

CONSTRUCCIÓN DE PROTOTIPO CLASIFICADOR DE TUBÉRCULOS DE PAPA MEDIANTE SUPERVISIÓN DE VISIÓN COMPUTACIONAL



**UNIVERSIDAD DE
SAN BUENAVENTURA**

**Jaither Bejarano Casas
John Jairo Caicedo Ferrer
Sergio Arley Rivera Jerez**

Ing. Jefferson David Salamanca Cardenas

Universidad de San Buenaventura sede Bogotá

Facultad de Ingeniería

Programa de Ingeniería Mecatrónica

Bogotá D.C., Colombia

2022

Contenido

| | | |
|----------|--------------------------------------------------|-----------|
| 1 | Introducción | 3 |
| 1.1 | Objetivos | 4 |
| 1.1.1 | Objetivo General | 4 |
| 1.1.2 | Objetivos Específicos | 4 |
| 1.2 | Alcances y Limitaciones | 4 |
| 1.2.1 | Alcances | 4 |
| 1.2.2 | Limitaciones | 5 |
| 1.3 | Justificación | 5 |
| 1.4 | Descripción y Formulación Del Problema | 5 |
| 2 | Marco Conceptual | 7 |
| 2.1 | Agricultura De Precisión | 7 |
| 2.2 | Redes Neuronales Artificiales | 7 |
| 2.3 | Convolución Dilatada | 7 |
| 2.4 | <i>Overfitting</i> | 8 |
| 2.5 | <i>Underfitting</i> | 8 |
| 2.6 | Data Augmentation | 8 |
| 2.7 | <i>Dropout</i> | 9 |
| 2.8 | Filtro de mediana | 9 |
| 2.9 | Imagen Binaria | 9 |
| 2.10 | <i>Max-pooling</i> | 9 |
| 2.11 | <i>Padding</i> | 10 |
| 2.12 | Procesamiento de imágenes | 11 |
| 2.13 | <i>Stride</i> | 11 |
| 2.14 | Aprendizaje por transferencia | 12 |
| 2.15 | Umbralización De una Imagen Digital | 12 |
| 2.16 | Visión artificial | 12 |
| 3 | Estado del arte | 13 |
| 3.1 | MatLab | 13 |
| 3.2 | Redes Neuronales Artificiales | 13 |
| 3.3 | Super Vector Machine | 14 |

| | |
|----------------------------------------------------------------------|-----------|
| Contenido | 1 |
| 3.4 Otros métodos | 15 |
| 3.5 Clasificación De Papa En Colombia | 16 |
| 4 Diseño Del Algoritmo de Clasificación | 18 |
| 4.1 Preparación y Adquisición De Las Imágenes | 20 |
| 4.2 Distribución del Dataset | 22 |
| 4.3 Optimización Bayesiana | 23 |
| 5 Algoritmo de Clasificación | 25 |
| 5.1 Red Neuronal Convolucional | 25 |
| 5.1.1 Preparación Del <i>Dataset</i> | 26 |
| 5.1.2 Funciones PotatoDataset y DataLoader | 30 |
| 5.1.3 Hiperparámetros Del Modelo | 32 |
| 5.1.4 Arquitectura del modelo | 34 |
| 5.1.5 Optimización De Hiperparámetros | 43 |
| 5.1.6 Comparación De Precisión De Modelos Implementados | 55 |
| 5.2 Clasificación Por Tamaño | 56 |
| 6 Prototipo | 58 |
| 6.1 Especificaciones principales | 58 |
| 6.2 Descripción de la maquina | 59 |
| 6.3 Esquema eléctrico | 60 |
| 6.4 Estructura | 61 |
| 6.4.1 Bombillos | 61 |
| 6.4.2 Sensor Fotoeléctrico | 62 |
| 6.4.3 Cámara Web | 62 |
| 6.5 Funcionamiento | 63 |
| 6.5.1 Funcionamiento Mecánico | 63 |
| 6.5.2 Implementación en Sistema Embebido | 64 |
| 7 Trabajo Futuro | 67 |
| Bibliografía | 68 |

1 Introducción

La agricultura ha sido uno de los pilares más importantes, tanto para el desarrollo económico como para el sustento de la sociedad, ya que gracias a esta los países pueden generar empleos y aumentar sus recursos económicos. En Colombia, según el censo nacional agropecuario realizado en el año 2014 por el DANE (Departamento Administrativo Nacional de Estadística), existen *2,7 millones* de productores residentes en el área rural y no rural del país, los cuales destinan la producción final de sus cultivos en un *74,3 %* a la comercialización en el territorio nacional [1].

En la investigación realizada por Ruth Liliana Goyeneche Ortegón [2], se encuentra que *"los productores de papa están sometidos a riesgos ocupacionales por la ejecución de tareas como el empleo de contaminantes químicos, el manejo de cargas, las posturas forzadas y el trabajo a la intemperie"*. Es por esto que con el avance tecnológico se ha buscado mejorar el sector de la agricultura, buscando facilitar a los agricultores las tareas que deben realizar, además de querer mejorar la calidad del producto, para garantizar que su comercialización tenga un mayor impacto en el mercado nacional.

La implementación de la visión artificial es uno de los principales avances tecnológicos en el sector agroindustrial, con esta se busca reducir las fallas en la intervención humana para mejorar el rendimiento y la eficiencia bajo la normativa INCONTEC la cual se debe tener en cuenta para la exportación de productos agrícolas. En este proyecto de investigación se tiene como objetivo implementar un algoritmo de clasificación con técnicas de visión artificial y un prototipo mecánico para la selección de tubérculos de papa según algunas características de calidad.

1.1. Objetivos

1.1.1. Objetivo General

Implementar un prototipo para la clasificación de características de calidad en los tubérculos de papa producidos en la región Andina de Colombia, mediante técnicas de visión por computadora.

1.1.2. Objetivos Específicos

1. Identificar a través de técnicas de visión artificial los rasgos de calidad y características físicas que se encuentran en los tubérculos de papa.
2. Evaluar el desempeño del prototipo propuesto para realizar la clasificación de los tubérculos de papa bajo las categorías de tamaño grande, mediana y pequeña establecidas por la norma *NTC 341 – 3*.
3. Comparar la precisión del modelo propuesto con modelos similares de clasificación

1.2. Alcances y Limitaciones

1.2.1. Alcances

Para la realización del proyecto se debe tener en cuenta que se enfocará en los tubérculos de papa, R12 y pastusa producidos en la región andina de Colombia. Se espera que el producto se encuentre limpio de suciedades como tierra y raíces, para llevar a cabo el análisis por técnicas de visión artificial. El análisis se enfocará en identificar las características de calidad del producto definidas por la norma *NTC 341 – 3*. La característica de daño será evaluada por daños mecánicos, defectos fisiológicos y daños causados por insectos. La característica de tamaño será evaluada con base en el diámetro del tubérculo en las categorías mediana, grande y muy grande establecidas por la norma. Se implementarán 4 arquitecturas pre entrenadas de redes neuronales artificiales para comparar el desempeño entre ellas, las redes utilizadas serán *AlexNet*, *ResNet18*, *VGG11* y *VGG19*. Finalmente, se propone un prototipo de banda transportadora para generar el desplazamiento de los tubérculos de papa por debajo de una cámara que realiza el análisis de visión artificial y extrae las características de calidad de la papa.

1.2.2. Limitaciones

El proyecto se encuentra limitado a los recursos de software y hardware, (computacionales), con los que cuentan los integrantes del proyecto y las plataformas de *Código Abierto* a las que se le puedan sacar provecho, para la implementación de los algoritmos que actualmente se encuentran en la literatura.

1.3. Justificación

La producción de papa en Colombia aporta el 3,3 % del Producto Interno Bruto (PIB), las siembras son de alrededor de 130 *mil* hectáreas y se cosechan cerca de 2,8 millones de toneladas [3]. Además, en Colombia la producción de papa genera anualmente alrededor de 264 *mil* empleos, aproximadamente 75 *mil* son trabajos directos y alrededor de 189 *mil* son indirectos [4]. El cultivo de la papa constituye el eje fundamental de la economía del país, en 283 municipios a nivel nacional, donde se involucran más de 90 *mil* familias principalmente en los departamentos de Boyacá, Cundinamarca, Antioquia y Nariño, los cuales concentran más del 85 % de la producción [5].

En la actualidad, uno de los retos que enfrenta el sector de la agricultura es el aumento en la calidad del producto necesaria para pasar a una etapa de comercialización. La automatización de procesos en la agricultura ayudan a mejorar el rendimiento y la eficiencia de la producción agrícola, sin embargo, en la etapa de comercialización también se puede implementar técnicas para mejorar la calidad del producto. Implementando técnicas de visión artificial, se desarrollará un prototipo que permita la selección de tubérculos de papa de forma automatizada y con mayor precisión, con el fin de mejorar el proceso actual que se lleva a cabo para la clasificación de calidad en los tubérculos de papa.

1.4. Descripción y Formulación Del Problema

La agricultura de Colombia es un componente fundamental en la economía del territorio, ya que juega un papel primordial en el desarrollo económico del país. Debido a que es la principal fuente de ingresos del área rural, hace un aporte relevante al desarrollo económico, la mitigación de la pobreza, y el desarrollo sustentable de Colombia. El sector papiculor en Colombia se caracteriza por tener poco desarrollo tecnológico que busca abastecer el consumo interno, sin mayor exploración en mercados internacionales. Frente a la coyuntura en la que se encuentra el sector, gracias a los retos que traen consigo los acuerdos comerciales firmados por el gobierno nacional, se requieren urgentes transformaciones que permitan aumentar su competitividad y lograr el crecimiento del sector.

Como consecuencia, en los últimos años, el sector agricultor de Colombia ha implementando herramientas tecnológicas que le permitan a los agricultores mejorar la calidad de sus productos para poder competir, tanto en el mercado local, como en el global. Pensando en esto, las diferentes técnicas de visión artificial han tenido un auge en la agricultura de precisión, ayudando a clasificar productos basándose en sus diferentes características de calidad sin importar su clase, sin embargo, las técnicas de visión artificial no han sido aprovechadas para mejorar el sector papiculor colombiano.

Las técnicas de visión artificial no se han implementado en la fase de almacenamiento de los tubérculos de papa en Colombia, la cual es importante ya que en esta fase es donde se separan los tubérculos que cumplen las condiciones de calidad de los que se encuentran defectuosos, ya que para la comercialización de este producto deben estar clasificados bajo la norma *NTC 341 – 3* [6].

¿Cómo construir un prototipo para la clasificación de tubérculos de papa mediante técnicas de visión artificial?

¿Qué tipo de proceso de clasificación permitirá agrupar los tubérculos de papa según sus características físicas y "patologías" mediante técnicas de visión artificial?

2 Marco Conceptual

2.1. Agricultura De Precisión

La agricultura de precisión (AP), es definida por *Marote (2010)*, como *"un concepto agronómico de gestión de parcelas agrícolas, basado en la existencia de variabilidad en campo"*. La cual para ser implementada requiere el uso de sistemas de posición Global (GPS), imágenes aéreas, sensores y satélites, para obtener datos del cultivo que permitan el análisis y entendimiento de dichas variaciones. El objetivo de la AP es evaluar la precisión y predicción de la producción de los cultivos, teniendo en cuenta entradas necesarias como la densidad óptima de siembra, estimar fertilizantes, precipitaciones, entre otras, con el fin de obtener mayor exactitud en los resultados, teniendo en cuenta que las bases fundamentales de la AP son la variabilidad espacial y temporal del terreno y los factores que pueden afectar la producción del cultivo [7].

2.2. Redes Neuronales Artificiales

Las redes neuronales artificiales (RNA), son definidas por *Salas, R. (2004)*, como *"un esquema de computación distribuida inspirada en la estructura del sistema nervioso de los seres humanos "*. Las RNA son métodos que ofrecen las herramientas necesarias para modelar de forma eficiente y eficaz problemas con una alta complejidad, esto se logra mediante la implementación de algoritmos de aprendizaje supervisado o no supervisado. El objetivo de las RNA es ajustar su arquitectura y parámetros en busca de minimizar su función de error, de tal manera que la salida generada sea lo más cercana a la verdadera salida según la entrada asignada [8].

2.3. Convolución Dilatada

Se trata de un método que busca ampliar la entrada insertando huecos entre sus elementos consecutivos. Siendo igual que una convolución normal, pero con la diferencia de que omite pixeles, con el fin de cubrir más área de la entrada, dando un campo de visión más amplio con el mismo coste computacional, por medio de un parámetro llamado factor de dilatación

l , indicando cuánto se expande la entrada [9]. La Figura 2-1 muestra un ejemplo de la implementación de la convolución dilatada.

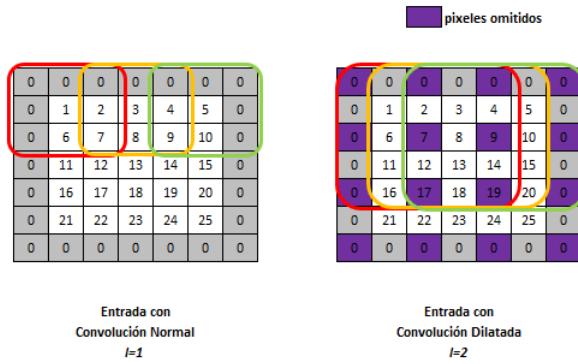


Figura 2-1: Ejemplo Convolución Dilatada
Fuente: Elaboración Propia

2.4. Overfitting

Un problema al momento de entrenar redes neuronales, es el *over-fitting*, el cual se da cuando la red neuronal en un momento específico del ciclo de entrenamiento no muestra un progreso en la capacidad de resolver problemas. Si no que solamente aprende una regularidad aleatoria dentro del conjunto de patrones de entrenamiento, esto quiere decir que la red está sufriendo un exceso de flexibilidad a la hora de agrupar los datos [10].

2.5. Underfitting

Es lo contrario al *over-fitting*, siendo este cuando el modelo creado es incapaz de percibir variabilidad alguna en los datos. Dando como resultado que el clasificador resultante no tenga la capacidad de realizar una predicción aceptable [10].

2.6. Data Augmentation

El Data augmentation consiste en la transformación de los datos existentes, para crear un *dataset* con mayor cantidad de datos diferentes. El objetivo de esta herramienta es generar nuevos datos que puedan ser añadidos al conjunto ya existente, y así contar con más muestras para un mejor desempeño del algoritmo. Además de esto, también es implementado para reducir el *overfitting*, ya que al añadir más información al conjunto de

entrenamiento se evita que el modelo se sobre ajuste, normalmente esta técnica es implementada en problemas de clasificación donde el Dataset base del algoritmo no cuenta con una cantidad significativa de muestras para el entrenamiento de la red neuronal [11].

2.7. Dropout

El Dropout es una técnica de regularización implementada en el entrenamiento de redes neuronales, el fin de esta técnica es evitar que las neuronas que se encuentran en la red memoricen parte de la entrada, ya que si la red memoriza la entrada, se presentará un sobre ajuste en la red. El Dropout cumple su objetivo desactivando de manera aleatoria un porcentaje de las neuronas ubicadas en cada capa oculta [12].

2.8. Filtro de mediana

Es una tecnología que procesa señales no lineales. El valor del ruido que proviene de la imagen digital o la secuencia es sustituida por el valor mediano de la vecindad (máscara). Los pixeles de la máscara se clasifican en el orden de sus niveles de gris, el valor mediano del grupo es almacenado para sustituir el valor del ruido [13].

2.9. Imagen Binaria

Una imagen binaria puede ser definida como una función de dos variables discretas $[m, n]$, las cuales pueden tomar dos valores, ‘0’ o ‘1’, de ahí su nombre de imagen binaria, estos valores pueden ser asignados dependiendo el nivel de gris que contenga la imagen (las imágenes binarias poseen dos niveles: blanco y negro). Las imágenes binarias son implementadas en visión artificial para realizar tareas de preprocesamiento de imágenes, el cual consiste en la eliminación del ruido y la simplificación del objeto que se encuentra en la imagen, además de esto, también se implementa para la descripción cualitativa de los objetos esto quiere decir que una imagen binaria facilita el cálculo de áreas, perímetros, diámetros, etc.[14].

2.10. Max-pooling

El pooling es conocido como una operación la cual permite analizar imágenes por regiones, el objetivo de esta operación es obtener la información más importante que se encuentra en las imágenes, esto se logra reduciendo la cantidad de datos que están presentes entre

una capa y otra, una ventaja de implementar el max-pooling en las redes neuronales es que permite facilitar el procesamiento de las imágenes y el entrenamiento de la red sin perder información en el proceso.

Un operador max-pooling, se puede usar para la reducción del muestreo sobre las bandas de salida convolucionales, haciendo una reducción de la variabilidad. Esto lo realiza a través de un valor máximo como se muestra en la Ecuación 2-1, dentro de un grupo de activaciones A. La banda máxima agrupada m está compuesta por f filtros relacionados $p_m = [p_{1m}, \dots, p_{fm}] \in A^f$.

$$p_{f,m} = \max(h_{f,(m-1)N+r}) \quad (2-1)$$

Donde $N \in (1, \dots, R)$ representado como un desplazamiento de agrupación cuando $N < R$. La capa de agrupación disminuye la dimensionalidad de salida de K bandas convolucionales a $M = (K - R)/N + 1$ bandas, que han sido agrupadas teniendo como resultante $p = [p_1, \dots, p_M] \in R^{M,J}$ [15], un ejemplo de max-pooling se puede observar en la Figura 2-2.



Figura 2-2: Ejemplo Max-Pooling

Fuente: Elaboración Propia

2.11. Padding

El padding es un método utilizado con el fin de añadir píxeles de valor cero a los bordes de la imagen original, esto con el objetivo de que la imagen resultante al realizar la convolución sea del mismo tamaño que la original. Se debe tener en cuenta que si el padding no es implementado y se realiza la convolución, la imagen resultante será de un tamaño menor a la original, además este método permite la creación de redes más profundas y la extracción de características más específicas durante el entrenamiento.

El más usado es el *zero padding*, debido a que mantiene la misma dimensionalidad al aplicar convoluciones. Por otro lado, permite a las redes neuronales convolucionadas codificar

información de posición absoluta, a pesar de capas de agrupamiento en su arquitectura [16], un ejemplo de padding se puede observar en la Figura 2-3.

| Imagen | | | | | |
|--------|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | | | | | 0 |
| 0 | | | | | 0 |
| 0 | | | | | 0 |
| 0 | | | | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Figura 2-3: Ejemplo Zero-Padding

Fuente: Elaboración Propia

2.12. Procesamiento de imágenes

El procesamiento de imágenes, se puede definir como la implementación de prácticas (disminuir el ruido, mejorar el contraste, ajustar el brillo, suavizar bordes, etc.), que buscan modificar ciertas características de la imagen con el fin de tener una mejor visualización de la misma, para realizar su posterior análisis de una manera más eficiente [17].

El procesamiento de imágenes no tiene como objetivo incrementar la información que se puede obtener de las imágenes, pues el enfoque de esta técnica es resaltar las características más significativas que se encuentren en la imagen para que esta puede ser procesada de la mejor manera [17].

2.13. Stride

Es un parámetro que especifica cuántos píxeles se traslada horizontalmente y verticalmente, mientras es convolucionada la imagen. En algunas arquitecturas el *stride* se utiliza en vez de *max pooling* para reducir el tamaño de la capa [18], un ejemplo de stride se puede observar en la Figura 2-4.

| | | | | | | |
|---|----|----|----|----|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 0 | |
| 0 | 5 | 6 | 7 | 8 | 0 | |
| 0 | 9 | 10 | 11 | 12 | 0 | |
| 0 | 13 | 14 | 15 | 16 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | |
|---|----|----|----|----|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 0 | |
| 0 | 5 | 6 | 7 | 8 | 0 | |
| 0 | 9 | 10 | 11 | 12 | 0 | |
| 0 | 13 | 14 | 15 | 16 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | |
|---|----|----|----|----|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 0 | |
| 0 | 5 | 6 | 7 | 8 | 0 | |
| 0 | 9 | 10 | 11 | 12 | 0 | |
| 0 | 13 | 14 | 15 | 16 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figura 2-4: Convolución con Zero Padding y Stride > 1

Fuente: Elaboración Propia

2.14. Aprendizaje por transferencia

El aprendizaje por transferencia, es definido por Lisa Torrey (2010) como "la mejora del aprendizaje en una nueva tarea, a través de la transferencia de conocimientos de una tarea relacionada que ya se ha aprendido". El objetivo del aprendizaje por transferencia es mejorar a un alumno con la transferencia de información de otro dominio relacionado. La mayor parte de algoritmos de aprendizaje automático fueron creados para ser empleados en tareas individuales [19].

2.15. Umbralización De una Imagen Digital

Es un método que divide una imagen en grupos parecidos según un conjunto de criterios ya definidos. Podemos encontrar varias técnicas de *Thresholding* teniendo un enfoque primordial de la segmentación, especialmente en aplicaciones en las que la velocidad es fundamental en el proceso. *Thresholding* puede ser adaptativo al utilizarse diferentes umbrales para distintas regiones de la imagen [20].

2.16. Visión artificial

La visión artificial, es conocida por intentar replicar la capacidad de algunos seres vivos para visualizar una imagen, entenderla y actuar con base en lo observado. Las diferentes aplicaciones industriales que requieren el uso de técnicas de visión artificial ha tenido un crecimiento considerable, es por esto que esta disciplina debe mantenerse en un continuo desarrollo y evolución de nuevos algoritmos y aplicaciones. Esto puede considerarse como una consecuencia a que en la actualidad existe una gran cantidad de contenido visual como imágenes o videos [21].

3 Estado del arte

3.1. MatLab

Se tiene el trabajo de *J. porras* [22], en donde se menciona que la implementación de MATLAB® como una herramienta de procesamiento de imágenes, radica en su facilidad para realizar cambios a las imágenes, es por esto que diseñó un sistema para la clasificación de objetos con base en su forma y color, usando métodos de visión artificial en MATLAB®, con el uso de la librería *ufm.dll*, para capturar y procesar imágenes. Se pudo evidenciar en los resultados del proyecto que *J. porras* implementó, que utilizando técnicas de visión artificial se puede automatizar un proceso de producción con el fin de reducir tiempos y costos de operación en una planta de producción industrial. El artículo de *C. Nandi* [23] presenta una investigación, en donde, se consiguió calcular el tamaño de diferentes mangos, estimando el área cubierta en una imagen binaria, (imagen digital que tiene únicamente dos valores posibles para cada píxel), con base en el número de píxeles, luego de ser procesadas las imágenes, se clasificaron implementando un algoritmo basado en *fuzzy logic*, teniendo en cuenta 5 variedades diferentes de mangos y como referencias el color de la cascara, tamaño, defectos superficiales, forma, firmeza, peso y olor.

3.2. Redes Neuronales Artificiales

Para la clasificación de imágenes usando técnicas de visión artificial, se tiene en primer lugar la literatura acerca de las *RNA* (*Redes Neuronales Artificiales*). Las *RNA* abarcan la temática del *Deep Learning* [24], definido por *LeCun* (2015), como *“el aprendizaje profundo permite que modelos computacionales compuestos por múltiples capas de procesamiento aprendan representaciones de datos con múltiples niveles de abstracción”*. Estos métodos permiten la implementación de aplicaciones que permiten el reconocimiento visual y detección de objetos así como la predicción de datos estadísticos.

El trabajo de *L. Pencue-Fierro y J. León Téllez* [25], aborda una investigación hecha en Perú, en donde se aplicó visión artificial usando *RNA* como método de clasificación de las principales características extraídas de las frutas, que son derivadas del análisis

de las superficies, tanto en su contenido cromático como en la cantidad y distribución de defectos externos. La revista *Multimedia Tools and Applications* [26] presentó una investigación, donde desarrollaron un sistema de visión artificial que usa un enfoque de categorización simplificado con un elevado índice de exactitud. El propósito del sistema es clasificar los granos de trigo de las especies *triticum aestivum* y *triticum durum* según sus propiedades visuales, usando una *RNA* del tipo *MLP*, (*Multilayer Perceptron*); las imágenes se obtienen por medio de una cámara que captura las propiedades de tamaño, color y textura de cada grano con el objeto de que sirvan de acceso al procedimiento de categorización. El trabajo publicado por *Ksh. Robert Singh y Saurabh Chaudhury* [27], en donde se propone el uso de redes neuronales *BPNN*, (*Back Propagation Artificial Neural Networks*), como método de clasificación y la descomposición mediante ondículas, (Tipo especial de transformada matemática que representa una señal en términos de versiones trasladadas y dilatadas de una onda finita), para clasificar los granos de arroz. El modelo de clasificación implemento una red neuronal *BPNN* de cuatro capas la cual presento mejores resultados en comparación con otros métodos.

Usando el método de clasificación por *RNA*, también se tiene el trabajo hecho por *Krzysztof Koszela* [28], donde se presenta una forma para garantizar la correcta clasificación de los productos y reducir las pérdidas durante su almacenamiento. La investigación abarca esfuerzos centrados en la evaluación sensorial de patatas, con el análisis de imágenes por ordenador y la modelización neuronal *RNA*. El objetivo de este estudio fue desarrollar un método para asistir a la identificación de cualquiera de las variedades y la turgencia de los tubérculos de patata, (Fenómeno que ocurre cuando una célula se dilata debido a la presión ejercida por los fluidos y por el contenido celular sobre las paredes de la célula), llevado a cabo sobre la base de los datos gráficos codificados en forma de imágenes digitales, obtenidos mediante algoritmos que interpretan los descriptores de imagen.

3.3. Super Vector Machine

El método de clasificación *SVM* (*Super Vector Machine*) revisado en la literatura, como el trabajo de *Tao Liu* [29], en donde realizaron un proceso para analizar granos de arroz. El procedimiento usa 4 fuentes de luz para crear la sombra del grano en 4 direcciones; la diferencia en medio de las siluetas de los granos llenos y no llenos, se evalúa por medio del estudio de imágenes y un clasificador *SVM*. El análisis se hace mediante el uso de imágenes *RGB* (Red-Green-Blue), de los granos con las siluetas, luego se segmentan desde la imagen binaria, para sustraer información como el sector del grano y de la sombra. En el documento publicado por *Chia-Lin* [30], se menciona un método para clasificar

plántulas (Embrión ya desarrollado como consecuencia de la germinación de una semilla), sanas e infectadas. Consiste en el análisis de imágenes mediante un escáner y el proceso de clasificación utilizando *SVM*, en donde se hace uso de dos clasificadores, el primero distingue entre las plantas sanas y contaminadas, por otra parte, el segundo mide los niveles de contaminación. En el trabajo de *Rillian Diello y Lucas Pires* [31], se propuso un método de detección automática de enfermedad en cultivos de soja, el cual se basa en descripciones locales, conocido como el método *BOV* (Bag of Values), luego de escanear la hoja. Se obtuvo a través de los vectores de entrada una clasificación en dos categorías, enfermo y sano, a partir de un clasificador *SVM*. El trabajo de *Chengming Sun y Tao Liu* [32], propone un sistema para analizar el porcentaje de granos en el arroz que se encuentran en condiciones para ser distribuidos. El algoritmo se encarga de la división de los granos de manera automática. Tras la segmentación, es viable obtener el número de granos presentes en la imagen y la información específica, la exactitud del procedimiento puede verse afectada si el germen no se extrae del todo, por esa razón, el sistema detecta la viable región de germen y estima esta información usando *SVM*.

3.4. Otros métodos

El trabajo hecho por *Tao Liu y Wen Chen* [33], puso en práctica un método para controlar la población de pulgones, (Familia de insectos hemípteros), en el trigo, usando los métodos *SVM*, el algoritmo *MSER* (*Maximally Stable Extremal Regions*), y el *HOG* (*Histogram of Gradients*). El uso de estos 3 métodos se conoce como *SMH* (Unión entre *SVM*, *MSER* y *HOG*), se basa en el análisis de imágenes tratadas a partir de unos parámetros, con la finalidad de detectar la presencia y/o ausencia de pulgones, se hizo uso de este método a partir del color y la densidad de población. Al comparar este método con otros cinco comúnmente utilizados, los resultados presentaron un rendimiento superior en la identificación de pulgones.

El trabajo publicado por *Rodica Sobolu* [34], propone un algoritmo de clasificación automática de patatas. Se realizaron dos tipos de clasificación: una en función del tamaño de las patatas y otra en función de su calidad. La segmentación de las zonas defectuosas se hizo mediante métodos como *global thresholding*, para extraer características morfológicas y estadísticas de las zonas segmentadas. Estas características se eligieron como entradas para los algoritmos de clasificación usando métodos como *SVM*, *Decision Tree* y *LDA* (*Linear Discriminant Analysis*), implementados en MATLAB® *Classification Toolbox*. Se llegó a la conclusión de que el método de *SVM* ha clasificado las patatas según su tamaño con una mayor tasa de éxito y en la clasificación por calidad, el método *LDA*.

Se encuentran unos métodos poco comunes en la literatura, como por ejemplo los presentados en el artículo de *Alberto Martínes Rodríguez* [35], donde se identificaron objetos en movimiento mediante la ayuda de la visión artificial y la transmisión de datos a un brazo robótico implementando *C++* y *Open CV*, usando el código de cadena, (Actualmente el grupo de reconocimiento de patrones e inteligencia artificial aplicada), para calcular las características de los objetos por color y forma. También, el artículo hecho por *L. Han y M. S. Haleem* [36], propone un algoritmo de detección automática de enfermedades en los cultivos, utilizando para este proceso el algoritmo *MCW* (*Marker-Controlled Watershed*) para separar el fondo de la imagen, de la hoja del cultivo. El *SLIC* (*Simple Linear Iterative Clustering*), se utilizó para obtener las características presentadas por la enfermedad sobre la planta y por último, la clasificación y textura se obtienen a través de *GLCM* (*Gray Level Co-occurrence Matrix*), el modelo de clasificación fue el *SVM*, este método propuesto presentó un mayor rendimiento.

El trabajo presentado por *Michael Barnes y Tom Duckett* [37], cuyo objetivo de investigación es introducir un método automático de detección de manchas en imágenes digitales de patatas. El sistema desarrollado es entrenable, de modo que pueda trabajar con diferentes variedades de patatas y variaciones en las estaciones, condiciones de iluminación, etc. Otro objetivo, pensando en su posible implantación en entornos industriales, es permitir el procesamiento de imágenes en tiempo real, posiblemente mediante la construcción de *Minimalist Boosted Classifier*, que extraigan un subconjunto mínimo de todas las características que optimicen el rendimiento de la detección con el menor coste computacional posible.

3.5. Clasificación De Papa En Colombia

La papa es un tubérculo conocido mundialmente, con una enorme producción y valores nutritivos, contiene muchos micro nutrientes primarios y cruciales como vitamina, vitamina B6, niacina, ácido fólico, potasio, hierro y magnesio, que es una fuente importante de carbohidratos, vitaminas y minerales para el ser humano. Según el plan de ordenamiento de la producción de papa en el país, presentado por el Ministerio de Agricultura y Desarrollo Rural en 2019 [38], en Colombia se siembran 130 mil hectáreas y se producen cerca de 2,8 millones de toneladas de papa al año. Los departamentos de Boyacá y Nariño constituyeron al año 2018 un total de 24 % y 21 % de participación en la producción. En Boyacá el 95 % de la producción se destina al consumo fresco y el 5 % al procesamiento industrial. En Nariño los porcentajes son de 90 % para el consumo fresco y 10 % para el procesamiento industrial.

El procesamiento industrial de la producción de papa genera un valor agregado en comparación a la papa que se destina al mercado en fresco. El valor agregado depende del nivel en términos de lavado del producto, selección, clasificación y empaque. El plan de ordenamiento de la producción de papa en el país indica que un 25 % de la papa que se comercializa en fresco se comercializa con un valor agregado. El informe agrega también que ”*anualmente se destina aproximadamente el 6% de la producción nacional al procesamiento industrial. Existen en el país cerca de 50 industrias dedicadas a la actividad de procesamiento de la papa, con diferente capacidad, diferentes niveles de desarrollo tecnológico y variada presencia en el mercado*”.

El control de calidad actualmente se realiza de forma manual, realizado por personas, clasificando según su defecto y tamaño, esta acción tiene algunas desventajas: es subjetiva, laboriosa y lleva bastante tiempo, llevando a que la clasificación decaiga con el tiempo. En el trabajo publicado por *Jender Mauricio Buitrago* en 2017 [39], se presenta un estudio de las prácticas agrícolas en el cultivo y comercialización de papa en la vereda Rechíniga en el departamento de Boyacá. Uno de los métodos utilizados, una vez se hace la recolecta del cultivo, es descrito como el método de bloque, que consiste en amontonar toda la papa producida en surcos y después los obreros se dividen el trabajo para clasificar el cultivo por tamaño, uno de los obreros se encarga de seleccionar la papa gruesa y otro la pareja para finalmente, empacarla en costales.

El estudio también menciona, que los campesinos de la vereda de Rechíniga, tienen en cuenta el estado de la papa, el autor menciona que ”*en el proceso de extracción hay algunas que son tajadas por la herramienta utilizada para dicho proceso y se deben escoger y apartar para que no dañen las demás papas debido a que están más propensas a descomponerse fácil y rápidamente*”.

Según el plan de ordenamiento de la producción de papa en el país, ”*la papa se comercializa en Colombia a través de un sistema considerado como altamente ineficiente tanto por el elevado número de niveles de intermediación como por la escasa o nula agregación de valor*”, esto genera altos costos en los procesos de producción de materia prima en el procesamiento industrial de la papa. Debido a esto, desde el Ministerio de Agricultura y Desarrollo Rural se han presentado informes y estrategias para incluir la tecnología en los procesos de producción del sector papicultor del país.

4 Diseño Del Algoritmo de Clasificación

Se diseñó un algoritmo de clasificación utilizando Redes Neuronales Convolucionales y técnicas de visión artificial con la librería *OpenCV*, que es una biblioteca de software de código abierto para realizar visión por computador y aprendizaje automático, para clasificar las características de tipo, daño y tamaño en tubérculos de papa. El Procedimiento que se llevó acabo se presenta en la Figura 4-1. Las características de tipo y daño serán analizadas con la red neuronal artificial y, la característica de tamaño, será analizada con el uso de *OpenCV*. De esta manera, ambos análisis dan como resultado la clasificación de la papa según las características de calidad definidas.

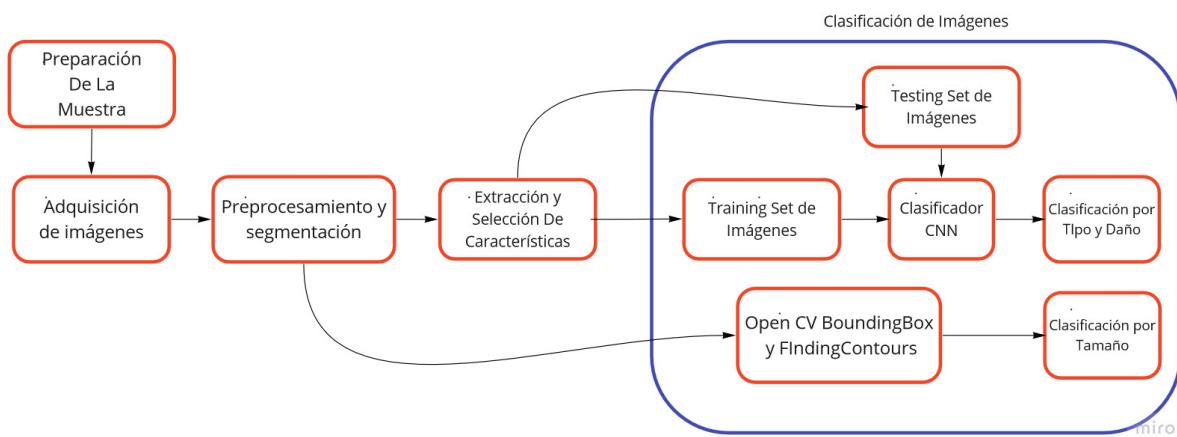


Figura 4-1: Arquitectura del Sistema de Clasificación

Fuente: Elaboración Propia

La estrategia propuesta para realizar la clasificación de tubérculos de papa en este documento, consiste desde la adquisición de muestras y creación de un *dataset* de tubérculos de papa, hasta el diseño de un algoritmo de clasificación con Redes Neuronales, utilizando la librería *Pytorch*, que es una biblioteca de software para la aplicación de aprendizaje profundo, y la librería de *OpenCV* en el lenguaje de programación de *Python*. El algoritmo de clasificación será implementado en un prototipo de banda transportadora, que permita el movimiento del elemento por debajo de una cámara, para ser analizado y clasificado. Los

sistemas de visión artificial utilizados en inspecciones visuales, permiten que la operación sea más eficiente si la producción requiere que el proceso aumente en velocidad, cantidad de producción y se encuentre en funcionamiento las 24 horas del día o la repetibilidad de las medidas [40].

Según la norma NTC 341 – 3, se considera que el daño en una papa es ”*una lesión o alteración de magnitud variable que presentan los tubérculos*”, se establecen 5 tipos de daños, sin embargo, los tubérculos seleccionados para el proyecto pertenecen a los siguientes tipos de daño, definidos según la norma.

- Daños Mecánicos: Lesiones causadas por agentes físicos.
- Daños y Defectos Fisiológicos: Lesiones o defectos que provienen de alteraciones no patogénicas de origen interno o externo.
- Daños Causados Por Insectos: Se presentan en forma de túneles o galerías de diámetro y longitud variable causados por insectos y moluscos.

De igual forma, la norma define el diámetro como ”*la máxima distancia tomada en ángulos rectos sobre el eje más largo del tubérculo*”, se establecen 4 tamaños según el diámetro, sin embargo, los tubérculos seleccionados para el proyecto serán clasificados por los tamaños presentados en la Tabla 4-1, establecidos según la norma.

| Denominación | Diámetro en mm |
|--------------|----------------|
| Muy Grande | Mayor a 90 |
| Grande | 65 - 90 |
| Mediana | 45 - 64 |

Tabla 4-1: Clasificación Por Tamaño del Tubérculo

Los tubérculos que serán analizados en el prototipo deben estar limpios. La norma define una papa limpia como ”*los tubérculos libres de tierra u otras impurezas adheridas*”. Debido a que una de las características que indica el tipo de tubérculo es el color, en este caso, se usarán los tubérculos de la variedad *R12* y *Pastusa*, es por esto que se deben tener los tubérculos limpios.

4.1. Preparación y Adquisición De Las Imágenes

Se seleccionó un total de 592 tubérculos de papa, que manualmente fueron clasificados en tres categorías definidas. Las etiquetas de cada imagen se encuentran dentro del nombre de cada archivo, que corresponde al *metadata* de cada foto tomada. Definido de la siguiente manera:

- *Tipo*: Pastusa o R12 [0, 1]
- *Daño*: Buena y Defectuosa [0, 1]
- *Tamaño*: Muy grande, Grande y Mediana [0, 1, 2]
- *Numero de la Imagen* Etiqueta asignada por la cámara.

Las fotos tomadas para la creación del *dataset*, poseen un tamaño de (3168, 4752) pixeles. Fueron tomadas con una cámara profesional de referencia *Canon EOS 50D*, que tiene 15.1 mega pixeles de resolución y se puede ajustar la sensibilidad *ISO* desde 100 hasta 3200. La cámara fue configurada con *ISO-800* que corresponde al parámetro de sensibilidad del sensor de ruido de la cámara, una velocidad de obturación de $\frac{1}{640}$ segundos, que corresponde al dispositivo que controla el tiempo en el que la luz incide sobre el sensor de la cámara, y finalmente una apertura de diafragma de $F = 7.1$ que corresponde a la apertura del lente que deja pasar la luz [41], tener en cuenta que a mayor apertura de diafragma, menor luz se deja pasar. Se construyó una caja con iluminación fija la cual se muestra en la Figura 4-2 usando bombillos de luz blanca de 6500 Kelvin de temperatura de color, para que la iluminación en todas las fotos tomadas fuera uniforme según la norma española *UNE-EN 12464-1* [42].



Figura 4-2: Récamara Para la Toma de Fotos
Fuente: Elaboración Propia

La cámara para la toma de las fotos fue fijada a un trípode ubicado a 30 cm desde el lente hasta la parte superior de la muestra, ya que al tener tubérculos de diferentes tamaños, se puede garantizar que los píxeles que definen el perímetro de las muestras son uniformes sin importar el tamaño del tubérculo. Utilizando la librería *Pandas* de *Python*, se creó un *Dataframe* que contiene el metadata de cada imagen. En la Tabla 4-2 se puede apreciar la distribución del *MetaData* de las imágenes almacenadas en el *Dataset*, donde se toman cinco imágenes al azar como muestra. A partir de esta tabla se crea un archivo *.CSV* donde se encuentra la información de las 592 imágenes.

| Tipo | Daño | Tamaño | Filename |
|---------|------------|---------|----------------|
| R12 | Defectuosa | Grande | 1_1_1_075.JPG |
| PASTUSA | Buena | Grande | 0_0_1_1973.JPG |
| R12 | Buena | Grande | 1_0_1_2054.JPG |
| PASTUSA | Buena | Mediana | 0_0_2_2042.JPG |
| PASTUSA | Buena | Grande | 0_0_1_1955.JPG |

Tabla 4-2: MetaData de 5 Imágenes de Muestra

Una vez la información de cada imagen fue guardada en el archivo *metadata.csv*, se agruparon las posibles combinaciones de las características de *Tipo* y *Daño*, para definir las *clases* dentro de la red neuronal. Para realizar este proceso, se creó una condición dentro del *Dataframe*, que contiene el *metadata* para generar una columna adicional con la información de la Tabla 4-3.

| PASTUSA | R12 | Buena | Defectuosa | Clase | Cantidad De Muestras |
|---------|-----|-------|------------|---------|----------------------|
| X | | X | | CLASE 1 | 140 |
| X | | | X | CLASE 2 | 110 |
| | X | X | | CLASE 3 | 106 |
| | X | | X | CLASE 4 | 236 |

Tabla 4-3: Clases Definidas

De esta forma se definieron las 4 *clases* que serán entrenadas en la red neuronal, para el posterior proceso de clasificación por visión artificial.

4.2. Distribución del Dataset

La librería *Plotly*, que permite realizar gráficas en *Python*, presenta la distribución en porcentaje de las imágenes pertenecientes a cada característica definida en el *dataset* como se muestra en la Figura 4-3.

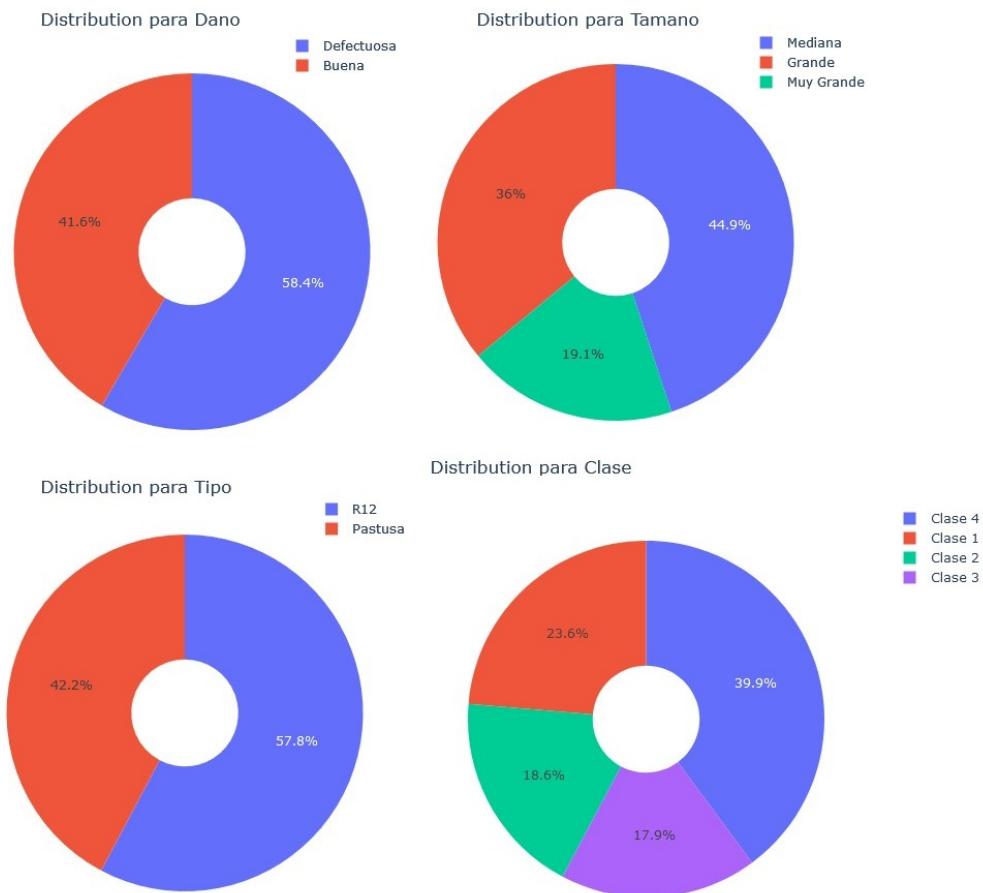


Figura 4-3: Distribución del Dataset Según Sus Características Y Clases Definidas

Fuente: Elaboración Propia

Se definieron los conjuntos de *entrenamiento* y *validación*, a partir del archivo *metadata.csv* de la Tabla 4-2. Se utilizó 80% para *entrenamiento* y 20% para *validación*. Se crearon los archivos *train.csv* y *test.csv* que contienen el *metadata* y la dirección de las imágenes separadas en el respectivo 80% y 20%.

4.3. Optimización Bayesiana

La mayor parte de modelos implementados en *machine learning* poseen una serie de parámetros que no pueden ser aprendidos de los datos, es por esto que deben ser establecidos antes del entrenamiento. Estos parámetros son conocidos como hiperparámetros.

Dos de las estrategias más empleadas para probar diferentes combinaciones de hiperparámetros y evaluarlas mediante métodos de validación, son *grid search* y *random search* [43]. En la estrategia de *grid search*, los valores estudiados de cada hiperparámetro, son distribuidos uniformemente dentro de un rango delimitado por el analista. En la estrategia de *random search*, los datos son aleatorios dentro de ese rango como se muestra en la Figura 4-4.

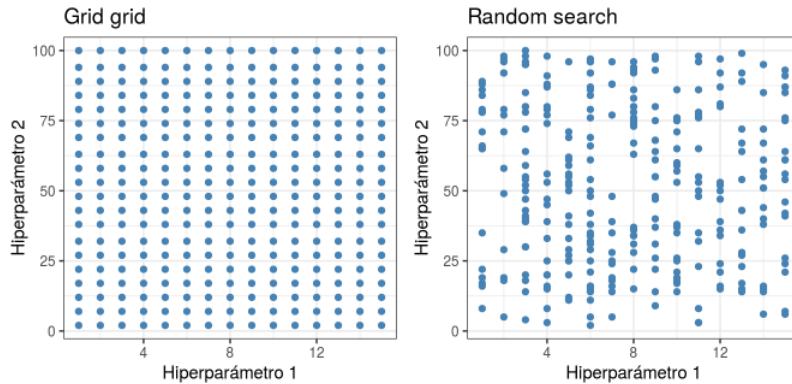


Figura 4-4: Métodos de Optimización De Hiperparámetros
Fuente: Amat, J. (2020)

A pesar de que las dos estrategias son válidas y se obtienen buenos resultados, sobretodo cuando se tiene criterio para acotar el rango de búsqueda, poseen una carencia similar: ninguna de las dos tiene en cuenta los resultados obtenidos hasta el momento, lo cual impide que se focalicen en la búsqueda de las regiones de mayor interés y evitando las regiones innecesarias.

Otro método utilizado para la búsqueda de los hiperparámetros de un modelo es la optimización bayesiana, definido por Peter Frazier (2018) [44] como *crear un modelo probabilístico en el que el valor de la función objetivo es la métrica de validación del modelo, en este caso, la precisión*. Con este método, se logra que la búsqueda se vaya redirigiendo en cada iteración hacia las regiones de mayor interés. Esto con el fin de reducir el número de combinaciones de los hiperparámetros con los que el modelo es evaluado, seleccionando únicamente los mejores candidatos. Esto significa que, la ventaja frente a las estrategias

descritas anteriormente, se maximiza cuando el espacio de búsqueda es muy amplio o la evaluación del modelo es muy lenta.

Para realizar la optimización bayesiana, se utilizó el módulo para *Python Botorch* [45]. Es el motor de optimización de *Ax-Services* compatible con *Pytorch*, que admite algunas funciones de minimización y maximización, como la mejora esperada (*EI*), la probabilidad de mejora y el límite superior de confianza. *EI* es una función de adquisición que recompensa la evaluación del objetivo $f(x)$, basándose en la mejora esperada en relación con el mejor momento f^* . La Ecuación 4-1, define el comportamiento de la mejora esperada.

$$EI(x) = \max(f(x) - f^*, 0) \quad (4-1)$$

El resultado de la parametrización, se selecciona y se evalúa en el siguiente paso. Una vez que se ha explorado adecuadamente el espacio de los parámetros, la mejora esperada se estrecha naturalmente en las ubicaciones donde hay una alta probabilidad de un buen valor objetivo. Este proceso se realiza por 20 etapas, en las que se varían los parámetros de acuerdo a lo descrito anteriormente.

5 Algoritmo de Clasificación

5.1. Red Neuronal Convolucional

La arquitectura general de una red neuronal convolucional (*RNC*) se muestra en la Figura 5-1, que consiste en varias capas de procesamiento de imágenes en donde se realizan convoluciones (operación matricial en la entrada de una red que extrae características y disminuye la complejidad del modelo) [46]. Se realizaron pruebas con las arquitecturas de (*RNC*) pre entrenadas *Alexnet*, *Resnet18*, *VGG11* y *VGG19* y se implementó la optimización bayesiana en ciertos hiperparámetros de las arquitecturas existentes, para mejorar los resultados.

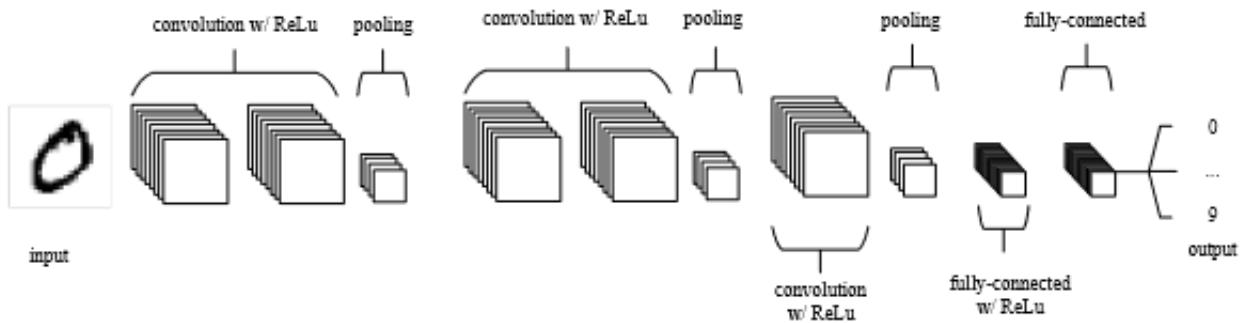


Figura 5-1: Arquitectura General De Una *RNC*

Fuente: O'Shea, K. (2015)

El procedimiento realizado para implementar la red neuronal convolucional se aprecia en la Figura 5-2, donde se indican los pasos a seguir previo a utilizar alguna de las arquitecturas pre entrenadas. Cada ítem será descrito en las próximas secciones.

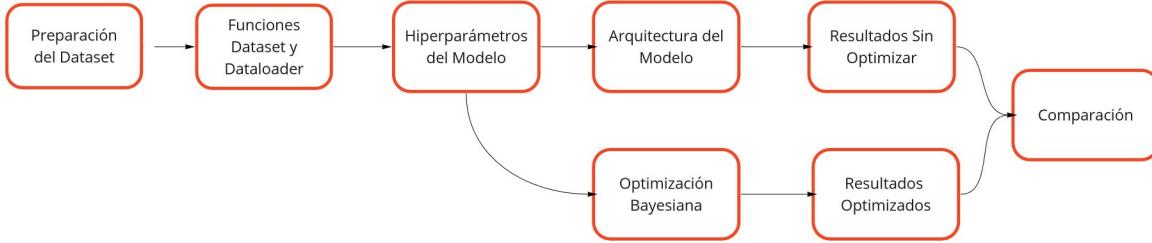


Figura 5-2: Procedimiento Para Implementar La *RNC*

Fuente: Elaboración Propia

5.1.1. Preparación Del Dataset

Para realizar un correcto entrenamiento del modelo de red neuronal convolucional es necesario aplicar un preprocesamiento a las imágenes, para que el modelo extraiga características pertinentes a las clases definidas. En este apartado se explica el procedimiento que se lleva a cabo para el preprocesamiento de imágenes y los diferentes filtros aplicados para realizar el *Data Augmentation*.

PREPROCESAMIENTO DE IMÁGENES

Usando el módulo *transforms.compose()*, de la librería *Pytorch*, se pueden encadenar unas determinadas funciones que aplican diferentes tipos de filtros y transformaciones al *Dataset* [47]. De esta forma, se crea la tarea de segmentación debido a que las transformaciones aumentan en consideración el tamaño original del *Dataset*, facilitando así, la extracción de características.

En la Tabla 5-1 se observa una parte de las transformaciones de imágenes disponibles en el módulo *torchvision.transforms.compose* de la librería *Pytorch*. En el caso del *Dataset* de papas creado, se deben tener en cuenta únicamente las transformaciones cuyo resultado sea relevante para la extracción de características de la imagen, por ejemplo, como se explicó anteriormente, se va a clasificar el tubérculo de papá en tipo y daño, con la red neuronal, por este motivo, el color es una característica que define si la papa pertenece a la clase *R12* o *Pastusa* y, en algunos casos, si se encuentra con algún tipo de daño, por este motivo, los filtros que transforman la imagen en escala de grises no son relevantes y podrían afectar el rendimiento del algoritmo.

| Image Transform | Use | Image Transform | Use |
|-----------------|--------------------------------------------------------------------------------------|--------------------|--------------------------------------------------------------------------------------|
| CenterCrop | Recorta la imagen en el centro | RandomResizedCrop | Recorta una porción aleatoria de la imagen y la redimensiona a un tamaño determinado |
| ColorJitter | Cambia aleatoriamente el brillo, el contraste, la saturación y el tono de una imagen | RandomRotation | Rota la imagen a un ángulo determinado |
| FiverCrop | Recorta la imagen en cuatro esquinas y el recorte central | RandomVerticalFlip | Voltea verticalmente la imagen al azar con una probabilidad dada |
| Grayscale | Convierte la imagen en escala de grises | Resize | Redimensiona la imagen al tamaño dado |
| Pad | Rellena la imagen en todos sus lados con el valor de <i>pad</i> dado | TenCrop | Recorta la imagen dada en cuatro esquinas y el recorte central |
| RandomAffine | Transformación aleatoria de la imagen manteniendo el centro invariante | GaussianBlur | Desenfoca la imagen con un filtro gaussiano |

Tabla 5-1: Funciones de transformación de imágenes del módulo de *Pytorch*

De igual forma, en la Tabla 5-2 se observan algunos filtros como recortes en la imagen, rotaciones, cambios de perspectiva, cambios en la nitidez y contraste de la imagen. El uso de los filtros en la fase de preprocesamiento permite enriquecer el *dataset* para garantizar que el modelo aprenda las características que debe clasificar y se disminuya el *overfitting* y *underfitting*.

| Image Transform | Use | Image Transform | Use |
|-------------------|----------------------------------------------------------------------------------------|--------------------|-------------------------------------------------------------------------|
| Randomcrop | Recorta la imagen aleatoriamente | RandomInvert | Invierte los colores de la imagen de forma aleatoria |
| RaandomGrayscale | Convierte la imagen en escala de grises aleatoriamente | RandomPosterize | Reduce el número de bits de cada canal de laa imagen de forma aleatoria |
| HorizontalFlip | Voltea horizontalmente la imagen al azar con una probabilidad dada | RandomSolarize | Invierte aleatoriamente el valor de los píxeles por encima de un umbral |
| RandomPerspective | Realiza una transformación de perspectiva aleatoria de la imagen | RandomAutocontrast | Autocontraste de los píxeles de la imagen dada aleatoriamente |
| AdjustSharpness | Ajusta la nitidez de la imagen de forma aleatoria | RandomEqualize | Equaliza el histograma de la imagen dada aleatoriamente |
| RandomApply | Aplicar aleatoriamente una lista de transformaciones con una probabilidad determinada. | | |

Tabla 5-2: Funciones de transformación de imágenes del módulo de *Pytorch*

Se escogieron los filtros considerados mejores, aquellos que no cambian la morfología del color, para el caso del *dataset* de tubérculos de papa creado en este documento. Los filtros que se utilizaron en el desarrollo del algoritmo de clasificación, teniendo en cuenta aquellas transformaciones que podrían afectar el rendimiento del algoritmo son:

- Resize
- ColorJitter
- RandomRotation
- GaussianBlur

- RandomPerspective
- RandomAdjustSharpness

Como se está utilizando la librería *Pytorch*, las imágenes deben ingresar en formato *Tensor*, que convierte los valores de los píxeles de una imagen *PIL* estándar, con un rango de [0, 255], a un tensor decimal de *Pytorch*, con valores en un rango con valores [0,0 , 1,0] de la forma (C, H, W) , siendo C el número de canales de la imagen, (1 si es en escala de grises, 3 si es *RGB*), H y W el tamaño de la imagen.

Una vez la imagen está en formato *PyTorch FloatTensor*, se decidió utilizar la función *torchvision.transforms.normalize()* para normalizar las imágenes. La normalización de una imagen consiste en modificar los valores del tensor, que se encuentran entre [0,0 , 1,0], para que el promedio y la desviación estándar sean 0 y 1 respectivamente. Para hacer esto se utiliza la Ecuación 5-1 [47]

$$output[Channel] = \frac{Input[Channel] - Mean[Channel]}{std[Channel]} \quad (5-1)$$

La normalización se utiliza debido a que ayuda a los datos a estar definidos dentro de un rango y reducir la asimetría entre ellos, lo que permite un aprendizaje más rápido. Se diseñó una función en *Python* que calcula el promedio y la desviación estándar, para el conjunto de datos de *entrenamiento* y *validación*. De esta forma se garantiza que la normalización sea correcta en el conjunto total de imágenes del *dataset*. Los valores obtenidos para el conjunto de *entrenamiento* se presentan en la Ecuación 5-2 y el conjunto de *validación* en la Ecuación 5-3.

$$mean = [0,7467, 0,7389, 0,7432] \quad std = [0,1288, 0,1633, 0,2045] \quad (5-2)$$

$$mean = [0,7507, 0,7430, 0,7486] \quad std = [0,1265, 0,1609, 0,2012] \quad (5-3)$$

La Figura 5-3 enseña una matriz de imágenes como parte del conjunto de *entrenamiento*, que muestra las transformaciones hechas al 80 % de las imágenes del *dataset*, y la clase a la que pertenece según la Tabla 4-3, verificando así que las funciones creadas están enlazando correctamente la imagen con su respectiva clase y aplicando de forma correcta las transformaciones.

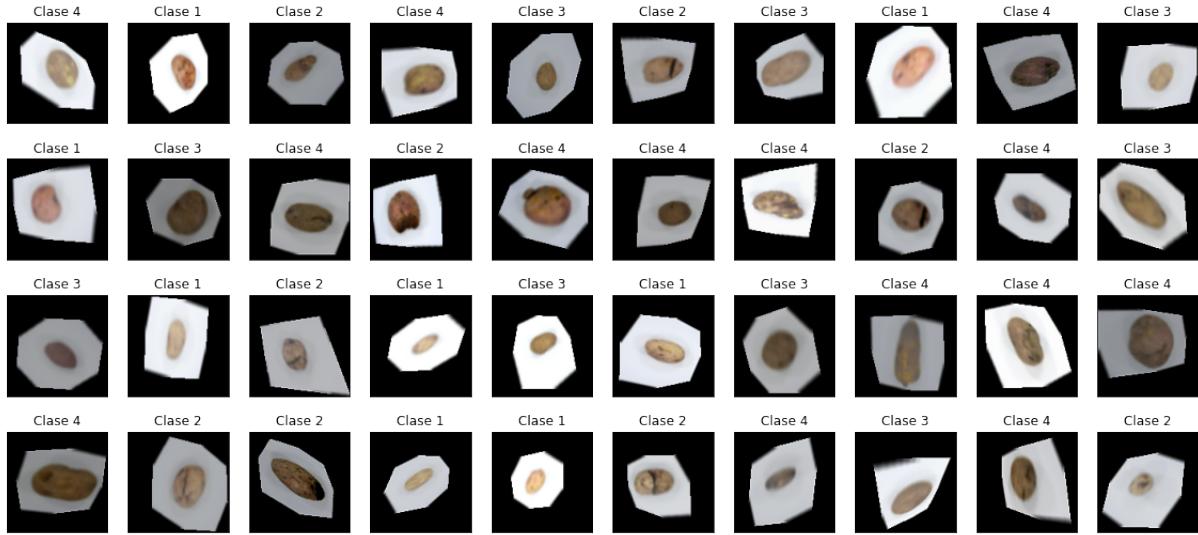


Figura 5-3: Matriz de Imágenes Con Transformaciones Aplicadas

Fuente: Elaboración Propia

5.1.2. Funciones PotatoDataset y DataLoader

Una vez se realiza el proceso de *Data Augmentation*, se deben cargar las imágenes en *Pytorch* para que puedan ser procesadas por el modelo de red neuronal. En esta sección se explica la creación de la función *PotatoDataset()*, y la implementación de la función *Dataloader*, ademas de la explicación de cada función, para cargar el conjunto de datos en *Pytorch*.

FUNCIÓN POTATODATASET

Se creó una clase de *Python (Python Class Object Constructor)*, con el nombre de *AttributesDataset()*, que se encarga de leer el *metadata* de cada imagen y convertirlo en *labelID*, en la Tabla 4-2 se muestra como el algoritmo lee las imágenes dentro del modelo, sin embargo, las características deben ser ingresadas como *labelID*, es decir, el *label* Pastusa corresponde al ID 0 en la categoría de Tipo. Este diccionario se presenta a continuación:

- Tipo: '*Pastusa*' : 0, '*R12*' : 1
- Daño: '*Buena*' : 0, '*Defectuosa*' : 1
- Tamaño: '*MuyGrande*' : 0, '*Grande*' : 1, '*Mediana*' : 2

En la red neuronal, únicamente se utilizaron *labels* y *labelID* de tipo y daño para definir las clases de la Tabla 4-3. La función *Dataset* de *Python* realiza el proceso de cargar y relacionar cada imagen con su respectivo *label*, para *Dataset's* que se encuentran organizados dentro de carpetas jerárquicamente. El *Dataset* creado en éste documento no se encuentra organizado de esta manera, sino que la organización jerárquica que define las clases se encuentra dentro del *metadata*, en el nombre de cada imagen, Tabla 4-2, debido a esto es que se desarrolló una función que cargara y relacionara de la misma manera que lo hace la función integrada de *Pytorch*. Esta función fue llamada *PotatoDataset()*, que utiliza los *labelID* generados por la función *AttributesDataset()*, y carga cada imagen relacionada con sus *labelID*. Como se desarrolló la función, de igual forma debe ser capaz de aplicar las transformaciones a cada imagen. El objeto regresa las 592 imágenes cargadas en *Python*, cada una relacionada con su respectivo *labelID* y *label*, y con una transformación o múltiples aplicadas. Esta función desarrollada, cuenta con los mismos atributos iterables que poseen los *dataset's* creados a partir de la función integrada en *Pytorch*.

FUNCIÓN DATALOADER

Generalmente, una vez se finaliza el proceso de cargar las imágenes utilizando la función *DataSet* de *Pytorch*, se procede a utilizar la función *DataLoader* para generar múltiples lotes de imágenes, a partir de las transformaciones utilizadas. La función *DataLoader()* de *Pytorch*, es implementada para realizar la importación de datos a gran escala.

El *dataset* que se generó a partir de la función creada *PotatoDataset()* es del tipo *iterable-style Datasets*, lo que permite utilizar la función integrada *DataLoader()* de *Pytorch*. La función permite la agrupación automática de muestras de datos individuales obtenidas en los lotes mediante argumentos. Los *DataLoader's* cuentan con una gran cantidad de argumentos que permiten que la carga de datos sea más eficiente [47]. Para la ejecución de la función se deben tener en cuenta los parámetros de la Tabla 5-3.

| FUNCIÓN | DESCRIPCIÓN |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Batch size | Cuántas muestras por Batch hay que cargar (Default: 1) |
| Shuffle | Se establece en TRUE para que los datos se reorganicen en cada época (Default: FALSE) |
| Num workers | Cuántos subprocessos se utilizarán para la carga de datos. 0 significa que los datos se cargarán en el proceso principal. (Default: 0) |

Tabla 5-3: Argumentos para el Dataloader

Los argumentos explicados en la tabla **5-3** fueron establecidos de la siguiente forma:

- *Batch Size* = 40
- *Shuffle* = *True*
- *Num workers* = 0

Esto con el objetivo de que el *Dataloader* cargue 40 muestras por época, y tome otras 40 diferentes para la siguiente época, y así sucesivamente hasta terminar todas las épocas establecidas.

5.1.3. Hiperparámetros Del Modelo

FUNCIÓN DE PÉRDIDAS

Pytorch posee una cantidad interesante de funciones de pérdidas para su uso en la clasificación de clases en redes neuronales. Una función de pérdidas, evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las muestras utilizadas durante el entrenamiento de la red. La eficiencia de una red neuronal se mide en objeto a la función de pérdidas, cuanto menor sea el valor, es un indicativo que es más eficiente [48].

Para el desarrollo de la red neuronal, se utilizó la pérdida de entropía cruzada entre la entrada y el objetivo. Es útil cuando se entrena un problema de clasificación con más de una clase. El objetivo que espera este criterio debe contener:

- Índices de clase en el rango $[0, C)$ donde C es el número de clases.
- Probabilidades de clase se requieren más allá de una sola clase cuando se encuentran etiquetas combinadas.

En cada lote usado en el entrenamiento de la red neuronal se encuentran etiquetas combinadas en el *metadata* de las imágenes, como se describió en la Tabla **4-3**, por lo tanto, una función de pérdida no reducida [47] se describe en la Ecuación 5-4.

$$l_n = - \sum_{c=1}^C w_c \log \left(\frac{\exp(x_n, c)}{\sum_{i=1}^C \exp(x_n, i)} y_n, c \right) \quad (5-4)$$

Donde x es la entrada, y el objetivo de predicción, w es el peso, C el número de clases y N es la dimensión del *Batch*. Usando la función *nn.CrossEntropyLoss()*, se calculan las

pérdidas entre entrada y salida del modelo entrenado, para posteriormente utilizar estas pérdidas como métricas en porcentajes de *Accuracy* y *Losses*.

El rendimiento de esta función de pérdidas es mejor cuando el objetivo contiene índices de clase y no las etiquetas, ya que esto permite optimizar el cálculo, por esta razón, se agruparon las etiquetas de las imágenes en 4 *Clases* definidas en la Tabla 4-3.

OPTIMIZADOR

Pytorch posee el paquete *torch.optim*, que implementa varios algoritmos de optimización derivados del *gradiente descendiente*, definido por *Sebastian Ruder (2016)* como "una forma de minimizar una función objetiva parametrizada por un modelo" [49]. A menudo se ha propuesto, por ejemplo *Rumelhart (1986)* [50], minimizar el riesgo empírico de una función objetivo $E_n(fw)$ utilizando el gradiente descendiente. Consiste en cada iteración actualizar los pesos w , como se muestra en la Ecuación 5-5.

$$w_{i+1} = w_i - \gamma \frac{1}{n} \sum_{i=1}^n \nabla Q(z_i, w_t) \quad (5-5)$$

Donde γ es la taza de aprendizaje elegida adecuadamente (*learning rate*) y $\nabla Q(z, w)$ es la función de pérdidas evaluada en un punto y un peso w . Cuando la estimación inicial w_0 está lo suficientemente cerca del óptimo y cuando la tasa de aprendizaje es suficientemente pequeña, este algoritmo alcanza una convergencia lineal.

En el algoritmo desarrollado en este documento, se utilizó el optimizador de descenso estocástico del gradiente *SGD*, que es una simplificación drástica del algoritmo del gradiente descendiente. En lugar de calcular exactamente el gradiente de $E_n(fw)$, cada iteración estima este gradiente sobre la base de un único ejemplo z_t elegido al azar [51].

$$w_{i+1} = w_i - \gamma_t \nabla_w Q(z_t, w_t) \quad (5-6)$$

Se espera que la Ecuación 5-5 se comporte como la Ecuación 5-6 a pesar del ruido introducido por este procedimiento simplificado. El algoritmo estocástico no necesita recordar qué ejemplos de datos fueron usados durante las iteraciones anteriores, por lo tanto, puede procesar ejemplos sobre la marcha en un sistema implementado. En esta situación, el descenso estocástico del gradiente optimiza directamente el riesgo esperado.

5.1.4. Arquitectura del modelo

Se utilizaron modelos pre entrenados para realizar el entrenamiento del *dataset*, utilizando la teoría de *Transfer Learning* que permite entrenar los modelos, con pesos establecidos entrenadas para otro tipo de datos, para obtener resultados a partir de ellos sin empezar desde cero la creación de una red neuronal y sus pesos. En este capítulo se presenta la estructura y los resultados obtenidos de los modelos pre entrenados *AlexNet*, *VGG19*, *VGG11* y *ResNet18*, sin el uso de un método de optimización de hiperparámetros.

MODELO PREENTRENADO ALEXNET

El modelo *AlexNet* fue implementado en el año 2012 en el desafío de reconocimiento visual a gran escala *ImageNet*. Este modelo tuvo un resultado tan satisfactorio, que los modelos de aprendizaje profundo, se convirtieron en la referencia para la investigación y desarrollo en los principales sectores de la industria [52]

El modelo se compone de 5 capas convolucionales y 3 capas densas, donde las capas convolucionales están normalizadas por lotes de datos. El *kernel* (Máscara aplicada en la convolución) tiene dimensiones diferentes que se distribuyen en 11×11 en la primera capa, 5×5 en la segunda y 3×3 en las demás. La primera, cuarta y quinta capas convolucionales están conectadas por una capa *Max pooling* con dimensiones de 3×3 , que posee un *stride* de 2. Las capas convolucionales y de *Max pooling* son seguidas por tres capas densas, en donde las primeras dos constan de 4096 neuronas cada una y la última capa es la de salida, la cual se compone de 1000 neuronas que poseen una función de activación *softmax*, esta capa fue modificada para que sean 4 neuronas de salida referentes a las 4 clases definidas para realizar la clasificación.

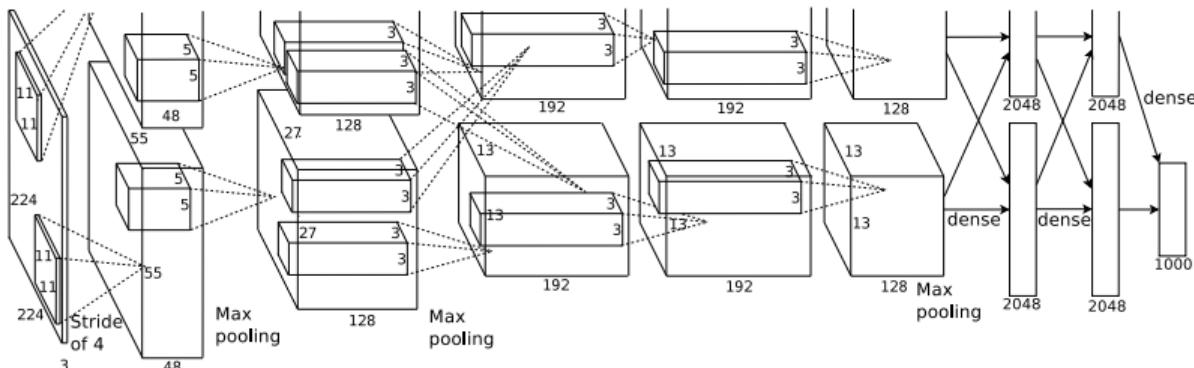


Figura 5-4: Arquitectura Del Modelo *AlexNet*

Fuente: Krizhevsky et al., (2012)

Como se aprecia en la Figura 5-4, el modelo de *AlexNet* toma una imagen de entrada con una resolución de 224×224 píxeles y 3 canales de color, la información recibida pasa por cinco capas convolucionales de neuronas con función de activación *ReLU* y 3 capas de *Max pooling*. La quinta capa de convolución posee 256 máscaras diferentes con dimensiones de 13×13 . Se implementan las dos capas densas de 4096 neuronas, las cuales están conectadas por una capa de salida de 1000 neuronas, que poseen una función de activación *softmax*. Las neuronas presentes en la capa de salida representan una clase diferente [53]. Los resultados del entrenamiento obtenidos fueron inferiores al 20%, por lo tanto, fueron excluidos en el análisis sin optimización de los hiperparámetros.

MODELO PREENTRENADO VGG19

La arquitectura *VGG* fue planteada por Simonyan y Zisserman [54], y es definida por Diaz-Gaxiola (2019) como "grupos lineales de bloques que se encuentran formados por una cantidad determinada de capas convolucionales, una función de activación no lineal y una capa de *Max pooling*, seguidos de tres capas densas y finalmente una capa de activación *softmax*". Esta arquitectura cuenta con 5 bloques los cuales están distribuidos como se muestra en la Figura 5-5.

| ConvNet Configuration | | | | | |
|-------------------------------------|------------------------|-------------------------------|--------------------------------------------|--------------------------------------------|---------------------------------------------------------|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224×224 RGB image) | | | | | |
| conv3-64 | conv3-64 LRN | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 |
| maxpool | | | | | |
| conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 conv1-256 | conv3-256 conv3-256 conv3-256 | conv3-256 conv3-256 conv3-256 conv3-256 |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 conv1-512 | conv3-512 conv3-512 conv3-512 | conv3-512 conv3-512 conv3-512 conv3-512 |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 conv1-512 | conv3-512 conv3-512 conv3-512 | conv3-512 conv3-512 conv3-512 conv3-512 |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Figura 5-5: Arquitectura Del Modelo VGG

Fuente: Simonya, K. (2015)

Como se puede observar en la Figura 5-5, la columna *E* muestra que la arquitectura del modelo *VGG19* cuenta con 2 capas convolucionales de 64 y 128 filtros respectivamente, en sus dos primeros bloques, el bloque intermedio se compone de 3 capas convolucionales de 256 filtros, y los últimos dos bloques están compuestos por 3 capas convolucionales de 512 filtros cada uno. El numero 19 representa la cantidad de capas que se pueden entrenar en la arquitectura, donde hay 16 capas convolucionales y 3 capas densas [55].

Las capas convolucionales de esta arquitectura presentadas en la Figura 5-5, cuentan con un campo receptivo de 3×3 , *stride* de 1×1 y *padding* de 1 pixel. Para realizar las operaciones de *Max pooling* se tienen en cuenta un *kernel* de dimensiones 2×2 , *stride* de 2×2 , y por ultimo se implementa una función *ReLU*. en cada capa oculta de la red para su activación. La Figura 5-6 muestra la precisión obtenida del entrenamiento hecho sobre esta arquitectura.

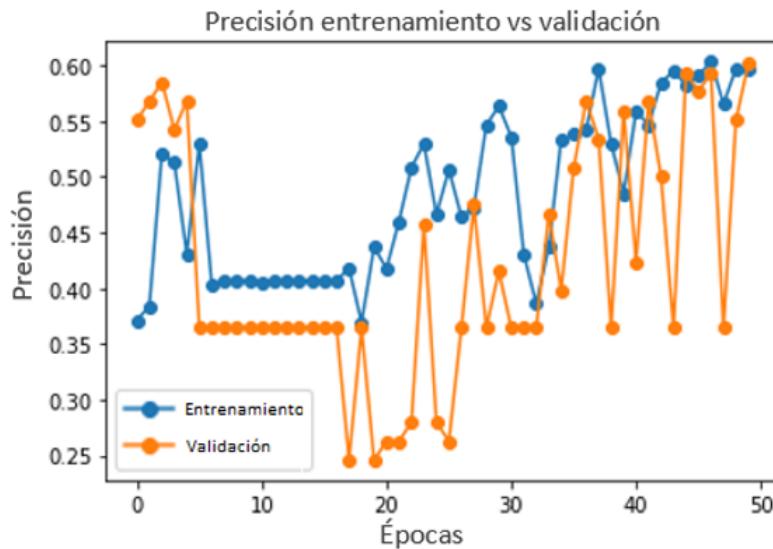
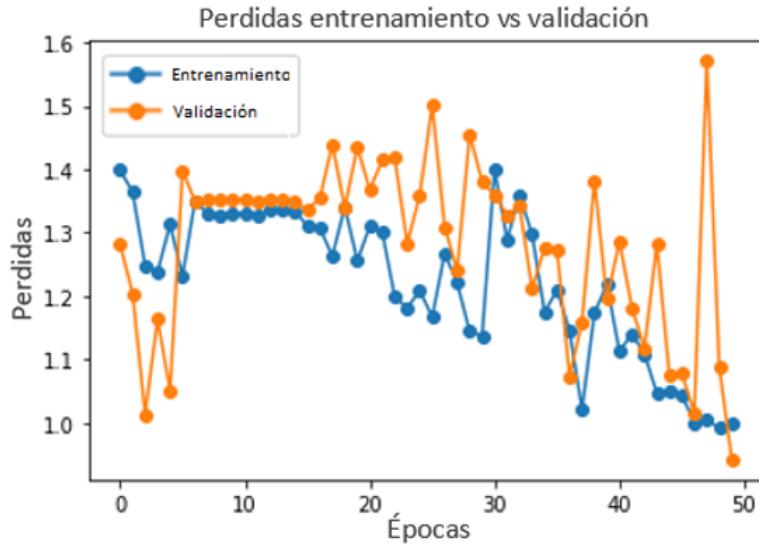


Figura 5-6: Precisión Obtenida Con Modelo *VGG19*

Fuente: Elaboración Propia

La precisión del modelo al completar las 50 épocas de entrenamiento es inferior al 60% como se observa en la Figura 5-6. La figura 5-7 muestra el comportamiento de la función de pérdidas, que tiende a 0 a medida que las épocas avanzan. La eficacia del modelo se mide a partir de que tan cercano a 0 se obtiene la función de pérdidas.

**Figura 5-7:** Pérdidas Obtenidas Con Modelo *VGG19*

Fuente: Elaboración Propia

El hecho de que en la última época, la precisión en el entrenamiento y en la validación dan valores similares, indica que el modelo no sufre de *overfitting* que significaría un mal aprendizaje en la salida del modelo. Una vez se obtiene el modelo entrenado con los hiperparámetros no optimizados, se calcula la matriz de confusión con el conjunto de datos de *validación*, los resultados se presentan en la Tabla 5-4.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 27 | 2 | 0 | 0 |
| | CLASE 2 | 21 | 1 | 0 | 2 |
| | CLASE 3 | 6 | 5 | 0 | 11 |
| | CLASE 4 | 20 | 15 | 0 | 8 |

Tabla 5-4: Matriz de confusión en conjunto de *validación*

En la Tabla 5-5 se presenta la precisión del modelo *VGG 19* obtenida a partir de la matriz de confusión del conjunto de datos de *validación* para cada una de las clases definidas.

| | |
|------------------------|-------|
| CLASE 1 | 75,90 |
| CLASE 2 | 50,00 |
| CLASE 3 | 90,90 |
| CLASE 4 | 51,20 |
| Precisión Total: 36,84 | |

Tabla 5-5: Precisión por clase en conjunto de *validación*

Para comparar los resultados posteriormente, también se calcula la matriz de confusión con el conjunto de datos de *entrenamiento*, los resultados se presentan en la Tabla 5-6.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 99 | 11 | 0 | 1 |
| | CLASE 2 | 64 | 22 | 0 | 0 |
| | CLASE 3 | 31 | 49 | 0 | 4 |
| | CLASE 4 | 84 | 103 | 0 | 6 |

Tabla 5-6: Matriz de confusión en conjunto de *entrenamiento*

En la Tabla 5-7 se presenta la precisión del modelo obtenida a partir de la matriz de confusión del conjunto de datos de *entrenamiento* para cada una de las clases definidas.

| | |
|------------------------|-------|
| CLASE 1 | 89.20 |
| CLASE 2 | 25.60 |
| CLASE 3 | 0.00 |
| CLASE 4 | 3.10 |
| Precisión Total: 26.47 | |

Tabla 5-7: Precisión por clase en conjunto de *entrenamiento*

MODELO PREENTRENADO VGG11

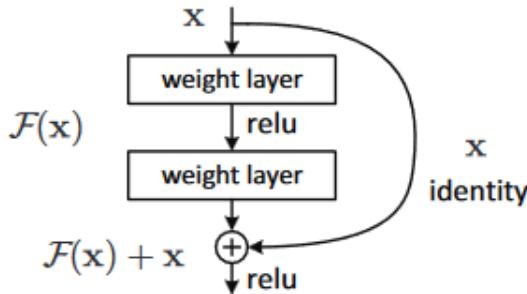
El modelo de aprendizaje profundo *VGG11* es la configuración más sencilla de los modelos de arquitectura *VGG*. Tiene 11 capas de peso en total, de ahí el nombre, 8 de ellas son capas convolucionales y 3 son capas completamente conectadas, como se aprecia en la Figura 5-5 en la columna *A*. Las 11 capas del modelo *VGG11* son:

- Convolución usando 64 filtros + *Max pooling*
- Convolución usando 128 filtros + *Max pooling*
- Convolución usando 256 filtros
- Convolución usando 256 filtros + *Max pooling*
- Convolución usando filtros 512
- Convolución usando 512 filtros + *Max pooling*
- Convolución usando filtros 512
- Convolución usando 512 filtros + *Max pooling*
- Totalmente conectado con 4096 neuronas
- Totalmente conectado con 4096 neuronas
- Capa de salida con activación Softmax con 1000 neuronas.

El modelo cuenta con 11 capas ponderadas, sus pesos representan la fuerza de las conexiones entre unidades de capas de red adyacentes, son implementadas con el fin de conectar cada neurona de una capa con todas las neuronas de la siguiente capa, se debe tener en cuenta que la capa *Max pooling* no se considera como una capa ponderada, debido a que es un mapa de características que contiene las características más destacadas. Los resultados del entrenamiento obtenidos fueron inferiores al 15 %, por lo tanto, fueron excluidos en el análisis sin optimización de los hiperparámetros.

MODELO PREENTRENADO RESNET18

La arquitectura *ResNet* (Residual Net), fue propuesta por un equipo de Microsoft Research liderado por *Kaiming He* [56], en donde se propone que si a una red neuronal de 20 capas se le intercalan 36 capas que calculan la simple función de identidad, la red resultante debería tener exactamente la misma eficacia y no ser inferior que la primera. Se debe recordar que la función identidad es la encargada de devolver exactamente el mismo valor que su argumento [57]. Para permitir que una red residual pueda variar su cantidad efectiva de capas, se introduce el concepto de bloque residual en la Figura 5-8.

**Figura 5-8:** Bloque Residual

Fuente: He, K. (2015)

Como se aprecia en la Figura 5-8, el bloque residual se compone de una ruta residual, (izquierda), y una conexión atajo (derecha), que las une. La ruta residual $F(x)$, esta compuesta de dos capas de pesos sinápticos (pueden ser densas o convolucionales), que se intercalan por una función rectificadora. El resultado se suma con la información que atraviesa la conexión atajo X “identidad”. Y por ultimo se aplica nuevamente la función rectificadora. La información puede atravesar con ello dos caminos diferentes que son: el de la función de identidad X o el de la ruta residual $F(x)$ [58].

En la arquitectura *ResNet18*, las 18 capas de esta arquitectura representan las 18 capas con pesos, en donde se incluye la capa de convolución y la capa totalmente conectada, excluyendo la capa de agrupación.

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|------------|-------------|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| conv1 | 112×112 | | | 7×7, 64, stride 2 | | |
| conv2_x | 56×56 | $\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$ | $\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$ | $\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$ | $\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$ | $\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | 1.8×10^9 | 3.6×10^9 | 3.8×10^9 | 7.6×10^9 | 11.3×10^9 |

Figura 5-9: Arquitectura *ResNet18*

Fuente: He, K. (2015)

Como se puede observar en la Figura 5-9, la primera capa de convolución implementa una máscara de $7 \times 77 \times 7$, con un tamaño de paso de 2 y un relleno de 3. Luego, se aplican las capas con la función de activación *ReLU* y *Max pooling*. Estos constituyen la primera parte del modulo de convolución *conv1*. Luego hay cuatro etapas, cada etapa tiene múltiples módulos, cada módulo se llama un bloque de construcción, por ende se evidencia que hay 8 bloques de construcción, como se presenta en la Figura 5-9 [58].

La Figura 5-10, muestra la precisión obtenida en el entrenamiento del modelo *ResNet18* durante 50 épocas, sin haber aplicado la optimización bayesiana.

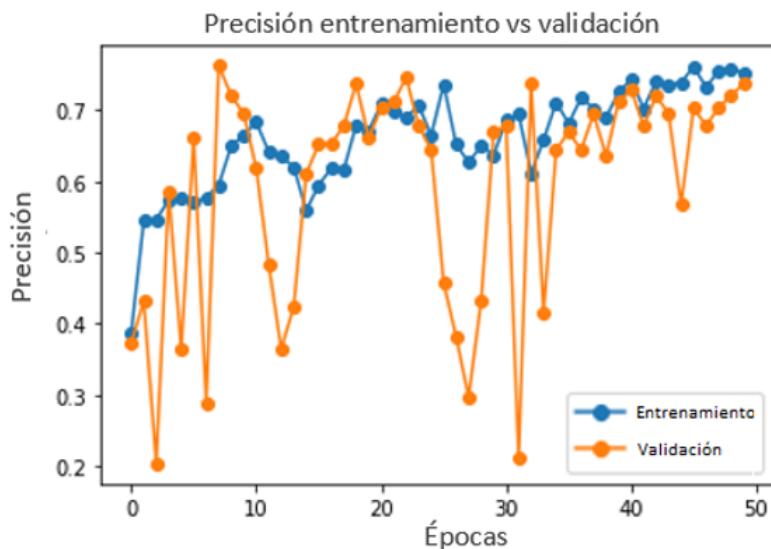


Figura 5-10: Precisión Obtenida Con Modelo *Resnet18*

Fuente: Elaboración Propia

La información presentada en la Figura 5-10 permite observar que el modelo obtuvo una precisión inferior al 80 % y, que el porcentaje de *overfitting* se reduce en las épocas finales. La figura 5-11 muestra que la función de pérdidas tuvo un comportamiento con mayor eficacia en el entrenamiento con el modelo *ResNet18*.

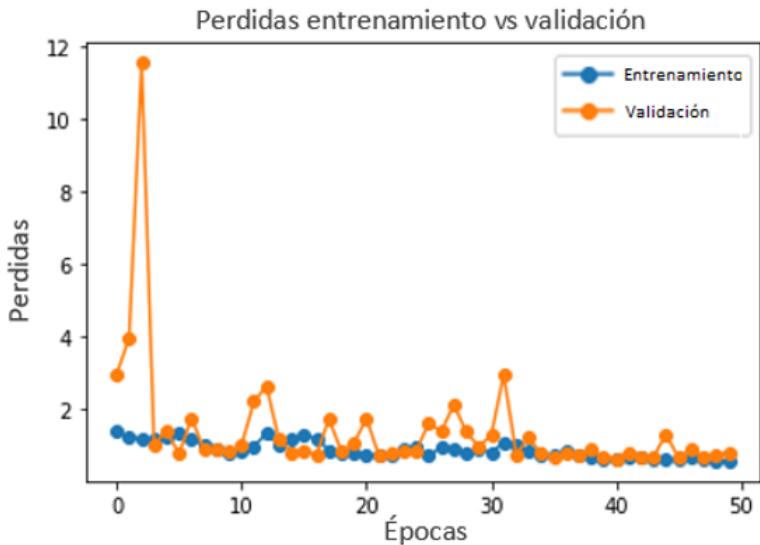


Figura 5-11: Pérdidas Obtenidas Con Modelo *Resnet18*
Fuente: Elaboración Propia

La matriz de confusión obtenida del modelo *ResNet18* con el conjunto de datos de *validación*, es presentada en la Tabla 5-8.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 0 | 0 | 29 | 0 |
| | CLASE 2 | 0 | 0 | 24 | 0 |
| | CLASE 3 | 0 | 0 | 22 | 0 |
| | CLASE 4 | 0 | 0 | 43 | 0 |

Tabla 5-8: Matriz de confusión en conjunto de *validación*

En la Tabla 5-9, se presenta la precisión del modelo aplicada al conjunto de datos de *validación* por clases, obtenidas a partir de la matriz de confusión.

| | |
|------------------------|--------|
| CLASE 1 | 0,00 |
| CLASE 2 | 0,00 |
| CLASE 3 | 100,00 |
| CLASE 4 | 0,00 |
| Precisión Total: 18,42 | |

Tabla 5-9: Precisión por clase en conjunto de *validación*

Para comparar los resultados posteriormente, se calcula la matriz de confusión con el conjunto de datos de *entrenamiento*, los resultados se presentan en la Tabla **5-10**.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 0 | 0 | 111 | 0 |
| | CLASE 2 | 0 | 0 | 86 | 0 |
| | CLASE 3 | 0 | 0 | 84 | 0 |
| | CLASE 4 | 0 | 0 | 193 | 0 |

Tabla 5-10: Matriz de confusión en conjunto de *entrenamiento*

En la Tabla **5-11** se presenta la precisión del modelo aplicada al conjunto de datos de *entrenamiento* por clases, obtenidas a partir de la matriz de confusión.

| | |
|------------------------|--------|
| CLASE 1 | 0,00 |
| CLASE 2 | 0,00 |
| CLASE 3 | 100,00 |
| CLASE 4 | 0,00 |
| Precisión Total: 23.52 | |

Tabla 5-11: Precisión por clase en conjunto de *entrenamiento*

5.1.5. Optimización De Hiperparámetros

La Tabla **5-12** muestra los hiperparámetros escogidos para optimizar utilizando la técnica de optimización Bayesiana. Estos hiperparámetros influyen en el comportamiento del optimizador, mejorando el proceso de aprendizaje de la *RNC*.

| Parámetro | Limite inferior | Limite Superior |
|---------------------|-----------------|-----------------|
| Taza de Aprendizaje | $1 * 10^{-6}$ | 0.4 |
| Momento | 0 | 1 |
| Tamaño del Paso | 20 | 40 |

Tabla 5-12: Hiperparámetros Optimizados

Botorch realiza 20 pruebas con parámetros distintos en los rangos definidos, y encuentra la combinación de parámetros que den el mejor resultado de precisión final del algoritmo. Estos resultados fueron utilizados para entrenar nuevamente los modelos descritos anteriormente, para comparar la eficacia de los algoritmos y modelos implementados.

MODELO ALEXNET OPTIMIZADO

En la Figura 5-12, se observa la precisión obtenida en el entrenamiento del modelo *AlexNet* con hiperparámetros optimizados. A pesar de que la precisión es superior al 60 %, que indica una mejora en los resultados con respecto a la versión no optimizada que no fueron tenidos en cuenta, se presenta un *overfitting*, por lo que la red no está aprendiendo de manera correcta las características de el *dataset*.

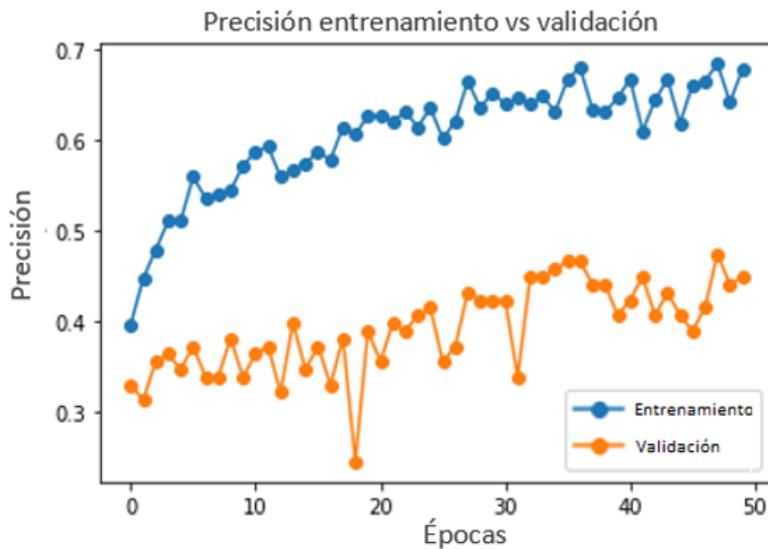


Figura 5-12: Precisión modelo AlexNet optimizado

Fuente: Elaboración Propia

La Tabla **5-13** almacena los datos de la matriz de confusión del modelo *AlexNet* optimizado en el conjunto de datos de *validación*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 0 | 0 | 29 | 0 |
| | CLASE 2 | 0 | 1 | 22 | 1 |
| | CLASE 3 | 0 | 0 | 22 | 0 |
| | CLASE 4 | 0 | 1 | 42 | 0 |

Tabla 5-13: Matriz de confusión en conjunto de *validación*

La Tabla **5-14** presenta la precisión del modelo por cada clase en el conjunto de datos de *validación*.

| | |
|------------------------|--------|
| CLASE 1 | 0,00 |
| CLASE 2 | 4,20 |
| CLASE 3 | 100,00 |
| CLASE 4 | 0,00 |
| Precisión Total: 21,05 | |

Tabla 5-14: Clasificación por clase en conjunto de *validación*

La Tabla **5-15** almacena los datos de la matriz de confusión del modelo *AlexNet* optimizado en el conjunto de datos de *entrenamiento*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 0 | 0 | 111 | 0 |
| | CLASE 2 | 1 | 3 | 81 | 1 |
| | CLASE 3 | 0 | 0 | 84 | 0 |
| | CLASE 4 | 0 | 0 | 192 | 1 |

Tabla 5-15: Matriz de confusión en conjunto de *entrenamiento*

La Tabla 5-16 presenta la precisión del modelo por cada clase en el conjunto de datos de *entrenamiento*.

| | |
|------------------------|--------|
| CLASE 1 | 0,00 |
| CLASE 2 | 3.50 |
| CLASE 3 | 100,00 |
| CLASE 4 | 0,50 |
| Precisión Total: 14.70 | |

Tabla 5-16: Clasificación por clase en conjunto de *entrenamiento*

A pesar de que la precisión en el entrenamiento del modelo superó el 60 % con los hiperparámetros optimizados, en la fase de validación del algoritmo, los resultados de precisión dieron una precisión del 14,7 %, que indica el mal aprendizaje obtenido por la red representado en el *overfitting* que presenta-

MODELO VGG19 OPTIMIZADO

La Figura 5-13, muestra que la precisión obtenida en el entrenamiento del modelo *VGG19* es de alrededor del 75 % en validación, el cual es un valor que se acerca mucho al valor del modelo en entrenamiento de aproximadamente el 80 %.

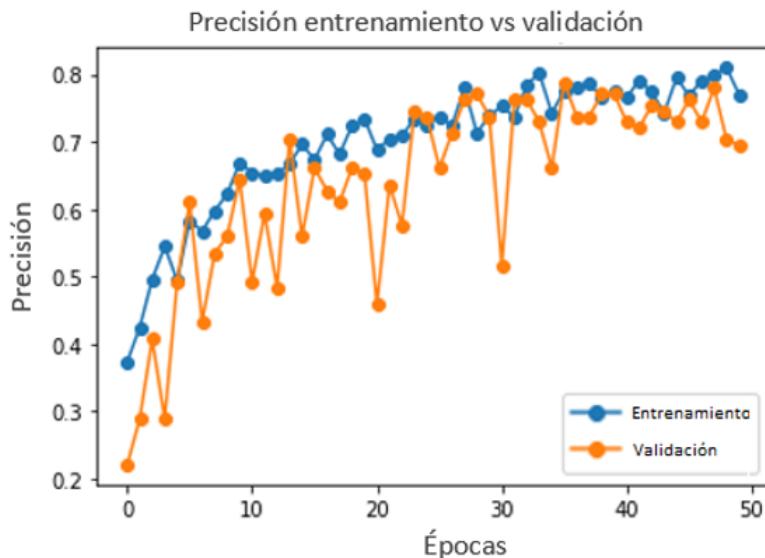


Figura 5-13: Precisión modelo VGG-19 optimizado

Fuente: Elaboración Propia

La Figura 5-14, de igual forma, permite verificar el comportamiento de la función de pérdidas en el entrenamiento de este modelo, que indica una eficacia superior a los demás modelos entrenados debido al cercano valor a 0 que presenta.

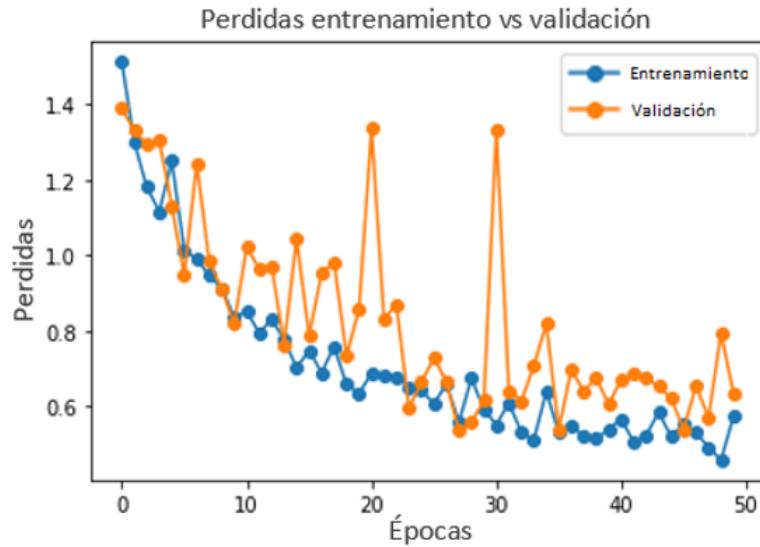


Figura 5-14: Perdidas modelo VGG-19 optimizado
Fuente: Elaboración Propia

En la Tabla 5-17 se presenta la matriz de confusión obtenida del modelo *VGG19* optimizado en el conjunto de datos de *validación*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 22 | 0 | 7 | 0 |
| | CLASE 2 | 4 | 12 | 1 | 7 |
| | CLASE 3 | 1 | 0 | 20 | 1 |
| | CLASE 4 | 1 | 1 | 19 | 22 |

Tabla 5-17: Matriz de confusión en conjunto de *validación*

La Tabla **5-18** presenta la precisión del modelo por cada clase en el conjunto de datos de *validación*.

| | |
|------------------------|-------|
| CLASE 1 | 75,90 |
| CLASE 2 | 50,00 |
| CLASE 3 | 90,90 |
| CLASE 4 | 51,20 |
| Precisión Total: 65,78 | |

Tabla 5-18: Clasificación por clase en conjunto de *validación*

La Tabla **5-19** almacena los datos de la matriz de confusión del modelo *VGG19* optimizado en el conjunto de datos de *entrenamiento*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 80 | 0 | 31 | 0 |
| | CLASE 2 | 8 | 66 | 3 | 9 |
| | CLASE 3 | 1 | 0 | 83 | 0 |
| | CLASE 4 | 0 | 5 | 73 | 115 |

Tabla 5-19: Matriz de confusión en conjunto de *entrenamiento*

La Tabla **5-20** presenta la precisión del modelo por cada clase en el conjunto de datos de *entrenamiento*.

| | |
|------------------------|-------|
| CLASE 1 | 72.10 |
| CLASE 2 | 76.70 |
| CLASE 3 | 98.80 |
| CLASE 4 | 59.60 |
| Precisión Total: 73.52 | |

Tabla 5-20: Clasificación por clase en conjunto de *entrenamiento*

La Figura 5-15 muestra una matriz con imágenes aleatorias del conjunto de *entrenamiento* para verificar las predicciones que el modelo está realizando. En la parte superior de cada imagen de tubérculo de papa se observa las siglas *R.C* que corresponde a la clase real a la que pertenece, y la sigla *Pred* que indica la predicción que realizó el modelo.

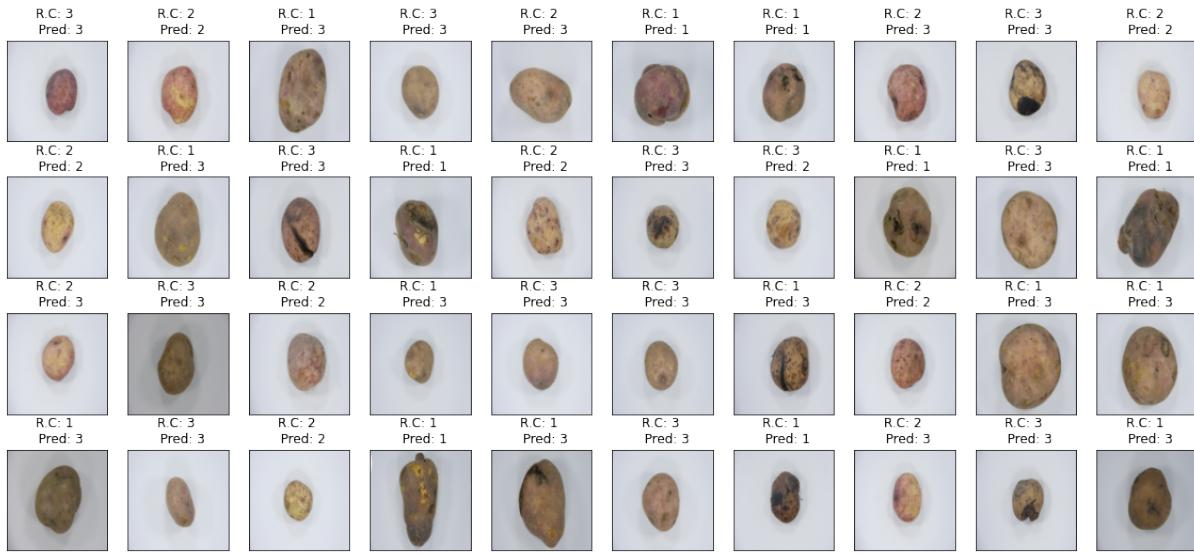


Figura 5-15: Predicciones En El Conjunto De *Entrenamiento*

Fuente: Elaboración Propia

De esta forma se puede verificar que la optimización de hiperparámetros mejora considerablemente la eficiencia de una red neuronal, permitiendo que se obtengan mejores resultados en la fase de validación del algoritmo.

MODELO VGG11 OPTIMIZADO

La Figura 5-16 muestra observar la precisión obtenida en el entrenamiento del modelo *VGG11* con parámetros optimizados. A pesar que la precisión es superior al 80 %, lo cual es una mejora a los resultados obtenidos en los modelos anteriores, el entrenamiento tiene un porcentaje de *overfitting* de aproximadamente 10 % que podría causar errores considerables en las predicciones.

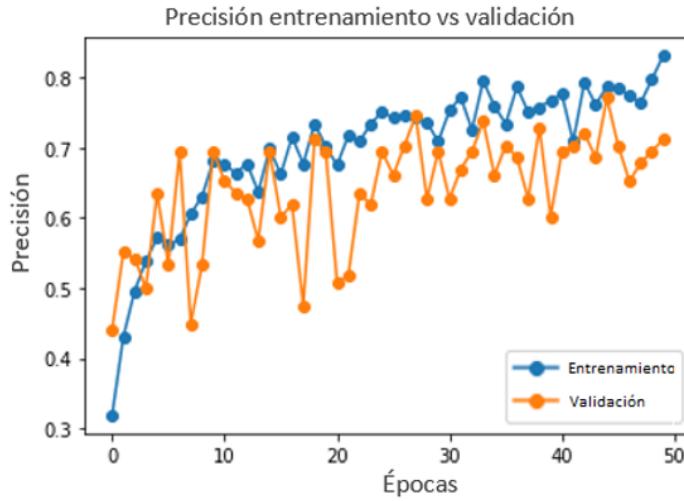


Figura 5-16: Precisión modelo VGG11 optimizado
Fuente: Elaboración Propia

La Figura 5-17 presenta las pérdidas del entrenamiento del modelo, a pesar que existe una clara desviación entre los datos predichos y los valores reales, las pérdidas indican valores inferiores a 1, que indica que el entrenamiento no es totalmente ineficiente.

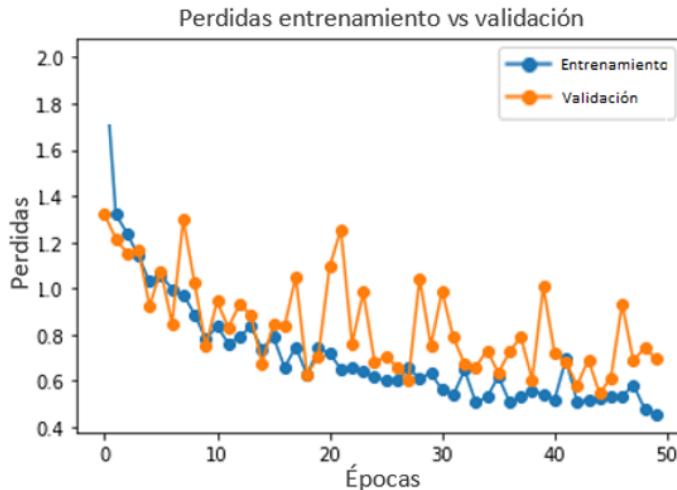


Figura 5-17: Perdidas modelo VGG11 optimizado

La Tabla 5-21, almacena los datos de la matriz de confusión del modelo *VGG11* optimizado en el conjunto de datos de *validación*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 18 | 0 | 11 | 0 |
| | CLASE 2 | 4 | 0 | 18 | 2 |
| | CLASE 3 | 2 | 0 | 20 | 0 |
| | CLASE 4 | 1 | 0 | 41 | 1 |

Tabla 5-21: Matriz de confusión en conjunto de *validación*

La Tabla 5-22 presenta la precisión del modelo por cada clase en el conjunto de datos de *validación*.

| | |
|------------------------|-------|
| CLASE 1 | 62,10 |
| CLASE 2 | 0,00 |
| CLASE 3 | 90,90 |
| CLASE 4 | 2,30 |
| Precisión Total: 39,47 | |

Tabla 5-22: Clasificación por clase en conjunto de *validación*

La Tabla 5-23 almacena los datos de la matriz de confusión del modelo *VGG11* optimizado en el conjunto de datos de *entrenamiento*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 69 | 0 | 42 | 0 |
| | CLASE 2 | 29 | 2 | 54 | 1 |
| | CLASE 3 | 2 | 0 | 82 | 0 |
| | CLASE 4 | 2 | 0 | 186 | 5 |

Tabla 5-23: Matriz de confusión en conjunto de *entrenamiento*

La Tabla 5-24 presenta la precisión del modelo por cada clase en el conjunto de datos de *entrenamiento*.

| | |
|------------------------|-------|
| CLASE 1 | 62,20 |
| CLASE 2 | 2.30 |
| CLASE 3 | 97.60 |
| CLASE 4 | 2,60 |
| Precisión Total: 40,35 | |

Tabla 5-24: Clasificación por clase en conjunto de *entrenamiento*

MODELO RESNET18 OPTIMIZADO

La precisión obtenida en el entrenamiento del modelo *ResNet18* se presenta en la Figura 5-18, en donde se puede apreciar que el modelo presenta un alto porcentaje de *overfitting*.

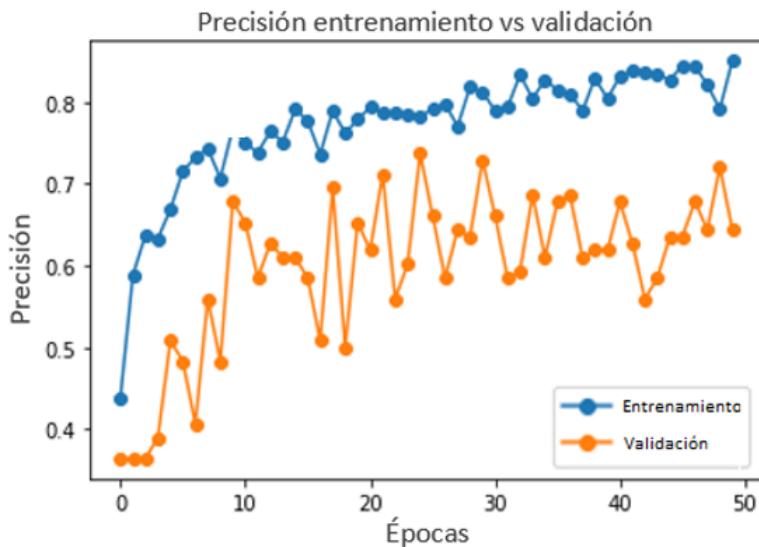


Figura 5-18: Precisión modelo ResNet18 optimizado

Fuente: Elaboración Propia

La Figura 5-19 muestra la tendencia del modelo a tener predicciones incorrectas e ineeficiencia del entrenamiento debido al valor alto de pérdidas en la validación del modelo.

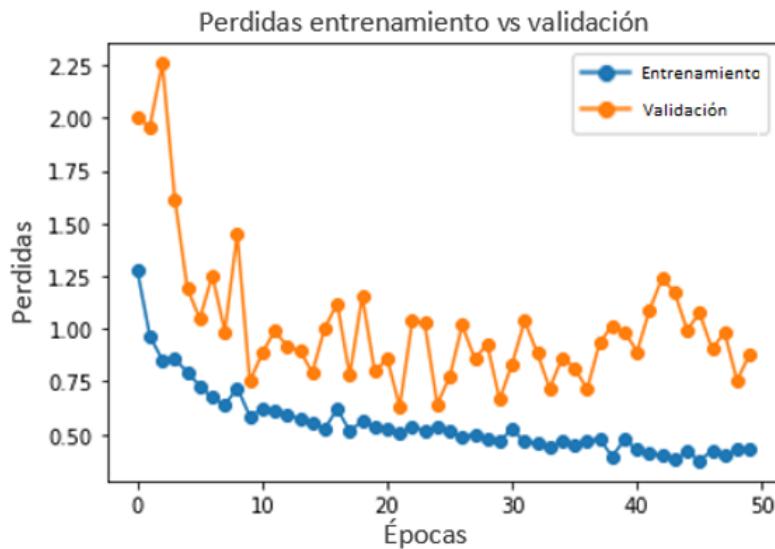


Figura 5-19: Perdidas modelo ResNet18 optimizado
Fuente: Elaboración Propia

En la Tabla 5-25, se puede evidenciar la matriz de confusión del modelo *ResNet18* optimizado en el conjunto de datos de *validación*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 29 | 0 | 0 | 0 |
| | CLASE 2 | 19 | 0 | 5 | 0 |
| | CLASE 3 | 12 | 0 | 10 | 0 |
| | CLASE 4 | 21 | 0 | 21 | 1 |

Tabla 5-25: Matriz de confusión en conjunto de *validación*

La Tabla 5-26 presenta la precisión del modelo por cada clase en el conjunto de datos de *validación*.

| | |
|------------------------|--------|
| CLASE 1 | 100,00 |
| CLASE 2 | 0,00 |
| CLASE 3 | 45,50 |
| CLASE 4 | 2,30 |
| Precisión Total: 36,84 | |

Tabla 5-26: Clasificación por clase en conjunto de *validación*

La Tabla 5-27 almacena los datos de la matriz de confusión del modelo *ResNet18* optimizado en el conjunto de datos de *entrenamiento*.

| | | Predicción | | | |
|------------|---------|------------|---------|---------|---------|
| | | CLASE 1 | CLASE 2 | CLASE 3 | CLASE 4 |
| Valor Real | CLASE 1 | 110 | 0 | 1 | 0 |
| | CLASE 2 | 68 | 0 | 18 | 0 |
| | CLASE 3 | 47 | 0 | 37 | 0 |
| | CLASE 4 | 101 | 0 | 92 | 0 |

Tabla 5-27: Matriz de confusión en conjunto de *entrenamiento*

La Tabla 5-28 presenta la precisión del modelo por cada clase en el conjunto de datos de *entrenamiento*.

| | |
|------------------------|-------|
| CLASE 1 | 99,10 |
| CLASE 2 | 0,00 |
| CLASE 3 | 44,00 |
| CLASE 4 | 0,00 |
| Precisión Total: 32,35 | |

Tabla 5-28: Clasificación por clase en conjunto de *entrenamiento*

5.1.6. Comparación De Precisión De Modelos Implementados

En la Tabla 5-29, se presenta la comparación de los cuatro modelos explicados anteriormente, aplicando optimización bayesiana con el objetivo de mejorar su precisión.

| MODELOS | AlexNet | VGG-19 | VGG-11 | ResNet-18 |
|---------------|---------|--------|--------|-----------|
| PRECISIÓN (%) | 21,05 | 65,78 | 39,47 | 36,84 |
| PERDIDAS (%) | 18,00 | 16,00 | 12,00 | 65,00 |

Tabla 5-29: Comparación modelos optimizados

En la Tabla 5-29, se puede observar que el modelo con mejor precisión es el modelo *VGG-19*, pero también se aprecia que es el modelo con mayores perdidas. Sin embargo, se puede apreciar que la precisión de los modelos documentada en la Tabla 5-29, es superior a la precisión documentada en la Tabla ???. Lo cual permite concluir que la implementación de la optimización bayesiana afecto de manera significativa a los modelos, para mejorar sus precisiones y reducir sus perdidas.

5.2. Clasificación Por Tamaño

Para realizar la clasificación por tamaño se utilizó el módulo de visión artificial para *Python OpenCV*. Primero se realizó un estudio a un grupo de 5 imágenes por cada uno de los tamaños. Se debe aplicar una umbralización para retirar el fondo de las imágenes, la Figura 5-20 muestra el uso de la herramienta de MATLAB® *Color Thresholder*.

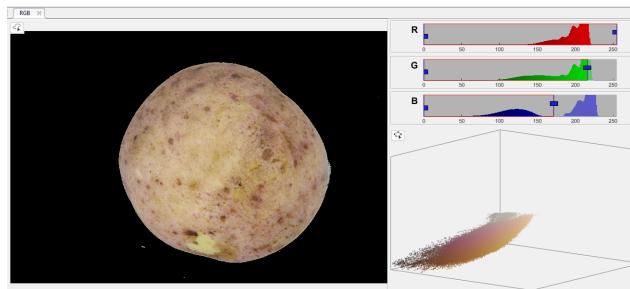


Figura 5-20: Matlab Color Thresholder

Fuente: Elaboración Propia

La herramienta permite, de forma interactiva, crear una función para quitar ciertos colores en el espacio *RGB* de la imagen de muestra, en este caso, se realizó el proceso para retirar el fondo de la imagen de la papa, A este proceso se le conoce como umbralización.

Se pretende encontrar el perímetro aproximado de la papa utilizando la función *FindContour* de *OpenCV*, debido a esto, es necesario realizar una serie de transformaciones que conviertan la imagen de muestra, en una imagen binaria (blanco y negro). Se utilizaron los filtros de dilatación y mediana para realizar este proceso como se muestra en la Figura 5-21.

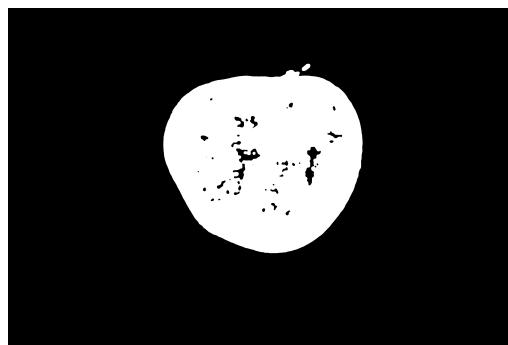


Figura 5-21: Imagen de Muestra Binarizada

Fuente: Elaboración Propia

Con la imagen aislada de la mayoría de características que puedan interferir en el cálculo del perímetro, utilizando la función de *FindContour*, se puede encontrar y dibujar un perímetro aproximado como se muestra en la Figura 5-22.



Figura 5-22: Perímetro Aproximado De Las Papas
Fuente: Elaboración Propia

En este punto, utilizando la función *ArcLength* de *OpenCV*, se puede calcular el perímetro del contorno encontrado en cantidad de píxeles. Debido a que las imágenes tomadas para crear el *dataset* fueron hechas desde una distancia fija desde el lente de la cámara, el tamaño relativo se puede asumir correcto en relación de tamaños. Se realizó un estudio en 5 imágenes de cada tamaño, en donde se calcularon los perímetros, se calculó un promedio y una desviación estándar, para definir los límites en el tamaño de píxeles por cada tamaño de papa definido. En la Tabla 5-30 se muestran los resultados de estudio realizado.

| Muy Grande | | Grande | | Mediana | |
|-----------------|----------------------|-----------------|----------------------|-----------------|----------------------|
| Imagen | Tamaño del Perímetro | Imagen | Tamaño del Perímetro | Imagen | Tamaño del Perímetro |
| 0_0_0_1963.jpg | 8580,08 | 0_0_1_1239.jpg | 6524,93 | 0_0_2_2011.jpg | 6074,97 |
| 0_1_0_1806.jpg | 9523,78 | 0_1_1_1828.jpg | 6802,54 | 0_1_2_1506.jpg | 5018,00 |
| 1_0_0_0777.jpg | 13275,00 | 1_0_1_0796.jpg | 10602,79 | 1_0_2_1995.jpg | 5090,17 |
| 1_1_0_0825.jpg | 27830,00 | 1_1_1_0770.jpg | 10285,33 | 1_1_2_0857.jpg | 5500,75 |
| 0_0_0_1964.jpg | 21607,00 | 0_0_1_1240.jpg | 6726,00 | 0_0_2_2013.jpg | 6046,62 |
| Promedio | 16163,19 | Promedio | 8188,33 | Promedio | 5546,12 |
| Variacion STD | 7425,23 | Variacion STD | 1846,76 | Variacion STD | 451,43 |
| Límite inferior | 9000,00 | Límite inferior | 6341,56 | Límite inferior | 5094,68 |
| Límite superior | 23588,42 | Límite superior | 9000,00 | Límite superior | 5997,56 |

Tabla 5-30: Perímetro en Tamaño de Píxeles

De esta forma, se realiza la clasificación por tamaño en nuevas imágenes de papas, si el valor del perímetro se encuentra dentro de los límites inferior y superior calculados para cada tamaño, la imagen será clasificada de acuerdo a los rangos.

6 Prototipo

6.1. Especificaciones principales

Se implementa una banda transportadora, tomada de una maquina selladora de banda continua horizontal FR-900, la cual es impulsada por un motor DC a 220V. Las especificaciones técnicas de la selladora, se presentan en la Tabla 6-1. La máquina se alimenta con 220 VAC a 50 HZ y su estructura es en acero inoxidable.

| MOTOR | |
|-------------|-----------------------|
| REFERENCIA: | ZYT 90-01 |
| CORRIENTE: | 0,32 A |
| VOLTAJE: | 220 VDC |
| VELOCIDAD: | 2000 r/min |
| POTENCIA: | 50 W |
| DIMENSIONES | |
| HORIZONTAL: | 850 x 420 x 320 (mm) |
| VERTICAL: | 850 x 320 x 550 (mm) |
| CONSOLA: | 850 x 320 x 1000 (mm) |

Tabla 6-1: Especificaciones Principales

6.2. Descripción de la maquina

En la Figura 6-1, se presenta un esquema general de la maquina selladora de banda continua horizontal FR-900, y en la Tabla 6-2 se describen las partes presentadas en la Figura 6-1.

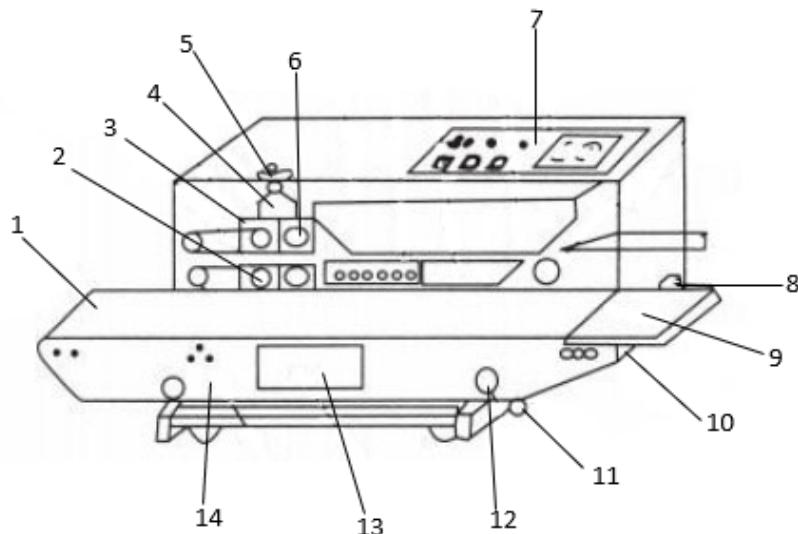


Figura 6-1: Maquina Banda Transportadora

| | |
|--------------------------------|-------------------------------------------------------------------------------|
| 1- Cinta transportadora | 8- Enchufe de corriente y protección |
| 2- Rueda de goma | 9- Mesa de trabajo fija |
| 3- Rodillo de goma | 10- Tornillo de regulación de la elasticidad de las cintas transportadoras |
| 4- Asiento rueda de tintorería | 11- Perilla de regulación de la entrada y salida de la estación de transporte |
| 5- Rueda reguladora de presión | 12- Perilla de regulación de la altura de la estación transportadora |
| 6- Rueda motriz | 13- Placa de identificación |
| 7- Caja de Control | 14- Estación de transporte |

Tabla 6-2: Descripción Maquina Transportadora

6.3. Esquema eléctrico

En la Figura 6-2 se presenta las conexiones eléctricas realizadas para el funcionamiento del prototipo, ademas de esto, se anexa en la Tabla 6-3 el listado de elementos implementados.

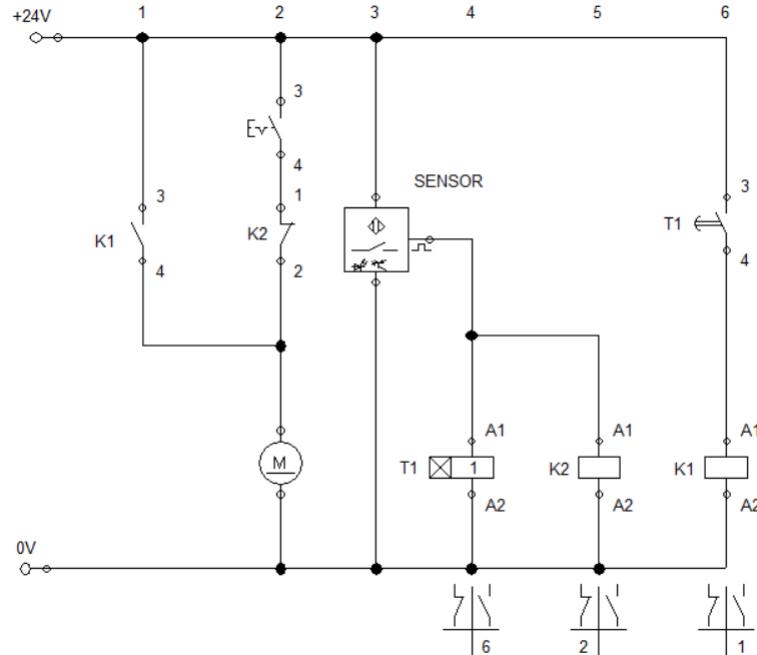


Figura 6-2: Esquema eléctrico

| SÍMBOLO | NOMBRE | CANTIDAD |
|---------|----------------------|----------|
| V1 | Fuente de Voltaje | 1 |
| T1 | Temporizador | 1 |
| K1 | Relé | 1 |
| M1 | Motor DC | 1 |
| SENSOR | Sensor Fotoeléctrico | 1 |
| K2 | Relé | 1 |

Tabla 6-3: Elementos Esquema Eléctrico

6.4. Estructura

La caja elaborada para el análisis de los tubérculos de papas, fue construida en acetato con medidas de, $35 \times 25(cm)$ las laminas frontal y trasera (Figuras 6-3a y 6-4b), $16 \times 25(cm)$ lamina superior (Figura 6-3b), $20 \times 16(cm)$ laminas laterales como se muestra en la Figura 6-4a.



(a) Lamina Frontal



(b) Lamina superior

Figura 6-3: Estructura de Análisis Laminas Principales



(a) Lamina lateral



(b) Lamina Trasera

Figura 6-4: Estructura de Análisis Laminas Laterales

Como se muestra en las Figuras , las laminas fueron diseñadas con las medidas necesarias para posicionar algunos elementos, que permitieran realizar la clasificación de los tuberculos, como lo son bombillos, un sensor optico y una camara web, los cuales seran definidos en los numerales 6.4.1, 6.4.2 y 6.4.3.

6.4.1. Bombillos

La estructura cuenta con dos agujeros, donde se ubican dos bombillos de luz blanca de 6500 Kelvin de temperatura, para mantener la iluminación fija, a una distancia de 30 cm de todas las fotos, al igual que se tomaron durante la construcción del Dataset.

6.4.2. Sensor Fotoeléctrico

Se implementa un sensor fotoeléctrico, de referencia MAGEWAY, el cual tiene como objetivo detener la banda transportadora una vez el tubérculo de papa se encuentre posicionado bajo la cámara, para realizar su respectiva inspección, el sensor cuenta con las siguientes especificaciones.

ESPECIFICACIONES TÉCNICAS:

- Método de detección: retroreflectante.
- Ángulo: 1,5 o aprox.
- Máx. Rango de detección: 4 m.
- Voltaje de alimentación: 12-240 VDC, 24-240V ACBR, Salida: Relé SPDT
- Capacidad de salida: 3 A/30 VCC, 3 A/250 VAC.
- Temperatura ambiente: -4-131 °F (-20-55 °C).

6.4.3. Cámara Web

Para la detección de los tubérculos de papa, se implementa una cámara *Web Camera America Store*, la cual será la encargada de la toma de fotos de los tubérculos, mientras son transportados en la banda transportadora para su análisis.

ESPECIFICACIONES TÉCNICAS:

- Lente: Lente de Cristal
- Tamaño del artículo: 8 x 4 x 8 cm
- Chip DSP: sin controlador
- Sensor de imagen: CMOS
- Resolución dinámica: 1920 x 1080
- Marco: 30 fps
- Longitud Focal: 8 mm - infinity
- Longitud del Cable: aproximadamente 138 cm

6.5. Funcionamiento

Aquí se describe el funcionamiento mecánico de la maquina y el funcionamiento de la red neuronal, la cual fue programada en la tarjeta Jetson Nano para realizar la clasificación de los tubérculos de papa.

6.5.1. Funcionamiento Mecánico

El funcionamiento mecánico inicia conectando la alimentación de la maquina, una vez la maquina se encuentre conectada, el indicador luminoso que se encuentra en el pulsador de inicio, se activará, como se muestra en la Figura 6-5, esto para dar a conocer que la maquina ya puede entrar en funcionamiento.

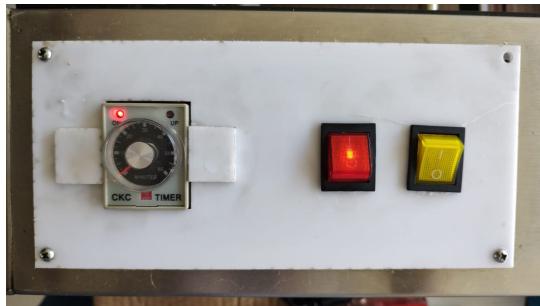


Figura 6-5: Indicador Luminoso

Se presiona el pulsador anteriormente mencionado, y se genera la conmutación del relé para la activación del motor, el cual se encuentra conectado a una caja reductora, que transmite la energía a tres engranajes rectos como se presentan en la Figura 6-6.



Figura 6-6: Caja Reductora De Transmisión A Engranajes Rectos

En donde el último engranaje se encuentra conectado, y transmite la energía a uno de los rodillos tensores de la banda transportadora, mediante un eje, como se aprecia en la Figura 6-7, con el fin de generar el movimiento de la banda.



Figura 6-7: Conexión Engranajes y Rodillo Tensor

Una vez se inicia el movimiento de la banda transportadora, este seguirá hasta que el tubérculo entre en contacto con el con la luz infrarroja del sensor foto eléctrico, como se observa en la Figura 6-8, cuando el sensor detecte el tubérculo, la señal enviada detendrá la banda y activara un timer de un segundo, esto con el fin de permitir que la cámara tome la foto del tubérculo, y la envié a la red neuronal para su análisis el cual se explicará en la sección 6.5.2 *Implementación en Sistema Embedido*. Culminado el tiempo establecido en el timer, la banda continuara su movimiento para desplazar y continuar con la clasificación de los tubérculos.



Figura 6-8: Detección del sensor fotoeléctrico

6.5.2. Implementación en Sistema Embedido

Para el funcionamiento del algoritmo en tiempo real, en el prototipo construido, se utilizó una tarjeta de desarrollo Nvidia Jetson Nano 2 GB, debido a que cuenta con una tarjeta gráfica y está diseñada para ejecutar modelos de redes neuronales. La implementación del algoritmo, debido a que la red fue entrenada con imágenes de tubérculos de papa estáticas, se realizó de igual forma.

El algoritmo captura una foto desde la webcam, realiza el procesamiento en la red neuronal y posteriormente, la clasificación por tamaño. Los resultados de cada imagen son mostrados por consola y la imagen capturada es guardada en formato *.jpg*, con la información de las predicciones, para verificar si la clasificación fue correcta. La imagen **6-9**, muestra una de las papas pasando dentro de la estructura hecha para la clasificación.



Figura 6-9: Tubérculo de Papa Dentro de La Estructura

Se realizó un experimento con 50 papas de las diferentes clases, para verificar la precisión del algoritmo. La Tabla **6-4** muestra los resultados de algunas de las papas utilizadas.

| Real | | Prediccion | |
|---------|------------|------------|------------|
| Clase | Tamaño | Clase | Tamaño |
| Clase 1 | Grande | Clase 1 | Mediano |
| Clase 1 | Muy Grande | Clase4 | Muy Grande |
| Clase 4 | Medianas | Clase 4 | Medianas |
| Clase 3 | Grande | Clase 3 | Grande |
| Clase 2 | Medianas | Clase 2 | Medianas |
| Clase 1 | Medianas | Clase 2 | Grande |
| Clase 4 | Grande | Clase 3 | Grande |
| Clase 3 | Medianas | Clase 3 | Medianas |
| Clase 1 | Medianas | Clase 1 | Medianas |

Tabla 6-4: Resultados Papas Predichas

De las 50 papas utilizadas para la verificación, se obtuvo un resultado de 80 % de clases con correcta predicción y 88 % de tamaños correctamente clasificados. Esto da una precisión total de 85 % para las papas que fueron correctamente clasificadas en ambas categorías.

El tiempo de ejecución del algoritmo, desde el momento en el que toma la foto y la procesa, hasta que realiza la clasificación, varía entre los 2 y 3 segundos. El tiempo que demora

la papa en llegar desde el comienzo de la banda transportadora, hasta la posición dentro de la estructura de análisis, es de 7 segundos, por lo tanto, se puede asumir un tiempo promedio de 10 segundos en analizar 1 tubérculo de papa. Este prototipo clasificaría 100 tubérculos de papa cada 15 minutos.

7 Trabajo Futuro

Debido a que el desarrollo del proyecto es un prototipo, la máquina puede ser mejorada para realizar la clasificación mecánica de las diferentes categorías, de igual forma, se puede implementar un variador para que la banda sea más rápida que el movimiento actual, mejorando así la velocidad de clasificación.

La red neuronal artificial que se obtuvo con una precisión superior al 83 %, puede ser mejorada de tal manera que se puedan obtener precisiones superiores al 90 %, para mejorar los resultados de las predicciones finales. Esto se puede lograr enriqueciendo el *Dataset*, mejorando la distribución de los datos iniciales, de manera que sean proporcionales entre sí y realizando una optimización de hiperparámetros, teniendo en cuenta diferentes parámetros utilizados en este proyecto. Si se desea realizar una mejor optimización, el poder de computo necesario para realizarla debe ser muy superior, en las condiciones actuales, la optimización tomó un tiempo de 6 horas por cada modelo en el que se aplicó.

Los tiempos de ejecución del algoritmo implementado en el sistema embebido, se deben mejorar, para que el prototipo sea viable en la agricultura de precisión. Mejor capacidad de procesamiento en la tarjeta de desarrollo escogida y mejorar la sintaxis y consumo de memoria por parte del diseño del algoritmo hecho en *Python*.

Bibliografía

- [1] Dane, “3 censo nacional agropecuario bogotá,” 2016.
- [2] R. L. Goyeneche Ortegón and Y. A. C. Jiménez Sánchez, “Dos miradas sobre el riesgo laboral: cultivadores de papa del municipio de toca, boyacá, colombia,” *Revista Ciencias de la Salud*, vol. 13, no. 2, pp. 249–259, 2015.
- [3] A. J. Avellaneda Barrantes, L. E. Ruiz Gómez, J. A. Monroy Naranjo, *et al.*, “Evaluación financiera del sistema de distribución de la producción de pequeños agricultores de papa en sabana centro cundinamarca,” B.S. thesis, Especialización en Administración Financiera Presencial, 2021.
- [4] “Sector papero se prepara para aumentar el consumo de papa en colombia — finagro.”
- [5] “El aporte de la papa a la economía colombiana • periódico el campesino – la voz del campo colombiano.”
- [6] “Industria alimentaria. papa para consumo. especificaciones: E: Food industries. potatoes for consumption. specifications,” 2018.
- [7] M. L. Marote, “Agricultura de precisión,” *Ciencia y tecnología*, pp. 143–167, 2010.
- [8] R. Salas, “Redes neuronales artificiales,” *Universidad de Valparaíso. Departamento de Computación*, vol. 1, pp. 1–7, 2004.
- [9] H. Wu, J. Zhang, K. Huang, K. Liang, and Y. Yu, “Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation,” *arXiv preprint arXiv:1903.11816*, 2019.
- [10] H. Jabbar and R. Z. Khan, “Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study),” *Computer Science, Communication and Instrumentation Devices*, vol. 70, 2015.
- [11] D. A. Van Dyk and X.-L. Meng, “The art of data augmentation,” *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001.
- [12] J. Durán, “Técnicas de regularización básicas para redes neuronales,” 2019.

- [13] Y. Zhu and C. Huang, “An improved median filtering algorithm for image noise reduction,” *Physics Procedia*, vol. 25, pp. 609–616, 2012.
- [14] J. C. Tello, “La visión artificial y las operaciones morfológicas en imágenes binarias,” *Una Perspectiva de la Inteligencia Artificial en su 50 Aniversario*, p. 570, 2006.
- [15] H. Gholamalinezhad and H. Khosravi, “Pooling methods in deep neural networks, a review,” *arXiv preprint arXiv:2009.07485*, 2020.
- [16] M. A. Islam, M. Kowal, S. Jia, K. G. Derpanis, and N. D. Bruce, “Position, padding and predictions: A deeper look at position information in cnns,” *arXiv preprint arXiv:2101.12322*, 2021.
- [17] G. I. Viera Maza, “Procesamiento de imágenes usando opencv aplicado en raspberry pi para la clasificación del cacao,” 2017.
- [18] J. Murphy, “An overview of convolutional neural network architectures for deep learning,” *Microway Inc*, pp. 1–22, 2016.
- [19] L. Torrey and J. Shavlik, “Transfer learning,” in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, IGI global, 2010.
- [20] N. Kulkarni, “Color thresholding method for image segmentation of natural images,” *International Journal of Image, Graphics and Signal Processing*, vol. 4, no. 1, p. 28, 2012.
- [21] I. García and V. Caranqui, “La visión artificial y los campos de aplicación,” *Tierra infinita*, vol. 1, no. 1, pp. 98–108, 2015.
- [22] J. Porras, M. De la Cruz, and A. Morán, “Clasification system based on computer vision,” *Escuela Profesional de Ingeniería Electrónica-Universidad Ricardo Palma: http://www. urp. edu. pe/pdf/ingenieria/electronica/CAP-1_Taller_de_Electronica_IV-b. pdf*, 2014.
- [23] C. Nandi, “Machine vision based automatic fruit grading system using fuzzy algorithm,” 02 2014.
- [24] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [25] L. Pencue-Fierro and J. León Téllez, “Detección y clasificación de defectos en frutas mediante el procesamiento digital de imágenes,” *REVISTA COLOMBIANA DE FISICA*, vol. 35, 01 2003.

- [26] “Soybean plant foliar disease detection using image retrieval approaches,” *Multimedia Tools and Applications*, vol. 76, pp. 26647–26674, 12 2017.
- [27] K. R. Singh and S. Chaudhury, “Efficient technique for rice grain classification using back-propagation neural network and wavelet decomposition,” *IET Computer Vision*, vol. 10, pp. 780–787, 12 2016.
- [28] K. Przybyl, P. Boniecki, K. Koszela, L. Gierz, and M. Lukomski, “Computer vision and artificial neural network techniques for classification of damage in potatoes during the storage process,” *Czech Journal of Food Sciences*, vol. 37, pp. 135–140, 5 2019.
- [29] T. Liu, W. Wu, W. Chen, C. Sun, C. Chen, R. Wang, X. Zhu, and W. Guo, “A shadow-based method to calculate the percentage of filled rice grains,” *Biosystems Engineering*, vol. 150, pp. 79–88, 2016.
- [30] C.-L. Chung, K.-J. Huang, S.-Y. Chen, M.-H. Lai, Y.-C. Chen, and Y.-F. Kuo, “Detecting bakanae disease in rice seedlings by machine vision,” *Computers and Electronics in Agriculture*, vol. 121, pp. 404–411, 2016.
- [31] R. D. L. Pires, D. N. Gonçalves, J. P. M. Oruê, W. E. S. Kanashiro, J. F. Rodrigues, B. B. Machado, and W. N. Gonçalves, “Local descriptors for soybean disease recognition,” *Computers and Electronics in Agriculture*, vol. 125, pp. 48–55, 2016.
- [32] C. Sun, T. Liu, C. Ji, M. Jiang, T. Tian, D. Guo, L. Wang, Y. Chen, and X. Liang, “Evaluation and analysis the chalkiness of connected rice kernels based on image processing technology and support vector machine,” *Journal of Cereal Science*, vol. 60, no. 2, pp. 426–432, 2014.
- [33] T. Liu, W. Chen, W. Wu, C. Sun, W. Guo, and X. Zhu, “Detection of aphids in wheat fields using a computer vision technique,” *Biosystems Engineering*, vol. 141, pp. 82–93, 2016.
- [34] R. SOBOLU, M. CORDEA, I. POP, L. ANDRONIE, D. PUSTA, *et al.*, “Automatic sorting of potatoes according to their defects.,” *Scientific Papers-Series B, Horticulture*, vol. 64, no. 1, pp. 462–468, 2020.
- [35] A. M. RODRIGUEZ, “Sistema automatizado de reconocimiento y manipulación de objetos usando visión por computadora y un brazo industrial,” 2014.
- [36] L. Han, M. S. Haleem, and M. Taylor, “A novel computer vision-based approach to automatic detection and severity assessment of crop diseases,” in *2015 Science and Information Conference (SAI)*, pp. 638–644, 2015.

- [37] M. Barnes, T. Duckett, G. Cielniak, G. Stroud, and G. Harper, “Visual detection of blemishes in potatoes using minimalist boosted classifiers,” *Journal of Food Engineering*, vol. 98, pp. 339–346, 2010.
- [38] V. D. A. AGROPECUARIOS and D. D. C. A. Y. FORESTALES, “Estrategia de ordenamiento de la producciÓn cadena productiva de la papa y su industria,” 2019.
- [39] J. M. B. Lópe, “DocumentaciÓn del proceso tÉcnico utilizado en el cultivo de papa en el municipio de chita boyacÁ vereda rechÍniga,” 2017.
- [40] V. Artificial, “Aplicación práctica de la visión artificial en el control de procesos industriales,” *Ministerio de Educación y Formación Profesional, Gobierno de España*, 2012.
- [41] “La camara fotografica cuerpo y controles basicos.”
- [42] A. E. de Normalización y Certificación, “Iluminacion de los lugares de trabajo en interiores,” 2003.
- [43] P. Liashchynskyi and P. Liashchynskyi, “Grid search, random search, genetic algorithm: A big comparison for nas,” *arXiv preprint arXiv:1912.06059*, 2019.
- [44] P. I. Frazier, “A tutorial on bayesian optimization,” *arXiv preprint arXiv:1807.02811*, 2018.
- [45] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, “Botorch: a framework for efficient monte-carlo bayesian optimization,” *Advances in neural information processing systems*, vol. 33, pp. 21524–21538, 2020.
- [46] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.
- [47] “Pytorch documentation — pytorch 1.10.1 documentation.”
- [48] V. Mathivet, *Inteligencia artificial para desarrolladores: conceptos e implementaciÓn en C*. Ediciones ENI., 2018.
- [49] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.
- [50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation, parallel distributed processing, vol. 1,” *Foundations. MIT Press, Cambridge*, 1986.

- [51] L. Bottou, “Stochastic gradient descent tricks,” pp. 421–436, 2012.
- [52] A. Krizhevsky, “One weird trick for parallelizing convolutional neural networks,” *arXiv preprint arXiv:1404.5997*, 2014.
- [53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [54] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [55] E. Díaz-Gaxiola, Z. E. Morales-Casas, O. Castro-López, G. Beltrán-Gutiérrez, I. F. V. López, and A. Y. Rendón, “Estudio comparativo de arquitecturas de cnns en hojas de pimiento morrón infectadas con virus phyvv o pepgmv.,” *Res. Comput. Sci.*, vol. 148, no. 7, pp. 289–303, 2019.
- [56] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [57] A. Atienza Arroyo *et al.*, “Detección e identificación automática de actrices y actores mediante el uso de algoritmos de deep learning,” 2019.
- [58] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.