UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

# Aplicación de Redes Neuronales en el proceso de inspección de calidad del aguacate Hass para exportación.

**Camilo Andrés Zuleta Varón**

**Merly Karina Triana Valderrama**

Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería
Proyecto Curricular Ingeniería Industrial
Bogotá D.C.
2022

# Aplicación de Redes Neuronales en el proceso de inspección de calidad del aguacate Hass para exportación.

**Camilo Andrés Zuleta Varón**

**Merly Karina Triana Valderrama**

Trabajo de grado presentado como requisito para optar al título de:
Ingeniero Industrial

Director:
Ingeniero Carlos Augusto Toledo Bueno

Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería
Proyecto Curricular Ingeniería Industrial
Bogotá D.C.
2022

# Tabla de contenido

Lista de figuras ..... 5
Lista de tablas ..... 6
Introducción ..... 7
1. Problema de investigación ..... 8
1.1 Planteamiento del problema ..... 8
1.2 Formulación del problema ..... 8
2. Justificación ..... 9
3. Antecedentes ..... 10
4. Objetivos ..... 12
4.1 Objetivo General ..... 12
4.2 Objetivos Específicos ..... 12
5. Marco Teórico ..... 13
5.1 Caracterización Aguacate Hass para exportación ..... 13
5.1.1 Proceso de inspección de unidades para exportación ..... 13
5.1.2 Criterios de calidad de exportación para aguacate Hass ..... 14
5.2 Inteligencia Artificial ..... 15
5.3 Machine Learning ..... 15
5.4 Algoritmos de aprendizaje automático ..... 15
5.5 Redes Neuronales Artificiales (ANN/RNA) ..... 16
5.5.1 Redes Neuronales simples ..... 16
5.5.2 Redes Neuronales convolucionales ..... 18
5.2.3 Funciones en Redes Neuronales Convolucionales ..... 20
5.6 Aprendizaje profundo CNN ..... 23
5.6.1 Redes Neuronales convolucionales profundas ..... 23
5.7 Algoritmos de Optimización ..... 24
5.7.1 RMSprop ..... 24
5.8 Framework ..... 25
5.8.1 Tensorflow ..... 25
5.8.2 Keras ..... 25
5.9 Métricas ..... 25
5.9.1 Accuracy ..... 25
5.9.2 Loss ..... 26

5.10 Aumento de Datos ....26
6. Metodología ....27
7. Desarrollo ....29
7.1 Criterios de clasificación de aguacate Hass ....29
7.2 Dataset ....29
7.3 Estructura de Redes Neuronales ....31
7.3.1 Estructura Red Neuronal Convolucional Simple ....32
7.3.2 Estructura Red Neuronal Convolucional Profunda - ResNet 50 ....33
7.3.3 Estructura Red Neuronal Convolucional Profunda - Inception Learning Google ....37
8. Resultados y Análisis ....39
8.2 Resultados CNN simple ....39
8.3 Resultados ResNet50 ....41
8.4 Resultados Inception Network ....43
8.5 Análisis general de los resultados ....45
9. Conclusiones ....46
10. Referencias ....48
11. Anexos ....53

# Lista de figuras

**Figura 1**. Unidad de aguacate Hass apta para exportación fase II.....14
**Figura 2**.Descarte de fruta para la exportadora Hass Diamond. .....15
**Figura 3**. Estructura general de una Red Neuronal artificial.....17
**Figura 4**.Estructura general de una Red Neuronal artificial Convolucional .....18
**Figura 5.** Representación gráfica de una operación de Kernel. .....19
**Figura 6.** Representación gráfica del Max Pooling y Average Pooling. .....20
**Figura 7.** Función RELU .....22
**Figura 8**. ResNet .....23
**Figura 9.** Ciclo de vida de la aplicación .....29
**Figura 10**. Banco de fotos de aguacate Hass- Conformes o aptas .....30
**Figura 11**. Banco de fotos de aguacate Hass-No conformes.....30
**Figura 12**. Número total de parámetros utilizados en el entrenamiento para la red neuronal CNN no profunda.....32
**Figura 13**. Primeras 10 capas de entrenamiento de la red neuronal CNN no profunda con algunos hiper-parámetros utilizados. .....33
**Figura 14.** Número total de parámetros utilizados en el entrenamiento para la red neuronal ResNet50. .....35
**Figura 15**. Bloque de convolución de la red neuronal ResNet50 con algunos hiper parámetros utilizados. .....36
**Figura 16**. Ejemplo de bloque de incepción utilizado en la implementación.....37
***Figura 17.*** *Dimensiones del Kernel por bloque* .....38
***Figura 18.*** *Número total de parámetros utilizados en el entrenamiento para la red neuronal Inception (Google Net).* .....39
***Figura 19.*** *Resultados de la precisión de la CNN no profunda para el conjunto de datos de entrenamiento y validación* .....40
**Figura 20**. Resultados de la pérdida (loss) de la CNN no profunda para el conjunto de datos de entrenamiento y validación.....41
**Figura 21**. Resultados de la precisión de la ResNet50 para el conjunto de datos de entrenamiento y validación. .....42
**Figura 22**. Resultados de la pérdida (loss) de la ResNet50 para el conjunto de datos de entrenamiento y validación.....43
**Figura 23**. Resultados de la precisión de la Inception network para el conjunto de datos de entrenamiento y validación.....44
**Figura 24.** Resultados de la pérdida (loss) de la Inception network para el conjunto de datos de entrenamiento y validación.....45

# Lista de tablas

**Tabla 1.** Fase objeto de la presente investigación. .....13
**Tabla 2.** Criterios de evaluación de unidades. .....27
**Tabla 3.** Fase objeto de la presente investigación. .....27
**Tabla 4.** Estructura de la Red. .....28
**Tabla 5**.Testeo de la Red. .....28
**Tabla 6.** Resultados de la CNN no profunda con los dataset de entrenamiento, validación y prueba .....40
**Tabla 7.** Resultados de la ResNet50 con los dataset de entrenamiento, validación y prueba...41
**Tabla 8**. Resultados de la Inception network con los dataset de entrenamiento, validación y prueba .....43

# Introducción

El aguacate Hass es considerada en los últimos años, una de las frutas tropicales con mayor prospectiva de crecimiento en exportación a nivel mundial gracias a sus múltiples beneficios y usos. El éxito del intercambio comercial depende en gran medida del cumplimiento de estándares físicos de calidad, los cuales son verificados en planta mediante inspección visual por medio de operadores humanos los cuales, uno a uno identifica el cumplimiento de conformidades basándose en su experticia y conocimiento. Sin embargo, al ser el proceso dependiente de operadores humanos, es susceptible a mayor gasto de recurso producto de omisiones en la verificación de unidades, mayor tiempo de clasificación y en caso tal de incurrir en incumplimientos en la calidad de la carga, la suspensión del intercambio comercial.

Es por ello, que en la presente investigación se evalúa la aplicación y uso de tres (3) redes neuronales convolucionales de aprendizaje supervisado (CNN no profunda, Inception Learning de Google y la ResNet 50) como mecanismos de inspección y clasificación por computadora. Esto dado que hoy día, la inteligencia Artificial, especialmente las Redes Neuronales son un método útil y práctico para automatizar o mejorar tareas que antes, por medio de tecnologías convencionales no era posible y ahora sí, gracias a la analogía del aprendizaje humano.

En el enfoque propuesto, en cada una de las arquitecturas se utiliza la función de activación ReLU, Batch Normalization para reducir el sesgo y variabilidad de la información. Así mismo se utiliza Glorot como esquema de inicialización y la precisión como métrica de desempeño de las redes.

En cuanto al banco de datos, se realiza una muestra fotográfica de 320 imágenes de unidades aptas y 325 de unidades no aptas para exportación, las cuales con el fin de robustecer el volumen de los datos fueron sometidas a Data Augmentation, mediante Python y la conocida api Keras, lográndose un banco total de 1.177 fotos de unidades aptas y 1.273 fotos de unidades no aptas. Así mismo para las fases de entrenamiento, validación y test, se utilizó la técnica 60-40.

Los resultados demuestran que el modelo propuesto de ResNet50 logra con un 95% de precisión la clasificación de las unidades. Valor superior con respecto a la CNN simple y la Inception Learning de Google.

# 1. Problema de investigación

## 1.1 Planteamiento del problema

Durante la última década, el consumo de aguacate se ha incrementado cerca de tres (3) puntos porcentuales en promedio cada año en los países de América del Norte y de Europa, así como en China y el Sureste Asiático (Bolívar & Castro, 2019). Siendo países latinoamericanos los mayores productores (Arias et al., 2018) (Ramírez-Gil et al., 2020) y encargados de satisfacer la demanda global (Ministerio de Agricultura y Desarrollo Rural, 2020).

El éxito del intercambio comercial se da en el marco del cumplimiento de normas fitosanitarias que garanticen la calidad del producto a exportar. Por lo cual los procesos de inspección de calidad de aguacate Hass (Aguacate mayormente comercializado en Colombia y el mundo ) son altamente rigurosos, con el fin de garantizar que las unidades sean aptas para exportación, siendo la apariencia física de la fruta uno de los criterios más importantes; el aguacate para exportación debe cumplir con los estándares establecidos por: el cliente, la Organización Mundial de la Salud- OMS y la Organización de las Naciones Unidas para la Alimentación y la Agricultura- FAO. Entre los cuales se encuentra que las unidades estén enteras, limpias, sanas, sin podredumbre, no presenten manchas, huecos, cambios de color (Deben poseer el color verde característico del aguacate sin madurar) y pedúnculo cortado idóneamente (OMS & FAO, 2019)(Ramírez-Gil et al., 2019).

Dichas características son identificadas normalmente por el operario mediante procesos de inspección visual no destructivos en planta; el operario debe recurrir a su experticia y agudeza visual para identificar el cumplimiento de los requisitos de calidad en cada una de las unidades a exportar. Por tanto, al ser el proceso de inspección de calidad (Identificación y clasificación de unidades aptas y no aptas) dependiente de operadores humanos da lugar a la existencia de inexactitudes en cuanto a la identificación del cumplimiento de los requerimientos mínimos de una unidad como producto apto para exportación; los operadores están sujetos a cometer errores u omisiones de verificación, producto de evaluaciones subjetivas, fatiga visual, trabajo rutinario, enfermedades y desórdenes ópticos, factores motivacionales y personales entre otros. (K. Rodriguez & Giraldo, 2017)(Vega, 2011). Siendo esta una de las mayores falencias en el proceso de inspección de calidad del aguacate Hass ya que pueden implicar pérdidas de tiempo, económicas y contractuales. Puesto que el cumplimiento de los estándares de calidad establecidos para el intercambio comercial de la fruta, garantizan el éxito del negocio. Es por ello que es indispensable que el proceso sea lo más eficiente y eficaz posible.

## 1.2 Formulación del problema

**Pregunta Rectora**

¿Son las redes neuronales artificiales (profundas y no profundas) en el proceso de inspección de calidad de Aguacate Hass para exportación un mecanismo útil para la clasificación de unidades aptas y no aptas?

# 2. Justificación

La Inteligencia Artificial (IA) hoy día representa una disrupción tecnológica sin precedentes que está cambiando vertiginosamente la cotidianidad. Está produciendo cambios en muchos ámbitos, entre ellos, la industria, debido a que las máquinas cada vez son "más inteligentes" y autónomas para realizar actividades que antes solo podía ejecutar un operador humano debido al alto nivel de complejidad de las tareas. Es ahí donde las redes Neuronales (Uno de los exponentes de la IA) juega un papel importante, ya que gracias a que tienen la capacidad de aprender automáticamente (Machine Learning) pueden realizar tareas de manera autónoma que hasta el día de hoy no había sido posible automatizar por medio de las tecnologías convencionales.

La razón por la cual las Redes Neuronales han sido ampliamente atractivas para la automatización de procesos y en general en el campo de la ingeniería, es que a partir del año 2012 con el descubrimiento de la Convolución se mejora sustancialmente el comportamiento de estas y año a año el rendimiento de la red ha disminuido considerablemente el error, hasta el punto de que hoy día supera las habilidades humanas en la ejecución de una determinada actividad, salvo que no se haya escogido adecuadamente la estructura de la red y los hiper parámetros. (Russakovsky et al., 2015).

Por tanto, al ser el proceso de inspección de calidad de aguacate Hass un proceso clave para el éxito del intercambio comercial (En la industria del aguacate Hass para exportación la identificación correcta y oportuna de unidades aptas, garantiza no solo el cumplimiento de estándares suficientes para exportación sino la expansión comercial a nivel mundial de la empresa ya que a mayor confiabilidad mayor crecimiento económico, sobre todo en un mercado altamente demandado a nivel mundial), se hace necesario que este sea lo más eficiente posible. Es por ello, que se plantea la posibilidad de evaluar el uso y pertinencia de Redes Neuronales en tareas de identificación y clasificación de unidades aptas para exportación, que hasta el momento se efectúan en planta únicamente de manera manual; el operario uno a uno valiéndose de su experticia identifica las variables que garantizan el éxito de la carga. Implicando no solo posibles omisiones en la clasificación sino el uso de un mayor volumen de recursos que por medio de la automatización se pueden disminuir, como se ha visto a través de la historia en los diferentes procesos industriales.

De esta manera la presente investigación pretende servir como marco de referencia y soporte teórico-practico sobre el uso e implementación de Redes Neuronales Artificiales Convolucionales de aprendizaje supervisado en procesos de inspección de calidad en mercados emergentes y de constante expansión como lo es la industria de Aguacate Hass para exportación, en el cual las normas fitosanitarias son de estricto cumplimiento, garantizando el éxito de la relación comercial. Entonces, por medio de una aplicación práctica se busca evaluar el posible uso de las Redes Neuronales para identificar conformidades en unidades de aguacate. Así mismo, es importante precisar dos cosas: Primero, que con la presente investigación no se pretende realizar un comparativo entre la eficiencia del humano respecto a la Red sino a la pertinencia de esta en el proceso y así contribuir a la masa de conocimiento que hoy día gira en torno a la automatización de procesos con base a la estructura misma del aprendizaje humano, y Segundo, que si bien es cierto las redes neuronales pueden ser útiles para identificar el cumplimiento de requisitos en cuanto a calibre, forma y peso, en la presente investigación, el interés principal gira en torno al cumplimiento de características visuales; color, completitud, apariencia, etc.

# 3. Antecedentes

Si bien es cierto que control de calidad hace hincapié en la prevención como factor clave para controlar errores o defectos (Cubillos & Rozo, 2009), en las industrias exportadoras de alimentos especialmente la del aguacate, este no es un factor suficiente y por ello, se hace necesario recurrir a la esencia del control; la inspección (técnica visual no destructiva de verificación) para garantizar el cumplimiento de estándares. Sin embargo, al ser el proceso de inspección y clasificación de aguacate para exportación (y en muchas más industrias) sujeto y dependiente de operadores humanos este está propenso a evaluaciones erróneas e ineficientes que afectan el desempeño del proceso. Es por ello que, a través del tiempo, este y muchos otros procesos de inspección y clasificación se han intentado mejorar, buscando automatizar el proceso de inspección y más aún automatizar el proceso de clasificación de unidades aptas para exportación ya que la integración de un sistema inteligente en el proceso de inspección de calidad conlleva a una mayor eficiencia y a un beneficio bidireccional; para el exportador y para el importador de la fruta. Consiguiendo “niveles óptimos de calidad, ahorro en el tiempo de clasificación, niveles de calidad mejorados, producción flexible y flujo de datos mejorado” (Cruz De La Cruz & Vera, 2020)

Realizando una revisión en la literatura para (Jaramillo et al., 2014) las características de inspección visual propias de la clasificación de productos de calidad, en el campo de la ingeniería, en 2014 se estaba aplicando diversas técnicas de visión artificial para poder automatizar el proceso y mitigar las falencias producto del yerro humano.

Las técnicas de visión artificial han tenido una evolución dramática en la última década. Hace menos de 10 años, hablar de visión artificial era todo un dolor de cabeza, se debía hacer un proceso de depuración de la información (imágenes), en las cuales se debían determinar mediciones de patrones y documentar toda esta información en grandes bases de datos que luego alimentarían algoritmos estadísticos como: regresiones lineales, regresiones logísticas, arboles de decisión o super vector machines (Russakovsky et al., 2015). Todo este esfuerzo manual y algorítmico en ocasiones no traían los resultados esperados y se manejaban márgenes de error por encima del 25% algo inaceptable si se piensa utilizar en procesos tales como los de inspección de calidad o algún otro campo de aplicación.

Por ejemplo, en (Escobar & Roa, 2016) se clasifica la granadilla según el grado de maduración por medio de visión artificial mediante el procesamiento de imágenes (técnicas RGB) con una precisión del 92.6%. Así mismo, (Cubero, 2012) en su tesis doctoral diseñó e implemento estructuras basadas en visión artificial para la inspección de frutas en campo, sin hacer hincapié en el aguacate. En (Orte et al., 2019) se plantean el uso de un algoritmo basado en visión por computadora para una identificar cumplimiento de requisitos de calidad con una eficiencia del 76,19% .

Sin embargo, con la aparición de las redes neuronales (Aprendizaje profundo) todas estas complicaciones y esfuerzo se redujeron notablemente. Además, de reducir el error de las predicciones a niveles que hoy en día pueden ser menores que el error del ser humano.

Esto llevó a que dentro de las técnicas de visión artificial se integraron las redes neuronales como algoritmos de aprendizaje dado que estas son un sistema que se adapta a las exigencias del entorno y pueden combinar técnicas que procesan la información de manera paralela. Así mismo

conforma todo un modelo computacional que, por sus componentes, potencian la interconexión y facilitan la adaptación (Figueredo & Ballesteros, 2016). En (Sabzi et al., 2018) se aplicaron y compararon tres clasificadores diferentes: red neuronal artificial híbrida – colonia de abejas artificial (ANN-ABC); red neuronal artificial híbrida - búsqueda de armonía (ANN-HS); y k-vecinos más cercanos (kNN). Los resultados experimentales muestran que los enfoques híbridos superan los resultados de kNN. La tasa de clasificación correcta promedio de ANN-HS fue del 94,28 %, mientras que ANN-ABS logró una precisión del 96,70 % con los datos disponibles, en contraste con la precisión inicial del 70,9 % de kNN. Por su parte, en (Azarmdel et al., 2020) se aplicaron redes neuronales artificiales (ANN) y Support Vector Machine (SVM) para clasificar la fruta de morera. La clasificación ANN con el método de extracción de características del subconjunto CFS dio como resultado una precisión del 100 %, 100 % y 99,1 % en entrenamiento, validación y prueba, respectivamente.

Según la literatura, existen estudios de clasificación de diversos frutos. En (Kumar et al., 2017) se plantea el uso de Redes Neuronales Artificiales para clasificar la granada para exportación, mediante criterios binarios. Al igual, (Vega, 2011) propuso una aplicación de red neuronal para el reconocimiento de la calidad morfológica de mangos exportables; mediante la cual, de acuerdo a la estructura del mango, esta podía seleccionar y clasificar los magos en tres tipos. Logrando un porcentaje de error del 2.33%, valor considerablemente bajo con respecto al generado por el operario, 17.5%. En (Fiona et al., 2019) se realiza una clasificación de cítricos por medio de redes neuronales simples con un porcentaje de precisión del 90%.

En cuanto a manejo de data, (M. Rodriguez et al., 2021) utilizan Redes Neuronales alimentadas y entrenadas por medio de un banco de datos sometido a Data Augmentation para determinar el grado de maduración de fresas y manzanas con un nivel de precisión del 96.34%. En (Hakami & Arif, 2019) clasifican dátiles por medio de procesamiento de imagines (566 imágenes). El método logró una precisión de clasificación del 99 % con 133 características y 50 grupos.

En (Ashok & Vinod, 2016) se realiza clasificación de mango por medio de Redes Neuronales y procesamiento de imágenes con una precisión de rendimiento del 90.09%, 90.26% y 90.26% para funciones de activación lineales, logísticas y softmax.

En cuanto al proceso del aguacate, (Cruz De La Cruz & Vera, 2020) por medio de LabView y redes neuronales convolucionales clasifican aguacate Hass, en (Guerrero & Benavides, 2014) el proceso lo realizan por medio de procesamiento de imágenes, logrando una precisión del 82.22%. (Moreno Hincapié et al., 2019) se realiza la clasificación de aguacates por medio de su color mediante sensores Kinect.

En general existen varios estudios referentes a clasificación que integran técnicas como arboles de decisión, Support vector machine algorithms y una de las más relevantes; las Redes Neuronales Convolucionales. Lo que denota que la integración de redes neuronales en los procesos de inspección visual por computadora son técnicas en crecimiento. Siendo estas aplicaciones un medio posible para lograr una mayor eficiencia en el proceso, disminuyendo la posibilidad de errores. Teniendo en cuenta que hoy día el mercado mundial de Aguacate Hass está en constante crecimiento.

# 4. Objetivos

## 4.1 Objetivo General

Evaluar la pertinencia y desempeño de redes neuronales en el proceso de inspección de calidad de aguacate Hass para exportación mediante el uso de procesamiento de imágenes como mecanismo de clasificación alternativo a los métodos tradicionales

## 4.2 Objetivos Específicos

- Determinar las características que rigen el proceso de inspección de calidad del aguacate Hass para exportación, mediante una revisión de la literatura y los procesos establecidos en la industria de exportación.

- Generar un banco de datos mediante la captura, procesamiento y aumento de imágenes (Data Augmentation) para el entrenamiento, validación y testeo de las redes neuronales.

- Entrenar y validar las estructuras CNN simple, Inception Net y ResNet 50 mediante la Api de Keras y Tensorflow.

- Validar el desempeño de las redes implementadas con respecto a un nivel de precisión del 90%.

# 5. Marco Teórico

Con el fin de orientar el camino de investigación de la presente aplicación de redes neuronales en procesos de inspección de calidad se hace necesario realizar una conceptualización de los principales tópicos para tener en cuenta durante el desarrollo investigación.

## 5.1 Caracterización Aguacate Hass para exportación

El aguacate Hass o Palta Hass es una fruta altamente perecedera con una alta tasa metabólica que resulta en una vida útil de aproximadamente 3 a 4 semanas cuando se almacena en el nivel óptimo de temperatura y humedad relativa.(González Cuello et al., 2017). Para exportación, uno de los criterios clave, para garantizar que la carga llegue al destino en condiciones óptimas, es la distancia a recorrer, puesto que, dependiendo de esta, el medio de transporte puede variar entre terrestre, marítimo e incluso aéreo. El transporte marítimo, por ejemplo, al implicar grandes distancias, se hace necesario que la carga sea traslada bajo entornos controlados de humedad y temperatura. Aun así, esto, no es suficiente para garantizar el éxito de la inversión. Se hace necesario que inicialmente, en planta se realicen las validaciones correspondientes de calidad de la fruta, con el fin de asegurar una carga en buen estado, capaz de llegar al destino, cumpliendo los estándares exigidos en el intercambio comercial; el proceso de exportación de aguacate está enmarcado en el cumplimiento de normas de calidad fitosanitarias, el aguacate para exportación en cuanto a la apariencia física debe cumplir con los estándares establecidos por la Organización Mundial de la Salud- OMS y la Organización de las Naciones Unidas para la Alimentación y la Agricultura- FAO, (OMS & FAO, 2019)(Ramírez-Gil et al., 2019). Normas que son de estricto cumplimiento y por ello el aguacate para exportación es sometido a un proceso de calidad robusto, en planta, en el que se garantice la calidad de la fruta.

### 5.1.1 Proceso de inspección de unidades para exportación

Una vez las unidades de aguacate llegan a las plantas transformadoras y comercializadoras de aguacate Hass, son sometidas al primer proceso de selección mecánico (El cual no es objeto de la presente investigación) para determinar calibre y peso. Aquellas unidades que cuenten con el tamaño establecido para exportación continúan el proceso de clasificación y las que no, pasan a consumo nacional. (Salazar, 2020). El proceso por fases se puede detallar en la tabla 1.

***Tabla 1.*** *Fase objeto de la presente investigación.*

| PROCESO DE INSPECCIÓN DE UNIDADES EN PLATA | | | |
|---|---|---|---|
| **FASE** | **EVALUACIÓN** | **MECANISMO** | **ALCANCE*** |
| I | Calibre y peso | Mecánico | No |
| II | Color, textura, completitud y homogeneidad | Manual | Si |
| III | Madurez y dureza | Manual | No |
| IV | Plagas internas | Manual | No |

*Nota*: En esta tabla se sintetiza el alcance de la presente investigación de acuerdo a cada una de las fases del proceso de inspección de aguacate Hass en las plantas procesadoras e importadoras.

Terminada la fase I, las unidades de aguacate una a una, deben ser inspeccionadas por parte de los operadores humanos para identificar cumplimiento de estándares; color, textura, completitud y homogeneidad. En la figura 1 se puede detallar una unidad conforme para exportación.

***Figura 1**. Unidad de aguacate Hass apta para exportación fase II*



*Nota*: Ejemplo de unidades aptas para exportación. La fruta no presenta daños mecánicos, pedúnculo visible y color característico de aguacate sin madurar.

Posterior a ello se someten a un tercer filtro, donde se evalúa, también, de manera manual el grado de madurez y dureza de la fruta (Proceso clasificatorio que no es objeto de la presente investigación ya que se requiere contacto físico y no visual de la unidad). Y, por último, una parte de la carga es cortada, para detectar que internamente no tengan plagas, tales como Heilipus lauri, H. trifasciatus y Stenoma catenifer.(Parra & Serrano, 2017).

### 5.1.2 Criterios de calidad de exportación para aguacate Hass

Los operadores deben garantizar que los aguacates a exportar en cuanto a apariencia física cuenten con las siguientes condiciones las cuales a modo de detalle se pueden observar en la figura 2:

- ✓ Posean color verde característico del aguacate sin madurar
- ✓ Posean pedúnculo
- ✓ No presenten manchas o grietas
- ✓ No presenten deformaciones notorias
- ✓ Esten enteros y exentos de daños mecánicos
- ✓ Estar exentos de plagas
- ✓ Estar exentos de daños producto de cambios de temperatura

***Figura 2**.Descarte de fruta para la exportadora Hass Diamond.*




*Nota*: Ejemplos de unidades no aptas para exportación según la comercializadora Hass Diamond. Tomado de comercializadora Hass Diamond.

## 5.2 Inteligencia Artificial

La inteligencia artificial en si, es comprendida como el diseño, implementación y uso de programas, máquinas y sistemas que exhiben inteligencia humana, siendo sus actividades más importantes la representación del conocimiento, el razonamiento y el aprendizaje (Whitson, 2020). En general, es la capacidad que tienen los artilugios artificiales de realizar tareas propias de la inteligencia humana entre ellas la capacidad de aprender. La inteligencia artificial (IA) es de suma importancia debido a sus vastas aplicaciones en todas las áreas del conocimiento. Existen integraciones de IA en diagnósticos médicos, reconocimiento facial, robótica, internet, minería de datos, procesos industriales y demás. (Górriz et al., 2020)

## 5.3 Machine Learning

El Machine Learnig, o aprendizaje automático/de máquina, es una inteligencia artificial que le permite a un sistema aprender de datos (Lu & Li, 2020). Fundamentalmente, implica la construcción de modelos matemáticos para ayudar a comprender los datos, integrando a dichos modelos parámetros ajustables que le permiten al sistema aprender; existe modificación de comportamientos con base al procesamiento de datos. El 'machine learning' se concibe como un maestro del reconocimiento de patrones, y es capaz de convertir una muestra de datos en un programa informático capaz de extraer inferencias de nuevos conjuntos de datos para los que no ha sido entrenado previamente" (Espinoza & BBVA, 2020).Dicho aprendizaje puede ser supervisado o No supervisado.

## 5.4 Algoritmos de aprendizaje automático

Existen 2 tipos principales de modelos de aprendizaje computacional, modelos de aprendizaje supervisado y modelos de aprendizaje no supervisados.

En los algoritmos No supervisados, no hay una distinción entre variables dependientes y no dependientes, en este caso se pretende encontrar la estructura subyacente que explique la

estructura de los datos.(Fabio González, 2015) Es decir, que no utilizan datos etiquetados sino similitudes. Por lo cual los algoritmos no están programados para detectar un tipo especifico de dato, sino que buscan asociaciones por similitud.

Por su parte, los algoritmos de aprendizaje supervisado, utiliza datos etiquetados. Según (Fabio González, 2015) “Se busca inducir modelos capaces de predecir el valor de ciertas variables dependientes a partir de variables independientes”.

El aprendizaje supervisado se divide en tres etapas: entrenamiento, validación y pruebas (Díaz et al., 2021)

Por medio del entrenamiento se optimizan los parámetros del modelo con base a errores previos; el error se obtiene comparando el modelo predicción con la etiqueta real asociada.

En la validación, se utiliza el modelo obtenido después del entrenamiento para predecir las etiquetas de un segundo conjunto de imágenes. En esta fase se realiza ajuste de parámetros y características.

Por último, en la etapa de pruebas, se evalúa el rendimiento del modelo final utilizando un conjunto de imágenes nuevas, nunca antes vistas. Los resultados de este análisis con este conjunto de imágenes determinan la precisión del modelo. (Díaz et al., 2021)

## 5.5 Redes Neuronales Artificiales (ANN/RNA)

Las redes neuronales son algoritmos bio-inspirados (Francisco González & Romero, 2018) (Sordo et al., 2008)(Sordo et al., 2008) que mediante el estudio e intento de réplica de los entornos vivos puede conllevar a encontrar soluciones o mejoras a situaciones en particular, mediante modelos matemáticos de procesamiento de datos (Peña & Minim, 2001) . El algoritmo en general consiste en un conjunto de capas interconectadas que comparten información de manera bidireccional (forward and backward propagation), lo que permite que la red aprenda con cada entrenamiento o iteración.(Ramos & García, 2019)(J & F, 2015).Cada una de estas capas está conformada por pequeñas unidades llamadas neuronas, interconectadas mediante fuerzas de conexión variables o ponderaciones- también llamados pesos, las cuales al inicio son aleatorias y con el paso a través de las capas, pueden incrementar o inhibir el estado de activación de las neuronas, en aras de reducir la función de pérdida (normalmente mediante propagación hacia atrás).

Las redes neuronales artificiales se pueden clasificar de acuerdo a su estructura (monocapa, multicapa, convolucionales, concurrentes, radiales, etc) y con base a su modelo de aprendizaje, el cual puede ser supervisado o no supervisado.

### 5.5.1 Redes Neuronales simples

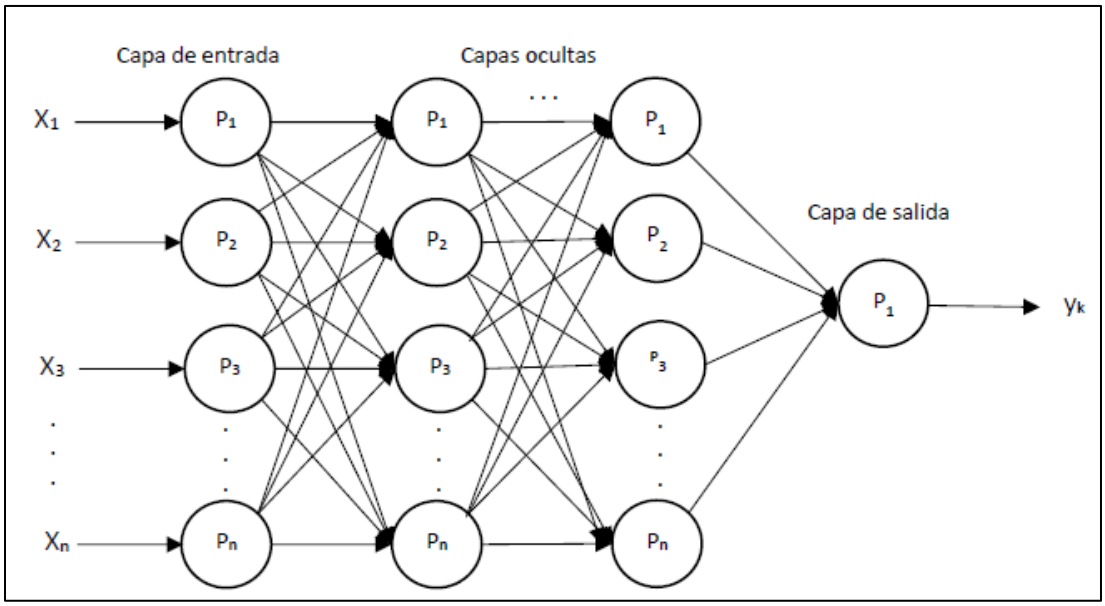
Las redes Neuronales simples, son concebidas como los modelos más básicos de redes neuronales. Generalmente, se componen de una serie de neuronas dispuestas en diferentes capas; una capa de entrada, una capa de salida y una o más capas ocultas.

Warren McCulloch y Walter Pitts propusieron un modelo muy simple de neurona biológica, que luego se conoció como neurona artificial monocapa (Géron, 2019) o también llamado perceptrón

simple, la cual está compuesta por una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se dan lugar a funciones de activación para definir el valor de salida.

Por su parte las redes multicapa (perceptrón multicapa-MLP), a diferencia del perceptrón simple contienen capas intermedias (ocultas) y estas pueden estar total o parcialmente conectadas entre sí, ver figura 3.

***Figura 3**. Estructura general de una Red Neuronal artificial.*




*Nota*: El gráfico representa una estructura de Red Neuronal Artificial simple. Tomado de Galindo, E. A., Perdomo, J. A., & Figueroa-García, J. C. (2020). Estudio comparativo entre máquinas de soporte vectorial multiclase, redes neuronales artificiales y sistema de inferencia neuro-difuso auto organizado para problemas de clasificación. Información Tecnológica, 31(1), 273–286. https://doi.org/10.4067/s0718-07642020000100273

Las neuronas de entrada reciben y procesan las señales de entrada y envían una señal de salida a otras neuronas de la red. Cada neurona se puede conectar a las otras neuronas y tiene una función de activación y una función de umbral, que pueden ser funciones continuas, lineales o no lineales. (Dolling & Varas, 2002).

Matemáticamente (Galindo et al., 2020) indica que “una neurona artificial *k* mapea *n* señales de entradas *x* a un único valor de salida $y_k$. primero calcula la suma ponderada entre las entradas y los costos de transferir información entre una neurona y otra ($w_k$), y agrega un término umbral, $b_k$. entonces una operación matemática es desarrollada sobre la suma ponderada, la función de activación Ω”

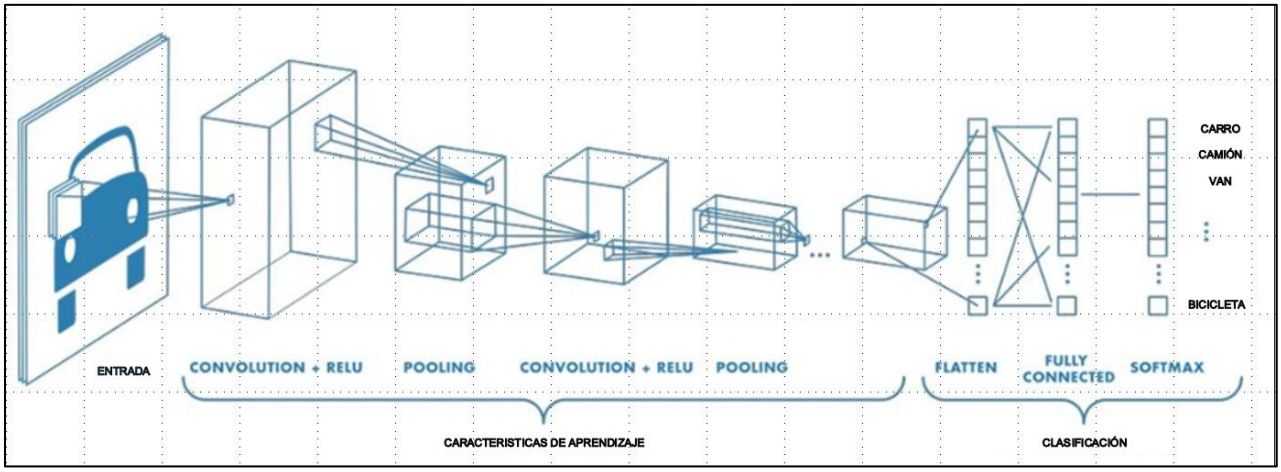
$$y_k = \Omega(\vec{x} * \overrightarrow{w_k} + b_k) \quad (1)$$

La señal que pasa por una neurona es transformada por pesos que modifican las funciones y por tanto la señal de salida que llega a la siguiente neurona. Modificando los pesos para todas las neuronas en la red, cambia la salida. (Rumelhart D.E, 1986) .

### 5.5.2 Redes Neuronales convolucionales

Las redes neuronales convolucionales (CNN) son redes neuronales artificiales (ANN) multicapas. Ver figura 4 Que a diferencia del perceptrón multicapa (MLP) las conexiones no son del todo conjuntas; cada neurona de la red no se une con todas y con cada una de las capas siguientes, sino que se generan conexiones entre grupos de manera interna en la misma capa. Están diseñadas para reconocer patrones visuales directamente a partir de imágenes de píxeles con un preprocesamiento mínimo (LeCun, 2010). Son un método eficaz para la clasificación de imágenes que utiliza capas convolucionales, agrupadas y completamente conectadas para el proceso de aprendizaje (Momeny et al., 2021). En cuanto a tareas de clasificación y regresión. Por ejemplo, las redes neuronales convolucionales han demostrado un alto rendimiento y funcionalidad (Grus, 2015). En (Sibo et al., 2016) se realiza una evaluación de rendimiento de algoritmos de aprendizaje profundo (Deep Learning) para CNN, Deep Neuron Networks (DNN) y Recurrent Neuron Network (RNN) donde se evidencia que los CNN tienen mejor desempeño en tareas de clasificación.

***Figura 4****.Estructura general de una Red Neuronal artificial Convolucional*




*Nota*: La figura representa una estructura de Red Neuronal Artificial simple. Tomado de Web Matlab y adaptado, https://la.mathworks.com/discovery/convolution.html

Al ver cada neurona como una unidad generadora de valor, las redes con múltiples capas aseguran un mejor aprendizaje y, por lo tanto, mejores resultados (Shen & Wang, 2018)(Heng & Weise, 2019). Sin embargo, también aumenta su complejidad, lo que hace que la red sea difícil de optimizar y más fácil de sobreajustar. (Gu et al., 2018). Al incrementar la densidad de capas en una red, el tiempo de procesamiento de la información aumenta significativamente, debido a la gran cantidad de neuronas presentes en la estructura que conllevan al incremento del número de operaciones por resolver (Song et al., 2019). Lo cual se ha venido solucionando mediante la

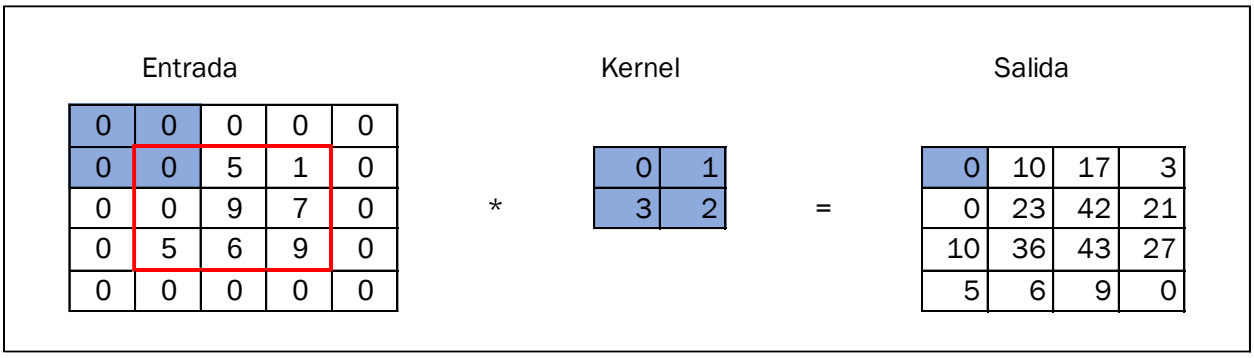
integración de GPU y TPU que permiten reducir el tiempo de procesamiento de la información considerablemente.

La convolución, en sí, consiste en tomar “grupos de pixeles cercanos” de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una matriz denominada kernel. Ese kernel “recorre” todas las neuronas de entrada y genera una nueva matriz de salida, que se convierte en una nueva capa de neuronas ocultas.

- Kernel o filtro

El kernel es una matriz que se mueve sobre los datos de entrada, realiza el producto escalar con la subregión de los datos de entrada y obtiene la salida como matriz de productos escalares. El Kernel se mueve en los datos de entrada por el valor de zancada (Stride). Si el valor de zancada es 2, el núcleo se mueve 2 columnas de píxeles en la matriz de entrada. Esto de manera gráfica se presenta en la figura 5.

***Figura 5.*** *Representación gráfica de una operación de Kernel.*




*Nota*: La figura representa una ilustración de una operación entre un Kernel 2x2. Autoría propia

Adicional, a la convolución en las CNN existen otras operaciones de gran relevancia: Padding, Stride y Pooling.

- Stride

En la convolución original, el kernel se desplaza un pixel hacia la derecha, o hacia abajo, durante cada iteración. Cada uno de estos desplazamientos se conoce como stride (o salto), y en la convolución.

- Padding

Consiste en agregar pixeles con valor de cero a los bordes de las imágenes originales, con el fin de una vez realizada la convolución, la imagen resultante como la original sean del mismo tamaño. Esto permite adquirir mayor información durante el entrenamiento.

- Pooling

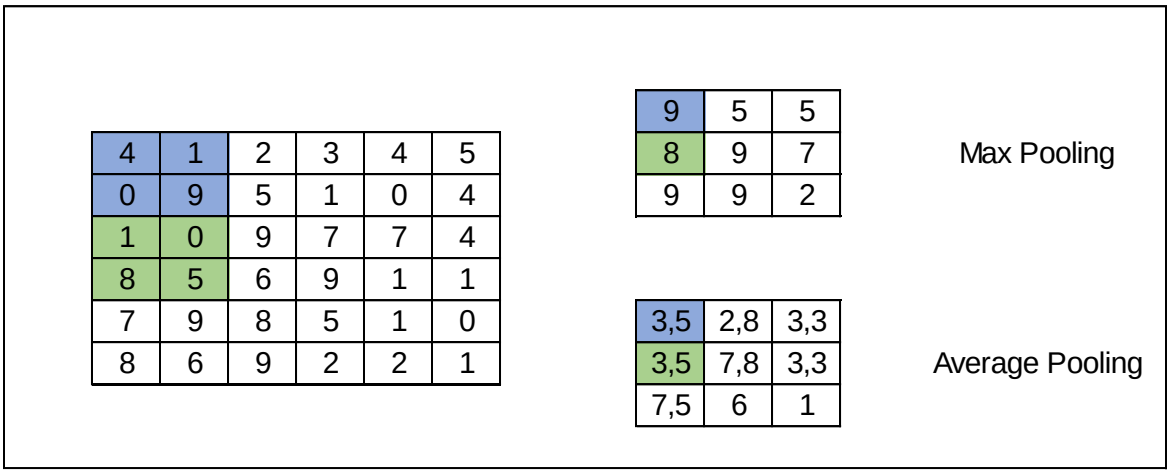
El pooling es una operación que permite analizar el contenido de una imagen por regiones (o bloques) para extraer la información más representativa de las mismas; permite reducir la

cantidad de datos entre una capa y otra, facilitando así el procesamiento de las imágenes y el entrenamiento de la red, pero a la vez preservando la información más relevante.(Salman et al., 2018)

Existen varios tipos de agrupaciones siendo la agrupación máxima (Max Pooling) y promedio (Average Pooling) las más usadas.

En la agrupación máxima, el filtro simplemente selecciona el valor de píxel máximo en el campo receptivo mientras que en la agrupación promedio funciona calculando el valor promedio de los valores de píxeles en el campo receptivo. Ver imagen 6

***Figura 6.*** *Representación gráfica del Max Pooling y Average Pooling.*




*Nota*: La figura representa una ilustración de los dos tipos de Pooling más usados y sus diferencias.

### 5.2.3 Funciones en Redes Neuronales Convolucionales

En general, en la red existen tres funciones: inicialización (o de entrada), activación y normalización que permiten el procesamiento de la información y le agrega valor a la misma. (Rocio et al., 2011) .

En la inicialización se computan los pesos de cada neurona con la información de entrada, permitiendo hacer un prejuicio que luego en la activación, se determina si es tomado en cuenta o no. (Grus, 2015) y por medio de la función de normalización se busca disminuir el sobreajuste de los datos en la neurona.

#### *5.2.3.1 Funciones de inicialización*

La función de inicialización básicamente define y computa los pesos, con el fin de evitar que las salidas de activación exploten o desaparezcan a través de una CNN, ya que si esto ocurre los gradientes de perdida imposibilitan o reducen la probabilidad de que la red converja.

### Glorot initialization

El enfoque estándar actual para la inicialización de los pesos de las capas y nodos de la red neuronal se denomina “glorot" o "Xavier Inicialization”. Lleva el nombre de Xavier Glorot, actualmente investigador científico en Google DeepMind, el cual consiste en un esquema de inicialización para redes neuronales. Los sesgos se inicializan en 0 y los pesos $W_{ij}$ en cada capa se inicializan como:

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}} \quad \frac{1}{\sqrt{n}}\right] \tag{2}$$

Dónde $U$ es una distribución uniforme y *n* es el tamaño de la capa anterior (número de columnas en W).

#### *5.2.3.2 Funciones de Activación*

Tanto en las redes neuronales artificiales como biológicas, una neurona no sólo transmite la entrada que recibe. Existe un paso adicional, una función de activación, que es análoga a la tasa de potencial de acción disparando en el cerebro. (Gu et al., 2018)) La función de activación utiliza la misma suma ponderada de la entrada anterior y la transforma una vez más como salida; devuelve una salida que será generada por la neurona dada una entrada o conjunto de entradas. Cada una de las capas que conforman la red neuronal tienen una función de activación que permitirá reconstruir o predecir. Las funciones de activación, mayormente transforman el valor de entrada neto de la neurona en un valor dentro del rango (0, 1) o (-1, 1). Cuando el valor de activación de una neurona (salida de su función de activación) es cero, se dice que la neurona está inactiva, ya que no pasa ningún tipo de información a las siguientes neuronas. Las funciones de Activación más comunes son: Rectified linear unit, Sigmoide, Tangente hiperbólica. (Amar Rodrigo, 2021)

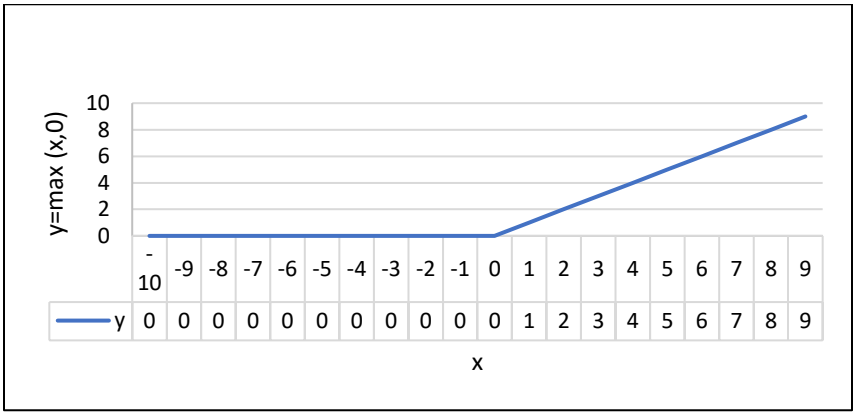
En la presente investigación se da uso a la función ReLu.

### Activación Rectified linear unit- Relu

La función de activación Rectified linear unit *ReLu* aplica una transformación no lineal muy simple, activa la neurona solo si el *input* está por encima de cero. Mientras el valor de entrada está por debajo de cero, el valor de salida es cero, pero cuando es superior de cero, el valor de salida aumenta de forma lineal con el de entrada.(Amar Rodrigo, 2021)

$$ReLu(x) = \max(x, 0) \tag{3}$$

***Figura 7.*** *Función RELU*




*Nota*: El gráfico representa la función RELU f(x)=max(x,0). La cual no toma valores negativos sino siempre acotados en cero.

En términos generales la ReLu es un clasificador que activa o no la neurona. Como se ve en la figura 7 dicha activación nunca es negativa.

### *5.2.3.3 Funciones de Normalización*

Las funciones de normalización tienen como finalidad evitar el sobreajuste en las Redes Neuronales sobre los datos. De ese modo se asegura que las diversas funciones tengan rangos de valores similares (escaldo de funciones) para que los descensos de gradiente puedan converger sin sobreajuste y a una mayor velocidad.

Una de las técnicas más comunes es la normalización por lotes (Batch Normalization).

#### Batch Normalization

Es un método que consiste en normalizar las salidas de las capas de una red. La normalización por lotes es una técnica para entrenar redes neuronales muy profundas que estandariza las entradas a una capa para cada mini-lote. Esto tiene el efecto de estabilizar el proceso de aprendizaje y reducir drásticamente la cantidad de épocas de entrenamiento necesarias para entrenar redes profundas. (Brownlee, 2019).En general ayuda a que los datos no se sobreajusten, ya que en el proceso existen, situaciones de desvanecimiento (Vanishing) que es una de las problemáticas asociadas a las redes de aprendizaje profundo(Song et al., 2019).

El Vanishing consiste en una pérdida de aprendizaje temprano. Donde el valor del gradiente cada vez se vuelve más y más pequeño lo que genera que la red al paso de las iteraciones no tenga un cambio sustancial y pierda su capacidad de aprender. Esto ha sido tratado por múltiples expertos y se evidenció que se puede evitar con técnicas de regularización inicial. Además, se ha comprobado que métodos como normalización por lotes (Batch normalization)(Joseph, 2016) también ayuda con este tipo de problemas. Por estas razones, utilizamos batch normalization para reducir el sesgo y variabilidad de la información y la inicialización glorot como técnica de normalización inicial(Glorot & Bengio, 2010)

## 5.6 Aprendizaje profundo CNN

El aprendizaje profundo (Deep Learning) es un subconjunto del aprendizaje máquina (Machine Learning), donde la característica principal es la gran cantidad de capas que se utiliza. El aprendizaje profundo tiene el propósito esencial de calcular características jerárquicas o representaciones de datos de observación, donde las características o factores de nivel superior se definen a partir de niveles inferiores (Aggarwal, 2018) . El Deep Learning, según (Moreno, 2020) "lleva a cabo el proceso de Machine Learning usando una red neuronal artificial que se compone de un número de niveles jerárquicos." Donde el aprendizaje se da capa a capa y gradualmente.

Una de las aplicaciones prácticas hoy día del DL (Deep Learning) es: Identificar marcas y logotipos de empresas en fotos publicadas en redes sociales, mejor comprensión de enfermedades, mutaciones de enfermedades y terapias genéticas, Localización de caras e identificación de emociones faciales, etc.

### 5.6.1 Redes Neuronales convolucionales profundas

Las redes neuronales convolucionales profundas (DCNN) son el tipo más utilizado para identificar patrones en imágenes y videos. Las DCNN han evolucionado a partir de las redes neuronales artificiales tradicionales, utilizando un patrón neuronal tridimensional inspirado en la corteza visual de los animales. Existen varios tipos de Redes: R-CNN, R-CNN Rapido, Red de Google, VGGNET y la Res Net que varía de acuerdo a su número de capas.

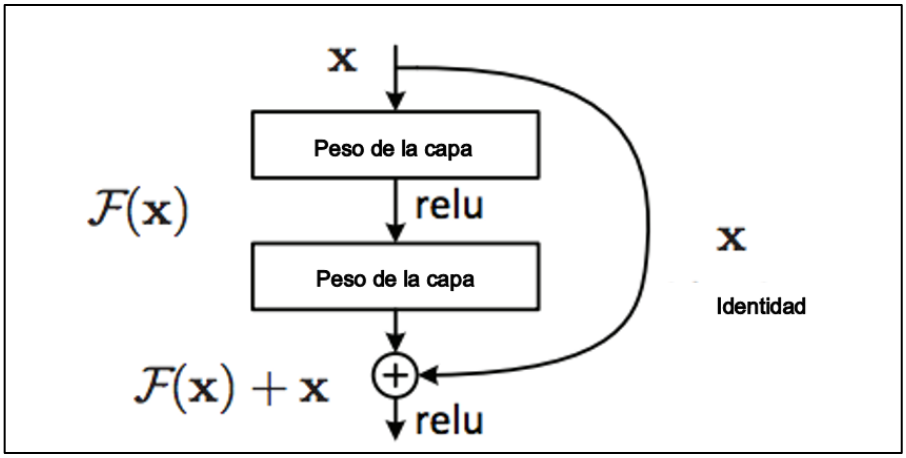
#### *5.6.1.1 ResNet*

La Red Neuronal Residual (ResNet) es un tipo de red artificial convolucional introducida en 2015 por Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun, mediante el articulo "Deep Residual Learning for Image Recognition", logrando en 2015 el primer lugar en la clasificación ILSVRC 2015 con una tasa de error entre los 5 primeros del 3,57 %. Estructuralmente, la ResNet usa "unidades cerradas" para omitir algunas capas convolucionales. Al igual que GoogleNet, utiliza una normalización por lotes. Ver figura 8.

El éxito de la ResNet, nace a razón de que sus antecesoras (AlexNet, VGG e incluso Google Net) presentaban problemas con el aumento de capas; entre más profunda se hacía más difícil entrenarla. Uno de esos problemas, es la degradación (Degrading) en la cual a mayor número de capas, en cierto punto, el error en lugar de continuar disminuyendo se estabiliza o en su defecto aumenta, por lo cual la precisión de la Red desciende ya que los solver no logran encontrar una mejor solución y por el contrario pierden precisión y calidad en la respuesta(Roy et al., 2018).

Para evitar caer en la degradación se comenzó a usar la red residual , específicamente una de 50 capas- ResNet 50-) (Joseph, 2016) que consiste en hacer saltos de información de una capa nodo a otra capa nodo garantizando que el solver no se pierda en la profundidad de la red y en el peor de los casos se obtenga el mismo resultado que una shallow net o red neuronal simple (He et al., 2016)

***Figura 8**. Estructura de una red ResNet*




Nota: Aprendizaje residual en un bloque de construcción, en donde ResNet intenta hacer que F(x)= 0 sea X=Y. Tomado de *Deep Residual Learning for Image Recognition*, por He,2016, CVF

La ResNet 50 es una variante del modelo ResNet que tiene 48 capas de convolución junto con 1 capa MaxPool y 1 capa de promedio (normalmente existen 2 tipos de bloques en la red, de identidad y de convolución). Tiene operaciones de 3,8 x 10^9 puntos flotantes.(Kaushik, 2018).

#### *5.6.2.2 Inception Learning Google*

Inception v2 es la segunda generación de arquitecturas de redes neuronales convolucionales de Inception que utiliza notablemente la normalización por lotes (Ioffe & Szegedy, 2015) (la primera, Inception v1, es una arquitectura CNN a gran escala que ganó el Desafío ImageNet en 2014). La arquitectura consiste en una CNN profunda basada en pequeñas convoluciones, llamadas "inicios" y normalización por lotes.

Nace con la intención de Reducir el cuello de botella representacional (perdida de información a causa de reducción de las dimensiones de la entrada) mediante métodos de factorización inteligentes que generan que las convoluciones sean más eficientes.(Szegedy et al., 2014)

## 5.7 Algoritmos de Optimización

En las redes Neuronales profundas existen diferentes tipos de algoritmos de optimización, como el descenso del gradiente simple y con momentum, la media móvil ponderada y el que usaremos en el presente trabajo, el RMSprop.

### 5.7.1 RMSprop

Su nombre proviene de Root Mean Square Propagation, es decir, la propagación por la raíz de la media de los cuadrados, propuesto por Geoff Hinton en la lección 6 del curso en línea " Redes neuronales para el aprendizaje automático " de Coursera. RMSprop escala la tasa de aprendizaje para que los algoritmos pasen por el punto de silla más rápido que la mayoría y es considerado como uno de los mejores algoritmos de optimización.(Vitaly, 2018).

Una de las falencias de los algoritmos de aprendizaje, como por ejemplo el de AdaGrad es que siempre se usa el mismo tamaño de paso a taza de aprendizaje, lo que produce un mayor consumo de recursos; mayor tiempo en búsqueda y en ocasiones soluciones sin punto óptimo. Por lo cual, al ser, RMSprop una extensión del descenso de gradiente y la versión AdaGrad del descenso de gradiente que utiliza un promedio decreciente de gradientes parciales en la adaptación del tamaño de paso para cada parámetro, permite que por medio del uso de un promedio móvil decreciente el algoritmo olvide los gradientes iniciales y se concentre en los gradientes parciales observados más recientemente durante el progreso de la búsqueda, superando la limitación de AdaGrad.(Mean & Propagation, 2021).

## 5.8 Framework

Para la aplicación de las arquitecturas se utilizan los frameworks: Tensorflow y Keras

### 5.8.1 Tensorflow

TensorFlow es una plataforma de código abierto de extremo a extremo para el aprendizaje automático, creada por Google en 2015, permite entre otras cosas construir y entrenar Redes Neuronales.(TensorFlow, 2020). TensorFlow agrupa una gran cantidad de modelos y algoritmos de aprendizaje automático y aprendizaje profundo que por medio de Python o JavaScript proporciona una API frontal conveniente para crear aplicaciones, mientras ejecuta esas aplicaciones en C++ de alto rendimiento (Yegulalp, 2022)

En si TF, presenta una arquitectura flexible que permite una fácil implementación en gran variedad de plataformas y dispositivos.(Agustina, 2018).

### 5.8.2 Keras

Keras es una librería para Redes Neuronales Artificiales de alto nivel escrita en lenguaje de programación Python. (Agustina, 2018) . Más allá de la facilidad de aprendizaje y la facilidad de creación de modelos, Keras ofrece las ventajas de una amplia adopción, soporte para una amplia gama de opciones de implementación de producción, integración con al menos cinco motores de back-end (TensorFlow, CNTK, Theano, MXNet y PlaidML), y fuerte soporte para múltiples GPU y entrenamiento distribuido..(Heller, 2019)

Empresa como Google, Microsoft, Amazon, Apple, y Uber utilizan la Apli para sus desarrollos, lo que denota la fiabilidad de la misma.

## 5.9 Métricas

La evaluación del modelo se puede realizar por medio de métricas las cuales dependen del objetivo del mismo, las más usadas con la pérdida y la presión del modelo.

### 5.9.1 Accuracy

La precisión es un método para medir el rendimiento de un modelo de clasificación que representa el ratio entre verdaderos positivos y número total de positivos predichos. (También se

expresa como un valor porcentual). En términos generales, indica el porcentaje de aciertos en las tareas de clasificación:

$$Precisón(x) = \frac{Aciertos}{Total} \quad (4)$$

### 5.9.2 Loss

Una función de pérdida, también conocida como función de costo, tiene en cuenta las probabilidades o la incertidumbre de una predicción en función de cuánto varía la predicción del valor real; A partir de la función de pérdida, se derivan los gradientes que se utilizan para actualizar los pesos. (O, 2021). Cuanto menor sea el valor, mejor ajuste de la Red Neuronal.

A diferencia de la precisión, la pérdida se puede utilizar tanto en problemas de clasificación como de regresión. Las funciones de pérdida más comunes son la pérdida logarítmica, la pérdida de entropía cruzada (que arrojan el mismo resultado al calcular tasas de error entre 0 y 1) y la entropía binaria cruzada.

La pérdida por entropía binaria cruzada es utiliza para procesos de clasificación que como su nombre lo indica solo tiene dos valores de respuesta (O, 2021) :

$$L = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (4)$$

Donde:

$y_i$ : representa el resultado esperado

$\hat{y}$ : representa el resultado producido por nuestro modelo.

## 5.10 Aumento de Datos

El aumento de datos abarca un conjunto de técnicas que mejoran el tamaño y la calidad de los conjuntos de datos de entrenamiento, de modo que se pueden construir mejores modelos de aprendizaje profundo con ellos.(Shorten & Khoshgoftaar, 2019), Estos aumentos inflan artificialmente el tamaño del conjunto de datos de entrenamiento mediante la deformación o el sobremuestreo de los datos.

El aumento de datos en si ayuda a reducir el sobreajuste, aumenta el número de datos para entrenamientos, y ayudan a generar variaciones en la data para mejorar el aprendizaje de la red.

# 6. Metodología

El presente capítulo del trabajo de investigación expone la propuesta metodológica para evaluar la pertinencia y desempeño de redes neuronales en el proceso de inspección de calidad de aguacate Hass para exportación mediante el uso de procesamiento de imágenes como mecanismo de clasificación alternativo a los métodos tradicionales.

En este sentido, la metodología utilizada se definió en cuatro fases las cuales corresponden a cada uno de los 4 objetivos específicos:

### Fase 1: Criterios de evaluación de unidades

En la primera fase se realiza la caracterización del proceso de inspección de calidad del aguacate Hass para exportación con el fin de identificar las variables que rigen el proceso y así mismo indagar sobre estudios realizados de aplicabilidad de redes neuronales en procesos de inspección y clasificación.

***Tabla 2.*** *Criterios de evaluación de unidades.*

| OBJETIVO | ACTIVIDADES | MÉTODO |
|---|---|---|
| Determinar las características que rigen el proceso de inspección de calidad del aguacate Hass para exportación, mediante una revisión de la literatura y los procesos establecidos en la industria de exportación. . | 1. Documentar proceso de inspección de aguacate Hass. | Método Exploratorio |
| | 2. Establecer criterios de clasificación de unidades de aguacate Hass aptas para exportación. | |
| | 3. Indagar sobre estudios de uso de redes neuronales en procesos de clasificación. | |
| | 4. Indagar sobre estudios de uso de redes neuronales en procesos de clasificación de aguacate Hass. | |

*Nota*: En esta tabla se resume las actividades y métodos a efectuar para el alcance del objetivo No 1.

### Fase 2: Dataset

En la segunda base se propende la construcción del banco de datos necesario para la alimentación de las redes neuronales.

***Tabla 3.*** *Fase objeto de la presente investigación.*

| OBJETIVO | ACTIVIDADES | MÉTODO |
|---|---|---|
| Generar un banco de datos mediante la captura, procesamiento y aumento de imágenes (Data Augmentation) para el entrenamiento, validación y testeo de las redes neuronales. | 1.Adquisición de unidades de aguacate Hass aptas y no aptas para exportación | Método Empírico |
| | 2.Realizar registro fotográfico | |
| | 3.Clasificación y etiquetado | |
| | 4.Aumento de volumen data mediante técnicas computacionales | |
| | 5.Validación y etiquetado de Data aumentada | |
| | 5.División de dataset | |

*Nota*: En esta tabla se resume las actividades y métodos a efectuar para el alcance del objetivo No 2

### Fase 3: Estructura de Red

En la tercera etapa se definen qué redes neuronales se deben implementar con el fin de validar la pertinencia de estas en el proceso de inspección.; posterior a la revisión en la literatura se determina que se evalúa el desempeño de redes neuronales artificiales de aprendizaje supervisado (profundo y no profundo) en el proceso de inspección de calidad. Mediante la evaluación de 3 arquitecturas: CNN simple, Inception Net y ResNet 50.

***Tabla 4.*** *Estructura de la Red.*

| OBJETIVO | ACTIVIDADES | MÉTODO |
|---|---|---|
| Entrenar y validar las estructuras CNN simple, Inception Net y ResNet 50 mediante la Api de Kerras y Tensorflow . | 1.Conocer las estructuras de las Redes Neuronales: CNN simple, Inception Net y ResNet 50 | Método Empírico y Exploratorio |
| | 2.Codificar las redes neuronales | |
| | 3.Entrenar las redes con el dataset. | |
| | 4.Validar las redes con el dataset | |

*Nota*: En esta tabla se resume las actividades y métodos a efectuar para el alcance del objetivo No 3

### Fase 4: Testeo de Red

Por último, se realiza validación de las estructuras neuronales codificadas, entrenadas y validadas mediante la comparación de porcentajes de precisión.

***Tabla 5****.Testeo de la Red.*

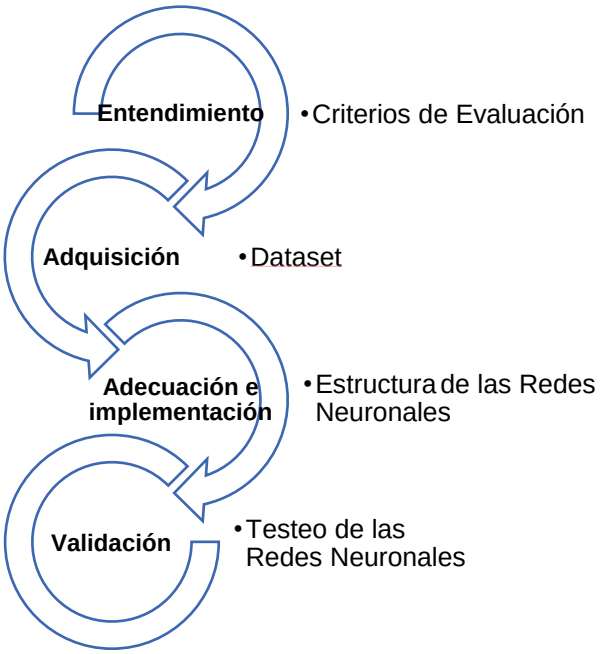
| OBJETIVO | ACTIVIDADES | MÉTODO |
|---|---|---|
| Validar el desempeño de las redes implementadas con respecto a un nivel de precisión del 90%. | 1.Testeo y tunning de las 3 redes neuronales con el dataset. | Método empírico, hipotético y deductivo |
| | 2.Comparación de desempeños | |
| | 3.Expecificación de resultados | |

*Nota*: En esta tabla se resume las actividades y métodos a efectuar para el alcance del objetivo No 4

El ciclo de vida de la aplicación se explica gráficamente mediante la figura 9. Este está subdividido en cuatro categorías en la que la primera es la fase de entendimiento, la segunda de adquisición, la tercera de adecuación e implementación y la última de validación.

A modo de síntesis en la figura 9 se detalla el ciclo de vida de la evaluación.

***Figura 9.*** *Ciclo de vida de la aplicación*




*Nota*: El gráfico representa el proceso metodológico de la implementación.

# 7. Desarrollo

## 7.1 Criterios de clasificación de aguacate Hass

Como criterios de aceptación de unidades conformes para el entrenamiento de la red neuronal se tiene que en la capa externa el aguacate no debe presentar fisuras, manchas, cambios de color ajenos al verde característico del aguacate sin madurar. Debe presentar el pedúnculo y este debe estar entero, sin grietas o faltantes.

Cabe resaltar que las unidades catalogadas como conformes para el entrenamiento de la red neuronal no tiene en cuenta factores como peso y tamaño. Estos mismos criterios (Aspecto físico) se dan para la fase de validación y pruebas.

## 7.2 Dataset

El Banco de datos se encuentra clasificado por dos categorías que son los aguacates que cumplen con los criterios de clasificación y los que no cumplen con dichos criterios.

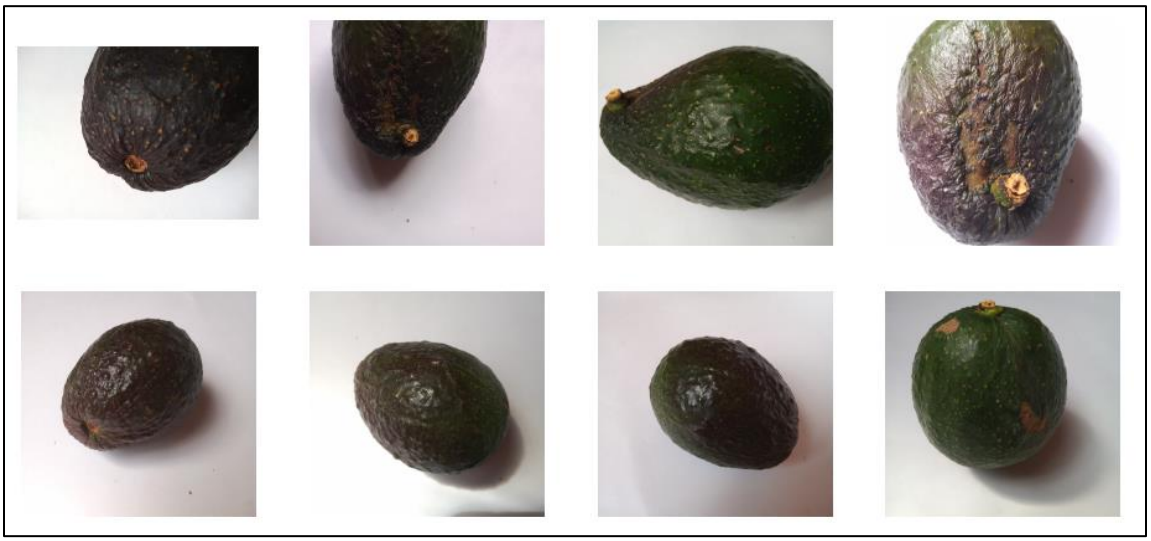
En la figura 10 se evidencia una muestra del registro que se catalogó como unidades aptas para exportar, las cuales presentan las características físicas descritas en el literal anterior. Por su parte en la figura 11 se evidencian algunas de las características que no permiten que las unidades sean objeto de exportación.

***Figura 10**. Banco de fotos de aguacate Hass- Conformes o aptas*



*Nota*: En esta figura se detalla el dataset de aguacates aptos para exportación y las diferentes ubicaciones dentro del banco de datos elaborado.

***Figura 11**. Banco de fotos de aguacate Hass-No conformes*



*Nota*: En esta figura se detalla el dataset de aguacates no aptos para exportación.

Para tomar el registro fotográfico se seleccionaron 93 aguacates que cumplían con las características de las unidades conformes, la mayor parte se fotografiaron por tres caras en

diferentes posiciones obteniendo al final un registro de 320 imágenes. Así mismo se procedió con las unidades no aptas. Donde se fotografiaron 90 aguacates obteniendo un registro fotográfico de 325 fotografías.

Las imágenes se capturaron en condiciones controladas donde el registro se hizo con una misma cámara y en el mismo espacio. Sin embargo, para incrementar la complejidad del banco de fotos se tuvieron en cuenta dos variables; iluminación y distancia. Las fotos presentan diferentes contrastes de luz en las que se encuentran diferencias en saturación e intensidad lumínica. Con respecto a la distancia, las imágenes en su mayoría fueron tomadas a 30 cm del modelo. En algunos casos esa distancia se redujo a 5 cm y en otros aumentó hasta 50 cm.

Así mismo, en aras de aumentar la complejidad del dataset, se incluyeron fotografías de limones, en el conjunto de datos de entrenamiento, clasificadas como no conformes.

Las redes neuronales al ser algoritmos estadísticos requieren de una muestra grande y diversa para reducir la variabilidad y para mitigar el sesgo se hace necesario que dicha muestra sea homogénea (los nichos tengan la misma proporción en la muestra que en la población). Es por ello que en el dataset original se utilizó la técnica “data augmentation” mediante la librería opencv disponible en python y la conocida api keras. Con ello, se logró aumentar el dataset a un total de 1.185 imágenes de aguacates no conformes y 1.281 fotos de aguacates conformes. Así mismo, para la verificación y validez de la data, se hizo una revisión imagen por imagen para garantizar la validez del dataset y en la depuración final quedaron 1.177 fotos de unidades aptas y 1.273 fotos de unidades no aptas.

Por otro lado, se dimensionan las imágenes para facilitar el procesamiento de estas, el tamaño final de cada foto es de 256 x 256 píxeles. Además, se agrupan en lotes de 32 imágenes tanto para optimizar recursos de hardware como para acelerar el tiempo de entrenamiento.

Para la alimentación de la red neuronal se utiliza la técnica 60-40. Donde el 60% del registro fotográfico (1.470 fotos) corresponde al entrenamiento y el 40% restante (980 fotos) restante se divide, el 50% para la validación y el restante 50% para la prueba. Teniendo lotes de alimentación de 32 imágenes.

## 7.3 Estructura de Redes Neuronales

Dado que el propósito de esta investigación es lograr clasificar las unidades de aguacate Hass aptas para exportación teniendo como información de entrada un banco de imágenes. Se recurre al uso e implementación de tres redes neuronales convolucionales donde una de ellas es una CNN (convolutional neural network) simple y dos de ellas son de aprendizaje profundo. La primera se caracteriza por su rápido entrenamiento y fácil convergencia de la solución. Mientras que las CNN profundas son destacadas por su complejidad en el cómputo de las soluciones y por la robustez de sus arquitecturas.

Como uno de los objetivos de este trabajo es demostrar que con redes neuronales se puede clasificar los aguacates de exportación con una precisión del 90%. En principio se determinó tres estructuras de redes neuronales a entrenar sin tener en cuenta los hiper parámetros. Dado el caso que ninguna de las redes presente un valor de precisión por encima del 90%. Se recurre a realizar la ingeniería de datos y ajuste de hiper parámetros (tunning) en la arquitectura con mayor precisión.

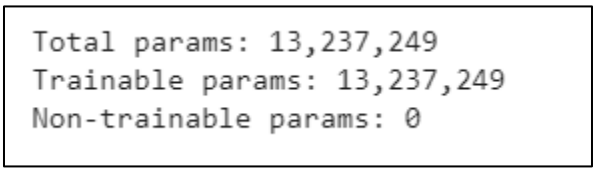
Para este caso las CNN profundas escogidas fueron la Resnet 50 y la Inception Learning de Google, debido a que son estructuras reconocidas para afrontar problemas comunes en las redes neuronales profundas como el vanishing y el degrading y además tienen una alta eficiencia en tiempo de entrenamiento.

### 7.3.1 Estructura Red Neuronal Convolucional Simple

Las redes neuronales convolucionales han demostrado una alta eficiencia a la hora de clasificar imágenes. Sin embargo, es conocido que al igual que una red neuronal simple pueden presentar dificultades en tiempo de entrenamiento que pueden bajar la eficacia de la red neuronal. Dentro de los problemas más conocidos está el sobreajuste (overfitting), el bajo ajuste de los pesos (underfitting), los cuellos de botella en el entrenamiento entre otros. Por tanto, se optó por una estructura relativamente simple, pero con las suficientes capas para evitar el underfitting. Donde cada una de las capas de entrenamiento utiliza dropout como técnica de normalización inicial con una tasa de 0.2. y como activación se utiliza la función Relu. Además de un maxpooling por capa reduciendo el tamaño de la imagen para disminuir el tiempo de entrenamiento.

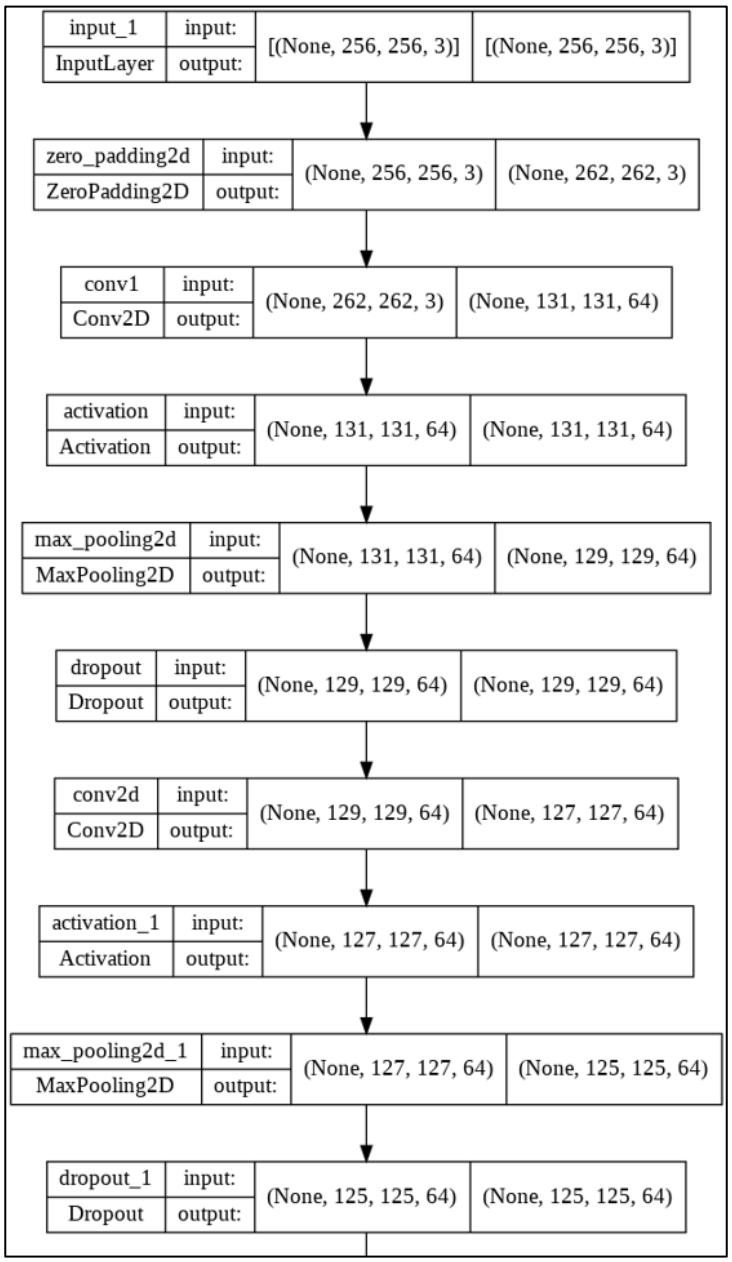
A continuación, se muestra la cantidad total de parámetros utilizados en tiempo de entrenamiento (Figura 12) y las primeras capas utilizadas en el modelo para entrenar la red neuronal en la figura 13.

***Figura 12***. *Número total de parámetros utilizados en el entrenamiento para la red neuronal CNN no profunda.*




*Nota*: Cantidad de parámetros utilizados en el entrenamiento de la CNN no profunda.

***Figura 13**. Primeras 10 capas de entrenamiento de la red neuronal CNN no profunda con algunos hiper-parámetros utilizados.*




*Nota*: En la figura se detallan las primeras 10 capas de la estructura. Para mayor detalle del total de las capas ver anexo 1

De igual manera, la estructura del código implementado se puede detallar en el anexo 4.

### 7.3.2 Estructura Red Neuronal Convolucional Profunda - ResNet 50

Dado que se cuenta con un dataset relativamente “pequeño” se opta por una red residual de 50 capas ya que una estructura más grande no agregaría valor al resultado. Para inicializar los pesos se utiliza la inicialización Glorot o mejor conocida como Xavier Uniform Initialization. En cuanto al mecanismo de activación, se utiliza la denominada activación RELU (Rectified Linear Unit) para cada una de las capas convolucionales.

Se utilizó en la última capa, la activación Sigmoid con el fin de clasificar las imágenes de forma binaria. Adicional, se emplea Batch normalization en cada una de las capas con el fin de simplificar la actualización de los parámetros y con ello acelerar el entrenamiento de la red.

Los bloques utilizados fueron:

- Bloque convolucional: Se usa cuando las dimensiones de la entrada y la salida no coinciden.
- Bloque identidad: Empleado cuando coinciden ambas dimensiones.

Cada bloque está compuesto por tres capas donde las dos primeras son capas convolucionales convencionales y el último contiene a su vez la unión entre la entrada y salida, para luego generar su correspondiente activación.

En la implementación de la RES NET 50 se utilizaron 4 etapas para el procesamiento de la información y dos etapas auxiliares; en una (etapa inicial) se acondicionaron las imágenes para poder utilizar los bloques de convolución y en la otra (etapa salida) se usó para aplanar la información para luego ser procesada por la neurona encargada de evaluar la red por medio de la activación sigmoid.

La conformación de las 4 etapas de procesamiento fue:

●Etapa 1: Dimensiones del filtro [256,256,256]

Bloque convolución

Bloque identidad

Bloque identidad

● Etapa 2: Dimensiones del filtro [512,512,512]

Bloque convolución

Bloque identidad

Bloque identidad

Bloque identidad

●Etapa 3: Dimensiones del filtro [1024,1024,1024]

Bloque convolución

Bloque identidad

Bloque identidad

Bloque identidad

Bloque identidad

Bloque identidad

●Etapa 4: Dimensiones del filtro [2048, 2048, 2048]

Bloque convolución

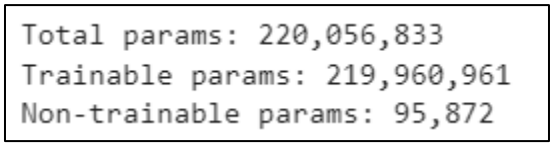
Bloque identidad

Bloque identidad

Por último, se utilizó la función de clasificación Binary Cross Entropy para la evaluación de los pesos y el optimizador RMS prop con una tasa de aprendizaje de 0.001 para buscar la solución en el espacio de soluciones.

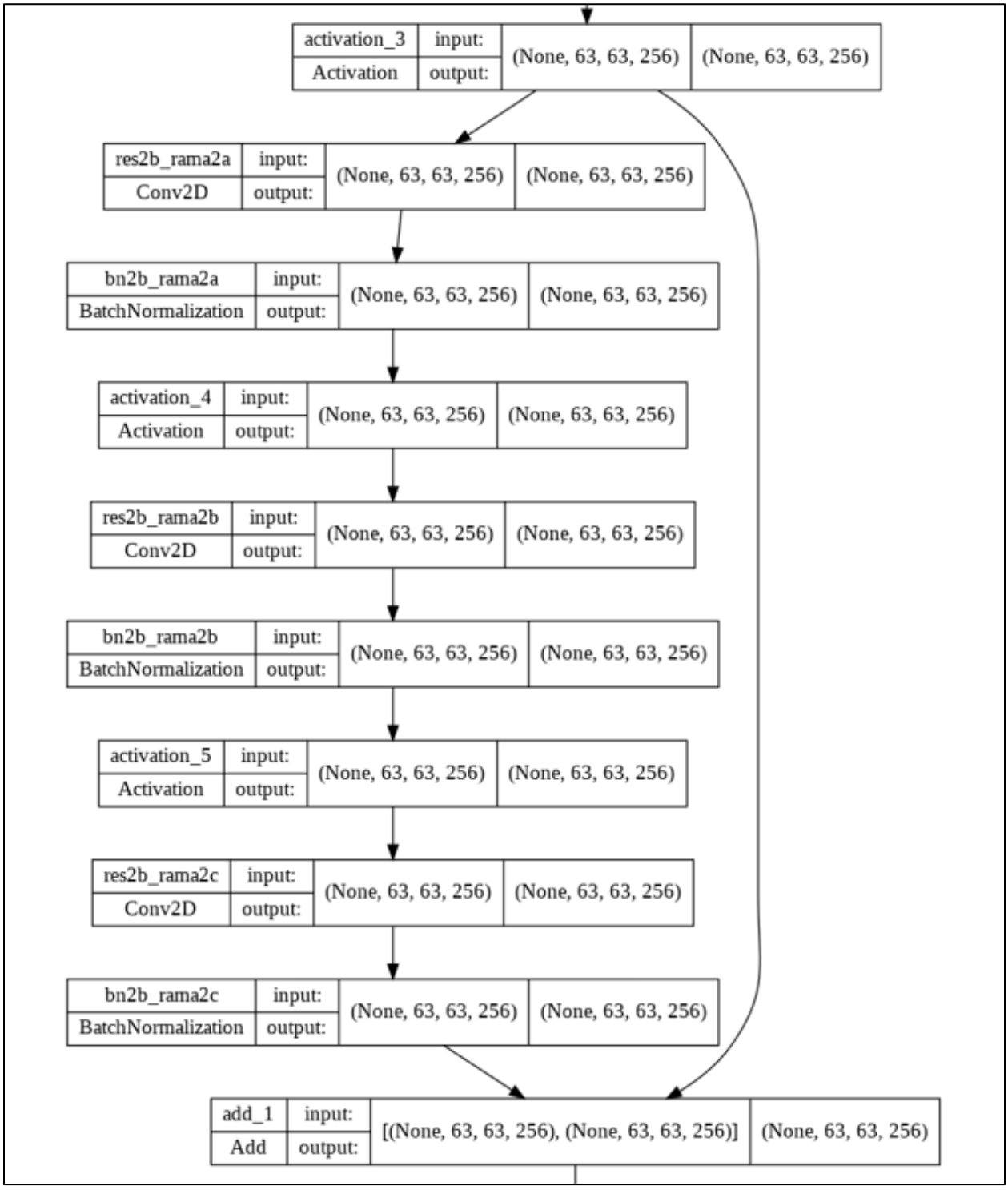
A continuación, se muestra la cantidad total de parámetros utilizados en tiempo de entrenamiento y las primeras capas utilizadas en el modelo para entrenar la red neuronal.

***Figura 14.*** *Número total de parámetros utilizados en el entrenamiento para la red neuronal ResNet50.*




*Nota*: Cantidad de parámetros utilizados en el entrenamiento de la ResNet 50.

***Figura 15***. *Bloque de convolución de la red neuronal ResNet50 con algunos hiper parámetros utilizados.*




*Nota*: En la figura se detallan uno de los bloques de convolución de la estructura. Para mayor detalle ver anexo 2

En la anterior imagen se logra apreciar los saltos de la capa inicial hasta la última capa de cada uno de los bloques de convolución y las dimensiones de cada una de ellas.

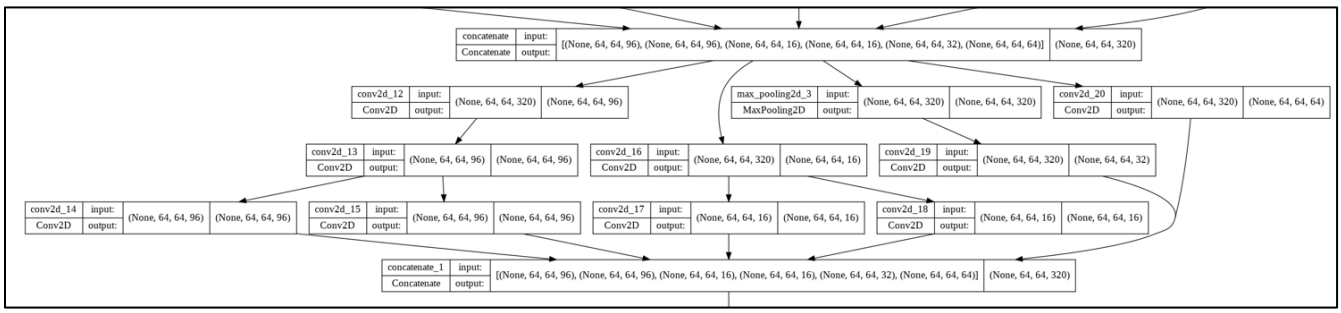
De igual manera, la estructura del código implementado se puede detallar en el anexo 5.

### 7.3.3 Estructura Red Neuronal Convolucional Profunda - Inception Learning Google

Se utiliza una Inception network en la cual se hace un acondicionamiento de los datos en las primeras capas de la red neuronal con el objetivo de reducir las dimensiones de las imágenes replicando la estructura de la Google Net. Luego de ello, se utilizan bloques de incepción donde se obtienen cuatro diferentes ramas que utilizan bloques de convolución con un padding same.

Para mantener las dimensiones de las imágenes y el volumen del bloque de convolución se utiliza un diferente número de filtros que determina el volumen de la capa de unión o capa final del bloque de incepción. Cada una de las ramas tiene diferentes tamaños del kernel. A su vez, dos de las ramas usadas se dividen en dos ramas más, donde las dimensiones del kernel de la primera rama es (1,j) y el de la segunda es (j,1). A continuación, se muestra uno de los bloques empleados a modo de ejemplo.

***Figura 16**. Ejemplo de bloque de incepción utilizado en la implementación*




*Nota*: En la figura se detalla uno de los bloques implementados en la estructura. Para mayor detalle ver anexo 3.

La estructura está conformada por 8 bloques de incepción distribuidos de la siguiente manera:

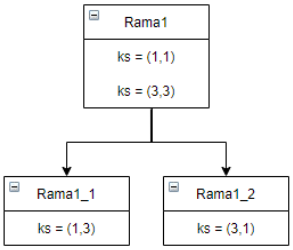
- Bloque de incepción [96, 16, 32, 64]
- Bloque de incepción MaxPooling
- Bloque de incepción
- Bloque de incepción MaxPooling
- Bloque de incepción
- Bloque de incepción
- Bloque de incepción

MaxPooling
- Bloque de incepción
- Bloque de incepción
  MaxPooling
- Bloque de incepción
- Bloque de incepción
  AveragePooling

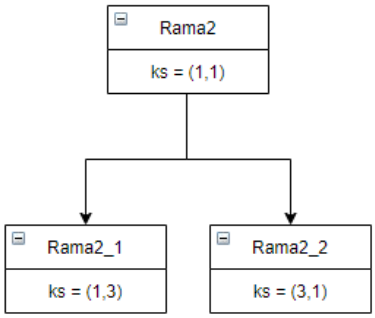
Solamente se muestra en el primer bloque de incepción el número de filtros utilizados por cada una de las ramas que constituyen el volumen final de cada bloque ya que el resto de los bloques utilizan el mismo número de filtros. Seguidamente, se especifican las dimensiones del kernel de convolución (ks).

***Figura 17.*** *Dimensiones del Kernel por bloque*

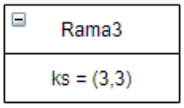
Rama 1



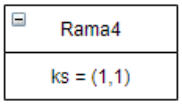

Rama 2




Rama 3



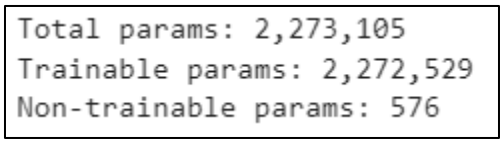

Rama 4




*Nota*: Cantidad de kernel por bloque.

En la figura 18, se muestra la cantidad total de parámetros utilizados en tiempo de entrenamiento y las primeras capas utilizadas en el modelo para entrenar la red neuronal.

***Figura 188.*** *Número total de parámetros utilizados en el entrenamiento para la red neuronal Inception (Google Net).*




*Nota*: Cantidad de parámetros utilizados en el entrenamiento de la Inception.

De igual manera, la estructura del código implementado se puede detallar en el anexo 6.

# 8. Resultados y Análisis

En este apartado se presentan los resultados conseguidos al realizar la implementación de las estructuras mencionadas en el capítulo anterior. Estos resultados muestran el poder de las redes neuronales para realizar tareas humanas que antes del machine learning y Deep learning eran impensables.

Los resultados son presentados para cada una de las estructuras y luego de ello, se hace un análisis en conjunto comparando las diferentes estructuras junto con sus resultados y las razones de las variaciones de los mismos.

Por cada una de las arquitecturas utilizadas se muestran las gráficas de pérdida (loss) y precisión (accuracy) en las diferentes épocas de entrenamiento. Sin embargo, para las gráficas de loss no se utilizan las primeras 10 épocas ya que son consideradas épocas de calentamiento (warm up epochs). En las gráficas de precisión se tienen en cuenta todas las épocas para poder evidenciar la evolución del entrenamiento y su adquisición de valor.

En todas las implementaciones se utilizó Detención temprana (Early stopping) como técnica de regularización, basado en la métrica de loss del conjunto de datos de validación. Con los pesos obtenidos por el callback (Función que implementa el early stopping) se hace la evolución del conjunto de datos de prueba (test set).

## 8.2 Resultados CNN simple

Los resultados de la estructura CNN no profunda muestra la incapacidad de la red neuronal para clasificar las imágenes de cada dataset con precisión, obteniendo en promedio un 52% de precisión en cada una de las fases como se muestra en la tabla 6. Por tanto, la probabilidad que la red neuronal no profunda clasifique correctamente las imágenes es equivalente en el largo tiempo a clasificar al azar los aguacates.

***Tabla 6.*** *Resultados de la CNN no profunda con los dataset de entrenamiento, validación y prueba*

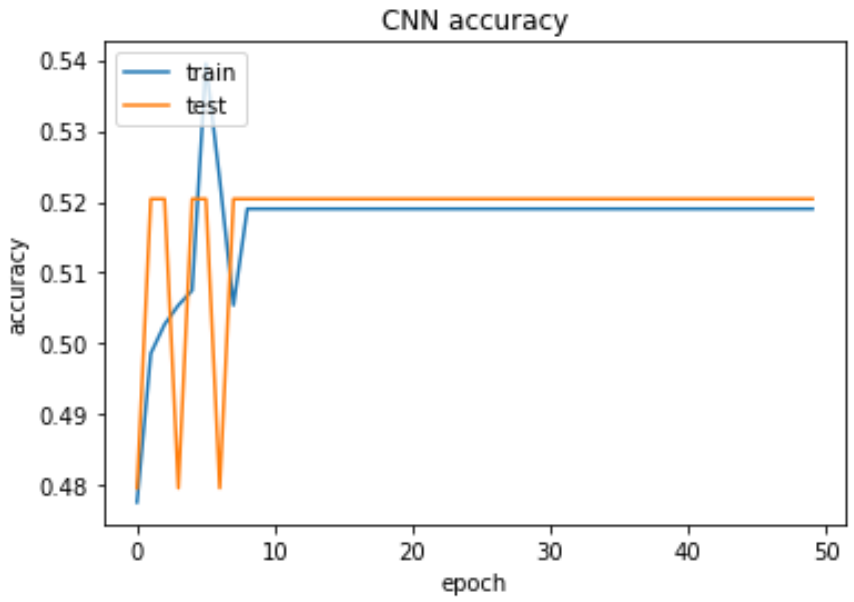
| FASE | PÉRDIDA | PRECISIÓN | TOTAL IMÁGENES | IMÁGENES EVALUADAS CORRECTAMENTE | IMÁGENES EVALUADAS ERRONAMENTE |
|---|---|---|---|---|---|
| Entrenamiento | 0.6925 | 0.5190 | 1470 | 763 | 707 |
| Validación | 0.6923 | 0.5204 | 490 | 255 | 235 |
| Prueba | 0.6923 | 0.5204 | 490 | 255 | 235 |

*Nota*: En la tabla se sintetizan los resultados obtenidos en las 3 fases y el número total de imágenes evaluadas.

Debido al bajo performance de la red neuronal se puede decir que hay un claro underfitting de la estructura frente a los datos. Sin embargo, al observar la figura 19. Se evidencia que en las 10 primeras etapas, la red neuronal intenta aprender y hay un movimiento significativo de la métrica de precisión por época o ciclo de entrenamiento. Pero una vez llega a la época 9 se estabiliza los resultados y se aprecia una pérdida de capacidad de entrenamiento.

En primera instancia se le puede atribuir la pérdida de capacidad de entrenamiento al vanishing. Sin embargo, no es común encontrar este problema en una red neuronal no profunda. Adicional se utilizó la función de inicialización Glorot que es una de las alternativas para contrarrestar este problema. Por tanto, se puede decir que la estructura simple no es capaz de abstraer las diferencias entre unidades aptas de las no conformes.

***Figura 199.*** *Resultados de la precisión de la CNN no profunda para el conjunto de datos de entrenamiento y validación*

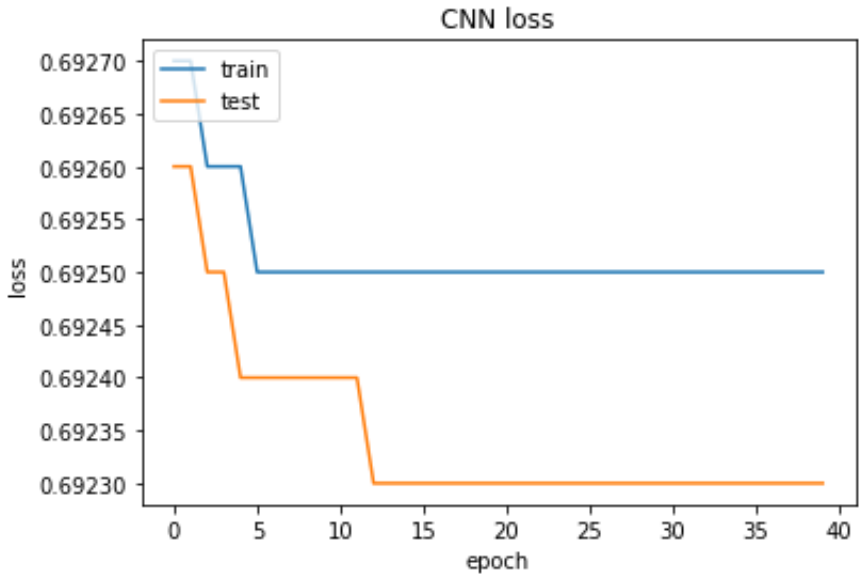



*Nota*: En la figura se muestra la precisión de la red en la fase de entrenamiento (línea azul) y validación (línea naranja) respecto al número de épocas.

Analizando el loss de la CNN no profunda se logra apreciar que los valores del gradiente en el conjunto de datos mejoran hasta la época 14 (figura 20) y en la validación hasta la 21. Luego de ello, los valores permanecen estáticos, lo que confirma la incapacidad de la CNN no profunda de clasificar el dataset.

.

***Figura 200***. *Resultados de la pérdida (loss) de la CNN no profunda para el conjunto de datos de entrenamiento y validación.*




*Nota*: En la figura solamente se tienen en cuenta las últimas 40 época, las primeras 10 se consideran warm up y se pierde la dimensión de la mejora en los ciclos posteriores.

## 8.3 Resultados ResNet50

La ResNet50 presenta unos resultados sobresalientes. Ver tabla 7. Sin ajuste de hiper parámetros la precisión en cada una de las fases es por encima del 90% lo cual cumple con los objetivos de este proyecto, en promedio se tiene un 95% de precisión en cada una de las fases.

***Tabla 7.*** *Resultados de la ResNet50 con los dataset de entrenamiento, validación y prueba*

| FASE | PÉRDIDA | PRECISIÓN | TOTAL IMÁGENES | IMÁGENES EVALUADAS CORRECTAMENTE | IMÁGENES EVALUADAS ERRONAMENTE |
|---|---|---|---|---|---|
| Entrenamiento | 0.1323 | 95.14% | 1470 | 1399 | 71 |
| Validación | 0.1619 | 94.29% | 490 | 462 | 28 |
| Prueba | 0.1189 | 95.71% | 490 | 469 | 21 |

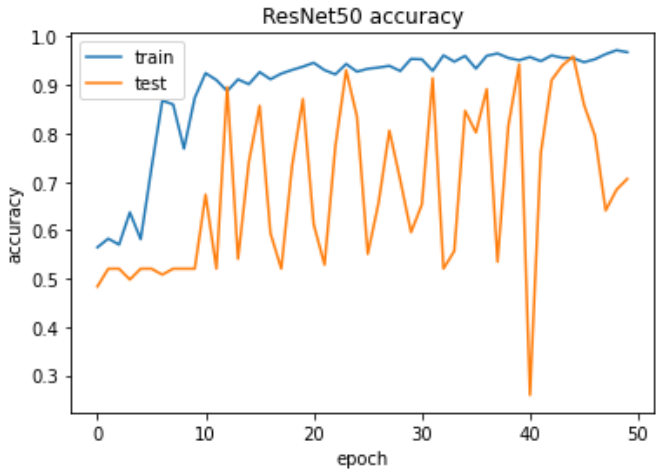
*Nota*: En la tabla se sintetizan los resultados obtenidos en las 3 fases y el número total de imágenes evaluadas.

En la figura 21 se evidencia que la red neuronal mantiene una tendencia creciente a lo largo del entrenamiento en el dataset de entrenamiento. Sin embargo, se observa una alta variación de la métrica de precisión en el conjunto de datos de validación. Lo cual hace que cobre relevancia el early stopping en tiempo de entrenamiento ya que la métrica de detención está basada en el dataset de validación.

Una de las razones por la cual la red neuronal no logra estabilizarse con el conjunto de datos de validación es porque el dataset no es lo suficientemente grande, lo que no permite que se generalice las características de los aguacates aptos y pequeñas variaciones de los parámetros de entrenamiento pueden tener una gran repercusión en la métrica de precisión.

Además, se observa que la función de normalización por lotes cumple el objetivo de regularizar la red no solo reduciendo el overfitting sino acelerando el tiempo de entrenamiento de la red neuronal.

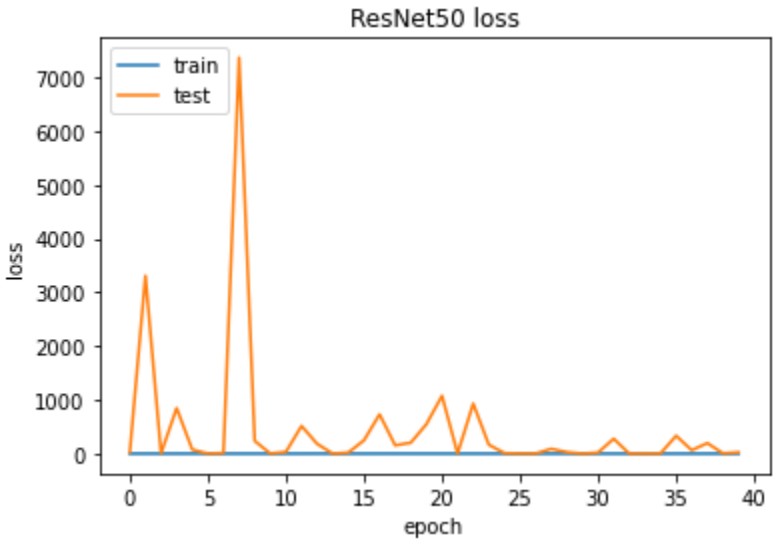
***Figura 211***. *Resultados de la precisión de la ResNet50 para el conjunto de datos de entrenamiento y validación.*




*Nota*: En la figura se muestra la precisión de la red en la fase de entrenamiento (línea azul) y validación (línea naranja) respecto al número de épocas.

En la figura 22 se aprecia en los primeros 20 ciclos de entrenamiento picos en el valor del gradiente (loss) en el dataset de validación lo que es congruente con los valores obtenidos en la gráfica de precisión. Sin embargo, se aprecia que hay una estabilización después de dicha época mostrando que no hay una estrecha relación entre los valores del optimizador y la precisión, lo que manifiesta la dificultad del conjunto de datos para ser clasificado correctamente. Aunque esto también se atribuye a la pequeña muestra de datos que se tiene.

***Figura 222**. Resultados de la pérdida (loss) de la ResNet50 para el conjunto de datos de entrenamiento y validación.*




*Nota*: En la figura 22 solamente se tienen en cuenta las últimas 40 época, las primeras 10 se consideran warm up y se pierde la dimensión de la mejora en los ciclos posteriores.

## 8.4 Resultados Inception Network

Los resultados de la Inception network no alcanzan a sobrepasar el 90% de precisión. Sin embargo, tiene en la validación y en el test un porcentaje de precisión cercano al 82%.

En la tabla 8 se ve una diferencia significativa entre la fase de entrenamiento y las otras dos fases de validación y prueba. Donde la fase de entrenamiento presenta un alto valor en el loss y una baja precisión con respecto a la validación y el conjunto de datos de prueba. Esto puede ser consecuencia que en el conjunto de imágenes de entrenamiento se encuentran algunas imágenes que no son aguacates precisamente para incrementar la rigurosidad del dataset. Pero puede castigar los resultados en tiempo de entrenamiento.

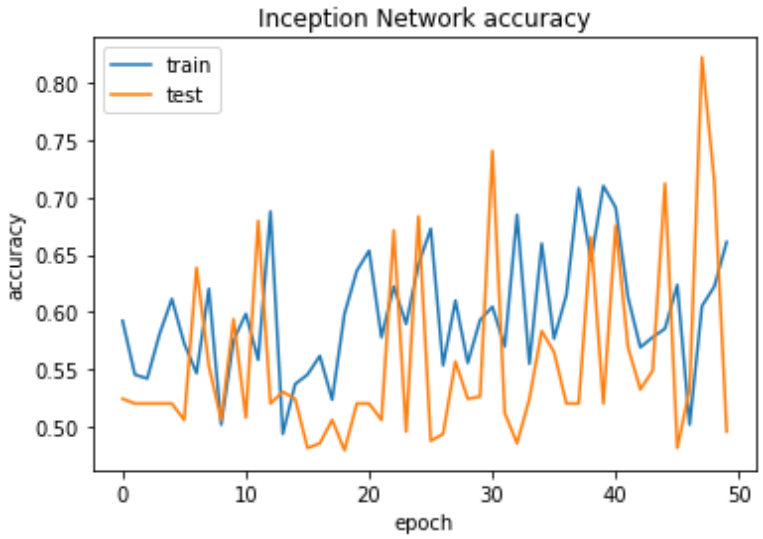
***Tabla 8**. Resultados de la Inception network con los dataset de entrenamiento, validación y prueba*

| FASE | PÉRDIDA | PRECISIÓN | TOTAL IMÁGENES | IMÁGENES EVALUADAS CORRECTAMENTE | IMÁGENES EVALUADAS ERRONAMENTE |
|---|---|---|---|---|---|
| Entrenamiento | 3.2679 | 0.6054 | 1470 | 890 | 580 |
| Validación | 0.4633 | 0.8224 | 490 | 403 | 87 |
| Prueba | 0.4227 | 0.8245 | 490 | 404 | 86 |

*Nota*: En la tabla se sintetizan los resultados obtenidos en las 3 fases y el número total de imágenes evaluadas.

La figura 23 muestra una irregularidad en la métrica de precisión donde no se logra una convergencia en el dataset de entrenamiento, ni en el conjunto de datos de validación. Esto es producto de la pequeña muestra de imágenes y las pocas épocas de entrenamiento que se utilizaron para entrenar la red neuronal.

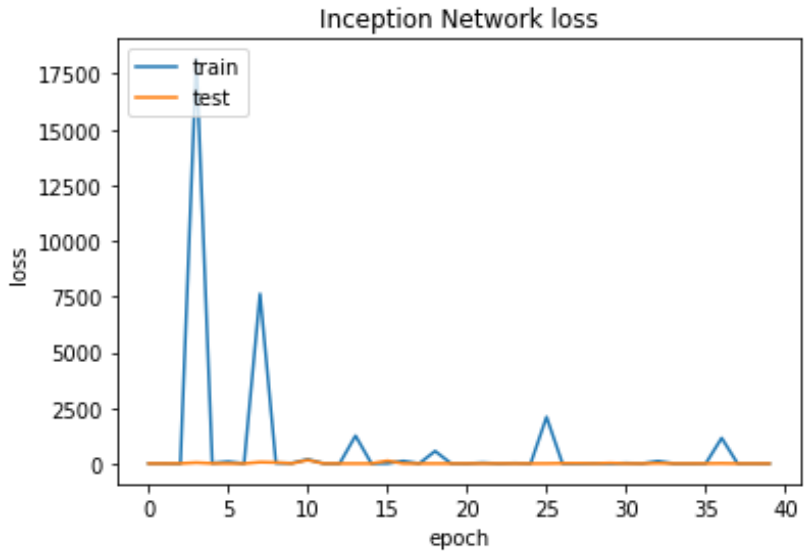
***Figura 233**. Resultados de la precisión de la Inception network para el conjunto de datos de entrenamiento y validación.*




*Nota*: En la figura se muestra la precisión de la red en la fase de entrenamiento (línea azul) y validación (línea naranja) respecto al número de épocas.

Al revisar la figura 24 se observan algunos picos en los valores de pérdida (loss) del dataset de entrenamiento a través de las diferentes épocas. Sin embargo, se observa que la pérdida en el dataset de validación permanece muy cercana a 0. Al comparar las gráficas de precisión y la de pérdida se aprecia una sustancial diferencia en la convergencia. Lo que significa que la red neuronal no es capaz de abstraer todas las características para diferenciar un aguacate apto de uno no conforme. Sin embargo, pese a las limitaciones de la red neuronal y del pequeño dataset se logra con early stopping guardar el modelo con los hiper parámetros entrenados con mayor precisión en la validación logrando resultados sobresalientes en el dataset de prueba.

***Figura 244.*** *Resultados de la pérdida (loss) de la Inception network para el conjunto de datos de entrenamiento y validación.*




*Nota*: En la figura solamente se tienen en cuenta las últimas 40 época, las primeras 10 se consideran warm up y se pierde la dimensión de la mejora en los ciclos posteriores.

## 8.5 Análisis general de los resultados

Se logra apreciar una gran diferencia entre el resultado de las redes neuronales profundas y no profundas, en las primeras se aprecia una imposibilidad de la red para aprender características que diferencian los aguacates aptos de los no aptos, lo que las hace menos sensibles y perceptivas a este tipo de redes. Sin embargo, se logra apreciar una convergencia rápida de este tipo de estructuras, aunque esto es causado por la pérdida de capacidad de aprendizaje.

Por otro lado, las redes neuronales profundas utilizadas presentan grandes diferencias a nivel de convergencia. Sin embargo, las dos tienen un comportamiento sobresaliente con respecto al dataset de prueba, pero solamente una de ellas logró sobrepasar el 90% de precisión que es la métrica establecida en los objetivos de este trabajo.

El comportamiento y tiempo de ejecución entre la resnet50 y la inception es realmente significativo. mientras la resnet50 utiliza aproximadamente 220 millones de parámetros que se pueden entrenar, la inception utiliza solamente alrededor de 2 millones de parámetros de entrenamiento lo que marca una diferencia sustancial en el tiempo de entrenamiento y recursos de máquina utilizados entre las dos redes.

Las estructuras de aprendizaje profundo claramente se vieron afectadas por el pequeño número de imágenes en el dataset, aun así, las dos estructuras sacaron provecho de la técnica de detención temprana ya que los dos tipos de redes no presentan una estabilización en la métrica de precisión.

Con respecto a la función de pérdida se ven dos diferencias marcadas en la inception network y en la resnet50. Por un lado, la resnet50 presenta variaciones en el conjunto de imágenes de

validación y los valores del loss en el entramiento se mantienen estables. Mientras que los valores en la inception se mantienen estables en el conjunto de imágenes de validación y tiene pequeñas variaciones en el entrenamiento.

En las gráficas de precisión se logra apreciar una clara convergencia de la resnet50 en el conjunto de datos de entrenamiento, pero hay grandes variaciones en el dataset de validación. Por otro lado, la red neuronal inception presenta una gran inestabilidad en los dos conjuntos de datos en el total de épocas entrenadas.

Por todo lo anterior, se dice que cada estructura abstrae diferentes características de los aguacates que definen la clasificación y por ello hay tantas variaciones en los datos de convergencia y precisión. Sin embargo, se logra evidenciar una gran capacidad de abstracción y precisión en las redes neuronales profundas y se logra el objetivo de entrenar al menos una red neuronal con una precisión por encima del 90%.

# 9. Conclusiones

En la presente investigación se aplicaron 3 tipos diferentes de redes neuronales convolucionales de aprendizaje supervisado para identificar el cumplimiento de requisitos mínimos de unidades de aguacate Hass para exportación, obteniéndose las siguientes conclusiones:

- La red Neuronal simple, aun cuando converge rápidamente, obtuvo un 52% de precisión en el proceso de inspección ya que pierde su capacidad de aprendizaje a partir de la época 10. Lo que denota que no es una opción viable para el proceso evaluado.

- La Inception Network de Google presenta resultados sobresalientes en la fase de validación y prueba, con un 82% de precisión. Sin embargo, en la fase de entrenamiento al utilizarse detención temprana (Early Stopping) con base en las métricas de la validación se obtuvo, un bajo rendimiento del 60% aproximadamente. Aun así se le logra apreciar que con un ajuste de hiper parámetros, se podría mejorar sustancialmente la precisión de la red.

- La ResNet 50 obtuvo resultados sobresalientes, con un promedio de precisión de 95% en las tres fases. Sin embargo, al igual que la Inception Network de Google, se hizo necesario el uso de detención temprana, como técnica de regularización, debido a que la Red Neuronal presentó una alta inestabilidad en los ciclos de entrenamiento con los datos de validación.

- Se evidencia con la aplicación que la Red Neuronal residual de 50 capas (ResNet50) es la que mayor porcentaje de precisión obtuvo, por encima del 90%. Permitiendo considerar que la aplicabilidad de redes en el proceso de inspección de aguacate Hass es viable y posible.

- En la implementación se hizo necesario recurrir a la técnica de early stopping, dado que ninguna de las arquitecturas validadas logró converger, producto del tamaño del Dataset; el número de datos fue demasiado pequeño. Para este tipo de estructuras se deben usar grandes volúmenes de datos.

- El sistema inteligente utilizado para la clasificación de fruta, evidencia que aun cuando no existió variación de hiper-parámetros como técnica de mejora en los resultados, en cuanto a precisión es equiparable con el sistema implementado en (M. Rodriguez et al., 2021) donde la técnica de aumento de datos es clave para una precisión por encima del 95%. En cuanto al número de capas implementado se evidencia que con respecto a (Cruz De La Cruz & Vera, 2020) donde el log fue del 60% con un menor número de capas, la red de 50 capas implementada en el presente trabajo tiene un mejor desempeño.

- La exportación de aguacate Hass requiere del cumplimiento de estándares de calidad rigurosos en la carga. Por lo cual se hace necesario que proceso de inspección de calidad de aguacate Hass en planta sea lo más eficiente posible en la detección de conformidades. Con la presente investigación, se logra evidenciar que las redes neuronales- específicamente la ResNet50 tienen un desempeño aceptable en el proceso de clasificación por encima del 95%. Lo que significa que la adecuación de este tipo de tecnologías - redes neuronales convolucionales profundas - en el proceso de inspección de calidad de aguacate Hass para exportación son una alternativa viable y altamente posible. Qué puede significar la integración de una inteligencia artificial con un proceso indispensable- proceso de clasificación- en el mundo del aguacate Hass para exportación.

Para próximos trabajos se debe tener en cuenta el uso de un mayor volumen de datos, el ajuste de los hiper parámetros y probar arquitecturas hibridas como Inception Network V3 y una ResNet con un mayor número de capas. Adicional, entrenar las redes con mayor número de épocas, todo esto con el fin de incrementar la precisión de las redes neuronales y garantizar la convergencia en cada conjunto de datos. Y así, poder validar en productivo, mediante la implementación de Hardtware y Software en plantas exportadoras.

# 10. Referencias


Aggarwal, C. C. (2018). Neural Networks and Deep Learning. In *Machine Learning*. Springer. https://doi.org/10.7551/mitpress/13811.003.0007

Agustina, D. (2018). *Aprendizaje y Análisis de RedesNeuronales Artificiales Profundas*.

Amar Rodrigo, J. (2021). *Redes neuronales con Python*. Ciencia de Datos. https://www.cienciadedatos.net/documentos/py35-redes-neuronales-python.html

Arias, F., Montoya, C., & Velásquez, O. (2018). Dinámica del mercado mundial de aguacate. *Revista Virtual Universidad Católica Del Norte*, *55*, 22–35. https://doi.org/10.35575/rvucn.n55a2

Ashok, V., & Vinod, D. S. (2016). A comparative study of feature extraction methods in defect classification of mangoes using neural network. *Proceedings - 2016 2nd International Conference on Cognitive Computing and Information Processing, CCIP 2016*, 7802873. https://doi.org/10.1109/CCIP.2016.7802873

Azarmdel, H., Jahanbakhshi, A., Mohtasebi, S. S., & Muñoz, A. R. (2020). Evaluation of image processing technique as an expert system in mulberry fruit grading based on ripeness level using artificial neural networks (ANNs) and support vector machine (SVM). *Postharvest Biology and Technology*, *166*, 111201. https://doi.org/10.1016/j.postharvbio.2020.111201

Bolívar, R., & Castro, C. (2019). Comportamiento del mercado internacional sobre consumo de aguacate. *Revista Observatorio de La Economía Latinoamericana*. https://www.eumed.net/rev/oel/2019/04/mercado-internacional-aguacate.html

Brownlee, J. (2019). *A Gentle Introduction to Batch Normalization for Deep Neural Networks*. Deep Learning Performance. https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/%0Ahttps://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/#:~:text=Batch normalization is a technique,required to train deep n

Cruz De La Cruz, J. E., & Vera, O. (2020). Convolutional neural networks for the Hass avocado classification using LabVIEW in an agro-industrial plant. *International Conference on Electronics, Electrical Engineering and Computing, INTERCON 2020*, 9–12. https://doi.org/10.1109/INTERCON50315.2020.9220246

Díaz, O., Rodríguez-Ruiz, A., Gubern-Mérida, A., Martí, R., & Chevalier, M. (2021). Are artificial intelligence systems useful in breast cancer screening programs? *Radiologia*, *63*(3), 236–244. https://doi.org/10.1016/j.rx.2020.11.006

Dolling, O. R., & Varas, E. A. (2002). Artificial neural networks for streamflow prediction. *Journal of Hydraulic Research*, *40*(5), 547–554. https://doi.org/10.1080/00221680209499899

Espinoza, J., & BBVA. (2020). ▷¿Qué es el Machine Learning? | Aprendizaje Automático | UCM. *BBVA*. https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/

Fiona, M. R., Thomas, S., Maria, I. J., & Hannah, B. (2019). Identification of ripe and unripe citrus fruits using artificial neural network. *Journal of Physics: Conference Series*, *1362*(1). https://doi.org/10.1088/1742-6596/1362/1/012033

Galindo, E. A., Perdomo, J. A., & Figueroa-García, J. C. (2020). Estudio comparativo entre

máquinas de soporte vectorial multiclase, redes neuronales artificiales y sistema de inferencia neuro-difuso auto organizado para problemas de clasificación. *Información Tecnológica*, *31*(1), 273–286. https://doi.org/10.4067/s0718-07642020000100273

Géron, A. (2019). Hands-on Machine Learning whith Scikit-Learing, Keras and Tensorfow. In *O'Reilly Media, Inc*.

Glorot, X., & Bengio, Y. (2010). *Understanding the difficulty of training deep feedforward neural networks* (Vol. 9, pp. 249–256).

González Cuello, R., Pérez Mendoza, J., & Gelvez Ordóñez, V. (2017). Incremento en la vida útil post cosecha del aguacate (Persea americana) utilizando recubrimientos a base de goma gelana. *Revista U.D.C.A Actualidad & Divulgación Científica*, *20*(1), 101–110. https://doi.org/10.31910/rudca.v20.n1.2017.67

González, Fabio. (2015). Modelos de aprendizaje computacional en reumatología TT - Machine learning models in rheumatology. *Rev. Colomb. Reumatol*, *22*(2), 77–78.

González, Francisco, & Romero, F. D. J. (2018). Comparativa de algoritmos bioinspirados para la optimización de redes neuronales. *Jovenes En La Ciencia*, *1*, 2850–2856.

Górriz, J. M., Ramírez, J., Ortíz, A., Martínez-Murcia, F. J., Segovia, F., Suckling, J., Leming, M., Zhang, Y. D., Álvarez-Sánchez, J. R., Bologna, G., Bonomini, P., Casado, F. E., Charte, D., Charte, F., Contreras, R., Cuesta-Infante, A., Duro, R. J., Fernández-Caballero, A., Fernández-Jover, E., … Ferrández, J. M. (2020). Artificial intelligence within the interplay between natural and artificial computation: Advances in data science, trends and applications. *Neurocomputing*, *410*(May), 237–270. https://doi.org/10.1016/j.neucom.2020.05.078

Grus, J. (2015). *Data Science from Scratch* (1st ed.).

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, *77*, 354–377. https://doi.org/10.1016/j.patcog.2017.10.013

Hakami, A., & Arif, M. (2019). Automatic Inspection of the External Quality of the Date Fruit. *Procedia Computer Science*, *163*, 70–77. https://doi.org/10.1016/j.procs.2019.12.088

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Heller, M. (2019). *What is Keras? The deep neural network API explained*. InfoWorld. https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html

Heng, L., & Weise, T. (2019). Intrusion Detection System Using Convolutional Neuronal Networks: A Cognitive Computing Approach for Anomaly Detection based on Deep Learning. *Proceedings of 2019 IEEE 18th International Conference on Cognitive Informatics and Cognitive Computing, ICCI*CC 2019*, 34–40. https://doi.org/10.1109/ICCICC46617.2019.9146088

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, *1*, 448–456.

J, V., & F, C. (2015). Vista de Predicción de caudales basados en Redes Neuronales Artificiales (RNA) para períodos de tiempo sub diarios. *Revista Politécnica*.

Joseph, S. (2016). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shif. *Journalism Practice*, *10*(6), 730–743. https://doi.org/10.1080/17512786.2015.1058180

Kaushik, A. (2018). *Understanding ResNet50 architecture*. *Ml*, 1. https://iq.opengenus.org/resnet50-architecture/

LeCun, Y. (2010). *Lenet-5 convolutional neural networks*. Yann.Lecun. http://yann.lecun.com/exdb/lenet/

Lu, M., & Li, F. (2020). *Survey on Lie Group Machine Learning*. *3*(4), 235–258. https://doi.org/10.26599/BDMA.2020.9020011

Mean, R., & Propagation, S. (2021). *Gradient Descent With RMSProp from Scratch*. https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch/

Ministerio de Agricultura y Desarrollo Rural. (2020). *Cadena Productiva Aguacate*. 33.

Momeny, M., Latif, A. M., Agha Sarram, M., Sheikhpour, R., & Zhang, Y. D. (2021). A noise robust convolutional neural network for image classification. *Results in Engineering*, *10*(February), 100225. https://doi.org/10.1016/j.rineng.2021.100225

Moreno, C. G. (2020). *¿Qué es el Deep Learning y para qué sirve?* Indra Company. https://www.indracompany.com/es/blogneo/deep-learning-sirve

Moreno Hincapié, G. A., Forero, M. G., Gómez, F., & Negrete, M. (2019). *Hass avocado classification by color and volume using a Kinect sensor*. 63. https://doi.org/10.1117/12.2529843

O, S. (2021). *An Introduction to Neural Network Loss Functions*. https://programmathically.com/an-introduction-to-neural-network-loss-functions/#:~:text=The loss function in a,all losses constitutes the cost

OMS, O. mundial de la salud, & FAO, O. de las naciones unidas para la alimentación y la agricultura. (2019). Programa Conjunto Fao/Oms Sobre Normas Alimentarias Comisión Del Codex Alimentarius. *Journal of Chemical Information and Modeling*, *53*(9), 1689–1699.

Orte, C. D. E. L. N., Ngenier, F. A. D. E. I., Del, T., Ema, T., On, D. E. U. S. O. Y. P., Norte, D. E. L., & Obra, D. E. L. A. (2019). *CLASIFICACION DE AGUACATES BASADO EN VISION POR COMPUTADOR*. UNIVERSIDAD TÉCNICA DEL NORTE.

Parra, D., & Serrano, S. (2017). Guía De Exportación De Aguacate Hass Para Pequeños Y Medianos Productores En Colombia. *Universidad Icesi*, 1–42. https://repository.icesi.edu.co/biblioteca_digital/bitstream/10906/82868/1/TG01730.pdf

Peña, W. E. L., & Minim, L. A. (2001). Aplicación De Redes Neuronales Artificiales En La Modelización Del Tratamiento Térmico De Alimentos Application of Neural Networks in the Modelling of the Thermal Treatment of Food Aplicación De Redes Neuronais Artificiais Na Modelización Do Tratamento Té. *Ciencia y Tecnologia Alimentaria*, *3*(2), 81–88. https://doi.org/10.1080/11358120109487650

Ramírez-Gil, J. G., Franco, G., & Henao-Rojas, J. C. (2019). Review of the concept of quality in Hass avocado and the pre-harvest and harvest factors that determine it under tropical

conditions. *Revista Colombiana de Ciencias Hortícolas*, *13*(3), 359–370. https://doi.org/10.17584/rcch.2019v13i3.10503

Ramírez-Gil, J. G., López, J. H., & Henao-Rojas, J. C. (2020). Causes of hass avocado fruit rejection in preharvest, harvest, and packinghouse: Economic losses and associated variables. *Agronomy*, *10*(1). https://doi.org/10.3390/agronomy10010008

Ramos, D. L., & García, L. A. (2019). Deep Learning for aspect extraction in textual opinions. *Revista Cubana de Ciencias Informáticas*, *13*(2), 105–145.

Rocio, C., Ramon, A., Mario, V., Waldo, O., & Abel, Q. (2011). Artificial neural networks in the estimation of reference evapotranspiration. *Revista Mexicana de Ciencias Agrícolas*, *2*(3), 433–447.

Rodriguez, K., & Giraldo, F. (2017). *Artificial neuronal networks for optical inspection in PCBquality control*. *11*(2), 169–178.

Rodriguez, M., Pastor, F., & Ugarte, W. (2021). Classification of fruit ripeness grades using a convolutional neural network and data augmentation. *Conference of Open Innovation Association, FRUCT*, *2021-Janua*. https://doi.org/10.23919/FRUCT50888.2021.9347597

Roy, P., Ghosh, S., Bhattacharya, S., & Pal, U. (2018). Effects of degradations on deep neural network architectures. *ArXiv*, 1–11.

Rumelhart D.E. (1986). *Parallel recognition in modem computers. In Proceeding: Explorations in the microestructure of Cognition* (Vol. 1, Issue November).

Sabzi, S., Abbaspour-Gilandeh, Y., & García-Mateos, G. (2018). A new approach for visual identification of orange varieties using neural networks and metaheuristic algorithms. *Information Processing in Agriculture*, *5*(1), 162–172. https://doi.org/10.1016/j.inpa.2017.09.002

Salazar, C. (2020). Modelo para la clasificación del aguacate Hass en sus estados comerciales y de exportación, mediante el uso de redes neuronales convolucionales. *Orphanet Journal of Rare Diseases*, *21*(1), 1–9.

Salman, H., Grover, J., & Shankar, T. (2018). *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review*. *2733*(March), 2709–2733. https://doi.org/10.1162/NECO

Shen, W., & Wang, W. (2018). Node identification in wireless network based on convolutional neural network. *Proceedings - 14th International Conference on Computational Intelligence and Security, CIS 2018*, 238–241. https://doi.org/10.1109/CIS2018.2018.00059

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Springer*, *6*(1). https://doi.org/10.1186/s40537-019-0197-0

Sibo, T., Hao, G., & Kai, Y. (2016). A Comparative Study Of Robustness Of Deep Learning Approaches For Vad. *Icassp*, 5695–5699. https://doi.org/978-1-4799-9988-0/16/

Song, W., Li, S., Fang, L., & Lu, T. (2019). Hyperspectral Image Classification with Deep Feature Fusion Network. *International Geoscience and Remote Sensing Symposium (IGARSS)*, *56*(6), 2714–2717. https://doi.org/10.1109/IGARSS.2019.8898520

Sordo, M., Vaidya, S., & Jain, L. (2008). *An Introduction to Computational Intelligence in Healthcare: New Directions* (Springer (ed.)).

Szegedy, C., Vanhoucke, V., Shlens, J., & Wojna, Z. (2014). *Rethinking the Inception Architecture for Computer Vision*.

TensorFlow. (2020). *Por qué TensorFlow*. TensoFlow. https://www.tensorflow.org

Vega, H. (2011). *Redes neuronales para el reconocimiento de la calidad morfológica de mangos exportables para la empresa Biofruit del Perú S.A.C.* 117.

Vitaly, B. (2018). *Understanding RMSprop — faster neural network learning*. Neurocomputing. https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a

Whitson, G. (2020). Artificial intelligence. In *ICECT 2011 - 2011 3rd International Conference on Electronics Computer Technology* (Vol. 4, pp. 124–129). Salem Press Encyclopedia of Science. https://doi.org/10.1109/ICECTECH.2011.5941871

Yegulalp, S. (2022). *¿Qué es TensorFlow? La biblioteca de aprendizaje automático explicada | InfoMundo*. https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

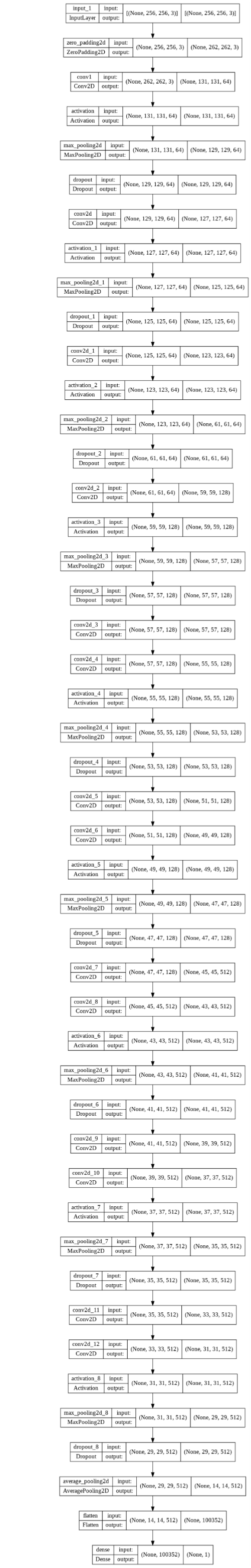
# 11. Anexos

Los anexos es mención en el presente documento se encuentran adjuntos al mismo.


| ANEXO | DESCRIPCIÓN |
|---|---|
| Anexo 1 | Capas de entrenamiento de la red neuronal CNN no profunda con algunos hiper-parámetros utilizados |
| Anexo 2 | Capas de entrenamiento de la red neuronal ResNet50 con algunos hiper parámetros utilizados |
| Anexo 3 | Capas de entrenamiento de la red neuronal Inception network con algunos hiper parámetros utilizados. |
| Anexo 4 | Código implementación CNN no profunda |
| Anexo 5 | Código implementación ResNet50 |
| Anexo 6 | Código implementación Inception Network |

# Anexo 1

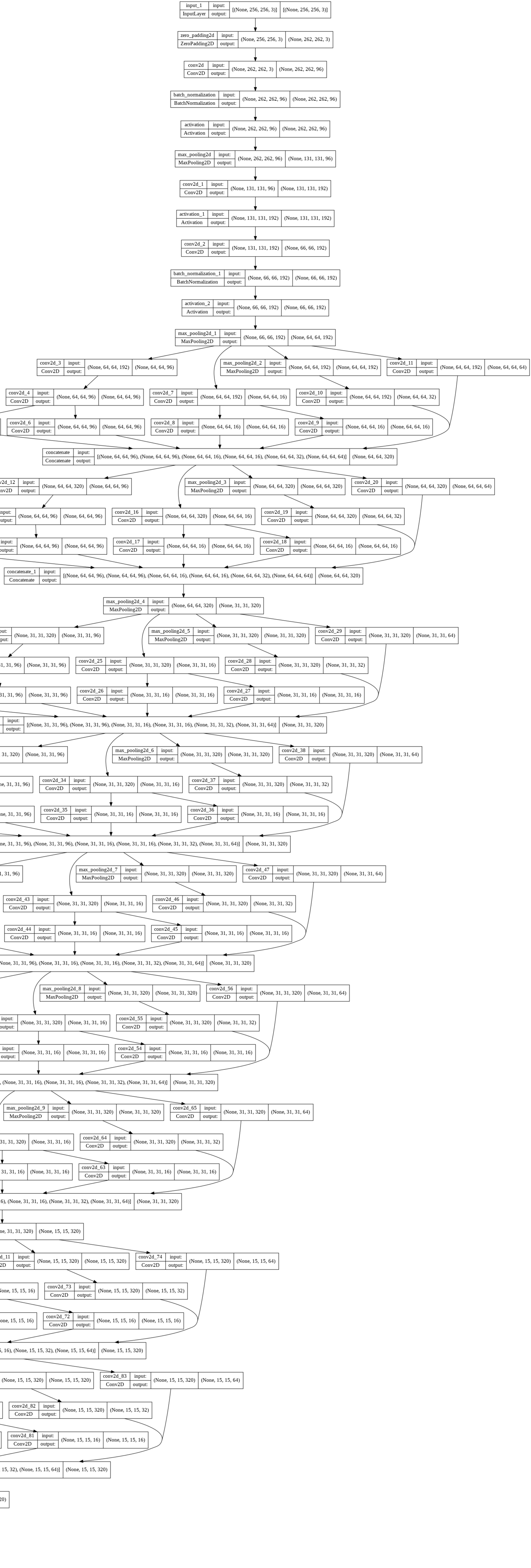
Capas de entrenamiento de la red neuronal CNN no profunda con algunos hiper-parámetros utilizados.



**Anexo 2**
Capas de entrenamiento de la red neuronal ResNet50 con algunos hiper parámetros utilizados.



**Anexo 3**
Capas de entrenamiento de la red neuronal Inception network con algunos hiper parámetros utilizados.



# Anexo 4.

Código implementación CNN no profunda

## Aplicación de Redes Neuronales en el proceso de inspección de calidad del aguacate Hass para exportación.

Red neuronal Convolucional simple

```
# Path to photos
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
# Importing the necessary libraries to train the neural network
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
from keras import layers
from keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Dr
from tensorflow.keras.optimizers import RMSprop
from keras.models import Model, load_model
from keras.preprocessing import image
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
import pydot
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.initializers import glorot_uniform
import scipy.misc
from keras.utils.vis_utils import plot_model
from sklearn.model_selection import train_test_split
import keras.backend as k
from keras.callbacks import ModelCheckpoint as model_checkpoint
import tensorflow.keras.losses as losses
%matplotlib inline
```

```
#=======================
train_cump = "/content/drive/MyDrive/Project/Images/Training/Cumple"
train_no_cump = "/content/drive/MyDrive/Project/Images/Training/No_Cumple"
test_cump = "/content/drive/MyDrive/Project/Images/Test/Cumple"
test_no_cump = "/content/drive/MyDrive/Project/Images/Test/No_Cumple"
dev_cump = "/content/drive/MyDrive/Project/Images/Validation/Cumple"
dev_no_cump = "/content/drive/MyDrive/Project/Images/Validation/No_Cumple"
```

```
print("Imagenes de entrenamiento totales de aguacates que cumplen con las características de
print("Imagenes de entrenamiento totales de aguacates que no cumplen con las características
print("Imagenes de prueba totales de aguacates que cumplen con las características de exporta
print("Imagenes de prueba totales de aguacates que no cumplen con las características de expo
print("Imagenes de validación totales de aguacates que cumplen con las características de exp
print("Imagenes de validación totales de aguacates que no cumplen con las características de
```

```
Imagenes de entrenamiento totales de aguacates que cumplen con las características de ex
Imagenes de entrenamiento totales de aguacates que no cumplen con las características de
Imagenes de prueba totales de aguacates que cumplen con las características de exportaci
Imagenes de prueba totales de aguacates que no cumplen con las características de export
Imagenes de validación totales de aguacates que cumplen con las características de expor
Imagenes de validación totales de aguacates que no cumplen con las características de ex
```

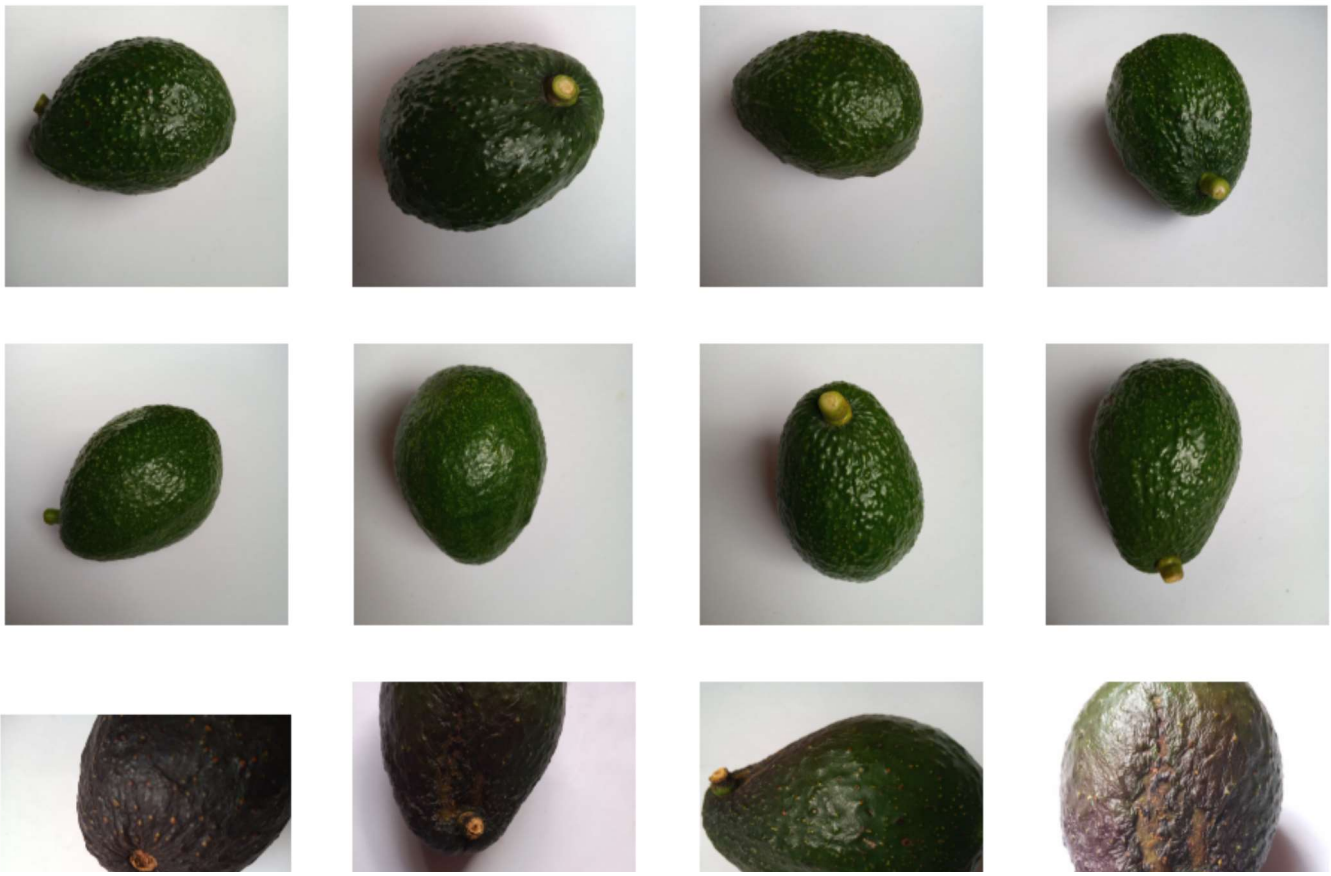
```
train_cumple_nombres =  os.listdir(train_cump)
train_no_cumple_nombres = os.listdir(train_no_cump)
validation_cumple = os.listdir(dev_cump)
validation_no_cumple = os.listdir(dev_no_cump)


nfilas = 4
ncolumnas = 4
indice_pic = 0
fig = plt.gcf()
fig.set_size_inches(ncolumnas*4, nfilas*4)
indice_pic += 8
aguacate_cumple = [os.path.join(train_cump,fnombre) for fnombre in train_cumple_nombres[indic
aguacate_no_cumple = [os.path.join(train_no_cump,fnombre) for fnombre in train_no_cumple_nomb

for i, img_path in enumerate(aguacate_cumple+aguacate_no_cumple):
  #subplot
  sp = plt.subplot(nfilas, ncolumnas, i+1)
  sp.axis("off")

  img = mpimg.imread(img_path)
  plt.imshow(img)
plt.show
```

```
<function matplotlib.pyplot.show>
```



```
def simple_CNN(input_shape = (256, 256, 3)):

    X_input = Input(input_shape)

    X = ZeroPadding2D((3, 3))(X_input)

    # Initial stage
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1', padding='same')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # stage 1
    X = Conv2D(64, (3,3), padding='valid')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # stage 2
    X = Conv2D(64, (3,3), padding = 'valid')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)
    X = Dropout(0.2)(X)

    # stage 3
    X = Conv2D(128, (3,3), padding = 'valid')(X)
```

```
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # stage 4
    X = Conv2D(128, (3,3), padding = 'same')(X)
    X = Conv2D(128, (3,3))(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # stage 4
    X = Conv2D(128, (3,3), padding = 'valid')(X)
    X = Conv2D(128, (3,3))(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # stage 5
    X = Conv2D(512, (3,3), padding = 'valid')(X)
    X = Conv2D(512, (3,3))(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # stage 6
    X = Conv2D(512, (3,3), padding = 'valid')(X)
    X = Conv2D(512, (3,3))(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # stage 7
    X = Conv2D(512, (3,3), padding = 'valid')(X)
    X = Conv2D(512, (3,3))(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(1, 1))(X)
    X = Dropout(0.2)(X)

    # AVGPOOL
    X = AveragePooling2D(pool_size=(2,2))(X)

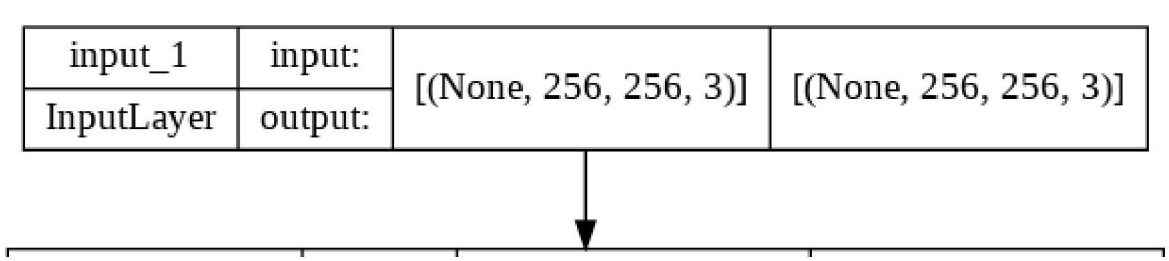
    # Output layer
    X = Flatten()(X)
    X = Dense(1, activation='sigmoid', kernel_initializer = glorot_uniform(seed=0))(X)

    # Create the model
    model = Model(inputs = X_input, outputs = X, name='CNN')

    return model
```

```
model = simple_CNN(input_shape = (256, 256, 3))


plot_model(model, to_file='model.png', show_shapes=True)
```




```
model.summary()
```

```
model.compile(loss = losses.BinaryCrossentropy(), optimizer = RMSprop(learning_rate= 0.001),
```

```
path_save = "/content/drive/MyDrive/Project/chekpoints/"
```

```
from keras.preprocessing.image import ImageDataGenerator

#Normalize the images to facilitate processing taking into account that they are color photos
train_datagen = ImageDataGenerator(rescale = 1/255)
validation_datagen = ImageDataGenerator(rescale = 1/255)
test_detagen = ImageDataGenerator(rescale= 1/255)

#Prepare training photos
train_generator = train_datagen.flow_from_directory(
    "/content/drive/MyDrive/Project/Images/Training",
    target_size = (256,256),
    batch_size = 32,
    class_mode = 'binary'
)

#Prepare the validation photos
validation_generator = validation_datagen.flow_from_directory(
    "/content/drive/MyDrive/Project/Images/Validation",
    target_size = (256,256),
    batch_size = 32,
    class_mode = 'binary'
)

#Prepare the test photos
test_generator = test_detagen.flow_from_directory(
    "/content/drive/MyDrive/Project/Images/Test",
    target_size = (256,256),
    class_mode = 'binary'
)
```

```
Found 1470 images belonging to 2 classes.
Found 490 images belonging to 2 classes.
Found 490 images belonging to 2 classes.
```

```
"""class myCallback(Callback):
  def on_epoch_end(self, epoch, logs ={}):
    if (logs.get('accuracy')>= 0.80):
      print("\n accuracy is higher, stops training")
      self.model.stop_training = True """

checkpoint = model_checkpoint(filepath = path_save + 'simple_CNN.h5', save_best_only=True, ve



history = model.fit(
      train_generator,
      callbacks = [checkpoint],
      steps_per_epoch=46,
      epochs=50,
      batch_size = 32,
      verbose=1,
      validation_data = validation_generator,
      validation_steps=16
      )
```

```
Epoch 1/50
46/46 [==============================] - ETA: 0s - loss: 5684.0879 - accuracy: 0.5027
Epoch 1: val_loss improved from inf to 0.69543, saving model to /content/drive/MyDriv
46/46 [==============================] - 1691s 36s/step - loss: 5684.0879 - accuracy:
Epoch 2/50
46/46 [==============================] - ETA: 0s - loss: 0.8939 - accuracy: 0.5116
Epoch 2: val_loss did not improve from 0.69543
46/46 [==============================] - 75s 2s/step - loss: 0.8939 - accuracy: 0.511
Epoch 3/50
46/46 [==============================] - ETA: 0s - loss: 1.9349 - accuracy: 0.5014
Epoch 3: val_loss improved from 0.69543 to 0.69233, saving model to /content/drive/My
46/46 [==============================] - 76s 2s/step - loss: 1.9349 - accuracy: 0.5014
Epoch 4/50
46/46 [==============================] - ETA: 0s - loss: 0.6950 - accuracy: 0.5007
Epoch 4: val_loss did not improve from 0.69233
46/46 [==============================] - 76s 2s/step - loss: 0.6950 - accuracy: 0.500
Epoch 5/50
46/46 [==============================] - ETA: 0s - loss: 4.5342 - accuracy: 0.5143
Epoch 5: val_loss did not improve from 0.69233
46/46 [==============================] - 75s 2s/step - loss: 4.5342 - accuracy: 0.514
Epoch 6/50
46/46 [==============================] - ETA: 0s - loss: 0.6944 - accuracy: 0.4850
Epoch 6: val_loss did not improve from 0.69233
46/46 [==============================] - 74s 2s/step - loss: 0.6944 - accuracy: 0.4850
Epoch 7/50
46/46 [==============================] - ETA: 0s - loss: 0.6930 - accuracy: 0.5190
Epoch 7: val_loss did not improve from 0.69233
46/46 [==============================] - 75s 2s/step - loss: 0.6930 - accuracy: 0.5190
Epoch 8/50
46/46 [==============================] - ETA: 0s - loss: 0.6928 - accuracy: 0.5190
Epoch 8: val_loss did not improve from 0.69233
46/46 [==============================] - 74s 2s/step - loss: 0.6928 - accuracy: 0.5190
```

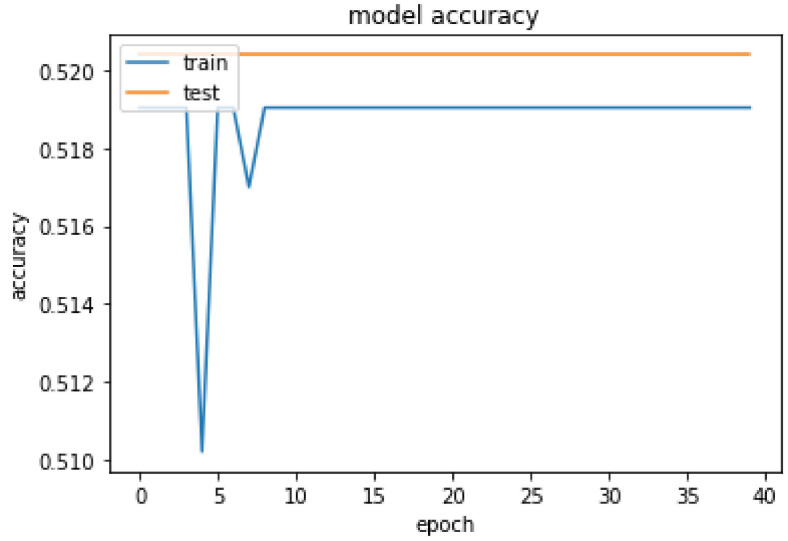
```
Epoch 9/50
46/46 [==============================] - ETA: 0s - loss: 0.6927 - accuracy: 0.5190
Epoch 9: val_loss did not improve from 0.69233
46/46 [==============================] - 75s 2s/step - loss: 0.6927 - accuracy: 0.5190
Epoch 10/50
46/46 [==============================] - ETA: 0s - loss: 0.6927 - accuracy: 0.5190
Epoch 10: val_loss did not improve from 0.69233
46/46 [==============================] - 74s 2s/step - loss: 0.6927 - accuracy: 0.5190
Epoch 11/50
46/46 [==============================] - ETA: 0s - loss: 0.6926 - accuracy: 0.5190
Epoch 11: val_loss did not improve from 0.69233
46/46 [==============================] - 75s 2s/step - loss: 0.6926 - accuracy: 0.5190
Epoch 12/50
46/46 [==============================] - ETA: 0s - loss: 0.6926 - accuracy: 0.5190
Epoch 12: val_loss did not improve from 0.69233
46/46 [==============================] - 74s 2s/step - loss: 0.6926 - accuracy: 0.5190
Epoch 13/50
46/46 [==============================] - ETA: 0s - loss: 0.6926 - accuracy: 0.5190
Epoch 13: val_loss did not improve from 0.69233
46/46 [==============================] - 75s 2s/step - loss: 0.6926 - accuracy: 0.5190
Epoch 14/50
46/46 [==============================] - ETA: 0s - loss: 0.6925 - accuracy: 0.5190
Epoch 14: val_loss did not improve from 0.69233
46/46 [==============================] - 74s 2s/step - loss: 0.6925 - accuracy: 0.5190
Epoch 15/50
```

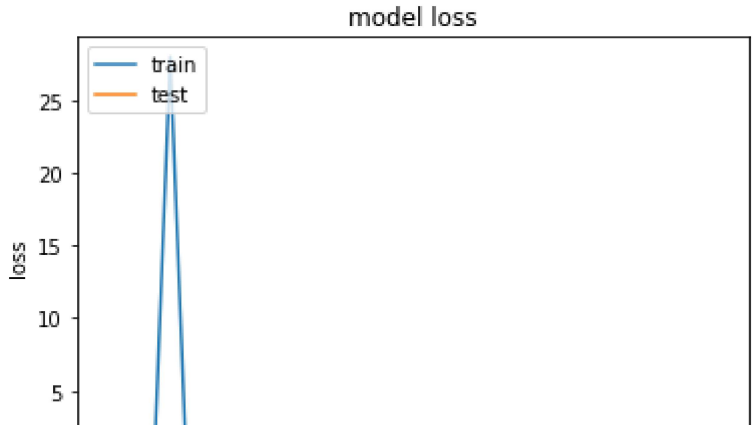
```
model = load_model(path_save + 'simple_CNN.h5')
```

```
from matplotlib import pyplot as plt
print(history.history.keys())

plt.plot(history.history['accuracy'][10:])
plt.plot(history.history['val_accuracy'][10:])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'][10:])
plt.plot(history.history['val_loss'][10:])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
evaluacion = model.evaluate(test_generator)
print ("Loss = " + str(evaluacion[0]))
print ("Test Accuracy = " + str(evaluacion[1]))
```

```
16/16 [==============================] - 406s 27s/step - loss: 0.6923 - accuracy: 0.520
Loss = 0.6923186182975769
Test Accuracy = 0.5204081535339355
```

He,K Zhang,X Ren,S, Sun,J (2015) *Deep Residual Learning for Image Recognition*

Estados Unidos: Microsoft Reseacrh

```
import os, signal
os.kill(os.getpid(), signal.SIGKILL)
```



**Anexo 5.**
Código implementación ResNet50

# Aplicación de Redes Neuronales en el proceso de inspección de calidad del aguacate Hass para exportación.

## Red neuronal Residual de 50 capas (ResNet50)

```
# Path to photos
from google.colab import drive
drive.mount('/content/drive/')
```

```
     Mounted at /content/drive/
```

```
# Importing the necessary libraries to train the neural network
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.utils.vis_utils import plot_model
import numpy as np
from keras import layers
from keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Fl
from tensorflow.keras.optimizers import RMSprop
from keras.models import Model, load_model
from keras.preprocessing import image
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
import pydot
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.initializers import glorot_uniform
import scipy.misc
from sklearn.model_selection import train_test_split
import keras.backend as k
from keras.callbacks import ModelCheckpoint as model_checkpoint
import tensorflow.keras.losses as losses
%matplotlib inline
```

```
#=======================
train_cump = "/content/drive/MyDrive/Project/Images/Training/Cumple"
train_no_cump = "/content/drive/MyDrive/Project/Images/Training/No_Cumple"
test_cump = "/content/drive/MyDrive/Project/Images/Test/Cumple"
test_no_cump = "/content/drive/MyDrive/Project/Images/Test/No_Cumple"
dev_cump = "/content/drive/MyDrive/Project/Images/Validation/Cumple"
dev_no_cump = "/content/drive/MyDrive/Project/Images/Validation/No_Cumple"
```

```
print("Imagenes de entrenamiento totales de aguacates que cumplen con las características de
print("Imagenes de entrenamiento totales de aguacates que no cumplen con las características
print("Imagenes de prueba totales de aguacates que cumplen con las características de exporta
print("Imagenes de prueba totales de aguacates que no cumplen con las características de expo
print("Imagenes de validación totales de aguacates que cumplen con las características de exp
print("Imagenes de validación totales de aguacates que no cumplen con las características de
```

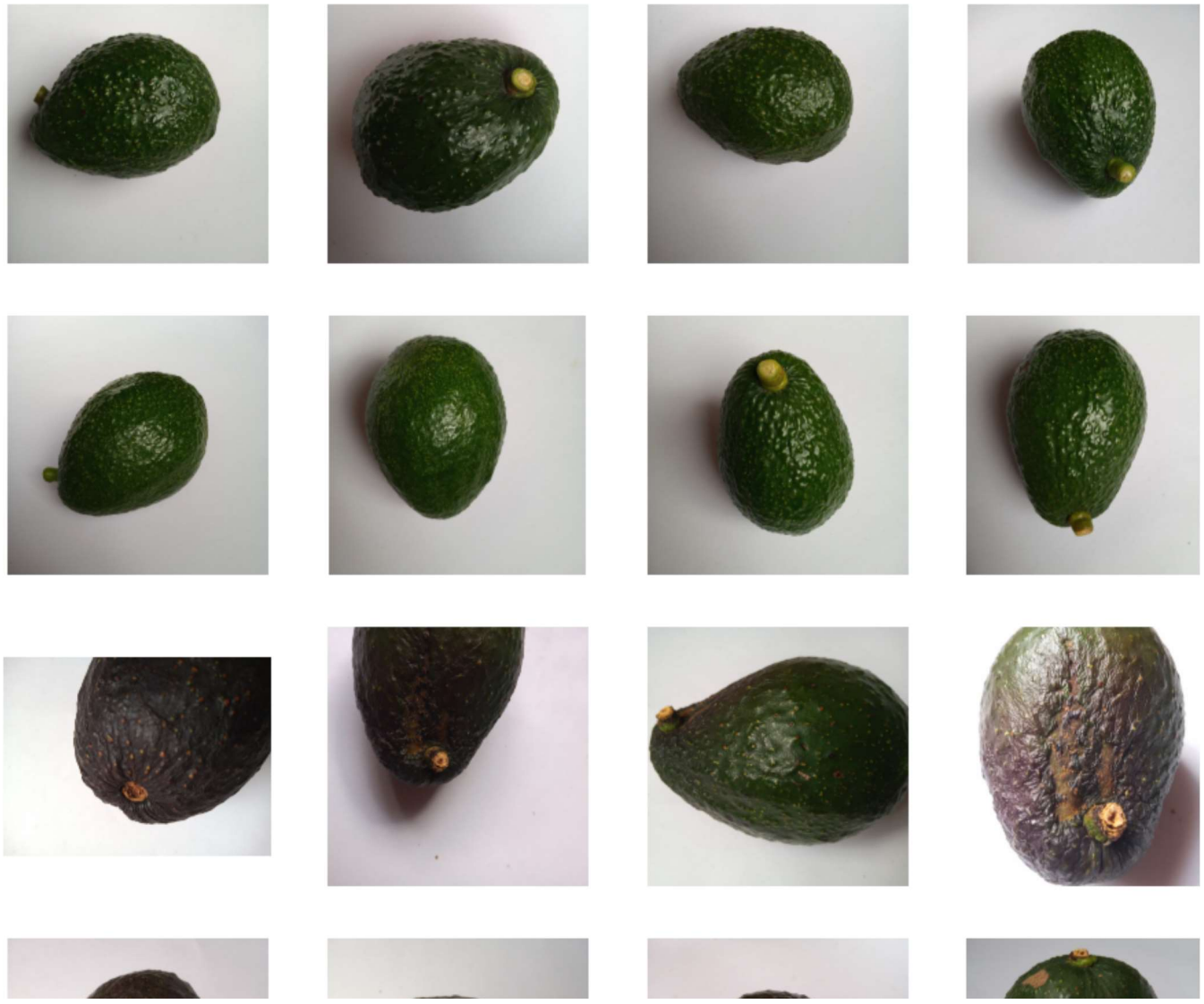
```
Imagenes de entrenamiento totales de aguacates que cumplen con las características de ex
Imagenes de entrenamiento totales de aguacates que no cumplen con las características de
Imagenes de prueba totales de aguacates que cumplen con las características de exportaci
Imagenes de prueba totales de aguacates que no cumplen con las características de export
Imagenes de validación totales de aguacates que cumplen con las características de expor
Imagenes de validación totales de aguacates que no cumplen con las características de ex
```

```
nfilas = 4
ncolumnas = 4
indice_pic = 0
fig = plt.gcf()
fig.set_size_inches(ncolumnas*4, nfilas*4)
indice_pic += 8
aguacate_cumple = [os.path.join(train_cump,fnombre) for fnombre in train_cumple_nombres[indic
aguacate_no_cumple = [os.path.join(train_no_cump,fnombre) for fnombre in train_no_cumple_nomb

for i, img_path in enumerate(aguacate_cumple+aguacate_no_cumple):
  #subplot
  sp = plt.subplot(nfilas, ncolumnas, i+1)
  sp.axis("off")

  img = mpimg.imread(img_path)
  plt.imshow(img)
plt.show
```

```
<function matplotlib.pyplot.show>
```



```
def bloque_identidad(X, f, filtros, etapa, bloque):

    # Define the basic names
    conv_name_base = 'res' + str(etapa) + bloque + '_rama'
    bn_name_base = 'bn' + str(etapa) + bloque + '_rama'
    # Receiving the filters
    f1, f2, f3 = filtros
    # Set the input block to be used later in the union with the other convolutional block
    salto = X

    # First layer or block
    X = Conv2D(filters = f1, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name =
    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    # Second layer
    X = Conv2D(filters=f2,kernel_size=(f,f),strides=(1,1),padding='same',name=conv_name_base
    X = BatchNormalization(axis=3,name=bn_name_base +'2b')(X)
    X = Activation('relu')(X)
```

```
    # Third layer
    X = Conv2D(filters=f3,kernel_size=(1,1),strides=(1,1),padding='valid', name=conv_name_bas
    X = BatchNormalization(axis=3, name = bn_name_base +'2c')(X)

    # Final bonding layer
    X = Add()([salto,X])
    X = Activation('relu')(X)

    return X


def bloque_de_convolucion(X, f, filtros, etapa, bloque, s = 2):

    # Define the basic names
    conv_name_base = 'res' + str(etapa) + bloque + '_branch'
    bn_name_base = 'bn' + str(etapa) + bloque + '_branch'
    # Receiving the filters
    f1, f2, f3 = filtros
    # Save the input value
    salto = X

    # First component of main path
    X = Conv2D(f1, (1, 1), strides = (s, s), padding = 'valid', name = conv_name_base + '2a',
    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    # Second component of main path (≈3 lines)
    X = Conv2D(f2, (f, f), strides = (1, 1), padding = 'same', name = conv_name_base + '2b',
    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    # Third component of main path (≈2 lines)
    X = Conv2D(f3, (1, 1), strides = (1, 1), padding = 'valid', name = conv_name_base + '2c',
    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

    # SHORTCUT PATH #
    salto = Conv2D(f3, (1, 1), strides = (s, s), padding = 'valid', name = conv_name_base + '
    salto = BatchNormalization(axis = 3, name = bn_name_base + '1')(salto)

    # Final step: Add shortcut value to main path, and pass it through a RELU activation (≈2
    X = Add()([X, salto])
    X = Activation('relu')(X)

    return X


def ResNet50(input_shape = (256, 256, 3)):

    X_input = Input(input_shape)

    X = ZeroPadding2D((3, 3))(X_input)
```

```
    # Initial stage
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1', kernel_initializer = glorot_unif
    X = BatchNormalization(axis = 3, name = 'bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # stage 1
    X = bloque_de_convolucion(X, f = 3, filtros = [256, 256, 256], etapa = 2, bloque='a', s =
    X = bloque_identidad(X, 3, [256, 256, 256], etapa=2, bloque='b')
    X = bloque_identidad(X, 3, [256, 256, 256], etapa=2, bloque='c')

    # stage 2
    X = bloque_de_convolucion(X,f=3,filtros=[512,512,256],etapa=3,bloque='a',s = 2)
    X = bloque_identidad(X,3,[512,512,256],etapa=3,bloque='b')
    X = bloque_identidad(X,3,[512,512,256],etapa=3,bloque='c')
    X = bloque_identidad(X,3,[512,512,256],etapa=3,bloque='d')

    # stage 3
    X = bloque_de_convolucion(X,f=3,filtros=[1024,1024,1024],etapa=4,bloque='a',s=2)
    X = bloque_identidad(X,3,[1024,1024,1024],etapa=4,bloque='b')
    X = bloque_identidad(X,3,[1024,1024,1024],etapa=4,bloque='c')
    X = bloque_identidad(X,3,[1024,1024,1024],etapa=4,bloque='d')
    X = bloque_identidad(X,3,[1024,1024,1024],etapa=4,bloque='e')
    X = bloque_identidad(X,3,[1024,1024,1024],etapa=4,bloque='f')

    # stage 4
    X = bloque_de_convolucion(X,f=3,filtros=[2048,2048,2048],etapa=5,bloque='a',s=2)
    X = bloque_identidad(X,3,[2048,2048,2048],etapa=5,bloque='b')
    X = bloque_identidad(X,3,[2048,2048,2048],etapa=5,bloque='c')

    # AVGPOOL
    X = AveragePooling2D(pool_size=(2,2))(X)

    # Output layer
    X = Flatten()(X)
    X = Dense(1, activation='sigmoid', kernel_initializer = glorot_uniform(seed=0))(X)

    # Create the model
    model = Model(inputs = X_input, outputs = X, name='ResNet50')

    return model


model = ResNet50(input_shape = (256, 256, 3))


model.compile(loss = losses.BinaryCrossentropy(), optimizer = RMSprop(learning_rate= 0.001),


path_save = "/content/drive/MyDrive/chekpoints/"


from keras.preprocessing.image import ImageDataGenerator
```

```
#Normalize the images to facilitate processing taking into account that they are color photos
train_datagen = ImageDataGenerator(rescale = 1/255)
validation_datagen = ImageDataGenerator(rescale = 1/255)
test_detagen = ImageDataGenerator(rescale= 1/255)

#Prepare training photos
train_generator = train_datagen.flow_from_directory(
    "/content/drive/MyDrive/Aguacates/Training",
    target_size = (256,256),
    batch_size = 32,
    class_mode = 'binary'
)

#Prepare the validation photos
validation_generator = validation_datagen.flow_from_directory(
    "/content/drive/MyDrive/Aguacates/Validation",
    target_size = (256,256),
    batch_size = 32,
    class_mode = 'binary'
)

#Prepare the test photos
test_generator = test_detagen.flow_from_directory(
    "/content/drive/MyDrive/Aguacates/Test",
    target_size = (256,256),
    class_mode = 'binary'
)
```

```
Found 1470 images belonging to 2 classes.
Found 490 images belonging to 2 classes.
Found 490 images belonging to 2 classes.
```

```
"""class myCallback(Callback):
  def on_epoch_end(self, epoch, logs ={}):
    if (logs.get('accuracy')>= 0.80):
      print("\n accuracy is higher, stops training")
      self.model.stop_training = True """
```

```
'class myCallback(Callback):\n  def on_epoch_end(self, epoch, logs ={}):\n    if (logs.g
t(\'accuracy\')>= 0.80):\n      print("\n accuracy is higher, stops training")\n      se
f.model.stop_training = True '
```

```
checkpoint = model_checkpoint(filepath = path_save + 'ResNet50_aguacates.h5', save_best_only=
```

```
history = model.fit(
      train_generator,
      callbacks = [checkpoint],
      steps_per_epoch=46,
```

```
 epochs=50,
 batch_size = 32,
 verbose=1,
 validation_data = validation_generator,
 validation_steps=16
 )
Epoch 00036: val_loss did not improve from 0.27217
Epoch 37/50
46/46 [==============================] - 99s 2s/step - loss: 0.1057 - accuracy: 0.960

Epoch 00037: val_loss did not improve from 0.27217
Epoch 38/50
46/46 [==============================] - 98s 2s/step - loss: 0.1059 - accuracy: 0.965

Epoch 00038: val_loss did not improve from 0.27217
Epoch 39/50
46/46 [==============================] - 99s 2s/step - loss: 0.1225 - accuracy: 0.956

Epoch 00039: val_loss did not improve from 0.27217
Epoch 40/50
46/46 [==============================] - 98s 2s/step - loss: 0.1323 - accuracy: 0.951


Epoch 00040: val_loss improved from 0.27217 to 0.16192, saving model to /content/driv
Epoch 41/50
46/46 [==============================] - 101s 2s/step - loss: 0.1063 - accuracy: 0.95

Epoch 00041: val_loss did not improve from 0.16192
Epoch 42/50
46/46 [==============================] - 99s 2s/step - loss: 0.1195 - accuracy: 0.949

Epoch 00042: val_loss did not improve from 0.16192
Epoch 43/50
46/46 [==============================] - 98s 2s/step - loss: 0.1329 - accuracy: 0.961

Epoch 00043: val_loss did not improve from 0.16192
Epoch 44/50
46/46 [==============================] - 99s 2s/step - loss: 0.1090 - accuracy: 0.956

Epoch 00044: val_loss did not improve from 0.16192
Epoch 45/50
46/46 [==============================] - 100s 2s/step - loss: 0.1144 - accuracy: 0.95

Epoch 00045: val_loss did not improve from 0.16192
Epoch 46/50
46/46 [==============================] - 98s 2s/step - loss: 0.1361 - accuracy: 0.947

Epoch 00046: val_loss did not improve from 0.16192
Epoch 47/50
46/46 [==============================] - 99s 2s/step - loss: 0.0975 - accuracy: 0.953

Epoch 00047: val_loss did not improve from 0.16192
Epoch 48/50
46/46 [==============================] - 99s 2s/step - loss: 0.0989 - accuracy: 0.964

Epoch 00048: val_loss did not improve from 0.16192
Epoch 49/50
```

```
Epoch 49/50
46/46 [==============================] - 98s 2s/step - loss: 0.0747 - accuracy: 0.972

Epoch 00049: val_loss did not improve from 0.16192
Epoch 50/50
46/46 [==============================] - 97s 2s/step - loss: 0.1121 - accuracy: 0.968

Epoch 00050: val_loss did not improve from 0.16192
```

```
model = load_model(path_save + 'ResNet50_aguacates.h5')
```

```
evaluacion = model.evaluate(test_generator)
print ("Loss = " + str(evaluacion[0]))
print ("Test Accuracy = " + str(evaluacion[1]))
```

```
16/16 [==============================] - 199s 10s/step - loss: 0.1190 - accuracy: 0.9571
Loss = 0.11899634450674057
Test Accuracy = 0.9571428298950195
```

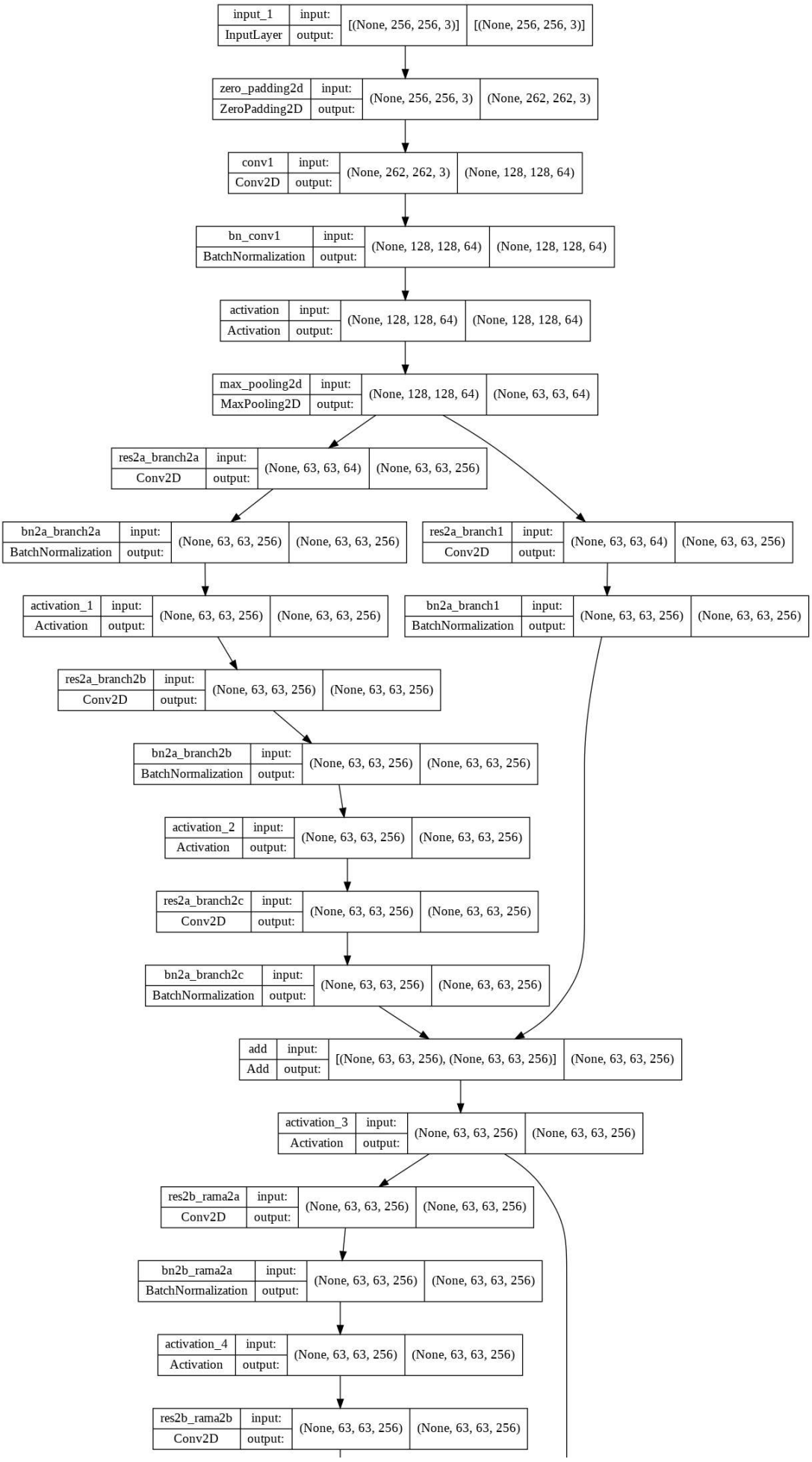
```
model.summary()
```

```
Model: "ResNet50"
```

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 256, 256, 3) | 0 | |
| zero_padding2d (ZeroPadding2D) | (None, 262, 262, 3) | 0 | input_1[0][0] |
| conv1 (Conv2D) | (None, 128, 128, 64) | 9472 | zero_padding2d[0][0] |
| bn_conv1 (BatchNormalization) | (None, 128, 128, 64) | 256 | conv1[0][0] |
| activation (Activation) | (None, 128, 128, 64) | 0 | bn_conv1[0][0] |
| max_pooling2d (MaxPooling2D) | (None, 63, 63, 64) | 0 | activation[0][0] |
| res2a_branch2a (Conv2D) | (None, 63, 63, 256) | 16640 | max_pooling2d[0][0] |
| bn2a_branch2a (BatchNormalizati | (None, 63, 63, 256) | 1024 | res2a_branch2a[0][0] |
| activation_1 (Activation) | (None, 63, 63, 256) | 0 | bn2a_branch2a[0][0] |
| res2a_branch2b (Conv2D) | (None, 63, 63, 256) | 590080 | activation_1[0][0] |
| bn2a_branch2b (BatchNormalizati | (None, 63, 63, 256) | 1024 | res2a_branch2b[0][0] |
| activation_2 (Activation) | (None, 63, 63, 256) | 0 | bn2a_branch2b[0][0] |
| res2a_branch2c (Conv2D) | (None, 63, 63, 256) | 65792 | activation_2[0][0] |

| | | | |
|---|---|---|---|
| res2a_branch1 (Conv2D) | (None, 63, 63, 256) | 16640 | max_pooling2d[0][0] |
| bn2a_branch2c (BatchNormalizati | (None, 63, 63, 256) | 1024 | res2a_branch2c[0][0] |
| bn2a_branch1 (BatchNormalizatio | (None, 63, 63, 256) | 1024 | res2a_branch1[0][0] |
| add (Add) | (None, 63, 63, 256) | 0 | bn2a_branch2c[0][0]<br>bn2a_branch1[0][0] |
| activation_3 (Activation) | (None, 63, 63, 256) | 0 | add[0][0] |
| res2b_rama2a (Conv2D) | (None, 63, 63, 256) | 65792 | activation_3[0][0] |
| bn2b_rama2a (BatchNormalization | (None, 63, 63, 256) | 1024 | res2b_rama2a[0][0] |
| activation_4 (Activation) | (None, 63, 63, 256) | 0 | bn2b_rama2a[0][0] |
| res2b_rama2b (Conv2D) | (None, 63, 63, 256) | 590080 | activation_4[0][0] |
| bn2b_rama2b (BatchNormalization | (None, 63, 63, 256) | 1024 | res2b_rama2b[0][0] |
| activation_5 (Activation) | (None, 63, 63, 256) | 0 | bn2b_rama2b[0][0] |
| res2b_rama2c (Conv2D) | (None, 63, 63, 256) | 65792 | activation_5[0][0] |
| bn2b_rama2c (BatchNormalization | (None, 63, 63, 256) | 1024 | res2b_rama2c[0][0] |

```
plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```


input_1 | input: | [(None, 256, 256, 3)] | [(None, 256, 256, 3)]
InputLayer | output:
zero_padding2d | input: | (None, 256, 256, 3) | (None, 262, 262, 3)
ZeroPadding2D | output:
conv1 | input: | (None, 262, 262, 3) | (None, 128, 128, 64)
Conv2D | output:
bn_conv1 | input: | (None, 128, 128, 64) | (None, 128, 128, 64)
BatchNormalization | output:
activation | input: | (None, 128, 128, 64) | (None, 128, 128, 64)
Activation | output:
max_pooling2d | input: | (None, 128, 128, 64) | (None, 63, 63, 64)
MaxPooling2D | output:
res2a_branch2a | input: | (None, 63, 63, 64) | (None, 63, 63, 256)
Conv2D | output:
bn2a_branch2a | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
BatchNormalization | output:
res2a_branch1 | input: | (None, 63, 63, 64) | (None, 63, 63, 256)
Conv2D | output:
activation_1 | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Activation | output:
bn2a_branch1 | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
BatchNormalization | output:
res2a_branch2b | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Conv2D | output:
bn2a_branch2b | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
BatchNormalization | output:
activation_2 | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Activation | output:
res2a_branch2c | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Conv2D | output:
bn2a_branch2c | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
BatchNormalization | output:
add | input: | [(None, 63, 63, 256), (None, 63, 63, 256)] | (None, 63, 63, 256)
Add | output:
activation_3 | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Activation | output:
res2b_rama2a | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Conv2D | output:
bn2b_rama2a | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
BatchNormalization | output:
activation_4 | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Activation | output:
res2b_rama2b | input: | (None, 63, 63, 256) | (None, 63, 63, 256)
Conv2D | output:



**Anexo 6.**
Código implementación Inception Network

# Aplicación de Redes Neuronales en el proceso de inspección de calidad del aguacate Hass para exportación.

## Red neuronal Inception

```
# Path to photos
from google.colab import drive
drive.mount('/content/drive/')
```

```
     Mounted at /content/drive/
```

```
# Importing the necessary libraries to train the neural network
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
from keras import layers
from keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Fl
from tensorflow.keras.optimizers import RMSprop
from keras.models import Model, load_model
from keras.preprocessing import image
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
import pydot
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.initializers import glorot_uniform
from keras.utils.vis_utils import plot_model
import scipy.misc
from sklearn.model_selection import train_test_split
import keras.backend as k
from keras.callbacks import ModelCheckpoint as model_checkpoint
import tensorflow.keras.losses as losses
from tensorflow.keras.applications import InceptionV3
from tensorflow.nn import local_response_normalization
%matplotlib inline
```

```
#=======================
train_cump = "/content/drive/MyDrive/Project/Images/Training/Cumple"
train_no_cump = "/content/drive/MyDrive/Project/Images/Training/No_Cumple"
test_cump = "/content/drive/MyDrive/Project/Images/Test/Cumple"
test_no_cump = "/content/drive/MyDrive/Project/Images/Test/No_Cumple"
```

```
dev_cump = "/content/drive/MyDrive/Project/Images/Validation/Cumple"
dev_no_cump = "/content/drive/MyDrive/Project/Images/Validation/No_Cumple"

print("Imagenes de entrenamiento totales de aguacates que cumplen con las características de
print("Imagenes de entrenamiento totales de aguacates que no cumplen con las características
print("Imagenes de prueba totales de aguacates que cumplen con las características de exporta
print("Imagenes de prueba totales de aguacates que no cumplen con las características de expo
print("Imagenes de validación totales de aguacates que cumplen con las características de exp
print("Imagenes de validación totales de aguacates que no cumplen con las características de
```

```
Imagenes de entrenamiento totales de aguacates que cumplen con las características de ex
Imagenes de entrenamiento totales de aguacates que no cumplen con las características de
Imagenes de prueba totales de aguacates que cumplen con las características de exportaci
Imagenes de prueba totales de aguacates que no cumplen con las características de export
Imagenes de validación totales de aguacates que cumplen con las características de expor
Imagenes de validación totales de aguacates que no cumplen con las características de ex
```

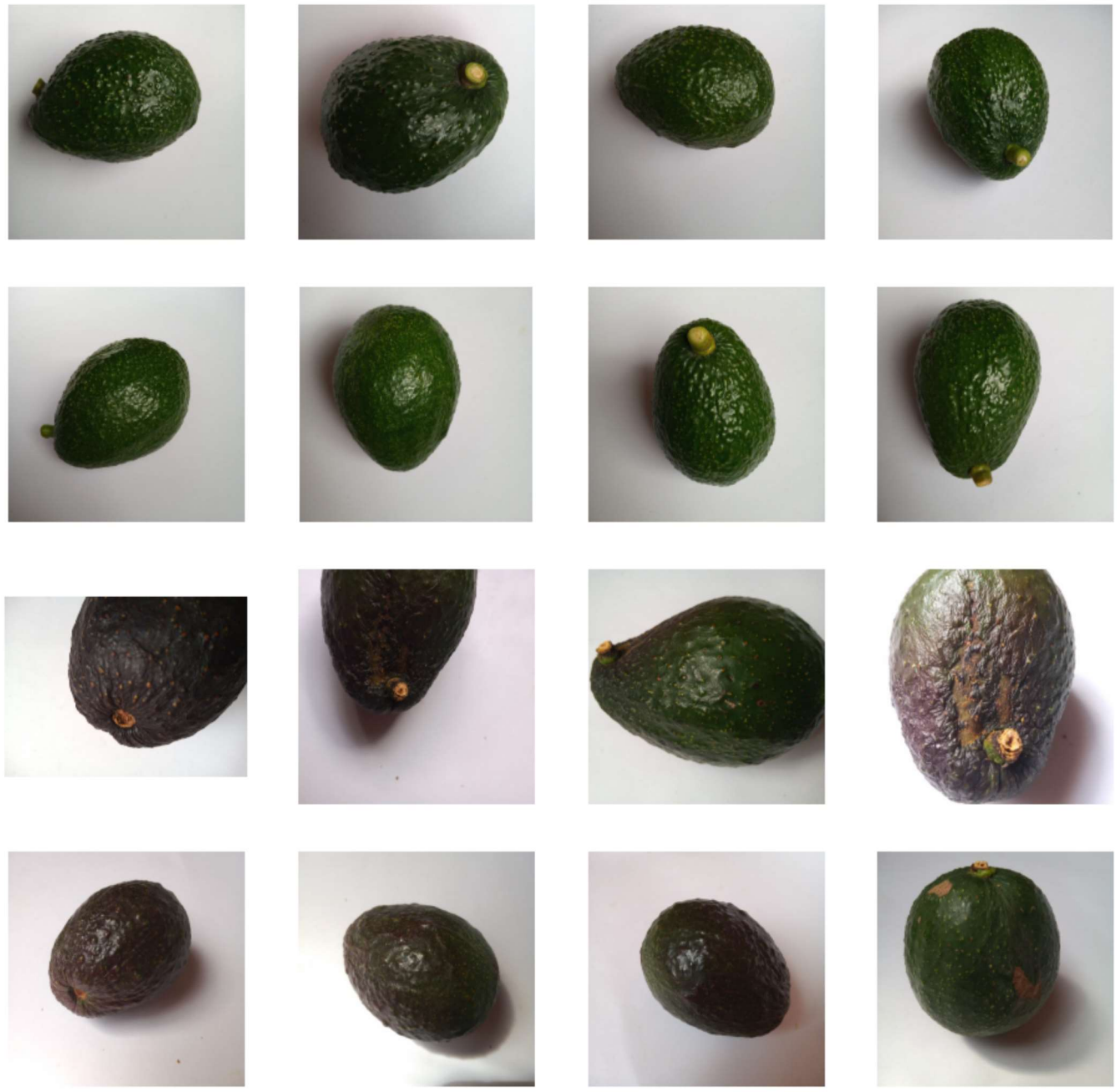
```
train_cumple_nombres =  os.listdir(train_cump)
train_no_cumple_nombres = os.listdir(train_no_cump)
validation_cumple = os.listdir(dev_cump)
validation_no_cumple = os.listdir(dev_no_cump)


nfilas = 4
ncolumnas = 4
indice_pic = 0
fig = plt.gcf()
fig.set_size_inches(ncolumnas*4, nfilas*4)
indice_pic += 8
aguacate_cumple = [os.path.join(train_cump,fnombre) for fnombre in train_cumple_nombres[indic
aguacate_no_cumple = [os.path.join(train_no_cump,fnombre) for fnombre in train_no_cumple_nomb

for i, img_path in enumerate(aguacate_cumple+aguacate_no_cumple):
  #subplot
  sp = plt.subplot(nfilas, ncolumnas, i+1)
  sp.axis("off")

  img = mpimg.imread(img_path)
  plt.imshow(img)
plt.show
```

```
<function matplotlib.pyplot.show>
```



```
def inception_block(X, filtros):

    f1, f2, f3, f4 = filtros


    # First layer or block
    conv1 = Conv2D(filters = f1, kernel_size = (1, 1), strides = (1,1), padding = 'same')(X)
    conv1 = Conv2D(filters = f1, kernel_size = (3, 3), strides = (1,1), padding = 'same')(con
    conv1_1 = Conv2D(filters = f1, kernel_size = (1, 3), strides = (1,1), padding = 'same')(c
    conv1_2 = Conv2D(filters = f1, kernel_size = (3, 1), strides = (1,1), padding = 'same')(c

    # Second layer
```

```
        conv2 = Conv2D(filters=f2, kernel_size=(1,1), strides=(1,1), padding='same')(X)
        conv2_1 = Conv2D(filters=f2, kernel_size=(1,3), strides=(1,1), padding='same')(conv2)
        conv2_2 = Conv2D(filters=f2, kernel_size=(3,1), strides=(1,1), padding='same')(conv2)

        # Third layer
        conv3 = MaxPooling2D((3,3), strides=(1,1), padding='same')(X)
        conv3 = Conv2D(filters=f3, kernel_size=(1,1), strides=(1,1), padding='same')(conv3)

        # Fourth layer
        conv4 = Conv2D(filters=f4, kernel_size=(1,1), strides=(1,1), padding='same')(X)

        # Final
        output = layers.concatenate([conv1_1,conv1_2, conv2_1, conv2_2, conv3, conv4], axis=-1)

        return output

    def inception_net(input_shape = (256, 256, 3)):

        X_input = Input(input_shape)

        X = ZeroPadding2D((3, 3))(X_input)

        local_response_normalization

        # Initial stage
        X = Conv2D(96, (7, 7), strides = (1, 1), kernel_initializer = glorot_uniform(seed=0), pad
        X = BatchNormalization(axis = 3)(X)
        X = Activation('relu')(X)
        X = MaxPooling2D((3, 3), strides=(2, 2), padding = 'same')(X)

        X = Conv2D(192, (1, 1), strides = (1, 1), kernel_initializer = glorot_uniform(seed=0))(X)
        X = Activation('relu')(X)
        X = Conv2D(192, (3, 3), strides = (2, 2), kernel_initializer = glorot_uniform(seed=0), pa
        X = BatchNormalization(axis = 3)(X)
        X = Activation('relu')(X)
        X = MaxPooling2D((3, 3), strides=(1, 1))(X)

        X = inception_block(X,[96,16,32,64])
        X = inception_block(X,[96,16,32,64])
        X = MaxPooling2D((3, 3), strides=(2, 2))(X)

        X = inception_block(X,[96,16,32,64])
        X = inception_block(X,[96,16,32,64])
        X = inception_block(X,[96,16,32,64])
        X = inception_block(X,[96,16,32,64])
        X = inception_block(X,[96,16,32,64])
        X = MaxPooling2D((3, 3), strides=(2, 2))(X)

        X = inception_block(X,[96,16,32,64])
        X = inception_block(X,[96,16,32,64])
```

```
    # AVGPOOL
    X = AveragePooling2D(pool_size=(5,5))(X)

    # Output layer
    X = Flatten()(X)
    X = Dense(1, activation='sigmoid', kernel_initializer = glorot_uniform(seed=0))(X)

    # Create the model
    model = Model(inputs = X_input, outputs = X, name='Inception')

    return model


model = inception_net(input_shape = (256, 256, 3))


model.compile(loss = losses.BinaryCrossentropy(), optimizer = RMSprop(learning_rate= 0.001),


path_save = "/content/drive/MyDrive/chekpoints/"


from keras.preprocessing.image import ImageDataGenerator

#Normalize the images to facilitate processing taking into account that they are color photos
train_datagen = ImageDataGenerator(rescale = 1/255)
validation_datagen = ImageDataGenerator(rescale = 1/255)
test_detagen = ImageDataGenerator(rescale= 1/255)

#Prepare training photos
train_generator = train_datagen.flow_from_directory(
    "/content/drive/MyDrive/Project/Images/Training",
    target_size = (256,256),
    batch_size = 32,
    class_mode = 'binary'
)

#Prepare the validation photos
validation_generator = validation_datagen.flow_from_directory(
    "/content/drive/MyDrive/Project/Images/Validation",
    target_size = (256,256),
    batch_size = 32,
    class_mode = 'binary'
)

#Prepare the test photos
test_generator = test_detagen.flow_from_directory(
    "/content/drive/MyDrive/Project/Images/Test",
    target_size = (256,256),
    class_mode = 'binary'
)
```

```
Found 1470 images belonging to 2 classes.
Found 490 images belonging to 2 classes.
Found 490 images belonging to 2 classes.
```

```
"""class myCallback(Callback):
  def on_epoch_end(self, epoch, logs ={}):
    if (logs.get('accuracy')>= 0.80):
      print("\n accuracy is higher, stops training")
      self.model.stop_training = True """

     'class myCallback(Callback):\n  def on_epoch_end(self, epoch, logs ={}):\n    if (logs.
     get(\'accuracy\')>= 0.80):\n      print("\n accuracy is higher, stops training")\n
     self.model.stop_training = True '
```

```
checkpoint = model_checkpoint(filepath = path_save + 'Inception_keras.h5', save_best_only=Tru
```

```
history = model.fit(
      train_generator,
      callbacks = [checkpoint],
      steps_per_epoch=46,
      epochs=50,
      batch_size = 32,
      verbose=1,
      validation_data = validation_generator,
      validation_steps=16
      )
```

```
Epoch 1/50
46/46 [==============================] - ETA: 0s - loss: 14519.4053 - accuracy: 0.592
Epoch 1: val_loss improved from inf to 13.39933, saving model to /content/drive/MyDri
46/46 [==============================] - 530s 11s/step - loss: 14519.4053 - accuracy:
Epoch 2/50
46/46 [==============================] - ETA: 0s - loss: 48.3374 - accuracy: 0.5456
Epoch 2: val_loss improved from 13.39933 to 7.79251, saving model to /content/drive/My
46/46 [==============================] - 80s 2s/step - loss: 48.3374 - accuracy: 0.54
Epoch 3/50
46/46 [==============================] - ETA: 0s - loss: 2646.7451 - accuracy: 0.5422
Epoch 3: val_loss did not improve from 7.79251
46/46 [==============================] - 78s 2s/step - loss: 2646.7451 - accuracy: 0.
Epoch 4/50
46/46 [==============================] - ETA: 0s - loss: 13.0246 - accuracy: 0.5810
Epoch 4: val_loss did not improve from 7.79251
46/46 [==============================] - 77s 2s/step - loss: 13.0246 - accuracy: 0.58
Epoch 5/50
46/46 [==============================] - ETA: 0s - loss: 925.1470 - accuracy: 0.6116
Epoch 5: val_loss did not improve from 7.79251
46/46 [==============================] - 77s 2s/step - loss: 925.1470 - accuracy: 0.6
Epoch 6/50
46/46 [==============================] - ETA: 0s - loss: 7.6679 - accuracy: 0.5728
Epoch 6: val_loss improved from 7.79251 to 1.95441, saving model to /content/drive/My
46/46 [==============================] - 78s 2s/step - loss: 7.6679 - accuracy: 0.572
Epoch 7/50
```

```
46/46 [==============================] - ETA: 0s - loss: 13.1778 - accuracy: 0.5469
Epoch 7: val_loss improved from 1.95441 to 0.64251, saving model to /content/drive/My
46/46 [==============================] - 78s 2s/step - loss: 13.1778 - accuracy: 0.54
Epoch 8/50
46/46 [==============================] - ETA: 0s - loss: 59.3506 - accuracy: 0.6204
Epoch 8: val_loss did not improve from 0.64251
46/46 [==============================] - 77s 2s/step - loss: 59.3506 - accuracy: 0.62
Epoch 9/50
46/46 [==============================] - ETA: 0s - loss: 88.5528 - accuracy: 0.5020
Epoch 9: val_loss did not improve from 0.64251
46/46 [==============================] - 77s 2s/step - loss: 88.5528 - accuracy: 0.50
Epoch 10/50
46/46 [==============================] - ETA: 0s - loss: 1531.7067 - accuracy: 0.5755
Epoch 10: val_loss did not improve from 0.64251
46/46 [==============================] - 77s 2s/step - loss: 1531.7067 - accuracy: 0.
Epoch 11/50
46/46 [==============================] - ETA: 0s - loss: 10.9858 - accuracy: 0.5986
Epoch 11: val_loss did not improve from 0.64251
46/46 [==============================] - 77s 2s/step - loss: 10.9858 - accuracy: 0.59
Epoch 12/50
46/46 [==============================] - ETA: 0s - loss: 11.9266 - accuracy: 0.5585
Epoch 12: val_loss did not improve from 0.64251
46/46 [==============================] - 77s 2s/step - loss: 11.9266 - accuracy: 0.55
Epoch 13/50
46/46 [==============================] - ETA: 0s - loss: 0.7616 - accuracy: 0.6878
Epoch 13: val_loss did not improve from 0.64251
46/46 [==============================] - 77s 2s/step - loss: 0.7616 - accuracy: 0.687
Epoch 14/50
46/46 [==============================] - ETA: 0s - loss: 18162.1582 - accuracy: 0.493
Epoch 14: val_loss did not improve from 0.64251
46/46 [==============================] - 77s 2s/step - loss: 18162.1582 - accuracy: 0
Epoch 15/50
```

```
model = load_model(path_save + 'Inception_keras.h5')
```

```
evaluacion = model.evaluate(test_generator)
print ("Loss = " + str(evaluacion[0]))
print ("Test Accuracy = " + str(evaluacion[1]))
```

```
16/16 [==============================] - 175s 12s/step - loss: 0.4227 - accuracy: 0.8245
Loss = 0.42273369431495667
Test Accuracy = 0.8244897723197937
```

```
model.summary()
```

```
Model: "Inception"
__________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================
 input_9 (InputLayer)        [(None, 256, 256, 3  0           []
                             )]
```

```
zero_padding2d_8 (ZeroPadding2  (None, 262, 262, 3)  0          ['input_9[0][0]']
D)

conv2d_600 (Conv2D)            (None, 262, 262, 96  14208       ['zero_padding2d_8[0
                               )                                 ]

batch_normalization_16 (BatchN  (None, 262, 262, 96  384        ['conv2d_600[0][0]']
ormalization)                  )

activation_24 (Activation)     (None, 262, 262, 96  0           ['batch_normalizatio
                               )                                 ]

max_pooling2d_94 (MaxPooling2D  (None, 130, 130, 96  0          ['activation_24[0][0
)                              )                                 ]

conv2d_601 (Conv2D)            (None, 130, 130, 19  18624       ['max_pooling2d_94[0
                               2)                                ]

activation_25 (Activation)     (None, 130, 130, 19  0           ['conv2d_601[0][0]']
                               2)

conv2d_602 (Conv2D)            (None, 65, 65, 192)  331968      ['activation_25[0][0

batch_normalization_17 (BatchN  (None, 65, 65, 192)  768        ['conv2d_602[0][0]']
ormalization)

activation_26 (Activation)     (None, 65, 65, 192)  0           ['batch_normalizatio

max_pooling2d_95 (MaxPooling2D  (None, 63, 63, 192)  0          ['activation_26[0][0
)

conv2d_603 (Conv2D)            (None, 63, 63, 96)   18528       ['max_pooling2d_95[0

conv2d_604 (Conv2D)            (None, 63, 63, 96)   83040       ['conv2d_603[0][0]']

conv2d_607 (Conv2D)            (None, 63, 63, 16)   3088        ['max_pooling2d_95[0

max_pooling2d_96 (MaxPooling2D  (None, 63, 63, 192)  0          ['max_pooling2d_95[0
)

conv2d_605 (Conv2D)            (None, 63, 63, 96)   27744       ['conv2d_604[0][0]']

conv2d_606 (Conv2D)            (None, 63, 63, 96)   27744       ['conv2d_604[0][0]']

conv2d_608 (Conv2D)            (None, 63, 63, 16)   784         ['conv2d_607[0][0]']

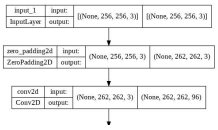
conv2d_609 (Conv2D)            (None, 63, 63, 16)   784         ['conv2d_607[0][0]']

conv2d_610 (Conv2D)            (None, 63, 63, 32)   6176        ['max_pooling2d_96[0
```

He,K Zhang,X Ren,S, Sun,J (2015) *Deep Residual Learning for Image Recognition*

Estados Unidos: Microsoft Reseacrh

```
plot_model(model, to_file='model.png', show_shapes=True)
```




```
import os, signal
os.kill(os.getpid(), signal.SIGKILL)
```

