

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**

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	4
1.3 Justificación	5
1.4 Descripción y formulación del problema	5
2 Marco Conceptual	7
2.1 Agricultura de precisión	7
2.2 Artificial Neural Networks	7
2.3 Convolución Dilatada	8
2.4 DATA AUGMENTATION	8
2.5 <i>Dropout</i>	9
2.6 Filtro de mediana	9
2.7 Imagen Binaria	9
2.8 <i>Max-pooling</i>	9
2.9 <i>Over-fitting</i>	10
2.10 <i>Padding</i>	10
2.11 Procesamiento de imágenes	11
2.12 <i>Stride</i>	11
2.13 Transfer Learning	12
2.14 <i>Thresholding</i>	12
2.15 <i>Under-fitting</i>	12
2.16 Visión artificial	12
3 Estado del arte	14
3.1 MatLab.	14
3.2 Artificial Neural Networks.	14
3.3 Super Vector Machine.	15

3.4	Otros métodos	16
3.5	Clasificación de Tuberculos de Papa En La Agricultura Colombiana	17
4	Diseño Del Algoritmo de Clasificación	18
4.1	Preparación y Adquisición De Las Imágenes	19
4.2	Distribución del Dataset	21
5	Algoritmo de Clasificación	22
5.1	Red Neuronal Artificial Convolucional	22
5.1.1	Preparación Del <i>Dataset</i>	23
5.1.2	Funciones PotatoDataset y DataLoader	26
5.1.3	Hiperparámetros Del Modelo	28
5.1.4	Arquitectura del modelo	30
5.1.5	Optimización Bayesiana	44
5.2	Clasificación Por Tamaño	46
6	Prototipo	49
6.1	Especificaciones principales	49
6.2	Descripción de la maquina	50
6.3	Esquema eléctrico	51
6.4	Estructura	52
6.4.1	Bombillos	52
6.4.2	Sensor Fotoeléctrico	52
6.4.3	Cámara Web	53
6.5	Funcionamiento	53
6.5.1	Funcionamiento Mecánico	53
6.5.2	Implementación en Sistema Embebido	56
7	Trabajo Futuro	58
	Bibliografía	59

1 Introducción

La agricultura ha sido uno de los pilares más importantes tanto para el desarrollo de la sociedad como para su manutención, ya que es gracias a ésta, que los países pueden generar empleos y aumentar sus recursos económicos, sin embargo las tareas de agricultura pueden llegar a ser tediosas o muy pesadas para las personas, es por esto que con el avance tecnológico se ha buscado mejorar el sector de la agricultura no solo para facilitar a los agricultores las tareas que deben hacer, sino también permitir que el producto ofrecido sea de mayor calidad, para garantizar que su comercialización tenga un mayor auge globalmente.

Uno de los principales avances tecnológicos en el sector agro-industrial, es la implementación de visión artificial en los procesos productivos, con esta se busca corregir las fallas humanas y garantizar que el producto mantenga siempre una buena y mejor calidad, que se busca garantizar con la normativa INCONTEC, la cual debe tener en cuenta para la exportación de productos agrícolas. El proyecto busca desarrollar un sistema mecatrónico implementando técnicas de visión artificial para mejorar el proceso de producción de la papa, esto podría ayudar al crecimiento del sector papiculor del país, debido a que se espera mejorar la calidad del producto y sea más eficiente su clasificación.

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 “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 NTC341.
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 acabo el análisis por técnicas de visión artificial. El análisis se enfocará en identificar las características de calidad del producto para ser comercializado. Se propone un sistema mecatrónico tipo 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 open sources a las que se le puedan sacar provecho, para la implementación de los algoritmos que actualmente se encuentran en la literatura.

1.3. Justificación

En el proyecto se desarrollará un sistema mecatrónico, implementando técnicas de visión artificial, para agilizar el proceso de producción de la papa, con el fin de mejorar el proceso actual que se lleva a cabo para la clasificación de calidad en los tubérculos de papa. 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 90.000 familias principalmente en los departamentos de Boyacá, Cundinamarca, Antioquia y Nariño, los cuales concentran más del 85 % de la producción [1].

Actualmente, el reto que enfrenta el sector es el aumento de la calidad del producto, el rendimiento en producción y en consecuencia, incrementar el consumo. Debido a que la producción de papa en Colombia aporta el 3.3 % del Producto Interno Bruto, (PIB), agropecuario, las siembras son de alrededor de 130 mil hectáreas y se cosechan cerca de 2,8 millones de toneladas, Además, en Colombia la producción de papa genera anualmente alrededor de 264.000 empleos, aproximadamente 75.000 son trabajos directos y alrededor de 189.000 son indirectos [2]. Así, el presente trabajo permitirá mostrar los beneficios que trae la implementación de técnicas de visión artificial que podrían generar un crecimiento al sector papicultor del país para competir con el mercado internacional.

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.

Uno de los sectores más grandes en la agricultura colombiana es el sector papicultor, en Colombia se caracteriza por tener poco desarrollo tecnológico y buscar 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, el sector papicultor requiere urgentes transformaciones que permitan aumentar su competitividad y por ende lograr el crecimiento del sector.

Como consecuencia, en los últimos años, el sector agricultor de Colombia ha implementado herramientas tecnológicas que le permitan a los agricultores, aumentar la calidad de sus productos y así 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 sin

importar su clase, sin embargo, las técnicas de visión artificial no han sido aprovechadas para mejorar el sector papicultor colombiano.

El sector agricultor de Colombia ha implementado diferentes técnicas de visión artificial, las cuales ayudan a clasificar los productos mediante las características que estos poseen, sin embargo, no se han implementado estas técnicas en la fase de almacenamiento de los tubérculos de papa, la cual es importante ya que en esta fase es donde se separan los tubérculos que cumplen las condiciones de calidad de los tubérculos que se encuentran defectuosos, ya que para la comercialización de este producto deben estar clasificados bajo la norma NTC 341.

¿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

En este capítulo se abordan algunos términos básicos necesarios, que ayuden al entendimiento y desarrollo del proyecto.

2.1. Agricultura de precisión

El término sobre el que se inspira la agricultura de precisión, es utilizar la porción adecuada de insumos, en el instante correcto y en el sitio preciso. La agricultura de precisión, (AP), implica la utilización de sistemas de posicionamiento universal, (GPS), y de otros medios electrónicos, para obtener datos del cultivo. Las tecnologías de la agricultura de precisión, permiten saciar una de las exigencias de la agricultura actualizada: el funcionamiento óptimo de enormes extensiones. Se muestra como primordial virtud, que la exploración de resultados de los ensayos se puede hacer por sectores diferentes en un mismo lote, y tal cual ajustar el desempeño diferencial en los mismos. La utilización de las tecnologías de la agricultura de precisión puede contribuir a mejorar los márgenes, por medio de un incremento del costo del rendimiento, (cantidad o calidad), de una reducción en la proporción de insumos, o de los dos paralelamente [3].

2.2. Artificial Neural Networks

Las RNA son sistemas de procesamiento de la información, cuya composición y manejo permanecen inspirados en las redes neuronales biológicas. Consisten en un enorme conjunto de recursos básicos de procesamiento, denominados nodos o neuronas, que permanecen organizados en capas. De esta forma, las RNA son sistemas adaptativos que aprenden de la vivencia, en otros términos, aprenden a realizar ciertas labores por medio de un entrenamiento con ejemplos ilustrativos [4].

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 [5]. La Figura 2-1 muestra un ejemplo de la implementación de la convolución dilatada.

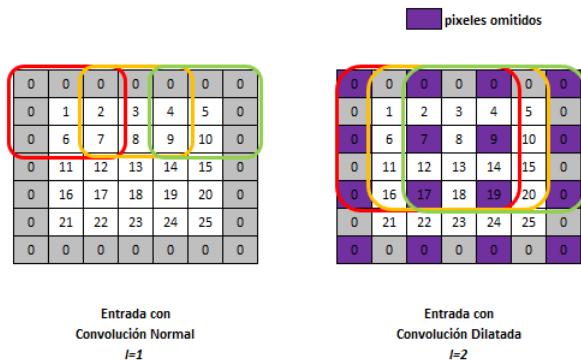


Figura 2-1: Ejemplo Convolución Dilatada

2.4. 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 crear 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 sobreajuste.

El concepto Data Augmentation tiene relación con los procedimientos para edificar algoritmos iterativos de mejora o muestreo, por medio de la introducción de datos no vigilados o cambiantes latentes. Para los algoritmos estocásticos, el procedimiento se popularizó en la literatura estadística por el algoritmo de incremento de datos de Tanner y Wong y en la literatura de física por el algoritmo de Swendsen y Wang, para el muestreo de los modelos de Ising y Potts y sus generalizaciones; en la literatura de física, el Data Augmentation, se conoce como el procedimiento de las cambiantes auxiliares. Generalmente, no obstante, la obra de esquemas de el Data augmentation es que den sitio a algoritmos básicos y rápidos,

debido a que las tácticas famosas varían de una manera significativa con respecto a los modelos.[6]

2.5. Dropout

Para cada entrenamiento de redes neuronales, la propagación hacia adelante involucra la eliminación aleatoria de la mitad de activaciones en cada capa. Haciendo que el error se extienda hacia atrás a través de las activaciones remanentes, reduciendo sustancialmente al sobreajuste. Se cree que es debido al impedimento de que los pesos de la red colaboren entre sí para memorizar las muestras del entrenamiento mejorando su rendimiento [7]

2.6. 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 [8].

2.7. Imagen Binaria

Se define una imagen binaria como una función de dos variables discretas $a[m,n]$ las cuales puede tomar dos valores, ‘0’ o ‘1’, dependiendo del nivel de gris de la imagen (una imagen binaria tiene dos niveles: blanco y negro). Existe una definición alternativa si consideramos que una imagen consiste en un conjunto de coordenadas discretas (también pueden ser reales pero no es el objetivo de este estudio). En este sentido, el conjunto corresponde a todos aquellos puntos o píxeles que pertenecen a la imagen. Por lo tanto, se puede decir que en morfología matemática los conjuntos representan objetos en una imagen [9].

2.8. Max-pooling

Una capa de agrupación normal, realiza un muestreo decreciente dividiendo la entrada en agrupaciones rectangulares y calculando los valores máximos de cada agrupación.

Un operador max-pooling, se puede hacer uso 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 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 $N \neq R$. 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 $p = [p_1, \dots, p_M] \in R^{M,J}$ [10], un ejemplo de max-pooling se puede observar en la Figura 2-2.

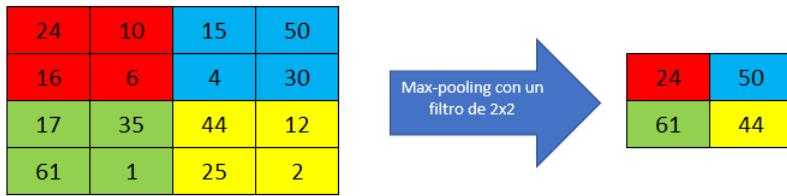


Figura 2-2: Ejemplo Max-Pooling

2.9. Over-fitting

Un problema al momento de entrenar redes neuronales, es el *over-fitting*, al hacer que la red neuronal en un momento específico del ciclo de entrenamiento no muestra un progreso en la capacidad de resolver problemas. Sino que solamente aprende una regularidad aleatoria dentro del conjunto de patrones de entrenamiento [11].

2.10. Padding

Se utiliza para acomodar el dominio finito de las muestras a lo cual permite que el soporte de la convolución se extienda mas allá de los límites de las mismas, reduciendo el impacto de los efectos de frontera.

El mas utilizado 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 [12], 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

2.11. Procesamiento de imágenes

El procesamiento digital de imágenes es el conjunto de prácticas que modifican imágenes digitales para mejorar la visibilidad de ciertas características de los objetos presentes en la imagen, para su posterior análisis o simplemente para mejorar la visualización de la imagen.

El objetivo del procesamiento de imágenes no es aumentar la información que se puede extraer de las imágenes, solo se busca realzar ciertas características de la imagen. Para procesar la imagen efectivamente se debe considerar el proceso de formación y las características de interés de la imagen [13].

2.12. 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 [14], un ejemplo de stride se puede observar en la Figura 2-4.

<table border="1"> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>0</td></tr> <tr><td>0</td><td>5</td><td>6</td><td>7</td><td>8</td><td>0</td></tr> <tr><td>0</td><td>9</td><td>10</td><td>11</td><td>12</td><td>0</td></tr> <tr><td>0</td><td>13</td><td>14</td><td>15</td><td>16</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	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	<table border="1"> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>0</td></tr> <tr><td>0</td><td>5</td><td>6</td><td>7</td><td>8</td><td>0</td></tr> <tr><td>0</td><td>9</td><td>10</td><td>11</td><td>12</td><td>0</td></tr> <tr><td>0</td><td>13</td><td>14</td><td>15</td><td>16</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	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	<table border="1"> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>0</td></tr> <tr><td>0</td><td>5</td><td>6</td><td>7</td><td>8</td><td>0</td></tr> <tr><td>0</td><td>9</td><td>10</td><td>11</td><td>12</td><td>0</td></tr> <tr><td>0</td><td>13</td><td>14</td><td>15</td><td>16</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table>	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	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	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	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																																																																																																									

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

2.13. Transfer Learning

El aprendizaje por transferencia se define 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. Aunque la mayoría de los algoritmos de aprendizaje automático están diseñados para abordar tareas individuales, el desarrollo de algoritmos que faciliten el aprendizaje por transferencia es un tema de interés permanente en la comunidad de aprendizaje automático. El aprendizaje por transferencia se utiliza para mejorar a un alumno de un dominio mediante la transferencia de información de un dominio relacionado. Podemos basarnos en experiencias no técnicas del mundo real para entender por qué es posible el aprendizaje por transferencia.

Como en el caso del aprendizaje semisupervisado, los métodos de aprendizaje por transferencia se utilizan cuando hay una escasez directa de los datos de entrenamiento subyacentes. Sin embargo, la diferencia con el aprendizaje semisupervisado es que, en lugar de utilizar datos sin etiquetar, se utilizan datos etiquetados de un dominio diferente para mejorar [15].

2.14. Thresholding

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 [16].

2.15. Under-fitting

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 [11].

2.16. Visión artificial

La visión artificial o visión por computador intenta replicar la capacidad que tienen algunos seres vivos para visualizar una imagen, entenderla y actuar en base a lo observado. Existe un crecimiento en los diferentes tipos de aplicaciones industriales que requieren el

uso de técnicas de visión artificial. El continuo desarrollo de nuevos algoritmos y aplicaciones hacen de esta disciplina una tecnología en constante evolución la cual ha experimentado un rápido avance en las últimas décadas, así lo demuestran las numerosas investigaciones y publicaciones existentes en la comunidad científica. Esto podría deberse a la gran cantidad de contenido visual como imágenes y video que se genera en la actualidad, así como la capacidad de procesamiento y almacenamiento que tienen los dispositivos electrónicos y la disponibilidad de herramientas, librerías y lenguajes de programación para dicho fin [17].

3 Estado del arte

El objetivo de los antecedentes, es analizar y realizar una revisión literaria sobre lo que se ha hecho a nivel teórico y práctico en cuanto a la visión artificial en agricultura de precisión. Además, determinar cuáles son los aportes que éstos le puedan dar a este trabajo de investigación.

3.1. MatLab.

Comenzando por la literatura con temática en el procesamiento de imágenes, se tiene el trabajo de textitJ. porras, A. Morian [18], 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 qué 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. El artículo de C. Nandi [19] presenta una investigación en dónde 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. Artificial Neural Networks.

En cuanto a la clasificación, se deben usar técnicas de visión artificial y en primer lugar, se revisó literatura acerca de ANN, (*Artificial Neural Networks*), como el trabajo de L. Pencue-Fierro y J. León Téllez [20], en donde se aborda una investigación hecha en Perú, en donde se aplicó visión artificial usando ANN 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. En el trabajo publicado en la revista *textitMultimedia Tools and Applications* [21], donde presentaron 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 ANN 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. Otro trabajo es el publicado por *Ksh. Robert Singh y Saurabh Chaudhury* [22], 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 implementó una red neuronal BPNN de cuatro capas la cual presentó mejores resultados en comparación con otros métodos.

Por último, usando el método de clasificación ANN, se tiene el trabajo hecho por *Krzysztof Koszela y Lukasz Gierz* [23], 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 ANN. 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.

Otro método de clasificación revisado en la literatura es el de SVM (*Super Vector Machine*), como el usado en el trabajo de *Tao Liu* y otros investigadores [24], 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* y otros investi-

gadoores [25], 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* [26], 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. Y por último, en este tipo de clasificador, el trabajo de *Chengming Sun y Tao Liu* [27], se 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

En la literatura se encontraron artículos que usaban más de un método de clasificación, un ejemplo es el trabajo hecho por *Tao Liu y Wen Chen* [28], el cual 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.

De igual manera, el artículo de *Rodica Sobolu* [29], 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, en donde se usaron los métodos SVM, *Decision Tree* y LDA, (*Linear Discriminant Analysis*), implementados en MATLAB® Classification Tool-

box. Se llegó a la conclusión de que la SVM ha clasificado las patatas según su tamaño con una mayor tasa de éxito. En el caso de la clasificación por calidad, se recomienda el método LDA.

Los anteriores son los métodos más comunes que se encontraron en la literatura investigada, sin embargo, hay unos métodos poco comunes como por ejemplo los presentados en el artículo de *Alberto Martínes Rodriguez* [30] en 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* [31], que se usa para la detección automática, fue necesario hacer uso inicialmente de separar el fondo de la hoja, para este proceso se usó el algoritmo MCW, (*Marker-Controlled Watershed*). El SLIC, (*Simple Linear Iterative Clustering*), se utilizó para obtener las características presentadas por la enfermedad sobre la planta y por ultimo, 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 presento un mayor rendimiento.

El último artículo revisado es presentado por *Michael Barnes y Tom Duckett* el [32], 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 Tuberculos de Papa En La Agricultara Colombiana

4 Diseño Del Algoritmo de Clasificación

Se diseñó un algoritmo de clasificación utilizando Redes Neuronales Convolucionales y técnicas de visión artificial con la librería *Open CV*, para clasificar las características de tipo, daño y tamaño de tubérculos de papa. El Procedimiento que se llevó acabo se presenta en la Figura 4-1, cada ítem será explicado en las siguientes secciones.

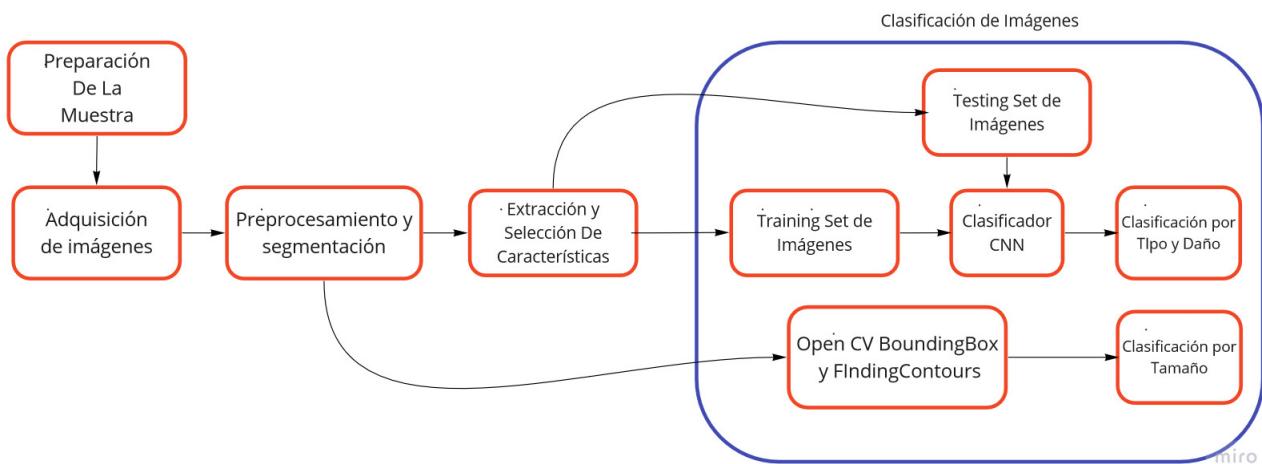


Figura 4-1: Arquitectura del Sistema de Clasificación

El sistema propuesto 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* y la librería *Open CV* de *Python*. Será implementado en un prototipo de banda transportadora, que permita el movimiento de los tubérculos de papa para ser clasificados, mediante un microcontrolador y una cámara. Los sistemas de Visión Artificial que se destinan a realizar inspecciones visuales, permiten que el proceso sea más eficiente si requieren alta velocidad, gran aumento en cantidad de producción y en funcionamiento las 24 horas del día o la repetibilidad de las medidas [33].

4.1. Preparación y Adquisición De Las Imágenes

Se preparó 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 *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 [34], 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, y la cámara fue fijada a un trípode ubicado a 30 cm de la base, para que la distancia del lente de la cámara con respecto a la muestra de papa fuera siempre la misma en las 592 fotos.



Figura 4-2: Récamara Para la Toma de Fotos

Utilizando la librería *Pandas* de *Python*, se creó un *Dataframe* que contiene el metadata de cada imagen. En la Tabla 4-1 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 4-1: 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 4-2.

PASTUSA	R12	Buena	Defectuosa	Clase
X		X		CLASE 1
X			X	CLASE 2
	X	X		CLASE 3
	X		X	CLASE 4

Tabla 4-2: 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

Para determinar la distribución total del dataset, con el uso de la librería *Plotly*, se crearon gráficas para determinar el porcentaje de imágenes pertenecientes cada característica definida en el *Dataset*. La Figura 4-3



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

Se definieron los *Set's* de *Train* y *Test*, que fueron creados para el entrenamiento y validación de la red neuronal, a partir del archivo *metadata.csv* de la Tabla 4-1. Se utilizó 80 % para *Train* y 20 % para *Test*. 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 %.

5 Algoritmo de Clasificación

Se desarrolló una red neuronal artificial, (*CNN*), para clasificar las características de tipo y daño, de los tubérculos de papas descritos en el *Metadata Tabla 4-1*. En este capítulo se presenta la definición y estructura de una red neuronal artificial convolucional, la preparación del dataset implementado durante el entrenamiento de la red neuronal y los diferentes modelos implementados en el desarrollo del proyecto.

5.1. Red Neuronal Artificial Convolutacional

La arquitectura general de una red artificial convolucional, (*CNN*), se muestra en la Figura 5-1 [35]. Se realizaron pruebas con las arquitecturas *Alexnet*, *Resnet18*, *VGG11* y *VGG19* y se implementó la optimización bayesiana en ciertos hiperparámetros de las arquitecturas existentes, para mejorar los resultados.

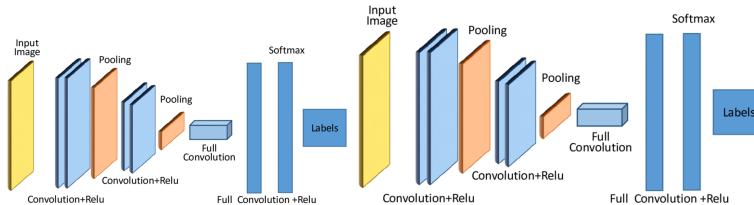


Figura 5-1: Arquitectura General De Una CNN

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

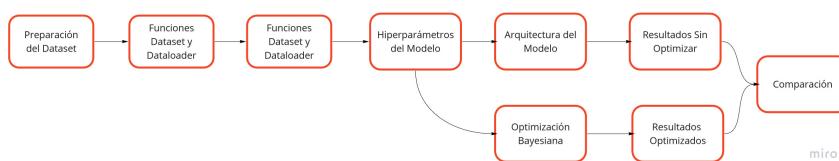


Figura 5-2: Procedimiento Diseñado Para Implementar La *CNN*

5.1.1. Preparación Del Dataset

Para realizar un correcto entrenamiento de un modelo de red neuronal, es necesario aplicar un preprocesamiento a las imágenes para que el modelo extraiga características pertinentes a las clases definidas. En este apartado se explica como se lleva a cabo 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*. 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. Las transformaciones disponibles se presentan en la Tabla 5-1.

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 "pad" dado	TenCrop	Recorta la imagen dada en cuatro esquinas y el recorte central
RandomAffine	Transformación afín aleatoria de la imagen manteniendo el centro invariante	GaussianBlur	Desenfoca la imagen con un filtro gausiano

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

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
RandomCrop	Recorta la imagen aleatoriamente	RandomInvert	Invierte los colores de la imagen de forma aleatoria
RandomGrayscale	Convierte la imagen en escala de grises aleatoriamente	RandomPosterize	Reduce el número de bits de cada canal de la imagen de forma aleatoria
HorizontalFlip	Voltea horizontalmente la imagen al azar con una probabilidad dada	RandomSolarize	Invierte aleatoriamente el valor de los pixeles por encima de un umbral
RandomPerspective	Realiza una transformación de perspectiva aleatoria de la imagen	RandomAutocontrast	Autocontraste de los pixeles 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*

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. Se escogieron los filtros considerados mejores, (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 esténdar sean 0 y 1 respectivamente. Para hacer esto se utiliza la Ecuación 5-1 [36]

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

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 esténdar, para *Trainset* y para el *Testset*. De esta forma se garantiza que la normalización sea correcta en el total del *Dataset*. Los valores obtenidos para el *Trainset* Ecuación 5-2 y el *Testset* Ecuación 5-3 respectivamente.

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

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

La Figura 5-3 enseña una matriz de imágenes como parte del *Trainset*, que muestra las transformaciones hechas al 80 % de las imágenes del *Dataset*, y la clase a la que pertenece la Tabla 4-2, verificando así que las funciones creadas están enlazando correctamente la imagen con su respectiva clase y aplicando de forma correcta las transformaciones.

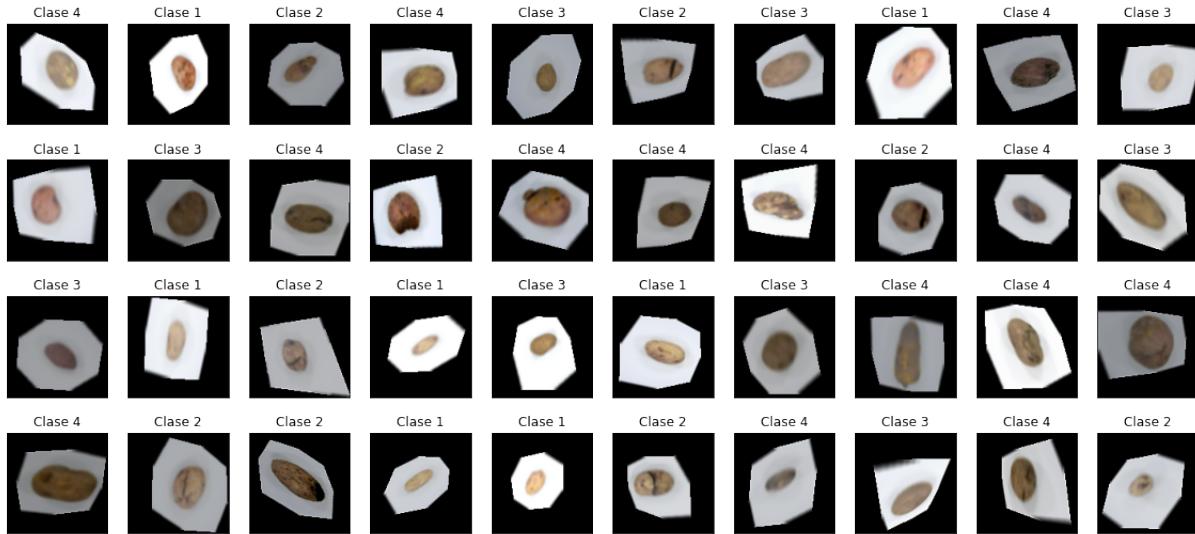


Figura 5-3: Matriz de Imágenes Con Transformaciones Aplicadas

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 este capítulo se presenta 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, (*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 4-1 se muestra como son ingresadas las imágenes en el algoritmo, 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 4-2. 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 4-1, debido a esto es que se desarrolló una función que cargara y relacionara de la misma manera que lo hace la función integrada de *Pythorch*. 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. Para el uso de esta función se debe tener en cuenta que el argumento más importante será el *dataset*, que fue generado por el objeto *PotatoDataset()*.

Según como se explico en el apartado anterior, se generó un *dataset* del tipo *iterable-style Datsets*, lo que permite utilizar la función integrada *DataLoader()* de *Pytorch*. El *DataLoader* permite la agrupación automática de muestras de datos individuales obtenidas en los *Batch's* mediante argumentos. Los *DataLoader's* cuentan con una gran cantidad de argumentos que ayudan a que la carga de datos sea más eficiente [36]. Es por esta razón, que para la ejecución de este dataloader se implementaron 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:

- $\text{Batch Size} = 40$
- $\text{Shuffle} = \text{True}$
- $\text{Num workers} = 0$

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

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. En este capítulo se abordan los hiperparámetros utilizados en el desarrollo de este proyecto.

FUNCIÓN DE PÉRDIDAS

Pytorch posee una cantidad interesante de funciones de pérdidas, para su uso 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. Cuanto menor sea el valor de la función, es más eficiente la red neuronal [37]. Para el desarrollo de la red neuronal, se utilizó la pérdida de entropía cruzada entre la entrada y el objetivo [36]. 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 4-2, por lo tanto, una función de pérdida no reducida [36] 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.

OPTIMIZADOR

Pytorch posee el paquete *torch.optim*, que implementa varios algoritmos de optimización basados en el gradiente descendiente. A menudo se ha propuesto, por ejemplo Rumhart [38], minimizar el riesgo empírico $E_n(fw)$ utilizando el gradiente descendiente. Consiste en cada iteración actualizar los pesos w .

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

Donde γ es la taza de aprendizaje elegida adecuadamente (*learning rate*). 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 [39].

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

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 definidos 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, VGG-19, VGG-11 y ResNet-18, 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. [36]

El modelo se compone de cinco capas convolucionales y tres capas densas, donde las capas convolucionales están normalizadas por lotes. El kernel 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 capa convolucional, están seguidas por una capa Max-pooling con dimensiones de 3×3 , que posee un stride de dos.

Las capas convolucionales y de Max-pooling son seguidas por tres capas densas, en donde las dos primeras poseen 4096 neuronas cada una y la ultima capa es la de salida, la cual se compone de 1000 neuronas que poseen una función de activación softmax. Estas neuronas son las encargadas de realizar la clasificación de la imagen. Como es habitual en estos sistemas de clasificación, la función de perdidas implementada es la entropía cruzada [40].

Como se aprecia en la Figura 5-4, el modelo de AlexNet, toma una imagen de entrada con una resolución de 227×227 pixeles y 3 canales de color, la información recibida pasa por cinco capas convolucionales, (en negro), de neuronas ReLu y tres capas de Max-pooling,

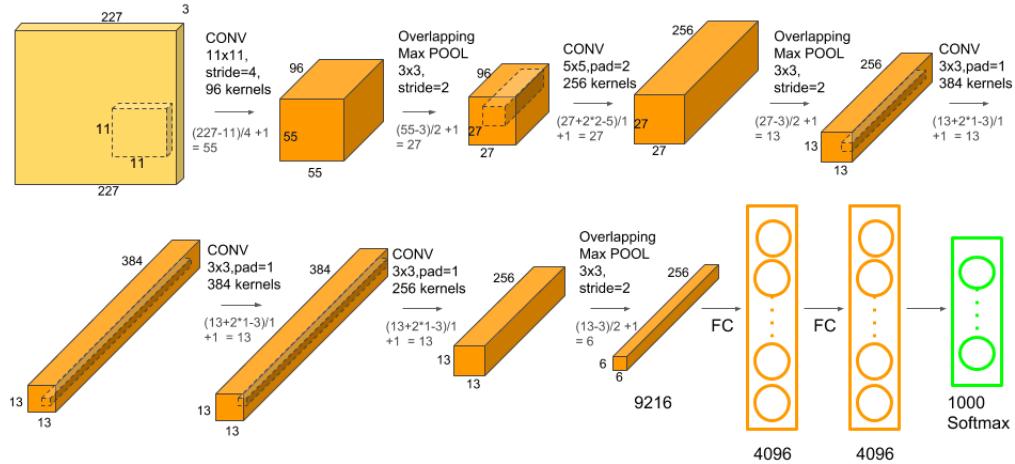


Figura 5-4: Modelo AlexNet

(en azul). La quinta capa de convolución posee 256 mapas diferentes con dimensiones de 13×13 . Seguidamente se aplican las dos capas densas de 4096 neuronas, las cuales son seguidas por una capa de salida de 1000 neuronas, dotadas con una función softmax. Cada neurona de la capa de salida representa un conjunto semántico diferente. Y su valor de activación indica la probabilidad de que la imagen de entrada pertenezca a dicho conjunto semántico.

MODELO PREENTRENADO VGG19

La arquitectura VGG fue planteada por Simonyan y Zisserman, la cual consiste en stacks lineales de bloques, que se encuentran formados por una cantidad determinada de capas convolucionales, una función de activación no lineal y una capa Max Pooling, seguidos de tres capas fully-conected y finalmente una capa softmax [41]. Esta arquitectura cuenta con cinco bloques los cuales están distribuidos como se muestra en la Figura 5-5.

Como se puede observar, la arquitectura cuenta con dos capas convolucionales de 64 y 128 filtros respectivamente, en sus dos primeros bloques, el bloque intermedio se compone de tres capas convolucionales de 256 filtros, y los últimos dos bloques están compuestos por tres capas concolucionales de 512 filtros cada uno. El numero 19 representa la cantidad de capas entrenables que posee la arquitectura, donde hay 16 capas convolucionales y 3 capas fully-connected.

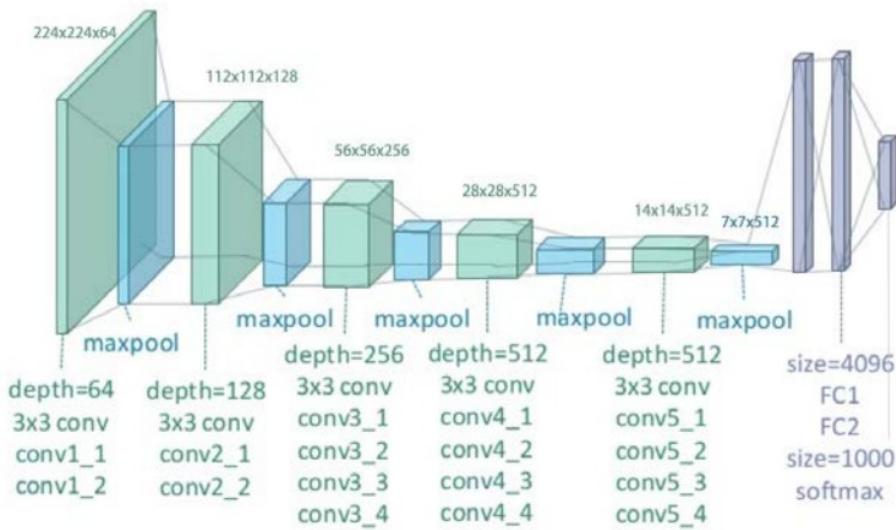


Figura 5-5: Modelo VGG19

Las capas convolucionales de esta arquitectura cuentan con un campo receptivo de 3×3 , stride de 1×1 y padding de 1 pixel. Para las operaciones del Max Pooling se implementa un kernel de 2×2 y un stride de 2×2 , y por último cada capa oculta de la red cuenta con la función de activación ReLu. La arquitectura VGG19 se resume en la Figura 5-6.

	Tipo	N filtros/parámetro
BLOQUE 1	Conv2D	64
	Conv2D	64
BLOQUE 2	Max Pool	N/A
	Conv2D	128
BLOQUE 3	Conv2D	128
	Max Pool	N/A
BLOQUE 4	Conv2D	256
	Conv2D	256
BLOQUE 5	Conv2D	256
	Conv2D	256
BLOQUE 4	Max Pool	N/A
	Conv2D	512
BLOQUE 5	Conv2D	512
	Conv2D	512
BLOQUE 4	Conv2D	512
	Conv2D	512
BLOQUE 5	Max Pool	N/A
	Fully-Conn	4096
BLOQUE 4	Fully-Conn	4096
	Fully-Conn	1000
BLOQUE 5	Softmax	N

Figura 5-6: Composición Modelo VGG19

Ya que se conoce la forma en que esta estructurado el modelo, se procede a mostrar la precisión del algoritmo antes de ser aplicada la optimización bayesiana.

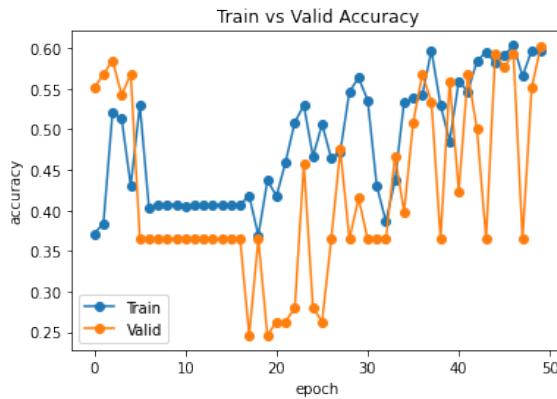


Figura 5-7: Accuracy Modelo VGG19

Como se puede observar en la Figura 5-7, la precisión del modelo al completar 50 épocas es inferior al 60%. En la Figura 5-8, se puede apreciar que el modelo cuenta con 16% aproximadamente de desviación entre las predicciones y los valores reales, implementados durante el entrenamiento del modelo.

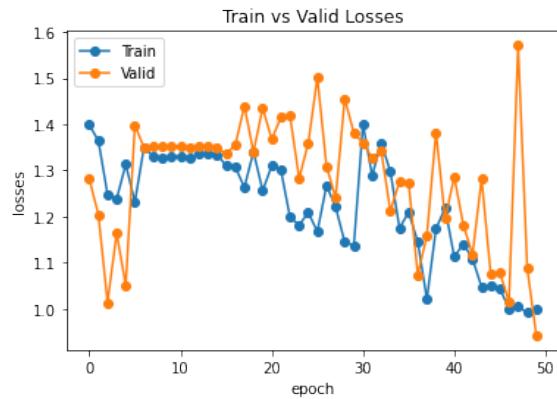
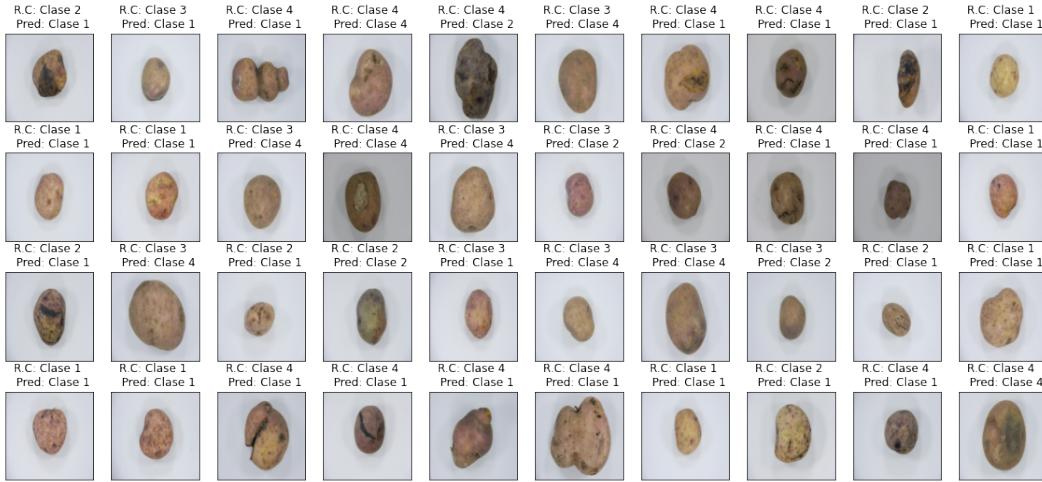


Figura 5-8: Losses Modelo VGG19

Teniendo en cuenta la información presentada en las Figuras 5-7 y 5-8, se puede apreciar que el modelo presenta *overfitting* durante el transcurso de las épocas 18 hasta la época 30, y finalmente culminada la época 50 del modelo, el porcentaje de *overfitting* es aproximadamente cero, y se procede a presentar en la Figura 5-9 la predicción realizada por el modelo.

**Figura 5-9:** Predicción Modelo VGG19

Se genera la matriz de confusión para validar los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos, y se plasman en la Figura 5-10.

		PREDICCIÓN			
		CLASE 1	CLASE 2	CLASE 3	CLASE 4
OBSERVACIÓN	CLASE 1	27	2	0	0
	CLASE 2	21	1	0	2
	CLASE 3	6	5	0	11
	CLASE 4	20	15	0	8

Figura 5-10: Matriz de confusión modelo VGG19

Para finalizar en la FIGURA 5-11, se presenta la precisión del modelo VGG-19 por clase.

CLASE 1	75,9
CLASE 2	50
CLASE 3	90,9
CLASE 4	51,2
Overall Accuracy:	65,78947

Figura 5-11: Precisión por clase modelo VGG19

MODELO PREENTRENADO VGG11

VGG es un modelo preentrenado, en un conjunto de datos que contiene el peso que representan las características del conjunto de datos entrenados. Los modelos VGG toman una imagen de entrada de dimensiones 224×224 pixeles en formato RGB, que son tratadas con un tamaño de imagen constante.

El modelo de aprendizaje profundo VGG11. Es la configuración mas sencilla. Tiene 11 capas de peso en total, de ahí el nombre VGG11. 8 de ellas son capas convolucionales y 3 son capas completamente conectadas, como se aprecia en la Figura 5-12 [42].

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
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
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
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 5-12: Composición Modelo VGG-11

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 nodos
- Totalmente conectado con 4096 nodos
- Capa de salida con activación Softmax con 1000 nodos.

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.

El Max-Pooling es una operación de pooling que calcula el valor máximo que contiene cada parche en cada mapa de características, como resultado se obtiene mapas de características muestreados que destacan la característica más relevante en el parche, en la práctica se ha comprobado que este método, es más eficiente que la agrupación media en tareas de visión por ordenador como la clasificación de imágenes.

Una vez conocida la forma en que se estructura el modelo VGG-11, se procede a evaluar su precisión antes de ser aplicada la optimización bayesiana y durante 50 épocas.

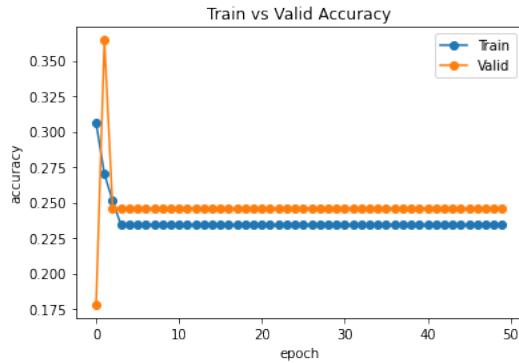


Figura 5-13: Precisión Modelo VGG-11

Como se evidencia en la Figura 5-13, el modelo cuenta con una precisión que ronda el 25 %.

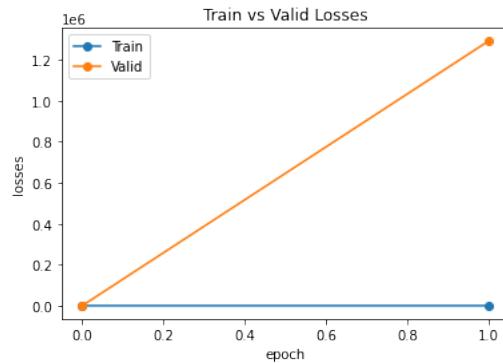


Figura 5-14: Losses Modelo VGG-11

En la Figura 5-14, se aprecia que el modelo cuenta con una desviación bastante significativa, entre las predicciones y los valores reales. Gracias a la información brindada por las figuras anteriores se deduce que el sistema posee un alto porcentaje de *overfitting* que no puede ser corregido en el transcurso de las épocas.

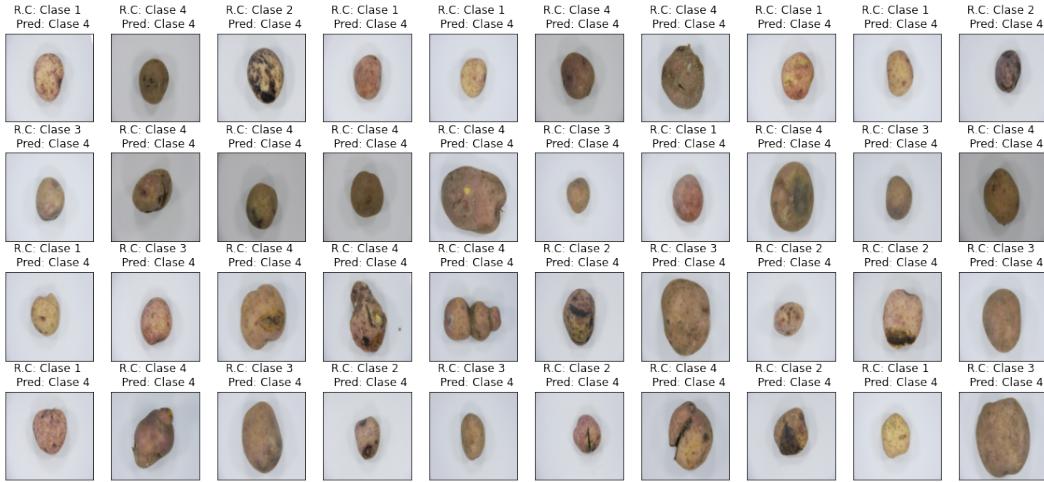


Figura 5-15: Predicción Modelo VGG-11

En la Figura 5-15, se presenta la predicción del modelo VGG-11 con las muestras implementadas para su validación.

		PREDICCIÓN			
		CLASE 1	CLASE 2	CLASE 3	CLASE 4
OBSERVACIÓN	CLASE 1	0	0	0	29
	CLASE 2	0	0	0	24
	CLASE 3	0	0	0	22
	CLASE 4	0	0	0	43

Figura 5-16: Matriz de confusión Modelo VGG-11

La figura 5-16, se presenta la matriz de confusión del modelo VGG-11, donde se aprecian los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

Para finalizar, la precisión de clasificación del modelo VGG-11 por clases sin optimización bayesiana se presenta en la Figura 5-17.

CLASE 1	0
CLASE 2	0
CLASE 3	0
CLASE 4	100
Overall Accuracy: 34,21052	

Figura 5-17: Precisión por clase Modelo VGG-11

MODELO PREENTRENADO RESNET18

La arquitectura Resnet, (Residual Net), fue propuesta por un equipo de Microsoft Research el cual fue liderado por Kaiming He, 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, (56 capas), debería tener exactamente la misma eficacia y no ser inferior que la primera. Debemos recordar que la función identidad es la encargada de devolver exactamente el mismo valor que su argumento [43].

Conociendo esto para permitir que una red pueda variar su cantidad efectiva de capas, se introduce el concepto de bloque residual:

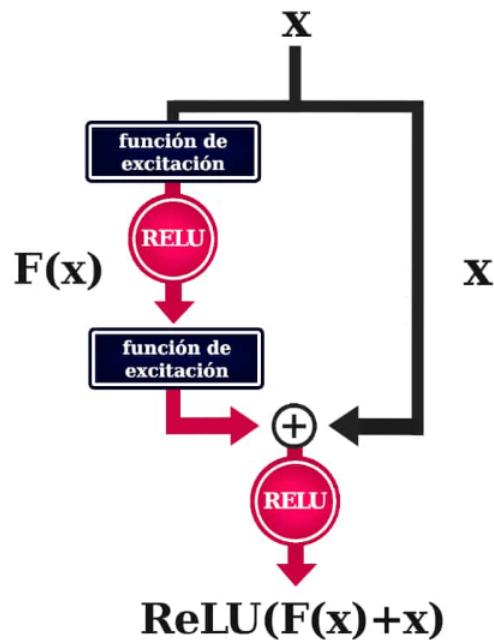


Figura 5-18: Bloque Residual

Como se aprecia en la Figura 5-18, el bloque residual se compone de una ruta residual, (izquierda), y una conexión atajo, (derecha), que la soslaya. 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 $F(x)$ [44].

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 y la capa BN.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			$7 \times 7, 64, \text{stride } 2$		
				$3 \times 3 \text{ max pool, stride } 2$		
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 5-19: Arquitectura ResNet18

Como se puede observar en la Figura 5-19, la primera capa de convolución implementa una plantilla de $7 \times 7 \times 7$, con un tamaño de paso de 2 y un relleno de 3, Seguido a eso, se realiza BN, ReLu y Maxpooling. Estos constituyen la primera parte del modulo de convolucion conv1.

Luego hay cuatro etapas, cada etapa tiene múltiples módulos, cada módulo se llama un bloque de construcción, por ende se tiene que ResNet=[2,2,2,2] se evidencia que hay 8 bloques de construcción, como se presenta en la Figura 5-20 [44].



Figura 5-20: Arquitectura Resnet18

En la FIGURA 5-21, se procede a presentar la precisión obtenida del modelo ResNet-18

durante 50 épocas, sin haber aplicado la optimización bayesiana.

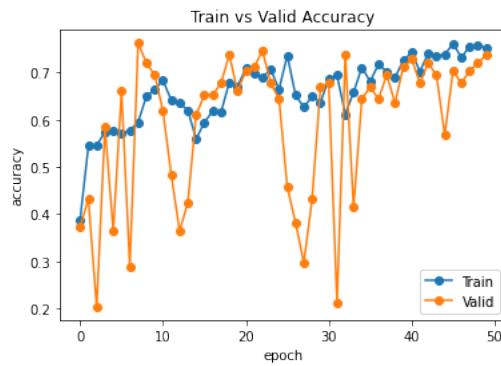


Figura 5-21: Precisión modelo Resnet-18

La información presentada en la Figura 5-22, permite observar que el modelo cuenta con una desviación muy baja entre las predicciones y los valores reales, permitiendo concluir que el modelo cuenta con un bajo porcentaje de *overfitting*.

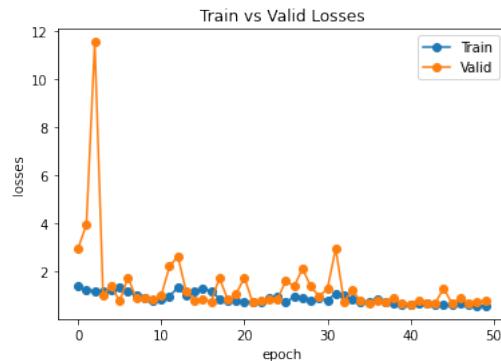


Figura 5-22: Perdidas modelo Resnet-18

La Figura 5-23, presenta las predicciones realizadas por el modelo ResNet-18 con las muestras implementadas durante su validación.

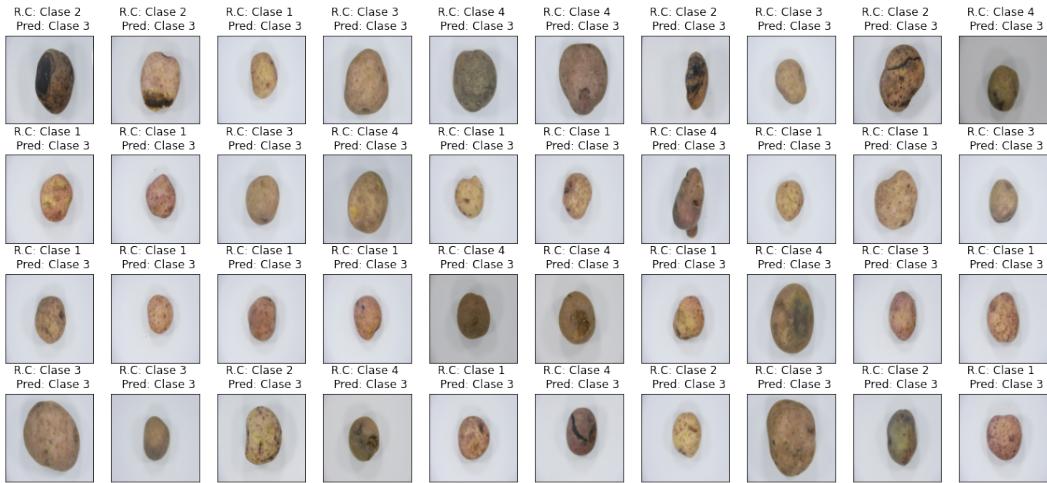


Figura 5-23: Predicción modelo Resnet-18

La matriz de confusión del modelos ResNet-18 es presentada en la Figura 5-24, con el fin de dar a conocer los verdaderos positivos, verdaderos negativos, falsos positivos, y falsos negativos del modelo

		PREDICCIÓN			
		CLASE 1	CLASE 2	CLASE 3	CLASE 4
OBSERVACIÓN	CLASE 1	0	0	29	0
	CLASE 2	0	0	24	0
	CLASE 3	0	0	22	0
	CLASE 4	0	0	43	0

Figura 5-24: Matriz de confusión modelo ResNet-18

Para finalizar, en la Figura 5-25, se presenta la precisión del modelo sin optimización bayesiana por clases.

CLASE 1	0
CLASE 2	0
CLASE 3	100
CLASE 4	0
Overall Accuracy: 18,42105	

Figura 5-25: Precisión por clase modelo ResNet-18

COMPARACIÓN DE ARQUITECTURAS

5.1.5. Optimización Bayesiana

Dos de las estrategias más empleadas para probar diferentes combinaciones de hipérparámetros y evaluarlas mediante métodos de validación, son *grid search* y *random search* [45]. En la primera estrategia, los valores estudiados de cada hiperparámetro son distribuidos de forma uniforme dentro de un rango delimitado por el analista. En la segunda estrategia, los datos son aleatorios dentro de ese rango como se muestra en la Figura 5-26.

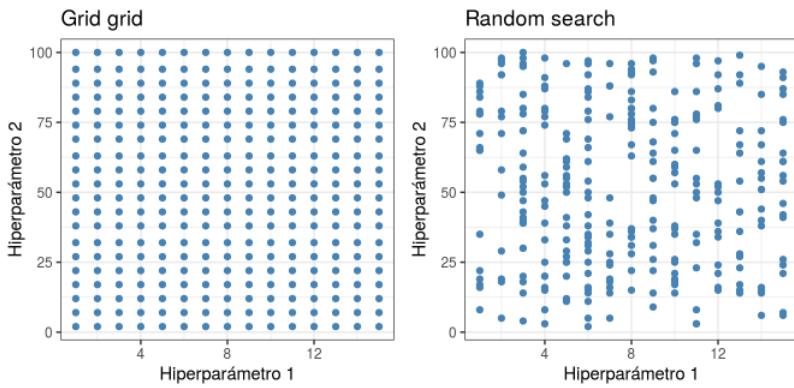


Figura 5-26: Métodos de Optimización De Hiperparámetros

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, la cual consiste en crear un modelo probabilístico en el que el valor de la función objetivo es la métrica de validación del modelo, en este caso, la presición. Con este método, se logra que la búsqueda se vaya redirigiendo en cada iteración hacia las regiones de mayor interés [46]. 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* [47]. 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 probabili-

dad de mejora y el límite superior de confianza. La mejora esperada es una función de adquisición recompensa la evaluación del objetivo f basándose en la mejora esperada en relación con el mejor momento. En la Ecuación 5-7 se define EI .

$$EI(x) = \max(f(x) - f^*, 0) \quad (5-7)$$

El resultado de la parametrización, con la mejora esperada, más alto 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 de acuerdo a los parámetros de optimización elegidos. La Tabla 5-4 muestra los parámetros escogidos para optimizar.

Parámetro	Límite inferior	Límite Superior
lr	$1 * 10^{-6}$	0.4
Momentum	0	1
Stepsize	20	40

Tabla 5-4: 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.

5.2. Clasificación Por Tamaño

La clasificación de los tubérculos de papa de acuerdo a su diámetro, se realiza para categorizar el producto bajo la norma técnica colombiana *NTC 341* establecida por la industria alimentaria, con el fin de garantizar la calidad de los tubérculos de consumo. La norma establece cuatro categorías para la clasificación de los tubérculos mediante la medición de su tamaños, estas categorías se presentan en la Tabla 5-5.

DENOMINACIÓN	DIÁMETRO (mm)
MUY GRANDE	MAYOR DE 90
GRANDE	65-90
MEDIANA	45-64
PEQUEÑA	30-44

Tabla 5-5: Tamaño De tubérculos de Consumo

De acuerdo a la creación del *Dataset* de este documento, las categorias utilizados fueron *Muy Grande, Grande y Mediana*. Para realizar la clasificación por tamaño se utilizó el módulo de visión artifical para *Python Open CV*. Primero se realizó un estudio a un grupo de 5 imágenes por cada uno de los tamaños retirando el fondo, la imagén 5-27, utilizando la herramienta de MATLAB® *Color Thresholder*.

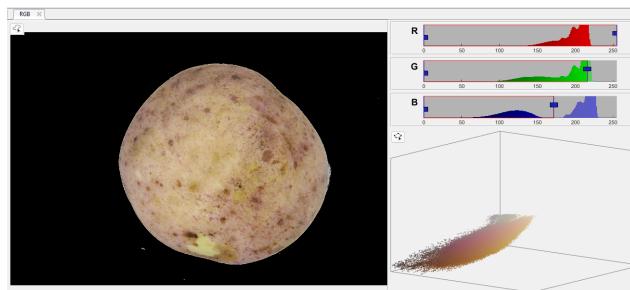


Figura 5-27: Matlab Color Thresholder

La herramienta permite, de forma interactiva, crear una función que permita quitar ciertos colores en el espacio *RGB* de la imagén de muestra, en este caso, se realizó el proceso para retirar el fondo de la imagén 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 5-28.

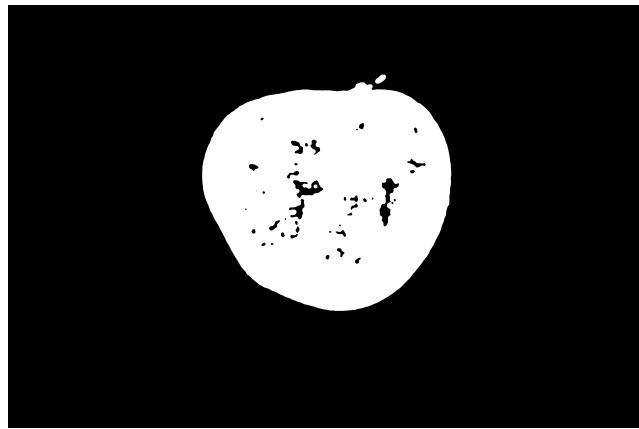


Figura 5-28: Imagen de Muestra Binarizada

Con la imagen aislada de la mayoría de las 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 5-29.



Figura 5-29: Perímetro Aproximado De Las Papas

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. Para clasificar estos 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 5-6 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	6524,936741	0_0_2_2011.jpg	6074,978377
0_1_0_1806.jpg	9523,78	0_1_1_1828.jpg	6802,549007	0_1_2_1506.jpg	5.018
1_0_0_0777.jpg	13.275	1_0_1_0796.jpg	10602,79825	1_0_2_1995.jpg	5090,177294
1_1_0_0825.jpg	27.830	1_1_1_0770.jpg	10285,33525	1_1_2_0857.jpg	5500,756232
0_0_0_1964.jpg	21.607	0_0_1_1240.jpg	6.726	0_0_2_2013.jpg	6046,628085
Promedio	16163,19	Promedio	8188,33	Promedio	5546,12
Variacion STD	7425,235484	Variacion STD	1846,765787	Variacion STD	451,4373313
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 5-6: Perímetro en Tamaño de Píxeles

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

6 Prototipo

En este capítulo se presentan diferentes numerales para detallar y explicar el prototipo implementado para la clasificación de tubérculos de papa. En el numeral 6.1 se dan a conocer las especificaciones de la maquina implementada. seguido en el numeral 6.2 se realiza la descripción de la maquina. En el numeral 6.3 se tiene la descripción del sistema eléctrico y sus componentes, y por ultimo se termina con los numerales 6.4 y 6.5 donde se explica la estructura construida para la clasificación de los tubérculos, y se finaliza con el funcionamiento de la maquina.

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 de banda continua horizontal RF-900 se presentan en la Figura 6-1.

ALIMENTACIÓN	
220 +- 10 V, 50 HZ	
MATERIAL	
Estructura en acero inoxidable y aleación de aluminio/acero	
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)

Figura 6-1: Especificaciones Principales

6.2. Descripción de la maquina

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

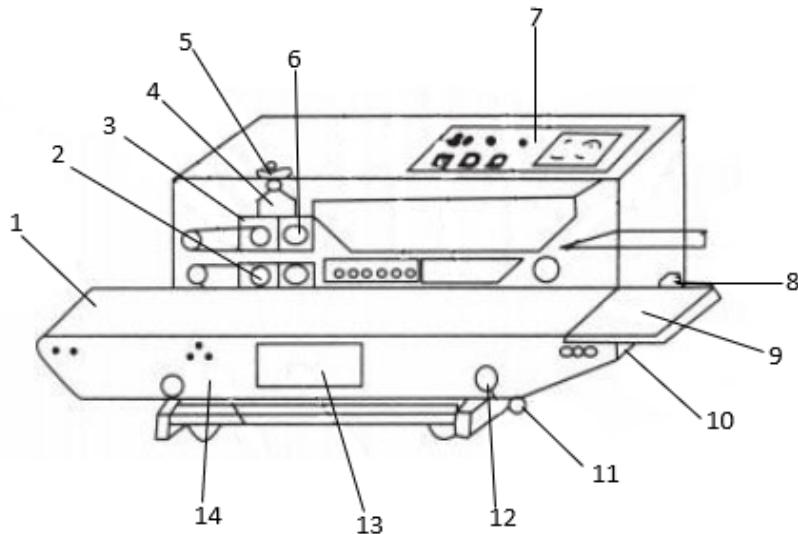


Figura 6-2: Maquina Banda Transportadora

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 6-1: Descripción Maquina Transportadora

6.3. Esquema eléctrico

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

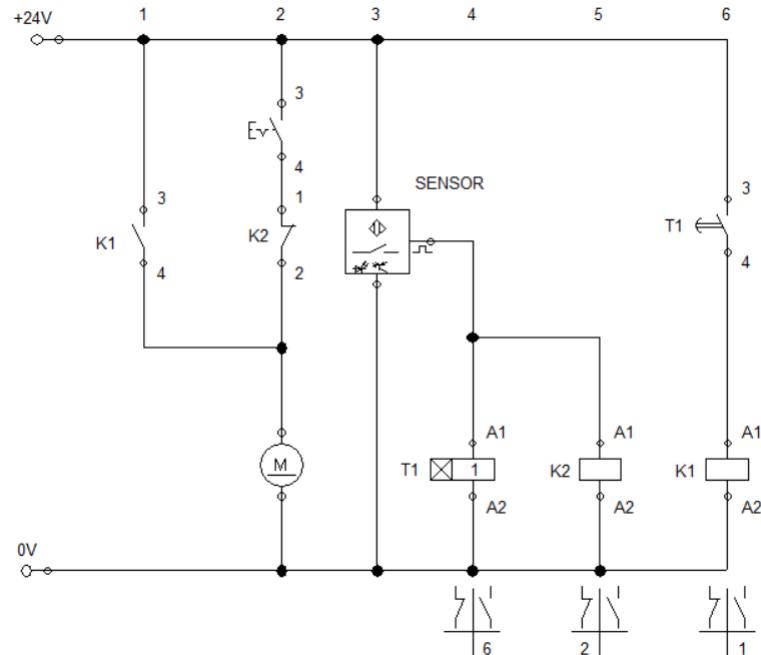


Figura 6-3: Esquema eléctrico

SÍMBOLO	NOMBRE	CANTIDAD
V1	Fuente de Voltaje	1
S1	Pulsador	1
K1	Timer	1
M1	Motor DC	1
B1	Sensor Fotoeléctrico	1
H1	Bombillos	2

Tabla 6-2: 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 lamas frontal y trasera, $16 \times 25(cm)$ lamina superior, $20 \times 16(cm)$ lamas laterales como se muestra en las Figuras.

Como se muestra en las Figuras , las lamas fueron diseñadas con las medidas necesarias que permitan posicionar algunos elementos necesarios para 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 de color para mantener la iluminación fija a una distancia de 30cm 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, esto para dar a conocer que la maquina ya puede entrar en funcionamiento. 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 (Figura 6-4), en donde el ultimo 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 6-5 con el fin de generar el movimiento de la banda.



Figura 6-4: Caja reductora conectada a los engranajes rectos



Figura 6-5: 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 6-6, 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 envíe a la red neuronal para su análisis. Culminado el tiempo establecido en el timer, la banda continuara su movimiento para desplazar y continuar con la clasificación de los tubérculos.



Figura 6-6: Detección del sensor fotoeléctrico

6.5.2. Implementación en Sistema Embebido

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 **6-7**, muestra una de las papas pasando dentro de la estructura hecha para la clasificación.



Figura 6-7: 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 **6-3** 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	Mediana	Clase 4	Mediana
Clase 3	Grande	Clase 3	Grande
Clase 2	Mediana	Clase 2	Mediana
Clase 1	Mediana	Clase 2	Grande
Clase 4	Grande	Clase 3	Grande
Clase 3	Mediana	Clase 3	Mediana
Clase 1	Mediana	Clase 1	Mediana

Tabla 6-3: 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 presición 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 varia 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 estrucutra de análisis es de 7 segundos, por lo tanto, se puede asumir un tiempo promedio de 10 segundos en analisar 1 tubérculo de papa. Este prototipo clasificaría 100 tubérculos de papa cada 15 minutos.

7 Trabajo Futuro

Debido a que es 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 dsitribución 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 di 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 debe 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] “El aporte de la papa a la economía colombiana • periódico el campesino – la voz del campo colombiano.”
- [2] “Sector papero se prepara para aumentar el consumo de papa en colombia — finagro.”
- [3] E. García and F. Flego, “Agricultura de precisión,” *Revista Ciencia y Tecnología*, vol. 1, no. 8, pp. 99–116, 2008.
- [4] A. Krogh, “What are artificial neural networks?,” *Nature biotechnology*, vol. 26, no. 2, pp. 195–197, 2008.
- [5] 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.
- [6] 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.
- [7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.
- [8] Y. Zhu and C. Huang, “An improved median filtering algorithm for image noise reduction,” *Physics Procedia*, vol. 25, pp. 609–616, 2012.
- [9] 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.
- [10] H. Gholamalinezhad and H. Khosravi, “Pooling methods in deep neural networks, a review,” *arXiv preprint arXiv:2009.07485*, 2020.
- [11] 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.

- [12] 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.
- [13] G. I. Viera Maza, “Procesamiento de imágenes usando opencv aplicado en raspberry pi para la clasificación del cacao,” 2017.
- [14] J. Murphy, “An overview of convolutional neural network architectures for deep learning,” *Microway Inc*, pp. 1–22, 2016.
- [15] 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.
- [16] 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.
- [17] I. García and V. Caranqui, “La visión artificial y los campos de aplicación,” *Tierra infinita*, vol. 1, no. 1, pp. 98–108, 2015.
- [18] 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.
- [19] C. Nandi, “Machine vision based automatic fruit grading system using fuzzy algorithm,” 02 2014.
- [20] 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.
- [21] “Soybean plant foliar disease detection using image retrieval approaches,” *Multimedia Tools and Applications*, vol. 76, pp. 26647–26674, 12 2017.
- [22] 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.
- [23] 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.

- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] A. M. RODRIGUEZ, “Sistema automatizado de reconocimiento y manipulación de objetos usando visión por computadora y un brazo industrial,” 2014.
- [31] 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.
- [32] 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.
- [33] V. Artificial, “Aplicación práctica de la visión artificial en el control de procesos industriales,” *Ministerio de Educación y Formación Profesional, Gobierno de España*, 2012.
- [34] “La camara fotografica cuerpo y controles basicos.”

- [35] T. Afzidi, A. Alam, M. N. Khan, and J. Khan, “A multimodal memes classification: A survey and open research issues,” 09 2020.
- [36] “Pytorch documentation — pytorch 1.10.1 documentation.”
- [37] V. Mathivet, *Inteligencia artificial para desarrolladores: conceptos e implementación en C*. Ediciones ENI., 2018.
- [38] 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.
- [39] L. Bottou, “Stochastic gradient descent tricks,” pp. 421–436, 2012.
- [40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [41] L. Moreno-Díaz-Alejo, “Análisis comparativo de arquitecturas de redes neuronales para la clasificación de imágenes,” Master’s thesis, 2020.
- [42] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [43] A. Atienza Arroyo *et al.*, “Detección e identificación automática de actrices y actores mediante el uso de algoritmos de deep learning,” 2019.
- [44] 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.
- [45] P. Liashchynskyi and P. Liashchynskyi, “Grid search, random search, genetic algorithm: A big comparison for nas,” *arXiv preprint arXiv:1912.06059*, 2019.
- [46] P. I. Frazier, “A tutorial on bayesian optimization,” *arXiv preprint arXiv:1807.02811*, 2018.
- [47] 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.