UAEM | Universidad Autónoma del Estado de México

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN

Clasificación de la fruta del Durazno mediante Redes Neuronales Convolucionales

---

Tesis que presenta

Ma. Dolores Arévalo Zenteno

Para obtener el Grado de

Doctora en Ciencias de la Computación

Asesor de Tesis:

Dr. en I.S. José Sergio Ruiz Castilla

Texcoco, Estado de México. Junio de 2021

Resumen

Doctorado en Ciencias de la Computación
Clasificación de la Fruta de Durazno a través de Redes Neuronales Convolucionales
Por Ma. Dolores Arévalo Zenteno

Existen métodos manuales para reconocer el grado de madurez del durazno, en esta actividad los agricultores requieren de tiempo de aprendizaje y se requiere de una acción visual para detectar aquellos frutos que se van a cortar del árbol. Con los sistemas de visión artificial, específicamente Redes Neuronales Convolucionales, se propuso una solución para realizar el reconocimiento de frutos de durazno maduros, así como la identificación de frutos dañados. La finalidad es obtener frutos con el nivel de calidad adecuado para su comercialización. Para lograr el propósito, se obtuvieron imágenes de duraznos en un ambiente no controlado. Se recortaron las imágenes digitales hasta obtener el área de interés enfocando a la imagen de la fruta del durazno. Se configuraron tres conjuntos de datos: el primero de duraznos maduros e inmaduros, el segundo de duraznos maduros e inmaduros, pero solo enfocado a un área textural y el tercero de duraznos sanos y dañados. Se aplicó una Red Neuronal Convolucional, que fue programada en el lenguaje de programación *Python*, las librerías de *Keras* y *Tensorflow*. Durante las pruebas se obtuvieron resultados del 95,31% de precisión, para elegir entre maduros y no maduros. Mientras que, al clasificar los duraznos sanos y dañados un 92.18% de precisión. Al clasificar las tres categorías: dañados, inmaduros y maduros se obtuvo un 83.33% de precisión. Los resultados anteriores indican que con inteligencia artificial, embebida en un dispositivo físico, se puede hacer la clasificación del fruto del durazno, obteniendo los de mejor calidad para su comercialización.

Índice general

Índice de figuras

Capítulo 1

1.1 Introducción

Existen métodos manuales para reconocer el grado de madurez del durazno, en esta actividad los agricultores requieren de tiempo de aprendizaje y se requiere de una acción visual para detectar aquellos frutos que se van a cortar del árbol. Con los sistemas de visión artificial, específicamente redes neuronales convolucionales, se propuso una solución para realizar el reconocimiento de frutos de durazno maduros, así como la identificación de frutos dañados. La finalidad es obtener frutos con el nivel de calidad adecuado para su comercialización. Para lograr el propósito, se obtuvieron imágenes de duraznos en un ambiente no controlado. Se recortaron las imágenes digitales hasta obtener el área de interés enfocando a la imagen de la fruta del durazno. Se configuraron tres conjuntos de datos: el primero de duraznos maduros e inmaduros, el segundo de duraznos maduros e inmaduros, pero solo enfocado a un área textural y el tercero de duraznos sanos y dañados. Se aplicó una Red Neuronal Convolucional, que fue programada en el lenguaje de programación *Python*, las librerías de *Keras* y *Tensorflow*.

1.2 Planteamiento del Problema

El durazno es conocido como melocotón, duraznero y su nombre científico es *Prunus Persica*, es un árbol *caducifolio*, pequeño, logra alcanzar una altura de 8 metros. Sus hojas son elípticas, acuminadas, aserradas, glabrescentes, con estípulas caducas denticuladas. Las flores son solitarias y con numerosas brácteas y de color rosado fuerte. El fruto es una drupa comestible *subglobosa*, de 4-8 cm de diámetro, con *mesocarpo* muy carnoso, amarillo o blanquecino de sabor dulce y aroma delicado. Su fruto es utilizado ampliamente en diversos usos culinarios por sus características y sabor, además es cultivado con fines comerciales. El durazno o melocotón es consumido como fruta fresca, también se puede procesar y obtener mermeladas, jaleas, almíbares y pulpa concentrada; además de obtener jugos y bases para otros productos agroindustriales. Se usa para preparar, pasteles, postres horneados y licores. Los distintos órganos de la planta (hojas, flores y frutos) poseen

múltiples propiedades medicinales, útiles contra afecciones hepáticas, tiñas, herpes, trastornos nerviosos, perlesías, tullimientos y decaimientos, además de excelentes propiedades vermífugas (Nava V., 2005). La madera obtenida de las podas se utiliza como leña de buenas propiedades calóricas. Existen mas de 15000 (Baíza Avelar, Guía Técnica del Cultivo del Melocotón, 2004) varidedades de Durazno y debido a los híbridos que se generan éste número crece.

El cultivo del Durazno en México se realiza en zonas cálidas o en zonas frías de Chihuahua, así como también en climas secos de Zacatecas, en realidad se puede plantar en la mayoría de los estados del país. El INIPAF (Instituto Nacional de Investigaciones Pecuarias, Agrícolas y Forestales de SAGARPA), tiene un listado de variedades duraznos, que incluyen entre otros: Los que tienen requerimientos de 150 a 200 horas frío anuales: Madrugada cel 10-17 y Madrugada Cel 79-10. De 200 a 250 horas frío anuales: Regio, Sol, Regina. De 350 a 450 horas frío anuales: MIchele, Aurora, y de 450 a 500 horas frío anuales: Fred, Atlax, Dorado, Irina, Escarch, etcetera, y en el nivel de los de pulpa blanca: Nieve y Rocio. Para los productores de duraznos es importante el tiempo de madurez de un cultivo, es importante identificar la etapa de madurez de la fruta en la cosecha, se deben eliminar los duraznos que no tendrán buen termino ya que no tiene caso alguno continúen absorviendo lo que otros buenos duraznos pueden asimilir y madurar con mayor calidad.

El humano puede realizar un análisis visual o táctil para poder elegir el fruto maduro y sano desde el ábol o bien ya cortado. Se trata de delegar la tarea a una máquina capaza de clasificar la fruta en: fruto imaduro, maduro y dañado. Una clasificación, de este nivel podría permitir la automatización de cosecha o seleeción del fruto antes de ser empacado y comercializado.

¿Cómo se puede entrenar a una computadora a identificar los frutos del durazno inmaduros, maduros y dañados?

1.3. Justificación

Si bien existen expertos en el cultivo del durazno que, mediante un análisis visual puede clasificar los frutos del Durazno, no existe un modelo computacional disponible que lleve a cabo ésta acción y use métodos basados en Inteligencia Artificial.

Retomando el análisis visual, qué caracteristicas hacen que el experto en el cultivo de Durazno pueda calsificar la fruta, así pues, se pueden ver las características de la fruta, cuando ya está madura, como su textura, tamaño, su color, entre otras. Algunas caracteristicas a considerar son la forma y el color del fruto, pero también hay algunas otras características que se podrán proporcionar información precisa para a clasificar esa fruta como la textura. Una Red Neuronal Convolucional hace que el aprendizaje sea automático, realizando la segmentación de la imagen de la hoja y extrayendo características de la imagen del fruto. El algoritmo estudia la imagen y extrae las mejores características y hacer una clasificación. Un algoritmo de aprendizaje automatizado se necesita debido a la gran necesidad de clasificar eficientemente las frutas del Durazno. En este caso no se encontró un conjunto de imágenes para poder entrenar el algortimo, por lo que se debió construir el conjunto de imágenes.

Esta línea de investigación se centra en la clasificación de objetos similares, usando características de éstos, utilizando técnicas de Aprendizaje Automático clásicas, junto con las nuevas Redes Neuronales Convolucionales, se analizan resultados de ambos métodos.

1.4 Objetivos y alcances

Objetivo General

Clasificar el fruto del Durazno en Inmaduros, maduros y dañados usando Redes Neuronales Convolucionales

Objetivos Específicos

- Construir un conjunto de imágenes de fruta del durazno.
- Configurar una arquitectura de una Red neuronal convolucional
- Entrenar la Red neuronal convolucional con los Dataset de la fruta del Durazno
- Obtener los resultados de entrenamiento y aplicar métricas de evaluación

1.5 Hipótesis

Una Red Neuronal Convolucional permite la clasificación de la fruta de durazno en inmaduros, maduros y dañados, partiendo de imágenes del fruto.

1.6 Estado del Arte

Para un usuario de dispositivos móviles la inteligencia artificial ha resultado en facilidad de traducción, reconocimiento de elementos en fotos, localización de objetos, clasificación de fotos, etc.; Sin embargo, su aplicación en la fruticultura puede producir información necesaria para la toma de decisiones en el cultivo de frutas, en éste caso, específicamente de Durazno. Las investigaciones que han sido realizadas como punto de partida para el desarrollo y progreso de máquinas de aprendizaje en el área de cultivo de frutos, sin embargo cada uno hacen su propia aportación singular, así como el presente trabajo. Para obtener el estado del arte se ha recurrido a trabajos con redes neuronales aplicados a las imágenes de hojas o frutos ya sea para clasificar o para identificar enfermedades o estudiar un tipo específico de fruto, es constante la evaluación de algoritmos de redes neuronales diferentes para evaluar su precisión en un mismo trabajo, así como también la aplicación de Redes Neuronales Convolucionales de aprendizaje profundo son las técnicas utillizadas en los últimos años.

1.6.1 Enfásis aplicando redes neuronales convolucionales

El Aprendizaje Profundo (DL por sus siglas en inglés) permite que los modelos computacionales compuestos de múltiples capas de procesamiento aprendan representaciones de datos con múltiples niveles de abstracción. Estos métodos han mejorado dramáticamente el estado del arte en reconocimiento de voz, reconocimiento visual de objetos, detección de objetos y muchos otros dominios como el descubrimiento de fármacos y la genómica (LeCun, Bengio, & Hinton, Deep Learning, 2015).

Fue hasta finales del siglo XIX cuando existieron las condiciones computacionales para poder implementar el aprendizade profundo de redes neuronales, (LeCun Y. B., Gradient-based learning applied to document, 1998) diseñó un red neuronal para hacer un OCR, un reconocimiento de dígitos de código postal se usó el conjunto de datos de MNIST para entrenar a su red y lograron un 99,7% de precision, la red se basó en una capa convolucional seguida una capa Maxpool. Es importante mencionar que el conjunto de datos para DL debe ser amplio para mejorar resultados (Morris, A Pyramid CNN for Dense-Leaves Segmentation, 2018).

Las redes neuronales convolucionales (CNN's) han disfrutado recientemente de un gran éxito en el reconocimiento de video e imagen a gran escala (Krizhevsky, Sutskever, & Hinton, ImageNet Clasification with Deep Convolutional Neural Networks, 2012), (Fergus & Zeiler, Matthew D, Visualizing and Understanding Convolutional Networks, 2014), (Simonyan & Zisserman, 2015), lo que ha sido posible debido a los grandes conjuntos de imágenes públicas, como ImageNet (Deng, y otros, 2009), y los sistemas informáticos de alto rendimiento, como las GPU o los clústeres distribuidos a gran escala (Dean, Corrado, Monga, & Chen, Large Scale Distributed Deep Networks, 2012); esto debido al reto ImageNet con ILSVRC, ImageNet large scale visual recognition Challenge (Russakovsky et al., 2014), que ha servido como un banco de pruebas para algunas generaciones de sistemas de clasificación de imágenes a gran escala, desde codificaciones de características poco profundas de alta dimensión (Perronnin et al., 2010) a Redes Neuronales Convolucionales Profundas (DCNN) (Krizhevsky, Sutskever, & Hinton, ImageNet Clasification with Deep Convolutional Neural Networks, 2012) cuya arquitectura se ha querido mejorar para lograr una mejor precisión. Las presentaciones de mejor desempeño al ILSVRC-2013 (Fergus & Zeiler, Matthew D, Visualizing and Understanding Convolutional Networks, 2014), (Sermanet, y otros, 2014) utilizaron un tamaño de ventana receptivo más pequeño y un paso más pequeño de la primera capa convolucional.

La red AlexNet de (Krizhevsky, Sutskever, & Hinton, ImageNet Clasification with Deep Convolutional Neural Networks, 2012) muestra que una Red Neuronal Convolucional grande y profunda es capaz de lograr resultados récord en un conjunto de datos utilizando un aprendizaje puramente supervisado, con ésta investigación concluyen que el rendimiento de la red se degrada si se elimina una sola capa convolucional. Sin embargo, Zeile (Zeile & Fergus, Visualizing and Understanding Convolutional Networks, 2014) presentó una nueva red convolucional llamada ZFNet que fue una mejora de AlexNet modificando algunos parámetros de la red en su trabajo obtienen un error de clasificación del 11%. En 2014, (Szegedy, y otros, 2015) presentaron la

red llamada GoogLeNet que implementa una red llamada Inception que reduce los parámetros de la red y utiliza una agrupación media en lugar de capas totalmente conectadas en el parte superior de la red, para optimizar la calidad, las decisiones arquitectónicas se basaron en el principio de Hebbian y la intuición del procesamiento de múltiples escalas. Con esa red ganan el concurso con un error de clasificación del 6%.

Los trabajos de investigación con Redes Neuronales Convolucionales (CNN) han avanzado y algunos estan orientados a probar modificaciones de la arquitectura (Simonyan & Zisserman, 2015) en el reto ImageNet presentaron 6 diferentes configuraciones de red cambiando la profundidad de las capas convolucionales, pilas de (3x3), con ello insertando la no linealidad y forzando a una regularización, y decrementando el número de parámetros en la pila de capas. Analizaron el efecto de la profundidad de la red convolucional en la precisión en la configuración de reconocimiento de imagen a gran escala. Se hizo una evaluación exhaustiva de las redes de gran profundidad utilizando una arquitectura con filtros de convolución de (3×3), lo que demuestra que se puede lograr una mejora significativa en las configuraciones de la técnica anterior al aumentar la profundidad hasta 19 capas, requiriendo menos épocas para coincidir.

16.2 Enfásis en madurez de Frutos

En identificación de madurez de frutas se tiene el estudio realizado por Kangune (K. Kangune, 2019), donde diseña una Metodología que clasifica la imagen de las uvas en maduras y no maduras. La metodología consta de 4 fases: recolección de 4000 imágenes, pre-procesamiento de imágenes, ingeniería de características y clasificación. Las imágenes se dividieron en dos categorías, uvas maduras y uvas no maduras, de acuerdo al color y forma de las uvas. Se eliminó el ruido con GaussianoBlur() de python, la extracción de características fue con histogramas de RGB y HSV, así como de forma con OpenCV. Se usó un modelo de clasificación con CNNs y uno con Máquina de Vectores de Soporte (SVM). Considerando las características de color como RGB y HSV y características morfológicas como la forma

de las uvas fueron elegidas como características para SVM. El resultado de la validación muestra que el modelo de CNN logra mayor precisión de clasificación con 79,49% que la SVM clasificador que tiene 69%. Probaron 2 optimizadores, RMSProp ofrece una mayor precisión con 79% que Adam con 74% en la CNN. Demostraron una mayor precisión con un gran conjunto de datos, entrenando primero con 2000 imágenes (precisión de 66%) y, posteriormente con 4000 imágenes (precisión de 79%). La CNN consta de 2 capas convolucionales, 1 capa completamente conectada y 1 capa de final y con 20 épocas.

También J. Rezgui (J. Rezgui, 2020) propone una plataforma para ayudar a prevenir el desperdicio de alimentos, utilizando un innovador sistema de inteligencia (IA) usando una CNN para analizar imágenes de frutas tomadas con teléfonos móviles para determinar su estado de madurez actual. Diseñan un flujo de trabajo para la plataforma y se hacen pruebas con diferentes CNN's. Proponen el uso de sensores alternativos para aumentar la evaluación de madurez la exactitud y la precisión que es parte del flujo de trabajo de IFMAP. Dichos sensores podrían tomar la forma de un sensor ultravioleta, un sensor de dureza o un sensor que puede analizar la construcción molecular de una fruta para brindar información sobre el estado actual de la fruta.

Presentan una plataforma inteligente de evaluación de madurez de fruta inteligente Plataforma de evaluación, IFMAP, por sus siglas en inglés que puede identificar frutas y su nivel de madurez utilizando CNN. Proponen opciones de utilizar este algoritmo para ayudar a los consumidores a estar más informados utilizando un dispositivo móvil. Así como argumentan la posibilidad de reutilizar el mismo algoritmo en muchas áreas de la cadena de suministro de alimentos, como procesamiento o fabricación; Su marco de trabajo está compuesto de cuatro componentes vitales: la red neuronal artificial, el servidor, la aplicación móvil y la alternativa sensor. Muestra diferentes resultados que para las CNN, como el mejor rendimiento de diferentes arquitecturas, que pueden clasificar los datos con la precisión de 97,11%.

Se sugiere en (Santi Kumari Behera, 2020)dos enfoques basados en el aprendizaje automático y el aprendizaje por transferencia, es decir CNN's para la clasificación de 3 estados de madurez de la papaya. Hacen una comparación de resultados con ambos métodos. La experimentación se realizó con 300 imágenes de muestra de frutos de papaya que incluyen 100 de cada tres etapas de madurez. El enfoque de aprendizaje automático incluye tres conjuntos de características y tres clasificadores con sus diferentes funciones de kernel. Las características y clasificadores utilizados en los enfoques de aprendizaje automático son Patrón Binario Local (LBP), Histograma de Gradientes Orientados (HOG), Matriz de Co-ocurrencia de Nivel de Gris (GLCM) y Vecino más cercano k (KNN), Máquina de Vectores de Soporte (SVM) Naïve Bayes respectivamente. Con respecto a CNN's usan 7 modelos previamente entrenados como ResNet101, ResNet50, ResNet18, VGG19, VGG16, GoogleNet y AlexNet.

Borrmann et al (Borrmann, Ferraciolli, Folego, & A. Rodrigues, 2020) usa *Deep Learning* para identificar tomates en condiciones de cultivo, lo que resulta complicado debido a la redundancia del color verde y dado que el tomate en su madurez continua verde, para obtener grande cantidades de imágenes usa rotación tanto vertical y horizontal, transformaciones y rotaciones de las originales. Usa redes ya preentrenadas como *AlexNet, ResNe18t, ResNet50, resNet152, VGG16, VGG19* y otros. En las condiciones trabajadas los mejores resultados los obtuvó con la red *ResNet18.*

Con respecto a trabajos recientes que tratan de mejorar las redes neuronales ya existentes las *CNN's* han mostrado un gran rendimiento en varios campos, como la clasificación de imágenes, el reconocimiento de patrones y la compresión multimedia (Zhang, y otros, Recent advances in convolutional neural network acceleration, 2018). La conectividad local y el compartir peso, pueden reducir la cantidad de parámetros y aumentar la velocidad de procesamiento. Incorporan una gran gama de disciplinas y proporciona una referencia completa para los investigadores interesados en la aceleración de las *CNN's*. Zhang proporciona avances recientes en los métodos de aceleración CNN de todos los niveles de estructura, nivel de algoritmo

y nivel de implementación. Concluye que la CNN se desempeña mejor en el campo de la visión por computadora a medida que su estructura se profundiza y la cantidad de datos aumenta, lo que hace que consuma mucho tiempo y sea computacionalmente costoso.

Es interesante leer el trabajo de Naranjo et al (Naranjo Torres, y otros, 2020) afirma que el Deep Learning se ha convertido en un método muy utilizado para identificar patrones en imágenes de frutos, ya sea con redes diseñadas o preentrenadas. Definen que las propiedades de calidad más relevantes de los productos agrícolas son el color, el tamaño, la textura, la forma y los defectos. Por lo tanto, presentan una descripción general de los diferentes métodos de preprocesamiento, segmentación, extracción de características y clasificación utilizando estas características. Estudian varios elementos para la clasificación en la evaluación de la calidad de los alimentos, incluidos KNN, SVM, ANN y CNN. Según ellos, los enfoques basados DL, como las redes neuronales convolucionales, que son muy eficientes para la clasificación y el reconocimiento de frutas, obteniendo minimos errores en la clasificación.

Recientes investigaciones han logrado analizar las CNNs de forma más detallada con una arquitectura de auto descompocisión jerárquica (SaiRam, Mukherjee, Patra, & Pratim Das, HSD-CNN: Hierarchically self decomposing CNN architecture, 2019), es decir, una red grande se convierte en redes pequeñas de clases pequeñas utilizando un análisis de sensibilidad de filtro específico de clase que cuantifica el impacto de características específicas en una predicción de clase. La red jerárquica descompuesta se puede utilizar y desplegar directamente para obtener subredes para un subconjunto de clases, y se muestra que funciona mejor sin el requisito de volver a capacitar a esas subredes.

Capítulo 2
Preliminares

Hay muchos problemas agrícolas que, de hecho, se abordan con técnicas clásicas de inteligencia artificial, específicamente visión artificial y redes neuronales. Este documento es un ejemplo de este comportamiento al aplicar métodos diversos de clasificación incluyendo redes neuronales convolucionales para la identificación automática de plantas. La identificación automática de frutos es un campo a estudiar ampliamente, grandes investigadores han desarrollado diversos extractores de características de las frutos para la mejora del proceso de clasificación, gran problema que ha sido atención de muchos trabajos de investigación en los últimos años, en particular aquellos basados en el análisis de la imagen de frutos parecidos, como la naranja y la toronja. Gran parte de este trabajo hace uso de las características que los ojos humanos pueden percibir. El objetivo de la automatización en este caso es evitar a los ojos de los expertos en duraznos para identificar la condición del fruto de durazno en maduro, inmaduro y dañado, de la misma forma reducir el tiempo de clasificación, para ello se hará uso de la visión por computadora.

2.1 Deep Learning

La inteligencia artificial (*AI*) y aprendizaje automático (*ML*) se están convirtiendo en una solución de problemas con dominantes técnicas en muchas áreas de la investigación y la industria, no solo por los recientes éxitos de aprendizaje profundo (*DL*) y sus respectivas redes convolucionales (*CNNs*). Sin embargo, son términos que no son similares y es importante plantear su relación. Estos campos comparten la misma hipótesis fundamental: El cálculo es una forma útil de modelar el comportamiento inteligente en máquinas. Se identifican las siguientes definiciones (McCarthy, What Is Artificial Intelligence? Technical report, Stanford University, Available online at: (Accessed June 2, 2018)., 2007), (Khan, Rahmani, Ali Sha, & Bennamound, A guide to convolutional neural networks for computer vision, 2018) y (Beysolow, Introduction to Deep Learning Using R, 2017) respectivamente que se complementan con la **¡Error! No se encuentra el origen de la referencia.** Figura 1

"AI es la ciencia y la ingeniería para hacer máquinas inteligentes"

"ML es un tipo de inteligencia artificial que permite a las computadoras aprender desde los datos, ejemplos y experiencias sin ser expecificamente programado"

"DL es un subcampo de ML que está dedicado a construir algoritmos que aprendan a través de bajo y alto nivel de abstracciones de datos"

"Las redes neuronales convolucionales son un tipo de aprendizaje profundo"

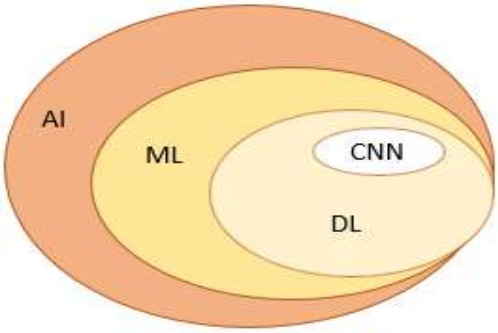

Figura 1: Inteligencia Artificial (*AI*), Aprendizaje Autómatico (*ML*), Aprendizaje Profundo (*DL*) y Redes Neuronales Convolucionales (*CNN*)

IA pretende que las máquinas hagan el trabajo de los humanos, en ML, las computadoras tienen las habilidad de aprender sin ser específicamente programadas, ML y DL están están orientados a a comprobaciones empíricas más que teóricas (como el análisis estadístico o el bayesiano), a diferencia de las estadísticas, el ML tiende a tratar con conjuntos de datos grandes y complejos (Chollet, 2018).

En ML, el aprendizaje se puede realizar de tres formas principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo, es importante resaltar que la semántica de aprendizaje, (más adelante descritos) en realidad se refiere a que las máquinas encuentren un modelo matemático que colabore de la mejor manera a lograr la meta. El modelo se obtiene de datos, que pueden audios, imágenes, documentos y más información digital. Aprendizaje, Figura 2, se refiere a que existe una técnica que analiza los datos (de entrenamiento) y encuentra el modelo por sí misma en lugar de que lo haga un humano (Kim, 2017).

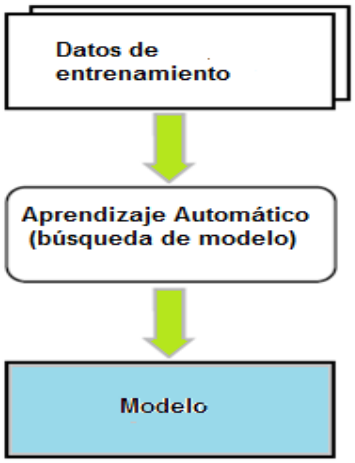

Figura 2: Proceso de aprendizaje automático

El DL es un enfoque especial en el aprendizaje automático que cubre las tres capas y busca también extenderlas para abordar otros problemas de inteligencia artificial que no suelen incluirse en el aprendizaje automático, como la representación del conocimiento, el razonamiento, la planificación y más.

DL, un subcampo de ML que en lugar de enseñar a la computadora un gran número de reglas para resolver el problema se requiere tener un modelo y pequeñas instrucciones en caso de que haya un error al que evaluar el dato. El modelo debe ser adecuado y capaz de resolver el problema con gran precisión, en éste caso de clasificación, que expresado matemáticamente es una función $h(x, \theta)$, donde x es un vector de características o vector de entrada (Nikhil, Fundamentals of Deep Learning. Designing Next-Generation Machine Intelligence Algorithms, 2017).

2.2 Aprendizaje supervisado, no supervisado y por refuerzo

En el *aprendizaje supervisado* se proporciona un gran conjunto de ejemplos de entrenamiento con las respuestas correctas (objetivos) y, en función de este conjunto de entrenamiento, el algoritmo se generaliza para responder correctamente (con alto nivel de precisión) a todas las entradas posibles (Zaccone & Karim, 2018). Esto también se llama aprender de los ejemplos. Las entradas son tomadas de una colección de datos y son pares

(dato: *x*, etiqueta: *y*) y el objetivo es generar una predicción de *y*. La entrada *x* puede ser un vector de características o bien una imagen, documento o algo más complejo, la salida *y* debe ser previamente analizada. La salida puede ser una etiqueta binaria, es decir sólo hay dos tipos de clasificación, pero también existen clasificación multiclases donde *y* se toma de un conjunto de etiquetas y las salidas es multidimensional, en lugar de sólo binaria.

Las funciones tradicionales f(x) para predecir la salida puede tener diferentes formas desde árboles de decisión, regresión logística (LR), máquinas de soporte vectorial (SVM), redes neuronales (NN), maquinas Kernel y clasificadores bayesianos, además de otros. En la Figura 3 se observa el ciclo del aprendizaje supervisado.

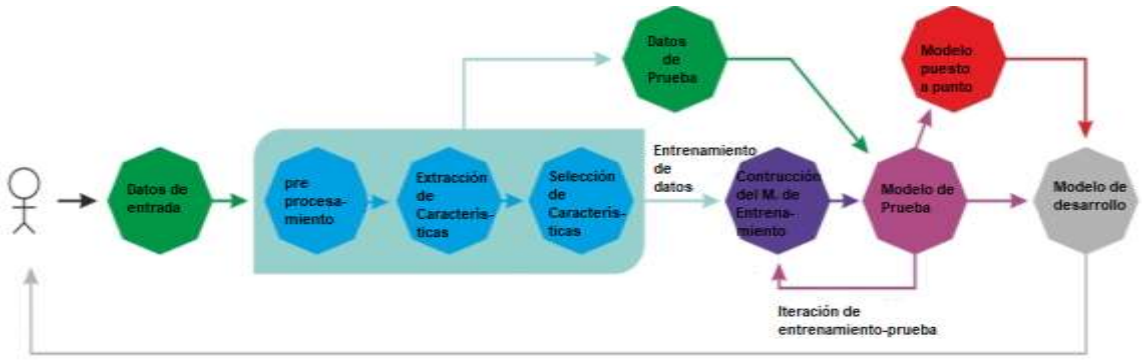

Figura 3: Ciclo de aprendizaje supervisado

El contexto del aprendizaje supervisado es la clasificación, usado para predecir a que clase pertenece un punto, usando valores discretos y la regresión que es usado para predecir valores continuos Figura 4.

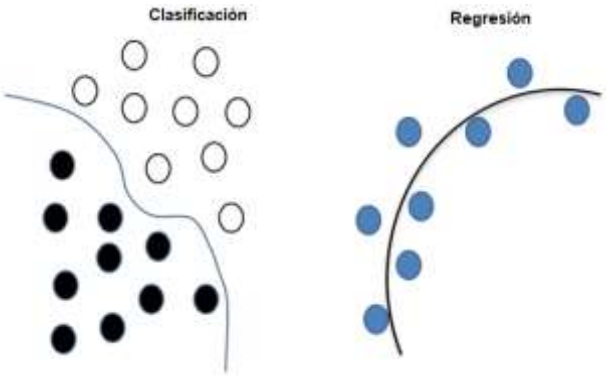

Figura 4: Aprendizaje supervisado: Clasificación y regresión

En el aprendizaje no supervisado, no se proporcionan etiquetas junto con las entradas Figura 5, sino que el algoritmo intenta identificar patrones, en realidad así es como trabaja el cerebro humano. El enfoque estadístico del aprendizaje no supervisado es conocido como estimación de densidad o distribución de probabilidad subyacente desconocida (Zaccone & Karim, 2018). La meta del aprendizaje no supervisado es crear una estructura lineal o de distribución de datos de tal manera que se identifiquen ciertas estructuras en los datos. Los métodos más comunes (Khan, Rahmani, Ali Sha, & Bennamound, A guide to convolutional neural networks for computer vision, 2018) son hierarchical clustering, k-means clustering, modelos combinados gausianos (GMMs), mapas de auto-organización (SOMs) y modelos escondidos de markov (HMMs). Se supone que, dado un conjunto de datos correctamente entrenados, cuando se registra como entrada un dato no conocido la ML debería colocar en el grupo correcto (cluster).

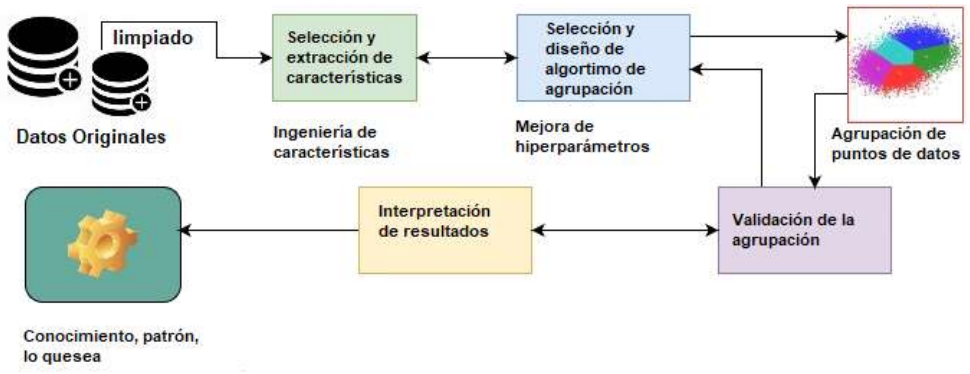

Figura 5: Flujo de trabajo para el aprendizaje no supervisado

Se resumen algunas características de los anteriores aprendizajes Figura 6.

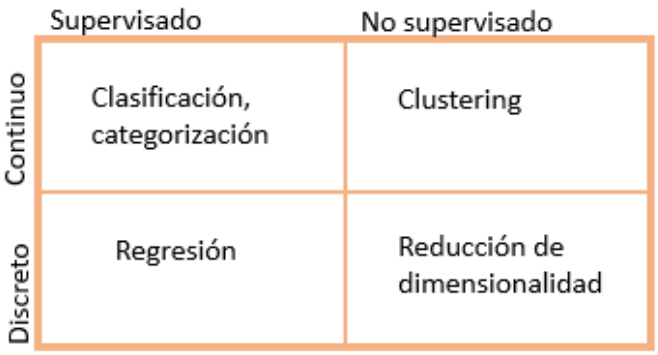

Figura 6: Tipos de aprendizaje y algunos métodos de clasificación, identificación

El aprendizaje por refuerzo es un punto medio entre el aprendizaje supervisado y el no supervisado, el aprendizaje es a través de retroalimentación con el ambiente. El algoritmo indica cuando la respuesta es incorrecta, pero no cómo corregirlo. Tiene que explorar y probar diferentes posibilidades hasta que descubra cómo obtener la respuesta correcta El aprendizaje por refuerzo se denomina en algún momento aprendizaje con un crítico debido a este monitor que puntúa la respuesta, pero no sugiere mejoras. El aprendizaje por refuerzo es usado cuando se tiene una gran cantidad de datos de entrada y sólo algunos de ellos tienen su etiqueta de salida.

Los dos tipos más comunes de aplicación del aprendizaje supervisado son la clasificación y la regresión. La primera puede ser la aplicación más predominante de ML. El problema de clasificación se centra en encontrar literalmente las clases a las que pertenecen los datos. La principal diferencia entre ellos es el tipo de valores a encontra. Mientras que los valores buscados de un problema de regresión son números reales o discretos, los valores buscados de un problema de clasificación son números categóricos que se denominan etiquetas. En lo que respecta a este proyecto el punto de interés es la clasificación.

2.3 Clasificadores

Para tener un clasificador con alta precisión, originalmente se dependía de la calidad de la imagen y del método para clasificar, alguno de estos elementos debía tener alta calidad

para obtener buenos resultados, el estado del arte descrito anteriormente describe los esfuerzos de grandes investigadores por lograr alta precisión al clasificar, algo muy arduo de lograr.

Para la Figura 7, se tiene la imagen de un gato que, suponiendo, tiene 248 pixeles de ancho, 400 pixeles de alto y como esta a color tiene 3 canales de color: rojo, verde y azul, por lo tanto la imagen está compuesta de 297,600 números (248*400*3) cada número es un entero de 0 a 255, entonces el clasficador debe dar la etiqueta (nombre del objeto) de Gato.

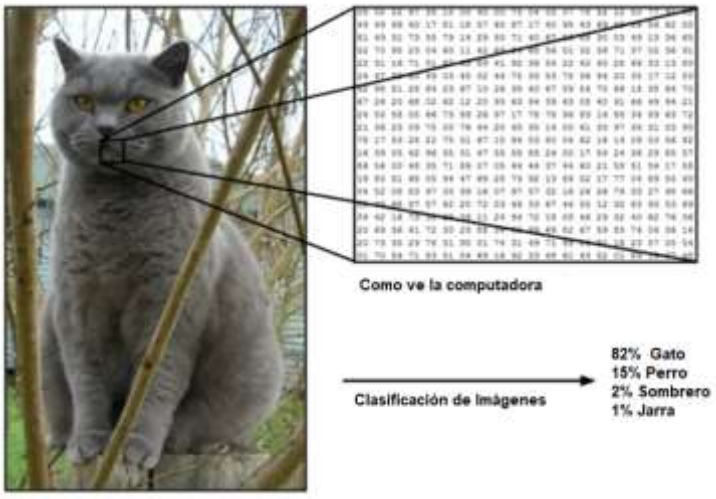

Figura 7: La representación de la imagen en la computadora (cs231n, 2019)

Clasificadores de Aprendizaje Autómáticos

En el *ML* los métodos de clasificación están agrupados en tres categorías, como la Figura 8 muestra.

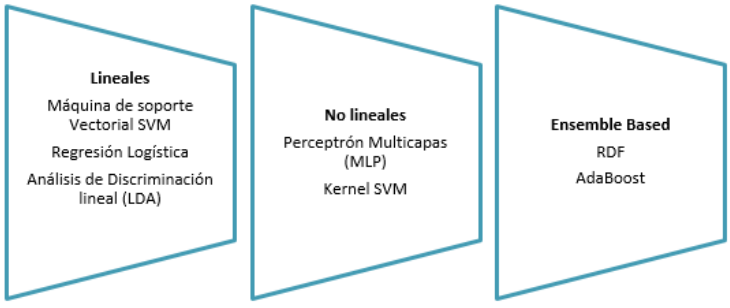

Figura 8: Categorias de clasificadores de aprendizaje autómatico

SVM es un algoritmo de ML para problemas de clasificación y regresión, su trabajo es encontrar, de manera óptima, un hiperplano lineal Figura 9, el cual separa los datos de entrenamiento en dos clases. El criterio para mejorar la separación, es que la distancia (margen) con los datos de entrenamiento más cercanos esté el más lejos posible, es decir, cuanto más largo sea el margen, menor será el error de generalización del modelo.

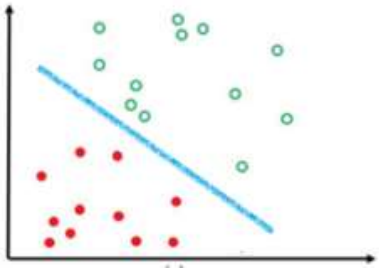

Figura 9: Hiperplano lineal

Matemáticamente $x\ es\ el\ conjunto\ de\ datos\ de\ la\ forma\ \{(x_{1,}, y_1) \dots (x_{n,}, n)\ \}$ donde x_i es un vector de características n dimensional, $y_i\ \in \{1|0\}$ que son las clases a las que puede pertenecer y_i, entonces se debe encontrar el máximo margen del plano cartesiano de los conjuntos de datos que pertenecen a la clase 1 y a la clase 0 y se logra cumpliendo la siguiente ecuación:

$$w^T x_i + b = 0 \tag{1}$$

Donde w es el vector del hiperplano, así entonces los ejemplos arriba del hiperplano pertenecen a la clase 1 y los elementos de abajo serán de la clase 0.

2.4 Arquitecturas de Deep learning

DL son redes neuronales con una gran cantidad de parámetros y capas con las cuales se ha formado arquitecturas básicas y con buen desempeño. En (Emmert-Streib, Yang, Feng, Tripathi, & Dehmer, 2020), basados en trabajos anteriores, realizaron un resumen de una gran variedad de arquitecturas de red utilizadas:

Tabla 1: Modelo de Aprendizaje profundo *DL*

Modelo	No Supervisado	Supervisado
Autoencoder	Si	
Convolutional Deep Belief Network (CDBN)	Si	Si
Convolutional Neural Network (CNN)	Si	Si
Deep Belief Network (DBN)	Si	Si
Deep Boltzmann Machine (DBM)		Si
Denoising Autoencoder (dA)	Si	Si
Long short-term memory (LSTM)		Si
Multilayer Perceptron (MLP)		Si
Recurrent Neural Network (RNN)		Si
Restricted Boltzmann Machine (RBM)	Si	Si
Recursive Neural Network	Si	Si

2.5 Redes Neuronales

En el cuerpo humano, un órgano especial es el cerebro ya que recibe información de los cinco sentidos que tiene el humano, también esta habilitado para almacenar memorias, emociones y sueños, sin él no se podría tener el más mínimo pensamiento. El cerebro es, sin duda, el órgano que hace inteligente al ser humano. La inteligencia artificial trata de hacer que las computadoras piensen y razonen como los humanos, los sistemas artificiales

intentan a copiar la estructura de las redes neuronales biológicas con el de alcanzar una funcionalidad similar.

2.5.1 Redes neuronales

Recordar que en *DL* se requiere encontrar un modelo que solucione el problema de clasificación, haciendo uso de redes neuronales. La unidad básica de una red neuronal es la neurona Figura 10 consiste en un mapeo de regla de propagación $h_1(x_1, \dots x_{n,} w_{i1}, \dots w_{in})$ a partir de las entradas x_i (vector de características de la imagen) y los pesos sinápticos w_{ij} (nivel de prioridad o rango de porcentajes de cada característica) que asigna todas las entradas $b, x_1, x_2, \dots x_n$ a una función de activación f_i que se aplica en la entrada real z para formar la salida o modelo $y = f(z) = f(x, \theta)$. Una función asigna valores de un dominio a un rango. Aquí, b representa una entrada externa llamada *bias* y $x_1, x_2, \dots x_n$ son entradas de otras neuronas de la red y θ es un vector de parámetros que el modelo usa. En una red neuronal, cada neurona está etiquetada de acuerdo con su salida, por lo tanto, para incluir el *bias* se incluye una neurona con valor 1. Si se observa, la función se asemeja a una regresion lineal, en éste caso es un algoritmo para clasificar. Se tienen parámetros que son los *pesos* y los *bias*, la *función* se asemeja a una de regresión lineal se debe modelar un buen vector de *pesos* y un *bias* para lograr una buena clasificación (Sandro, Introduction to Deep Learning. From Logical Calculus to Artifical Intelligence, 2018).

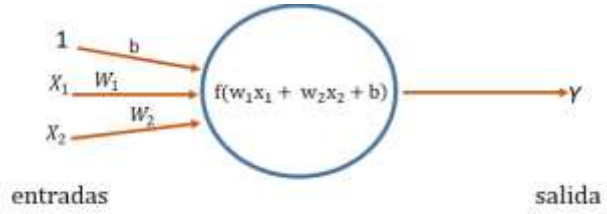

Figura 10: Una neurona, elemento básico de una red Neuronal

La regla de propagacion más comunmente utilizada consiste en combinar linealmente las entradas, un vector $x \in \mathbb{R}^n$, y los pesos sinápticos, un vector $w \in \mathbb{R}^n$, en una suma ponderada:

$$f(y) = h_1(x_1, \dots x_{n,}\, \theta) = \sum_{i=1}^{n} w_{ij}\, x_j + b \qquad (2)$$

Donde $f: \mathbb{R} \to \mathbb{R}$ es la función de Activación. Si se observa la función (Sandro, Introduction to Deep Learning. From Logical Calculus to Artifical Intelligence, 2018), la neurona corresponde a una ecuación de regresión lineal, que para ML es una neurona, el bias permite que la línea se mueva en el plano cartesiano para poder dividir (clasificar) las entradas $x_1, \dots x_{n,}$, ver ecuación (2).

Un vector de parámetros óptimo θ permite que un gran número de predicciones correctas, las configuraciones de los parámetros θ son muchas y puestas en el plano cartesiano se podría verificar que están muy cerca una de la otra. Si el caso no fuera así, se debe tener un conjunto de datos (vectores de entrada) mucho más grande para tener la mejor elección θ, para ello también se requiere de una técnica optimización. Un optimizador apunta a maximizar el rendimiento de un modelo de ML iterativo de sus parámetros hasta que se minimice el error y para ello se debe hacer una medición de error.

2.5.2 Estructura de una red neuronal

Así como el cerebro humano está conformado por miles de neuronas, una neurona en inteligencia artificial está compuesta por muchas neuronas entrelazadas, esto se puede observar en la estructura de la Figura 11.

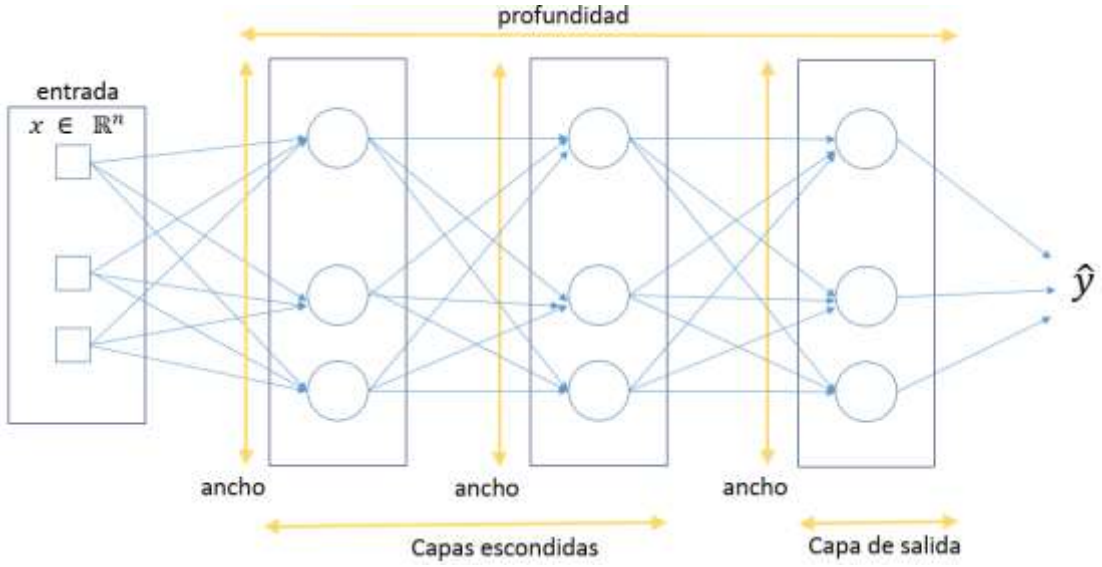

Figura 11: Estructura de una red neuronal

Los elementos referenciados en la Figura 15 son los componentes de la red neuronal, se presentan las peculiaridades de cada una de ellas. Se hace notar que están agrupadas en capas de neuronas, existe una alineación de neuronas en cada capa. La neurona tiene una entrada, un vector $x \in \mathbb{R}^n$, capas escondidas de neuronas y capas de salida; con estas capas se conforma una estructura que tiene un ancho (de neuronas) y una profundidad (de capas) Se mencionan algunas peculiaridades:

Se deben tener en cuenta los siguientes puntos:

- Las unidades se organizan en capas, y cada capa contiene uno o más neuronas.
- La última capa se denomina capa de salida.
- Todas las capas antes de las capas de salida se denominan capas ocultas.
- El número de neuronas en una capa corresponden al ancho de la capa.
- No es necesario que el ancho de cada capa sea el mismo, pero la dimensión debe estar alineada.
- El número de capas se denomina profundidad de la red. De aquí proviene la noción de profundidad.
- Cada capa considera su entrada, la salida producida por la capa anterior, excepto la primera capa, que toma la entrada.
- La salida de la última capa es la salida de la red y es la predicción generada en

base a la entrada.

- Como hemos visto anteriormente, una red neuronal puede verse como una función $f_{\theta:}\, x \to y$, que toma como entrada $x \in \mathbb{R}^n$ y produce como salida $y \in \mathbb{R}^m$ y cuyo comportamiento está parametrizado por $\theta \in \mathbb{R}^p$. Ahora se puede ser más preciso sobre θ; es simplemente una colección de todos los pesos w para todas las unidades de neruonas de la red.
- El diseño de una red neuronal implica, entre otras cosas, definir la estructura general de la red, incluido el número de capas y el ancho de estas capas.

Es importante notar el detalle de las anotaciones anteriores para que se pueda entender mejor el proceso interno del la red neuronal artificial.

Se describe una red neuronal simple en la Figura 12 que la entrada es un vector $x \in \mathbb{R}^n$ y se asume que, la primera capa tiene unidades de neuronas, entonces cada neurona simple tiene un vector $w \in \mathbb{R}^n$ que son pesos asociados a ellas. Los pesos asociados a la primera capa es una matriz de la forma $w_1 \in \mathbb{R}^{nx\,p_1}$ y cada neurona simple de P_1 tiene asociado un bias

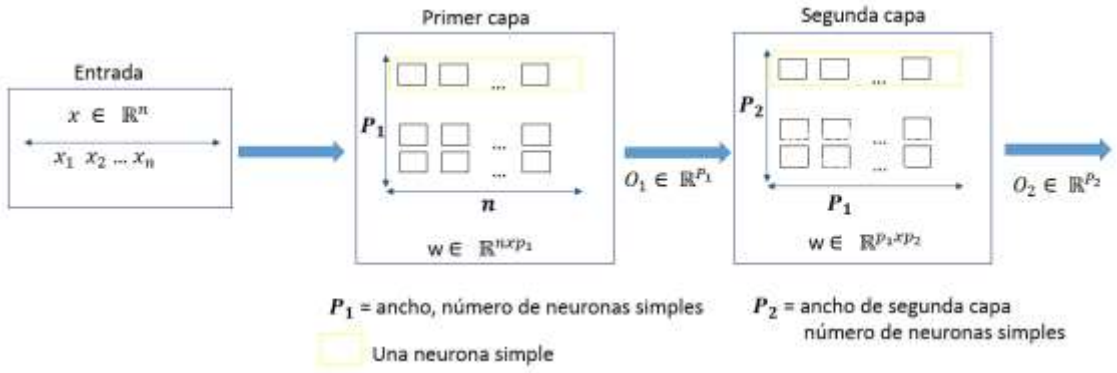

Figura 12: Entrada y salida de primera capa y segunda capa

La salida de la primera capa es $O_1 \in \mathbb{R}^{P_1}$ donde $O_i = f(\sum_{k=1}^{n} x_k\, w_k + b_i)$ aquí k es el índice de cada una de las entradas de las entradas y pesos, e i corresponde a las neuronas simples de la primera capa.

La entrada a la segunda capa será $O_1 \in \mathbb{R}^{P_1}$ que dará como salida $O_2 \in \mathbb{R}^{P_2}$ que

vectorizado tendrá la forma $f(o_1 w_2 + b_2)$.

2.5.3 Funciones de Activación

El propósito de una función de activación (también llamada función de transferencia o función de umbral) es determinar la salida del último nodo de una manera más dinámica, no lineal para poder extraer información compleja para representar y asignar funciones para alimentar la siguiente entrada de la siguiente capa, su importancia radica en la dependencia de la precisión en la predicción de la red neuronal teniendo como las funciones de activación no lineales como las más comúnmente usadas.

La articulación de una función de activación se da por medio del adecuamiento de un límite de amplitud de la unidad de salida dentro de un rango limitado, a este proceso se le llama función de aplastamiento pues “aplasta” la amplitud de la señal de salida dentro de un valor finito. (Sharma, 2017).

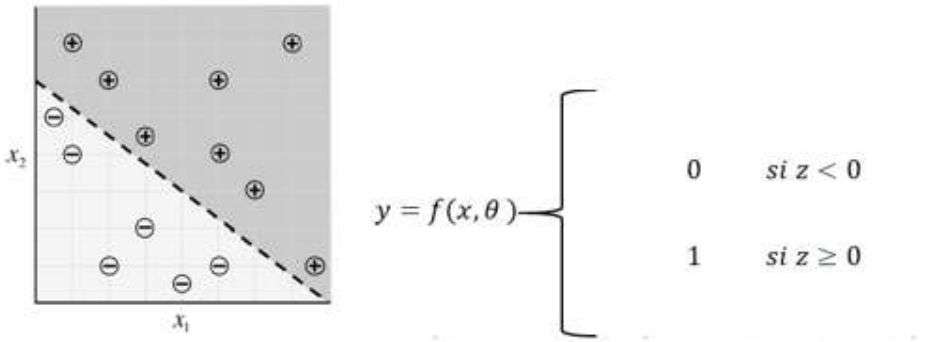

Figura 13: *Modelo de Perceptron Lineal*

Función de activación lineal

La *función de activación lineal* Figura 13 es directamente proporcional a la entrada, los datos de entrada se representarán mediante vectores de la forma (x_1, x_2) que indican sus coordenadas en el plano cartesiano como una simple función lineal polinomial de grado uno donde la función devolverá 0 ó 1 (por encima o por debajo de la línea) para saber si debe ser clasificado como "positivo" o "negativo". La línea o

clasificador se puede definir por:

$$z = f_1(x_1, \dots x_{n,}\, \theta) = \sum_{i=1}^{n} w_{ij}\, x_j + b \tag{3}$$

El resultado de esto se debe pasar a una función de activación no lineal para producir una salida y de 0 o 1, en éste caso, para dividir con una línea las entradas, supongamos los parámetros $\theta = [-24\ 3\ 4]^T$

$$y = f(x, \theta) \begin{cases} 0 & si\ 3x_1 + 4x_2 - 24 < 0 \\ 1 & si\ 3x_1 + 4x_2 - 24 \geq 0 \end{cases}$$

Función de activación para el ejemplo

Dado que esta es muy simple y de complejidad limitada no sirve para aprender, modelar y reconocer mapas complejos de datos, es necesario usar una función de activación no lineal. Sin embargo, si se necesita una neurona sin aplicar ningún umbral esta función es la correcta.

Función de activación sigmoidea

La *función de activación sigmoidea* Figura 14 es no lineal y es la más comúnmente usada en redes multicapa que se entrenan usando el algoritmo de retro propagación debido a que es diferenciable, transforma los valores de salida siempre positivos en un rango de [0,1], la función de activación está dada por $y = \frac{1}{1+ e^{-(wx+b)}}$ que se reduce a la función adyacente $y = \frac{1}{1+ e^{-x}}$. La función no es simétrica sobre cero es decir los signos de los valores de salida de las neuronas serán iguales, satura a 0 cuándo los valores de z son muy negativos y satura a 1 cuando z se vuelve muy positivo.

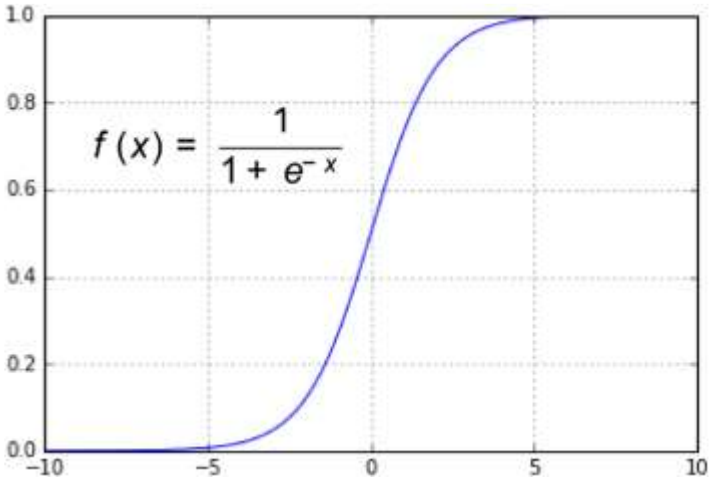

Figura 14: (Ketkar, Deep Learning with Python, 2017) Sigmoidea

Estos modelos de funciones de activación siguen siendo limitados porque las distribuciones de las entradas gráficamente se pueden presentar, tal que podrían no clasificarse con una línea y entonces usar funciones de activación más sofisticadas.

Función de activación tangente

La *función de activación tangente* Figura 15 es parecida a la función sigmoidea solo que es simétrica alrededor al origen como una *función de tangente hiperbólica* con el gradiente de la función más empinada, esto hace que los valores de salida tengan diferentes signos de capas anteriores que se alimentarán como entrada a la siguiente capa. Los valores de esta función se encuentran entre -1 y 1.

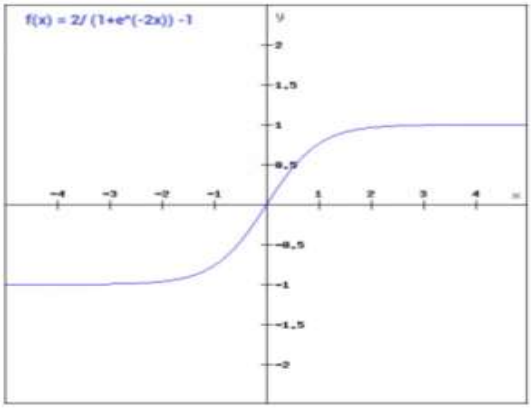

Figura 15: (Ketkar, Deep Learning with Python, 2017) tangente

Función de activación softmax

La *función de activación Softmax* es parecida a la función de activación sigmoidea, pero esta puede ser usada para problemas de clasificación multiclase es decir para las probabilidades de los puntos de datos de una clase en particular. Esta función es expresada de la siguiente forma:

$$y = \frac{e^{zj}}{\sum_{k=1}^{K} e^{zk}} \tag{4}$$

Función de activación ReLU

La neurona de unidad lineal de rectificación (*ReLU*) Figura 16 utiliza un tipo diferente de no linealidad. Utiliza la función $f(x) = max(0, wx + b)$ considerando como función de activación subyacente $f(x) = max(0, x)$. La ventaja de usar esta función es que no todas las neuronas son activadas al mismo tiempo, una neurona va a ser desactivada solo cuando la salida de una transformación lineal es cero, solo cierto número de neuronas es activada a la vez, esto la hace más eficiente y recomendable otras funciones de activación.

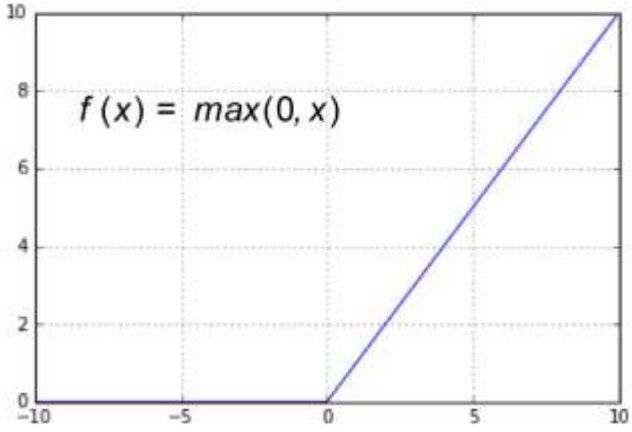

Figura 16: (Ketkar, Deep Learning with Python, 2017) ReLU

Hay diversas funciones de activación y su uso depende del número de capas de la red neuronal, método de entrenamiento y el acoplamiento de los hiperparámetros, no hay una regla o un manual para escoger la función adecuada depende mucho de la tarea que se quiere lograr. A pesar de ello Sharma, S (2017) desglosa ventajas y desventajas del uso de las funciones:

- Para problemas de clasificación el uso combinado de *funciones de activación sigmoidea* es adecuado.

- *Funciones sigmoidea y tangente* no son recomendables con el uso de gradientes que comprenden cero.

- La *función de activación ReLu* funciona mejor, ya que tienen bajo costo computacional, mejor propagación del gradiente para valores mayores a 1| y es la más comúnmente usada pero debe de ser usada en capas ocultas y no en la capa de salida.

Cuando se procesa una imagen, cada capa de convolución debe capturar algún patrón en la imagen y pasarla en la siguiente capa de convolución. Los valores negativos no son importantes en el procesamiento de imágenes y, por lo tanto, se establecen en 0. Pero los valores positivos después de la convolución deben pasar a la siguiente capa. Es por eso que *ReLU* se está utilizando como una función de activación. Si utilizamos *sigmoide* o *tanh*, la información se pierde ya que ambas funciones modificarán las entradas a un rango muy cerrado

2.5.4 Funciones de pérdida

Se requiere evaluar la salida de la red neuronal frente a los datos etiquetados o predicción esperada, para ello se deben tener ciertas consideraciones:

Los datos tienen la forma $D = \{ (x_1, y_1), (x_2, y_2) \dots (x_n, y_n) \}$ donde $x \in \mathbb{R}^n$ y $y \in \{0,1\}$, o bien vector de etiquetas asignadas si se trata de clasificación. Para un solo punto de datos, se puede calcular la salida de la red neuronal, que se denota como $\hat{y}$. Ahora se requiere aplicar la función de pérdida para calcular qué tan buena es la

predicción de nuestra red neuronal ŷ en comparación con y, la salida real esperada. La función de pérdida mide el desacuerdo entre ŷ y y que se denota por l, Figura 17. Hay una serie de funciones de pérdida apropiadas para la tarea en cuestión: clasificación binaria, multiclasificación o regresión. Una función de pérdida normalmente calcula el desacuerdo entre ŷ y y sobre un número de puntos de datos en lugar de un solo punto de datos.

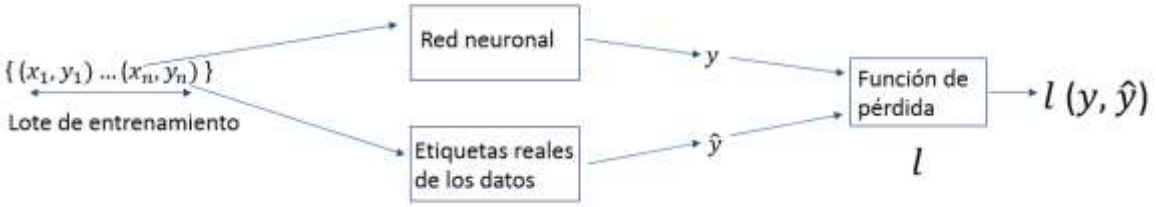

Figura 17: Función de pérdida

Existen elementos claves en las funciones de pérdida y la idoneidad de una función de pérdida dado un problema.

Entropía cruzada binaria

Esta dada por la expresión

$$-\sum_{i=1}^{n} y_i Log f(x_i, \theta) + (1 - y_i)\, log(1 - f(x_i, \theta)) \tag{5}$$

Es la función de pérdida recomendada para la clasificación binaria. Esta función de pérdida debería utilizarse normalmente cuando la red neuronal está diseñada para predecir la probabilidad del resultado. En tales casos, la capa de salida tiene una sola unidad con un sigmoide adecuado como función de activación.

Función de entropía cruzada

Está dada por la expresión

$$-\sum_{i=1}^{n} y_i Logf(x_i, \theta) \tag{6}$$

Es la función de pérdida recomendada para la clasificación múltiple. Esta función de pérdida normalmente se debe utilizar con la red neuronal y está diseñada para predecir la probabilidad de los resultados de cada una de las clases. En tales casos, la capa de salida tiene unidades softmax (una para cada clase).

Función de pérdida al cuadrado

Está dada por

$$-\sum_{i=1}^{n} (y - \hat{y})^2 \tag{7}$$

Debe usarse para problemas de regresión. La capa de salida en este caso tendrá una sola unidad.

2.5.5 Funciones de optimización

Los optimizadores toman el valor de la función de pérdida como su entrada, y con ello elige como modificar los pesos de la red para que hacer que se acerquen los resultados a los valores establecidos como reales, es decir, actualizan los parámetros de peso *w* para minimizar la *función de pérdida*.

ADAptive GRADient algorithm, Adagrad

Un método que introduce el aprendizaje adaptativo, intenta adaptar la tasa de aprendizaje global usando un histórico de gradientes. Esta tasa de aprendizaje se escala inversamente con respecto a la raíz cuadrada de la suma de los cuadrados de todos los gradientes históricos del parámetro. El concepto geométrico es que, si en el descenso de gradiente se avanza en la dirección de la mayor pendiente en cada momento, pudiendo esa pendiente no apuntar directamente al mínimo global, en *Adagrad* se intenta que la dirección de

descenso, aunque no sea la máxima, apunte más en la dirección del mínimo. Tiene, el problema de una tasa de aprendizaje menor paulatinamente.

Root Mean Square prop, RMSprop

Utiliza una media móvil de los cuadrados del gradiente y normaliza ese valor empleando para ello las magnitudes recientes de los gradientes anteriores. La normalización se realiza, evidentemente, por que si no, el hecho de elevar al cuadrado hace que, caso de no normalizar, tengamos un valor de gradiente exageradamente alto (el cuadrado, en concreto).

ADAptive Moment stimation, Adam

Viene a ser una combinación de *momentum* y *RMSprop*. Utiliza una media móvil exponencial de los gradientes para ajustar las tasas de aprendizaje. Es un algoritmo computacionalmente eficiente y que usa poca memoria. Y es uno de los optimizadores más populares hoy día.

Adadelta

Una extensión de *Adagrad* más robusta que usa una ventana móvil de actualizaciones de gradientes.

Adamax

Una variante algo sutil de *Adam*, en que en lugar de un momento de segundo orden, se utiliza un momento de orden infinito.

2.6 Redes Neuronales Convolucionales

Una *Red Neuronal Convolucional (CNNs)* son redes neuronales con *Aprendizaje Profundo DL* ofrecen resultados óptimos en el reconocimiento de imagenes impactando profundamente en el área de visión por computadora (Torres, Deep Learning, Introducción Práctica con Keras, 2018). Están formadas por neuronas que tienen parámetros en forma de *pesos* y *bias*, como las redes neuronales, y pueden aprender. Pero un rasgo diferencial de las *CNN* es que hacen la suposición explícita de que las entradas son imágenes, cosa que

permite codificar ciertas propiedades en la arquitectura para reconocer elementos concretos en las imágenes.

En la imagen de un rostro algunas características pueden estar escondidas, entonces antes se debe saber cómo es una oreja o una nariz para saber si están en una imagen; es decir, previamente debemos identificar líneas, bordes, texturas o formas que sean similares a las que contiene las orejas o narices que se han visto antes. Y esto es lo que las capas de una Red Neuronal Convolucional tienen encomendado hacer, va marcando características de una cara Figura 20.

Pero identificar estos elementos no es suficiente para poder decir que algo es una cara. Además, se debe poder identificar cómo las partes de una cara se encuentran entre sí, tamaños relativos, etc., de lo contrario, la cara no se parecería a lo estándar. La siguiente Figura 18 es una idea de como aprenden las capas.

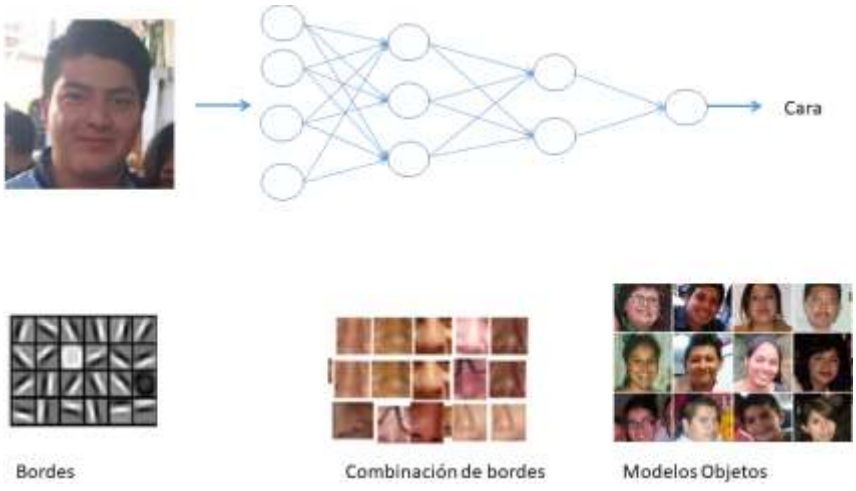

Figura 18: Descripción intuitiva de aprendizaje de una *CNN*

La idea que se quiere dar con este ejemplo visual es que, en realidad, en una CNN cada capa va aprendiendo diferentes niveles de abstracción. Con redes con muchas capas se pueden conseguir identificar estructuras más complejas en los datos de entrada.

El nombre de convolucional es porque emplea una operación matemática. La operación llamada convolucional que es un tipo especializado de operación lineal, Las redes convolucionales son simplemente redes neuronales que utilizan la convolución

en lugar de la multiplicación general de matrices en al menos una de sus capas (GoodFellow, bengio, & Courvile, Deep Learning, 2016). Una capa convolucional detecta características o rasgos visuales en las imágenes como aristas, líneas, gotas de color, lo que sea; una vez aprendida una característica en un punto concreto de la imagen la puede reconocer después en cualquier parte de la misma Figura 21.

Las capas convolucionales pueden aprender jerarquías espaciales de patrones preservando relaciones espaciales. Por ejemplo, una primera capa convolucional puede aprender elementos básicos como aristas, y una segunda capa convolucional puede aprender patrones compuestos de elementos básicos aprendidos en la capa anterior. De manera sucesiva, hasta ir aprendiendo patrones complicados.

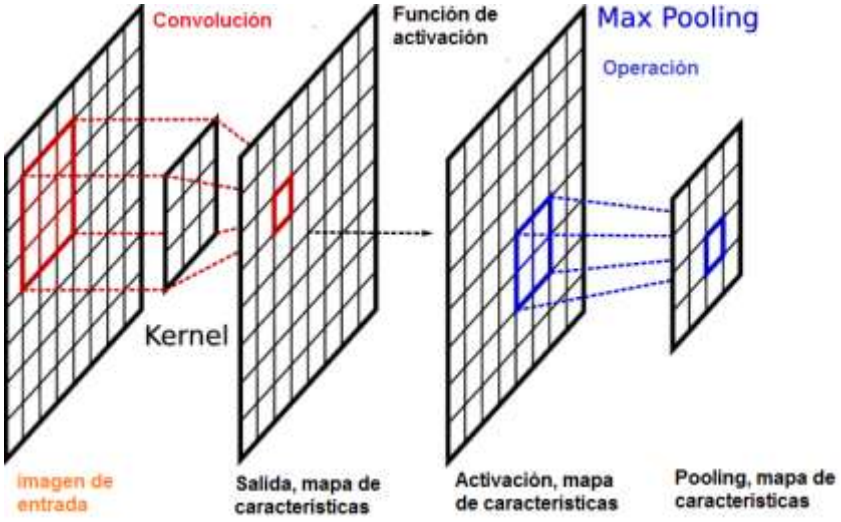

Figura 19: (Georgiou, Liu, Chen, & Lew, 2019)

Las capas convoluciones operan sobre tensores (vectores) de 3D, llamados mapas de características (feature maps), con dos ejes espaciales de altura y anchura (height y width), además de un eje de canal (channels) también llamado profundidad (depth). Para una imagen de color RGB, la dimensión del eje de profundidad es 3, pues la imagen tiene tres canales: rojo, verde y azul (red, green y blue). Unas primeras capas de neuronas ocultas conectadas a las neuronas de la capa de entrada realizarán las operaciones convolucionales, No se conectan todas las neuronas de entrada con todas las neuronas de este primer nivel de neuronas ocultas, solo se hace por pequeñas

zonas localizadas del espacio de las neuronas de entrada que almacenan los píxeles de la imagen.

Se empieza con la ventana en la esquina arriba-izquierda de la imagen, y esto le da la información necesaria a la primera neurona de la capa oculta. Después, se desliza la ventana una posición hacia la derecha para conectar las neuronas de la capa de entrada incluidas en esta ventana con la segunda neurona de la capa oculta. Y así, sucesivamente, se va recorriendo todo el espacio de la capa de entrada, de izquierda a derecha y de arriba abajo **¡Error! No se encuentra el origen de la referencia.**Figura 20.

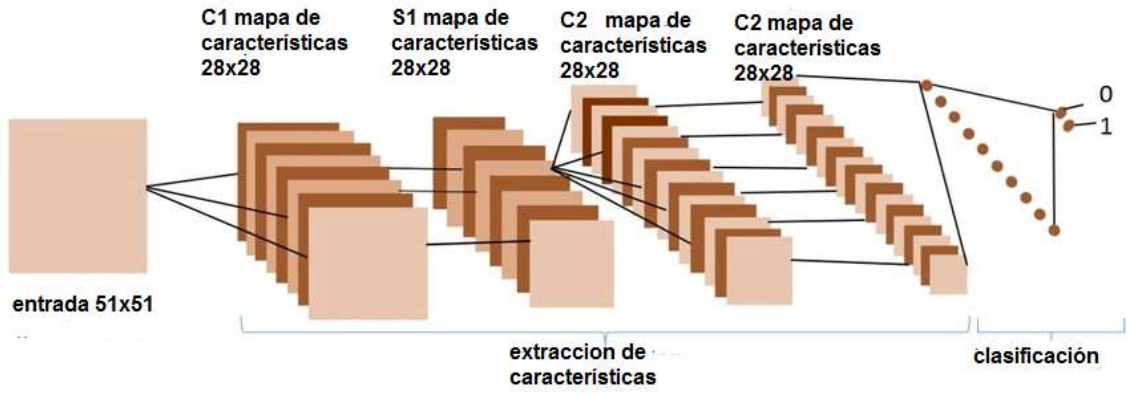

Figura 20: Ejemplo de CNN, proceso de traslado de información

2.6.1 Componentes de Arquitectura CNN

Una *CNN* se compone de varios bloques de construcción básicos, llamados capas *CNN*, se considera que algunas de estas capas implementan funcionalidades básicas como normalización, agrupación, convolución y capas completamente conectadas. También existen capas más complejas que están compuestas por múltiples bloques de construcción.

Capas de Preprocesamiento

- Los pasos que se utilizan consisten en lo siguiente:
-
- Resta de Medias: los datos de entrada, tanto de entrenamiento como de prueba, están centrados en cero restando la media calculada en todo el conjunto de

entrenamiento. Dadas N imágenes de entrenamiento, cada una denotada por $\mathbb{R}^{hxwxc}$, podemos denotar el paso de resta media de la siguiente manera

- $$x' = x - \hat{x} \qquad \text{Donde } \hat{x} = \frac{1}{N}\sum_1^N X_i \tag{8}$$

- Normalización: De igual manera los datos de entrada, tanto de entrenamiento y de prueba son divididos por la desviación estandar de cada dimensión de entrada, calculada en el conjunto de entrenamiento para normalizar la desviación estándar a un valor unitario. Puede ser representado como:
  - $$x'' = \frac{x'}{\sqrt{\frac{\sum_1^N (x-\hat{x})^2}{N-1}}} \tag{9}$$

- Blanqueamiento PCA: el objetivo es reducir las correlaciones entre las diferentes dimensiones de datos al normalizarlas independientemente. Este enfoque comienza con los datos centrados en cero y calcula la matriz de covarianza que codifica la correlación entre las dimensiones de los datos. Esta matriz de covarianza se descompone a través del algoritmo de Descomposición de Valor Singular (SVD) y los datos se correlacionan proyectándolos en los vectores propios encontrados a través de SVD. Después, cada dimensión se divide por su valor propio correspondiente para normalizar todas las dimensiones respectivas en el espacio de datos.
- Normalización de contraste local: Obtiene características más destacadas. Genera un vecindario local para cada píxel, por ejemplo, para un radio unitario, se seleccionan ocho píxeles vecinos. Posteriormente, el píxel está centrado en cero con la media calculada utilizando sus propios valores de píxel y vecinos. Del mismo modo, el píxel también se normaliza con una desviación estándar propia y vecina. Valores de píxeles (solo si la desviación estándar es mayor que uno). El valor de píxel resultante se utiliza para otros cálculos.

Capas Convolucionales

- Las capas de convolución aplican un kernel, de convolución al conjunto de

mapas de características, El tamaño de éste kernel esta definido por la arquitectura de la red, y sus valores son aprendidos durante el entrenamiento. Una convolución es:

$$x_l'^{(i,j,k)} = \sum_{0}^{m}\sum_{0}^{n} W_{q,p}\, Z_l^{\left(\left(i+p-\frac{m}{2}\right),\left(j'k-\frac{n}{2}\right),k\right)} \quad (10)$$

Donde (i,j) es un pixel del mapa de caracteres con dimensiones (W, H

- El objetivo de la convolución es para filtrar la entrada e identificar patrones locales en diversas partes de la imagen. Lo hace aplicando una ventana deslizante a través de todo el mapa de entrada. Se listan las diferentes capas de convolución**:**
- Convolución 2D
- Convolución 2D stride
- Convolución 2D transpuesta
- Convolución 2D Dilatada
- Convolución 2D Depthwise

Capas Pooling

También llamadas capas de redimensionamietnto, es una forma de agregar información y aumentar el radio de recepción de una neurona reduciendo el mapa de características, sin embargo se amplia la información de espacios más grandes. Las capas pooling tienen la caracteristica de reducir el tamaño de las imágenes de entrada con distintas operaciones: se tienen las siguientes capas pooling principales:

- Max Pool, Con esta capa se obtiene el valor máximo de una ventana de tamaño dado, la reducción dependerá del stride
- Average Pool, Trabaja igual que Max Pool sin embargo, se obtiene la media de los valores de los pixels.
- Convolución 2d Depthwise

Capas de Activación

La función de activación determina el estado de activación actual de la neurona. Se realiza porque se quiere añadir elementos no lineales al modelo de red. Las funciones de activación más communes son sigmoide, tangente hiperbólica, rectificaciones lineales (ReLu), Leaky ReLu o ELU. Tienen las siguientes formulaciones matemáticas:

$$f(x) = \frac{1}{1 + e^{x}} \tag{11}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{12}$$

$$f(x) = Max(x, 0) \tag{13}$$

$$f(x) = x\ si\ x \geq 0\ ;\ \alpha(e^{x} - 1)\ de\ otra\ forma \tag{14}$$

$$f(x) = x\ si\ x > 0\ ;\ \alpha x\ de\ otra\ forma \tag{15}$$

La tangente hiperbólica y sigmoide son clásicas, sin embargo, su costo computacional es alto, para contrarrestar se tiene ReLU que tiene bajo costo computacional, invariabilidad a la escala, mejor propagación del gradiente para valores mayores a 1.

Cuando se utiliza ReLU se presentan problemas de neuronas inútiles ya que no se propaga el gradiente para valores menores a 0, por lo que podría no haber activación independiente de la entrada. Para contrarrestar esto, se puede usar leaky ReLu que si permite una propagación del gradiente para valores menores a 0.

2.6.2 Entrenamiento de una red neuronal

Se representa por θ la colección de todos los pesos y términos de bias de todas las capas de la red. Se asume que θ se ha inicializado con valores aleatorios. Se expresa por f_{NN} la función general que representa la red neuronal. Se puede tomar un solo punto de datos y calcular la salida de la red neuronal $\hat{y}$. También se calcula el desacuerdo con la salida real y usando la función de pérdida $l(\hat{y}$, y) que es $l(f_{NN}(x, \theta)$

, y). Se calcula el gradiente de esta función de pérdida y se escribe $\nabla l(f_{NN}(x, \theta)$.

Se actualiza θ usando el descenso más pronunciado como $\theta_s = \theta_{s-1} - \alpha . l(f_{NN}(x, \theta)$, y), donde s representa un solo paso, se toman muchos de esos pasos en diferentes puntos de datos en nuestro entrenamiento se repite una y otra vez hasta obtener un valor razonablemente bueno para $l(f_{NN}(x, \theta)$, y).

2.6.3 Arquitecturas de CNN

2.7 Métricas de evaluación

Las métricas son útiles para cuantificar la diferencia entre desempeño deseado y línea de base, desempeño deseado y desempeño actual, medir el progreso a lo largo del tiempo.

Para la medir la eficiencia de un modelo de *CNN*, la especificidad, sensitividad y precisión son estadísticas utilizadas para cuantificar que tan buena y confiable es una clasificación, es decir, ayuda a conocer cuál clasificador es mejor y estima el error esperado. Los términos que se usan comúnmente en las métricas de sensibilidad, especificidad y precisión son las mismas usadas en una matriz de confusión: verdaderos positivos (*TP*), verdaderos negativos (*TN*), falsos negativos (*FN*) y falsos positivos (*FP*).

Las siglas TP, FN, FP y TN significan:

- *TP* = verdadero positivo, el número de casos positivos que se identifican correctamente como positivos.
- *FN* = falso negativo, el número de casos positivos que se clasifican erróneamente como casos negativos.
- *FP* = falso positivo, el número de casos negativos que se clasifican incorrectamente como casos positivos.
- *TN* = verdadero negativo, el número de casos negativos que se clasifican correctamente como casos negativos.

Tabla 2: Matriz de Confusión

	Prediccion		
Actual		Positivo	Negativo

	Positivo	TP	FN
	Negativo	FN	TN

2.7.1 Exactitud (*Accuracy)*

Se refiere a la proporción del número total de predicciones, de la clasificación, que fueron correctas. Esta no es una medida válida del rendimiento del modelo cuando se tienen un conjunto de datos desbalanceado.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Tabla 3: Exactitud basada en la matriz de confusión.

	Prediccion		
Actual		Positivo	Negativo
	Positivo	TP	FN
	Negativo	FN	TN

2.7.2 Precisión

Se evalúan los datos por su desempeño en predicciones positivas.

$$Precisión = \frac{TP}{TP + FP}$$

Tabla 4: Precisión basada en la matriz de confusión.

	Prediccion		
Actual		Positivo	Negativo
	Positivo	TP	FN
	Negativo	FN	TN

2.7.3 Sensitividad (*recall*)

Se refiere la proporción de casos positivos que fueron correctamente identificados divido por el número de positivos.

$$recall = \frac{TP}{TP + FN}$$

Tabla 5: Sensitividad basada en la matriz de confusión

	Prediccion		
Actual		Positivo	Negativo
	Positivo	TP	FN
	Negativo	FN	TN

2.7.4 Especificidad

Se refiere a la proporción de casos negativos que han sido incorrectamente clasificados como positivos:

$$especificidad = \frac{TN}{FP + TN}$$

Tabla 6: Especificidad basada en la matriz de confusión.

	Prediccion		
Actual		Positivo	Negativo
	Positivo	TP	FN
	Negativo	FN	TN

2.7.5 *ROC*

Muestra la proporción de verdaderos positivos contra la proporción de falsos positivos. En otras palabras es la curva de características operativas del receptor, es la tasa de falsos positivos (eje x) frente a la tasa de verdaderos positivos (eje y) para varios valores de umbral candidatos diferentes entre 0,0 y 1,0. Dicho de otra manera, traza la tasa de falsas alarmas frente a la tasa de aciertos. Describe qué tan bueno es el modelo para predecir la clase positiva cuando el resultado real es positivo

2.7.6 F-measure

Promedio ponderado de precisión y sensitividad, involucra ambas métricas.

$$F - measure = \frac{2 * precisión * sentitividad}{precisión + sensitividad}$$

Capítulo 3
Desarrollo de Metodología

La finalidad de la visión artificial es la de extraer información del mundo físico mediante imágenes con el uso de dispositivos digitales como cámaras y su procesamieto a través de computadoras, se trata de cuantificar detalles del mundo real como: el brillo, el color, la forma, que pueden provenir de imágenes estáticas, tridimensionales o de vídeo. A lo largo de los años se han estudiado muchos fundamentos que han dado forma a la ciencia de la visión artificial y gracias al avance de nuevos computadores con mayor capacidad de procesamiento se pueden realizar operaciones matriciales de imágenes en tiempo real.

La metodología del presente trabajo consiste en hacer uso de la visión artificial en combinación con algoritmos de clasificación para obtener resultados certeros acerca de la madurez a la que pertenece una imagen de fruto de durazno; pero también se implementa una red neuronal convolucional, con ello se podrá evaluar los resultados obtenidos en ambas propuestas.

3.1 Estrategía de Clasificación

Para llevar a cabo la primera propuesta construcción de clasificador se debe tomar en cuenta la siguiente secuencia Para la primera parte se utiliza el siguiente software de visión artificial y lenguaje multipropósitos:

- Anaconda (framework)

  Es un gestor de entornos de código abierto, es una plataforma en donde los científicos de datos comparten trabajo y datasets, tienen un ambiente gráfico que permite trabajar en cloud o bien en escritorio con machine learning, usando python, jupiter u otros editores.

- Jupiter

  Jupyter es un entorno de desarrollo interactivo basado en web para cuadernos, código y datos de Jupyter. JupyterLab es flexible: configure y organice la interfaz de usuario para admitir una amplia gama de flujos de trabajo en ciencia

de datos, informática científica y aprendizaje automático. JupyterLab es extensible y modular: escribe complementos que agregan nuevos componentes y se integran con los existentes

- Python(3.7)

Python fue creado a principios de la década de 1990 por Guido van Rossum en Stichting Mathematisch Centrum en los Países Bajos como sucesor de un idioma llamado ABC. En 1995, Guido continuó su trabajo en Python en la Corporación para Iniciativas de Investigación Nacional en Reston, Virginia, donde lanzó varias versiones del software. En mayo de 2000, Guido y el equipo de desarrollo central de Python se trasladaron a BeOpen.com para formar el equipo de BeOpen PythonLabs. En octubre del mismo año, el equipo de PythonLabs se trasladó a Digital Creations En 2001, se formó la Python Software Foundation.

- Keras

Keras es una API de aprendizaje profundo escrita en Python, que se ejecuta sobre la plataforma de aprendizaje automático

Los pasos que a realizados para operar con las imágenes y los algoritmos de clasificación, son los mostrados en la siguiente figura:

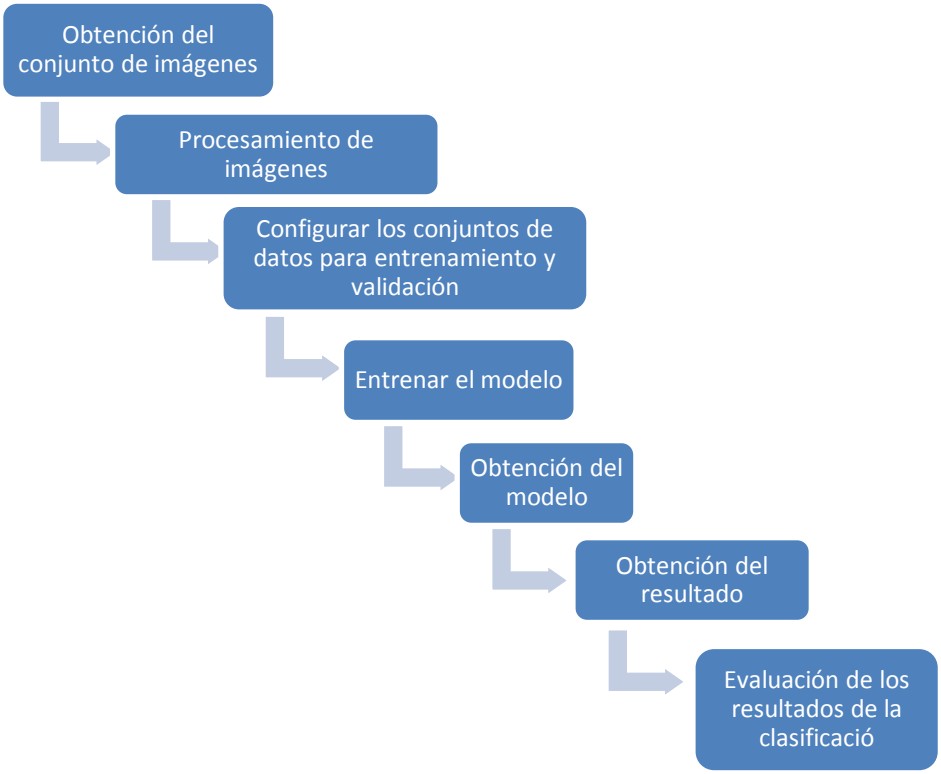

Ilustración 1: Etapas para la clasificación usando una red neruronal convolucional

3.2 Obtención del conjunto de datos

Las imágenes fueron tomadas con una cámara profesional Nikon d3500 de 24.2 megapíxeles con los lentes AF-P DX Nikkor 18 mm - 55 mm y el AF-P DX Nikkor 70 mm - 300 mm. Luego se recortaron hasta dejar el área de interés. Cada imagen muestra un durazno completo. Se recortó la imagen hasta dejar el área de la fruta para considerar únicamente la textura. Se hicieron conjuntos de imágenes con Duraznos maduros e incluso sanos y dañados. El conjunto de datos se denomina PEACH. Su recolección fue en las zonas de San Juan Teotihuacán, Tequexquinahuac, San Miguel Tlaixpan, San Simón, y árboles frutales de casa en la zona de Texcoco, Estado de México. Los grupos de imágenes se pueden ver en la Figura 21:

.

Figura 21: El conjunto de datos de imágenes digitales de frutos
a) Maduros e inmaduros, b) maduros e inmaduros (textura) y c) sanos y dañados

3.3 Preprocesamiento del Conjunto de Datos

Los conjuntos de imágenes de duraznos maduros, inmaduros, sanos y dañados para los tres escenarios se detallan en la Tabla 7.

Tabla 7: *Cantidad de imágenes para cada conjunto de datos de imágenes digitales*.

Durazno complete	Duraznos maduros e inmaduros	Área de textura	Cantidad	Durazno completo	Cantidad
Maduros entrenamiento	**160**	Maduros entrenamiento	**200**	Sanos entrenamiento	**320**
Maduros validación	**40**	Maduros validación	**40**	Sanos entrenamiento	**160**
Inmaduros entrenamiento	**128**	Inmaduros entrenamiento	**100**	Dañados entrenamiento	**80**
Inmaduros Validación	**32**	Inmaduros validación	**20**	Dañados validación	**40**
Total	**360**	Total	**360**	Total	**600**

Los conjuntos de datos de imágenes digitales han sido clasificados considerando que el corte del durazno implica, detectar el fruto maduro y cortar. Sin embargo, cuando el fruto está dañado debería cortarse, pero separarse de los frutos sanos y maduros. Los frutos no maduros deberían de no cortarse hasta que alcance su madurez.

3.4 Modelo Propuesto

La *CNN* utilizada tiene tres capas con filtros de (2 x 2) y de (3 x 3). En las primeras dos capas con acción pooling. La *CNN* tiene como entrada las imágenes de los *data set*. En la salida, se obtienen los datos de la clasificación en dos o tres categorías. Se pueden ver los detalles en la Figura 22.

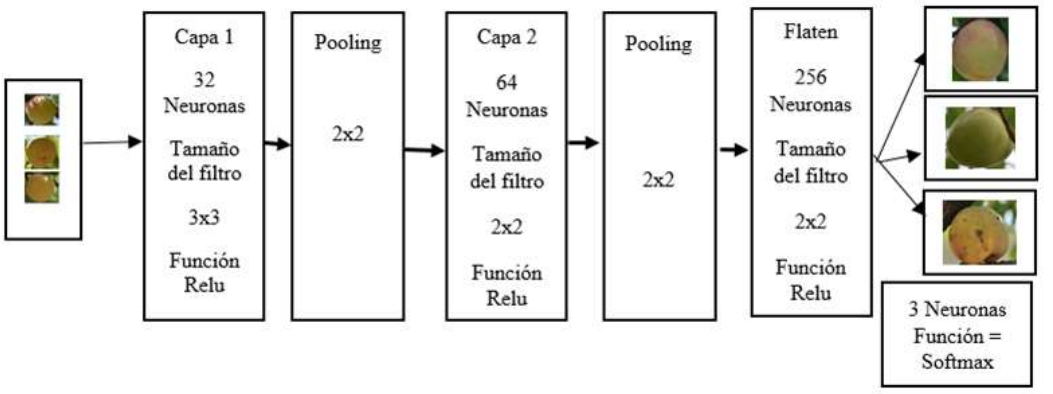

Figura 22: Arquitectura de la *CNN* usada para el entrenamiento.

La *CNN* en la capa uno: con 32 neuronas de (3 x 3) y se aplica la función *Relu*. En seguida, la capa dos: con 64 neuronas de (2 x 2) y la función *Relu*. Mientras que la Capa *Flaten*: Con 256 neuronas, un filtro de (2 x 2) y la *función Relu*. Finalmente, la capa *Dropout*: con tres neuronas y la función *softmax* lo que quiere decir que la categoría se clasifica con el valor más alto de similitud encontrado durante el reconocimiento.

En el código en Python fue necesario implementar un conjunti de líneas para definir los parámetros. Ver el código 1.

Código 1. Parámetros para la CNN.

```
data_entrenamiento = 'C:/DMIT/Train'
data_validacion    = 'C:/DMIT/Valid'
epocas             = 8
altura, longitud = 100, 100
batch_size         = 2
pasos              = 3000
validacion_pasos = 600
filtrosConv1   = 32
filtrosConv2   = 64
tamano_filtro1 = (3, 3)
tamano_filtro2 = (2, 2)
tamano_pool    = (2, 2)
clases         = 2
lr             = 0.0004
```

Se pueden obervar los directorios donde están las imágenes para el entrenamiento y la validación. En este caso *epocas*, indica las veces que se procesan las imágenes para el entrenamiento. Igualmente, podemos ver la *altura* y *longitud* de las imágenes, es decir, se redimensionan las imágenes antes de ingresar a la red. El *batch_size* se encarga de indicar cuantas imágenes se cargan para su proceso, en este caso fueron 2. Los *pasos* y *validación_pasos* se refieren a las veces que se hacen pasar los filtros por la imagen. También se define la cantidad de filtros y el tamaño de cada filtro. Finalmente, tenemos la cantidad de clases en este caso la red arroja dos clases en la salida. Finalmente tenemos la tasa de aprendizaje, en este caso *lr* es 0.0004.

Es necesario preparar las imágenes para la entrada a la red. Por lo que se codificaron las siguientes instrucciones. Ver el código 2.

Código 2. Preparación de las imágenes para la entrada a la red.

```
entrenamiento_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

validacion_generador = test_datagen.flow_from_directory(
    data_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')
```

Los valores de la matriz se dividen entre 255 con el fin de normalizar los valores. De esa forma todos los valores quedan entre 0 y 1. Se definen los conjuntos de datos de entrada con los parámetros correspondientes.

Después, se codificaron las capas de la red convolucional con los parámetros previamente definidos. Ver el código 3.

Código 3. Capas de la red neuronal convolucional.

```
cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, padding ="same", input_shape=(altura,
longitud, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation='softmax'))

cnn.compile(loss='categorical_crossentropy',
            optimizer=optimizers.Adam(lr=lr),
            metrics=['accuracy'])

cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epocas,
    validation_data=validacion_generador,

    validation_steps=validacion_pasos)
```

Se ha usado la arquitectura *secuential*, en la cual se ha agregado una capa convolucional y una capa de *pooling*, luego otra capa intermedia convolucional y una más de *pooling* y una capa de salida tipo *flaten*. En la capa de salida de ha usado la función *relu*, para obtener valores entre 0 y 1 y eliminar valores negativos. Para lograr la categorización, de ha usado la finción *softmax* que permite definir la categoría.

Una vez realizado el entrenamiento se ha guardado el conocimiento aprendido en un modelo en un archivo. Ver el código 4.

Código 4. Archivos de salida del entrenamiento.

```
target_dir = './modeloDMIT/'

if not os.path.exists(target_dir):
  os.mkdir(target_dir)

cnn.save        ('C://MODELS/modeloDMIT/modeloDMIT.h5')
cnn.save_weights('C:/MODELS/modeloDMIT/pesosDMIT.h5')

print ("Los resultados han sido guardados...")
```

En este caso se guardan dos archivos con la extensión *.h5*. Se gusrda el mpodeilo usado así como los *pesos* ajustados después del entrenamiento.

Al ejecutar el programa de entrenamiento se pueden observar los detalles como se puede ver en el código 5.

Código 5. Vista de la ejecución del entrenamiento.

```
Found 300 images belonging to 2 classes.
Found 60 images belonging to 2 classes.
Epoch 1/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.5203 - acc: 0.7618Epoch 1/8
3000/3000 [==============================] - 498s 166ms/step - loss: 0.5203 - acc: 0.7618 -
val_loss: 0.2391 - val_acc: 0.88332:15 - loss: - ETA: 2:13 - loss: 0.2 - ETA: 2:12 - loss: 0.2
- ETA: 2:12 - loss: - ETA
Epoch 2/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.3552 - acc: 0.8396Epoch 1/8
3000/3000 [==============================] - 499s 166ms/step - loss: 0.3552 - acc: 0.8397 -
val_loss: 0.2760 - val_acc: 0.9000
Epoch 3/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.2877 - acc: 0.8796Epoch 1/8
3000/3000 [==============================] - 503s 168ms/step - loss: 0.2877 - acc: 0.8797 -
val_loss: 0.4408 - val_acc: 0.8500
Epoch 4/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.2398 - acc: 0.9018Epoch 1/8
3000/3000 [==============================] - 502s 167ms/step - loss: 0.2397 - acc: 0.9018 -
val_loss: 0.4007 - val_acc: 0.8500
Epoch 5/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.2047 - acc: 0.9183Epoch 1/8
3000/3000 [==============================] - 504s 168ms/step - loss: 0.2046 - acc: 0.9183 -
val_loss: 0.6002 - val_acc: 0.8000
Epoch 6/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.1707 - acc: 0.9325Epoch 1/8
3000/3000 [==============================] - 504s 168ms/step - loss: 0.1707 - acc: 0.9325 -
val_loss: 1.2071 - val_acc: 0.7833
Epoch 7/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.1615 - acc: 0.9318Epoch 1/8
3000/3000 [==============================] - 504s 168ms/step - loss: 0.1614 - acc: 0.9318 -
val loss: 0.7623 - val acc: 0.7833
Epoch 8/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.1307 - acc: 0.9493Epoch 1/8
3000/3000 [==============================] - 509s 170ms/step - loss: 0.1306 - acc: 0.9493 -
val_loss: 0.8037 - val_acc: 0.8667
```

En la ejecución de puede observer que se procesan 360 imágenes. Que cadad *época* tarda 500 segundos en promedio. Entonces, la ejecución tarda unos 4000 segundos, es decir una hora y seis minutos. Al final podemos ver al valor de *accuracy* obtenido. Es necesario hacer varias ejecuciones con diferentes parámetros hasta obtener los valores más óptimos.

Después del entrenamiento se requiere otro programa para hacer las predicciones de clasificaciones de otro conjunto de imágenes y conocer el desempeño del modelo del entreanamiento de la red. Lo primero es cargar los archivo y las imágenes para hacer la predicción.Ver el código 6.

Código 6. Archivos del modelo y pesos para la predicción.

```
longitud, altura = 100, 100

modelo =       'C:/MODELS/modeloDMIT/modeloDMIT.h5'
pesos_modelo = 'C:/MODELS/modeloDMIT/pesosDMIT.h5'

cnn = load_model(modelo)
cnn.load_weights(pesos_modelo)
```

En este caso, se redimensionan las imágenes a 100 por 100 pixeles. Luego, se cargan los archivos del modelo y los pesos.

Para hacer la predicción de una nueva imagen se usó el código 7.

Código 7. Función principal para la predicción.

```
def predict(file):
  x = load_img(file, target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict(x)
  result = array[0]
  answer = np.argmax(result)
  if answer == 0:
    print("pred: Peach")
  elif answer == 1:
    print("pred: Peach")

  return answer
```

Se puede observar que recibe un archivo, por lo que esta función de ejecutará tantas imágenes se utilicen en la prueba. La predicción será 0 o 1.

Como ya se ha definido, cada imagen cuenta con una etiqueta indicando la categoría, mientras que las nuevas imágenes no cuentan conellas, pero se les debe asignar, y luego comparar para ver si la predcicción es aceratada. Ver el código 8.

Código 8. Predicciones y contadores de valores de cada vector.

```
contadorPeach=0;
for dir_pathPeach, dir_namesPeach, file_namesPeach in walk('C:/TEST/TESTDMIT/TInmat/'):
    for fnPeach in file_namesPeach:
        if fnPeach[-3:] == 'JPG':
            contadorPeach= contadorPeach + 1

print("ContadorPeach: " + str(contadorPeach))
print(file_namesPeach)
print(file_namesPeach[0])
x=0

contadorPeach=0
for dir_pathPeach, dir_namesPeach, file_namesPeach in walk('C:/TEST/TESTDMIT/TMat/'):
    for fnPeach in file_namesPeach:
        if fnPeach[-3:] == 'JPG':
            contadorPeach= contadorPeach + 1

print("ContadorPeach: " + str(contadorPeach))
print(file_namesPeach)
print(file_namesPeach[0])

y=0
```

En este caso, tenemos dos conjuntos, duraznos maduros y no maduros. Las predicciones

deben contarse, así como las predicciones acertadas para evaluar nuestro algoritmo.

Para evaluar nuestro algoritmo se construyen los vectores de categorías. En este caso de 0s y 1s. Ver el código 9.

Código 9. Obtención de vectores de las predicciones.

```
vectorSalidaPeach=[]
for i in range(contadorPeach):
    print("NamesPeach: " + file_namesPeach[i])
    pPeach=predict('C:/TEST/TESTDMIT/TInmat/' + file_namesPeach[i])
    print(pPeach)
    if(pPeach==0):
        #print("Entro cero")
        x=0
    if (pPeach == 1):
        #print("Entro uno")
        x = 1
    vectorSalidaPeach.append(x)
vectorSalidaPeach=[]
for jj in range(contadorPeach):
    print("NamesPeach: " + file_namesPeach[jj])
    pPeach = predict('C:/TEST/TESTDMIT/TMat/' + file_namesPeach[jj])
    print(pPeach)
    if (pPeach == 0):
        # print("Entro cero")
        y = 0
    if (pPeach == 1):
        # print("Entro uno")
        y = 1

    vectorSalidaPeach.append(y)
```

Como se puede observar se obtienen dos vectores, uno de duraznos maduros y otro de duraznos no maduros. Luego se hará la comparación.

Para obtener el desempeño de nuestro algoritmo se aplican las métricas a la *matriz de confusión*. Ver el código 10.

Código 10. Obtención de la matriz de confusión y cálculo de métricas.

```
print("####################---Matrix de Confusion --#################")
matriz_Confusion = confusion.eval(session=sess)
print(matriz_Confusion)

vp= matriz_Confusion[0,0]
print("vp," + str(vp))

fp= matriz_Confusion[0,1]
print("fp," + str(fp))

fn= matriz_Confusion[1,0]
print("fn," + str(fn))

vn= matriz_Confusion[1,1]
print("vn," + str(vn))

precision = (vp+vn)/(vp+fp+fn+vn)
```

```
print("precision," + str(precision))
sensibilidad = (vp)/(vp+fn)
print("sensibilidad," + str(sensibilidad))

especificidad = (vn)/(vn+fp)
print("especificidad," + str(especificidad))
```

Una vez obtenidas las métricas de confusión se almacenan en variables que nos permitan conocer el desempeño del algoritmo. Ver el código 11.

Código 11. Matriz de confusión y métricas obtenidas.

```
###################---Matrix de Confusion --################
[[25  7]
 [ 3 29]]
vp,25
fp,7
fn,3
vn,29
precision,0.84375
sensibilidad,0.8928571428571429
especificidad,0.8055555555555556
```

Con la matriz de confusión y las métricas de precisión, sensibilidad y especificidad podemos determinar la eficiencia de nuestro algoritmo. Esto se repitió para cada clasificación de este proyecto de investigación.

Se hizo otra, clasificación de duraznos: Maduros, inmaduros y dañados. Con 560 imágenes y 25 épocas. Después de entrenamiento se hizo la prueba con 60 imágenes.

Capítulo 4
Resultados y conclusiones

Se aplicó una prueba con 64 imágenes para cada caso. Se obtuvieron: verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos, lo que permitió obtener otros indicadores como la *Precisión*, la *Sensibilidad* y la *Especificidad*. Ver la Tabla 8.

Tabla 8: Resultados obtenidos durante las pruebas con 64 imágenes.

	Matriz de confusión			Precisión	Sensitividad	Especificidad	f-measure
Duraznos maduros e inmaduros		Positivos	Negativos	**95.31**	**93.93**	**96.77**	**95.4**
	Positivos	**31**	**1**				
	Negativos	**2**	**30**				
Duraznos maduros e inmaduros (textura)	Positivos	**25**	**7**	**84.37**	**89.28**	**80.55**	**83.3**
	Negativos	**3**	**29**				
Duraznos sanos y dañados	Positivos	**31**	**1**	**92.18**	**88.57**	**96.55**	**92.5**
	Negativos	**4**	**28**				

En los resultados, se observa una precisión del 95.31% al clasificar duraznos maduros e inmaduros es decir que solo se clasifica mal uno de 20 duraznos. Por otro lado, con las imágenes en las cuales solo se consideró la textura se obtuvo un valor más bajo del 84.37%. Quiere decir que se clasificarían mal tres de 20 duraznos. Finalmente, se obtuvo el 92.18% al clasificar los duraznos sanos y dañados, por lo que clasificaría mal al menos dos de 20 duraznos. En la clasificación de tres categorías se obtuvo la matriz de confusión de la **Tabla 9**.

Tabla 9. Matriz de confusión de la clasificación de duraznos Dañados, Inmaduros y maduros.

Duraznos	Dañados	Inmaduros	Maduros
Dañados	**19**	**0**	**1**
Inmaduros	**4**	**14**	**2**
Maduros	**3**	**0**	**17**

	Matriz de confusión			Precisión	Sensitividad	Especificidad	f-measure
Duraznos dañados e inmaduros		Positivos	Negativos	**82.6**	**100**	**22.2**	**90.5**
	Positivos	**19**	**0**				
	Negativos	**4**	**14**				
Duraznos dañados y maduros	Positivos	**19**	**0**	**86.4**	**100**	**100**	**92.7**
	Negativos	**3**	**0**				
Duraznos maduros e inmaduros	Positivos	**3**	**0**	**42.9**	**100**	**22.2**	**60**
	Negativos	**4**	**14**				

Durante el entrenamiento, en la clasificación de las tres categorías se obtuvo una precisión del 95.54%. Mientras que durante las pruebas con 60 imágenes se obtuvo la precisión de 83.33%, al dividir los verdaderos positivos de las tres categorías entre el total de unidades a clasificar. Esta clasificación resulta clave, ya que el algoritmo debe de identificar el tipo de durazno una vez obtenida la imagen.

El comportamiento de la *Pérdida* (*Lost*) y de la *precisión* (*Accuracy*) durante el entrenamiento durante 25 épocas se puede ver en la figura 3.

Con respecto a los resultados de otros trabajos similares, se obtiene la siguiente Tabla 10, donde se puede indentificar algunos trabajos similares que usan *CNN's* preentrenadas o bien *ML*, es interesante notar que las es importante la cantidad de imágenes y uso de diferentes configuraciones de lass redes neuronales y de las *CNN's* para poder determinar una metodología idónea para la clasificación.

Tabla 10: Comparación de Resultados

Autores	Técnica	Presición
(K. Kangune, 2019)	Uvas maduras y no maduras. 4000 imágenes , pre-procesamiento de imágenes de uvas, in geniería de características y clasificación con CNN	**79%**
(J. Rezgui, 2020)	AlexNet y fotos de diversas frutas	**97.11%**
(Santi Kumari Behera, 2020)	300 imagénes, con KNN ponderado, SVM G aussiano y VGG19 entrenada	Con tuvieron los 3 presicion de **100%** con los tiempos má s bajos de entrenamiento

Propuesta del trabajo	Imágenes 550 2 capas convolucionales, la primera con 32 f iltros y la segunda con 64 filtros	**95.31%** al clasificar durazno s maduros e inmaduros. Para duraznos sanos y dañados el **92.18%.** Para las 3 categorias maduro, inmaduro y dañ ado se tiene una precisión de **83.33%**

Las investigaciones usando *CNN* tienen se realizan sobre diversa áreas de aplicación, la solidez del conjunto de datos es esencial, por ello debe ser resultado de una sistemática toma de imágenes en las condiciones óptimas de las características que se desean capturar como importantes para la detección de éstas y con ello la clasificación del conjunto de datos de imágenes digitales, que en el caso de ésta investigación son fotos de duraznos: maduros, no maduros y dañados. También el modelo de la *CNN* complementa la correcta identificación de intereses particulares en las imágenes. Como el entrenamiento de una *CNN* debe servir para casos reales, entonces el conjunto de datos debe tener características de balanceo, es decir, igual número de imágenes para los diferentes grupos a utilizar. De igual manera, la cantidad de imágenes para validación debe ser suficiente para evitar el sobreajuste ó estandarización de imágenes de la *CNN*. Se obtuvo una precisión del 95.31% al clasificar duraznos maduros e inmaduros se puede valorar que cuando se trata de sólo 2 clases usando características de textura, se pueden obtener buenos resultados y al realizar la clasificación de más clases: maduros, no maduros, dañados se obtuvo una precisión de 92.8%, es decir aunque el tercer grupo tiene muy explicitas sus características físicas, dentro de las imágenes, fue un factor para que la precisión disminuyera considerablemente. Se deben realizar más investigaciones, aplicación de redes especializadas para que la clasificación de más de 2 grupos tenga mejores resultados.

Después de analizar los resultados se puede concluir que la precisión sugiere que la clasificación se puede implementar en alguna máquina que pudiese cosechar duraznos de manera automatizada. También es importante notar que las imágenes que solo muestran la textura del

durazno empeoran los resultados, por lo que es mejor usar la imagen completa del durazno. Clasificar las tres categorías mostró resultados aceptables, ya que el 83.33% implica que acierta cuatro y falla uno de cada cinco clasificaciones. Al aplicar redes neuronales convolucionales es conveniente que se realice con una gran cantidad de datos (data set de fotos), para mejorar la precisión, se deben reunir más fotografías para aumentar la confiabilidad.

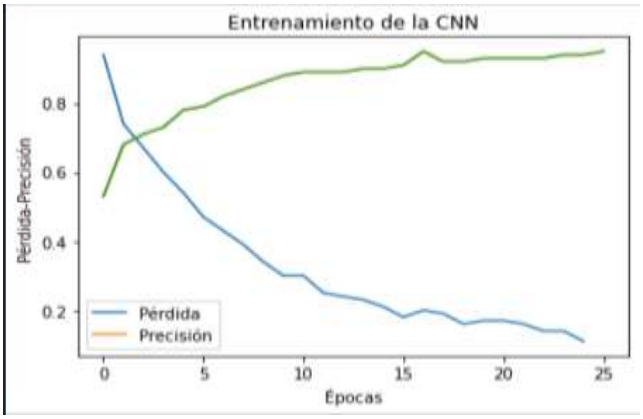

Figura 23: Comportamiento de la pérdida de la precisión

Se puede observar una pérdida con tendencia a 0.0 mientras que la precisión con tendencia al 100%.

Discusión

Las investigaciones usando *CNN* tienen se realizan sobre diversa áreas de aplicación, la solidez del conjunto de datos es esencial, por ello debe ser resultado de una sistemática toma de imágenes en las condiciones óptimas de las características que se desean capturar como importantes para la detección de éstas y con ello la clasificación del conjunto de datos de imágenes digitales, que en el caso de ésta investigación son fotos de duraznos: maduros, no maduros y dañados. También el modelo de la *CNN* complementa la correcta identificación de intereses particulares en las imágenes. Como el entrenamiento de una *CNN* debe servir para casos reales, entonces el conjunto de datos debe tener características de balanceo, es decir, igual número de imágenes para los diferentes grupos a utilizar. De igual manera, la cantidad de imágenes para validación debe ser suficiente para evitar el sobreajuste ó estandarización de imágenes de la *CNN*. Se obtuvo una precisión del 95.31%

al clasificar duraznos maduros e inmaduros se puede valorar que cuando se trata de sólo dos clases usando características de textura, se pueden obtener buenos resultados y al realizar la clasificación de más clases: maduros, no maduros, dañados se obtuvo una precisión de 92.8%, es decir aunque el tercer grupo tiene muy explicitas sus características físicas, dentro de las imágenes, fue un factor para que la precisión disminuyera considerablemente. Se deben realizar más investigaciones, aplicación de redes especializadas para que la clasificación de más de dos grupos tenga mejores resultados.

Conclusión

Después de analizar los resultados se puede concluir que la precisión sugiere que la clasificación se puede implementar en alguna máquina que pudiese cosechar duraznos de manera automatizada.

También es importante notar que las imágenes que solo muestran la textura del durazno empeoran los resultados, por lo que es mejor usar la imagen completa del durazno.

Clasificar las tres categorías mostró resultados aceptables, ya que el 83.33% implica que acierta cuatro y falla uno de cada cinco clasificaciones.

Al aplicar Redes Neuronales Convolucionales es conveniente que se realice con una gran cantidad de datos (data set de fotos), para mejorar la precisión, se deben reunir más fotografías para aumentar la confiabilidad.

Revista Iberoamericana de las Ciencias Biológicas y Agropecuarias

ISSN: 2007 - 9990

A quien corresponda:

La Revista Iberoamericana de las Ciencias Biológicas y Agropecuarias CIBA, editada por el Centro de Estudios e Investigaciones para el Desarrollo Docente CENID A.C. con registro 14658 a través del Registro Nacional de Instituciones y Empresas Científicas y Tecnológicas RENIECYT hace **CONSTAR** que **Ma. Dolores Arévalo Zenteno, José Sergio Ruíz Castilla y Joel Ayala de la Vega** son autores del artículo titulado; **"Clasificación de frutos del Durazno en maduros, no maduros y dañados hacia la cosecha automatizada"** Mismo que será publicado en nuestro volumen 9, número 18, del año 2020 semestre Enero – Junio.

Se extiende la presente a petición del interesado, para los efectos legales y formales que convengan.

ATENTAMENTE

Guadalajara, Jalisco a 01 de Diciembre del 2020

Dr. Francisco Santillán Campos
Director Editor

Clasificación de frutos del Durazno en maduros, no maduros y dañados hacia la cosecha automatizada

Classification of peach fruits in ripe, unripe and damaged towards automated harvest

Ma. Dolores Arévalo Zenteno
Universidad Autónoma del Estado de México, Centro Universitario UAEM Texcoco
mdarevaloz@uaemex.mx
https://orcid.org/0000-0002-4615-6890

José Sergio Ruíz Castilla
Universidad Autónoma del Estado de México, Centro Universitario UAEM Texcoco
jsergioruizc@gmail.com
https://orcid.org/0000-0001-7821-4912

Joel Ayala de la Vega
Universidad Autónoma del Estado de México, Centro Universitario UAEM Texcoco
Joelayala2001@yahoo.com.mx
https://orcid.org/0000-0003-3279-4143

Apéndice A

Código Elaborado

Algoritmo CNN para Clasificar duraznos maduros e inmaduros solo textura. El siguiente código usa una red convoluciona sencilla para hacer la clasifiación de duraznos, primero se preparan los datos, se define el tamaño del filto y del pooling, el rango de aprendizaje para posteriormente definir la red convolucional

```
import sys
import os

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras        import optimizers
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.python.keras.layers import Convolution2D, MaxPooling2D
from tensorflow.python.keras        import backend as K
K.clear_session()

data_entrenamiento = 'C:/DMIT/Train'
data_validacion    = 'C:/DMIT/Valid'

epocas           = 8
altura, longitud = 100, 100
batch_size       = 2
pasos            = 3000
validacion_pasos = 600

filtrosConv1   = 32
filtrosConv2   = 64
tamano_filtro1 = (3, 3)
tamano_filtro2 = (2, 2)
tamano_pool    = (2, 2)
clases         = 2
lr             = 0.0004

##Preparamos nuestras imagenes
entrenamiento_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

validacion_generador = test_datagen.flow_from_directory(
    data_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

cnn = Sequential()
```

```
cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, padding ="same",
input_shape=(altura, longitud, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation='softmax'))

cnn.compile(loss='categorical_crossentropy',
            optimizer=optimizers.Adam(lr=lr),
            metrics=['accuracy'])

cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epocas,
    validation_data=validacion_generador,
    validation_steps=validacion_pasos)

#target_dir = './modeloDMIT/'

#if not os.path.exists(target_ir):
#  os.mkdir(target_dir)
#cnn.save        ('C:/MODELS/modeloDMIT/modeloDMIT.h5')
#cnn.save_weights('C:/MODELS/modeloDMIT/pesosDMIT.h5')

target_dir = './modeloDMIT/'

if not os.path.exists(target_dir):
  os.mkdir(target_dir)

cnn.save        ('C://MODELS/modeloDMIT/modeloDMIT.h5')
cnn.save_weights('C:/MODELS/modeloDMIT/pesosDMIT.h5')
print ("Los resultados han sido guardados...")
```

Salida del entrenamiento

```
Found 300 images belonging to 2 classes.
Found 60 images belonging to 2 classes.
Epoch 1/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.5203 - acc:
0.7618Epoch 1/8
3000/3000 [==============================] - 498s 166ms/step - loss: 0.5203 -
acc: 0.7618 - val_loss: 0.2391 - val_acc: 0.88332:15 - loss: - ETA: 2:13 - loss:
0.2 - ETA: 2:12 - loss: 0.2 - ETA: 2:12 - loss: - ETA
Epoch 2/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.3552 - acc:
0.8396Epoch 1/8
3000/3000 [==============================] - 499s 166ms/step - loss: 0.3552 -
acc: 0.8397 - val_loss: 0.2760 - val_acc: 0.9000
Epoch 3/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.2877 - acc:
0.8796Epoch 1/8
3000/3000 [==============================] - 503s 168ms/step - loss: 0.2877 -
acc: 0.8797 - val_loss: 0.4408 - val_acc: 0.8500
Epoch 4/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.2398 - acc:
0.9018Epoch 1/8
3000/3000 [==============================] - 502s 167ms/step - loss: 0.2397 -
acc: 0.9018 - val_loss: 0.4007 - val_acc: 0.8500
Epoch 5/8
2999/3000 [============================>.] - ETA: 0s - loss: 0.2047 - acc:
```

```
0.9183Epoch 1/8
3000/3000 [==============================] - 504s 168ms/step - loss: 0.2046 -
acc: 0.9183 - val_loss: 0.6002 - val_acc: 0.8000
Epoch 6/8
2999/3000 [==============================>.] - ETA: 0s - loss: 0.1707 - acc:
0.9325Epoch 1/8
3000/3000 [==============================] - 504s 168ms/step - loss: 0.1707 -
acc: 0.9325 - val_loss: 1.2071 - val_acc: 0.7833
Epoch 7/8
2999/3000 [==============================>.] - ETA: 0s - loss: 0.1615 - acc:
0.9318Epoch 1/8
3000/3000 [==============================] - 504s 168ms/step - loss: 0.1614 -
acc: 0.9318 - val_loss: 0.7623 - val_acc: 0.7833
Epoch 8/8
2999/3000 [==============================>.] - ETA: 0s - loss: 0.1307 - acc:
0.9493Epoch 1/8
3000/3000 [==============================] - 509s 170ms/step - loss: 0.1306 -
acc: 0.9493 - val_loss: 0.8037 - val_acc: 0.8667
Los resultados han sido guardados...
```

Prueba de clasificación, ya que se hizo el entrenamiento, se procede a las pruebas con los datos correspondientes

```
import sys
import os
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from os import walk, path
import numpy as np
from tensorflow.python.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.models              import load_model
import tensorflow as tf
from tensorflow.python import Session

longitud, altura = 100, 100

modelo =       'C:/MODELS/modeloDMIT/modeloDMIT.h5'
pesos_modelo = 'C:/MODELS/modeloDMIT/pesosDMIT.h5'

cnn = load_model(modelo)
cnn.load_weights(pesos_modelo)

def predict(file):
  x = load_img(file, target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict(x)
  result = array[0]
  answer = np.argmax(result)
  if answer == 0:
    print("pred: Peach")
  elif answer == 1:
    print("pred: Peach")
  return answer
```

#----------------------imagenes para la pruena---------------------------------

```
#D:/k-fold/sinpiel/001-t/b001t/
contadorPeach=0;
for dir_pathPeach, dir_namesPeach, file_namesPeach in
walk('C:/TEST/TESTDMIT/TInmat/'):
    for fnPeach in file_namesPeach:
        if fnPeach[-3:] == 'JPG':
            contadorPeach= contadorPeach + 1

print("ContadorPeach: " + str(contadorPeach))
print(file_namesPeach)
```

```
print(file_namesPeach[0])
x=0

contadorPeach=0
for dir_pathPeach, dir_namesPeach, file_namesPeach in
walk('C:/TEST/TESTDMIT/TMat/'):
    for fnPeach in file_namesPeach:
        if fnPeach[-3:] == 'JPG':
            contadorPeach= contadorPeach + 1

print("ContadorPeach: " + str(contadorPeach))
print(file_namesPeach)
print(file_namesPeach[0])
y=0

#--------------------------------------------------------------------------

vectorSalidaPeach=[]
for i in range(contadorPeach):
    print("NamesPeach: " + file_namesPeach[i])
    pPeach=predict('C:/TEST/TESTDMIT/TInmat/' + file_namesPeach[i])
    print(pPeach)
    if(pPeach==0):
        #print("Entro cero")
        x=0
    if (pPeach == 1):
        #print("Entro uno")
        x = 1
    vectorSalidaPeach.append(x)
vectorSalidaPeach=[]
for jj in range(contadorPeach):
    print("NamesPeach: " + file_namesPeach[jj])
    pPeach = predict('C:/TEST/TESTDMIT/TMat/' + file_namesPeach[jj])
    print(pPeach)
    if (pPeach == 0):
        # print("Entro cero")
        y = 0
    if (pPeach == 1):
        # print("Entro uno")
        y = 1
    vectorSalidaPeach.append(y)

print("peach-------------------------------------------------------------------
------")
print(vectorSalidaPeach)

print("Peach-------------------------------------------------------------------
------")
print(vectorSalidaPeach)

unidos=np.append(vectorSalidaPeach,vectorSalidaPeach)
print("Unidos------------------------------------------------------------------
----------")
print(unidos)

print("Valores reales ----------------------------------------------------------
---------")

vZeros = np.zeros((32,),dtype=int)
print(vZeros)

vUnos=np.ones((32,),dtype=int)
print(vUnos)

unidos2 = np.append(vZeros,vUnos)
print(unidos2)
```

```
#sess = tf.Session()
sess= tf.compat.v1.Session()
print(sess)
#confusion = tf.confusion_matrix(labels=[1,2,4],
predictions=[2,2,4],num_classes=3)
confusion = tf.math.confusion_matrix(labels=unidos2, predictions=unidos)
#Estaba sin math

#print(type(confusion))
#print(confusion.eval(session=sess))
print("####################---Matrix de Confusion --#################")
matriz_Confusion = confusion.eval(session=sess)
print(matriz_Confusion)

vp= matriz_Confusion[0,0]
print("vp," + str(vp))

fp= matriz_Confusion[0,1]
print("fp," + str(fp))

fn= matriz_Confusion[1,0]
print("fn," + str(fn))

vn= matriz_Confusion[1,1]
print("vn," + str(vn))

precision = (vp+vn)/(vp+fp+fn+vn)
print("precision," + str(precision))
sensibilidad = (vp)/(vp+fn)
print("sensibilidad," + str(sensibilidad))

especificidad = (vn)/(vn+fp)
print("especificidad," + str(especificidad))
<tensorflow.python.client.session.Session object at 0x0000020329592080>
####################---Matrix de Confusion --#################
[[25  7]
 [ 3 29]]
vp,25
fp,7
fn,3
vn,29
precision,0.84375
sensibilidad,0.8928571428571429
especificidad,0.8055555555555556
```

CNN para clasificar duraznos maduros inmaduros y dañados

```
import sys
import os
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras        import optimizers
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.python.keras.layers import Convolution2D, MaxPooling2D
from tensorflow.python.keras        import backend as K

K.clear_session()

data_entrenamiento = 'C:/FRUTO/DMID/Train'
data_validacion    = 'C:/FRUTO/DMID/Valid'


epocas           = 25
altura, longitud = 100, 100
batch_size       = 2
pasos           = 1200
validacion_pasos = 400
filtrosConv1   = 32
filtrosConv2   = 64
tamano_filtro1 = (3, 3)
tamano_filtro2 = (2, 2)
tamano_pool    = (2, 2)
clases         = 3
lr             = 0.0004

##Preparamos nuestras imagenes
entrenamiento_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

validacion_generador = test_datagen.flow_from_directory(
    data_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, padding ="same",
input_shape=(altura, longitud, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))
```

```
cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation='softmax'))

cnn.compile(loss='categorical_crossentropy',
            optimizer=optimizers.Adam(lr=lr),
            metrics=['accuracy'])

cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epocas,
    validation_data=validacion_generador,
    validation_steps=validacion_pasos)

target_dir = 'C:/FRUTO/MODELS/modeloDMID/'

if not os.path.exists(target_dir):
  os.mkdir(target_dir)

cnn.save        ('C:/FRUTO/MODELS/modeloDMID/modeloDMID.h5')
cnn.save_weights('C:/FRUTO/MODELS/modeloDMID/pesosDMID.h5')
print ("Los resultados han sido guardados...")

Found 448 images belonging to 3 classes.
Found 112 images belonging to 3 classes.


Epoch 1/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.9427 - acc: 0.5325
ETA: 11 - ETA: 0s - loss: 0.9420 - acc: 0.53Epoch 1/25
1200/1200 [==============================] - 251s 209ms/step - loss: 0.9426 -
acc: 0.5325 - val_loss: 1.0912 - val_acc: 0.5537
Epoch 2/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.7486 - acc:
0.6872Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.7488 -
acc: 0.6871 - val_loss: 0.9891 - val_acc: 0.5288
Epoch 3/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.6779 - acc:
0.7185- ETA: 0s - loss: 0.6776 - acc: 0.718Epoch 1/25
1200/1200 [==============================] - 207s 172ms/step - loss: 0.6773 -
acc: 0.7188 - val_loss: 1.7046 - val_acc: 0.4563
Epoch 4/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.6024 - acc:
0.7381Epoch 1/25
1200/1200 [==============================] - 207s 172ms/step - loss: 0.6025 -
acc: 0.7379 - val_loss: 1.6926 - val_acc: 0.5238
Epoch 5/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.5475 - acc: 0.7684
- ETA: 4s - loss: 0.5464 - ETA: 3s - los - ETA: 0s - loss: 0.5474 - acc:
0.7686Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.5473 -
acc: 0.7688 - val_loss: 1.5217 - val_acc: 0.5575
Epoch 6/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.4760 - acc:
0.7998- ETA: 0s - loss: 0.4772 - acc: Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.4764 -
```

```
acc: 0.7992 - val_loss: 1.3329 - val_acc: 0.5788
Epoch 7/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.4395 - acc:
0.8249Epoch 1/25
1200/1200 [==============================] - 205s 171ms/step - loss: 0.4392 -
acc: 0.8250 - val_loss: 2.1684 - val_acc: 0.5250
Epoch 8/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.3937 - acc:
0.8407Epoch 1/25
1200/1200 [==============================] - 207s 172ms/step - loss: 0.3940 -
acc: 0.8404 - val_loss: 1.8900 - val_acc: 0.6263
Epoch 9/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.3438 - acc:
0.8666Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.3441 -
acc: 0.8662 - val_loss: 2.7441 - val_acc: 0.5013
Epoch 10/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.3012 - acc:
0.8837Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.3011 -
acc: 0.8838 - val_loss: 2.4416 - val_acc: 0.5100
Epoch 11/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.3054 - acc:
0.8924- ETA: 0s - loss: 0.3054 - acc: 0.89Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.3052 -
acc: 0.8925 - val_loss: 1.9017 - val_acc: 0.4663
Epoch 12/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.2602 - acc:
0.8962Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.2599 -
acc: 0.8963 - val_loss: 2.2566 - val_acc: 0.5475
Epoch 13/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.2424 - acc:
0.9078- ETA: 1s - loss: 0.2444 -Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.2422 -
acc: 0.9079 - val_loss: 3.2490 - val_acc: 0.5475
Epoch 14/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.2325 - acc:
0.9049Epoch 1/25
1200/1200 [==============================] - 205s 171ms/step - loss: 0.2329 -
acc: 0.9046 - val_loss: 2.1462 - val_acc: 0.4837
Epoch 15/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.2165 - acc:
0.9166- ETA: 2s - loss:Epoch 1/25
1200/1200 [==============================] - 206s 171ms/step - loss: 0.2163 -
acc: 0.9167 - val_loss: 3.3639 - val_acc: 0.4888
Epoch 16/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1886 - acc:
0.9254- ETA: 6 - ETA: 3s - losEpoch 1/25
1200/1200 [==============================] - 205s 171ms/step - loss: 0.1884 -
acc: 0.9254 - val_loss: 3.1595 - val_acc: 0.4638
Epoch 17/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.2071 - acc:
0.9204Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.2070 -
acc: 0.9204 - val_loss: 3.1858 - val_acc: 0.4462
Epoch 18/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1919 - acc: 0.9254
- ETA: 6s - loss: 0.1891 - accEpoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.1917 -
acc: 0.9254 - val_loss: 3.1436 - val_acc: 0.4900
Epoch 19/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1621 - acc:
0.9345Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.1620 -
acc: 0.9346 - val_loss: 3.2658 - val_acc: 0.5288
```

```
Epoch 20/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1709 - acc:
0.9387- ETEpoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.1708 -
acc: 0.9388 - val_loss: 2.7040 - val_acc: 0.4450
Epoch 21/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1707 - acc:
0.9354- ETA: 4s - loss: 0.1716 - acc: 0 - ETA: 4s - loss: 0.1712 - acc: 0.936 -
ETA: 4s - loss: 0.1710 - acc: 0.9 - ETA: 3s Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.1706 -
acc: 0.9354 - val_loss: 3.4150 - val_acc: 0.5263
Epoch 22/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1691 - acc:
0.9362- ETA: 0s - loss: 0.1684 - acc: 0.Epoch 1/25
1200/1200 [==============================] - 205s 171ms/step - loss: 0.1692 -
acc: 0.9362 - val_loss: 3.3085 - val_acc: 0.4462
Epoch 23/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1441 - acc:
0.9450Epoch 1/25
1200/1200 [==============================] - 206s 172ms/step - loss: 0.1440 -
acc: 0.9450 - val_loss: 3.8585 - val_acc: 0.4600
Epoch 24/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1499 - acc:
0.9458Epoch 1/25
1200/1200 [==============================] - 205s 171ms/step - loss: 0.1498 -
acc: 0.9458 - val_loss: 3.0273 - val_acc: 0.5437
Epoch 25/25
1199/1200 [============================>.] - ETA: 0s - loss: 0.1199 - acc:
0.9554Epoch 1/25
1200/1200 [==============================] - 206s 171ms/step - loss: 0.1198 -
acc: 0.9554 - val_loss: 3.3208 - val_acc: 0.5100
Los resultados han sido guardados...
```

Pruebas con un predictor

```
import numpy as np
from tensorflow.python.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.models              import load_model

longitud, altura = 100, 100
modelo =       'C:/FRUTO/MODELS/modeloDMID/modeloDMID.h5'
pesos_modelo = 'C:/FRUTO/MODELS/modeloDMID/pesosDMID.h5'
cnn = load_model(modelo)
cnn.load_weights(pesos_modelo)

def predict(file):
  x = load_img(file, target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict(x)
  result = array[0]
  answer = np.argmax(result)
  print (answer)
  if answer == 0:
    print("pred: Damage")
  elif answer == 1:
    print("pred: Inmature")
  elif answer == 2:
    print ("pred: Mature")

  return answer

print("Damage-------------------------------------------------------------")

predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (6).JPG')
```

```
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (58).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (85).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (90).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (96).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (116).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (136).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (159).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (227).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (262).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (372).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (412).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros (415).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros1 (47).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros1 (48).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros1 (58).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros1 (69).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros1 (96).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros1 (136).JPG')
predict('C:/FRUTO/TEST/TestDMID/TDam/maduros1 (244).JPG')



print("Inmature-----------------------------------------------------------")

predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (22).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (23).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (29).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (34).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (39).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (49).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (55).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (74).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (87).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (99).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (112).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (117).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (125).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (132).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (140).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (149).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (151).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (154).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (158).JPG')
predict('C:/FRUTO/TEST/TestDMID/TInmat/verdes (175).JPG')



print("Mature-------------------------------------------------------------")

predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (2).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (4).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (5).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (6).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (11).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (12).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (20).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (21).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (23).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (24).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (27).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (29).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (30).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (31).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (36).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (43).JPG')
```

```
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (48).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (49).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (51).JPG')
predict('C:/FRUTO/TEST/TestDMID/TMat/maduros (54).JPG')
```

Salida, aquí se tiene los resultados obtenidos de acuerdo a las imagénes

```
Imágenes------------------------------------------------------------
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
0
pred: Damage
2
pred: Mature
0
pred: Damage
0
pred: Damage
Inmature------------------------------------------------------------
1
pred: Inmature
1
pred: Inmature
0
pred: Damage
0
pred: Damage
1
pred: Inmature
1
pred: Inmature
1
pred: Inmature
1
pred: Inmature
1
```

```
pred: Inmature
0
pred: Damage
1
pred: Inmature
2
pred: Mature
1
pred: Inmature
1
pred: Inmature
1
pred: Inmature
0
pred: Damage
1
pred: Inmature
1
pred: Inmature
1
pred: Inmature
2
pred: Mature
Mature------------------------------------------------------------
0
pred: Damage
0
pred: Damage
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
2
pred: Mature
0
pred: Damage
2
pred: Mature
2
pred: Mature
```

Bibliografía

Alipasandi, A., Ghaffari, H., & Zohrabi, A. S. (2013). Classification of three Varieties of Peach Fruit Using Artificial Neural. *International Journal of Agronomy and Plant Production. 4*, págs. 2179-2186. VictorQuest. Recuperado el 05 de 2019, de http:// www.ijappjournal.com

Amlekar, M., Manza, R. R., Yannawar, P., & Gaikwad, A. T. (2014). Leaf Features Based Plant Classification Using Artificial Neural Network. *IBMRD's Journal of Management and Research,, 3*(1), 224 - 232.

Anaconda. (2019). Obtenido de https://anaconda.org/

Baíza Avelar, V. H. (octubre de 2004). *Guía Técnica del Cultivo del Melocotón.* El Salvador: Ministerio de Agricultura y Ganadería. Recuperado el marzo de 2019, de https://hortintl.cals.ncsu.edu/es/articles/guia-t-cnica-del-cultivo-del-melocot-n

Bertrand, S., Ben Ameur, R., Cerutti, G., Coquin, D., Valet, L., & Tougne, L. (Junio de 2018). Bark and Leaf Fusion Systems to Improve Automatic Tree Species Recognition. (Elsevier, Ed.) *Ecological Informatics, 46*, 57-73. Recuperado el 2019, de https://doi.org/10.1016/j.ecoinf.2018.05.007.

Beysolow, T. (2017). *Introduction to Deep Learning Using R*. San Francisco California: Apress. doi:10.1007/978-1-4842-2734-3_1

Borrmann, Y. F., Ferraciolli, M., Folego, G., & A. Rodrigues, L. H. (2020). Identifying Mature andd Inmature tomatoes in a Greenhouse with Deep Learning. *XXVIII Congreso {virtual} de Iniciacao Cientifica da Unicamp*. Brasil: Pro Reitoria de Pesquisa Unicamp. Obtenido de https://www.prp.unicamp.br/inscricao-congresso/resumos/2020P17585A33995O179.pdf

Cervantes, J., García, F., Taltempa, J., Ruiz Castilla, J. S., Rendon, A. Y., & Jalili, L. D. (2017). Análisis Comparativo de las técnicas utilizadas en un Sistema de Hojas de Planta. *Revista Iberoamericana de Automática e Informática industrial, 14*(1), 104-114. Obtenido de http://dx.doi.org/10.1016/j.riai.2016.09.005

Chollet, F. (2018). *Deep Learning with Python*. Shelter Island, NY: Manning. doi:9781617294433

cs231n. (septiembre de 2019). *CS231n Convolutional Neural Networks for Visual Recognition*. Obtenido de https://cs231n.github.io/classification/

D. Jalili, L., Morales, A., Cervantes, J., & Ruíz-Castilla, J. (2016). Improving the Performance of Leaves Identification by Features Selection with Genetic Algorithms. En J. Figueroa-García, E. López-Santana, & Ferro-Escoba (Ed.), *Applied Computer Sciences in Engineering WEA. 657*, págs. 103-114. Springer, Cham. doi:10.1007/978-3-319-508801-10

Dean, J., Corrado, G. S., Monga, R., & Chen, K. (2012). Large Scale Distributed Deep Networks. En F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Ed.), *NIPS Neural Information Processing Systems (NIPS). 1*, págs. 1223-1231. Curran Associates Inc. Recuperado el 2019

Dean, J., Corrado, G. S., Monga, R., & Chen, K. (2012). Large Scale Distributed Deep Networks. En F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Ed.), *NIPS Neural Information Processing Systems (NIPS). 1*, págs. 1223-1231. Curran Associates Inc.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. *Dept. of Computer Science, Princeton University, USA*.

Duarte, V. M., & Chang, F. L. (2015). *Clasificación de objetos en imégenes usando SIFT*. InstitutoNacional de Astrofisica öptica y Electrónica. Recuperado el 10 de Mayo de 2019

Emmert-Streib, F., Yang, Z., Feng, H., Tripathi, S., & Dehmer, M. (2020). An Introductory Review of Deep Learning for Prediction Models With. (F. Riguzzi, Ed.) *Frontiers in Artificial Intelligence, 3*(4). doi:doi: 10.3389/frai.2020.00004

Fergus, R., & Zeiler, Matthew D. (2014). Visualizing and Understanding Convolutional Networks. (Springer, Ed.) *European conference on computer vision*, 818-833.

Flores, M. J., & D. J. Robayo, y. D. (2015). Histograma del gradiente con múltiples orientaciones orientaciones (hog-mo) detección de personas. *Vínculos, 12*(2), 138-147. Recuperado el 10 de Mayo de 2019, de https://revistas.udistrital.edu.co/ojs/index.php/vinculos/article/view/10991/11830

Flores, M. J., & D. J. Robayo, y. D. (2015). Histograma del gradiente con múltiples orientaciones orientaciones (hog-mo) detección de personas. *Vínculos, 12*(2), 138 - 147. Obtenido de https://revistas.udistrital.edu.co/ojs/index.php/vinculos/article/view/10991/11830

García Lamont, F., López, A., Cervantes, J., & Rodríguez, L. (7 de Marzo de 2018). *Segmentation of images by color features: A survey*. Recuperado el Noviembre de 2020, de Elsevier: www.elsevier.com/locate/neucom

García, F., Cervantes, J., López, A., & Alvarado, M. (Julio de 2016). Fruit Classification by Extracting Color Chromaticity, Shape and Texture Features: Towards an Application for Supermarkets. *IEEE Latin America Transactions, 14*(7), 3434 - 3443. doi:10.1109/TLA.2016.7587652

Georgiou, T., Liu, Y., Chen, W., & Lew, M. (22 de Nov de 2019). *A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision*. Obtenido de Springer Link: https://link.springer.com/article/10.1007/s13735-019-00183-w

GoodFellow, I., bengio, Y., & Courvile, A. (2016). *Deep Learning*. (MIT, Ed.) Massachuset, Massachusses: Massachuset Institute of Technology. Recuperado el Mayo de 2019

Grinblat, G. L., Uzal, C., Larese, M. G., & M., G. P. (2016). Deep Learning, Machine Vision,. *COMPUTERS AND ELETRONICS IN AGRICULTURE., 127*, 418-424. Obtenido de http://dx.doi.org/10.1016/j.compag.2016.07.003

Hancock, J. F., Scorza, R., & Lobos, G. A. (2008). Peachs. *Kluwer Academic Publishers*, 265-298. doi:10.1007/978-1-4020-6907-9-9

Hati, S., & G, S. (2013). Plant Recognition from Leaf Image through Artificial Neural Network. *International Journal of Computer Applications, 62*(17), 15-18. Recuperado el 2019

Hati, S., & G, S. (2013). Plant Recognition from Leaf Image through Artificial Neural Network. *International Journal of Computer Applications, 62*(17), 15 - 18.

Heras, D. (2017). Clasificador de imágenes de frutas basado en inteligencia artificial. *Revista Killkana Técnica, 1*(2), 21-30. Recuperado el 2019

Heyan, Z., Qinglin, L., & Yuankai, Q. (2018). Plant identification based on very deep convolutional. *Springer Natura*, 29780-29793. Recuperado el 16 de Mayo de 2019, de https://doi.org/10.1007/s11042-017-5578-9

J. Rezgui, T. T. (2020). Intelligent Fruit Maturity Assessment Platform Using Convolutional Neural Network: IFMAP. *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, (págs. 1-6). Montreal, QC,. doi:10.1109/ISNCC49221.2020.9297356

K. Kangune, V. K. (2019). Grapes Ripeness Estimation using Convolutional Neural network and Support Vector Machine. *Global Conference for Advancement in Technology (GCAT)*, (págs. 1-5). BANGALURU, India. doi:10.1109/GCAT47503.2019.8978341

Kadir, A., Nugroho, L., Susanto, A., & Insap Santosa, P. (2011). Leaf Classification Using Shape, Color, and Texture Features. En S. S. Group (Ed.), *International Journal of Computer Trends and Technology*, (págs. 225 - 230).

Kadir, A., Nugroho, L., Susanto, A., & Insap Santosa, P. (2011). Leaf Classification Using Shape, Color, and Texture Features. (S. S. Group, Ed.) *International Journal of Computer Trends and Technology*, 225 - 230. Recuperado el 2019

Ketkar, N. (2017). *Deep Learning with Python*. Bangalore, Karnataka, India: Apress. doi:10.1007/978-1-4842-2766-4

Khan, S., Rahmani, H., Ali Sha, S., & Bennamoun, M. (2018). *A Guide to Convolutional Neural Networks for Computer Vision*. Teh University of Western Australia. Crawley, Western Australia: Morgan & Claypool Publishers. doi:(https://doi.org/10.2200/S00822ED1V01Y201712COV015)

Khan, S., Rahmani, H., Ali Sha, S., & Bennamound, M. (2018). *A guide to convolutional neural networks for computer vision*. Morgan & Claypool. doi:10.2200/S00822ED1V0Y2017COV015

Kim, P. (2017). *MATLAB Deep Learning, With Machine Learning, Neural*. Seúl Corea: Apress.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet Clasification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*, 1097-1105.

LeCun, Y. B. (1998). Gradient-based learning applied to document. *Proc. of the IEEE*, 2278-2324.

LeCun, Y., Bengio, Y., & Hinton, G. (27 de Mayo de 2015). Deep Learning. *Nature, International Journal of Science*, 436-444.

Lim, K.-L., & Galoogahi, H. K. (2010). IEEE Computer Society. *2010 Fourth Pacific-Rim Symposium on Image and Video Technology*, 115-120. doi:DOI 10.1109/PSIVT.2010.26

Liu, Z., Zhu, L., Zhang,, X.-P., Zhou, X., Shang, L., Huang, Z.-K., & Gan, Y. (2015). Hybrid Deep Learning for Plant Leaves. En H. De-Shuang, V. Bevilacqua, & V. Prashan (Ed.), *Intelligent Computing Theories and Methodologies: 11th International Conference, ICIC 2015* (págs. 115-123). Switzerland: Springer International Publishing. doi:10.1007/978-3-319-22186-1_11

López, A., Rojas, R., Trujillo, V., Cervantes, J., Rodríguez, L., & García, F. (Noviembre de 2017). Detection of Compound Leaves for Plant Identification. *IEEE Latin America Transactions*. doi:10.1109/TLA.2017.8070425

Ludewig, D., & Sobottka, a. G. (2013). A Gabor Filter-Based Approach to Leaf Vein. *Springer-Verlag Berlin Heidelberg*, 150–159.

Ma, J., Keming, D., Zhenga, F., Zhang, L., Gong, Z., & Sun, Z. (2018). A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network. *Computers and Electronics in Agriculture*, 18-24.

Ma, J., Keming, D., Zhenga, F., Zhang, L., Gong, Z., & Sun, Z. (2018). A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network. *Computers and Electronics in Agriculture*, 18 - 24.

McCarthy, J. (2007). *What Is Artificial Intelligence? Technical report, Stanford University, Available online at: (Accessed June 2, 2018).* Técnico, Stanford University. Recuperado el 1 de Mayo de 2019, de http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html

Morris, D. D. (05 de 04 de 2018). A Pyramid CNN for Dense-Leaves Segmentation. *arXiv*. Recuperado el 30 de 04 de 2019, de https://arxiv.org/pdf/1804.01646.pdf

Muthukannan, k., Latha, P., Pon Selvi, R., & Nisha, P. (2015). Classification of Diseased Plant leaves using Neural Network Algorithms. *ARPN Journal of Engineering and Applied Sciences*, 1913 - 1919.

Naranjo Torres, J., Mora, M., Hernández García, R., J. Barrientos, R., Fredes, C., & Valenzuela, A. (2020). Una revisión de la red neuronal convolucional aplicada al procesamiento de imágenes de frutas. *Ciencias Aplicadas, 10*(10). doi:doi:10.3390/app10103443

Nava Vega, A. (2005). *Cultivo y manejo del durazno Pronus pérsica L.* Buenavista Saltillo, Coahuila, México: Universidad Autónoma Agraria.

Nikhil, B. (2017). *Fundamentals of Deep Learning. Designing Next-Generation Machine Intelligence Algorithms* (1ra ed.). Estados Unidos de América: O'Really. Recuperado el 2019

Oyallon, E., & Rabin, J. (30 de Mayo de 2015). An Analysis of the SURF Method. *Image Processing On Line*, 176–218. doi:https://doi.org/10.5201/ipol.2015.69

Rashad, M. Z.-D., & Khawasik, M. (2011). Plants Images Classification Based on Textural Features using Combined. *International Journal of Computer Science & Information Technology (IJCSIT)*, 93-98. doi:10.5121/ijcsit.2011.3407

SaiRam, K., Mukherjee, J., Patra, A., & Pratim Das, P. (18 de December de 2019). HSD-CNN: Hierarchically self decomposing CNN architecture. *ICVGIP*. Recuperado el 09 de 05 de 2019

Sandro, S. (2018). *Introduction to Deep Learning. From Logical Calculus to Artifical Intelligence.* (U. T. Computer, Ed.) Cham, Switzerland: Springer. Recuperado el 2019, de https://doi.org/10.1007/978-3-319-73004-2

Santi Kumari Behera, A. K. (2020). *Maturity status classification of papaya fruits based on machine learning and transfer learning approach*. (I. P. Agriculture, Editor) doi:10.1016/j.inpa.2020.05.003

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2014). OverFeat:Integrated Recognition, Localization and Detection using Convolutional Networks. *ICLR*.

Sharpe SM, S. A. (2019). Detection of Carolina Geranium (Geranium carolinianum) Growing in Competition with Strawberry Using Convolutional Neural Networks. *Weed Sci*, 239-245. doi:10.1017/wsc.2018.66

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large Scale Image Recognition. *ICLR*.

Sladojevic, S., Arsenovic, M., & Anderla, A. (2016). Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification. *Computational Intelligence and Neuroscience, 2016*(Article ID 3289801), 11 Páginas. doi:http://dx.doi.org/10.1155/2016/3289801

Szegedy, C., Wei, L., Yangqing, J., Semarnet, P., Reed , S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. Obtenido de https://arxiv.org/abs/1409.4842

Thyagharajan, .. K., & Raji, I. K. (2018). A Review of Visual Descriptors and Classification Techniques Used in Leaf Species Identification. En M. Kleiber, & E. Oñate (Ed.), *Archives of Computational Methods in Engineering*. *26*, págs. 933–960. Barcelona: Springer. doi:https://doi.org/10.1007/s11831-018-9266-3(0123456789().,-volV)(0123456789().,-volV)

Torres, J. (2018). *Deep Learning, Introducción Práctica con Keras*. Barcelona, España: Watch This Space. Recuperado el Mayo de 2019

Wäldchen, J., & Mäder, P. (2018). Plant Species Identifcation Using Computer Vision Techniques:. En S. Netherlands (Ed.), *Archives of Computational Methods in Engineering*, *25*, págs. 507-543. doi:https://doi.org/10.1007/s11831-016-9206-z

Yanikoglu, B., Aptoula, E., & Tirkaz, C. (2014). Automatic plant identification from photographs. *Machine Vision & Applications manuscript*, 1-15. doi:10.1007/s00138-014-0612-7

Zaccone, G., & Karim, M. R. (2018). *Deep Learning with TensorFlow*.

Zeile, M., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. (Springer, Ed.) *European conference on computer vision*, 818-833.

Zhang, Q., Zhang, M., Chen, T., Sun, Z., Ma, Y., & Yu, B. (2018). Recent advances in convolutional neural network acceleration. *Neurocomputing*, 37-51. doi:https://doi.org/10.1016/j.neucom.2018.09.038

Zhu, H., Liu, Q., Qi, Y., Huang, X., Jiang, F., & Zhang, S. (2018). Plant identification based on very deep convolutional. *Springer Science Business Media*, 1-19. doi:https://doi.org/10.1007/s11042-017-5578-9