidencia en los resultados obtenidos, especialmente en los verdaderos positivos y falsos positivos, donde se puede apreciar que la precisión a la hora de detectar una planta saludable es bastante baja.

AGRADECIMIENTOS

Agradecimientos al profesor Cristhian Montoya por la ayuda en la contrucción del proyecto, entendimiento de conceptos y apoyo para llevarlo más allá.
De igual manera el apoyo por parte del profesor de inteligencia artificial en el entendimiento de las redes neuronales.

REFERENCIAS

[1] V. Praharsha, "ReLU (Rectified Linear Unit) Activation Function" OpenGenus IQ [internet] https://iq.opengenus.org/relu-activation/ .
[2] Math papers with code, .Average Pooling"Pooling Operations [internet]https://math.paperswithcode.com/method/average-pooling
[3] R. Shen, "MILP Formulations for Unsupervised and Interactive Image Segmentation and Denoising," Thesis, Universidad Heidelberg. [Internet] .Link
[4] Fischetti, M., Jo, J. Deep neural networks and mixed integer linear optimization. Constraints 23, 296–309 (2018). https://doi.org/10.1007/s10601-018-9285-6 optimization" 2018
[b5] dama, El gran tomate se siembra en colombia. [Internet] https://www.adama.com/colombia/es/noticias-y-actualidad/el-gran-tomate-se-siembra-en-colombia